Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Implementación de Controladores Predictivos Utilizando ADMM y Estructuras en Semi-Banda

Autor: Lorenzo Vázquez Camacho

Tutor: Teodoro Álamo Cantarero

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Implementación de Controladores Predictivos Utilizando ADMM y Estructuras en Semi-Banda

Autor:
Lorenzo Vázquez Camacho

Tutor:
Teodoro Álamo Cantarero
Catedrático

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado: Implementación de Controladores Predictivos Utilizando ADMM y Estructuras en Semi-Banda

Autor: Lorenzo Vázquez Camacho
Tutor: Teodoro Álamo Cantarero

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar quiero agradecer toda mi familia por el apoyo incondicional que me han dado a lo largo de estos 4 años.

A mi padre que me ha acompañado y animado estos cuatros años.

A mi madre por estar siempre detrás de mi obligándome a seguir adelante.

Y a mi hermana por convencerme de que podría terminar la carrera.

En segundo lugar a todos mis compañeros y amigos que a lo largos de estos 4 años han sido un pilar más en mi vida y han supuesto una ayuda de valor incalculable para mi.

En último lugar a mi tutor, Teodoro. Su amabilidad y guía han sido piezas claves para el desarrollo del proyecto. También a Víctor por ayudarme a resolver dudas y a evitar que me atascara.

Lorenzo Vázquez Camacho
Huelva, 2023

Resumen

En este proyecto se ha querido llegar una implementación del método *ADMM* más óptima para la formulación para tracking del MPC. Para ello ha sido necesario estudiar desde los conceptos más básicos hasta los más avanzados.

Se ha introducido al problema de optimización convexa para poder desarrollar los algoritmos que permitan resolver las formulaciones del MPC. Se ha profundizado en distintas formulaciones del MPC y en cómo resolverlas con los métodos desarrollados en el proyecto.

El objetivo principal del proyecto radicaba en conseguir mediante la descomposición de una matriz en semibanda conseguir unos tiempos de ejecución más óptimos para el método *ADMM* que los del método clásico usado para el MPCT, *EADMM*.

Los resultados obtenidos en el desarrollo del proyecto han sido que no solo el método *ADMM* tiene más rango de aplicación y ofrece más estabilidad, sino que también es más rápido.

Abstract

In this project, the aim was to achieve an optimal implementation of the *ADMM* method for the MPC for tracking formulation. To do this, it was necessary to study from the most basic concepts to the most advanced ones.

Convex optimization has been introduced to tackle the problem, allowing the development of algorithms to solve MPC formulations. Various MPC formulations and their resolution using the methods developed in the project have been explored.

The main goal of the project was to achieve more efficient execution times for the ADMM method compared to the classical method used for MPC tracking, *EADMM*, by decomposing a matrix into a semi-band structure.

The results obtained in the project's development show that not only does the *ADMM* method have a wider range of applications and offer more stability, but it is also faster.

Índice Abreviado

Índice

Notación

$\mathbb{R}$	Números reales
$\mathbb{Z}$	Números enteros
MPC	Model Predictive Control (Control basado en Modelo Predictivo)
$MPCR$	Model Predictive Control Regulación
$MPCT$	Model Predictive Control for Tracking (Control basado en Modelo Predictivo para Seguimiento)
$\parallel z \parallel_M^2$	$z^T Mz$
$j \in \mathbb{Z}_K^N$	j es un número entero que va desde K hasta N
$H \succ 0$	Los autovalores de la matriz H son estrictamente positivos
$H \succeq 0$	Los autovalores de la matriz H son iguales o mayores que 0
$O()$	Cota superior asintótica
$x^* = arg\ \min_x F(x)$	vector x que haga mínima a la función $F(x)$
$\mathbf{x}$	**x** es un vector cuyas componentes son otros vectores
$dom(h)$	dominio de la función h
$I_{n\times n}$	Matriz identidad de dimensiones $n \times n$
$0_{n\times n}$	Matriz rellena de 0 de dimensiones $n \times n$
0_n	Vector de n rellenas de 0

1 Introducción

Vida antes que muerte, fuerza antes que debilidad, viaje antes que destino

Brandon Sanderson

En este trabajo se ha profundizado en el MPC lineal, en concreto en dos formulaciones que salen de esta estrategia de control. Lo que se ha pretendido con el desarrollo de este documento ha sido mostrar una forma de implementar una de las formulaciones del MPC con un método más simple que el que se suele usar, *EADMM* [8]. En nuestro caso se ha implementado el *ADMM* que mejora bastante el rendimiento. Este último método permite optimizarse aún más haciendo uso de la estructura de una matriz que permite resolver un sistema de ecuaciones de forma más eficiente. En este trabajo se toman influencias de trabajos anteriores como [4], [6], [5], [9] y [10].

1.1 MPC

El MPC, por sus siglas Model Predictive Control (Control Predictivo basado en Modelo) [4] es una estrategia de control avanzada que tiene de particular que calcula las acciones de control futuras, necesarias para llegar al objetivo de control (referencia) de forma óptima.

Necesita un modelo de planta que le permita realizar predicciones de los futuros estados del sistema durante un intervalo temporal denominado *horizonte de predicción*. La estrategia de control que se calcula se aplica durante un intervalo finito para evitar errores de modelado.

El intervalo finito que calcula las acciones futuras del sistema recibe el nombre de *horizonte de control*. En esta estrategia de control se aplica siempre la primera acción de control óptima calculada.

Una vez aplicada la acción de control óptima se desplaza el *horizonte de predicción* y se ajustan las nuevas condiciones iniciales para volver a calcular de nuevo el valor óptimo. Este tipo de estrategia de control se puede usar para seguir trayectorias o para problemas de regulación, siendo en este último caso donde más se da.

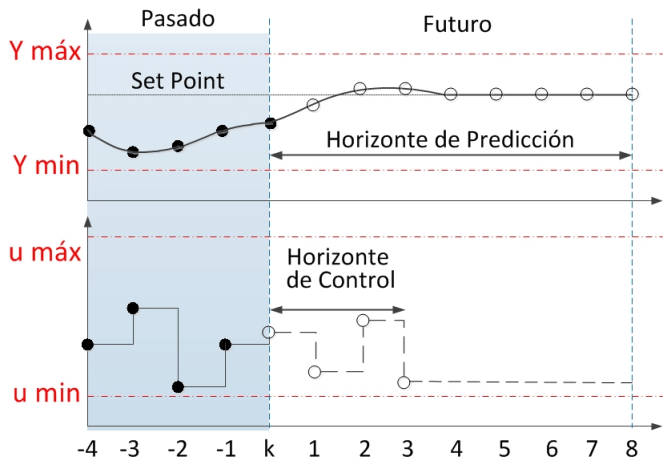

Figura 1.1 Ejemplo de MPC.

1.2 Métodos numéricos

La implementación del MPC requiere de la resolución de un problema de optimización. Para el caso del MPC en sistemas lineales, este problema se podrá plantear como la minimización de una función cuadrática sujetas a restricciones de igualdad (típicamente restricciones lineales de igualdad y desigualdad). Para este tipo de problemas hay muchos métodos numéricos que permiten resolverlo, aunque en este trabajo solo se desarrolla uno, el *ADMM* (Alternating Direction Method of Multipliers) [6] [1]. En posteriores capítulos se profundizará más en este método. En este capítulo se pretende introducir 3 métodos numéricos para resolver este tipo de problemas entre los que también se incluye el *ADMM*.

1.2.1 Método ISTA

Todos los problemas de optimización que se tratan, requieren que estén expresados de forma cuadrática para hacer uso de estos algoritmos:

$$J^* = \min_{z \in Z} \frac{1}{2} z^T H z + q^T z,$$
$$s.t.\ Gz = b.$$

Donde H es una matriz definida positiva y Z es un conjunto convexo.

Cuando se tiene el problema expresado de esta forma se puede aplicar este método directamente:

Algorithm 1 ISTA

1: Se elige el parámetro de tolerancia ε.
2: $k = 0, \lambda_k = 0$.
3: $z_\lambda = \underset{z \in Z}{argmin}\ \frac{1}{2} z^T H z + q^T z + \lambda_k^T (Gz - b)$.
4: $\lambda_{k+1} = \lambda_k + (GH^{-1}G^T)^{-1}(Gz_\lambda - b)$.
5: Si $\| Gz_\lambda - b \|_2 \leqslant \varepsilon$. $\rightarrow$ EXIT.
6: $k = k + 1 \rightarrow$ PASO 3.

λ es una variable dual que está asociada a la restricción de igualdad $Gz = b$.

El problema que presenta este algoritmo es que no tiene una buena tasa de convergencia. En los años 80 Yuri Nesterov desarrolló una nueva familias de métodos llamados acelerados que aumentaban mucho la tasa de convergencia [11]. Dentro de esta familia, Amir Beck y Marc Teboulle desarrollarían el método *FISTA* (Fast ISTA) [1].

1.2.2 Método FISTA

El tipo de problema que resuelve es exactamente el mismo, pero como se ha expuesto antes, la tasa de convergencia es mucho mayor:

Algorithm 2 FISTA

1: Se elije el parámetro de tolerancia ε.
2: $x_1 = x_0 = \lambda_0 = 0$.
3: $k = 1, t_0 = 1$.
4: $t_k = \frac{1}{2}\left(1 + \sqrt{1 + 4t_{k-1}^2}\right)$.
5: $\lambda_k = x_k + \frac{t_{k-1}-1}{t_k}(x_k - x_{k-1})$.
6: $z_\lambda = \underset{z \in Z}{argmin}\ \frac{1}{2} z^T H z + q^T z + \lambda_k^T (Gz - b)$.
7: $x_{k+1} = \lambda_k (GH^{-1}G^T)^{-1}(Gz_\lambda - b)$.
8: Si $\| Gz_\lambda - b \|_2 \leqslant \varepsilon \rightarrow$ EXIT.
9: $k = k + 1 \rightarrow$ PASO 4.

Los algoritmos *FISTA* e *ISTA* están a que el problema a resolver sea estrictamente convexo ($H > 0$). El siguiente algoritmo está libre de esta limitación.

1.2.3 ADMM

A continuación se expone el algoritmo *ADMM* para resolver un problema genérico. Más adelante, se desarrolla su versión específica para la forma del problema cuadrático. Este algoritmo permite resolver problemas de la forma:

$$\begin{aligned} &\underset{\mathbf{x} \in X}{min}\ J(x), \\ &s.t.\ Gx = b. \end{aligned}$$

Se dualiza el problema descomponiendo $J(x) = \psi(x) + Q(x)$ y definiendo una variable dual $y = x$ de manera que el problema quedaría:

$$\begin{aligned} \underset{\mathbf{x} \in X, \mathbf{y} \in \mathbb{R}^n}{min}\ &\psi(x) + Q(x), \\ s.t.\ &Ay = b, \\ &x = y. \end{aligned}$$

Algorithm 3 *ADMM* formulación genérica

1: Se elige el parámetro de tolerancia ε.
2: Se eligen $\rho > 0$ y $x_0 \in X$.
3: $k = 0, \lambda_0 = 0$.
4: $y_{k+1} = arg\ \underset{y \in Y}{min}\ Q(y) - \lambda_k^T y + \frac{\rho}{2} \| x_k - y \|_2^2$.
5: $x_{k+1} = arg\ \underset{x \in X}{min}\ \psi(x) + \lambda_k^T x + \frac{\rho}{2} \| x - y_{k+1} \|_2^2$.
6: $\lambda_{k+1} = \lambda_k + \rho(x_{k+1} - y_{k+1})$.
7: Si $max\{\| x_{k+1} - y_{k+1} \|_2^2, \| x_{k+1} - x_k \|_2^2\} \leqslant \varepsilon \rightarrow$ EXIT.
8: $k = k + 1 \rightarrow$ PASO 4.

1.3 Sistemas en semi banda

En muchas de las formulaciones del MPC existente, y como se verá en capítulos siguientes, la implementación de los algoritmos anteriormente mencionados requieren de la solución de sistemas de ecuaciones en banda y semi-banda.

Podemos definir una matriz en banda de la siguiente forma $M_{ij} = 0$, para $\| j-i \| > L_{Banda}$. L_{Banda} es el ancho de la banda. Una matriz diagonal tendría un ancho de banda de 1. Se muestra un ejemplo cuando el ancho es 2. En el ejemplo se tiene una matriz de $M \in \mathbb{R}^{6\times 6}$. Se puede apreciar fácilmente que el ancho es 2 observando la esquina superior izquierda de la matriz. si nos vamos a la primera fila, no hay elemento nulo hasta la tercera columna, lo mismo ocurre si empezamos por la primera columna. Por lo tanto se puede ver fácilmente que $M_{ij} = 0$, para $\| j-i \| \geq 2$.

$$M = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 3 & 4 & 7 & 0 & 0 & 0 \\ 0 & 3 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 9 & 0 \\ 0 & 0 & 0 & 13 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 3 \end{bmatrix}.$$

Este tipo de matrices se pueden descomponer de muchas formas permitiendo resolver sistemas de ecuaciones de la forma $Mx = y$ siendo M una matriz en banda, de formas mucho más eficientes. Softwares como *Matlab* implementan ya algoritmos que descomponen estas matrices de tal forma que se reduce mucho la complejidad computacional. El comando *decomposition* descompone una matriz de forma automática.

En el contexto del MPC, los sistemas de ecuaciones en bandas se corresponden a matrices simétricas y definidas positivas, para las que existen métodos más eficaces que las resuelven.

Cuando tratamos de resolver un sistema de ecuaciones, la complejidad de un sistema es de $O(n^3)$ [7]. Si ese sistema de ecuaciones es en banda, o lo que es lo mismo, tenemos que $Mx = y$ donde M es una matriz en banda, la complejidad se puede reducir a a $O(n \cdot AB^2)$. Por lo tanto cuanto más pequeña la banda menos compleja resulta la resolución.

Esto va a tomar relevancia porque cuando se resuelve el problema del MPC, hay que resolver muchas veces un sistema de tipo $Mx = y$, por lo que es muy conveniente optimizar lo máximo posible la resolución de este tipo de sistemas, que es donde recaerá la mayor carga computacional de la resolución del problema.

El problema que nos vamos a encontrar es que la matriz que define al sistema es en semi-banda, lo que quiere decir que hay elementos no nulos fuera de la banda debido a la perturbación por matrices de menor rango que la original. Por eso es conveniente hallar un método que permita la descomposición de una matriz en semi-banda en una banda más otra matriz que cargue sus elementos no nulos ajenos a la banda. En posteriores capítulos se desarrollará un método que permita descomponer una matriz en semi-banda.

1.3.1 Descomposiciones en *Matlab*

Matlab tiene muchas opciones para realizar descomposiciones de matrices de forma totalmente automática. Entre ellas, está el comando *decomposition* que devuelve la matriz descompuesta en la descomposición que se le indique (Cholesky, LU, QR,...). Este comando tiene la particularidad de que ahorra un pasos respecto a descomponer de forma normal. La llamada a la función devuelve un elemento que se puede usar directamente para resolver el sistema de ecuaciones con el que se esté tratando. Por ejemplo, si se quisiese resolver varias veces un sistema clásico de $Ax = b$ donde A se mantiene constante y b puede variar, entonces podemos expresarlo de la siguiente manera:

$$\begin{aligned} dA &= decomposition(A), \\ R &= chol(A), \\ x &= dA \setminus b, \\ x &= R \setminus (R' \setminus b). \end{aligned}$$

En el ejemplo se ha usado la descomposición de Cholesky. A continuación se introduce un pequeño recordatorio acerca de esta descomposición y otra también muy usada, la descomposición LU.

Recordatorio

La descomposición *LU* es una forma de factorización de una matriz como el producto de una matriz triangular inferior con una superior [14].

$$A = L \cdot U.$$

La descomposición de Cholesky es una factorización solo aplicable a matrices matrices simétricas y definidas positivas. Descompone la matriz en un producto de una matriz triangular inferior y su traspuesta [13].

$$A = R \cdot R'.$$

Resolver una ecuación usando cualquiera de estás dos descomposiciones se realizaría de la siguiente forma:

$$\begin{aligned} Ax &= b, \\ x &= L \setminus (U \setminus b), \\ x &= R \setminus (R' \setminus b). \end{aligned}$$

Cholesky es una opción muy interesante para las matrices en bandas que estudiaremos ya que estas serán definidas y positivas, por lo que será el método más adecuado para este tipo de matrices.

Usar este comando es relativamente sencillo, se puede usar dándole solo la matriz $dA = decomposition(A)$, también se le puede especificar el tipo de descomposición que se le quiera aplicar a la matriz $dA = decomposition(A,tipo)$ estos tipos son "qr", "cod", "lu", "ldl", "chol", "triangular", "permutedTriangular", "banded", "hessenberg" y "diagonal". Además también se le puede indicar si se quiere usar solo la porción triangular o inferior ("upper" o "lower") de A cuando se usan los tipos "ldl", "chol" o "triangular" $dA = decomposition(A,chol,ParteTriangular)$.

Decomposition hace muy cómoda la implementación de algoritmos eficientes en *matlab*, pero cuando se quiera trabajar con sistemas embebidos se suele implementar la descomposición en *C* o *C++* ya sea en línea o fuera de línea y se usaría en el bucle de optimización.

1.4 Estructura del proyecto

Este proyecto está dividido en 4 capítulos más el capítulo introductorio y un anexo en el que se exponen los códigos desarrollados en el transcurso del mismo.

Introducción a la optimización

Se realizará una pequeña introducción a la optimización convexa y además se entrará ya en profundidad en el método con el que se trabajará en el resto del documento.

MPC Regulación

Se presentan los conceptos básicos del MPC y además se emepiezan ya a trabajar con una de sus formulaciones.

MPC para seguimiento

Es el capítulo más importante del proyecto y en él se expone una formulación del MPC algo más compleja y se desarrolla una forma de conseguir que el método numérico *ADMM* resuelva el problema de optimización más rápido utilizando la estructura en semi-banda del problema.

Conclusiones

Se analizan los resultados del proyecto y se reflexiona sobre futuras vías de investigación.

2 Introducción a la optimización convexa y al *ADMM*

El paso más importante que puede dar alguien. No es el primero, ¿verdad? Es el próximo. Siempre el próximo paso

BRANDON SANDERSON

En este capítulo se pretende realizar una sencilla introducción al problema de optimización convexa y finalizar explicando el algoritmo *ADMM* que jugará un rol clave en el desarrollo de este trabajo.

2.1 Optimización convexa

La formulación que se plantea es un problema de optimización que puede ser tratado como un problema de optimización convexa. Por ello vamos a dedicar una sección de este capítulo para introducirnos en la optimización convexa.

2.1.1 Conjuntos convexos

Un conjunto $Q \subseteq \mathbb{R}^n$ es convexo para todo (x_a, x_b) de elementos de Q:

$$(1-\eta)x_a + \eta x_b \in Q, \forall \eta \in [0,1].$$

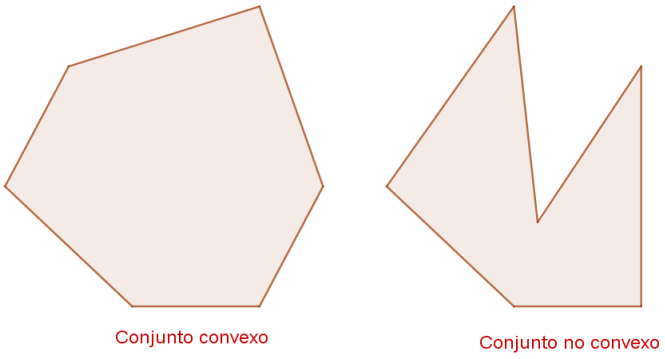

Figura 2.1 Conjunto convexo y conjunto no convexo.

2.1.2 Funciones convexas y funciones cóncavas

Una función h es convexa si y solo si su dominio es convexo y para todo $\alpha \in [0,1]$:

$$h(\alpha x + (1-\alpha)y) \leqslant \alpha h(x) + (1-\alpha)h(y), \quad \forall x \in dom(h), \forall y \in dom(h).$$

Y es estrictamente convexa si:

$$h(\alpha x + (1-\alpha)y) < \alpha h(x) + (1-\alpha)h(y), \quad \forall x \in dom(h), \forall y \in dom(h).$$

Una función cóncava es lo opuesto a una función convexa. Una función h es cóncava si y solo si su dominio es convexo y para todo $\alpha \in [0,1]$:

$$h(\alpha x + (1-\alpha)y) \geq \alpha h(x) + (1-\alpha)h(y), \quad \forall x \in dom(h), \forall y \in dom(h).$$

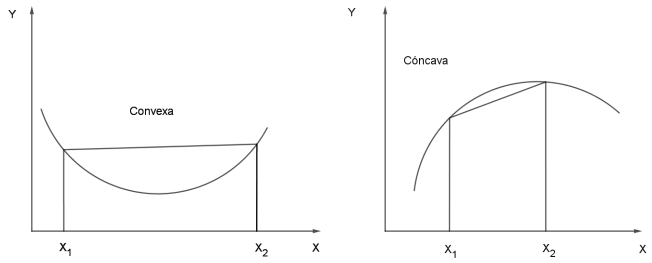

Figura 2.2 Función convexa y función cóncava.

Las funciones convexas que más aparecen en el contexto del MPC lineal son las cuadráticas.

$$h(x) = \frac{1}{2}x^T Hx + q^T x + c.$$

La convexidad está garantizada si H es una matriz semidefinida positiva. Y si H es definida positiva la función será estrictamente convexa.

2.1.3 Epígrafo

El epígrafo de una función se define:

$$epi(h) = \{(x,t) \in dom(h) \times \mathbb{R} : h(x) \leqslant t\}.$$

La función h es convexa si y solo si, su epígrafo es un conjunto convexo.

2.1.4 Subgradientes

Supongamos que tenemos una función h que es convexa. El vector g es denominado subgradiente en el punto x_0, si para todo $x \in dom(h)$ se cumple:

$$h(x) \geq h(x_0) + \langle g, x - x_0 \rangle.$$

El conjunto de todos los subgradientes de h en x_0, $\partial h(x_0)$ se llama subdiferencial de la función h en el punto x_0. El concepto de subdiferencial permite caracterizar los valores de x que optimizan una función convexa. En particular, una condición que se debe cumplir para que $x^* = arg \min_{x \in dom(h)} h(x)$ es que 0 pertenezca al conjunto de todos los subdiferenciales de h en x^*.

$$0 \in \partial h(x^*).$$

2.1.5 Subgradientes para una función cuadrática

Teniendo una función cuadrática $h(x) = \frac{1}{2}x^T Hx + q^T x + c$ donde $H \geq 0$ se puede obtener para un punto x_0 un subgradiente de forma sencilla. Se puede demostrar que para todo x:

$$\begin{aligned} h(x) = &\quad \frac{1}{2}(x+x_0-x_0)^T H(x+x_0-x_0) + q^T x + c, \\ = &\quad \frac{1}{2}x_0^T Hx_0 + x_0^T H(x-x_0) + \frac{1}{2}(x-x_0)^T H(x-x_0) + q^T x + c, \\ \geq &\quad \frac{1}{2}x_0^T Hx_0 + x_0^T H(x-x_0) + q^T x + c, \\ = &\quad h(x_0) + x_0^T H(x-x_0) + q^T (x-x_0), \\ = &\quad h(x_0) + (Hx_0+q)^T (x-x_0), \\ \rightarrow &\quad Hx_0 + q \in \partial h(x_0). \end{aligned}$$

Esto tiene sentido ya que $H \geq 0 \rightarrow (x-x_0)^T H(x-x_0) \geq 0$.

2.1.6 Formulación primal

El problema de optimización puede ser expresado de la siguiente forma:

$$J^* = \min_{\mathbf{z}\in Z} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + \psi(\mathbf{z}), \tag{2.1a}$$

$$s.t \quad G\mathbf{z} = b. \tag{2.1b}$$

Donde:

1. $Z \subseteq \mathbb{R}^n$ es un conjunto convexo.
2. La matriz H es definida positiva ($H \succ 0$).
3. $\psi : \mathbb{R}^n \rightarrow \psi$ es una función convexa en Z.
4. Se supone que el problema es estrictamente factible $\rightarrow \exists\, \mathbf{z} \subseteq Z$ que cumple la condición $G\mathbf{z} = b$.

2.1.7 Enfoque dual

La formulación primal (2.1a) puede resolverse de una forma más sencilla gracias al enfoque dual que permite relajar las restricciones de igualdad (2.1b). Comenzamos definiendo el funcional dual como:

$$\varphi(\lambda) = \min_{\mathbf{z}\in Z} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + \psi(\mathbf{z}) + \lambda^T (G\mathbf{z} - b). \tag{2.2}$$

Cabe resaltar que cuando tratemos con funciones estrictamente convexas, estas tienen solo un mínimo.

Como el problema es estrictamente convexo ($H \succ 0$), existe una única solución que denotaremos como $\mathbf{z}(\lambda)$:

$$\mathbf{z}(\lambda) = arg\min_{\mathbf{z}\in Z} \mathbf{z}^T H\mathbf{z} + \psi(\mathbf{z}) + \lambda^T (G\mathbf{z} - b). \tag{2.3}$$

Sabemos que la solución óptima debe satisfacer la condición de igualdad (2.1b). Por lo que si denotamos la solución óptima como $\mathbf{z}^*$ podemos asegurar que $G\mathbf{z}^* = b$ y $\mathbf{z}^* \in Z$. Es demostrable que el funcional dual es menor o igual para todo λ a la función de coste optimizada.

$$\begin{aligned} \varphi(\lambda) = &\quad \min_{\mathbf{z}\in Z} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + \psi(\mathbf{z}) + \lambda^T (G\mathbf{z} - b), \\ \leqslant &\quad \frac{1}{2}\mathbf{z}^{*T} H\mathbf{z}^* + \psi(\mathbf{z}^*) + \lambda^T (G\mathbf{z}^* - b), \\ = &\quad \frac{1}{2}\mathbf{z}^{*T} H\mathbf{z}^* + \psi(\mathbf{z}^*) = J^*, \\ J^* \geq &\quad \varphi(\lambda), \forall\lambda. \end{aligned} \tag{2.4}$$

Con esto en mente pongamos la siguiente situación:

1. $H \succ 0$ es diagonal: H=diag(h_1,h_2,...,h_n).
2. Z es una caja: $Z = \{\mathbf{z} \in \mathbb{R}^n : z_i^- \leqslant z_i \leqslant z_i^+\}$.
3. $\psi(\mathbf{z}) = q^T\mathbf{z}$.

A la componente i-ésima del vector que resulta de la operación $G^T\lambda + q$ se denotará como c_i

$$\begin{aligned}\mathbf{z}(\lambda) = &\quad arg\ \min_{\mathbf{z}\in\mathbb{Z}} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + q^T\mathbf{z} + \lambda^T(G\mathbf{z} - b),\\ = &\quad arg\ \min_{\mathbf{z}\in\mathbb{Z}} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + (G^T\lambda + q)^T\mathbf{z},\\ = &\quad arg\ \min_{\mathbf{z}\in\mathbb{Z}} \sum_{i=1}^{n} h_i\frac{z_i^2}{2} + c_i z_i. \end{aligned} \tag{2.5}$$

Viendo la ecuación (2.5) se puede observar que para obtener $\mathbf{z}(\lambda) = [z_1(\lambda),...,z_n(\lambda)]$ hay que resolver n problemas de optimización escalares.

$$z_i(\lambda) = \quad arg \min_{z_i\in[z_i^-,z_i^+]} h_i\frac{z_i^2}{2} + c_i z_i, \quad i = 1,...,n. \tag{2.6}$$

En caso de que no hubiese restricciones de caja, los valores óptimos serían aquellos que hacen cero al gradiente respecto $\mathbf{z} : h_i z_i(\lambda) + c_i = 0,\ i = 1,...,n$. Como hay restricciones de caja hay que comprobar si el óptimo sin restricciones $-\frac{c_i}{h_i}$ pertenece a la caja.

$$z_i(\lambda) = \begin{cases} -\frac{c_i}{h_i}, & \text{si } z_i^- \leqslant -\frac{c_i}{h_i} \leqslant z_i^+ \\ z_i^-, & \text{si } -\frac{c_i}{h_i} < z_i^- \\ z_i^+, & \text{si } -\frac{c_i}{h_i} > z_i^+ \end{cases}. \tag{2.7}$$

2.1.8 $\varphi(\lambda)$ es una función cóncava

Se va denotar al vector que contiene todos los $z_i(\lambda)$ obtenidos de la ecuación (2.7) como $\mathbf{z}_\lambda$. Se puede demostrar que el funcional dual es una función cóncava.

$$\mathbf{z}_\lambda = \quad arg\ \min_{\mathbf{z}\in\mathbb{Z}} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + \psi(z) + \lambda^T(G\mathbf{z} - b). \tag{2.8}$$

Obtenemos:

$$\begin{aligned}\varphi(\lambda + \Delta\lambda) = &\quad \min_{\mathbf{z}\in\mathbb{Z}} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + \psi(\mathbf{z}) + (\lambda + \Delta\lambda)^T(G\mathbf{z} - b),\\ \leqslant &\ \frac{1}{2}\mathbf{z}_\lambda^T H\mathbf{z}_\lambda + \psi(\mathbf{z}_\lambda) + (\lambda + \Delta\lambda)^T(G\mathbf{z}_\lambda - b),\\ = &\quad \varphi(\lambda) + \Delta\lambda^T(G\mathbf{z}_\lambda - b). \end{aligned} \tag{2.9}$$

En el punto óptimo de $\varphi(\lambda)$ se encuentra también la solución al problema de optimización convexa. Dado que se ha asumido que el problema primal es extrictamente factible, el máximo de la función dual coindide con el máximo de la función primal. Es decir, no existe brecha de dualidad [3].

Sabemos que $\varphi(\lambda)$ es una cota inferior (2.4) respecto de λ. Hay muchas formas de resolver este problema pero en este trabajo se centrará principalmente en el algoritmo *ADMM*.

2.2 Algoritmo *ADMM*

En el contexto de este trabajo consideramos tres algoritmos: *ISTA*, *FISTA* y *ADMM*. De estos 3, nos centramos en el último pues no solo el más eficiente de los 3 sino que es el único que permite afrontar una mayor gama

de problemas, pues no sufre limitaciones como que el objetivo primal deba ser estrictamente convexo o que la matriz que defina el término cuadrático deba ser diagonal [2] [6].

2.2.1 Formulación de problema

Comenzamos definiendo el siguiente problema de optimización

$$\begin{aligned} \min_{\mathbf{x}\in X} \quad & J(\mathbf{x}), \\ s.t. \quad & G\mathbf{x} = b. \end{aligned}$$

J es una función que recibe una variable que pertenece a $\mathbb{R}^n$ y devuelve un escalar. La dificultad radica en que obtener una x que cumpla las restricciones impuestas no es algo trivial. Aunque, si podemos afirmar que encontrar una $\mathbf{x} \in X$ (asumiremos siempre que X es una caja) y por otro lado otra que cumpla $G\mathbf{x} = b$ es normalmente más fácil.

Vamos a reescribir la función $J(\mathbf{x})$ como la suma de dos funciones $\psi(\mathbf{x})$ y $Q(\mathbf{x})$. Esta última es una función cuadrática convexa, que no es necesario que sea estrictamente convexa.

$\psi(\mathbf{x})$ es una función separable, lo que implica que se puede reescribir como un sumatorio de funciones escalares que dependen de cada componente por separado.

$$J(\mathbf{x}) = \quad \psi(\mathbf{x}) + Q(\mathbf{x}), \tag{2.10}$$

$$\psi(\mathbf{x}) = \quad \psi(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}) = \sum_{i=1}^{n} h_i(x_i). \tag{2.11}$$

Reescribimos el problema de la siguiente manera:

$$\min_{\mathbf{x}\in X, \mathbf{y}\in\mathbb{R}^n} \quad \psi(\mathbf{x}) + Q(\mathbf{y}), \tag{2.12}$$

$$s.t. \quad G\mathbf{y} = b, \tag{2.13}$$

$$\mathbf{x} = \mathbf{y}. \tag{2.14}$$

Podemos dualizar la restricción $\mathbf{x} = \mathbf{y}$ obteniendo:

$$\varphi_0(\lambda) = \quad \min_{\mathbf{x}\in X, \mathbf{y}\in\mathbb{R}^n} \psi(\mathbf{x}) + Q(\mathbf{y}) + \lambda^T(\mathbf{x}-\mathbf{y}). \tag{2.15}$$

Definimos: $Y = \{\mathbf{y} : G\mathbf{y} = b\}$ y obtenemos:

$$\varphi_0(\lambda) = \quad \min_{\mathbf{x}\in X}\{\psi(\mathbf{x}) + \lambda^T\mathbf{x}\} + \min_{\mathbf{y}\in Y}\{Q(\mathbf{y}) - \lambda^T\mathbf{y}\}. \tag{2.16}$$

Tenemos dos variables que hallar:

$$\mathbf{x}_\lambda = \quad arg\min_{\mathbf{x}\in X} \psi(\mathbf{x}) + \lambda^T\mathbf{x}, \tag{2.17}$$

$$\mathbf{y}_\lambda = \quad arg\min_{\mathbf{y}\in Y} Q(\mathbf{y}) - \lambda^T\mathbf{y}. \tag{2.18}$$

Estas dos variables son fáciles de computar, ya que para hallar (2.17) solo hay que resolver n problemas escalares sujetos a restricciones intervalares y para (2.18) es una minimización cuadrática sujeta a restricciones lineales. Podemos redefinir $\varphi_0(\lambda)$ y añadirle un término cuadrático que penalice si no se cumple la restricción (2.14). Esto se añade con una constante positiva ρ. El objetivo de este término es evitar que las variables primales cambien de formar brusca creando una dependencia suave respecto a λ.

$$\varphi_\rho(\lambda) = \quad \min_{\mathbf{x}\in X, \mathbf{y}\in\mathbb{R}^n} \psi(\mathbf{x}) + Q(\mathbf{y}) + \lambda^T(\mathbf{x}-\mathbf{y}) + \frac{\rho}{2} \parallel \mathbf{x}-\mathbf{y} \parallel_2^2 . \tag{2.19}$$

Como se puede observar a simple vista, en esta formulación x e y dejarían de estar desacopladas. Esto no supone ningún inconveniente por que el propio algoritmo *ADMM* desacopla el problema fijando primero y para obtener x, para luego obtener y con el valor de x fijado a lo hallado anteriormente.

2.2.2 Implementación *ADMM*

ADMM formulación genérica

1: Se elige el parámetro de tolerancia ε.
2: Se eligen $\rho > 0$ y $x_0 \in X$.
3: $k = 0, \lambda_0 = 0$.
4: $y_{k+1} = arg\ \min_{y \in Y} Q(y) - \lambda_k^T y + \frac{\rho}{2} \parallel x_k - y \parallel_2^2$.
5: $x_{k+1} = arg\ \min_{x \in X} \psi(x) + \lambda_k^T x + \frac{\rho}{2} \parallel x - y_{k+1} \parallel_2^2$.
6: $\lambda_{k+1} = \lambda_k + \rho(x_{k+1} - y_{k+1})$.
7: Si $max\{\parallel x_{k+1} - y_{k+1} \parallel_2^2, \parallel x_{k+1} - x_k \parallel_2^2\} \leqslant \varepsilon \rightarrow$ EXIT.
8: $k = k + 1 \rightarrow$ PASO 4.

En el capítulo anterior (1) vimos el algoritmo que implementa el *ADMM*, en este capítulo se pretende desarrollar el método para su implementación. En el problema que vamos a resolver Q es cuadrática, $\psi()$ es 0 y X son restricciones en caja. El primer problema que resolvemos dentro del algoritmo (3) es calcular $\mathbf{y}_\lambda$. Comenzamos definiendo la función $Q(\mathbf{y})$ de la siguiente manera:

$$Q(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T H\mathbf{y} + q^T\mathbf{y} + l. \tag{2.20}$$

Entonces podemos definir:

$$\begin{aligned} \Gamma_k(\mathbf{y}) &= Q(\mathbf{y}) - \lambda_k^T\mathbf{y} + \frac{\rho}{2} \parallel \mathbf{x}_k - \mathbf{y} \parallel_2^2, \\ &= \frac{1}{2}\mathbf{y}^T H\mathbf{y} + q^T\mathbf{y} + l - \lambda_k^T\mathbf{y} + \frac{\rho}{2} \parallel \mathbf{x}_k - \mathbf{y} \parallel_2^2, \\ &= \frac{1}{2}\mathbf{y}^T (H + \rho I)\mathbf{y} + (q - \lambda_k - \rho\mathbf{x}_k)^T\mathbf{y} + l + \frac{\rho}{2}\mathbf{x}_k^T\mathbf{x}_k. \end{aligned} \tag{2.21}$$

Si tenemos que $Y = \{\mathbf{y} : G\mathbf{y} = b\}$ entonces la solución del sistema vendrá dada por el siguiente sistema de ecuaciones [3]:

$$\begin{bmatrix} H + \rho I & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y}_{k+1} \\ \nu \end{bmatrix} = \begin{bmatrix} \lambda_k + \rho x_k - q \\ b \end{bmatrix}. \tag{2.22}$$

Tenemos dos incógnitas, la solución y_{k+1} y ν. De la primera igualdad se obtiene:

$$\begin{aligned} &(H + \rho I)\mathbf{y}_{k+1} + G^T\nu = \lambda_k + \rho\mathbf{x}_k - q, \\ &\mathbf{y}_{k+1} = (H + \rho I)^{-1}(\lambda_k + \rho\mathbf{x}_k - q - G^T\nu). \end{aligned} \tag{2.23}$$

Con el valor obtenido para y_{k+1} en (2.23) se introduce la restricción $Gy_{k+1} = b$ para obtener:

$$\begin{aligned} &G(H + \rho I)^{-1}(\lambda_k + \rho\mathbf{x}_k - q - G^T\nu) = b, \\ &G(H + \rho I)^{-1}G^T\nu = -b + G(H + \rho I)^{-1}(\lambda_k + \rho\mathbf{x}_k - q), \\ &\nu = (G(H + \rho I)^{-1}G^T)^{-1}(-b + G(H + \rho I)^{-1}(\lambda_k + \rho\mathbf{x}_k - q)). \end{aligned} \tag{2.24}$$

Se tiene que ν se halla directamente de la ecuación (2.24) y luego sustituyendo en (2.23) se obtendría y_{k+1} resolviendo así el primer problema del algoritmo (paso 4).

A continuación vamos a desarrollar el segundo problema de este algoritmo (paso 5), $\mathbf{x}_{k+1}$. En este punto se tiene todo lo que se necesita para poder resolverlo. Vamos a suponer que $\psi(\cdot) = 0$ y vamos a denotar

$c = \rho \mathbf{y}_{k+1} - \lambda_k$.

$$\begin{aligned} \Upsilon_k = & \quad \psi(\mathbf{x}) + \lambda_k^T \mathbf{x} + \frac{\rho}{2} \parallel \mathbf{x} - \mathbf{y}_{k+1} \parallel_2^2, & (2.25) \\ = & \quad \frac{\rho}{2}\mathbf{x}^T\mathbf{x} + (\lambda_k - \rho \mathbf{y}_{k+1})^T x + \frac{\rho}{2}\mathbf{y}_{k+1}^T\mathbf{y}_{k+1}, \\ = & \quad \frac{\rho}{2}\mathbf{x}^T\mathbf{x} - c^T\mathbf{x} + \frac{\rho}{2}\mathbf{y}_{k+1}^T\mathbf{y}_{k+1}. & (2.26) \end{aligned}$$

Como $\mathbf{y}_{k+1}$ está fijado podemos obviar el término cuadrático en $\mathbf{y}_{k+1}$ ya que no depende de $\mathbf{x}$. Por lo tanto, el problema de optimización quedaría de la siguiente forma:

$$\mathbf{x}_{k+1} = \quad arg \min_{\mathbf{x} \in X} \frac{\rho}{2}\mathbf{x}^T\mathbf{x} - c^T\mathbf{x}. \tag{2.27}$$

Como este problema se desarrolla en una restricción de caja de la forma $X = \{\mathbf{x} \in \mathbb{R}^n : \underline{x} \leqslant \mathbf{x} \leqslant \bar{x}\}$, se tiene que la componente i-ésima de $\mathbf{x}_{k+1}$ viene dada por:

$$x_{k+1,i} = \quad \begin{cases} \frac{c_i}{\rho}, & \text{si } \frac{c_i}{\rho} \in [\underline{x}_i, \bar{x}_i] \\ \underline{x}_i, & \text{si } \frac{c_i}{\rho} < \underline{x}_i \\ \bar{x}_i, & \text{si } \frac{c_i}{\rho} > \bar{x}_i \end{cases} . \tag{2.28}$$

Se actualizaría ahora la variable dual λ (paso 6). Cuando se implemente este algoritmo, en el paso 7 se generan valores residuales que serán cero cuando se alcance el valor óptimo. En la implementación, no tienen por que llegar cero, sino que se pone una cota dada por el valor de ε.

Para finalizar este capítulo se expone una función en *Matlab* que lo implementa.

Código 2.1 Implementación ADMM.

```
%-----------------------------------
%      ENTRADAS
%-----------------------------------
%   - epsilon: Tolerancia del problema.
%   - ro: Constante para regularizar el problema.
%   - x0: Condición inicial al problema (después de resolverlo por primera vez
    se le da siempre la solución en las siguientes ejecuciones)
%   - xmas y xmenos:Restricciones de caja del problema.
%   - Hroinv: Inversa la matriz H+\rho del problema QP.
%   - G: Matriz que define más restricciones de igualdad.
%   - b: Vector de las restricciones de igualdad.
%   - dW: Descomposición de la matriz dada por W = G*Hinv*G'. dW=decomposition(
    W,'banded')
%-----------------------------------
%      SALIDAS
%-----------------------------------
%   -xk: solución
%   -k: Número de iteraciones realizadas.
function [xk,k] = ADMM(epsilon,ro,x0,q,xmas,xmenos,Hroinv,G,b,dW)
% Condiciones inciales
k=0;
xk=x0;
xk_ant=xk;
yk=x0;
lambdak=zeros(length(xk),1);
while (max(norm(xk-yk),norm(xk-xk_ant)) > epsilon || k==0 )
    xk_ant=xk;
    lambdak_ant=lambdak;
```

```
    %----------------------yk+1----------------------
    v=dW\(-b+G*Hroinv*(lambdak+ro*xk-q));
    yk=Hroinv*(lambdak+ro*xk_ant-q-G'*v);
    %-----------------------------------------------
    %----------------------xk+1-----------------
    c=ro*yk-lambdak_ant;
    xk=ARGminADMM(ro,xmas,xmenos,c);
    %-----------------------------------------
    %---------------------Lambdak+1-------------
    lambdak=lambdak_ant+ro*(xk-yk);
    %-----------------------------------------
    k=k+1;
end
end
```

Dentro de este código hay otra función cuyo objetivo es resolver el problema con restricciones cajas.

Código 2.2 ARG mínimo para ADMM.

```
%-----------------------------------
%      ENTRADAS
%-----------------------------------
%   - ro: Constante para regularizar el problema.
%   - xmas: Restricción de caja superior.
%   - xmas: Restricción de caja inferior.
%   - c: \rho*ro*y_{k+1}-\lambda_{k}
%-----------------------------------
%      SALIDAS
%-----------------------------------
%   -x: solución al problema con restricción de caja
function [x] = ARGminADMM(ro,xmas,xmenos,c)
n=lenght(c);
x=zeros(n,1);
for i=1:n
x(i)=max(xmenos(i),min(c(i)/ro,xmas(i)));
end
end
```

Con estos conceptos claros, se puede empezar ya a introducir en el Control Predictivo. En el siguiente capítulo se introduce el *MPC* y una de sus formulaciones, el MPC de regulación.

3 MPC de Regulación

El caos siempre derrota al orden porque está mejor organizado

Terry Pratchett

En esta sección del trabajo se va a desarrollar una introducción al *MPC* y una particularización de la formulación general. Se va no solo a describir la formulación del *MPC*, también se desarrollarán distintos métodos para su resolución. Uno de los métodos se ve en mayor profundidad en el trabajo [6].

3.1 Introducción al Control Predictivo

3.1.1 Sistemas a controlar

Vamos a considerar un modelo discreto-temporal, lineal, en espacio de estados [4].

$$x(t+1) = Ax(t) + Bu(t). \tag{3.1}$$

En esta formulación del espacio de estados n y m corresponden corresponden a las dimensiones del sistemas. n equivale al número de estados y m al de entradas del sistemas. Esto equivale a que $x(t) \in \mathbb{R}^n$ y $u(t) \in \mathbb{R}^m$. Las dimensiones de las matrices del sistema quedarían $A \in \mathbb{R}^{n\times n}$ y $B \in \mathbb{R}^{n\times m}$.

El objetivo del control es dirigir el sistema a una referencia dada (x_r,u_r) mientras se respetan las restricciones del sistema. Estas restricciones se asumen que afectan tanto a los estados como a las entradas y que son de tipo caja.

Definiendo:

$$Y = \{(x,u) \in \mathbb{R}^n \times \mathbb{R}^m : \underline{x} \leqslant x \leqslant \overline{x}, \underline{u} \leqslant u \leqslant \overline{u}\},$$

las restricciones sobre los estados y entradas son:

$$(x(t),u(t)) \in Y \subseteq \mathbb{R}^n \times \mathbb{R}^m, \forall t.$$

En la restricción en caja tenemos que $\underline{x} \in \mathbb{R}^n$, $\overline{x} \in \mathbb{R}^n$, $\underline{u} \in \mathbb{R}^m$ y $\overline{u} \in \mathbb{R}^m$. Matemáticamente son los valores límites tanto superiores como inferiores que pueden tomar las variables del sistema. Físicamente pueden significar varias cosas, por ejemplo si se está controlando un sistema, puede que las acciones de control que gobiernan a los actuadores se limiten a unos valores máximos y mínimos hasta ciertos valores para evitar desgastes innecesarios por lo que los valores de la actuación quedarían restringidos a un rango fijo. Con los estados ocurre algo parecido. Si se está trabajando con alguna reacción química puede ser interesante mantener variables como la temperatura o la presión en un rango fijo (restricción de caja).

A continuación se realizan una serie de suposiciones con el fin de poder formular el problema [6].
1º) El modelo es controlable.

2º) La restricción X no tiene un interior vacío.

La primera suposición implica que la matriz de controlabilidad tiene rango igual a n o sea tiene rango completo. La matriz de controlabilidad de un sistema $n \times m$ se define de la siguiente manera:

$$C = \begin{bmatrix} B & AB & A^2B & A^3B & \dots & A^{n-1}B \end{bmatrix}.$$

La segunda suposición, implica directamente que $\underline{x} < \overline{x}$ y $\underline{u} < \overline{u}$. Se asume que la referencia es un punto de consigna (por partes) constante, es decir, la referencia se mantiene constante durante un intervalo de tiempo desconocido, tras el cual puede saltar a otro valor.

3.1.2 Introducción a la formulación del MPC

El MPC es una estrategia de control avanzada. La acción de control es obtenida en cada muestreo de la solución de un problema de optimización en el que un modelo predictivo es usado para hallar el futuro del sistema sobre un horizonte de predicción [6].

Consideramos la formulación lineal del MPC descrita por el siguiente problema de optimización paramétrica.

$$\min_{\mathbf{x},\mathbf{u}} \quad \left\{ J \doteq \sum_{j=0}^{N-1} \ell_j(\mathbf{x},\mathbf{u};x_r,u_r) + V_t(\mathbf{x},\mathbf{u};x_r,u_r) \right\} \tag{3.2a}$$

$$s.t. \quad x_{j+1} = Ax_j + Bu_j, j \in \mathbb{Z}_0^{N-1}, \tag{3.2b}$$

$$x_0 = x(t), \tag{3.2c}$$

$$(x_j,u_j) \in Y, j \in \mathbb{Z}_0^{N-1}, \tag{3.2d}$$

$$x_N \in \chi_t. \tag{3.2e}$$

$\mathbf{x} = (x_0,\dots,x_N)$ y $\mathbf{u} = (u_0,\dots,u_{N-1})$ son vectores que contienen los estados $x_j \in \mathbb{R}^n$ y las acciones $u_j \in \mathbb{R}^m$ predichas. El par $x_r \in \mathbb{R}^n$ y $u_r \in \mathbb{R}^m$ son las referencias que se le da al sistema. $x(t)$ es el estado actual del sistema cuando se procede a resolver el problema. Y es un conjunto que contiene las restricciones de cajas y χ_t es el set terminal, que se suele traducir en una restricción de igualdad. ℓ_j y V_t son la función de coste de fase para cada paso j y la función de coste terminal.

La estabilidad del MPC que se diseñe es muy dependiente de la función de coste, por lo que elegir bien l_j y V_t es muy importante. Las restricciones juegan también un papel muy importante ya que pueden llevar a que el problema deje de ser factible.

Una vez se resuelve el problema, solo se usa la primera componente del vector $\mathbf{u}$ para aplicar la acción de control.

3.2 Formulaciones del MPC Regulación

El MPC Regulación presenta dos formulaciones con una diferencia en las restricciones. La primera de todas, tiene una restricción terminal de igualdad.

$$\min_{\mathbf{x},\mathbf{u}} \sum_{j=0}^{N-1} \| x_j - x_r \|_Q^2 + \| u_j - u_r \|_R^2, \tag{3.3a}$$

$$s.t. \, x_{j+1} = A_j + Bu_j, \quad j \in \mathbb{Z}_0^{N-1}, \tag{3.3b}$$

$$x_0 = x(t), \tag{3.3c}$$

$$\underline{x} \leqslant x_j \leqslant \overline{x}, \quad j \in \mathbb{Z}_1^{N-1}, \tag{3.3d}$$

$$\underline{u} \leqslant u_j \leqslant \overline{u}, \quad j \in \mathbb{Z}_0^{N-1}, \tag{3.3e}$$

$$x_N = x_r. \tag{3.3f}$$

En cualquier formulación del MPC, N es el horizonte de predicción, es el intervalo temporal en el que se predice los siguientes estados y acciones del sistema. Por lo que al resolver un problema como (3.3a) se tiene que se han calculado $N-1$ acciones de control. De toda estas acciones solo se aplica la primera calculada y se vuelve a resolver el problema el siguiente instante de muestreo, con el nuevo valor del vector de estado.

La segunda formulación no contiene restricción terminal.

$$\min_{\mathbf{x},\mathbf{u}} \sum_{j=0}^{N-1} (\| x_j - x_r \|_Q^2 + \| u_j - u_r \|_R^2) + \| x_N - x_r \|_T^2, \tag{3.4a}$$

$$s.t.\ x_{j+1} = A_j + Bu_j, \quad j \in \mathbb{Z}_0^{N-1}, \tag{3.4b}$$

$$x_0 = x(t), \tag{3.4c}$$

$$\underline{x} \leqslant x_j \leqslant \overline{x}, \quad j \in \mathbb{Z}_1^{N-1}, \tag{3.4d}$$

$$\underline{u} \leqslant u_j \leqslant \overline{u}, \quad j \in \mathbb{Z}_0^{N-1}. \tag{3.4e}$$

Para poder garantizar convergencia y satisfacción de las restricciones, se necesita, en ambas formulaciones, que el par (x_r,u_r) sea un estado admisible del sistema y que x_0 esté dentro de la región admisible del sistema [6].

La ventaja que presenta la formulación con restricción terminal (3.3a) es que es más fácil de implementar y menos demandante que la que no la tiene.

La segunda formulación es algo más demandante y difícil de implementar pero a cambio el rango de control es más amplio.

3.2.1 Problema Cuadrático de la formulación MPC Regulación con restricción de igualdad

Esta formulación permite ser presentada como un problema cuadrático y por tanto resoluble por el algoritmo (3). El objetivo sería representar el problema de la siguiente forma:

$$J(\mathbf{z}) = \min_{\mathbf{z}\in Z} \{\frac{1}{2}\mathbf{z}^T H\mathbf{z} + q^T\mathbf{z}\}, \tag{3.5a}$$

$$s.t.\ G\mathbf{z} = b, \tag{3.5b}$$

$$Z : \{\mathbf{z} : \underline{\mathbf{z}} \leqslant \mathbf{z} \leqslant \overline{\mathbf{z}}\}. \tag{3.5c}$$

Hay que elegir la variable de decisión $\mathbf{z}$ primero y a partir de ella construir el resto. Tomando $\mathbf{z}$ como:

$$\mathbf{z} = \quad (u_0,x_1,u_1,x_2,\ldots,x_{N-1},u_{N-1}). \tag{3.6}$$

$\mathbf{z}$ no contiene x_0 ni x_N pues están fijados por las restricciones (3.3c) y (3.3f). A continuación se construye el resto de ingredientes necesarios para el desarrollo del problema cuadrático.

$$H = \begin{bmatrix} R & 0 & 0 & 0 & 0 & 0 \\ 0 & Q & 0 & 0 & 0 & 0 \\ 0 & 0 & R & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & Q & 0 \\ 0 & 0 & 0 & 0 & 0 & R \end{bmatrix}, \tag{3.7}$$

$$q = - \begin{bmatrix} Ru_r \\ Qx_r \\ Ru_r \\ Qx_r \\ \vdots \\ Qx_r \\ Ru_r \end{bmatrix}, \tag{3.8}$$

$$G = \begin{bmatrix} B & -I_n & 0 & \cdots & \cdots & 0 & 0 \\ 0 & A & B & -I_n & \cdots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \cdots & A & B & -I_n & 0 \\ 0 & 0 & \cdots & 0 & 0 & A & B \end{bmatrix}, \tag{3.9}$$

$$b = \begin{bmatrix} -Ax(t) \\ 0_n \\ \vdots \\ 0_n \\ x_r \end{bmatrix}, \tag{3.10}$$

$$Z = \{\mathbf{z} : \underline{\mathbf{z}} \leqslant \mathbf{z} \leqslant \overline{\mathbf{z}}\}. \tag{3.11}$$

Quizás la matriz G en primera instancia pueda parecer algo complicada de entender, pero no es más que una forma de expresar la restricción (3.3b). Del producto de G y z se obtiene b. Pues fijándonos bien, se ve como la primera fila de G sale la ecuación $Bu_0 - x_1$. Como bien sabemos $x_1 = Ax_0 + Bu_0$. Las demás filas de b salvo la última saldrán 0 ya que estaremos realizando $Ax_i + Bu_i - x_{i+1} = 0$. La última fila simplemente incluye la restricción terminal.

Falta por definir los vectores $\underline{\mathbf{z}}$ y $\overline{\mathbf{z}}$ y las dimensiones del problema.

$$\underline{\mathbf{z}} = (\underline{u},\underline{x},\underline{u},\underline{x},...,\underline{u},\underline{x},\underline{u}), \tag{3.12}$$

$$\overline{\mathbf{z}} = (\overline{u},\overline{x},\overline{u},\overline{x},...,\overline{u},\overline{x},\overline{u}). \tag{3.13}$$

Para las dimensiones del problema solo necesitamos fijarnos en la variable de decisión. Viendo que contiene N-1 veces x y N veces u filas inferimos que la dimensión n_z del vector de variables de decisión $\mathbf{z}$ es $n_z = (n+m)(N-1)+m$ y para las dimensiones de b se tiene que $m_z = Nn$.

3.2.2 Problema Cuadrático de la formulación MPC Regulación sin restricción terminal

La variable de decisión es idéntica a la anterior solo que ahora incluirá el término x_N

$$\mathbf{z} = (u_0,x_1,u_1,x_2,...,x_{N-1},u_{N-1},x_N). \tag{3.14}$$

Las matrices que definirán al problema serán:

$$H = \begin{bmatrix} R & 0 & 0 & 0 & 0 & 0 \\ 0 & Q & 0 & 0 & 0 & 0 \\ 0 & 0 & R & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & R & 0 \\ 0 & 0 & 0 & 0 & 0 & T \end{bmatrix}, \tag{3.15}$$

$$q = - \begin{bmatrix} Ru_r \\ Qx_r \\ Ru_r \\ Qx_r \\ \vdots \\ Ru_r \\ Qx_r \\ Rx_r \\ Tu_r \end{bmatrix}, \tag{3.16}$$

$$G = \begin{bmatrix} B & -I_n & 0 & \cdots & \cdots & 0 \\ 0 & A & B & -I_n & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & A & B & -I_n \end{bmatrix}, \tag{3.17}$$

$$b = \begin{bmatrix} -Ax(t) \\ 0_n \\ \vdots \\ 0_n \\ 0_n \end{bmatrix}, \tag{3.18}$$

$$Z = \{\mathbf{z} : \underline{\mathbf{z}} \leqslant \mathbf{z} \leqslant \overline{\mathbf{z}}\}. \tag{3.19}$$

Definimos por último los límites del problema y sus dimensiones:

$$\underline{\mathbf{z}} = (\underline{u},\underline{x},\underline{u},\underline{x},...,\underline{u},\underline{x}), \tag{3.20}$$

$$\overline{\mathbf{z}} = (\overline{u},\overline{x},\overline{u},\overline{x},...,\overline{u},\overline{x}). \tag{3.21}$$

De la misma forma que con la anterior formulación las dimensiones del problema vienen dada por la dimensión de la variable de decisión $\mathbf{z} \in \mathbb{R}^{n_z}$, $n_z = N(n+m)$ y las restricciones representadas en el vector $b \in \mathbb{R}^{m_z}$, $m_z = Nn$.

3.3 Implementación MPC Regulación

La implementación de ambas formulaciones es exactamente igual, solo habría que concentrarse en la construcción de las matrices de cada una. En esta sección trabajaremos exclusivamente con la formulación con restricción terminal. Se mostrarán 3 formas distintas de resolver el problema de optimización que plantea la formulación. Las pruebas realizadas al control solo cubren el problema de regulación, en este capitulo no se aborda el problema de seguimiento, que se abordará más adelante con la formulación del *MPCT*.

Se probará el control en los 2 siguientes sistemas:

$$x_{k+1} = \begin{bmatrix} 0.2769 & 0.09713 \\ 0.04617 & 0.82345 \end{bmatrix} x_k + 2 \cdot \begin{bmatrix} 0.5881 & 1.0617 \\ 0.4747 & 0.1830 \end{bmatrix} u_k, \tag{3.22a}$$

$$0 \leqslant x_j \leqslant 10,\ j \in \mathbb{Z}_0^N, \tag{3.22b}$$

$$-10 \leqslant u_j \leqslant 10,\ j \in \mathbb{Z}_0^N. \tag{3.22c}$$

$$x_{k+1} = \begin{bmatrix} 0.1056 & 0.4235 & 0.1537 \\ 0.6110 & 0.0908 & 0.2810 \\ 0.7788 & 0.2665 & 0.4401 \end{bmatrix} x_k + 5 \cdot \begin{bmatrix} 0.4849 & 0.7413 & 0.1500 \\ 0.3935 & 0.5201 & 0.5861 \\ 0.6714 & 0.3477 & 0.2621 \end{bmatrix} u_k, \tag{3.23a}$$

$$0 \leqslant x_j \leqslant 10,\ j \in \mathbb{Z}_0^N, \tag{3.23b}$$

$$-10 \leqslant u_j \leqslant 10,\ j \in \mathbb{Z}_0^N. \tag{3.23c}$$

El sistema (3.22) tiene dos entradas y un estado de dimensión dos. El segundo sistema, (3.23) tiene tres entradas y un estado de dimensión 3. A continuación se va a proceder a describir el algoritmo que simulará el bucle de control.

Algorithm 4 Bucle de control

1: Se elige el número de simulaciones: $Nsim$.
2: Se eligen u_0,x_0.
3: $k = 0$.
4: $x_{k+1} = Ax_k + Bu_k$.
5: $k = k + 1$.
6: $u_k \leftarrow MPC(x_k, u_{k-1})$.
7: Si $k < Nsim \rightarrow$ PASO 4.
8: Si $k > Nsim \rightarrow$ EXIT.

Este bucle es muy genérico. El paso 6 tiende a variar según el método de resolución que se tome para la formulación, pero la idea es la misma. Se calcula el estado del sistema, se calcula la $\mathbf{z}^*$ para conocer la acción óptima y se procede de nuevo a calcular el nuevo estado del sistema.

3.3.1 Resolución por quadprog (MATLAB)

MATLAB incluye una función que resuelve problemas cuadráticos muy sencilla de usar, quadprog. Se utiliza definiendo las variables de un problema cuadrático de la forma:

$$\min_x \frac{1}{2} x^T H x + f^T x \text{ tal que: } \begin{cases} A \cdot x \leqslant b \\ A_{eq} \cdot x = b_{eq} \\ lb \leqslant x \leqslant ub \end{cases}. \tag{3.24}$$

Devuelve un vector x que minimiza la función dada y que cumpla las condiciones impuestas. Relacionando con (3.5a) podemos definir el problema de optimización del MPC de la forma siguiente.

$$\begin{aligned} z &\rightarrow x, \\ H &\rightarrow H, \\ q &\rightarrow f, \\ G &\rightarrow A_{eq}, \\ \underline{z} &\rightarrow lb, \\ \bar{z} &\rightarrow ub. \end{aligned}$$

Además, quadprog también permite introducir un punto inicial al problema, que en nuestro caso será siempre la última solución hallada, salvo en el primer caso que será un vector que contenga $(u_1, x_1, \ldots)$ donde las últimas componentes corresponden a la referencia (u_r, x_r). Una forma de definir el problema sería llamar a una función en *Matlab* que devuelva una estructura con los datos del problema:

Código 3.1 Estructura del problema QP.

```
function Estructura = Estructura(H,q,G,b,zmenos,zmas,z)
% Estructura del problema
Estructura=struct;
```

```
Estructura.H=H;
Estructura.f=q;
Estructura.Aineq=[];
Estructura.bineq=[];
Estructura.Aeq=G;
Estructura.beq=b;
Estructura.lb=zmenos;
Estructura.ub=zmas;
Estructura.x0=z;
Estructura.solver='quadprog';
Estructura.options=optimoptions("quadprog","Algorithm","active-set");
end
```

La última opción es para elegir un algoritmo que tome en consideración el punto inicial que se aporte, quadprog de base no usa puntos iniciales para calcular una solución.

Es muy importante que antes de ejecutar quadprog se actualice en la estructura del problema la componente b con x_k calculada y en caso de cambiar de referencia habría que actualizar q también.

3.3.2 Resolución por *ADMM*

Al final del anterior capítulo se expuso el algoritmo *ADMM* (3) y también se desarrolló una función que implementaba este algoritmo. La idea de esta subsección sería indicar cómo llamar a esta función para que pueda resolver el problema.

Para llamar a la función habría que definir primero la tolerancia ε y ρ, luego x_0 se le da los valores del sistema en ese instante y se rellenan de unos los estados y acciones futuras para la primera ejecución del algoritmo, a partir de ese momento se usará el resultado obtenido. Para y_0 se le da el mismo valor que a x_0, esto no choca con el funcionamiento del algoritmo ya que pondremos una condición con la que se forzará siempre una primera ejecución. Los siguientes dos elementos a meter son los límites $\overline{x}$ y $\underline{x}$. Por último se introducen las matrices del problema.

Dentro de la propia función *ADMM* hay otra llamada a una función, ARGminADMM. Está no es más que una función que equivale a la ecuación (2.28).

3.3.3 Resolución por *ADMM* QP

En el capítulo anterior se desarrolló el método para una formulación genérica, en este capítulo se de desarrolla para formular un problema cuadrático.
Para definir el problema se tiene que $H \succeq 0$, $\underline{\mathbf{v}} \in \mathbb{R}^m$ y $\overline{\mathbf{v}} \in \mathbb{R}^m$

$$J^* = \min_{\mathbf{z}\in\mathbb{R}^n} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + q^T\mathbf{z}, \tag{3.25a}$$

$$s.t \quad G\mathbf{z} = b, \tag{3.25b}$$

$$\underline{\mathbf{v}} \leqslant C\mathbf{z} \leqslant \overline{\mathbf{v}}. \tag{3.25c}$$

$\mathbf{v}$ es una variable auxiliar que cumple $\mathbf{v} = C\mathbf{z} \in \mathbb{R}^m$

$$\begin{aligned} J^* = & \min_{\mathbf{z}\in\mathbb{R}^n} \frac{1}{2}\mathbf{z}^T H\mathbf{z} + q^T\mathbf{z} \\ s.t \quad & G\mathbf{z} = b, \\ & \underline{\mathbf{v}} \leqslant \mathbf{v} \leqslant \overline{\mathbf{v}}, \\ & C\mathbf{z} = \mathbf{v}. \end{aligned}$$

La formulación dual obtenida de la restricción $C\mathbf{z} = \mathbf{v}$ se puede resolver con *ADMM*. A continuación se procede a definir al problema cuadrático nuevo a resolver.

$$Z = \{\mathbf{z} \in \mathbb{R}^n \ : \ G\mathbf{z} = b\}, \ V = \{\mathbf{v} \in \mathbb{R}^m \ : \ \underline{\mathbf{v}} \leqslant \mathbf{v} \leqslant \overline{\mathbf{v}}\}, \tag{3.26a}$$

$$J^* = \min_{\mathbf{z}\in\mathbb{R}^n} \frac{1}{2}\,\mathbf{z}^T H\mathbf{z} + q^T\mathbf{z}, \tag{3.26b}$$

$$s.t.\ G\mathbf{z} = b, \tag{3.26c}$$

$$\underline{\mathbf{v}} \leqslant C\mathbf{z} \leqslant \overline{\mathbf{v}}. \tag{3.26d}$$

El algoritmo *ADMM* que resuelve el problema tendría esta forma:

Algorithm 5 *ADMM* QP

1: Se escoge la tolerancia ε.
2: Se escogen $\rho > 0$ y $v_0 \in V$.
3: $k = 0, \lambda_0 = 0$.
4: $\mathbf{z}_{k+1} = arg\ \min_{\mathbf{z}\in Z}\ \frac{1}{2}\,\mathbf{z}^T H\mathbf{z} + q^T\mathbf{z} + \lambda_k^T C\mathbf{z} + \frac{\rho}{2} \parallel C\mathbf{z} - \mathbf{v}_k \parallel_2^2$.
5: $\mathbf{v}_{k+1} = arg\ \min_{\mathbf{v}\in V} - \lambda_k^T\mathbf{v} + \frac{\rho}{2} \parallel C\mathbf{z}_{k+1} - \mathbf{v} \parallel_2^2$.
6: $\lambda_{k+1} = \lambda_k + \rho(C\mathbf{z}_{k+1} - \mathbf{v}_{k+1})$.
7: Si $\max\{\parallel C\mathbf{z}_{k+1} - \mathbf{v}_{k+1} \parallel_2 , \parallel \mathbf{v}_{k+1} - \mathbf{v}_k \parallel_2\} \leqslant \varepsilon \rightarrow$ EXIT .
8: $k = k + 1 \rightarrow$ PASO 4.

En este trabajo siempre vamos a tomar C como la matriz identidad $I_{n\times n}$ por lo que $\mathbf{z}$ y $\mathbf{v}$ tienen las mismas dimensiones.

El paso 4 trata de resolver un problema cuadrático restringido a una restricción de igualdad y a una caja respectivamente. Centrándonos en el problema con restricción de igualdad si se dan una serie de condiciones que presenta la formulación la resolución es simple [6].

Si se tiene que H es semidefinida positiva y de dimensiones $n_z \times n_z$, $q \in \mathbb{R}^{n_z}$, $G \in \mathbb{R}^{n_z \times n_z}$ y $b \in \mathbb{R}^{m_z}$. Se puede hallar la solución óptima $\mathbf{z}^*$ del problema de optimización si y solo si $\exists \mu \in \mathbb{R}^{m_z}$ que $\mathbf{z}^* = -H^{-1}(G^T\mu + q)$:

$$\begin{aligned} G\mathbf{z}^* &= b, \\ H\mathbf{z}^* + q + G^T\mu &= 0, \\ HG^{-1}b + q + G^T\mu &= 0, \\ G^T\mu &= -(HG^{-1}b + q), \\ \mu &= -(G^T)^{-1}\cdot(HG^{-1}b + q). \end{aligned}$$

Definimos una matriz $W = GH^{-1}G^T$.

$$\begin{aligned} W\mu &= -GH^{-1}G^T(G^T)^{-1}\cdot(HG^{-1}b + q), \\ W\mu &= -(b + GH^{-1}q), \\ \mathbf{z}^* &= -H^{-1}(G^T\mu + q). \end{aligned} \tag{3.27}$$

Crear una función que permita solucionar este tipo de problemas resulta entonces inmediata. Los dos códigos que se presentan ahora han sido extraídos de un trabajo previo [9].

Código 3.2 Solucionador de ecuaciones QP.

```
%-------------------------------------
%              ENTRADAS:
%-------------------------------------
%   - q: Vector de la función de coste del problema cuadrático.
%   - b: Vector de las restricciones de igualdad.
```

```
%   - Hinv: Inversa de una matriz Hessiana.
%   - G: Matriz que define más restricciones de igualdad.
%   - dW: descomposición de la matriz dada por W = G*Hinv*G'
%   -GHinv: Productp de G*H^{-1}
%   -HinvG: Productp de Hinv*G'
%-------------------------------------
%             SALIDAS:
%-------------------------------------
%   - z_opt: Solución al problema
%
function z_opt = solve_eqQP(q, b, GHinv, HinvG,dW,Hinv)
    h=-GHinv*q-b;
    mu=dW\h;
    z_opt = -HinvG*mu - Hinv*q;
end
```

En el caso de este primer código, se ha modificado para poder incluir el elemento dW que es una matriz que sale de descomponer en Matlab la matriz W, además de estar calculados fuera del bucle $G \cdot H^{-1}$ y $H^{-1} \cdot G'$.

El siguiente problema que nos queda por resolver es problema con restricción de caja, paso 5 del algoritmo *ADMM QP* (5). Si tenemos una matriz diagonal Hessiana como es el caso entonces la solución a un problema con restricciones de caja de la forma:

$$\min_{\mathbf{z}\in\mathbb{R}^{n_z}} \mathbf{z}^T H_\rho \mathbf{z} + q_k^T \mathbf{z},$$
$$s.t\ \underline{\mathbf{v}} \leqslant \mathbf{v} \leqslant \overline{\mathbf{v}}.$$

Se puede expresar la solución al problema de igualdad como $\mathbf{v}^*_{(j)} = max\{min\{-H^{-1}_{\rho(j,j)} q_{k(j)}, \overline{\mathbf{v}}_{(j)}\}, \underline{\mathbf{v}}_{(j)}\}$.Descrito en una función se pondría mediante un bucle que vaya desde 1 hasta n_z y recorra la diagonal H, q y los límites de la caja.

Código 3.3 Solver caja QP.

```
function z_opt = solve_boxQP(q, Hinv, z_lb, z_ub)

    % Consigue el número de varianles de decisión
    n_z = length(q);

    % Inicializar la solución óptima
    z_opt = zeros(n_z, 1);

    % Computa cada componente
    if length(Hinv) == 1
        for j = 1:n_z
            z_opt(j) = max( min( -Hinv*q(j), z_ub(j) ), z_lb(j) );
        end
    else
        for j = 1:n_z
            z_opt(j) = max( min( -Hinv(j,j)*q(j), z_ub(j) ), z_lb(j) );
        end
    end

end
```

Con estos dos códigos no serían suficiente para implementar el algoritmo del *ADMM QP* (5). Hay que definir completamente los problemas que resuelven. Para $\mathbf{z}$ tenemos:

$$J = \mathbf{z}^T H_\rho \mathbf{z} + q_k^T \mathbf{z} \tag{3.28a}$$
$$s.t. G\mathbf{z} = b, \tag{3.28b}$$
$$H_\rho = H + \rho I, \tag{3.28c}$$
$$q_k = q + \lambda_k - \rho \mathbf{v}_k. \tag{3.28d}$$

Definir así el problema nos permite poder llamar a la función [3.2] de forma directa. Antes de cada llamada se actualizaría el vector q_k que depende de los valores de λ_k y $\mathbf{v}_k$ anteriores. Para resolver $\mathbf{v}_{k+1}$ el problema resultante es mucho más sencillo:

$$J = \rho \mathbf{v}^T \mathbf{v} + \hat{q}_k^T \mathbf{v}, \tag{3.29a}$$
$$\underline{\mathbf{v}} \leqslant \mathbf{v} \leqslant \overline{\mathbf{v}}, \tag{3.29b}$$
$$\hat{q}_k^T = -\rho \mathbf{z}_{k+1} - \lambda_k. \tag{3.29c}$$

Con estos dos problemas definidos es muy fácil implementar ya el algoritmo (5). A continuación se expone una función en Matlab que lo implementa.

Código 3.4 ADMM QP.

```
% Entradas:
%   - q: Vector de la función de coste del problema cuadrático.
%   - b: Vector de las restricciones de igualdad.
%   - ro: Constante para regularizar el problema.
%   - epsilon: Tolerancia del problema
%   - G: Matriz que define más restricciones de igualdad.
%   - zmas y zmin:Restricciones de caja del problema
%   - z0: Condición inicial al problema (después de resolverlo por primera vez
    se le da siempre la solución en las siguientes ejecuciones)
%   - nz: Dimensión del problema QP
%   - Hroinv: Inversa la matriz H+\rho del problema QP.
%   - invro: 1/rho
%   - dW: descomposición de la matriz dada por W = G*Hinv*G'
% Salidas:
%   - z_opt: Solución del ADMM
%   - k: Número de iteraciones realizadas.

function [z_opt,k] = ADMM_QP(q,b,ro,epsilon,G,zmas,zmin,z0,nz,Hroinv,invro,dW)
% Condiciones inciales
k=0;
lambda=zeros(nz,1);
lambdakk=lambda;
zkk=z0;
vk=z0;
vkk=z0;
while (max(norm(zkk-vkk),norm(vkk-vk)) > epsilon || k==0)
   vk=vkk;
   lambda=lambdakk;
   qk=q+lambda-ro*vk;         %qk
   zkk=solve_eqQP(qk,b,Hroinv,G,dW); %zk+1
   q_=-ro*zkk-lambda;           %q^{^}
   vkk=solve_boxQP(q_,invro,zmin,zmas); % vk+1
   lambdakk=lambda+ro*(zkk-vkk); %lambdak+1
   k=k+1;
```

```
end
    z_opt=vkk;
end
```

3.4 Resultados

En esta sección se muestra una prueba del *MPCR* con un horizonte de predicción $N = 30$ el método que se use para la representación no es relevante por que todos llegan a la misma solución. A lo largo de la sección se mostrarán los tiempos de ejecución para cada algoritmo y la dependencia que pueden llegar a desarrollar con parámetros como ρ y ε.

3.4.1 Prueba control

Para finalizar este capítulo pondremos el MPC Regulación a prueba, concretamente la formulación (3.3a) vista anteriormente. Primero con en el sistema 2x2 (3.22) y luego en el sistema 3x3 (3.23). Se mostrará también el rendimiento de los algoritmos *ADMM* en función del parámetro ρ. Todas las pruebas de rendimiento se han hecho sobre el sistema 2x2. Finalmente se mostraran tanto gráficamente como en una tabla los tiempos de ejecución de las 3 implementaciones en función de N. En las 2 implementaciones de *ADMM* se tomará el ρ que más las optimice.

La pruebas de control se han realizado dándole al sistema una entrada estática durante varias iteraciones hasta activar el *MPC*. Una vez activo, se resolverá en cada iteración un problema de optimización para sacar la acción de control que se deberá aplicar para alcanzar la referencia. La referencia que se le da al sistema busca que se activen las restricciones de igualdad, de forma que se aprecie como el control evita que el sistema se salga de la caja a la que ha sido restringido.

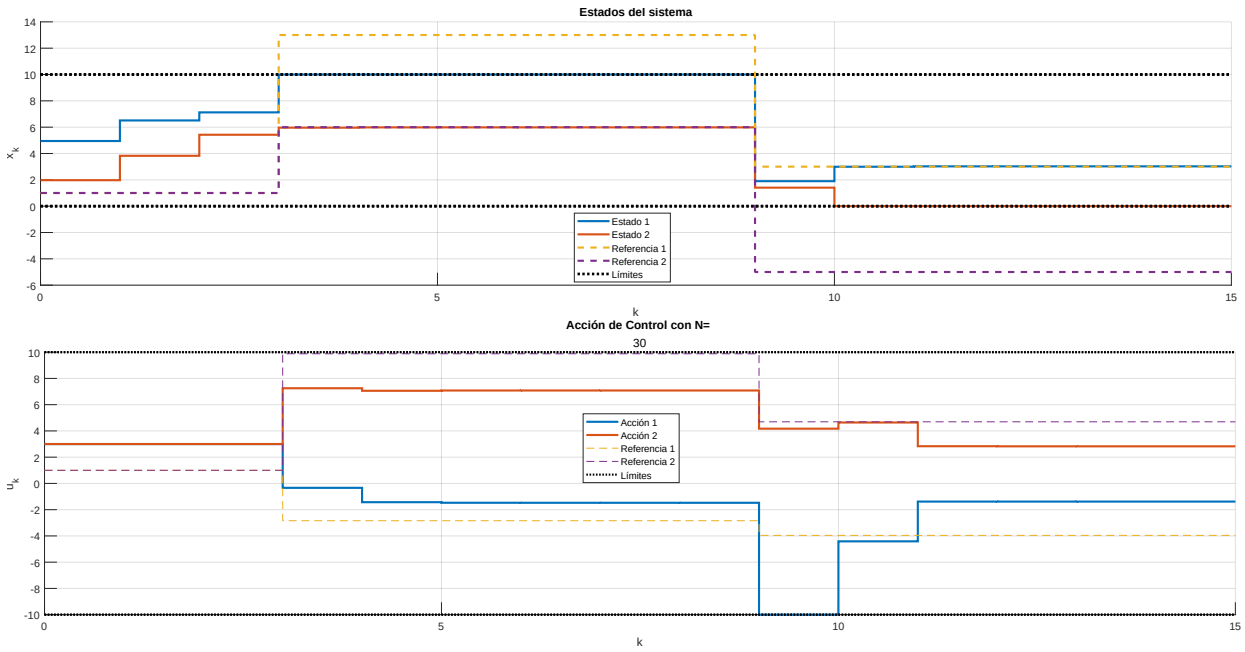

Figura 3.1 Simulación MPCR en un sistema 2x2.

Como es de esperar, cuando se le da al sistema una referencia que se sale de la caja, el control lleva el sistema al límite de la caja ya sea la acción de control o el estado, lo que primero alcance un límite.

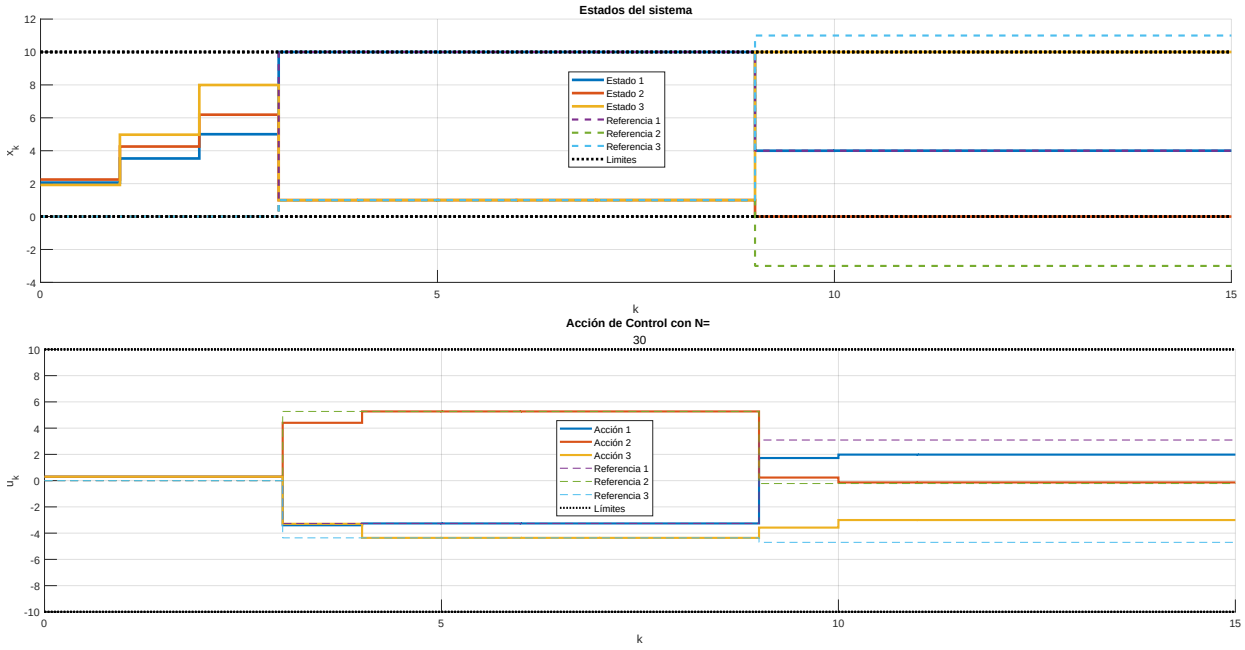

Figura 3.2 Simulación MPCR en un sistema 3x3.

3.4.2 Dependencia de ρ

ρ es un parámetro clave para agilizar la resolución del problema de optimización. Cuando se activan las restricciones de caja tener un parámetro ρ de un tamaño lejano de cero hará que sea más rápido, pero también ralentizará la resolución cuando no están activadas estas restricciones.

La resolución con quadprog no permite modificar este parámetro por lo que en este apartado se comparan el *ADMM* general y el *ADMM QP*. En los resultados se podrá observar que la formulación QP es más rápida,

esto se debe simplemente a que el código de esta se ha depurado mucho más para su posterior uso en el MPCT.

Se mostrará gráficamente el tiempo medio que tarda el algoritmo en resolver cada problema de optimización que se le plantea. Además, ya que el tiempo que tarda es una medida que depende mucho del equipo con el que se este simulando, también se mostrarán la iteración media dependiente de ρ. Además se darán 2 tablas tanto para las iteraciones como para el tiempo con sus valores máximos, mínimos, mediana y media dependientes de ρ. Notar que para las iteraciones solo hace falta una tabla ya que estamos comparando un *ADMM* que es un poco más rápido que el otro en cuanto a ejecución de iteraciones.

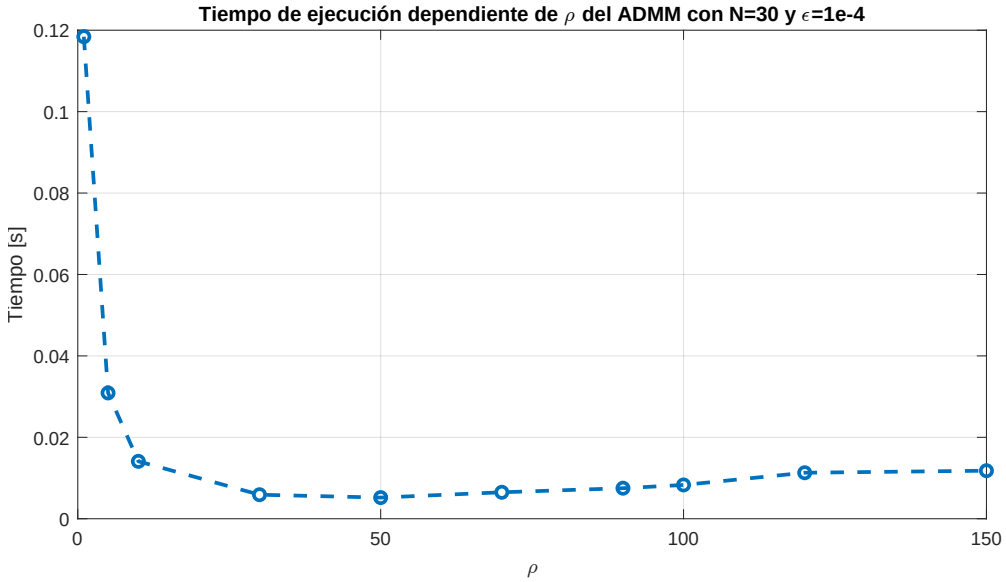

Figura 3.3 Dependencia del ADMM respecto a ρ.

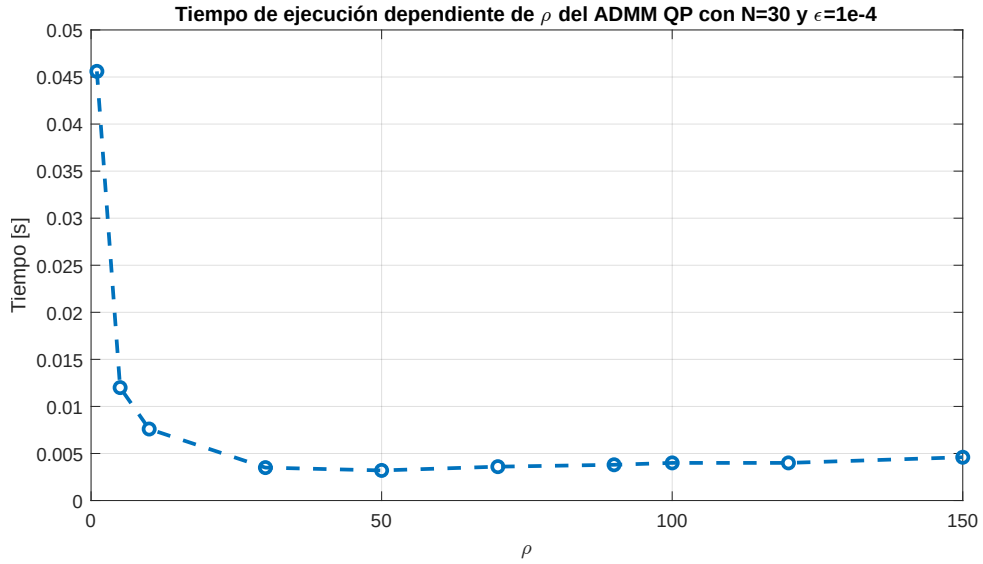

Figura 3.4 Dependencia del ADMM QP respecto a ρ.

La gráfica de las iteraciones coincide en forma con la del tiempo, lo que es coherente, ya que a más iteraciones más tiempo.

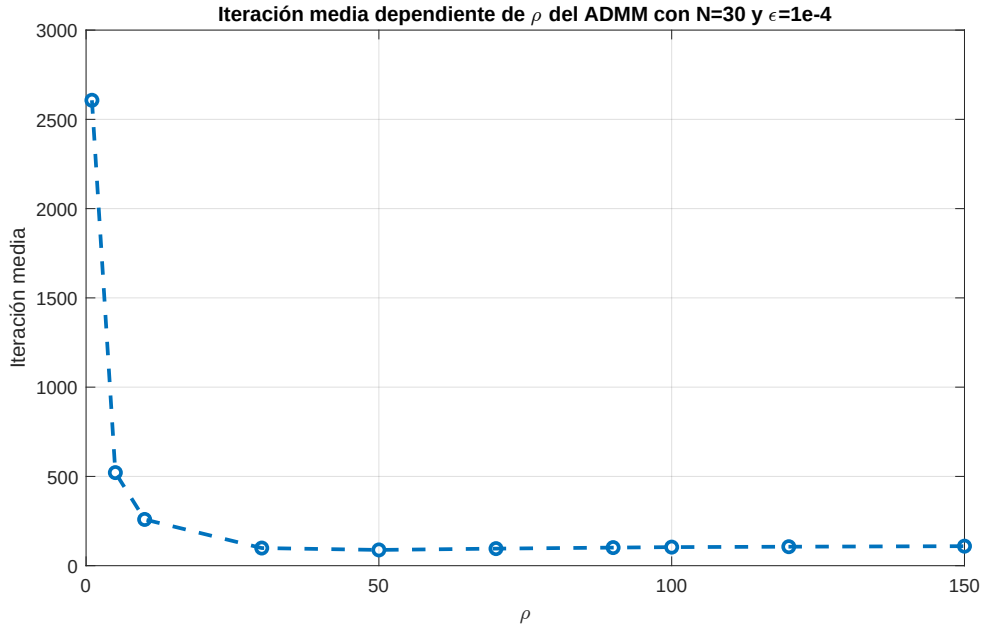

Figura 3.5 Número medio de iteraciones del ADMM respecto a ρ.

Tabla 3.1 Tabla de iteraciones dependientes de ρ en el *MPCR* del *ADMM*.

ρ	*ADMM*			
	Iteraciones			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	3542	1516	3542	2606,9
5	705	307	705	521,3077
10	348	155	348	259
30	138	54	137	98,7692
50	114	59	112	88,2308
70	121	65	117	95,5385
90	125	70	120	101,3077
100	127	71	122	104
120	128	72	121	106,3077
150	149	77	116	109,3077

Tabla 3.2 Tabla de tiempos dependientes de ρ en el *MPCR* del *ADMM* genérico.

ρ	*ADMM* Genérico			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	0,1625	0,0679	0,1443	0,1184
5	0,0486	0,0127	0,0279	0,0309
10	0,0232	0,007	0,0151	0,0141
30	0,0165	0,0027	0,0057	0,0059
50	0,0166	0,0026	0,0046	0,0052
70	0,0169	0,0033	0,0059	0,0065
90	0,0091	0,0054	0,0088	0,0075
100	0,0103	0,0065	0,0084	0,0083
120	0,0331	0,0051	0,0112	0,0113
150	0,035	0,0082	0,0097	0,0118

Tabla 3.3 Tabla de tiempos dependientes de ρ en el *MPCR* del *ADMM* QP.

ρ	*ADMM* QP			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	0,0577	0,0253	0,0567	0,0456
5	0,023	0,0064	0,0131	0,012
10	0,0195	0,0034	0,0066	0,0076
30	0,018	0,0011	0,0024	0,0035
50	0,0176	0,0011	0,0021	0,0032
70	0,0175	0,0014	0,0025	0,0036
90	0,0183	0,0014	0,0027	0,0038
100	0,0235	0,0015	0,0025	0,004
120	0,0255	0,0015	0,0023	0,004
150	0,0251	0,0021	0,0025	0,0046

3.4.3 Dependencia de ε

ε es un parámetro que afecta directamente al número de iteraciones que realizará el *ADMM*, no influye en lo rápido que se llega a lo solución como tal sino más bien podría decirse que acerca más la solución al punto inicial del problema. Esto último puede ser problemático ya que ante un ε demasiado alto el algoritmo queda inutilizado ya que este solo se ejecuta una vez dado que la tolerancia al ser muy grande, considera que se cumplen las condiciones para finalizar el bucle. Esto puede ocasionar que el control no respete las restricciones. Para ilustrarlo se muestra la misma prueba que se mostró en (3.1) con $\varepsilon = 100$.

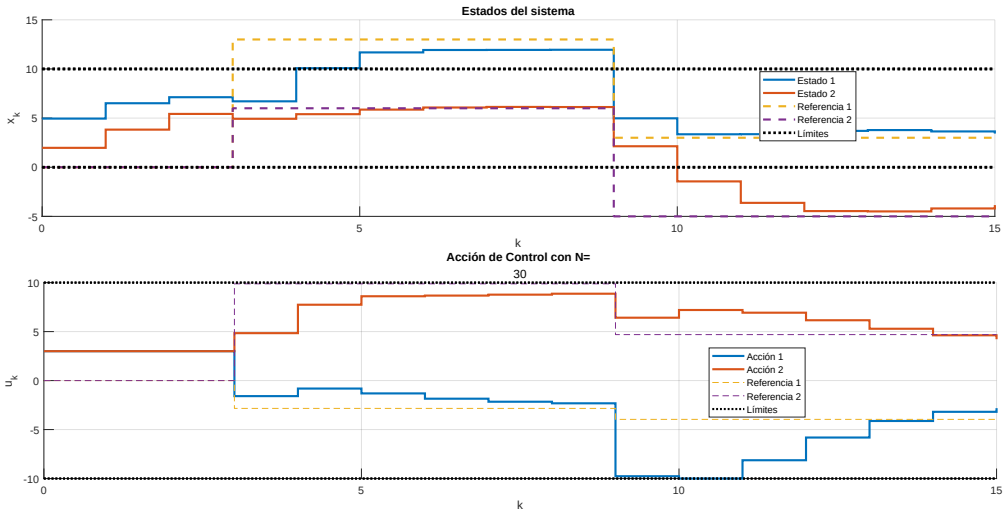

Figura 3.6 MPCR 2x2 con $\varepsilon = 100$.

Como se podrá deducir, mientras que el parámetro ρ afecta a la rapidez del algoritmo, el parámetro ε puede afectar también a la velocidad de ejecución, pero también puede desembocar en que no se resuelva el problema de forma satisfactoria.

Las pruebas que se realizarán variando este parámetro, se harán todas dentro de un rango en el que la solución sea coherente con las restricciones, para evitar una situación como la de la figura anterior (3.6).

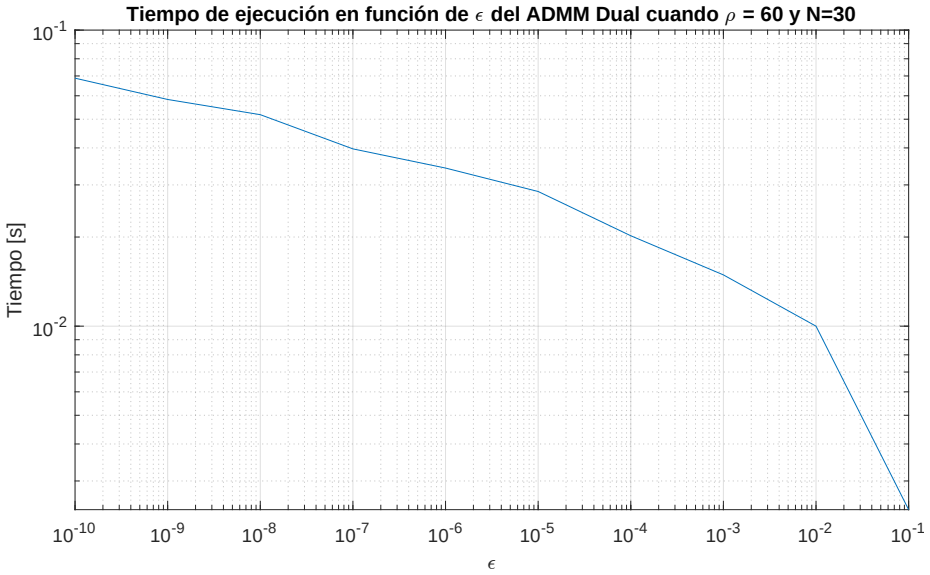

Figura 3.7 Dependencia del ADMM respecto a ε.

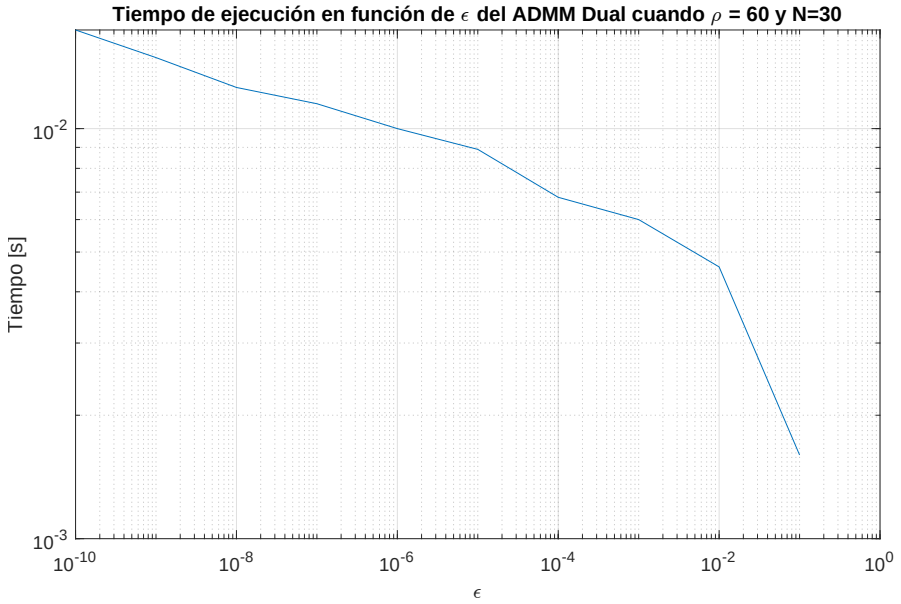

Figura 3.8 Dependencia del ADMM QP respecto a ε.

Como cabría esperar, cuando más pequeña es ε mayor es el tiempo medio que tarda el método en resolver el problema. En la siguiente sección se estudiará ya la dependencia de los 3 métodos expuestos en este capítulo respecto al parámetro N. Se van a dejar fijos un valor de $\rho = 50$ y una valor de $\varepsilon = 1e-4$.

3.4.4 Dependencia de N

En esta sección se podrá apreciar como la dependencia del *ADMM* con N puede llegar a ser lineal cuando el algoritmo se optimiza lo suficiente. Se mostrará una gráfica donde se pondrá comparar como afecta N a cada algoritmo en su tiempo medio de resolución y también se darán las tablas de rendimientos de cada uno en función de N.

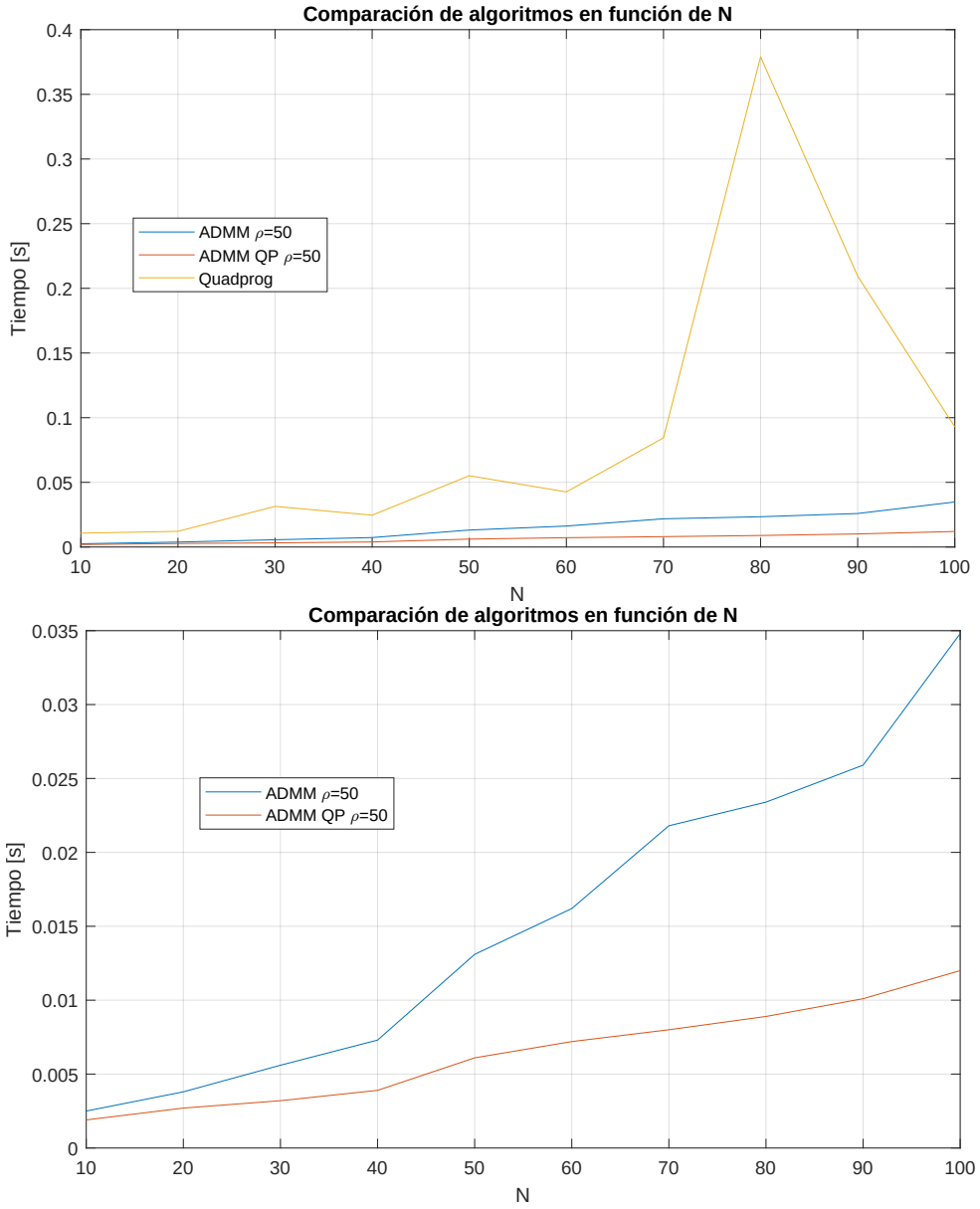

Figura 3.9 Comparación de los algoritmos.

El *ADMM QP* es más rápido que el genérico porque está más pulido. Lo que se ha hecho es tomar cualquier cálculo que, dentro del bucle, dé siempre el mismo resultado, sacarlo fuera y calcularlo fuera del bucle. De esta forma, se obtiene una dependencia casi lineal respecto a N.

Tabla 3.4 Tabla dependencia N en el *MPCR* del QuadProg.

ρ	Quadprog			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
10	0,0531	0,0015	0,0023	0,0107
20	0,0543	0,0025	0,004	0,0121
30	0,1268	0,0041	0,0103	0,0314
40	0,0873	0,008	0,0111	0,0246
50	0,368	0,01	0,0193	0,0551
60	0,156	0,0162	0,0244	0,0426
70	0,5672	0,0215	0,032	0,0844
80	1,7223	0,0331	0,0503	0,379
90	0,8944	0,0376	0,0613	0,2094
100	0,2021	0,013	0,0762	0,0929

Tabla 3.5 Tabla dependencia N en el *MPCR* del *ADMM*.

ρ	*ADMM* con $\rho = 50$ y $\varepsilon = 1e-4$			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
10	0,0138	0,000937	0,0014	0,0025
20	0,018	0,0017	0,0028	0,0038
30	0,0215	0,0022	0,0043	0,0056
40	0,0213	0,0047	0,0058	0,0073
50	0,0246	0,0076	0,0143	0,0131
60	0,0297	0,096	0,018	0,0162
70	0,0333	0,0135	0,0245	0,0218
80	0,0342	0,0147	0,027	0,0234
90	0,0389	0,0191	0,0231	0,0259
100	0,0491	0,0248	0,0343	0,0348

Tabla 3.6 Tabla dependencia N en el *MPCR* del *ADMM* QP.

ρ	*ADMM QP* con $\rho = 50$ y $\varepsilon = 1e-4$			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
10	0,0151	0,000439	0,00073	0,0019
20	0,0174	0,000795	0,0015	0,0027
30	0,0171	0,0012	0,0022	0,0032
40	0,0173	0,0017	0,0029	0,0039
50	0,0219	0,003	0,0054	0,0061
60	0,0199	0,0033	0,0062	0,0072
70	0,0249	0,0041	0,0075	0,008
80	0,0244	0,0047	0,0087	0,0089
90	0,0252	0,0053	0,0101	0,0101
100	0,026	0,0068	0,013	0,012

Hemos explorado el *MPCR* y optimizado un algoritmo *ADMM* para su resolución. En el siguiente capítulo se profundizará en una formulación más compleja y en otro algoritmo para su resolución, *EADMM*.

4 MPC para Seguimiento

La pluma es más poderosa que la espada, si la espada está envainada y la pluma muy afilada

Terry Pratchett

Una vez ya se ha introducido al MPC de Regulación, se puede pasar a una formulación algo más compleja, el MPC for Tracking.

4.1 Formulación MPC para Seguimiento

Esta formulación tiene notables diferencias respecto a la anteriormente vista, pero lo más llamativo es la inclusión de nuevas variables de decisión que llamaremos referencias artificiales:

$$(x_s,u_s) \in \mathbb{R}^n \times \mathbb{R}^m. \tag{4.1}$$

La formulación que se va a desarrollar tiene restricciones terminales.

$$\min_{\mathbf{x},\mathbf{u},x_s,u_s} \sum_{j=0}^{N-1} (\| x_j - x_s \|_Q^2 + \| u_j - u_s \|_R^2) + \| x_s - x_r \|_T^2 + \| u_s - u_r \|_S^2 \tag{4.2a}$$

$$s.t.\ x_0 = x(t), \tag{4.2b}$$

$$x_{j+1} = Ax_j + Bu_j,\ j \in \mathbb{Z}_0^{N-1}, \tag{4.2c}$$

$$\underline{x} \leqslant x_j \leqslant \overline{x},\ j \in \mathbb{Z}_1^{N-1}, \tag{4.2d}$$

$$\underline{u} \leqslant u_j \leqslant \overline{u},\ j \in \mathbb{Z}_0^{N-1}, \tag{4.2e}$$

$$x_s = Ax_s + Bu_s, \tag{4.2f}$$

$$\underline{x} + \varepsilon_x \leqslant x_s \leqslant \overline{x} - \varepsilon_x, \tag{4.2g}$$

$$\underline{u} + \varepsilon_u \leqslant u_s \leqslant \overline{u} - \varepsilon_u, \tag{4.2h}$$

$$\varepsilon_u \in \mathbb{R}^m,\ \varepsilon_x \in \mathbb{R}^n. \tag{4.2i}$$

La función de coste penaliza la discrepancia entre la acción de control y los estados del sistema con el par (x_s,u_s), la referencia artificial. También penaliza las discrepancias que haya entre la referencia artificial y la referencia real.

La inclusión de una referencia artificial conlleva una serie de ventajas respecto a la formulación de regulación. Una de las más considerables es que requiere un horizonte de predicción más pequeño para conseguir un dominio de control aceptable. También permite que dada una referencia no alcanzable se llegue a un estado admisible que minimiza la función de coste [8] [6].

Esta formulación permite ser resuelta por el *EADMM* aunque como veremos más adelante no es muy eficiente en cuanto a tiempo de ejecución. En este capítulo se explorarán 2 métodos para su resolución,

ADMM que ya hemos visto con anterioridad y *EADMM* [8]. Respecto al primer método, se aprovechará la estructura en semi banda que tiene el problema para optimizar el algoritmo.

4.1.1 Problema cuadrático de la formulación MPC para Seguimiento

En el anterior capítulo (3) ya se mostró el tipo de problema que resuelve el algoritmo *ADMM*.

$$\begin{aligned} J^* = \quad & \min_{\mathbf{z}\in\mathbb{R}^n} \frac{1}{2}\mathbf{z}^t H\mathbf{z} + q^t\mathbf{z} \\ & s.t \quad G\mathbf{z} = b, \\ & \quad \underline{\mathbf{v}} \leqslant C\mathbf{z} \leqslant \overline{\mathbf{v}}. \end{aligned}$$

Considerando un vector de optimización de variables $\mathbf{z} = (x_0,u_0,x_1,u_1,...,x_{N-1},u_{N-1},x_s,u_s)$ y un vector de variables auxiliares $\mathbf{v} = (\tilde{x}_0,\tilde{u}_0,\tilde{x}_1,\tilde{u}_1,...,\tilde{x}_{N-1},\tilde{u}_{N-1},\tilde{x}_s,\tilde{u}_s)$ se puede transformar el problema del MPCT en un problema cuadrático.

Volvemos a suponer de nuevo que $C = I_{nxn} \rightarrow \mathbf{z} = \mathbf{v}$. Hay que que definir H, G, b y q para poder resolver el problema por *ADMM*.

$$H = 2\cdot\begin{bmatrix} Q & 0 & \cdots & 0 & 0 & -Q & 0 \\ 0 & R & \cdots & 0 & 0 & 0 & -R \\ 0 & 0 & \ddots & 0 & 0 & \vdots & \vdots \\ \vdots & \vdots & \ddots & Q & 0 & -Q & 0 \\ 0 & 0 & \cdots & 0 & R & 0 & -R \\ -Q & 0 & \cdots & -Q & 0 & NQ+T & 0 \\ 0 & -R & \cdots & 0 & -R & 0 & NR+S \end{bmatrix}, \tag{4.3a}$$

$$q = -2\cdot\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ Tx_r \\ Su_r \end{bmatrix}, \tag{4.3b}$$

$$G = \begin{bmatrix} I & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ A & B & -I & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & A & B & -I & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & A & B & -I & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & (A-I) & B \end{bmatrix}, \tag{4.3c}$$

$$b = \begin{bmatrix} x_0 \\ 0_n \\ \vdots \\ 0_n \\ 0_n \end{bmatrix}, \tag{4.3d}$$

$$\underline{\mathbf{v}} = (\underline{x},\underline{u},\ldots,\underline{x},\underline{u},\underline{x}+\varepsilon_x,\underline{u}+\varepsilon_u), \tag{4.3e}$$

$$\overline{\mathbf{v}} = (\overline{x},\overline{u},\ldots,\overline{x},\overline{u},\overline{x}+\varepsilon_x,\overline{u}+\varepsilon_u). \tag{4.3f}$$

Las dimensiones del problemas salen de forma instantánea. Mirando el vector z podemos ver por sus componente que su tamaño será $n_z = (N+1)(n+m)$ ya que el vector tiene $N+1$ componentes de n y m filas. Por lo tanto $q \in \mathbb{R}^{n_z}, H \in \mathbb{R}^{n_z x n_z}, G \in \mathbb{R}^{(N+2)nxn_z}, b \in \mathbb{R}^{(N+2)n}$ y $\underline{v},\overline{v} \in \mathbb{R}^{n_z}$.

Para las dimensiones de b tenemos que en las restricciones de igualdad hay que sumarle a N 2 más de las referencias artificiales, y entonces las dimensiones de G saldrían de forma natural para que sea compatible

con $\mathbf{z}$ y b.

Con la construcción de las matrices y vectores descritos arriba se puede resolver el problema usando *ADMM*. El algotitmo que se implementa se puede optimizar haciendo uso de la estructura del problema, como se verá más tarde.

4.1.2 EADMM para MPCT

Una de las formas más naturales con las que se podría abordar este problema sería mediante el uso del *EADMM* (Extended-ADMM). Extender el *ADMM* consiste en pasar de tener dos variables primales a tres o más [8]. De esta forma llegamos una complejidad computacional parecida a la que tienen los métodos de *ADMM* del MPCR. Para atacar el MPCT con *EADMM* es necesario reformular el problema de otra forma haciendo uso de de las variables auxiliares $\tilde{x}_k = x_k - x_s$ y $\tilde{u}_k = u_k - u_s$:

$$\min_{\tilde{\mathbf{x}},\tilde{\mathbf{u}},\mathbf{x},\mathbf{u},x_s,u_s} \sum_{k=0}^{N} (\| \tilde{x}_k \|_Q^2 + \| \tilde{u}_k \|_R^2) + \| x_s - x_r \|_T^2 + \| u_s - u_r \|_S^2 \tag{4.4a}$$

$$s.t.\ x_0 = x(t), \tag{4.4b}$$

$$\tilde{x}_{k+1} = A\tilde{x}_k + B\tilde{u}_k,\ k \in \mathbb{Z}_0^{N-1}, \tag{4.4c}$$

$$\underline{x} \leqslant x_k \leqslant \overline{x},\ k \in \mathbb{Z}_1^{N-1}, \tag{4.4d}$$

$$\underline{u} \leqslant u_k \leqslant \overline{u},\ k \in \mathbb{Z}_0^{N-1}, \tag{4.4e}$$

$$\underline{x} + \varepsilon_x \leqslant x_N \leqslant \overline{x} - \varepsilon_x, \tag{4.4f}$$

$$\underline{u} + \varepsilon_u \leqslant u_N \leqslant \overline{u} - \varepsilon_u, \tag{4.4g}$$

$$x_s = Ax_s + Bu_s, \tag{4.4h}$$

$$\tilde{x}_k + x_s - x_k = 0_n,\ k \in \mathbb{Z}_0^N, \tag{4.4i}$$

$$\tilde{u}_k + u_s - u_k = 0_m,\ k \in \mathbb{Z}_0^N, \tag{4.4j}$$

$$x_N = x_s, \tag{4.4k}$$

$$u_N = u_s. \tag{4.4l}$$

En esta formulación tenemos 2 nuevas variables de decisión respecto a la original, $(\tilde{x},\tilde{u})$. Las restricciones (4.4i) y (4.4j) imponen que haya coherencia con las variables de decisión del problema (4.2). Las desigualdades (4.2g) y (4.2g) se omiten directamente ya que están expresadas por las desigualdades (4.4f) y (4.4g) junto con las restricciones de igualdad (4.4k) y (4.4l).
Usando EADMM realmente estamos resolviendo tres problemas de optimización cuyas variables de optimización son:

$$\mathbf{z}_1 = (x_0,u_0,x_1,u_1,...,x_{N-1},u_{N-1},x_N,u_N), \tag{4.5a}$$

$$\mathbf{z}_2 = (x_s,u_s), \tag{4.5b}$$

$$\mathbf{z}_3 = (\tilde{x}_0,\tilde{u}_0,\tilde{x}_1,\tilde{u}_1,...,\tilde{x}_{N-1},\tilde{u}_{N-1},\tilde{x}_N,\tilde{u}_N). \tag{4.5c}$$

Para emplear este métodos es necesario construir 4 matrices que lleven las restricciones del sistema con el fin de que una vez hallada la solución se cumpla:

$$\sum_{i=1}^{3} C_i \mathbf{z}_i^* - b = 0. \tag{4.6}$$

Cada C_i son matrices que llevan ligadas las restricciones de sus respectivos problemas:

$$C_1 = \begin{bmatrix} [I_n 0_{n\times n}] & 0 & 0 \\ \hline -I_{n+m} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & -I_{n+m} \\ \hline 0 & 0 & -I_{n+m} \end{bmatrix}, \qquad C_2 = \begin{bmatrix} 0 \\ \hline \\ I_{n+m} \\ \vdots \\ I_{n+m} \\ \hline \\ I_{n+m} \end{bmatrix},$$

$$C_3 = \begin{bmatrix} 0 & \dots & 0 \\ \hline \\ I_{n+m} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & I_{n+m} \\ \hline \\ 0 & \dots & 0 \end{bmatrix}, \qquad b = \begin{bmatrix} x_0 \\ \hline \\ 0 \\ \vdots \\ 0 \\ \hline \\ 0 \end{bmatrix}.$$

Las matrices C_1, C_2 y C_3 contienen las restricciones de igualdad (4.4b), (4.4i), (4.4j), (4.4k) y (4.4l). Las líneas de las matrices separan las restricciones por grupos. Se puede apreciar que las primeras n filas contienen la restricción (4.4b), y que las $n+m$ últimas filas imponen las restricciones (4.4k) y (4.4l). Lo que queda en medio impondría el resto de restricciones.

Con estas matrices definidas podemos ya definir los 3 problemas que resolveremos con el siguiente algoritmo:

Algorithm 6 EADMM

Require: $\mathbf{z}_2, \mathbf{z}_3, \lambda^0, \rho > 0, \varepsilon > 0$

1: Se actualiza b con x_0
2: Se actualiza q_2 con x_r y u_r
3: Se inicializa k, $k \leftarrow 0$
4: $q_1 \leftarrow \rho C_1^T C_2 \mathbf{z}_2^k + \rho C_1^T C_3 \mathbf{z}_3^k + C_1^T \lambda^k - \rho C_1^T b$
5: $\mathbf{z}_1^{k+1} \leftarrow$ solve_boxQP $(q_1, H_1^{-1}, \underline{\mathbf{z}}_1, \overline{\mathbf{z}}_1)$
6: $q_2 \leftarrow -\begin{bmatrix} Tx_r \\ Su_r \end{bmatrix} + \rho C_2^T C_1 \mathbf{z}_1^{k+1} + \rho C_2^T C_3 \mathbf{z}_3^k + C_2^T \lambda^k - \rho C_2^T b$
7: $\mathbf{z}_2^{k+1} \leftarrow M_2 q_2$
8: $q_3 \leftarrow \rho C_3^T C_1 \mathbf{z}_1^{k+1} + \rho C_3^T C_2 \mathbf{z}_2^{k+1} + C_3^T \lambda^{k+1} - \rho C_3^T b$
9: $\mathbf{z}_3^{k+1} \leftarrow$ solve_eqQP$(q_3, b_3, H_3^{-1}, G_3)$
10: $\Gamma \leftarrow \sum_{i=1}^{3} C_i \mathbf{z}_i^{k+1} - b$
11: $\lambda^{k+1} \leftarrow \lambda^k + \rho\Gamma$
12: Si $max(\| \Gamma \|_\infty, \| \mathbf{z}_2^k - \mathbf{z}_2^{k-1} \|_\infty, \| \mathbf{z}_3^k - \mathbf{z}_3^{k-1} \|_\infty) \leqslant \varepsilon \rightarrow$ EXIT
13: $k \leftarrow k+1 \rightarrow$ PASO 4

El primer problema que se resuelve en los pasos 3 y 4 queda definido de la siguiente forma:

$$\min_{\mathbf{z}_1} \frac{1}{2} \mathbf{z}_1^T H_1 \mathbf{z}_1 + q_1^T \mathbf{z}_1, \tag{4.7a}$$

$$s.t.\ \underline{\mathbf{z}}_1 \leqslant \mathbf{z}_1 \leqslant \overline{\mathbf{z}}_1. \tag{4.7b}$$

Definimos los componentes del problema:

$$
\begin{aligned}
H_1 &= \rho C_1^T C_1, \\
q_1 &= \rho C_1^T C_2 \mathbf{z}_2^k + \rho C_1^T C_3 \mathbf{z}_3^k + C_1^T \lambda^k - \rho C_1^T b, \\
\underline{\mathbf{z}}_1 &= (-M_n, \underline{u}, \underline{x}, \ldots, \underline{u}, \underline{x} + \varepsilon_x, \underline{u} + \varepsilon_u), \\
\overline{\mathbf{z}}_1 &= (M_n, \overline{u}, \overline{x}, \ldots, \overline{u}, \overline{x} - \varepsilon_x, \overline{u} - \varepsilon_u).
\end{aligned}
$$

El vector $M_n \in \mathbb{R}^n$ y todas sus componentes son positivas de valor arbitrariamente alto. Por como es C_1, H_1 es una matriz diagonal de dimensiones $(N+1)(n+m)$. Este problema puede ser resuelto por el código (3.3) ya que es un problema con restricción de caja.

Para $\mathbf{z}_2$ lo que tenemos es un problema con una restricción de igualdad (pasos 6 y 7).

$$
\min_{\mathbf{z}_2} \frac{1}{2} \mathbf{z}_2^T H_2 \mathbf{z}_2 + q_2^T \mathbf{z}_2, \tag{4.8a}
$$

$$
s.t.\ G_2 \mathbf{z}_2 = b_2. \tag{4.8b}
$$

Se definen los ingredientes para montar el problema:

$$
\begin{aligned}
H_2 &= \begin{bmatrix} T & 0 \\ 0 & S \end{bmatrix} + \rho C_2^T C_2, \\
q_2 &= -\begin{bmatrix} T x_r \\ S u_r \end{bmatrix} + \rho C_2^T C_1 \mathbf{z}_1^{k+1} + \rho C_2^T C_3 \mathbf{z}_3^k + C_2^T \lambda^k - \rho C_2^T b, \\
G_2 &= \begin{bmatrix} (A - I_n) & B \end{bmatrix}, \\
b_2 &= 0_n.
\end{aligned}
$$

Se podría resolver usando el código (3.2), pero también se puede obtener teniendo en cuenta las estructuras de las matrices se puede simplificar en la siguiente expresión [6]:

$$
\mathbf{z}_2^* = M_2 q_2
$$

Siguiendo los pasos que nos lleva a la ecuación (3.27) llegar a M_2 es bastante simple

$$
\begin{aligned}
&\mathbf{z}_2 = -H_2^{-1}(G_2^T \mu + q_2), \\
&W_2 \mu_2 = -(b_2 + G_2 H_2^{-1} q_2), \\
&W_2 = G_2 H_2^{-1} G_2^T, \\
&\mu_2 = -(G_2 H_2^{-1} G_2^T)^{-1}(b_2 + G_2 H_2^{-1} + q_2), \\
&\mathbf{z}_2 = -H_2^{-1}(G_2^T(-(G_2 H_2^{-1} G_2^T)^{-1}(b_2 + G_2 H_2^{-1} q_2)) + q_2). \\
&\text{Como } b_2 \text{ está llena de ceros podemos simplificar y llegar a:} \\
&\mathbf{z}_2 = -H_2^{-1}(G_2^T(-(G_2 H_2^{-1} G_2^T)^{-1}(G_2 H_2^{-1} q_2)) + q_2) \\
&\text{Sacamos la } q_2 \text{ y obtendremos } M_2\text{:} \\
&\mathbf{z}_2 = (H_2^{-1} G_2^T (G_2 H_2^{-1} G_2^T)^{-1} G_2 H_2^{-1} - H_2^{-1}) q_2 \\
&M_2 = H_2^{-1} G_2^T (G_2 H_2^{-1} G_2^T)^{-1} G_2 H_2^{-1} - H_2^{-1}
\end{aligned}
$$

Por último resta el definir el último problema (pasos 8 y 9) que también lleva consigo otra restricción de igualdad que resolveremos directamente usando el código anteriormente descrito para este resolver este tipo de restricción (3.2):

$$
\min_{\mathbf{z}_3} \frac{1}{2} \mathbf{z}_3^T H_3 \mathbf{z}_3 + q_3^T \mathbf{z}_3, \tag{4.9a}
$$

$$
s.t.\ G_3 \mathbf{z}_3 = b_3. \tag{4.9b}
$$

Definimos a continuación los componentes del problema:

$$H_3 = \begin{bmatrix} Q & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & R & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & Q & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & R & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & Q & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & R \end{bmatrix} + \rho C_3^T C_3,$$

$$q_3 = \rho C_3^T C_1 \mathbf{z}_1^{k+1} + \rho C_3^T C_2 \mathbf{z}_2^{k+1} + C_3^T \lambda^{k+1} - \rho C_3^T b,$$

$$G_3 = \begin{bmatrix} A & B & -I_n & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & A & B & -I_n & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \cdots & 0 & A & B & -I_n & 0 \end{bmatrix},$$

$$b = 0_{Nn}.$$

Para finalizar solo quedaría mostrar el código que implementa este algoritmo.

Código 4.1 EADMM.

```
%--------------------------------------
%              ENTRADAS:
%--------------------------------------
%   -z2_0: Valor inicial al problema de z2.
%   -z3_0: Valor inicial al problema de z3.
%   -lambda_0: Valor inicial para lambda
%   -C1,C2,C3: Matrices de los problemas de z1,z2 y z3.
%   -ro: Constante de regulación.
%   -b: Vector con restricciones de igualdad para C1,C2,C3
%   -b3: Vector con restricciones de igualdad del problema de z3.
%   -G3: Matriz con restricciones de igualdad del problema de z3.
%   -zmenos1 y zmas1: Caja del problema de z1.
%   -T,S: Matrices de "penalización" de la formulación.
%   -xr,ur: par de referencia del problema.
%   -epsilon: tolerancia del problema.
%   -invH1,invH3: inversas de las matrices Hessianas H1 y H3.
%   -M2: Matriz que permite calcular directamente z2_opt.
%   -W_U y W_L: descomposición de la matriz W3 por LU.
%   -GHinv3: Producto de G_3*H_3^{-1}.
%   -HinvG3: Producto de Hinv_3*G_3'.
%--------------------------------------
%              SALIDAS:
%--------------------------------------
%   - z1_opt,z2_opt,z3_opt: Soluciones al problema (z1_opt es donde está la
%   acción de control).
%   - lambda: Lambda final del problema
%   - k: Número de iteraciones realizadas.
function [z1_opt,z2_opt,z3_opt,lambda_opt,k] = EADMM(z2_0,z3_0,lambda_0,C1,C2,
    C3,ro,b,b3,zmenos1,zmas1,T,S,xr,ur,epsilon,invH1,Hinv3,M2,W_U,W_L,GHinv3,
    HinvG3)
k=0;
z2=z2_0;
z3=z3_0;
z2_ant=z2;
z3_ant=z3;
```

```
lambda=lambda_0;
euler=0;
% Bucle del algoritmo
while (max(max(norm(euler,"inf"),norm(z2-z2_ant,"inf")),norm(z3-z3_ant,"inf"))
    >= epsilon || k==0)
    z2_ant=z2;
    z3_ant=z3;
    q1=ro*C1'*C2*z2+ ro*C1'*C3*z3 +C1'*lambda-ro*C1'*b;
    z1=solve_boxQP(q1,invH1,zmenos1,zmas1);
    q2=-[T*xr;S*ur] + ro*C2'*C1*z1+ro*C2'*C3*z3+C2'*lambda-ro*C2'*b;
    z2=M2*q2;
    q3=ro*C3'*C1*z1+ro*C3'*C2*z2+C3'*lambda-ro*C3'*b;
    z3=solve_eqQP(q3,b3,GHinv3,HinvG3,W_U,W_L,Hinv3);
    euler=(C1*z1 + C2*z2 + C3*z3)-b;
    lambda=lambda+ro*euler;
    k=k+1;
end
z1_opt=z1;
z2_opt=z2;
z3_opt=z3;
lambda_opt=lambda;
end
```

4.2 Ecuaciones en Semi-Banda

En esta sección se ahondará más en la formulación resuelta por *ADMM* y como aprovechar su estructura para conseguir un desempeño óptimo. Los resultados teóricos en los que se basará vienen de otros trabajos como por ejemplo [5].

El problema que tenemos es que, en el algoritmo *ADMM* aplicado al *MPCT*, dado un valor muy grande N, el sistema de ecuaciones que hay que resolver en el paso 3 del algoritmo (5) se vuelve bastante grande y denso.

Es concretamente en la ecuación $W\mu = -(b + GH^{-1}q)$. El costo viene de que W deja de ser una matriz diagonal a bloques y pasa a ser en semi banda por lo que el costo computacional que supone aumenta muchísimo respecto al caso del MPCR. En esta sección se pretende mostrar como se puede descomponer la matriz W para conseguir un sistema de ecuaciones mucho menos costoso de resolver.

4.2.1 Fundamento Teórico

Para resolver un sistema de ecuación en semi banda se desarrolla un método que se basa en poder descomponer la matriz del sistema:

$$W\mu = b,$$

en:

$$W = B + U \cdot V. \tag{4.10}$$

B es una matriz de banda estricta. U y V son matrices de cuyo producto salen los elementos no nulos de la fuera de la banda.

Del trabajo previo [5] sabemos que para resolver el sistema y hallar μ se resuelven 3 sistemas de ecuaciones que son menos complejos que el original. A continuación se realizará el desarrollo que permite llegar a estos tres sistemas de ecuaciones.

Sabiendo que tanto W como B son matrices invertibles, aplicando el teorema del determinante de Sylvester [15] se llega:

$$0 \neq det(B+UV) = det(B) \cdot det(I+VB^{-1}U).$$

Como B es invertible el $det(B)$ no puede ser 0 y como W también es invertible el $det(I+VB^{-1}U)$ tampoco puede ser 0 lo que nos lleva directamente a poder afirmar que $I+VB^{-1}U$ es invertible.

Esta condición es muy importante que se cumpla para el siguiente paso ya que vamos a aplicar la matriz de identidad de Woodbury [16]:

$$W^{-1} = (B+UV)^{-1} = B^{-1} - B^{-1}U(I+VB^{-1}U)^{-1}VB^{-1}.$$

Podemos sustituir en el sistema $W\mu = b$ para obtener el vector de soluciones de la siguiente forma:

$$\begin{aligned}
\mu &= W^{-1}b, \\
&= (B^{-1} - B^{-1}U(I+VB^{-1}U)^{-1}VB^{-1})b, \\
&= \mu_1 - B^{-1}U(I+VB^{-1}U)^{-1}V\mu_1, \\
&= \mu_1 - B^{-1}U\mu_2, \\
&= \mu_1 - \mu_3.
\end{aligned}$$

Reagrupando y separando llegamos a:

$$B\mu_1 = b, \tag{4.11}$$
$$(I+VB^{-1}U)\mu_2 = V\mu_1, \tag{4.12}$$
$$B\mu_3 = U\mu_2. \tag{4.13}$$

Resolviendo las ecuaciones anteriores, se halla μ:

$$\mu = \mu_1 - \mu_3. \tag{4.14}$$

4.2.2 Implementación

Para poder implementar esta metodología el primer problema que surge es como llegar a descomponer una matriz de la forma (4.10). En este caso se mostrará como se podría descomponer una matriz W para el caso concreto que nos ocupa.

Tenemos que $W = GH_\rho^{-1}G^T$. El primer paso que vamos a seguir consiste en descomponer H_ρ:

$$H_\rho = \Gamma + \left[\begin{array}{c|c} 0_{NxN} & Y \\ \hline Y^T & 0_{MxM} \end{array}\right] \in \mathbb{R}^{(N+M)x(N+M)}.$$

Γ es una matriz diagonal a bloques que contiene la banda de H_ρ. Una vez se tiene esta descomposición se separa la matriz que contiene los elementos ajenos a la banda en dos matrices de la siguiente forma:

$$H_\rho = \Gamma + \left[\begin{array}{c|c} Y & 0_{NxM} \\ \hline 0_{MxN} & I_M \end{array}\right] \left[\begin{array}{c|c} 0_{MxN} & I_M \\ \hline Y^T & 0_{MxM} \end{array}\right], \tag{4.15}$$
$$U = \left[\begin{array}{c|c} Y & 0_{NxM} \\ \hline 0_{MxN} & I_M \end{array}\right] \in \mathbb{R}^{(N+M)x2M}, \tag{4.16}$$
$$V = \left[\begin{array}{c|c} 0_{MxN} & I_M \\ \hline Y^T & 0_{MxM} \end{array}\right] \in \mathbb{R}^{2Mx(N+M)}. \tag{4.17}$$

El segundo paso que hay que seguir es aplicar la matriz de identidad de Woodbury:

$$\begin{aligned}
H_\rho^{-1} &= (\Gamma+UV)^{-1} \\
&= \Gamma^{-1} - \Gamma^{-1}U(I_{2M}+V\Gamma^{-1}U)^{-1}V\Gamma^{-1}.
\end{aligned}$$

El último paso consistiría en escribir $GH_\rho^{-1}G^T$ como una semibanda:

$$GH_\rho^{-1}G^T = G\Gamma^{-1}G^T - G\Gamma^{-1}U(I_{2M}+V\Gamma^{-1}U)^{-1}V\Gamma^{-1}G^T.$$

Designamos B, U, y V para W

$$B = G\Gamma^{-1}G^T, \tag{4.18}$$
$$\hat{U} = G\Gamma^{-1}U(I_{2M}+V\Gamma^{-1}U)^{-1}, \tag{4.19}$$
$$\hat{V} = V\Gamma^{-1}G^T. \tag{4.20}$$

Todas estas operaciones se pueden llevar a código de la siguiente forma:

Código 4.2 Descomposicón en semi banda.

```
%-------------------------------------
%              ENTRADAS:
%-------------------------------------
%   -Y: Matriz con los elementos de la semibanda de H_{\rho}.
%   -G: Matriz G.
%   -Gamma: Matriz que solo contiene la diagonal de H_{\rho}.
%-------------------------------------
%              SALIDAS:
%-------------------------------------
%   -B: Matriz estrictamente en banda.
%   - U y V: Matrices que cuyo producto tienen los elementos ajenos a la
%   banda.
function [B,U,V] = Descomposicion(Y,G,Gamma)

N=length(Y);
M=width(Y);

U_=[Y , zeros(N,M) ; zeros(M,M), eye(M)];
V_=[zeros(M,N), eye(M); Y' zeros(M,M)];
% Hrec=Gamma+U_*V_;

B=G*inv(Gamma)*G';
V=V_*inv(Gamma)*G';
U=-G*inv(Gamma)*U_*inv(eye(2*M)+V_*inv(Gamma)*U_);

% Comprobación final comp-> 0
% W=G*inv(H)*G';
% RectW=B+U*V;
% Comp=norm(W-RectW)
end
```

Una vez tenemos descompuesto la matriz hay que pensar como podemos usarla de manera eficiente. A partir de este punto se deja de usar el comando *decomposition* de *Matlab* y se usarán solo descomposiciones "clásicas" y matrices dispersas con el fin de generar unos resultados aplicables en otros lenguajes de código como $C++$. Para la matriz W se usará una descomposición LU, y para la matriz B se usará la descomposición de Cholesky.

La idea sería convertir estas matrices en dispersa y luego aplicarles la descomposición:

$$\begin{aligned} W_s &= sparse(W), \\ B_s &= sparse(B), \\ [W_U, W_L] &= LU(W_s), \\ B_{chol} &= LU(B_s). \end{aligned}$$

En el caso donde no se aprovecha la estructura no habría que modificar muchas líneas del código, solo se debería tocar las ecuaciones resueltas don W y meterle la descomposición. Cuando se aprovecha la estructura podemos identificar términos que no son necesarios calcular cada vez que se entre en el bucle. Despejando cada ecuación tenemos lo siguiente:

$$h = -GH_\rho^{-1} q_k - b,$$

$$\mu_1 = B^{-1} h, \tag{4.21}$$

$$\mu_2 = (I + VB^{-1}U)^{-1} V\mu_1, \tag{4.22}$$

$$\mu_3 = B^{-1} U \mu_2. \tag{4.23}$$

En la ecuación donde se resuelve μ_2 hay un término que no varía mientras que no toquemos parámetros del control y el sistema, por lo que podemos calcularlo antes del bucle, para mayor comodidad renombraremos al término: $M_{z2} = (I + VB^{-1}U)$ de forma que la ecuación quedaría así: $\mu_2 = M_{z2} \setminus (V \cdot \mu_1)$.

M_{z2} es un término que se puede calcular fuera del bucle, por lo que nos queda una resolución donde solo hay que resolver dos sistemas de ecuaciones en banda y uno denso pero de pequeña dimensión. El código que resuelve el sistema de ecuaciones que vimos anteriormente se ha modificado para quedar de la siguiente forma:

Código 4.3 solver semibanda.

```
%-------------------------------------
%             ENTRADAS:
%-------------------------------------
%   - q: Vector de la función de coste del problema cuadrático.
%   - b: Vector de las restricciones de igualdad.
%   - Hinv: Inversa de una matriz Hessiana.
%   - dB: descomposición de la matriz que forma la banda de W.
%   - dMz2: (I +V B^{-1}U) Calculado antes del bucle en sparse.
%   - GHinv: Producto G*Hinv
%   -HinvG: Producto de Hinv*G'.
%   -V y U:Matrices que cuyo producto tienen los elementos ajenos a la % %
    banda
%-------------------------------------
%             SALIDAS:
%-------------------------------------
%   - z_opt: Solución al problema
function z_opt = solve_eqQP_sparse(q, b, Hinv,dB,dMz2,GHinv,HinvG,V,U)
    % Hay que hallar MU -- > Wmu=-GH^-1q -b
    %W está descompuesta en B+UV
    h=-GHinv*q-b;

    mu1=dB\(dB'\h);
    mu2=(dMz2)\(V*mu1);
    mu3=dB\(dB'\(U*mu2));
    mu=mu1-mu3;
```

```
    % Solución Óptima
    z_opt = -HinvG*mu - Hinv*q;
end
```

4.3 Resultados

Mismo método que en el capítulo anterior, se realiza una prueba con $N = 30$. Se mostrará como el control funciona, los rendimientos de cada algoritmo desarrollado y las dependencias que tienen con los parámetros.

4.3.1 Prueba control

Para la prueba de control se ha forzado que la referencia se salga completamente de la caja, es decir, ni para la acción de control ni para la salida estaría en rango. En la prueba se aprecia como llega a un punto que minimice el error de la formulación respetando las restricciones. Las pruebas se realizan en los mismos sistemas que en el capítulo anterior, (3.22) y (3.23).

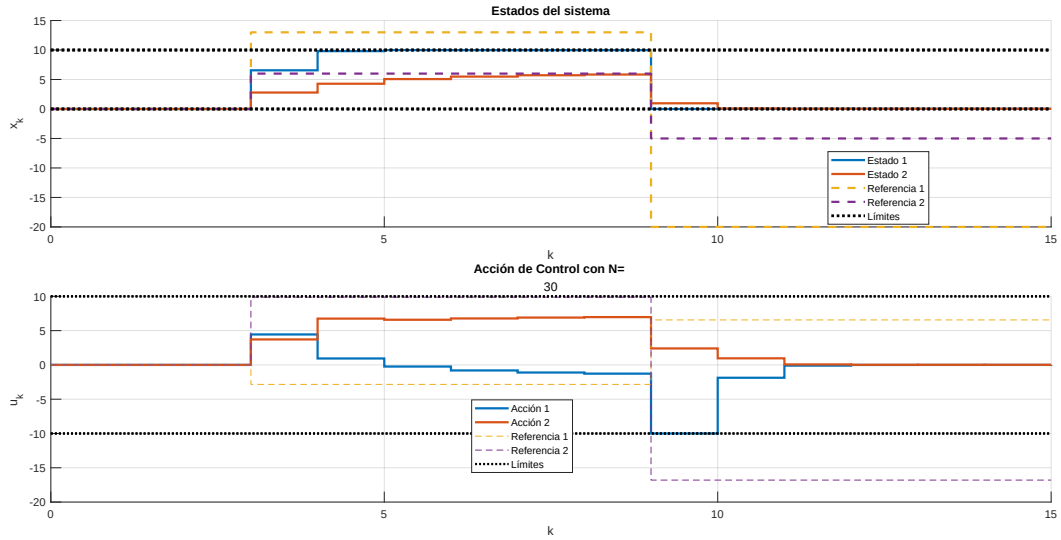

Figura 4.1 Prueba MPCT en un sistema 2x2.

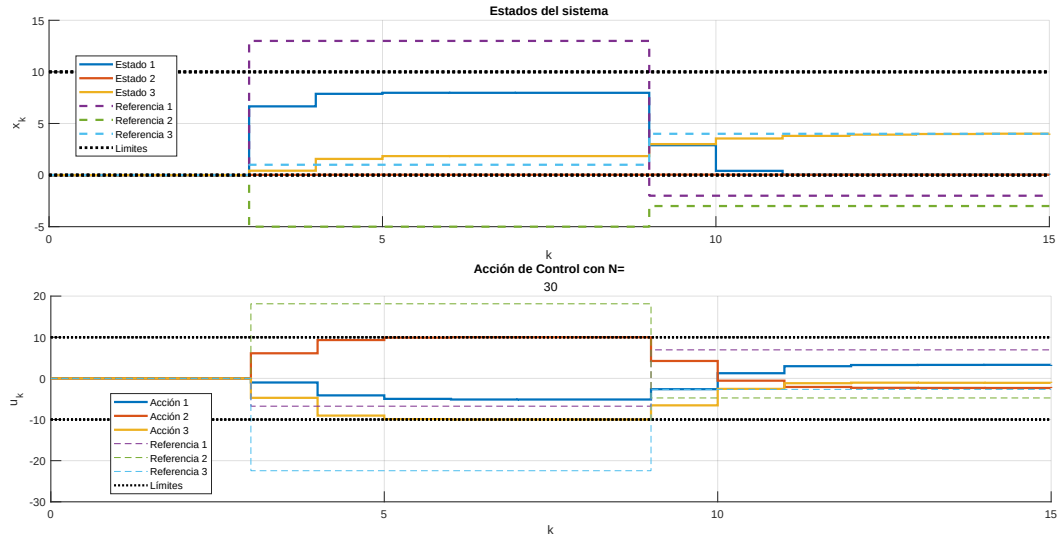

Figura 4.2 Prueba MPCT en un sistema 3x3.

4.3.2 Dependencia de ρ

Como se dijo en el capítulo anterior, ρ es un parámetro que sirve para agilizar en el problema de optimización cuando las restricciones de igualdad se activan. En este capítulo se da una situación especial y es que el *EADMM* tiene un rango de teórico en el que se puede ubicar este parámetro $\rho \in \left(0, \frac{6\mu_3}{17\|C3^T C3\|}\right)$ [8]. En muchos usos se le da a ρ un valor mayor al de su rango teórico, pero esto puede llegar a que el algoritmo pierda las garantías de convergencia.

Los resultados que se mostrarán son los mismos que en el capítulo anterior, gráfica de tiempo medio en resolver un problema de optimización, gráfica de iteraciones medias y tabla con todos los datos. En este caso si se darán 2 tablas para las iteraciones ya que se tratan con 2 métodos bastantes diferenciados. De esta forma se podrá estudiar que método tiene las iteraciones más costosas.

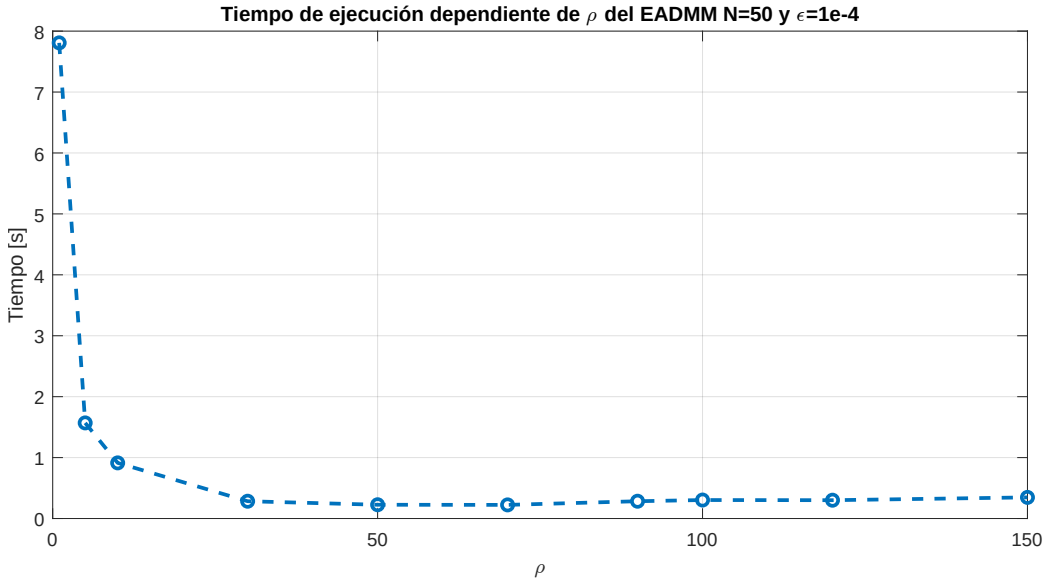

Figura 4.3 Tiempo medio de resolución del *EADMM* en el *MPCT* dependiente de ρ.

La forma de estás gráficas no cambian respecto a las vistas en el capítulo anterior. Y al igual que ocurrió en dicho capítulo, la gráfica de tiempo e iteraciones tienen la misma forma.

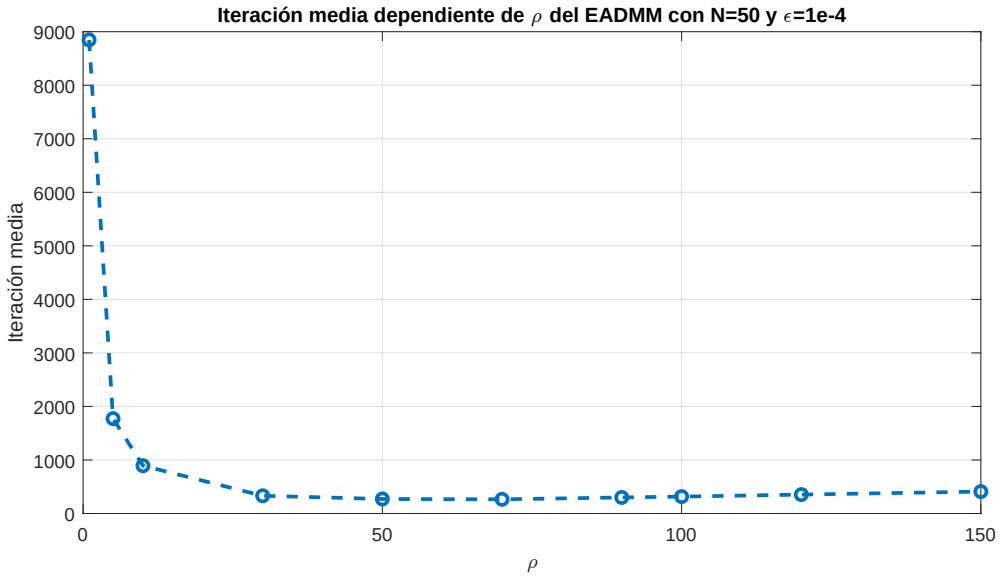

Figura 4.4 Iteraciones media para resolver el *MPCT* con *EADMM* dependiente de ρ.

En las siguientes gráficas, se puede apreciar como el algoritmo *ADMM* es más rápido que el *EADMM*, esto tiene bastante sentido ya que si nos fijamos en el número de operaciones que realizan se ve que el *EADMM* tiene que realizar dos resoluciones de cajas, una de igualdad y además tiene entremedio que hallar tres q, frente al *ADMM* que solo resuelve un problema en caja e igualdad y tiene que hallar solo una q.

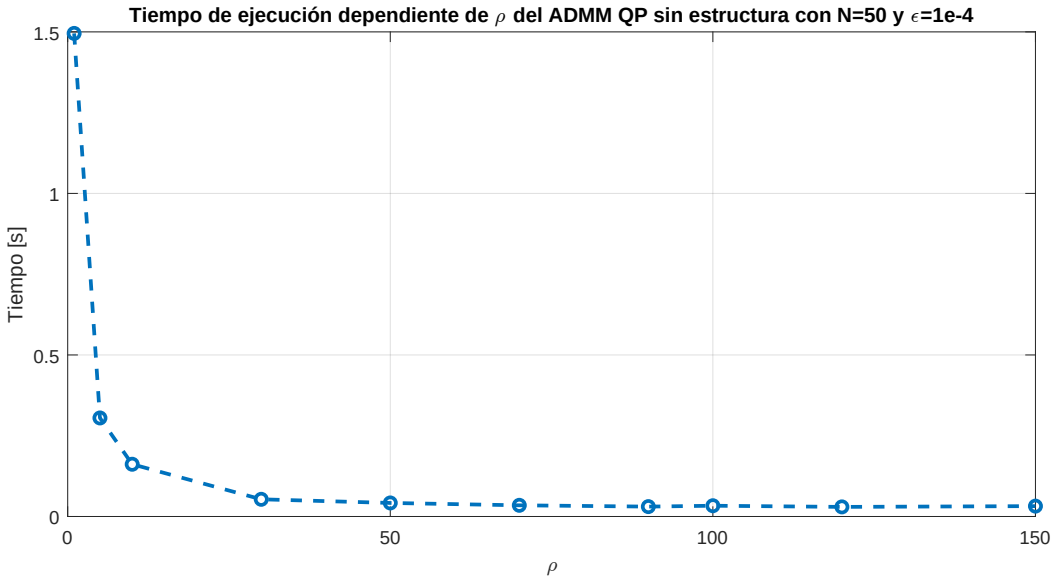

Figura 4.5 Tiempo medio de resolución del *ADMM* sin estructura en el *MPCT* dependiente de ρ.

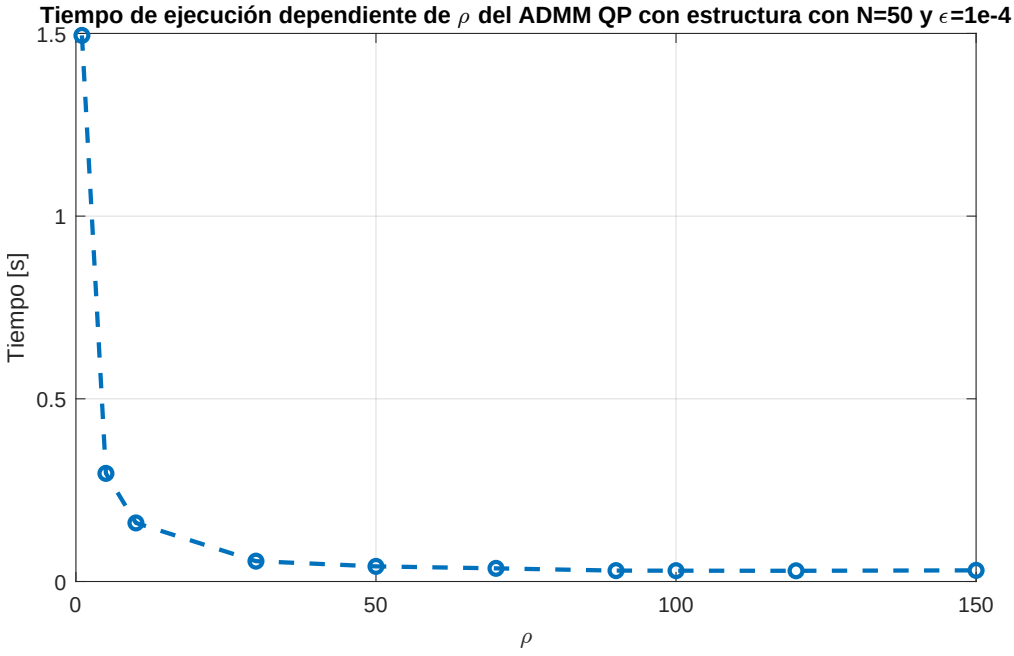

Figura 4.6 Tiempo medio de resolución del *ADMM* con estructura en el *MPCT* dependiente de ρ.

Podemos deducir a partir de estas gráficas, que aprovechar la estructura del problema agiliza el algoritmo y aumenta su rapidez resolviendo los problemas del *MPCT* de forma más eficiente. Esto se verá en más detalle en la gráfica donde se comparará la dependencia con el horizonte de predicción.

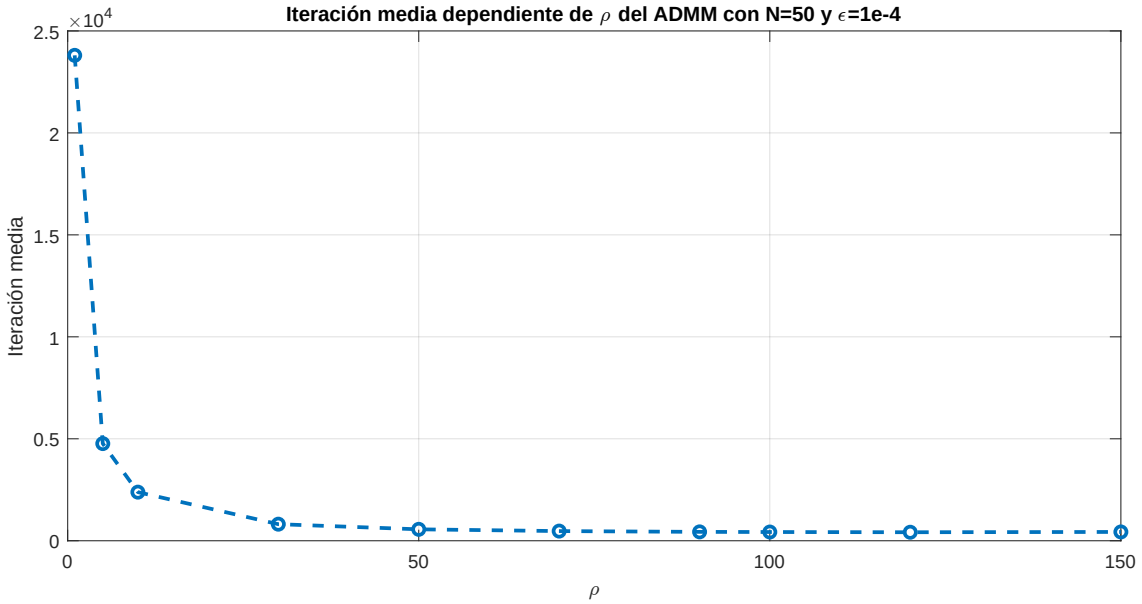

Figura 4.7 Iteraciones media para resolver el *MPCT* con *ADMM* dependiente de ρ.

En las gráficas de iteraciones podemos observar que al *EADMM* le lleva menos iteraciones alcanzar la solución que al *ADMM*, pero aun así es más lento. Esto se debe a que el *EADMM* realiza muchas más operaciones en cada iteración que el *ADMM* lo que lleva a que sea más lento.

Para finalizar esta sección, se dejan las tablas con los datos obtenidos en más detalle. Se mostrarán primero las tablas de los tiempos de ejecución.

Tabla 4.1 Tabla dependencia del tiempo según ρ en el *MPCT* del *EADMM*.

ρ	*EADMM*			
	Tiempo[s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	37,8367	0,0011	1,2908	7,8072
5	7,6804	0,0016	0,2891	1,5694
10	4,517	0,002	0,1818	0,9133
30	1,2002	0,000839	0,1357	0,2839
50	0,6848	0,0018	0,1608	0,2261
70	0,5236	0,000844	0,1755	0,2237
90	0,6842	0,0023	0,2516	0,2839
100	0,737	0,0019	0,2478	0,3034
120	0,7509	0,0017	0,2562	0,3008
150	0,8494	0,0031	0,2718	0,3469

Tabla 4.2 Tabla dependencia del tiempo según ρ en el *MPCT* del *ADMM* sin estructura.

ρ	*ADMM* sin estructura			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	2,3799	0,0019	2,0632	1,4953
5	0,448	0,448	0,3502	0,305
10	0,2594	0,0045	0,1941	0,1617
30	0,0824	0,0105	0,064	0,0532
50	0,0611	0,0153	0,0488	0,0419
70	0,0469	0,0197	0,0406	0,0346
90	0,0441	0,0186	0,0343	0,0306
100	0,051	0,0216	0,0354	0,0333
120	0,0483	0,0177	0,0307	0,0297
150	0,0581	0,0209	0,0314	0,0321

Tabla 4.3 Tabla dependencia del tiempo según ρ en el *MPCT* del *ADMM* con estructura.

ρ	*ADMM* con estructura			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	2,2653	0,0021	1,8586	1,4946
5	0,5048	0,0034	0,3608	0,2959
10	0,2868	0,0047	0,1868	0,1603
30	0,09	0,0098	0,0679	0,0557
50	0,0621	0,0146	0,0437	0,0414
70	0,0488	0,0181	0,0395	0,036
90	0,0432	0,0175	0,0337	0,0298
100	0,0447	0,0192	0,0314	0,0296
120	0,0429	0,0201	0,0288	0,0293
150	0,0522	0,0185	0,0305	0,0305

A continuación las tablas de las iteraciones.

Tabla 4.4 Tabla dependencia de las iteraciones según ρ en el *MPCT* del *EADMM*.

ρ	*EADMM*			
	Iteraciones			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	43456	1	1386	8847
5	8692	2	309	1773
10	4340	2	184	893,5385
30	1407	1	145	331,3846
50	871	2	176	271,1538
70	639	1	222	265,3846
90	631	3	264	299,7692
100	690	2	282	316
120	804	2	318	353,3846
150	974	3	365	409,4615

Tabla 4.5 Tabla dependencia de las iteraciones según ρ en el *MPCT* del *ADMM*.

ρ	*ADMM*			
	Iteraciones			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
1	36044	11	29943	23803
5	7208	30	5991	4764,1
10	3599	52	2995	2383,6
30	1201	128	991	812,6923
50	796	198	602	560,5385
70	640	263	478	473,9231
90	567	293	409	435,4615
100	543	293	392	427,9231
120	494	286	455	417,3077
150	551	280	470	431,9231

De los resultados obtenidos tenemos que ρ tiene un valor óptimo de 120 para el *ADMM* y 70 para el *EADMM*. Como se dijo antes, el rango de ρ del *EADMM* está limitado y en este caso el rango es el siguiente $\rho \in (0,0.3529)$. Por lo que estaríamos dando un valor a ρ fuera de rango, esto hace que ante ciertas situaciones el algoritmo puede llegar a no funcionar.

4.3.3 Dependencia de ε

En esta subsección se muestra la dependencia con el parámetro ε, como ya se explicó en el capítulo anterior, el valor de ε influye en la rapidez de la resolución pero también en la calidad de la misma, pudiendo llegar a arrojar resultados que no satisfacen las restricciones del problema.

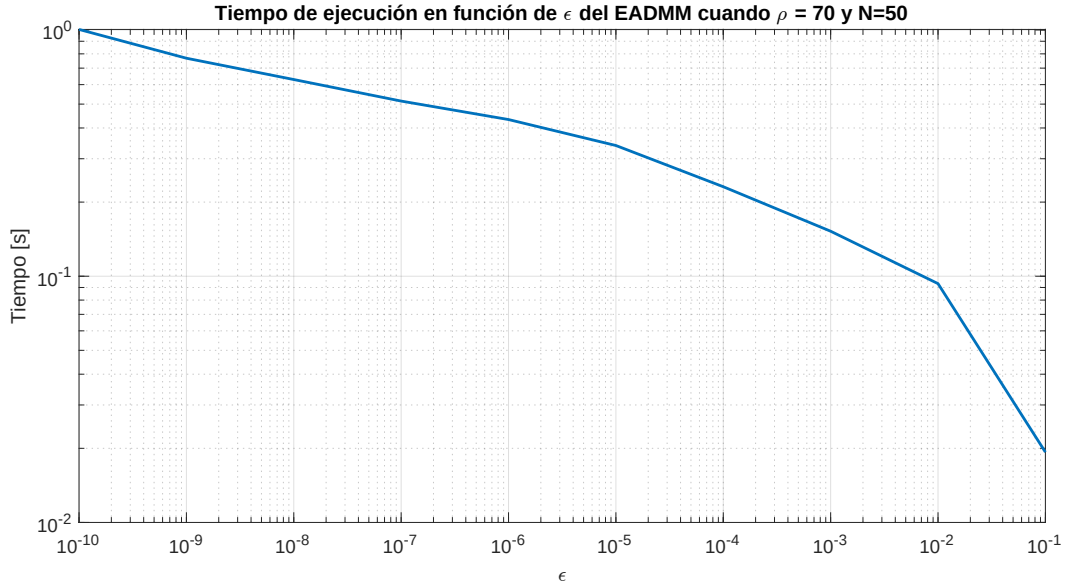

Figura 4.8 Tiempo medio de resolución del *EADMM* en el *MPCT* dependiente de ε.

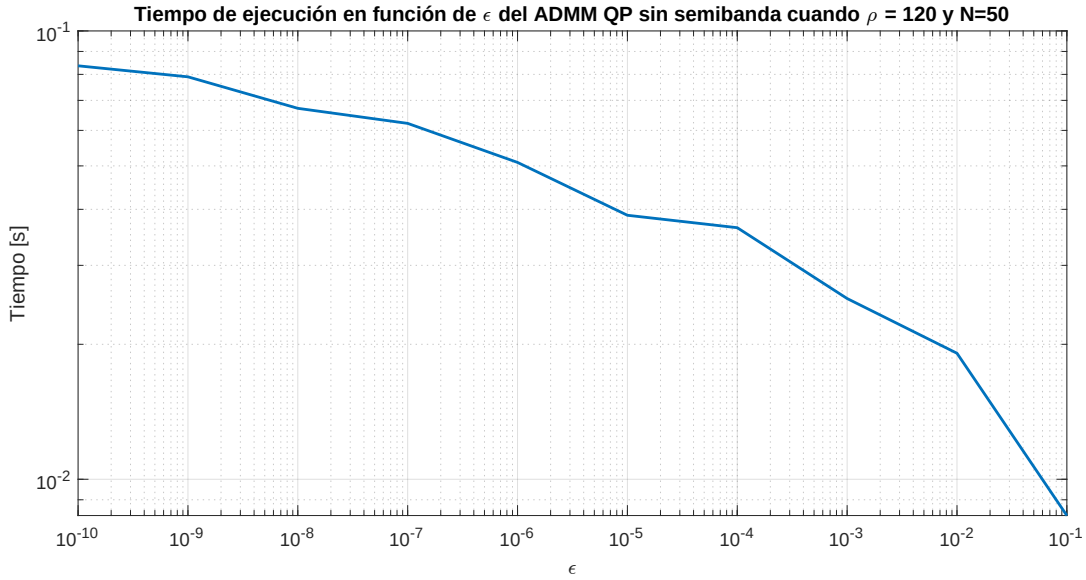

Figura 4.9 Tiempo medio de resolución del *ADMM* sin estructura en el *MPCT* dependiente de ε.

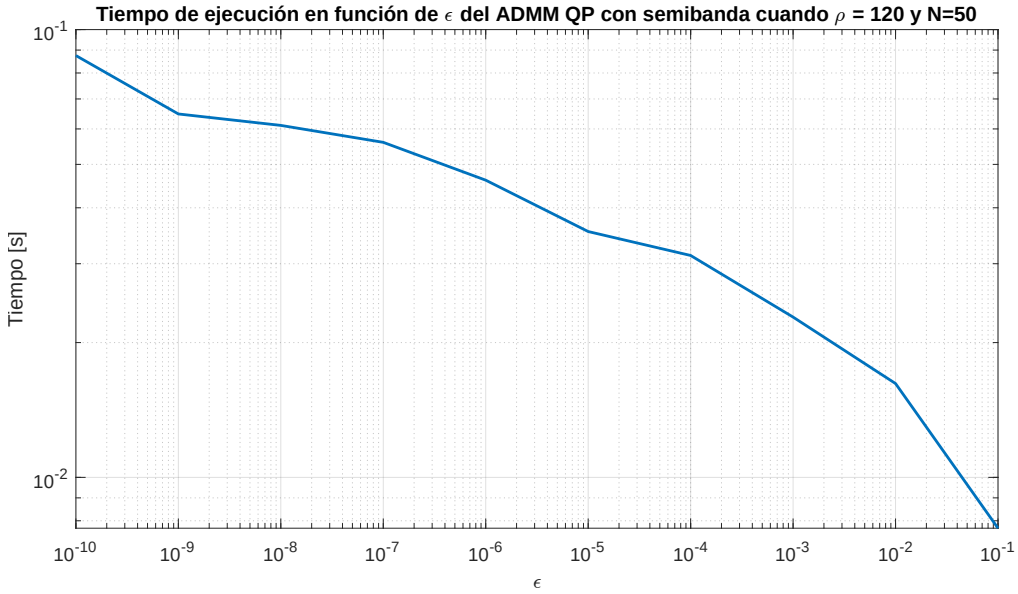

Figura 4.10 Tiempo medio de resolución del *ADMM* con estructura en el *MPCT* dependiente de ε.

Para concluir este capítulo veremos la dependencia con el horizonte de predicción.

4.3.4 Dependencia de N

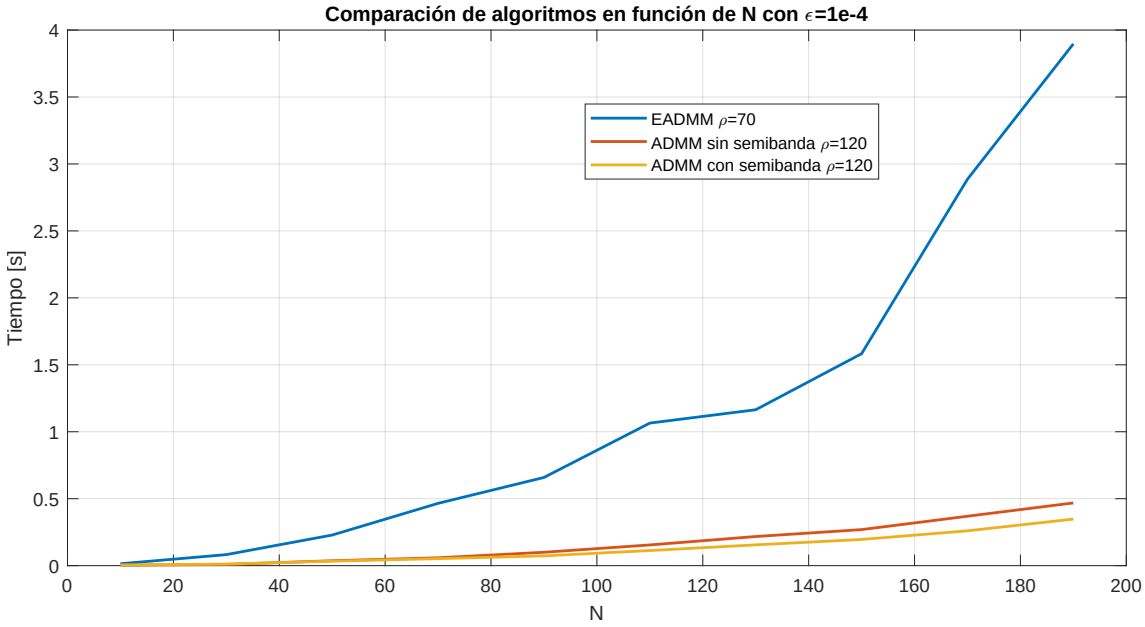

Figura 4.11 Comparación de algoritmos respecto a N en el $MPCT$.

Como se puede observar, el $ADMM$ es más rápido que el $EADMM$, además también queda claro que la incorporación de la descomposición en banda permite hacer al algoritmo más rápido. Para concluir este capítulo se mostrarán las tablas con los valores más en detalle.

Tabla 4.6 Tabla de tiempos en el $MPCT$ del $EADMM$ dependientes de N con $\rho = 70$.

N	$EADMM$ con $\rho = 70$			
	Tiempo[s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
10	0,0537	0,000064	0,0122	0,0141
30	0,201	0,000351	0,072	0,0823
50	0,5911	0,0000873	0,188	0,2285
70	1,1679	0,0032	0,3651	0,4652
90	1,5618	0,004	0,5517	0,6586
110	2,6658	0,0095	0,7688	1,0647
130	2,7358	0,009	0,9976	1,1643
150	400,3077	0,0088	1,4163	1,5829
170	8,0758	0,0175	2,2677	2,8861
190	10,2416	0,0353	3,1179	3,8968

Tabla 4.7 Tabla de tiempos en el *MPCT* del *ADMM* sin estructura dependientes de N con $\rho = 120$.

N	*ADMM* sin estructura y $\rho = 120$			
	Tiempo[s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
10	0,0211	0,0011	0,0015	0,0032
30	0,0296	0,0045	0,0073	0,009
50	0,0475	0,0255	0,04	0,0367
70	0,0811	0,0422	0,061	0,0587
90	0,1355	0,0652	0,1042	0,0999
110	0,2029	0,1113	0,1471	0,1546
130	0,342	0,1303	0,185	0,2175
150	0,3629	0,1831	0,2323	0,2692
170	0,486	0,224	0,3473	0,369
190	0,6351	0,2895	0,4292	0,4683

Tabla 4.8 Tabla de tiempos en el *MPCT* del *ADMM* con estructura dependientes de N con $\rho = 120$.

N	*ADMM* QP con estructura y ρ=120			
	Tiempo [s]			
	Valor Máximo	Valor Mínimo	Valor Mediana	Valor Medio
10	0,0177	0,0022	0,0032	0,0046
30	0,0303	0,0077	0,0096	0,0118
50	0,065	0,0218	0,0312	0,0334
70	0,0715	0,0348	0,052	0,0522
90	0,1006	0,0482	0,0779	0,0729
110	0,1896	0,071	0,11	0,1128
130	0,1978	0,0978	0,1705	0,1552
150	0,2784	0,1279	0,2052	0,1955
170	0,3316	0,1656	0,3026	0,2601
190	0,492	0,2178	0,3629	0,3477

En las tablas podemos observar que efectivamente, el objetivo de este trabajo se ha conseguido, que era conseguir una implementación del *MPCT* usando el *ADMM* de la forma más óptima posible. Para finalizar este capítulo y dar paso a las conclusiones se definirá el entorno usado para las pruebas.

Para las pruebas en las que se ha medido tiempo, el uso de cualquier computador podría suponer un problema pues no se tiene mucho control sobre los procesos que puede llegar a ejecutar en segundo plano, lo que muchas veces se traducen en unas diferencias de tiempos entre una prueba y otra demasiado grandes, por ello en este trabajo se ha optado por usar el entorno que *online* que ofrece *Matlab*. Al estar alojado en un servidor y no en nuestro ordenador se obtienen unos tiempos muy estables en las pruebas que se realizan. Para acabar este trabajo, se procederá con las conclusiones en las que se definirán posibles futuras líneas de investigación a partir de los resultados obtenidos.

5 Conclusiones

No es el conocimiento, sino el acto de aprendizaje, y no la posesión, sino el acto de llegar allí, que concede el mayor disfrute

Carl Friedrich Gauss

En todo trabajo es bueno una vez finalizado pararse a reflexionar sobre los objetivos cubiertos y las aportaciones realizadas al *MPC*.

5.1 Objetivos

Los objetivos de este proyectos eran conseguir una implementación con *ADMM* en el *MPCT* que superase a la implementación clásica con *EADMM*. En este proyecto se ha conseguido dicha implementación y optimizarla usando la estructura del problema.

A lo largo del trabajo se ha indagado desde los conceptos más sencillos en el *MPC* hasta adentrarnos en formulaciones más avanzadas, como es el caso de la formulación de seguimiento (*MPCT*).

Recapitulando un poco todo lo realizado, se ha visto como la implementación de un método resulta más óptima que métodos que ya vienen dentro del propio *Matlab* como se ha visto con el *quadprog*. La implementación del método *ADMM* ha resultado mucho más eficiente que la implementación con *EADMM*, debido a que el método *ADMM* es un método mucho más simple. Por último sea aprovechado de la estructura en semibanda de la matriz W para descomponerla de forma que nos de un sistema de ecuaciones mucho más eficiente para resolver.

5.2 Análisis de resultados

Analizar los resultados es una parte vital de cualquier proyecto. Sirve de guía para saber si lo que se está realizando puede ir a buen puerto. Analizando los resultados de este proyecto como es obvio no se ha intentado buscar una implementación del MPC "fina", por eso en ningún momento se hacen mención a los valores de matrices como Q y R ya que no eran el objeto de estudio. Se muestra una prueba de control en la que se fuerzan las restricciones de desigualdad, para enseñar que efectivamente funciona pero no es el objetivo de este trabajo. El objetivo siempre fue buscar una implementación con *ADMM* que desbancara completamente a la de *EADMM* y como pudimos analizar en el capítulo anterior la implementación fue todo un éxito.

La ventaja se supone la implementación con *ADMM* no solo está en que es más rápida. Este método está mucho más estudiado que el método *EADMM* y no tiene las limitaciones que tiene este respecto al parámetro ρ por lo que siempre se puede usar su valor óptimo y que la teoría garantice un control estable.

5.3 Líneas futuras

Parámetros variables

La implementación de este proyecto no permite que se pueda modificar Q y R, una vez se inicia la prueba de control. Podría resultar interesante desarrollar una implementación que permita la modificación de estas matrices para buscar siempre una acción de control y evitar un control demasiado agresivo.

También sería interesante un desarrollar una implementación que permita variar parámetros del propio sistema, A y B de forma que permita afinar el modelo en caso de que se modificase algo dentro del propio sistema que se este controlando.

Liberación de restricciones

Se ha trabajado durante todo el proyecto con restricción terminal. Una vía que podría resultar de interés podría ser probar los métodos desarrollado cuando se elimina esta. Eliminar la restricción terminal no afectaría ni al algoritmo *ADMM* ni a la descomposición en semibanda, solo habría que modificar H, q, G y b.

Otra restricción que sería interesante liberar es la restricción inicial, de esta forma se pueden evitar situaciones en las que debido a factores como el ruido, la posición inicial del sistema pueda provocar que el problema que deba resolverse deje de ser factible. Es aquí donde entraría esta formulación:

$$\min_{\mathbf{x},\mathbf{u}} J(x,u) + \gamma \parallel x_0 - \hat{x} \parallel_1$$

En la formulación para seguimiento tendría bastante potencial. Como se puede deducir de la expresión, x_0 que es una variable del problema dejaría de valer $\hat{x}$ que es el último valor medido de x antes de volver a resolver de nuevo el problema. Esto es lo que "libera" al problema de la condición inicial.

$$\begin{aligned}
J = \min_{\substack{\mathbf{x},\mathbf{u},\\ x_s,u_s}} & \sum_{j=0}^{N-1} (\parallel x_j - x_s \parallel_Q^2 + \parallel u_j - u_s \parallel_R^2) + \parallel x_s - x_r \parallel_T^2 + \parallel u_s - u_r \parallel_S^2 + \gamma \parallel x_0 - \hat{x} \parallel_1, \\
& x_{j+1} = Ax_j + Bu_j, \\
& x_N = x_s, \\
& x_s = Ax_s + Bu_s, \\
& \underline{x} \leqslant x_j \leqslant \overline{x},\ j \in \mathbb{Z}_1^{N-1}, \\
& \underline{u} \leqslant u_j \leqslant \overline{u},\ j \in \mathbb{Z}_0^{N-1}, \\
& \underline{x} + \varepsilon_x \leqslant x_s \leqslant \overline{x} - \varepsilon_x, \\
& \underline{u} + \varepsilon_u \leqslant u_s \leqslant \overline{u} - \varepsilon_u,
\end{aligned}$$

Se trata de una formulación para seguimiento con restricción terminal y con la condición inicial liberada. Este tipo de formulaciones se estudian en trabajos anteriores como [12].

Implementación en *C*

Una implementación en *C* permitiría introducir el *MPCT* optimizado en un microcontrolador, permitiendo de esta forma su aplicación a sistemas reales. Con la implementación que se ha realizado en *Matlab*, solo se podría introducir en un *Arduino*, lo que nos limita una gran gama de microcontroladores a los que tendríamos acceso con una implementación en *C*.

6 Anexo: códigos de Matlab

En el anexo se exponen todos los códigos desarrollados en Matlab para la obtención de los resultados mostrados a lo largo de este documento. Todo códidigo mostrado a lo largo del trabajo no se volverá a enseñar en este anexo.

6.1 Anexo A: Códigos para el MPCS

Construcción del vector q

La siguiente función se ha usado para actualizar el vector q en caso de cambio de referencia.

Código 6.1 construcción de q.

```
function q = ConstruccionQ(R,Q,ur,xr,nz)
q=[];
while width(q)<(nz-width((R*ur)'))
       q=[q (R*ur)'];
       q=[q (Q*xr)'];
end
q=[q (R*ur)'];
q=-q';
end
```

Construcción de la matriz G

Construye la matriz G del problema cuadrático dadas las matrices del sistema, sus dimensiones y el horizonte de predicción

Código 6.2 construcción de G.

```
function G = ConstruccionG(A, B, N, n, m)

nz=(N-1)*(n+m)+m;
mz=N*n;
I=eye(n);
G=zeros(mz,nz);
i=1;
for j=1:n:mz
    if (j==1 && i==1)
        G(1:n,1:(m+n))=[B -I];

   elseif (j~=mz && j~=1)
```

```
        if(j==(mz-n+1))
          G(j:j+n-1,nz-m-n+1:nz)=[A B];
        else
        ini=(m+1)+(i-2)*(m+n);
        fin=ini+2*n+m-1;
          G(j:j+n-1,ini:fin)=[A B -I];
        end
    end
    i=i+1;
end
end
```

Simulación del MPCS

Se monta todo y se prueba. En el código se permite cambiar el horizonte de predicción, la tolerancia, el parámetro ρ, el sistema que se desee controlar. En el bucle de control se deja sin comentar la implementación que se desee usar y se dejan las otras 2 comentadas.

Código 6.3 Simulación MPCS.

```
%% Definimos el sistema
%-------------------------------------------------
%          Sistema
%-------------------------------------------------
clc
clear
n=3;
m=2;
N=10; % Horizonte de predicción
% Matrices del sistema

if (n==2)
    A=[0.257846170112605 0.152234012862946;-0.331665238742629
        0.348007659716113];
    B=[0.121658454307726; 0.884153057749660];
    xplus=6*ones(n,1);
    xmin=-2*ones(n,1);
    uplus=5*ones(m,1);
    umin=-5*ones(m,1);
    xr=[1.60127860072888 ;4.60975320704808];
    ur=4;
elseif (n==3)
    A=[0.184194097515527 0.134122932828682 0.0714528127870445;...
    0.597211350337855 0.212601533358843 0.242486558936719;...
    0.299936990089789 0.894941675440814 0.0537543922087138];
    B=[0.441722057064424 0.196658191367632;...
    0.0132832004672525 0.0933705167550930;...
    0.897191350973572 0.307366899587920];
    xplus=10*ones(n,1);
    xmin=-2*ones(n,1);
    uplus=5*ones(m,1);
    umin=-5*ones(m,1);
    xr=[2.68735511058330;4.42592230171853;7.25883744758796];
    ur=[2;1];
end
```

```
x=zeros(n,N+1);
u=zeros(m,N);

u(:,1)=-1*ones(m,1);
x0=zeros(n,1);

%Restriccion terminal
x(:,N+1)=xr;
x(:,1)=x0;

%% Construcción Problema QP
%-----------------------------------------------------
%          Problema QP
%-----------------------------------------------------
clc
I=eye(n);
nz=(N-1)*(n+m)+m;
mz=N*n;
z=[];
zmas=[];
zmenos=[];

z=[];
for k=1:1:N
    z=[z; u(:,k)];
    zmas=[zmas;uplus];
    zmenos=[zmenos;umin];
    if (k+1~=N+1)
    z=[z; x(:,k+1)];
    zmas=[zmas;xplus];
    zmenos=[zmenos;xmin];
    end
end


% Construimos Q y R

% Construimos Q y R
Q=diag(ones(n,1));
R=diag(ones(m,1));


%Matriz H
H=blkdiag(R,Q);
while rank(H)<(nz-n)
    H=blkdiag(H,R,Q);
end
H=blkdiag(H,R);
% Definimos q

q=[];

while width(q)<(nz-width((R*ur)'))
       q=[q (R*ur)'];
       q=[q (Q*xr)'];
end
q=[q (R*ur)'];
```

```
q=-q';

% Definimos la matriz G que es mzxnz
G=ConstruccionG(A,B,N,n,m);

b= zeros(mz,1);
b(1:n,:)=-A*x0;
b(mz-n+1:mz,:)=xr';


%-------------------------------------
%      Estructuramos el problema
%-------------------------------------
% Estructura del problema
problem=Estructura(H,q,G,b,zmenos,zmas,z);

%% Simulación del MPC Standar
%---------------------------------
%      Bucle de control
%---------------------------------
clc
Nsim=10;
X_Sim=zeros(n,Nsim);
U_Sim=zeros(m,Nsim);
U_ref=ones(m,Nsim);
X_ref=ones(n,Nsim);
tiempo_ej=zeros(Nsim,1);
z_opt=z;
T_sim=0:1:Nsim-1;
xk=x0; % Condición inicial
u_manual=3*ones(m,1);
epsilon=1e-4;
ro=1;

tic
for i=1:1:Nsim

%  if i > 10
% xr=[0.800639300364441 ;2.30487660352404];
% ur=2;
% end

if i<2
    %  x_j+1= x_jA + u_jB
x_kk=A*xk+B*(u_manual); % Sistema
X_Sim(:,i)=x_kk;
U_Sim(:,i)=u_manual;
xk=x_kk;


%Empezamos a controlar
else % Modo automatico
q=ConstruccionQ(R,Q,ur,xr,nz);
b(1:n,:)=-A*xk; % x0 varía, actualizamos b
z=z_opt; % Nuevo punto de inicio
problem.x0=z;
problem.beq=b;
```

```
problem.f=q;

%z_opt=quadprog(problem); %quadprog
z_opt=ADMM_Dual(q,b,ro,epsilon,H,G,zmas,zmenos,z,nz); %ADMM con estructura
%z_opt=ADMM(epsilon,ro,z,zmenos,q,zmas,zmenos,H,G,b); % ADMM
u_aut=z_opt(1:m,1);


x_kk=A*xk+B*(u_aut);
X_Sim(:,i)=x_kk;
U_Sim(:,i)=u_aut;
X_ref(:,i)=xr;
U_ref(:,i)=ur;
xk=x_kk;
end
end
t_medido=toc



%%
%---------------------------------------
%   GRÁFICAS
%---------------------------------------
if n==3
figure(1)
subplot(2,1,1)
hold on
plot(T_sim,X_Sim,LineWidth=1.5)
stairs(T_sim,X_ref',LineStyle="--",LineWidth=2)
plot(T_sim,ones(1,Nsim).*xplus,LineWidth=2.5,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim).*xmin,LineWidth=2.5,Color='Black',LineStyle=':')
title("Estados de sistema")
legend("Estado 1","Estado 2","Estado 3","Referencia 1","Referencia 2","
    Referencia 3","Limites")
xlabel("k")
ylabel("x_{k}")
grid on
hold off
subplot(2,1,2)
hold on
stairs(T_sim,U_Sim',LineWidth=2)
stairs(T_sim,U_ref',LineStyle="--")
plot(T_sim,ones(1,Nsim).*uplus,LineWidth=2,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim).*umin,LineWidth=2,Color='Black',LineStyle=':')
title('Acción de Control con N= ',num2str(N))
legend("Acción 1","Acción 2","Referencia 1","Referencia 2","Límites")
xlabel("k")
ylabel("u_{k}")
grid on
hold off
end
if n==2
figure(1)
subplot(2,1,1)
hold on
plot(T_sim,X_Sim,LineWidth=1.5)
```

```
stairs(T_sim,X_ref',LineStyle="--",LineWidth=2)
plot(T_sim,ones(1,Nsim).*xplus,LineWidth=2.5,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim).*xmin,LineWidth=2.5,Color='Black',LineStyle=':')
title("Estados de sistema")
legend("Estado 1","Estado 2","Referencia 1","Referencia 3","Límites")
xlabel("k")
ylabel("x_{k}")
grid on
hold off
subplot(2,1,2)
hold on
stairs(T_sim,U_Sim',LineWidth=2)
stairs(T_sim,U_ref',LineStyle="--")
plot(T_sim,ones(1,Nsim).*uplus,LineWidth=2,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim).*umin,LineWidth=2,Color='Black',LineStyle=':')
title('Acción de Control con N= ',num2str(N))
legend("Acción 1","Referencia","Límites")
xlabel("k")
ylabel("u_{k}")
grid on
hold off
end
```

6.2 Anexo B: Códigos para el MPCT

Simulación MPCT con EADMM

Código 6.4 Simulación MPCT con EADMM.

```
%% Definimos el sistema
%-------------------------------------------------
%          Sistema
%-------------------------------------------------
clc
clear all
n=2;
m=2;
N=50; % Horizonte de predicción
% Matrices del sistema
if (n==2 && m==2)
    A=[0.276922984960890 0.0971317812358475;0.0461713906311539
        0.823457828327293];
    B=2*[0.294066333758628 0.530872257027928;0.237373019705579
        0.0914987313394122];
    C=[B A*B];
    sist_cont=rank(C)
    xplus=10*ones(n,1);
    xmin=0*ones(n,1);
    uplus=10*ones(m,1);
    umin=-10*ones(m,1);
    xr1=[13;6];
    ur1=B\(xr1-A*xr1);
    xr2=[-20;-5];
    ur2=B\(xr2-A*xr2);
```

```
elseif (n==3 && m==3)
A=[0.105629203329022 0.423452918962738 0.153656717591307;...
0.610958658746201 0.0908232857874395 0.281005302533871;...
0.778802241824093 0.266471490779072 0.440085139001721];
B=2*[0.484853333552102,0.741257943454207,0.149997253831683;...
    0.393456361215266,0.520052467390387,0.586092067231462;...
    0.671431139674026,0.347712671277525,0.262145317727807];
    xplus=10*ones(n,1);
    xmin=0*ones(n,1);
    uplus=10*ones(m,1);
    umin=-10*ones(m,1);
    xr1=[13 ;-5; 1];
    ur1=B\(xr1-A*xr1);
    xr2=[-2;-3;4];
    ur2=B\(xr2-A*xr2);
elseif (n==2 && m==1)
     A=[0.257846170112605 0.152234012862946;-0.331665238742629
         0.348007659716113];
    B=[0.121658454307726; 0.884153057749660];
    xplus=6*ones(n,1);
    xmin=-2*ones(n,1);
    uplus=5*ones(m,1);
    umin=-5*ones(m,1);
    xr1=[1.60127860072888 ;4.60975320704808];
    ur1=4;
    xr2=xr1;
    ur2=ur1;
end
xr=xr1;
ur=ur1;

x=zeros(n,N+1);
u=zeros(m,N);



u(:,1)=-1*ones(m,1);
x0=zeros(n,1);
%Restriccion terminal
x(:,N+1)=xr;
x(:,1)=x0;

In=eye(n);
Inm=eye(n+m);
ceronm=zeros(n+m,m+m);
cero=zeros(n,m);
%% Adaptación problema
% Construimos C1
clc
FilasC1=n+n+m+(n+m)*(N+1);
ColumnasC1=(N+1)*(n+m);
C1=zeros(FilasC1,ColumnasC1);
C1(1:n,1:n)=In;
diagonalC1=-eye((n+m)*(N+1));
C1(n+1:FilasC1-n-m,:)=diagonalC1;
% Ultima fila de C1
C1(FilasC1-n-m+1:end,((N+1)*(n+m))-(n+m-1):(N+1)*(n+m))=-Inm;
```

```
%Construimos C2
ColumnasC2=n+m;
FilasC2=n+n+m+(n+m)*(N+1);
C2=zeros(FilasC2,ColumnasC2);
k=n+1;
while k <= FilasC2
C2(k:k+n+m-1,:)=Inm;
k=k+n+m;
end
%Construccion C3
FilasC3=n+n+m+(n+m)*(N+1);
ColumnasC3=(N+1)*(n+m);
C3=zeros(FilasC3,ColumnasC3);
diagonalC3=-eye((n+m)*(N+1));
C3(n+1:FilasC1-n-m,:)=diagonalC3;

% Construccion de b
b=zeros(FilasC1,1);
b(1:n)=x0;


%-----------------------------
%      Matrices R,Q,S,T
%-----------------------------
% Construimos Q y R
Q=diag(ones(n,1))*120;

R=diag(ones(m,1));

 % Construimos S y T
T=diag(ones(n,1))*120;

S=diag(ones(m,1));

mu3=min(eig(blkdiag(Q,R)));
rango_Rho= 6*mu3/(17*norm(C3'*C3))
VectorRo=[1 5 10 30 50 70 90 100 120 150];
ro=VectorRo(6); % Más rápido Ro=70 (posición 6)

epsilons=[1e-1 1e-2 1e-3 1e-4 1e-5 1e-6 1e-7 1e-8 1e-9 1e-10];
epsilon=epsilons(4); % La 4 por defecto



%% Problema 1
clc
H1=ro*C1'*C1;
zmas1=[];
zmenos1=[];
epsilon_u=0.01*ones(m,1);
epsilon_x=0.03*ones(n,1);
Mn=rand(n,1)*10000; % Numero grande
zmas1=Mn;
zmenos1=-Mn;
k=1;
while length(zmas1)<((n+m)*(N+1)-n-m)
```

```
    zmas1=[zmas1;uplus];
    zmenos1=[zmenos1;umin];
    if (k+1~=N+1)
    zmas1=[zmas1;xplus];
    zmenos1=[zmenos1;xmin];
    end
    k=k+1;
end
zmas1=[zmas1;xplus-epsilon_x;uplus-epsilon_u];
zmenos1=[zmenos1;xmin+epsilon_x;umin+epsilon_u];
invH1=inv(H1);
%% Problema 2
clc
H2=blkdiag(T,S)+ro*(C2'*C2);
G2=[(A-In) B];
b2=zeros(n,1);
invH2=inv(H2)
invH2G2=inv(G2*invH2*G2')
M2=invH2*G2'*invH2G2*G2*invH2-invH2;

%% Problema 3
clc
H3=blkdiag(Q,R);
while rank(H3)<((n+m)*(N+1))
    H3=blkdiag(H3,Q,R);
end
H3=H3+ro*C3'*C3;
b3=zeros(N*n,1);

G3=zeros(N*n,(n+m)*(N+1));
G3(1:n,1:n+m+n)=[A B -In];
k=1;
for i=n+1:n:N*n
    columnaobj=(n*k)+m + n +m +n;
    G3(i:n+i-1,1:columnaobj)=[zeros(n,(n*k)+m) A B -In];
    k=k+1;
end
Hinv3=inv(H3);
W3=G3*Hinv3*G3';
dW3=decomposition(W3);
[W_L,W_U]=lu(sparse(W3));


GHinv3=G3*Hinv3;
HinvG3=Hinv3*G3';

%% Implementacion MPCT con EADMM

%-----------------------------------
%      Bucle de control
%-----------------------------------

inicio=3;
xr=xr1;
ur=ur1;
Nsim=15;
X_Sim=zeros(n,Nsim+1);
```

```
U_Sim=zeros(m,Nsim+1);
U_ref=zeros(m,Nsim+1);
X_ref=zeros(n,Nsim+1);
K_Sim=zeros(Nsim+1-inicio,1);
T_sim=0:1:Nsim;
Tiempo_Res=zeros(1,Nsim+1-inicio);
xk=x0; % Condición inicial
u_manual=0*ones(m,1);


% z1_0=ones((n+m)*(N+1),1);
z1_0(1:n)=x0;
z2_0=zeros(n+m,1);
z3_0=zeros((n+m)*(N+1),1);
lambda_0=zeros(n+n+m+(n+m)*(N+1),1);


tic
for i=1:1:Nsim+1

if i==10
xr=xr2;
ur=ur2;
end

if i<=inicio
    %  x_j+1= x_jA + u_jB
x_kk=A*xk+B*(u_manual); % Sistema
X_Sim(:,i)=x_kk;
U_Sim(:,i)=u_manual;
xk=x_kk;
else % Modo automatico
b(1:n)=x_kk;
tic
[z1_opt, z2_opt, z3_opt ,lambda_opt,iteraciones]=EADMM(z2_0,z3_0,lambda_0,C1,C2
    ,C3,ro,b,b3,zmenos1,zmas1,T,S,xr,ur,epsilon,invH1,Hinv3,M2,W_U,W_L,GHinv3,
    HinvG3);
time=toc;
K_Sim(i-inicio)=iteraciones;
z1_0=z1_opt;
z2_0=z2_opt;
z3_0=z3_opt;
lambda_0=lambda_opt;


u_aut=z1_opt(n+1:n+m,1);
x_kk=A*xk+B*(u_aut);
X_Sim(:,i)=x_kk;
U_Sim(:,i)=u_aut;
X_ref(:,i)=xr;
U_ref(:,i)=ur;
xk=x_kk;
Tiempo_Res(1,i-inicio)=time;

end
end
```

```
it_max=max(K_Sim)
it_min=min(K_Sim)
it_mediana=median(K_Sim)
it_medio=mean(K_Sim)

Tiempo_max=max(Tiempo_Res)
Tiempo_min=min(Tiempo_Res)
Tiempo_mediana=median(Tiempo_Res)
Tiempo_medio=mean(Tiempo_Res)


%% Graf

%---------------------------------------
%   GRÁFICAS
%---------------------------------------
if n==3
figure(1)
subplot(2,1,1)
hold on
stairs(T_sim,X_Sim',LineWidth=2)
stairs(T_sim,X_ref',LineStyle="--",LineWidth=2)
plot(T_sim,ones(1,Nsim+1).*xplus,LineWidth=2.5,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim+1).*xmin,LineWidth=2.5,Color='Black',LineStyle=':')
title("Estados del sistema")
legend("Estado 1","Estado 2","Estado 3","Referencia 1","Referencia 2","
    Referencia 3","Limites")
xlabel("k")
ylabel("x_{k}")
grid on
hold off
subplot(2,1,2)
hold on
stairs(T_sim,U_Sim',LineWidth=2)
stairs(T_sim,U_ref',LineStyle="--")
plot(T_sim,ones(1,Nsim+1).*uplus,LineWidth=2,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim+1).*umin,LineWidth=2,Color='Black',LineStyle=':')
title('Acción de Control con N= ',num2str(N))
legend("Acción 1","Acción 2","Acción 3","Referencia 1","Referencia 2","
    Referencia 3","Límites")
xlabel("k")
ylabel("u_{k}")
grid on
hold off
end
if n==2
figure(1)
subplot(2,1,1)
hold on
stairs(T_sim,X_Sim',LineWidth=2)
stairs(T_sim,X_ref',LineStyle="--",LineWidth=2)
plot(T_sim,ones(1,Nsim+1).*xplus,LineWidth=2.5,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim+1).*xmin,LineWidth=2.5,Color='Black',LineStyle=':')
title("Estados del sistema")
legend("Estado 1","Estado 2","Referencia 1","Referencia 2","Límites")
xlabel("k")
```

```
ylabel("x_{k}")
grid on
hold off
subplot(2,1,2)
hold on
stairs(T_sim,U_Sim',LineWidth=2)
stairs(T_sim,U_ref',LineStyle="--")
plot(T_sim,ones(1,Nsim+1).*uplus,LineWidth=2,Color='Black',LineStyle=':')
plot(T_sim,ones(1,Nsim+1).*umin,LineWidth=2,Color='Black',LineStyle=':')
title('Acción de Control con N= ',num2str(N))
legend("Acción 1","Acción 2","Referencia 1","Referencia 2","Límites")
xlabel("k")
ylabel("u_{k}")
grid on
hold off
end
```

Simulación del MPCT con ADMM

En el bucle de control se deja sin comentar que versión del ADMM se desee usar.

Código 6.5 Simulación del MPCT con ADMM.

```
%% Definimos el sistema
clear all
%-------------------------------------------------
%          Sistema
%-------------------------------------------------
clc
n=2;
m=2;
N=50; % Horizonte de predicción
% Matrices del sistema
if (n==2 && m==2)
    A=[0.276922984960890 0.0971317812358475;0.0461713906311539
        0.823457828327293];
    B=2*[0.294066333758628 0.530872257027928;0.237373019705579
        0.0914987313394122];
    C=[B A*B];
    sist_cont=rank(C)
    xplus=10*ones(n,1);
    xmin=0*ones(n,1);
    uplus=10*ones(m,1);
    umin=-10*ones(m,1);
    xr1=[13;6];
    ur1=B\(xr1-A*xr1);
    xr2=[-20;-5];
    ur2=B\(xr2-A*xr2);
elseif (n==3 && m==3)
A=[0.105629203329022 0.423452918962738 0.153656717591307;...
0.610958658746201 0.0908232857874395 0.281005302533871;...
0.778802241824093 0.266471490779072 0.440085139001721];
B=2*[0.484853333552102,0.741257943454207,0.149997253831683;...
    0.393456361215266,0.520052467390387,0.586092067231462;...
    0.671431139674026,0.347712671277525,0.262145317727807];
    xplus=10*ones(n,1);
```

```
    xmin=0*ones(n,1);
    uplus=10*ones(m,1);
    umin=-10*ones(m,1);
    xr1=[13 ;-5; 1];
    ur1=B\(xr1-A*xr1);
    xr2=[-2;-3;4];
    ur2=B\(xr2-A*xr2);
elseif (n==2 && m==1)
     A=[0.257846170112605 0.152234012862946;-0.331665238742629
         0.348007659716113];
    B=[0.121658454307726; 0.884153057749660];
    xplus=6*ones(n,1);
    xmin=-2*ones(n,1);
    uplus=5*ones(m,1);
    umin=-5*ones(m,1);
    xr1=[1.60127860072888 ;4.60975320704808];
    ur1=4;
    xr2=xr1;
    ur2=ur1;
end
xr=xr1;
ur=ur1;

x=zeros(n,N+1);
u=zeros(m,N);

u(:,1)=-1*ones(m,1);
x0=zeros(n,1);
%Restriccion terminal
x(:,N+1)=xr;
x(:,1)=x0;

In=eye(n);
Inm=eye(n+m);
ceronm=zeros(n+m,m+m);
cero=zeros(n,m);
%% Adaptación del problema
%-----------------------------
%      Matrices R,Q,S,T -> Usamos la Identidad
%-----------------------------
clc
% Construimos Q y R
Q=diag(ones(n,1))*120;

R=diag(ones(m,1));


 % Construimos S y T
T=diag(ones(n,1))*120;

S=diag(ones(m,1));

nz=(n+m)*(N+1);

% Construcción de H
H=blkdiag(Q,R);
while rank(H)<(nz-(n+m))
```

```
    H=blkdiag(H,Q,R);
end
H=blkdiag(H,N*Q+T,N*R+S);
gamma=2*H;

filasR=[];
while (length(filasR)<(nz-m-n))
       filasR=[filasR ;zeros(n,m)];
       filasR=[filasR ;-R];
end
filasR;

filasQ=[];
while (length(filasQ)<(nz-m-n-m))
       filasQ=[filasQ ;-Q];
       filasQ=[filasQ ;zeros(m,n)];
end
filasQ;

H(1:length(filasR),length(H)-width(R)+1:end)=filasR;
H(length(H)-width(R)+1:end,1:length(filasR))=filasR';
H(1:length(filasQ),length(H)-width(Q)-width(R)+1:end-width(R))=filasQ;
H(length(H)-width(Q)-width(R)+1:end-width(R),1:length(filasQ))=filasQ';
H=2*H;
%Construcción de q
q=zeros(nz,1);
q(end-(m-1):end)=S*ur;
q(end-(m+n-1):end-m)=T*xr;
q=-2*q;
%Construccion G

I=eye(n);
mz=(N+1)*n;
G=zeros(mz,nz);

i=1;
for j=1:n:mz
    if (j==1 && i==1)
        G(1:n,1:n)=[I];
   elseif (j~=mz && j~=1)
         if(j==(mz-n+1))
           G(j:j+n-1,nz-m-n+1:nz)=[A-I B];
         else
         ini=(n+m)*(i-2)+1;
         fin=ini+(n+n+m)-1;
           G(j:j+n-1,ini:fin)=[A B -I];
         end
    end
    i=i+1;
end
b=zeros(mz,1);
b(1:n)=x0;

vmas=[];
vmenos=[];
epsilon_u=0.01*ones(m,1);
epsilon_x=0.03*ones(n,1);
```

```
k=1;
while length(vmas)<(nz-n-m-1)
    vmas=[vmas;xplus];
    vmenos=[vmenos;xmin];

    vmas=[vmas;uplus];
    vmenos=[vmenos;umin];
end

vmas=[vmas;xplus-epsilon_x;uplus-epsilon_u];
vmenos=[vmenos;xmin+epsilon_x;umin+epsilon_u];

VectorRo=[1 5 10 30 50 70 90 100 120 150];
ro=VectorRo(9);

Y=[2*filasQ 2*filasR];
Hro=H+eye(nz)*ro;
gamma=gamma +eye(nz)*ro;
[Bb,U,V]=Descomposicion(Y,G,gamma); % Problema descompuesto

Bb_s=sparse(Bb);
Bb_chol=chol(Bb_s);
dB_s=decomposition(Bb_s,"banded"); % Descomponemos en banda


Hro=H+eye(nz)*ro;
Hroinv=inv(Hro);
invro=inv(ro);
W=G*Hroinv*G';
% dW=decomposition(W); % W descompuesta sin aprovechar estructura
%[W_L,W_U]=lu(W);
[W_L,W_U]=lu(sparse(W));

I_U=eye(width(U));
invBb=inv(Bb);
Mz2=(I_U+V*invBb*U);
dMz2=sparse(Mz2);
GHinv=G*Hroinv;
HinvG=Hroinv*G';


%% Implementacion MPCT con ADMM

%-----------------------------------
%      Bucle de control
%-----------------------------------
clc

inicio=3;
xr=xr1;
ur=ur1;
Nsim=15;
X_Sim=zeros(n,Nsim+1);
U_Sim=zeros(m,Nsim+1);
U_ref=zeros(m,Nsim+1);
X_ref=zeros(n,Nsim+1);
K_Sim=zeros(Nsim+1-inicio,1);
```

```
Tiempo_Res=zeros(1,Nsim+1-inicio);
z0=zeros(nz,1);
T_sim=0:1:Nsim;
xk=x0; % Condición inicial
u_manual=0*ones(m,1);
epsilons=[1e-1 1e-2 1e-3 1e-4 1e-5 1e-6 1e-7 1e-8 1e-9 1e-10];
epsilon=epsilons(10); % Por defecto 4

for i=1:1:Nsim+1

if i==10
xr=xr2;
ur=ur2;
end

if i<=inicio
 %  x_j+1= x_jA + u_jB
x_kk=A*xk+B*(u_manual); % Sistema
X_Sim(:,i)=x_kk;
U_Sim(:,i)=u_manual;
xk=x_kk;

else % Modo automatico
b(1:n)=x_kk;
q=zeros(nz,1);
q(end-(m-1):end)=S*ur;
q(end-(m+n-1):end-m)=T*xr;
q=-2*q;
tic
%[z_opt,iteraciones]=ADMM_QP(q,b,ro,epsilon,HinvG,vmas,vmenos,z0,nz,GHinv,invro
    ,W_U,W_L,Hroinv);
[z_opt,iteraciones]=ADMM_QP_ESB(q,b,ro,epsilon,vmas,vmenos,z0,nz,Bb_chol,dMz2,
    Hroinv,invro,GHinv,HinvG,V,U);
time=toc;
z0=z_opt;
K_Sim(i-inicio)=iteraciones;
u_aut=z_opt(n+1:n+m,1);
x_kk=A*xk+B*(u_aut);

X_Sim(:,i)=x_kk;
U_Sim(:,i)=u_aut;
X_ref(:,i)=xr;
U_ref(:,i)=ur;
Tiempo_Res(1,i-inicio)=time;
xk=x_kk;
end
end
fprintf("El valor de Rho es %d \n",ro)
fprintf("El valor de N es %d \n",N)
it_max=max(K_Sim)
it_min=min(K_Sim)
it_mediana=median(K_Sim)
it_medio=mean(K_Sim)

Tiempo_max=max(Tiempo_Res)
Tiempo_min=min(Tiempo_Res)
Tiempo_mediana=median(Tiempo_Res)
```

```
Tiempo_medio=mean(Tiempo_Res)

%
%
%
% K_Sim=K_Sim';
```

Índice de Figuras

Índice de Algoritmos

Índice de Tablas

Índice de Códigos

Bibliografía

[1] A. Beck and M. Teboulle, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, A fast iterative shrinkage-thresholding algorithm for linear inverse problems (2009).

[2] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al., *Distributed optimization and statistical learning via the Alternating Direction Method of Multipliers*, Foundations and Trends® in Machine learning **3** (2011), no. 1, 1–122.

[3] Stephen P Boyd and Lieven Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

[4] Eduardo Camacho and Carlos Bordons, *"Model Predictive Control"*, "Springer London", "2007".

[5] Alejandro Martín Cuevas, *Resolución de sistemas de ecuaciones en semi banda*, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2017.

[6] Pablo Krupa García, *Implementation of MPC in embedded systems using first order methods*, Ph.D. thesis, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2001.

[7] Gene H Golub and Charles F Van Loan, *Matrix computations*, JHU press, 2013.

[8] Pablo Krupa, Ignacio Alvarado, Daniel Limon, and Teodoro Alamo, *Implementation of Model Predictive Control for Tracking in Embedded Systems Using a Sparse Extended ADMM Algorithm*, IEEE Transactions on Control Systems Technology **30** (2022), no. 4, 1798–1805.

[9] Pablo Krupa, Daniel Limon, and Teodoro Alamo, *Spcies: Suite of Predictive Controllers for Industrial Embedded Systems*, *https://github.com/GepocUS/Spcies*, Dec 2020.

[10] Pablo Krupa García, José Cámara, Ignacio Alvarado Aldea, Daniel Limón Marruedo, and Teodoro Álamo Cantarero, *Real-time implementation of MPC for tracking in embedded systems: application to a two-wheeled inverted pendulum*, (2021).

[11] Yu E Nesterov, *A method for solving the convex programming problem with convergence rate* $O(1/k^2)$, Dokl. akad. nauk Sssr, vol. 269, 1983, pp. 543–547.

[12] Javier Maestro Valenzuela, *Algoritmos de primer orden para la implementación del Control Predictivo para Seguimiento*, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2021.

[13] Wikipedia, *Factorización de Cholesky — Wikipedia, La enciclopedia libre*, *https://es.wikipedia.org/w/index.php?title=Factorizaci%C3%B3n_de_Cholesky&oldid=150378597*, 2023.

[14] Wikipedia, *Factorización LU — Wikipedia, La enciclopedia libre*, *https://es.wikipedia.org/w/index.php?title=Factorizaci%C3%B3n_LU&oldid=149307705*, 2023.

[15] Wikipedia, *Weinstein–Aronszajn identity — Wikipedia, The Free Encyclopedia*, *https://en.wikipedia.org/w/index.php?title=Weinstein%E2%80%93Aronszajn_identity&oldid=1163734413*, 2023.

[16] Wikipedia, *Woodbury matrix identity — Wikipedia, The Free Encyclopedia*, *https://en.wikipedia.org/w/index.php?title=Weinstein%E2%80%93Aronszajn_identity&oldid=1163734413*, 2023.