Trabajo de Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Control Predictivo Económico Basado en Técnicas Avanzadas de Optimización

Autor: Juan Moreno Nadales

Tutores: Dr. Daniel Limón Marruedo y José María Manzano Crespo

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

Sevilla, 2018

CONTROL PREDICTIVO ECONÓMICO BASADO EN TÉCNICAS AVANZADAS DE OPTIMIZACIÓN

JUAN MORENO NADALES

Trabajo fin de Máster

Supervisado por Dr. Daniel Limón Marruedo y José María Manzano Crespo

Universidad de Sevilla

septiembre 2018

Publicado en septiembre 2018 por
Juan Moreno Nadales

jmnadales@outlook.com

Yo, Juan Moreno Nadales con NIF número 45889790Y,

DECLARO

mi autoría del trabajo que se presenta en la memoria de este trabajo fin de máster que tiene por título:

CONTROL PREDICTIVO ECONÓMICO BASADO EN TÉCNICAS AVANZADAS DE OPTIMIZACIÓN

Lo cual firmo,

Fdo. Juan Moreno Nadales
en la Universidad de Sevilla
06/09/2018

Dedicado a todas aquellas personas que viven para investigar, y sobre todo, a las que investigan para que otras vivan.

Nankurunaisa

Agradecimientos

A todos aquellos que me han apoyado a lo largo del desarrollo de este proyecto, familiares, amigos, compañeros, profesores y tutores.

Resumen

Las estrategias de control predictivo supusieron un revulsivo en la industria de procesos, especialmente en industrias del sector petroquímico, donde gracias a los estudios realizados por la academia en este campo fueron capaces de incrementar enormemente sus beneficios gracias a la buena actuación de dichos controladores. La principal razón del gran éxito de este tipo de controladores fue su capacidad de poder trabajar con restricciones, siendo así la única estrategia de control avanzado que ha tenido éxito en esta cuestión.

Sin embargo, a pesar de su buen funcionamiento, son aún numerosas las limítaciones que el diseñador del sistema de control ha de afrontar. Dicha limitaciones vienen dadas entre otros motivos, por la dificultad que se presenta a la hora de llevar a cabo el proceso de optimización involucrado en cualquier estrategia de control predictivo. Existen numerosos motivos que hacen que el el problema de optimización sea difícil de resolver, como pueden ser la no convexidad de la función a optimizar, la presencia de no linealidades en el modelo o en las restricciones, etc.

Para solventar esta problemática, y a pesar de que aún no se encuentra muy extendido en la industria debido a la filosofía de "Si algo funciona no se toca", han surgido en los últimos tiempos numerosas herramientas específicas para llevar a cabo complejos procesos de optimización. De esta forma, es posible llevar a cabo la resolución de problemas de optimización arbitrariamente complejos. De está forma, es posible en la actualidad llevar a cabo el diseño de controladores predictivos no lineales, evitándose así el proceso de linealización que se ha venido llevando a cabo tradicionalmente de manera necesaria.

El objetivo principal del presente trabajo de fin de máster es el de presentar nuevas herramientas y metodologías para el desarrollo de controladores predictivos, así como

que el trabajo llevado a cabo sirva de iniciación a la investigación en el ámbito del control predictivo de cara a una futura tesis doctoral. En concreto, se tiene la mirada puesta en la utilización de los conceptos presentados en este trabajo para su aplicación al área del control predictivo económico, línea de investigación la cual se encuentra actualmente en auge, y de la que se espera se convierta en un paradigma extendido en la industria en años venideros, debido al gran impacto económico que esta puede tener.

Dicho esto, se lleva a cabo un pequeño resumen a modo de guía para el lector, en el que se muestran cuales serán los contenido tratados en el presente trabajo.

Como previamente se ha comentado, uno de los objetivos a llevar a cabo en el presente trabajo es que sirve como introducción a la investigación en el campo del control predictivo no lineal. Por esta razón, en el **capítulo 1** se lleva a cabo un pequeño resumen general de los conceptos básicos de control predictivo no lineal. Primeramente, se resumen los aspectos generales de esta metodología de control, presentando el problema a resolver así como presentando las distintas razones que justifican su utilización frente a otras metodologías de control. Una vez que se ha llevado a cabo una pequeña introducción al control predictivo no lineal, se lleva a cabo un pequeño resumen del estudio de la estabilidad y robustez de dichos controladores. Primeramente, se estudian los conceptos básicos de estabilidad según la teoría de Lyapunov en sistemas no lineales, tanto autónomos como no autónomos. En segundo lugar, una vez comprendidos dichos conceptos, se lleva a cabo su aplicación particularizando a sistemas no autónomos cuya ley de control es obtenida mediante la aplicación de una estrategia de control predictivo.

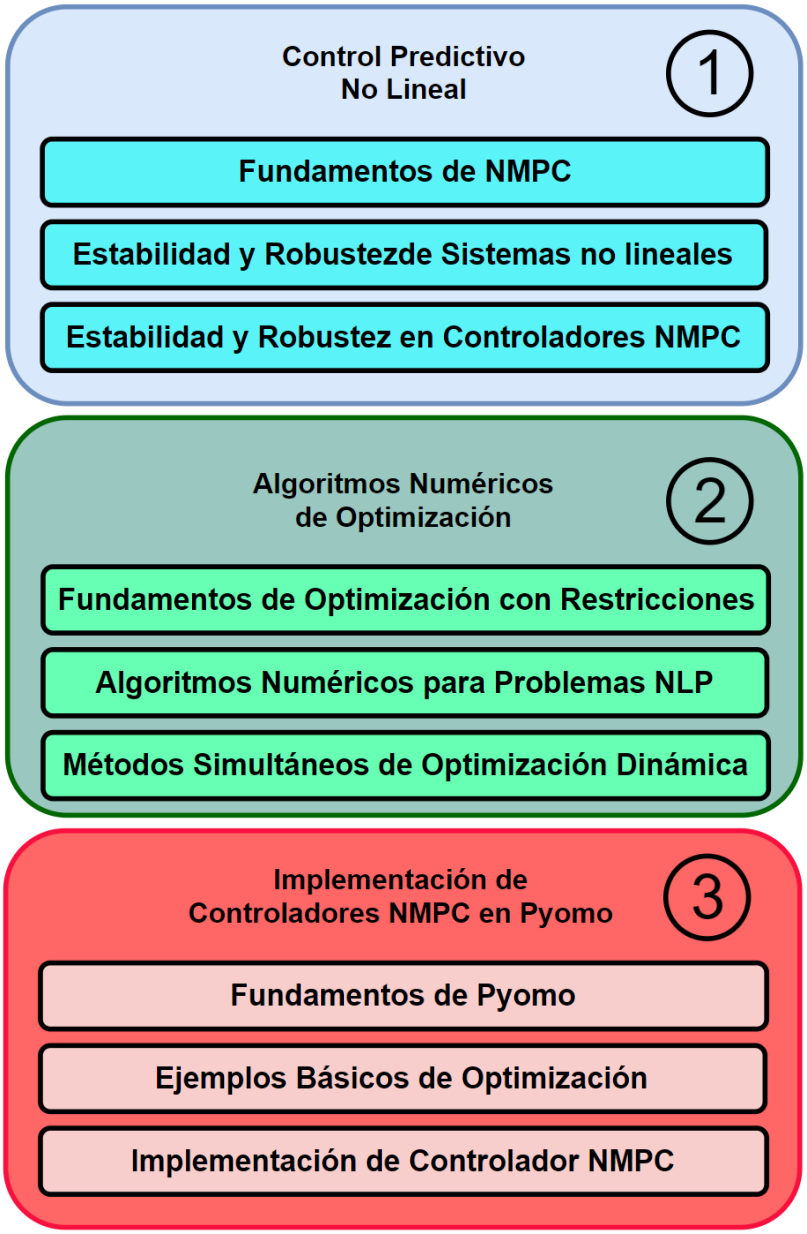
Una vez que se han estudiado los conceptos generales acerca del control predictivo, su estabilidad, y su robustez, en el **capítulo 2** se presentan algunos de los algoritmos matemáticos y métodos numéricos avanzados para la resolución del problema de optimización. Primeramente, y suponiendo que el lector ya cuenta con unos conocimientos básicos de optimización de problemas no sujetos a restricciones, se lleva a cabo un resumen y análisis de los problemas de optimización sujetos a restricciones. Hecho esto, se lleva a cabo el análisis de algunos algoritmos numéricos utilizados para la resolución de estos problemas. Frente a los tradicionales algoritmos de optimización basados

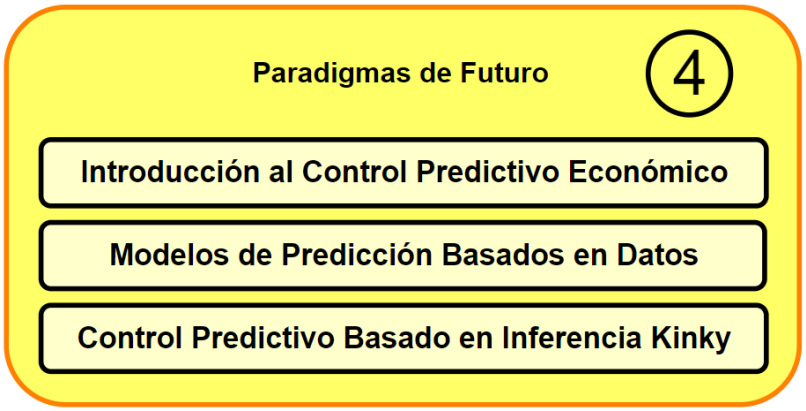
en métodos de Newton, se presentan una serie de algoritmos y métodos pertenecientes al campo de la programación matemática no líneal debido a su gran utilización por la mayoría de los solvers matemáticos de optimización de problemas no lineales. Por último, es este capítulo se presentan una serie de nuevos métodos de optimización dinámica. Dichos métodos, conocidos como métodos de colocación, pueden ser utilizados, tal y como ser verá posteriormente, como métodos de discretización y predicción de trayectorias del solver de optimización.

Presentada ya toda la teoría matemática necesaria para la comprensión de los controladores predictivos propuestos en el presente trabajo, se lleva a cabo la presentación de Pyomo, una herramienta mediante la cual será posible implementar controladores predictivos no lineales en tiempo continuo. Primeramente, se comenzará analizando los aspectos básicos de la herramienta, realizando una pequeña guía a modo de manual de usuario donde se presentan los conocimientos necesarios para su utilización así como para la construcción de los modelos de los sistemas que se desean controlar. Seguidamente, en aras de poder ilustrar todo lo analizado anteriormente, se muestran algunos ejemplos de utilización de la herramienta en procesos de optimización de diversa naturaleza. Por último, se presentan ya los controladores predictivos de sistemas no lineales propuestos en el presente trabajo. Dichos controladores predictivos serán de carácter completamente no lineal, tendrán como modelo de predicción las ecuaciones dinámicas que definen el comportamiento del sistema introducidas como restricciones del problema, y serán llevado a cabo completamente en tiempo continuo, de tal forma que no será necesario llevar a cabo el muestreo del sistema.

Al final del proyecto, en el **capítulo 5** una vez que ya se ha cumplido el objetivo de utilizar la herramienta de programación propuesta para la implementación de controladores predictivos, se ha incluido un apartado en el cual se realiza una introducción a dos futuras líneas de investigación y ampliación del trabajo ya realizado. Por un lado, se lleva a cabo en el **capítulo 4** una introducción al concepto del control predictivo económico, objetivo futuro del uso de las herramientas presentadas. Por otro lado, se lleva a cabo en el **capítulo 5** una introducción al control predictivo basado en datos. Para ello, se muestran algunas de las principales metodologías utilizadas en el proceso de construcción de modelos a partir de datos históricos obtenidos mediante ensayos de la planta. Especialmente, se hará hincapié en el uso de la técnica conocida como inferencia kinky. Basándonos en esta técnica, y en aras de mostrar un ejemplo de desa-

rrollo de control predictivo basado en datos, se ha llevado a cabo la implementación del control de un sistema aplicando dichas técnica. Dicha implementación, y siguiendo los desarrollos llevado a cabo previamente a lo largo del trabajo, se ha realizado en Python, actual lenguaje de programación más extendido entre la comunidad mundial de programadores, y del cual se espera que se convierta en la principal herramienta en los próximos años en la comunidad de *machine learning*. Se muestra a continuación un esquema visual de la distribución de contenidos del presente trabajo.





Palabras Clave: *Control Predictivo, Machine-Learning, Python, Pyomo, Optimización, Estabilidad, Tobustez, Técnicas Basadas en Datos, Control Económico, Métodos de Punto Interior, Métodos de Colocación, Inferencia Kinky*

Índice general

Índice de figuras

Índice de Tablas

Parte I

Marco Teórico

Control Predictivo No Lineal

1.1 Introducción

Como previamente se ha comentado, el objetivo principal de este trabajo es analizar el uso de nuevas herramientas que nos permitan de una manera sencilla llevar a cabo la implementación de controladores predictivos no lineales, sujetos a restricciones, y que sean estables por diseño. Para ello, es necesario previamente tener unas nociones básicas acerca del problema de diseño de controladores predictivos no lineales, entender los procedimientos de diseño aplicados en la actualidad para abordar este problema, y los distintos desafíos a los cuales se ha enfrentado el diseñador de este tipo de controladores.

A lo largo de este capítulo se llevará a cabo de manera resumida una introducción teórica al control NMPC (Non-Linear Model Predictive Control), que si bien no se encuentra actualmente extendido en la industria debido a los problemas que hasta la fecha ha presentado su diseño e implementación, muchos de los cuales se intentarán abordar en el desarrollo del presente trabajo, está pensado que sea un revulsivo en los próximos años gracias a la aparición de nuevas herramientas que permiten de una manera sencilla abordar dicho problema, como la que será presentada en este trabajo.

El control predictivo no lineal tiene tiene su origen en los controladores predictivos desarrollados tradicionalmente por la comunidad académica, y en cuyos procedimientos de diseño se lleva a cabo la optimización de una determinada función de coste, con el objetivo de minimizar el coste o índice de desempeño definido por el ingeniero encargado de llevar a cabo el diseño. La principal novedad en este tipo de controladores se basa en la utilización de modelos no lineales, tanto en el modelo de predicción como en las restricciones, a diferencia de los controladores predictivos tradicionales (MPC),

en los cuales o bien se hace uso de un modelo lineal o bien se utiliza una linealización del modelo no lineal en torno a un punto de trabajo, con todos los inconvenientes que esto conlleva y los riesgos que el diseñador ha de asumir.

En su globalidad, el objetivo principal de todo controlador predictivo, lineal o no lineal, es el de dar respuesta práctica a la problemática que presenta la utilización de controladores óptimos de horizonte infinito, los cuales, a pesar de ser la mejor alternativa en cuanto a costo y desempeño, difícilmente tienen solución, especialmente en el caso de la inclusión de restricciones. Así pues, el control predictivo gira en torno a dos ejes principales, la utilización en la optimización de un horizonte de predicción finito, y el hecho de que este horizonte sea deslizante. Así, en un determinado instante de tiempo t_k o período de muestreo k, se predecirá el comportamiento del sistema en un horizonte de predicción de N períodos de muestreo o ΔT espacio temporal. En el siguiente instante de muestreo $k+1$, o paso temporal $t_{t_k+\Delta t}$ se volverá a llevar a cabo una nueva predicción para un horizonte de control de igual duración N.

Tradicionalmente, el diseño e implementación de controladores predictivos se ha llevado a cabo, en aras de una mejor implementabilidad, en tiempo discreto. En el presente trabajo, gracias a las posibilidades que nos ofrecen las nuevas herramientas de diseño, se hará uso de modelos en tiempo continuo. Para que la solución obtenida sea implementable, una vez que se ha llevado a cabo la optimización de la función objetivo y se ha encontrado la ley de control a ser utilizada, será necesario llevar a cabo un proceso de discretización que nos permita encontrar un valor discreto de las variables de control para ser aplicadas al sistema.

Las expresiones que serán utilizadas en este capítulo hacen referencia tanto al caso continuo como el caso discreto, así la expresión $f(x)$ hace referencia de igual manera al caso continuo $f(x(t))$ y al caso discreto $f(x(k))$. En este capítulo utilizaremos modelos de predicción de la forma:

$$x^+ = f(x,u) \tag{1.1}$$

Donde x^+ es el estado del sistema en el instante de tiempo o período de muestreo siguiente, y $f(x,u)$ es una función no lineal que depende del estado actual del sistema (en principio perfectamente conocido) y de la actuación o señal de control aplicada en el instante actual.

Al igual que en el resto de las metodologías de control, se presentan dos variantes distintas: el problema de regulación, en el cual simplemente se conduce el sistema hasta un punto de equilibrio del mismo, y el problema de seguimiento de referencias o *trackcing*, en el cual deseamos que los estados del sistema evolucionen siguiendo una determinada trayectoria a lo largo del tiempo. En cualquiera de estos problemas, el objetivo será encontrar una serie de acciones de control $[u_0, u_1,..., u_N]$ admisibles tal que el estado del sistema tenga el comportamiento deseado, se cumplan todas las restricciones del sistema, se garantice la estabilidad del mismo, y se acepten unos determinados márgenes de robustez, donde u_n es la acción a aplicar en el instante de tiempo t_n o el período de muestreo k_n.

Estas acciones de control aplicadas al sistema, serán, por lo general, las primeras acciones de control calculadas en un determinado instante de tiempo para todo el horizonte de predicción. De esta forma, si durante una cierta evaluación del problema de optimización con horizonte de predicción igual a cinco se calcula una serie de acciones de control de la forma $[u(2 \mid 1),...,u(6 \mid 1)]$ (donde $u(n,n_0)$ es la acción de control en el instante n calculada en el instante actual n_0), sólo será aplicada un número determinado de las primeras acciones de control, por lo general, la primera de ellas.

Estas acciones de control calculadas en cada instante de tiempo serán los valores de la señal de control que minimicen cierta función de coste definida por el usuario. Tradicionalmente, a la hora de llevar a cabo la minimización de la función de coste $J(x,u)$, se ha tomado como variable de optimización sólo la señal de control, de tal forma que el problema de optimización ha tenido la forma:

$$u = arg \min_{u} J(x,u) \tag{1.2}$$

Sin embargo, y debido a las posibilidades que nos brindan las nuevas herramientas de optimización, como la presentada en este trabajo, el proceso de optimización será llevado a cabo con respecto a todas y cada una de las variables del sistema, incluyendo las variables de control y las variables que definen el estado, de tal forma que la señal de control calculada será obtenida como:

$$u = arg \min_{u,x} J(x,u) \tag{1.3}$$

A lo largo del desarrollo de este capítulo hablaremos largo y tendido de cual debe ser la correcta elección de la función J, aspecto crítico y clave, y que junto con la sintonía y elección de los parámetros del controlador serán las dos labores principales del ingeniero encargado de realizar el diseño, diferenciando así un buen diseño de uno deficiente o erróneo. Se tratarán pues, además de las soluciones adoptadas tradicionalmente, las nuevas alternativas que han surgido en los últimos tiempos para garantizar la estabilidad y robustez del controlador ya durante el proceso de diseño.

1.2 Fundamentos de Control NMPC

En esta sección se asentarán las bases del control predictivo no lineal, desarrollando de manera más extensiva los conceptos presentados de manera resumida en el apartado anterior. Como ya se ha explicado, el control predictivo no lineal se basa en la optimización de una determinada función de coste, en cada uno de los instantes de muestreo (en caso discreto) o instantes de tiempo (en caso continuo), y a lo largo de un determinado horizonte de predicción. Del conjunto de acciones de control calculadas en cada uno de los procesos de optimización para todo el horizonte, en el siguiente paso sólo será aplicada la primera (o primeras) acción de control obtenida, repitiéndose este proceso durante todo el intervalo de tiempo en el cual se desea que el controlador esté funcionando.

Antes de seguir con el desarrollo, es importante considerar una serie de suposiciones, que si bien serán justificadas más adelante, son conceptualmente de vital importancia, a saber:

- El desarrollo será inicialmente llevado a cabo para resolver el problema de regulación, en el que se desea que el estado del sistema converja hacia un único estado de refencia x_{ref}.
- El punto de referencia elegido hacia el cual el sistema ha de converger ha de ser un punto de equilibrio del sistema (x_0,u_0) de tal forma que el estado permanezca en dicho punto de equilibrio una vez que este sea alcanzado. Esto es de vital importancia para poder poder encontrar una ley de control que estabilice el sistema en lazo cerrado. El problema de estabilidad será tratado ampliamente en apartados posteriores.
- Ha de existir una ley de control $u(x)$, perteneciente al conjunto de valores factibles U, de tal forma que el problema pueda llevarse a cabo.

Teniendo en cuenta esto, el primer paso a llevar a cabo para el diseño del controlador es el de elegir una adecuada función de coste. No cualquier tipo de coste nos permitirá cumplir con los requerimientos preestablecidos, y como se verá más adelante, las soluciones adaptadas tradicionalmente en los controladores predictivos convencionales no pueden cumplir con los requerimientos de estabilidad necesarios para dotar a nuestro sistema de unos determinados niveles de garantía.

En una elección de función de coste tradicional son dos los elementos básicos considerados. El primero de ellos, es la desviación del estado en el instante actual con respecto al valor del estado de referencia, es decir, se desea que durante el proceso de optimización se penalice el error o distancia que el sistema aún debe de recorrer hasta poder alcanzar el estado deseado. El segundo de los elementos básicos a incluir en nuestra función de coste es un término que penalice la desviación de la acción de control con respeto a su referencia. Con este objetivo, son varias las alternativas que han sido utilizadas en la literatura, siendo las más habituales aquellas que penalizan la desviación de la señal de control entre el instante inicial y el instante anterior, y aquella que penaliza la desviación de la señal de control en el estado actual y el valor que

tomaría la señal de control, al fin y al cabo una variable más, cuando el sistema haya alcanzado su estado de referencia. De cualquiera de las maneras, la penalización de la acción de control es muy recomendable debido a que la mayoría de los problemas de optimización son más fácilmente resolubles computacionalmente si la variable de control es penalizada [1]. Además, la penalización de la señal de control nos permitirá poder evitar cambios bruscos de la misma, siendo esto de vital importancia en algunos de problemas de control.

Con todo lo comentado anteriormente, se desea diseñar una ley de control tal que el coste cuando el sistema haya alcanzado el estado de referencia sea cero o lo más cercano a cero, teniéndose que:

$$J(x_0, u_0) = 0 \tag{1.4}$$

Donde x_0 hace referencia al estado en el punto de equilibro, u_0 la señal de control cuando el sistema ha alcanzado dicho estado, y $\ell(x_0, u_0)$ el coste en dicho punto, siendo éste definido positivo para cualquier valor del estado $x \in X$ y señal de control $u \in U$. Siempre que estemos trabajando en un espacio Euclídeo será posible considerar nuestro punto de referencia o punto de equilibro como el origen del sistema, de tal forma que $x_0 = 0$ y $u_0 = 0$ sin que se produzca una pérdida de generalidad. Lo dicho anteriormente simplemente se traduce en llevar a cabo una traslación de los ejes o sistema de referencia para forzar que nuestro punto de referencia coincida con el origen.

Una vez que se se han incluido en nuestra función de coste términos que penalicen tanto el estado del sistema como la señal de control con respecto a sus puntos de referencia, es necesario elegir de qué manera serán penalizados. Aunque en la literatura se pueden encontrar diversas alternativas en lo que respecta a esta elección, la más utilizada con diferencia es la función de coste como función cuadrática de sendas normas Euclídeas del estado y de la variable de actuación, de la forma:

$$J(x, u) = \|x\|_Q^2 + \|u\|_R^2 \tag{1.5}$$

Donde x hace referencia a la diferencia entre el estado actual y el estado de referencia, u la diferencia entre la señal de control actual y la señal de control en el estado de referencia, y Q y R son matrices de ponderación que nos permitirán llevar a cabo, junto con la elección del horizonte de predicción, la sintonía del controlador. Por lo general, grandes valores de Q con respecto a R indicarán las intención del diseñador de conducir nuestro estado rápidamente al punto de equilibrio a consta de grandes valores y cambios de la señal de control. Por el contrario, valores grandes de R con respecto a Q indicarán la intención del diseñador de suavizar la trayectoria de la señal de control a consta de aumentar el tiempo que el sistema tardará en alcanzar el punto de equilibrio [2].

Con esta descripción, el problema general a resolver tomará la forma:

$$arg \min_{x,u} J_N = arg \min_{x,u} \sum_{k=k_0}^{N-1} \|x\|_Q^2 + \|u\|_R^2 \tag{1.6}$$

para su versión en tiempo discreto, o en tiempo continuo:

$$arg \min_{x,u} J_N = arg \min_{x,u} \int_{t=t_0}^{t=t_0+N} \|x\|_Q^2 + \|u\|_R^2 \, dt \tag{1.7}$$

Las expresiones anteriores hacen referencia al problema de encontrar un ley de control factible, de un sistema por lo general no lineal, a lo largo de un horizonte de predicción finito igual a N, partiendo de un punto $k = k_0$ en el caso discreto o $t = t_0$ en el caso continuo, y con una función de coste cuadrática que penaliza las normas ponderadas de las desviaciones del estado y de la señal de control con respecto a su valor de referencia. Esta optimización será realizada para todos y cada uno de los pasos del sistema, aplicándose sólo y exclusivamente el primer valor de la trayectoria futura de la señal de control predicha. Partiendo de aquí, los siguientes pasos a dar son el de encontrar alguna manera de introducir en el problema las ecuaciones dinámicas que definen el mismo, y el de restringir los valores del estado y de la señal de control que se consideren aceptables.

Una vez pues que se ha definido la función de coste (por el momento) a ser utilizada por nuestro controlador NMPC, es momento de introducir las restricciones del sistema. La forma en la que los controlador NMPC desarrollados en el presente trabajo tratan con las restricciones difiere levemente con respecto a la forma en la que tradicionalmente se han tratado las restricciones en los controladores MPC convencionales, no tanto conceptualmente, pero sí en cuanto a la forma en la que estas son definidas, y sobre todo, en qué entendemos por restricciones.

En el diseño de controladores MPC tradicionales, la forma en la que se ha llevado a cabo la inclusión de las ecuaciones dinámicas que definen el sistema en el proceso de optimización ha sido mediante la sustitución del valor de salida predicho según el modelo en la función de costes a ser optimizada. A partir de aquí, se llevaba a cabo la optimización de la función de coste y se obtenían los valores de la trayectoria predicha de la variable de control, la cual ya estaba condicionada por las ecuaciones dinámicas del sistema. Esto era así debido a que no suponía ninguna dificultad el llevar a cabo la optimización de la función de coste tras haber incluido las ecuaciones del sistema, al tratarse de ecuaciones lineales fácilmente derivables. En los controladores NMPC presentados en este trabajo, las ecuaciones dinámicas que definen el sistema pueden ser de una complejidad arbitrariamente grande: sistemas de ecuaciones diferenciales con un gran número de variables de estado y de control, y sus respectivas derivadas.

Por ello, en los controladores NMPC como los tratados en el presente trabajo, las ecuaciones del sistema serán introducidas en la manera en la que lo requieren los solvers matemáticos utilizados para la resolución del problema de optimización, los cuales serán descritos en capítulos posteriores, que no es nada más y nada menos que en forma de restricciones del sistema. Al fin y al cabo, cuando en un problema de desarrollo de las expresiones que definían un controlador MPC tradicional se sustituía la salida predicha del sistema en la función de coste a optimizar, se estaba en realidad restringiendo el sistema de forma explícita para que cumpliese con el comportamiento dictado por las ecuaciones dinámicas del sistema. Por lo tanto, los procedimientos llevados a cabo en los controladores NMPC sólo introducen como novedad la forma en la que dichas restricciones son incluidas en el problema de optimización para ser adaptadas a los requerimientos de los solvers utilizados.

Por lo general, en un sistema dinámico bien condicionado, el número de ecuaciones dinámicas que definirán el comportamiento del sistema, y que serán tratadas como restricciones, será igual al número de variables de estado, ya que es necesario conocer la variación en el tiempo de cada una de ellas. Es importante tener en cuenta que se han de tener al menos tantas variables de estado como restricciones dinámicas tenga el sistema, ya que de otra forma el solver no podrá tratar con ellas, algo de lo cual hablaremos con más detenimiento en capítulos posteriores. De esta forma, en un sistema que cuente con un total de n variables de estado, las restricciones en forma de ecuaciones dinámicas del sistema pueden venir dadas en la forma de cualquier ecuación diferencial de orden arbitrario. Téngase en cuenta que en este conjunto no se suelen incluir ecuaciones que definan la variación de las variables de control, ya que esta ha de venir definida por la ley de control y no por el sistema en sí. De esta forma, las restricciones que definan el comportamiento del sistema serán generalmente de la forma:

$$\begin{aligned}
\frac{d^n x_1}{dx_1^n} &= f(x_1, u_1, x_1', u_1', \ldots, x_n, u_n, x_n', u_n') \\
\frac{d^n x_2}{dx_2^n} &= f(x_1, u_1, x_1', u_1', \ldots, x_n, u_n, x_n', u_n') \\
&\vdots \\
\frac{d^n x_i}{dx_i^n} &= f(x_1, u_1, x_1', u_1', \ldots, x_n, u_n, x_n', u_n')
\end{aligned}$$

El segundo tipo de restricciones que se han de incluir en el problema de optimización son aquellas que restrinjan el conjunto de valores que pueden tomar nuestras variables, tanto de estado como de control. Se incluyen aquí los conjuntos de valores del sistema tales que, si una determinada variable de estado o variable de control saliere de dichos conjuntos, el comportamiento del sistema sería no aceptable. La salida de cualquiera de las variable de su conjunto aceptable podría significar o bien un fallo grave del sistema, lo cual podría suponer o bien un riesgo o bien cualquier clase de perjuicio económico. A continuación se detallan una serie de conceptos imprescindibles a la hora de tratar con este tipo de restricciones.

Definición 1.
Se dice que un determinado conjunto de variables de estado es admisible, si para la totalidad del desarrollo del sistema, el valor de todos y cada uno de los estados se encuentra dentro de un determinado conjunto X aceptable definido por el diseñador. Si a lo largo del desarrollo del sistema alguna de las variables sale fuera de dicho conjunto, el comportamiento será considerado no admisible.

Definición 2.
Se dice que un determinado conjunto de variables de control es admisible, si para la totalidad del desarrollo del sistema, el valor de toda y cada una de las variables de control se encuentra dentro de un determinado conjunto U definido por el diseñador. Si alguno de los valores que tome la variable de control sale fuera de dicho conjunto, el sistema sería considerado no admisible.

Es decir, a la hora de llevar a cabo la resolución del problema de optimización para el cálculo de la ley de control, se ha de tener en cuenta que no cualquier valor de las variables de estado y de control serán consideradas admisibles, lo cual afectará en gran medida la trayectoria que el sistema seguirá desde el punto de partida hasta el punto de equilibrio o referencia establecido. Así pues, en todas y cada una de las iteraciones que sean llevadas a cabo para calcular la trayectoria óptima en cada paso o instante de tiempo considerado, el conjunto de todos los valores de la trayectoria ha de pertenecer al conjunto definido por la restricción. Eso implica, que para todos y cada uno de los valores iniciales del estado pertenecientes al conjunto de restricciones $x_{init} \in X$ se ha de encontrar un conjunto de acciones de control restringidas por $u \in U$ tal que el estado siga perteneciendo durante todo su desarrollo al conjunto de restricciones $x \in X$.

En resumen, el problema de optimización ha de tener como objetivo que el sistema pueda evolucionar hasta su punto de equilibro, en función de las ecuaciones dinámicas que definen el sistema, y encontrando una ley de control $\psi(x) \in U$ tal que $f(x, \psi(x) \in X)$. En esta serie de conjuntos de restricciones sólo y exclusivamente se ha tenido en cuenta la voluntad del diseñador en cuanto a los valores que desea que las variables de estado y variables de control no tomen. Como se verá más adelante, este problema no es para nada trivial, y escoger adecuadamente el conjunto de valores tomados como restricción supone aún un desafío a la hora de describir un problema de optimización. De cualquier forma, y teniendo única y exclusivamente en cuenta las premisas descritas hasta el momento, nuestro problema de optimización toma la siguiente forma, para el caso continuo:

$$arg \min_{x,u} J_N = arg \min_{x,u} \int_{t=t_0}^{t=t_0+N} \|x(t)\|_Q^2 + \|u(t)\|_R^2 \, dt$$

$$s.a$$

$$\frac{d^n}{dx_i^n} = f(x_1, u_1, x_1', u_1', ..., x_n, u_n, x_n', u_n') \; for \; i \; in \; m$$

$$x \in X$$

$$u \in U$$

donde n es el número de variables de estado del sistema, x el conjunto de estados del sistema, y u el conjunto de las variables de control, restringidos por las regiones X y U respectivamente.

1.3 Control Óptimo de Horizonte Infinito

En esta sección se realizará una breve revisión del control óptimo de horizonte infinito, lo cual será de vital importancia para llevar a cabo a cabo el cálculo del coste terminal de controladores predictivos no lineales tal y como se verá a la hora de realizar la implementación práctica. Tal y como se demostrará en la sección dedicada al análisis de estabilidad, el control predictivo de horizonte infinito es capaz bajo ciertas circunstancias de llevar a cabo la estabilización asintótica del sistema conduciéndolo a su punto de equilibrio. Además, la función de coste óptima de horizonte infinito se trata de una función de Lyapunov del sistema en lazo cerrado [1]. Estas buenas propiedades son aprovechadas para llevar a cabo el desarrollo de los controladores predictivos no lineales con estabilidad garantizada por diseño, conduciéndonos así al desarrollo de los conocidos como controladores subóptimos.

Llevar a cabo la resolución del problema de control óptimo no es nada más y nada menos que resolver el problema de optimización de una función de coste tal y como la descrita en (2.7), con la salvedad de que el horizonte de predicción para el cual se llevaba a cabo el problema de optimización toma ahora un valor infinito, en contrapartida con el valor finito N tomado anteriormente. De esta forma, la función de coste del problema de optimización tomaría la forma, para el caso de estar trabajando en tiempo

continuo:

$$arg \min_{x,u} J_\infty = arg \min_{x,u} \int_{t=t_0}^{t=\infty} \|x\|_Q^2 + \|u\|_R^2 \, dt \tag{1.8}$$

La pregunta que se deriva es, ¿por qué no se utiliza la formulación de control óptimo en casos prácticos si se ha podido demostrar su optimalidad así como la garantía de estabilización del sistema? La respuesta a esta pregunta, tal y como se verá a continuación, radica en la enorme complejidad de llevar a cabo la resolución del problema. A la hora de plantear este problema, son dos las posibles alternativas. La primera de ellas es la aplicación de las ecuaciones de *Euler-Lagrange*. La gran problemática de la aplicación de este método es que la solución encontrada conduce a soluciones locales y en bucle abierto [3], siendo deseada la obtención por el contrario de soluciones globales en bucle cerrado. La otra alternativa, la cual consigue solventar este problema, es la resolución de las ecuaciones de *Hamilton-Jacobi-Bellman*. El mayor inconveniente de este método es la enorme dificultad de ser llevado a cabo, resultando casi imposible encontrar una solución del problema en la mayoría de los casos.

Este problema sólo es resoluble bajo ciertas condiciones muy restrictivas, como es el caso en el que la función de coste utilizada es de carácter cuadrático, el sistema lineal o en su defecto linealizado no se encuentra sometido a ningún tipo de restricciones, y se desea resolver el problema de regulación del sistema al punto de equilibrio, el cual se plantea a continuación, y que es conocido como LQR (Linear Quadratic Regulator), cuya solución será utilizada más adelante para el planteamiento de controladores NMPC. Supóngase un sistema lineal de la forma:

$$x^+ = Ax + Bu \tag{1.9}$$

Donde x es el estado del sistema, y u la señal de control del mismo. A continuación, supóngase un de coste cuadrático de la forma:

$$\ell(x,u) = \|x\|_Q^2 + \|u\|_R^2 \tag{1.10}$$

A esta función de coste utilizada hasta el momento se le agregará un nuevo término, conocido como coste terminal, y el cual, tal y como se verá en secciones posteriores, será de vital importancia para poder garantizar la estabilidad del controlador predictivo en bucle cerrado, y que tiene forma también cuadrática:

$$\ell_f(x) = \|x_f\|_P^2 \tag{1.11}$$

Donde x_f hace referencia a la desviación con respecto al punto de equilibrio o referencia del valor del estado predicho al final del horizonte. Por lo tanto, la función de coste del sistema vendrá dada por la expresión:

$$arg \min_{x,u} J_\infty = arg \min_{x,u} \int_{t=t_0}^{t=\infty} \|x\|_Q^2 + \|u\|_R^2 \, dt + \|x_f\|_{Q_f}^2 \tag{1.12}$$

A continuación, se aplica la ecuación de *Hamilton-Jacobi-Bellman*, la cual tiene la forma:

$$0 = \min_u H = \min_u \; x^T Q x + u^T R u + \frac{\partial V}{\partial x}(Ax + Bu) \tag{1.13}$$

Para encontrar el mínimo de esta función, derivamos con respecto a la señal de control u, teniéndose:

$$\frac{\partial H}{\partial u} = 2u^T R + \frac{\partial V}{\partial x} B \tag{1.14}$$

Igualando a cero se obtiene un valor de la variable de control que minimiza el problema igual a:

$$u^* = -\frac{1}{2}R^{-1}B^T\frac{\partial V}{\partial x} \tag{1.15}$$

Sustituyendo este valor óptimo de la variable de control en la función Hamiltoniana se obtiene una función Hamiltoniana óptima:

$$H^* = x^TQx + \frac{\partial V}{\partial x}Ax - \frac{1}{4}\frac{\partial V^T}{\partial x}BR^{-1}B^T\frac{\partial V}{\partial x} \tag{1.16}$$

Igualando a cero esta expresión y tomando el valor integrable V como:

$$V(x,t) = x^TSx \tag{1.17}$$

Se obtiene una ecuación de la forma:

$$x^T\frac{\partial S}{\partial t}x + 2x^TSAx - x^TBR^{-1}B^TSx + x^TQx = 0 \tag{1.18}$$

Reorganizando términos en esta expresión:

$$x^T\left(\frac{\partial S}{\partial t} + SA + A^TS - BR^{-1}B^TS + Q\right)x = 0 \tag{1.19}$$

De donde se extrae que necesariamente:

$$\frac{\partial S}{\partial t} + SA + A^T S - BR^{-1}B^T S + Q = 0 \tag{1.20}$$

Expresión conocida como ecuación de Ricatti, a partir de la cual se obtiene, sustituyendo en (1.15), la ley de control LQR que viene dada por la expresión:

$$u^* = -\frac{1}{2}R^{-1}B^T Sx = K_{LQR}x \tag{1.21}$$

Como puede observarse, la ley de control obtenida consiste en una simple realimentación del vector de estado a través de una determinada ganancia K_{LQR}. Este procedimiento de cálculo se ha podido llevar cabo ya que en el problema no se han incluido ningún tipo de restricciones. En el caso de que alguna de las variables hubiese estado acotada, o se hubiese incluido cualquier tipo de restricción dinámica de alguna variable, el problema difícilmente habría tenido solución. Es precisamente aquí donde entran en juego los controladores predictivos MPC. Podría decirse que el control predictivo surge como una alternativa factible del controlador óptimo de horizonte infinito, teniendo como precio a pagar la optimalidad de este tipo de controlador así como la pérdida de garantía de estabilidad conseguida con la convergencia de la función de coste infinito al punto de equilibrio. El principal reto a la hora de llevar a cabo la implementación del controlador predictivo es el de asimilarse todo lo posible al comportamiento óptimo de un controlador con función de coste de horizonte infinito y encontrar alguna estrategia alternativa para recuperar la estabilidad garantizada por diseño. En la siguiente sección introduciremos la teoría de Lyapunov para el análisis de la estabilidad en controladores predictivo no lineales, lo cual dará soporte teórico a las estrategias de diseño utilizadas para llevar a cabo la realización de controladores estables por diseño.

1.4 Estabilidad de Lyapunov en Sistemas No Lineales

En esta sección se lleva a cabo una pequeña introducción a la teoría de estabilidad de Lyapunov para su aplicación a sistemas no lineales, y que es completamente independiente de la metodología de control utilizada. Por ello, todos y cada uno de los conceptos desarrollados aquí pueden ser aplicados a cualquier sistema, lineal o no lineal, y el cual no tiene por qué estar controlado por una ley de control predictivo. Una vez que se hayan entendido los conceptos fundamentales de la teoría de estabilidad de Lyapunov, es fácil llevar a cabo la extrapolación de estos conceptos para el cálculo de leyes de control o funciones de control de Lyapunov que garanticen estabilidad. Primeramente, comencemos suponiendo un sistema autónomo, es decir, en el cual no existe presencia de ninguna variable de control sobre el mismo, y cuyo comportamiento viene definido por la expresión:

$$x^+ = f(x_{init}) \tag{1.22}$$

donde x^+ hace referencia al estado en el instante siguiente, y $f(x_{init})$ es una función que depende del estado inicial y que define la evolución del sistema hacia el siguiente estado. Nuevamente, téngase en consideración que no se está realizando ningún tipo de distinción entre sistemas discretos y sistemas continuos, por lo cual la ley anterior podría hacer referencia a la evolución del estado desde un instante de muestreo al siguiente punto de muestreo, o la evolución en un determinado incremento de tiempo de un sistema continuo. La primera definición importante es la de punto de equilibrio del sistema la cual se presenta a continuación.

Definición 3.
Se dice que un estado o punto del sistema es un estado de equilibrio, si el sistema permanece en el mismo punto una vez que ha conseguido llegar a él. Es decir, el siguiente punto de evolución del sistema, ya sea en tiempo continuo o como siguiente período de muestro de un sistema discreto, es el punto de partida o punto de origen en el instante o período anterior, lo cual puede ser expresado de la forma:

$$x_{equilibrium} = f(x_{equilibrium}) \tag{1.23}$$

Este punto de equilibrio no tiene por qué coincidir con el origen del sistema de referencia del espacio de las variables de estado. Sin embargo, y por conveniencia, se realizará la traslación del eje de referencia hasta hacerlo coincidir con el punto de equilibrio, de tal forma que :

$$x(0) = f(0) \tag{1.24}$$

A continuación, y teniendo como referencia el trabajo elaborado en [3], se llevan a cabo una serie de definiciones básicas para el estudio de la estabilidad.

Definición 4.
Se dice que una determinada función Ψ: $\mathbb{R} \to \mathbb{R}$ es una función K, si reúne las propiedades de:

- Ser continua en todo su recorrido
- Ser estrictamente creciente, es decir, $\Psi(a) > \Psi(b)$ si $a > b$
- Pasar por el origen del sistema de referencia, es decir, $\Psi(0) = 0$

Definición 5.
Una función Ψ: $\mathbb{R} \to \mathbb{R}$ se dice que es una función K_∞ si cumple:

- Ser una función K
- Tiende asintóticamente a infinito cuando el parámetro a tiende a infinito tal que $\Psi(a) \to \infty$ cuando $a \to \infty$

Definición 6.
Una determinada función $\Phi : \mathbb{R} \to \mathbb{R}$ se dice que es una función KL si para dos parámetros de la función a y b se cumple que:

- Es una función K para todo $b > 0$ fijo
- Es una función decreciente en b para todo $a > 0$ fijo de tal forma que $\Psi(a, b) \to 0$ cuando $b \to \infty$

Definición 7.
Se dice que una determinada función $F : \mathbb{R} \to \mathbb{R}$ es una función definida positiva, si se cumple que existe una determinada función K, tal que la función Ψ de la norma del estado es menor o igual que la función F del estado, de tal forma que:

$$\Psi(\|x\|) \leqslant F(x)$$

Este conjunto de definiciones realizadas servirán como herramientas a la hora de llevar a cabo la definición de estabilidad de un sistema según la teoría de Lyapunov. A partir de este punto, se llevará a cabo un recorrido por las diferentes definiciones de estabilidad aplicables a un sistema, siguiendo nuevamente la estructura del desarrollo llevado a cabo en [3]. Para ello, se comenzará llevando a cabo una serie de definiciones relativas a la estabilidad de sistemas autónomos no sujetos a restricciones, es decir, aquellos en lo que cualquier valor del estado es considerado admisible. Seguidamente, se realizará una ampliación de las definiciones de estabilidad para incluir en ellas las posibles restricciones a las cuales pueda estar sometido el sistema, dejando así de ser aceptable o admisible cualquier valor del estado del sistema. Por último, se llevará a cabo una ampliación final de las definiciones de estabilidad para poder extrapolar todos estos conceptos a sistemas no autónomos o controlados, surgiendo así el concepto de función de Lyapunov de control o CLF (Control Lyapunov Function).

1.4.1 Estabilidad en Sistemas Autónomos

En este apartado de presentan las principales definiciones del concepto de estabilidad establecidas en [4] para sistemas autónomos, sin variable de control, así como las condiciones necesarias que ha de reunir un sistema para ser estable según estas definiciones. Nuevamente, no se hará distinción alguna entre sistemas continuos y sistemas discretos. Considérese el sistema autónomo definido por la expresión:

$$x^+ = f(x_{init}) \tag{1.25}$$

Donde $f(x_{init}, u)$ hace referencia a una expresión por lo general no lineal.

Definición 8.
Un sistema, tal y como el definido en (1.25) es estable, en un determinado punto de equilibrio, hecho coincidir con el origen del sistema de referencia, si para una región Θ en torno al punto de equilibrio existe una vecindad también en torno a dicho origen del sistema Λ tal que si el sistema parte de ella, éste evoluciona siempre acotado por la región Θ.

$$\forall x_{init} : \|x_{init}\| \leqslant \Lambda \Rightarrow \|x_{t_k}\| \leqslant \Theta \; \forall t_k > 0 \tag{1.26}$$

Es decir, el sistema nunca abandona una determinada región en torno a su punto de equilibrio una vez que este ha alcanzado una determinada zona acotada en torno a dicho punto. Esto nos garantiza que nuestro sistema siempre se va a encontrar dentro de una determinada región conocida, y nunca va a salir de ella.

Definición 9.
Un sistema, tal y como el definido en (2.25) es asintóticamente estable para un determinado punto de equilibrio si es estable, cumpliendo con la definición previa, y además la trayectoria seguida por el sistema tiende asintóticamente al propio punto de equilibrio.

$$\forall x_{init} : \|x_{init}\| \leqslant \Lambda \Rightarrow \|x_{t_k}\| \leqslant \Theta \; \forall t_k > 0, \; x_{t_k} \rightarrow x_0 \; cuando \; t_k \rightarrow \infty \tag{1.27}$$

Es decir, el sistema evoluciona acotado dentro de una determinada región del espacio y además tiende asintóticamente a su punto de referencia origen del sistema. Cabe resaltar que el hecho de que el sistema tienda al su punto de equilibrio u origen de referencia no implica que el sistema sea estable, ya que puede tender a dicho punto y a su vez salir de la región acotada en torno a la cual se define la estabilidad del sistema a lo largo de su trayectoria [3].

Definición 10.
Un sistema, tal y como el definido en (2.25) es exponencialmente estable para un punto de equilibrio del sistema, si cumple ser asintóticamente estable y además existen unas determinadas constantes $\epsilon > 0$, $\alpha > 0$, y $k \in [0,1]$ tal que:

$$\|x_{t_k}\| < \alpha \|x_{init}\| \lambda^k \tag{1.28}$$

para todo valor de x_{init} acotado por la región Λ.

Esto se traduce en la existencia de una función α tipo KL por la cual el sistema se ha de encontrar acotado [3], de tal forma que:

$$\Phi(\|x_{init}\|, k) = \alpha \|x_{init}\| \lambda^k \tag{1.29}$$

1.4.2 Teoría de Lyapunov en Sistemas Autónomos

En este apartado se llevará a cabo la introducción del concepto de *Función de Lypunov*. Según la teoría de estabilidad de Lyapunov, la estabilidad de un sistema está asociada a la existencia o no de una determinada función, conocida como función de Lyapunov, de tal forma que se podrá garantizar la estabilidad, estabilidad asintótica, y estabilidad exponencial, para un sistema dinámico en torno a un determinado punto de equilibrio, bajo ciertas hipótesis asociadas a dicha función [3].

Teorema 1.
Dado un sistema autónomo $x^+ = f(x_{init})$ cuyo origen de referencia es un punto de equilibrio del mismo, y dada una determinada función definida positiva $V(x)$, candidata función de Lyapunov, se dice que un determinado punto del sistema es estable y por lo tanto $V(x)$ función de Lyapunov asociada al mismo, si se cumple que :

- Existen dos funciones K, Ψ_1 y Ψ_2 tales que la función $V(x)$ se encuentra acotada por ellas, de la forma:

$$\Psi_1(\|x\|) \leqslant V(x) \leqslant \Psi_2(\|x\|) \tag{1.30}$$

- La función $V(x)$ es una función decreciente a lo largo de su evolución, de tal forma que:

$$\Delta V(x) = V(x_{t_{k+\Delta t}}) - V(x_{t_k}) \leqslant 0 \; \forall x \in B \tag{1.31}$$

donde x_{t_k} es el estado en el instante actual o período de muestreo actual y $x_{t_{k+\Delta t}}$ el estado en el instante siguiente o período de muestreo siguiente, y donde B es una región vecindad del origen.

Por todo ello, demostrar la estabilidad de un determinado sistema dinámico se traduce en encontrar una determinada función $V(x)$ asociada a él que cumple la condición de ser una función de Lyapunov del sistema.

Definición 11.
Un determinado conjunto perteneciente al espacio $\mathbb{R}^n$, se dice que es un conjunto de invariancia positiva o conjunto invariante positivo $\mho$, si para cualquier punto inicial del estado perteneciente a dicho conjunto x_{init}, el sistema permanece en dicho conjunto a lo largo de toda su evolución.

Es decir, si un determinado sistema, a lo largo de su evolución en el tiempo, consigue alcanzar un conjunto que cumpla las condiciones de invariancia positiva, entonces el sistema permanecerá acotado en el interior de dicho conjunto.

Por lo tanto, la estabilidad de un determinado sistema está estrechamente ligada a la existencia de un conjunto invariante positivo de tal forma que el sistema, una vez alcanzado este, evolucione acotado por dicho conjunto. Recuérdese así que la definición de estabilidad en torno a un punto de equilibrio se fundamenta en la existencia de una región acotada en torno a dicho punto tal que el sistema evolucionase acotado por él. Dicho de otra forma, se puede decir que la estabilidad de un sistema está condicionada por la existencia de un conjunto invariante positivo en torno a un punto de equilibrio, o de manera inversa, que un determinado punto del sistema será un punto de equilibrio si se encuentra acotado por un conjunto de invariancia positiva.

Todo lo comentado aquí justifica las condiciones establecidas con anterioridad para la definición de función de Lyapunov y la condición de su existecia para la garantía de la estabilidad de un determinado sistema. Una de las condiciones establecidas era la característica de la función de Lyapunov de ser decreciente. Esto implica que a lo largo de todo su desarrollo $\Delta V(x)$ es menor que cero, por lo que la trayectoria evolucionará acotada en una determinada región B. Es aquí también donde entra en juego la condición de la función de Lyapunov de estar acotada por sendas funciones K, ya que esto garantiza la existencia de una vecindad B_ϵ contenida en un conjunto invariante $\mho$ [3].

Teorema 2.
Un determinado punto del sistema autónomo no controlado se dice que es un punto asintóticamente estable del sistema si se cumple que:

- Existen sendas funciones K, Ψ_1 y Ψ_2 tales que la función $V(x)$ está acotada por ellas, de la forma:

$$\Psi_1(\|x\|) \leqslant V(x) \leqslant \Psi_2(\|x\|) \tag{1.32}$$

- Existe una determinada función K, Ψ_3, tal que $\Delta V(x)$ evoluciona a lo largo de toda su trayectoria acotada superiormente por el negativo de la función Ψ_3 de la forma:

$$\Delta V(x) \leqslant -\Psi_3(\|x\|) \tag{1.33}$$

Para todo valor de x perteneciente a la vecindad del estado B.

Si se cumplen ambas condiciones y además se extienden a todo $\mathbb{R}^n$ entonces el punto es globalmente asintóticamente estable [3].

Teorema 3.
Un determinado punto de un sistema autónomo se dice que es un punto de equilibrio exponencialmente estable si se cumple que:

- Existen unas constantes $a > 0$, $b > 0$, $\sigma > 0$ tales que :

$$a\|x\|^{\sigma} \leqslant V(x) \leqslant b\|x\|^{\sigma} \tag{1.34}$$

- Existe una determinada constate $c > 0$ tal que:

$$\Delta V(x) \leqslant -c \|x\|^{\sigma} \tag{1.35}$$

Para todo x perteneciente a la vecindad B

Antes de terminar este apartado relativo a la estabilidad de sistemas autónomos según la teoría de Lyapunov, hay que decir que si bien las definiciones anteriores son suficientes para demostrar estabilidad, bajo ciertas condiciones también son necesarias [3].

1.4.3 Estabilidad en sistemas con restricciones

Hasta el momento, todo lo dicho anteriormente hace referencia a la estabilidad de sistemas autónomos, en los cuales hemos considerado que no existe ningún tipo de restricción, es decir, se está considerando que el estado del sistema puede tomar cualquier valor arbitrariamente. Sin embargo, en la mayoría de los sistemas, autónomos y no autónomos, existen restricciones que o bien limitan el comportamiento del sistema o bien restringen los valores de las variables del mismo. En este apartado abordaremos brevemente esta segunda cuestión. Supóngase un sistema autónomo descrito por la ecuación:

$$x^{+} = f(x_{init}) \tag{1.36}$$

Donde x^{+} representa el estado del sistema en el instante de tiempo o período de meustreo siguiente y x_{init} el estado en el instante anterior. Supóngase un determinado conjunto de valores X, región en el espacio $\mathbb{R}^n$ que restringe los valores que x puede tomar a lo largo de la evolución del sistema a encontrarse incluidos en dicho conjunto, de tal forma que:

$$x_{t_k} \in X \tag{1.37}$$

Para todo valor de t_k mayor que cero.

Este tipo de restricción afecta tanto a sistemas autónomos no controlados como sistemas no autónomos en los existe ciertas variables de control, los cuales serán tratados en apartados posteriores. Por lo general, el comportamiento de dichos sistemas no autónomos vendrá dado por la expresión:

$$x^+ = f(x_{init}, u_{init}) \tag{1.38}$$

Donde u_{init} hace referencia a la ley de control que define el valor de las variables de actuación del sistema y que se puede a su vez encontrar restringida, al igual que pasaba con el estado, por un determinado conjunto U de tal forma que $u_{t_k} \in U$. Sin embargo, es posible llevar a cabo una redefinición del problema teniendo en cuenta que la ley de control ha de depender en todo momento del estado del sistema, de tal forma que :

$$u_{t_k} = h(x_{t_k}) \tag{1.39}$$

De esta forma, sustituyendo (2.39) en (2.38) la ecuación dinámica que describe la evolución del sistema en bucle cerrado puede ser expresada como:

$$x^+ = f(x_{init}, h(x_{init})) \tag{1.40}$$

sujeto a las restricciones:

$$\{x \in X : h(x) \in U\} \subseteq X \tag{1.41}$$

Por lo tanto, en el caso de sistemas autónomos, diremos que el sistema evoluciona de forma aceptable o admisible si éste evoluciona hacia una determinada región en torno a un punto de equilibrio del sistema tal que una vez alcanzada el sistema evolucione acotado con ella y cumpliendo en todo momento las restricciones impuestas. Teniendo en cuenta el concepto previamente introducido de invariancia positiva, lo dicho anteriormente puede traducirse como la necesidad de existencia de un conjunto invariante positivo $\mho$ contenido a su vez en el conjunto de restricciones impuestas, de

tal forma que:

$$x_0 \in \mho \in X \tag{1.42}$$

Por lo cual una vez que el sistema alcance dicha región su evolución será acotada por ella, teniéndose que:

$$x_{t_k} \in \mho \in X \tag{1.43}$$

1.4.4 Teoría de Lyapunov en Sistemas No Autónomos

Hasta el momento todo el análisis realizado hace referencia a la teoría de Lyapunov aplicada al análisis de la estabilidad en sistemas autónomos. Sin embargo, la realidad es que la mayoría de los sistemas encontrados en la industria se tratan de sistemas no autónomos gobernados por una determinada ley de control, la cual define el comportamiento de las variables manipuladas o variables de control, que son aquellas con las cuales se lleva a cabo una interacción directa para la conducción del sistema al estado deseado. Por ello en esta sección se intentará llevar a cabo la extrapolación de los resultados obtenidos según la teoría de Lyapunov al análisis de sistemas no autónomos. Es aquí a partir de donde surge el concepto de *función de Lyapunov de control* CLF.

Considérese un sistema cuya evolución viene definida por la ecuación dinámica:

$$x^+ = f(x_{init}, u_{init}) \tag{1.44}$$

Donde x_{init} es el estado actual del sistema, y u_{init} el conjunto de acciones de control a ser aplicado, siendo el origen del sistema de referencia un punto de equilibrio del mismo. El conjunto de estados del sistema puede estar restringido por una cierta región X de tal forma que $x_{t_k} \in X$, y el conjunto de acciones de control , definidas por una ley dependiente del estado, restringido de la forma $u_{t_k} \in U$

Dicho esto, el concepto de sistema estable definido para sistemas autónomos no controlados se transforma en el concepto de sistema estabilizable para el caso de sistemas no autónomos, de tal forma que un sistema es estabilizable si existe una determinada ley de control que estabilice el sistema. A continuación se lleva a cabo la redefinición de algunos conceptos aplicados a sistemas no autónomos.

Definición 12.
Se dice que un determinado conjunto $\mho$ es un conjunto invariante de control para un sistema no autónomo si para todo $x_0 \in \mho$ existe una determinada ley de control $u_{t_k} = h(x_{tk})$ tal que el sistema permanece acotado por $\mho$ y además u_{t_k} es una actuación admisible cumpliéndose que $u_{t_k} \in U$.

Definición 13.
Una determinada función $V(x)$ se dice que es una función de Lyapunov de control, si para un determinado sistema no autónomo descrito según la ecuación (1.44) se cumple que:

$$\Delta V(x) = \min_{u}\{V(f(x_{init}, u_{init})) - V(x_{init})\} \leqslant 0 \tag{1.45}$$

Cumpliéndose en todo momento las restricciones sobre la vairable de control de tal forma que $u_{t_k} \in U$.

En un principio, podría pensarse que un planteamiento lógico del problema sería el de llevar a cabo el diseño de una determinada ley de control del sistema y a partir de aquí encontrar una determinada función de Lyapunov de control asociada al sistema de tal forma que se garantizase la estabilidad del sistema. Sin embargo, y siguiendo los procedimientos descritos en [3], el problema debe seguir el orden inverso, y llevarse a cabo el diseño de una determinada función ley de control que de por sí satisfaga una determinada función de control de Lyapunov, procedimiento el cual será descrito en apartados posteriores y que será la piedra angular en el proceso de diseño de controladores predictivos estabilizantes por diseño para los cuales se presentan las herramientas analizadas en el presente trabajo.

1.4.5 Robustez de sistemas no lineales

Hasta este momento se ha tratado única y exclusivamente el problema de la estabilidad en sistemas no lineales. Sin embargo no se hablado nada de un aspecto fundamental a tener en cuenta a la hora de llevar a cabo el diseño de cualquier sistema de control, como es el tema de la robustez del sistema. Las distintas metodologías de control robusto tienen como objetivo garantizar el buen funcionamiento del sistema ante la presencia de una serie de incertidumbres. Estas incertidumbres pueden tener distintos orígenes, desde las conocidas como incertidumbres estructurales, ocasionadas por la tenencia de modelos que no tienen en cuenta ciertas dinámicas del sistema o en los cuales no se han considerado ciertas no linealidades, hasta las incertidumbres paramétricas, originadas por la mala estimación de los parámetros del modelo, pasando por incertibumbres causadas por fuentes externas de perturbación o ruido no tenidas en cuenta.

El problema de control robusto consiste pues en el diseño de un controlador capaz de garantizar el buen funcionamiento del sistema bajo la presencia de una serie de perturbaciones. Para ello, el primer paso a realizar es la identificación de las distintas fuentes de incertidumbre que pueden afectar a nuestro sistema, para así llevar a cabo un cierta acotación de dichas incertidumbres para las cuales se realizará el diseño del sistema de control garantizando que éste funcionará en cualquiera de los casos. Ligado a esto, el diseñador del sistema de control se enfrenta también al problema de garantizar la estabilidad del sistema en presencia de incertidumbre, lo cual da origen al concepto de estabilidad robusta. En este apartado nos centraremos en este último problema, suponiéndose que ya se ha llevado a cabo el proceso de identificación y estimación de incertidumbres, y para lo cual se introducirá el concepto de función de Lyapunov entrada a estado, lo cual nos conducirá al análisis de estabilidad conocidad como estabilidad ISS (Input-to-State-Stability).

Para comenzar, supóngase un sistema autónomo no controlado de la forma:

$$x^+ = f(x_{init}, \omega) \tag{1.46}$$

Donde ω hace referencia al vector de incertidumbres que afectan al sistema y que ha sido previamente estimado, estando acotadas de la forma:

$$\omega \in W \tag{1.47}$$

Para lo cual se han tenido en cuenta una serie de márgenes de seguridad elegidos por el ingeniero de diseño. Si el sistema se encuentra exento de perturbaciones, de tal forma que el origen de referencia del sistema de referencia sigue siendo un punto de equilibrio del mismo tal que $f(0,0) = 0$. Si se supone que el conjunto de incertidumbres dependen en todo momento del estado del sistema, de tal forma que $\omega = \omega(x)$, se podría llevar a cabo el diseño del sistema de control considerando la teoría de Lyapunov y diseñando el sistema para la máxima incertidumbre posible en W que pueda afectar al sistema, de tal forma que:

$$\Delta V(x) = \underset{\omega \in W}{\text{máx}}\{V(x,\omega) - V(x)\} \tag{1.48}$$

De tal forma que si se puede encontrar una función de Lyapunov tal que $\Delta \leqslant 0$ se podrá garantizar que el sistema será estable para cualquier valor de la incertidumbre ω comprendida en W, siendo además asintóticamente estable si dicho decremento puede ser acotado por cierta función K, Ψ tal que:

$$\Delta V(x) \leqslant \Psi(\|x\|) \tag{1.49}$$

Véase como todo lo dicho anteriormente guarda cierta similitud, a pesar de ser de carácter completamente distinto, con el control robusto de norma infinita H_∞, en el cual se diseña el sistema de control considerando el peor de los casos posibles a través de la minimización de la norma infinito de una función que tiene en cuenta el máximo valor posible de las incertidumbres dentro del rango considerado.

El problema a todo lo anterior surge cuando las incertidumbres del sistema se encuentran simplemente acotadas, en cuyo caso la teoría de estabilidad de Lyapunov

aplicada al problema deja de surgir efecto [3], siendo ante esta necesidad cuando surge el concepto previamente comentado de estabilidad entrada a estado del sistema, basada también en la teoría de Lyapunov.

Definición 14.
Un determinado sistema autónomo dado por la expresión $x^+ = f(x_{init}, \omega)$, donde ω es el vector de incertidumbre, se dice que es estable entrada a estado si existe una determinada función tipo KL, Φ, y una determinada función K, Ψ, tal que el sistema evoluciona acotado de la forma:

$$\|x_{t_k}\| \leqslant \Phi(\|x_{t_{init}}\|, k) + \Psi(\mu) \tag{1.50}$$

Donde μ es la cota superior de la norma de ω tal que $\|w_k\| \leqslant \mu$ para todo valor de k. Esta expresión anterior puede ser interpretada como que el sistema será estable en ausencia de incertidumbres mientras que en el caso de haberlas estas permanecerán acotadas y por lo tanto el sistema no divergirá. Al igual que ocurría en el caso sin incertidumbre, la garantía del cumplimiento de un sistema de ser estable entrada estado estará asociado a la existencia de una determinada función de Lyapunov conocida como función de Lyapunov entrada a estado.

Definición 15.
Una determinada función V(), se dice que es una función de Lyapunov entrada a estado si se cumple que:

- Existen dos funciones K,Ψ_1 y Ψ_2 tales que el sistema $V(x)$ se encuentra acotada por ellas de la forma:

$$\Psi_1(\|x\|) \leq V(x) \leq \Psi_2(\|x\|) \forall x \in B \tag{1.51}$$

- Existe dos funciones K, Ψ_3 y Ψ_4 tales que:

$$V(f(x,\omega)) - V(x) \leqslant \Psi_3(\|x\|) + \Psi_4(\mu)\ \forall x \in B, \forall \omega \in \{\|\omega\| \leqslant \mu\} \tag{1.52}$$

Por lo tanto, un sistema se puede considerar estable entrada a estado en la región B si existe una función de Lyapunov entrada a estado asociada a él, lo cual está relacionado con la existencia de un un conjunto invariante robusto.

Definición 16.
Dado un sistema autónomo descrito por la ecuación:

$$x^+ = f(x_{init}, \omega) \tag{1.53}$$

Siendo ω el vector de incertidumbres del sistema acotado por W tal que $\omega \in W$, se define conjunto invariante positivo robusto a aquella región del espacio $\mho$ tal que para cualquier punto inicial del sistema x_{init} el estado a lo largo de su evolución permanece en $\mho$ para cualquier valor de la incertidumbre $\omega \in W$.

1.4.6 Robustez de Sistemas No Autónomos

Nuevamente, todo lo comentado ya sobre de la robustez de sistemas autónomos mediante el uso de la teoría de Lyapunov se puede extender a sistemas no autónomos gobernados por una determinada ley de control en bucle cerrado. Supóngase un sistema dado por la ecuación:

$$x^+ = f(x_{init}, u_{init}, \omega) \tag{1.54}$$

Donde u_{init} es el conjunto de variables de control en el instante actual, x_{init} el estado del sistema en el instante actual, y ω el vector de incertidumbre, acotadas de tal forma que:

$$\omega \in W = \{\omega \in \mathbb{R}^q = \|x\| \leqslant \mu\} \tag{1.55}$$

Siendo el origen de referencia del sistema un punto de equilibrio del mismo.

Para este caso, la función de Lyapunov de cotrol robusta puede ser expresada como:

$$\Delta V(x) = \min_{u \in U}\{max_{\omega in W}\{V(f(x,y,\omega)) - V(x)\}\} \leqslant -\Psi(\|x\|) \tag{1.56}$$

Para todo valor de x perteneciente a un conjunto B, y siendo Ψ una función K. Así, si un sistema tiene asociada una determinada función de Lyapunov de control, entonces el sistema será estabilizable a pesar de las incertidumbres para una determinada región $\mho$.

Asociado al concepto de función de Lyapunov de control, se encuentra el concepto de conjunto invariante de control de Lyapunov.

Definición 17.
Considérese el sistema no autónomo sometido a incertidumbre dado por la expresión:

$$x^+ = f(x_{init}, u_{init}, \omega) \tag{1.57}$$

Donde x_{init} es el estado del sistema en el instante actual, u_{init} el conjunto de variables de control en el instante actual, y ω el conjunto de incertidumbres.

Entonces un determinado conjunto $\mho$ se dice que es un conjunto invarinate de control robusto si para cualquier valor del estado inicial perteneciente a dicho conjunto tal que $x_0 \in \mho$ se tiene una ley de control admisible $u = h(x)$ tal que el estado evoluciona acotado por $\mho$ y sometido a cualquier incertidumbre $\omega \in W$.

1.5 Robustez y Estabilidad en Controladores NMPC

Hasta el momento todo el análisis de estabilidad y robustez ha sido aplicado a sistemas no lineales de carácter general. En esta sección se llevará a cabo el análisis de la estabilidad y robustez de sistemas no lineales particularizando a sistemas controlados por una ley de control predictivo no lineal basada en modelo.

De esta forma se comenzará llevando a cabo una descripción del problema de estabilidad presentado en los controladores predictivos convencionales. Hecho esto, se procederá a analizar algunas de las principales estrategias desarrolladas para realizar un diseño que garantice estabilidad de los controladores predictivos, aplicando la teoría de Lyapunov analizada en la sección anterior.

1.5.1 Estabilidad en controladores NMPC

El origen del control predictivo basado en modelo, tal y como ya se ha comentado en secciones anteriores, se encuentra en la necesidad de encontrar una manera práctica de implementar los tradicionales controladores óptimos para los cuales el problema de optimización difícilmente tiene solución (o de haberla no es posible calcularla) cuando se añaden restricciones al sistema.

Fue ante esta problemática, y en aras de promover la resolubilidad del problema que se modificó el problema de control óptmimo mediante la inclusión de dos novedades, que dieron lugar a lo que hoy en día entendemos por control MPC: el horizonte de control finito, y la cualidad de dicho horizonte de ser deslizante en el tiempo. Mediante la inclusión de estas modificaciones el problema de optimización se hizo factible, llegando incluso, en el caso de sistemas lineales, a convertirse en un problema de "papel y lápizz de muy fácil implementación en lo que refiere a la programación.

En el control óptimo se lleva a cabo la aplicación de la ley de control obtenida mediante la optimización de una función de coste a lo largo de toda su evolución desde

el estado actual hasta el infinito, de tal forma dicha función de coste toma la siguiente forma:

$$J_\infty(x_{t_k}) = \int_{t_{init}}^{\infty} L(x(t_k|t_{init})), K_\infty(x(t_k|t_{init})))dt \tag{1.58}$$

Donde la ley de control viene definida por la expresión $u = K_\infty(x_{t_k})$. De esta forma, el problema de optimización, teniendo en cuenta las restricciones, será de la forma:

$$\begin{aligned} &\min_{K_\infty} \ J_\infty(x_{t_k}) \\ &s.a \\ &\quad u(t_k|t_{init}) \ \in U \ \forall \ t_k > 0 \\ &\quad x(t_k|t_{init}) \ \in \ X \ \forall \ t_k > 0 \end{aligned} \tag{1.59}$$

Dado este problema de control óptimo, y bajo ciertas condiciones de controlabilidad del sistema, estabiliza asintóticamente todo estado del sistema en el cual exista una solución con un coste acotado [3]. Como ya se comentó previamente, a pesar de las buenas cualidades establizantes de este control, el problema de su utilización radica en la difícil resolución del problema de optimización, especialmente en el caso de existir restricciones sobre el sistema, siendo aquí donde surge la necesidad de la introducción de un horizonte de predicción finito, invariante y deslizante.

Simplemente llevando a cabo la sustitución del horizonte de predicción infinito en el problema descrito por (1.59) se obtiene el problema tradicional de un controlador predictivo.

$$\begin{aligned} &\min_{u_F}(t_k \ J_T(x_{t_k}, u_F(t_k)) \\ &s.a \\ &\quad u(t_k|t_{init}) \ \in U \ \forall \ 0 < t_k < T \\ &\quad x(t_k|t_{init}) \ \in \ X \ \forall \ 0 < t_k < T \\ &\quad x(T|t_{init}) \ \in \ \mho \end{aligned} \tag{1.60}$$

Siendo la función de coste del problema de optimización:

$$J_T(x_{t_k}, u_f(t_k)) = \int_{t_{init}}^{T} L(x(t_k|t_{init}), u(t_k|t_{init}))dt + V(x(T|t_{init})) \tag{1.61}$$

Donde $V(x)$ es el conocido como coste terminal, el cual será justificado más adelante, y que penaliza el valor del estado al final del horizonte. Además de ello, en el conjunto de restricciones se ha incluido una restricción que hace referencia al conjunto de posibles valores que el estado debe tomar al final del horizonte de predicción, lo cual será también justificado más adelante. A dicho conjunto $\mho$ se le denomina región terminal.

Esta sustitución del horizonte de predicción infinito por un horizonte de predicción finito tiene como ventaja que nuestro problema, anteriormente irresoluble, se convierta en un problema matemático de solución numérica [3].

Por otro lado, y como ya se ha comentado, dicho horizonte de predicción ha de ser deslizante en el tiempo. Esto supone que en cada uno de los instante de tiempo considerados o períodos de muestro, se llevará a cabo la resolución del problema para todo el horizonte finito descrito anteriormente, se aplicará el primer valor del conjunto de valores calculados mediante la resolución del problema de optimización, y se volverá a resolver el problema dando un paso hacia adelante en el tiempo y llevando a cabo la optimización para la misma longitud del horizonte de predicción que en el paso anterior pero partiendo de un instante de tiempo posterior.

En el problema de control óptimo, se llevaba a cabo la predicción para todo el horizonte desde el instante inicial hasta infinito. Esto hacía que el horizonte de predicción se redujese si se aplicaba de manera recursiva, pudiénsode así aplicar el teorema de optimalidad de Bellman, por el cual la trayectoria óptima calculada sería la misma en cada instante de cálculo. Sin embargo, al aplicar un horizonte de predicción finito, ya no es aplicable el principio de optimalidad de Bellman, la trayectoria óptima calculada en cada instante de tiempo es distinta, y se pierde así la convergencia del controlador de horizonte infinito, no pudiendo ser posible garantizar que el sistema evolucione

hasta su punto de equilibrio . Además de esto, también se produce el problema de una posible pérdida de factibilidad, ya que que un problema sea resoluble en un determinado instante no garantiza que lo sea en el siguiente [3]. Es a partir de aquí que se pone de manifiesto la necesidad de llevar a cabo una serie de cambios en el controlador en aras de garantizar la estabilidad del mismo, lo cual dio lugar a la aparición de técnicas de diseño de controladores predictivos con estabilidad garantizada por diseño.

Fueron numerosas las alternativas ideadas para intentar resolver este problema, sin embargo fueron dos las que finalmente consiguieron asentarse: la inclusión de coste y región terminal. Mediante el coste terminal se incluye en la función de coste del problema de optimización un término que penaliza el valor predicho al final del horizonte, mientras que mediante una región terminal se impone como restricción del sistema que el valor predicho al final del horizonte pertenezca a dicha región. Con estas modificaciones es posible garantizar la estabilidad del sistema si se cumplen una serie de condiciones, las cuales son:

- La región terminal impuesta como restricción, $\mho$, ha de tratarse de un conjunto invariante positivo, de tal forma que la ley de control estabilice el sistema y cumpla con las restricciones de estado y de la señal de control en todo momento, lo cual se traduce en que la evolución del sistema y las señales de control han de ser admisibles en dicho conjunto.

- El coste añadido $V(x)$ ha de tratarse de una función de Lyapunov estrictamente decreciente, de tal forma que dado el sistema $x^+ = f(x_{init}, h(x_{init}))$ se cumpla que:

$$V(f(x_{init}, h(x_{init}))) - V(x_{init}) \leqslant -L(x_{init}, h(x_{init})) \tag{1.62}$$

para todo valor del estado perteneciente al conjunto $\mho$. A esta formulación de control predictivo con coste y región terminal se le denomina formulación general de control predictivo con estabilidad garantizada. Los condicionantes descritos anteriormente se justifican de la siguiente manera:

- Si la región terminal es elegida como un invariante positivo, entonces el conjunto de estados factibles es aquel que permite estabilizar el sistema en un determinado tiempo T [3].

- Esto se debe a que la función de coste necesariamente ha de ser estrictamente decreciente. En el control MPC general esto no puede garantizarse ya que en cada una de las iteraciones del proceso de optimización se integra el mismo número de instantes temporales. Por ello, es necesaria la adición de este termino para garantizar que el coste en el estado siguiente siempre será menor que el coste en el estado actual, y ello se consigue mediante la elección de $V(x)$ como función de Lyapunov. Si se lleva a cabo la sustracción del coste calculado en el instante anterior J_p y el coste actual J_c se tiene que:

$$\begin{aligned} &J_c - J_p = -L(x,u_p) + \{L(x_p(T|t_{init}),h(x_p(T|t_{init}))+ \\ &V(f(x_p(T|t_{init}),h(x_p(T|t_{init})))- \\ &V(x_p(T|t_{init})) \end{aligned} \tag{1.63}$$

En dicha expresión se puede observar como si se lleva a cabo la elección del coste terminal como función de Lyapunov el segundo término será negativo y por lo tanto, el coste actual tiene un coste menor que el coste en el estado anterior. Es decir, el coste óptimo calculado es también una función de Lyapunov.

1.5.2 Robustez en controladores NMPC

Una vez que se han llevado a cabo las modificaciones necesarias para garantizar la estabilidad del controlador NMPC basándonos en el modelo nominal del mismo, se muestra a continuación como se ha de tratar el tema de robustez, de tal forma que el sistema sea capaz de seguir garantizando un correcto comportamiento y estabilidad en presencia de incertidumbre. Para ello, se comenzará realizando una introducción a la robustez en el control predictivo para proseguir con el diseño robusto de este tipo de controladores basado en estabilidad entrada a estado, ya comentada en secciones anteriores.

Para poder afirmar que un determinado controlador es robusto, se han de cumplir dos condiciones simultáneas: que el controlador garantice que el sistema tiene un comportamiento robusto, y que el controlador garantice que el sistema sea estable. A dichos objetivos de control se le conoce como *comportamiento robusto* y *estabilidad robusta*.

El problema de la robustez radica en que, a la hora de llevar a cabo el diseño de un determinado controlador, se tiene en cuenta un modelo, calculado bien por identificación directa del sistema, o por aplicación de principios fundamentales, cuya dinámica difiere del comportamiento del sistema real. Es decir, existe un cierto error de modelo. Debido a que nunca será posible llevar a cabo el cálculo del modelo real por muchas consideraciones que se tengan en cuenta, dicho error de modelado tampoco será conocido. Surge aquí el concepto de incertidumbre, la cual ha de ser de alguna forma estimada o acotada, tal y como se ha comentado en secciones anteriores.

Dicho esto, existen dos maneras alternativas de tratar el problema de diseño de controladores en presencia de incertidumbre. El primero de ellos es el de diseñar el sistema en función de su modelo nominal y ver qué grado de incertidumbre es capaz de soportar. El segundo de ellos por el contrario, consiste en llevar a cabo el diseño del sistema de control tal que sea capaz de soportar un determinado grado de incertidumbre sin perder sus buenas características en cuanto a comportamiento y estabilidad [3].

Para que un determinado sistema tenga garantías de estabilidad robusta, es decir, se mantenga estable en presencia de incertidumbres, son dos las condiciones generales que le sistema ha de cumplir. La primera de ellas es la de ser exponencialmente estable, lo cual asegura que la función de coste decrece distinta , y la segunda de ellas es la propiedad del coste óptimo obtenido de ser continuo en el sentido Lipschitz, lo cual asegura que el efecto de las incertidumbres sobre el sistema se encontrará acotado [3].

Por lo tanto, es necesario de alguna forma conseguir que la garantía de robustez se generalice a sistemas autónomos con estabilidad asintótica, lo cual se consigues mediante el uso de la estabilidad entrada a estado (ISS) ya comentada previamente en secciones anteriores.

Llevar a cabo la realización de un controlador robusto que garantice que el sistema sea capaz de proporcionar buenos resultados y estabilidad ante cualquier grado de incertidumbre no es una labor sencilla. Este problema presenta dificultad incluso en las metodologías tradicionales de diseño de controladores robustos, como puede ser el control H_∞. En ella, el problema de optimización de la norma infinito de la función de coste no siempre es posible para todo valor de la incertidumbre, por lo que en la mayoría de los casos hay que acotar los valores de la norma para que el problema sea realizable, llegándose así al diseño de un controlador H_∞ subóptimo.

En el caso de los controladores NMPC, en los cuales se intenta llevar a cabo la optimización de una función de coste en la cual se han incluido una serie de restricciones al sistema, la inclusión de incertidumbres hace que el problema se complique aún más siendo prácticamente intratable de manera práctica, por lo que tradicionalmente se ha ha de probar con diversos grados de incertidumbre para ver hasta qué punto el controlador puede tratar con ellos.

Se ha de tener en cuenta que la adición de incertidumbres al problema de control supone que en cada instante de tiempo el sistema, a través de la ley de control calculada, ha de ser capaz de cumplir con las restricciones impuestas a pesar de dichas incertidumbres. Una de las alternativas es la de comprobar que efectivamente el sistema funciona adecuadamente y cumple con las restricciones en presencia del máximo valor posible de incertidumbre, es decir de manera análoga al control H_∞, en el pero de los casos. De esta forma, la función objetivo podría ser expresada como:

$$J_T(x_{t_k}, u_F(t_k), W) = \max_{\omega_F} \left\{ \int_{t=0}^{T} L(x(t_k|t_{init}), u(t_k|t_{init}), \omega_{t_{k+\Delta t}}) + V(x(t_{init}+T|t_{init})) \right\} \tag{1.64}$$

Sin embargo, esta forma de plantear el problema presenta la problemática de que la inclusión de incertidumbres en la función de coste puede conducir a la pérdida de factibilidad, es decir, el problema será factible si es posible garantizar el proceso de optimización tiene solución para cualquier valor de la incertidumbre. Además se ha

de garantizar que se cumplen las restricciones en todo momento independientemente de la incertidumbre. Esto hace que el conjunto de estados dada cierta incertidumbre para los cuales el problema tiene solución se reduce a un conjunto, de aquí en adelante denominado $X_{w_{ba}}$.

Fueron numerosas las propuestas surgidas para tratar esta problemática, entre las cuales destacan el controlador min-max y el controlador en modo dual, el cual se basa en la existencia de un determinado conjunto invariante robusto $\mho$. Con todo esto, se intentará encontrar una ley de control de tal forma que la actuación aplicada en cada instante dependa del estado, el cual se encuentra ya afectado por la incertidumbre, de tal forma que la actuación dependa de dicha incertidumbre y la tenga en cuenta para compensarla. Existe pues un determinado conjunto de estados mucho mayor capaz de satisfacer estas restricciones , de forma que $X_{ba} \subset\subset X_{bc}$, lo cual da origen a la conocida como formulación del controlador en bucle cerrado, el cual, a pesar de ser difícilmente implementable, introduce un nuevo marco teórico en la manera de tratar las incertidumbres del problema.

1.5.3 Controlador Robusto en Bucle Cerrado

En este apartado se describen brevemente los fundamentos de la formulación del controlador robusto en bucle cerrado sin entrar en mucho detalle al no ser el objetivo principal del presente trabajo. Supóngase un sistema no autónomo expuesto a incertidumbre de la forma:

$$x^+ = f(x_{init}, u_{init}, \omega_{init}) \tag{1.65}$$

Donde x_{init}, u_{init} y ω_{init} hacen referencia al estado, la señal de control y la incertidumbre en el instante inicial respectivamente. Considérese también que dichas incertidumbres pertenecen a un determinado conjunto acotado tal que:

$$\omega_{init} \in W$$

Además, los valores de la señal de control y del estado se encuentran restringidos, de tal forma que:

$$u_{init} \in U$$
$$x_{init} \in X$$

La idea principal que subyace detrás de la formulación robusta en bucle cerrado es la sustitución de la secuencia de actuaciones predichas en el instante actual para todo el horizonte de control, donde no se tienen en cuenta las incertidumbres a las cuales se encuentra sometido el sistema, por una secuencia de leyes de control que sí las tiene en cuenta, de la forma:

$$\Upsilon(t_k) = \{u(t_k|t_k), v_1(), v_2(), ..., v_T\} \tag{1.66}$$

Así, el problema de optimización cambia, y es que ahora se incluyen como variables de decisión la secuencia de leyes de control en cada instante de tiempo considerado, de forma que éste toma la forma:

$$\begin{aligned} &\min_{\Upsilon_{init}} \ J_T(x_{init}, \Upsilon_{init}, W) \\ &s.a \\ &u(t_{init}|t_{init}) \in U \\ &v(x(t_k|t_{init})) \in U \\ &x(t_k|t_{init}) \in X \\ &x(t_{init} + T|t_{init}) \in \mho \end{aligned} \tag{1.67}$$

donde la nueva función de coste viene dada por la expresión:

$$J_T(x_{init}, \Upsilon_{init}, W) = \int_0^T L(x(t_k|t_{init}), v(x(t_k|t_{init}))) + V(x(T|t_{init})) \tag{1.68}$$

De alguna forma, será necesario, a la hora de llevar a cabo la optimización, predecir el valor de la incertidumbre en cada uno de los instantes del horizonte considerados. Ante esta problemática, una opción es la de ser conservadores y considerar siempre el peor de los casos. Este problema es considerablemente más complicado de resolver que el problema convencional, al tratarse las variables de decisión de leyes de control infinito dimensionales [3].

1.5.4 Cálculo del Coste Terminal

Una vez que se han presentado los aspectos fundamentales relativos a la estabilidad y robustez de controladores predictivos no lineales, mediante el uso de la teoría de Lyapunov, se desarrolla a continuación un procedimiento que de una manera relativamente sencilla permite calcular el coste terminal asociado al sistema como función de Lyapunov. Para ello supóngase una función de coste de etapa de la forma:

$$L(x,u) = \|x\|_Q^2 + \|u\|_R^2 \tag{1.69}$$

Donde Q y R son sendas matrices semidefinida y definida positiva respectivamente. Dicho coste, se aplica a un sistema dinámico de la forma:

$$x^+ = f(x_{init}, u_{init}) \tag{1.70}$$

Sistema que, una vez discretizado y linealizado, toma la forma:

$$x_{k+1} = Ax_k + Bu_k \tag{1.71}$$

Donde x_{k+1} es el estado del sistema en el instante de muestreo siguiente, x_k y u_k el estado y la señal de control del sistema en el instante de muestreo actual respectivamente, y A y B sendas matrices calculadas como:

$$A = \left.\frac{\partial f(x,u)}{\partial x}\right|_{(0,0)} \quad B = \left.\frac{\partial f(x,u)}{\partial u}\right|_{(0,0)} \tag{1.72}$$

El siguiente paso es llevar a cabo el cálculo de un determinado controlador por realimentación de estado (LMI, LQR, etc) cuya ley de control venga dada por la expresión:

$$u_k = Kx_k \tag{1.73}$$

Es decir, la actuación a aplicar depende única y exclusivamente del valor del estado en el instante actual y una determinada ganancia K. Una vez calculada esta ganancia, se ha de calcular una matriz P tal que cumpla la expresión:

$$A_K^T P A_K = -\hat{Q} \tag{1.74}$$

donde:

$$\begin{aligned} A_K &= A + BK \\ \hat{Q} &= \lambda Q^* \\ Q^* &= Q + K^T R K \end{aligned} \tag{1.75}$$

donde λ es un factor de ponderación mayor o igual que la unidad. Definida dicha matriz P el coste terminal viene dado por la expresión:

$$V(x) = x^T P x \tag{1.76}$$

Añadiendo este término a la función de coste, la expresión a optimizar es de la forma:

$$L(x,u) = \sum_{i=0}^{i=N} (\|x\|_Q^2 + \|u\|_R^2) + \|x_N\|_P^2 \tag{1.77}$$

Éste será el coste utilizado para garantizar que el sistema es estable con diseño. Además se puede comprobar como, para cualquier constante λ mayor que uno, se produce un incremento de la región de atracción del sistema, por lo cual a menudo se considerará una matriz de ponderación del coste terminal de la forma:

$$P_k = \lambda P \tag{1.78}$$

Algoritmos Numéricos para Optimización con Restricciones

2.1 Introducción

En el capítulo anterior se ha desarrollado la teoría correspondiente al análisis y diseño de controladores predictivos no lineales, así como la metodología a seguir para garantizar estabilidad y robustez en dichos controladores basándose en la teoría de Lyapunov. Como ha quedado patente a lo largo del desarrollo previo, uno de los aspectos fundamentales y críticos a la hora de llevar a cabo el desarrollo de cualquier controlador predictivo es la resolución del problema de optimización de la función de coste, tarea especialmente compleja cuando el sistema se encuentra sujeto a restricciones, lo cual incrementa enormemente la complejidad en la resolución del problema.

La problemática de llevar a cabo la resolución de problemas de optimización sujetos a restricciones ha sido extensamente estudiada y desarrollada en la literatura, especialmente en el ámbito de la academia, encontrándose numerosas alternativas de carácter analítico para las resolución de problemas de optimización tanto sujetos a restricciones como libres de ellas. Sin embargo, la gran complejidad de los problemas que se presentan actualmente hace que dichos métodos de resolución analítica no sean abordables en la mayor parte de las situaciones, sumado esto a la necesidad de adaptar el problema a las plataformas hardware utilizadas para llevar a cabo tal labor, intentando reducirse al máximo el coste computacional asociado. Para afrontar este nuevo escenario, se han desarrollado en los últimos tiempos diversos algoritmos de carácter numérico, los cuales son el objeto principal del presente capítulo.

Primeramente, se comenzará realizando una pequeña introducción general a los problemas de optimización sujetos a restricciones, llevándose un análisis de las condiciones necesarias y suficientes para garantizar la existencia de una solución óptima dada una determinada función a optimizar y una serie de restricciones a cumplir. Posteriormente, una vez comprendidos los aspectos fundamentales que afectan al proceso

de optimización, se llevará a cabo un análisis de los principales métodos numéricos desarrollados para la resolución de problemas de optimización, algunos de los cuales serán utilizados para la implementación práctica de los controladores predictivos no lineales propuestos en el presente trabajo.

2.2 Fundamentos de Optimización con Restricciones

En esta sección se presenta el problema general de optimización sujeto a restricciones así como la factibilidad de dicho problema. Por lo general, cualquier problema de de optimización no lineal (en el caso de querer minimizar) puede expresarse como:

$$\begin{aligned} &\text{mín } f(x) \\ &s.a \\ &\quad g(x) \leqslant 0 \\ &\quad h(x) = 0 \end{aligned} \tag{2.1}$$

Donde $f(x)$ es la función que se desea optimizar, y $h(x)$ y $g(x)$ las funciones que definen las restricciones de igualdad y desigualdad establecidas, todas ellas dependientes del conjunto de variables x, asumiéndose que tanto $f(x)$ como $g(x)$ tienen definidas derivadas continuas de primer y segundo orden. El objetivo es el de encontrar una solución x^o tal que $f(x)$ se haga mínima.

Para ello supóngase primeramente una serie de conjuntos $\Im$ y $\aleph$, donde $\Im$ hace referencia a la región o conjunto de valores de la variable de optimización para los cuales el problema de optimización es factible, y $\aleph$ el conjunto de valores para los cuales la norma del error entre el valor de la varible y el valor óptimo de la misma en inferior de un detemrinado valor ϵ tal que $\aleph(x^o) = \|x - x^o\| \leqslant \epsilon$. A partir de aquí, se establece que:

- Un determinado valor x^o es un mínimo global si $f(x^o) \leqslant f(x)\ \forall x \in \Im$.
- Un determinado valor x^o es un mínimo local si $f(x^o) \leqslant f(x)\ \forall x \in \aleph(x^o) \cap \Im$.

- Un determinado valor x^o es un mínimo local estricto si $f(x^o) < f(x)\ \forall x \in \aleph(x^o) \cap \Im$.
- Un determinado valor x^o es un mínimo local aislado si es único en el conjunto $\aleph(x^o) \cap \Im$.

El objetivo de todo problema de optimización será el de, una vez encontrada una solución al problema, en caso de existir, determinar si la solución encontrada se trata de un mínimo así como si existe alguna alternativa al problema (2.1) tal que se pueda obtener dicha solución a costa de un menor coste de cálculo.

2.2.1 Convexidad en Problemas Sujetos Restricciones

A la hora de llevar a cabo la resolución del problema de optimización, es siempre una notoria ventaja que este sea de carácter convexo, es decir, cuando la función objetivo a optimizar es convexa y la región de factibilidad del problema también lo es. Una determinada función objetivo $f(x)$ se trata de una función convexa en la variable x en un determinado dominio X si, para todos los puntos x_a y x_b pertenecientes al conjunto X, se satisface que:

$$\alpha f(x_a) + (1-\alpha) f(x_b) \geqslant f(\alpha x_a + (1-\alpha) x_b) \tag{2.2}$$

Donde α es cualquier valor entre cero y uno. De igual manera, se dice que una determinada región del espacio Z es convexa, si para todos los puntos x_a y x_b pertenecientes al conjunto Z, se cumple que:

$$\alpha x_a + (1-\alpha) x_b \in Z\ \forall \alpha \in (0,1) \tag{2.3}$$

A partir de aquí, y aplicado al problema de optimización definido por (2.1) se establecen los siguientes teoremas, tal y como se demuestran en [5].

Teorema 4.
Si la función $g(x)$ es una función convexa, y la función $h(x)$ es lineal, entonces la región $\Im$ es una región convexa.

Teorema 5.
Si la región $\Im$ es convexa y la función $f(x)$ también lo es, entonces todos y cada uno de los mínimos en dicha región es un mínimo global. De igual manera, si la región $\Im$ es estrictamente convexa, entonces cualquier mínimo local son un mínimo global óptimo.

Siempre y cuando exista convexidad, será posible llevar a cabo el cálculo de soluciones locales a un problema de optimización. En los casos en los que no existe convexidad, no es posible definir condiciones de optimalidad que garanticen la existencia de una solución global del problema, y se incrementa considerablemente el coste de la búsqueda de la solución en caso de que ésta existiese.

2.2.2 Condiciones Karush-Kuhn-Tucker

En esta sección se describen las condiciones necesarias para garantizar la optimalidad en un problema de optimización, conocidas como condiciones de KKT (Karush-Kuhn-Tucker). Para el problema de optimización general presentado en (2.1), la función de optimizacion Lagrangiana L puede ser expresada como:

$$L(x,u,v) = f(x) + g(x)^T u + h(x)^T v \tag{2.4}$$

Donde u y v son variables artificiales conocidas como multiplicadores de Lagrange para la introducción de las restricciones de desigualdad e igualdad respectivamente. Definida esta función Lagrangiana, se ha de cumplir la siguiente serie de condiciones.

- En el punto conrrespondiente a la solución encontrada para el problema, se ha de cumplir la condición de **estacionariedad**, es decir, la derivada de la función Lagrangiana en el punto de equilibrio ha de ser igual a cero, tal y como refleja la siguiente expresión.

$$\nabla_x L(x^o,u^o,v^o) = \nabla f(x^o) + \nabla g(x^o)^T u^o + \nabla h(x^o)^T v^o \tag{2.5}$$

- Se deben cumplir las restricciones tanto de igualdad como de desigualdad, es decir, el problema ha de tener **factibilidad**.

$$g(x^o) \leqslant 0, \quad h(x^o) = 0 \tag{2.6}$$

- La condiciones de desigualdad se han de satisfacer estrictamente o de manera inactiva, en cuyo caso no influyen en el proceso de optimización, lo cual se conoce como condición de **complementariedad** y que puede ser expresada como:

$$g(x^o)^T u^o = 0, \quad u^o \geqslant 0 \tag{2.7}$$

- Los gradientes de las restricciones activas en el punto de solución x^o han de ser linealmente independientes, lo cual se traduce en que las matrices columna de $\nabla h(x^o)$ y $\nabla g_i(x^o)$ sean de rango completo.

- Para tener en cuenta la curvatura de las restricciones activas del problema, se han de cumplir las condiciones de segundo orden.

$$\nabla p^T \nabla_{xx} L(x^o, u^o, v^o) p \geqslant 0 \tag{2.8}$$

donde p son las direcciones distintas de cero tales que el mínimo global es de gradiente cero y curvatura no negativa.

2.2.3 Problemas Sujetos a Restricciones Lineales

Establecidas ya las condiciones suficientes para la optimalidad del problema, se llevará a cabo el desarrollo para la obtención de la solución al problema descrito en (2.1).

Para ello, se considera primeramente el caso en el que el problema de optimización se encuentra linealmente restringido, es decir, toma la forma:

$$\begin{aligned} &\text{mín } f(x) \\ &s.a \\ &\quad Bx \leqslant b \\ &\quad Cx = c \end{aligned} \tag{2.9}$$

donde b y c son vectores y B y C sendas matrices. Para este tipo de restricciones lineales, las direcciones que acotan las posibles secuencias de valores factibles hasta un determinado punto x pueden ser representadas como un sistema de ecuaciones lineales y desigualdades conocido como cono de restricciones, el cual se hace coincidir con todas las direcciones factibles en la vecindad del punto x. Gracias a esto, es posible llevar a cabo una modificación del problema (2.9) tal y como se describe a continuación. Primeramente, considérese el siguiente teorema.

Teorema 6.
Si x^o es un punto solución de (2.9) sujeto a restricciones activas dadas por B y b, es decir, $B_i x^o = b$, entonces:

$$\nabla f(x^o)^T d \geqslant 0 \tag{2.10}$$

Para todo d dentro del cono definido por $Cd = 0$ y $B_i d \geqslant 0$

Además, considerando el teorema de Motzkin, se tiene que:

Teorema 7.
O bien el sistema descrito por :

$$\alpha^T z > 0, \;\; Az \geqslant 0, \;\; Dz = 0 \tag{2.11}$$

tiene solución, o bien la tiene el sistemas descrito por:

$$\begin{aligned} &\alpha y_1 + A^T y_2 + D^T y_3 = 0 \\ &y_1 > 0, \;\; y_2 \geqslant 0 \end{aligned} \tag{2.12}$$

pero nunca ambos sistemas. Ambos teoremas permiten demostrar la siguiente propiedad.

Teorema 8.
Si x^o es una solución local para (2.9) entonces se satisfacen las siguientes condiciones KKT:

$$\begin{aligned} &\nabla f(x^o) + B^T u^o + C^T v^o = 0 \\ &Bx^o \leqslant b, \;\; Cx^o = 0 \\ &(Bx^o - b)^T u^o = 0, \;\; u^o \geqslant 0 \end{aligned} \tag{2.13}$$

Si estas condiciones son satisfechas y además $f(x)$ es convexa en un determinado punto x^*, entonces x^* es una solución global de (2.9), tal y como se demuestra en [5].

2.2.4 Problemas Sujetos a Restricciones No Lineales

En el caso en el que las restricciones eran de carácter lineal, tal y como se ha visto en el apartado anterior, era necesario llevar a cabo la representación de las direcciones límites de las secuencias factibles mediante el uso de un cono de restricciones. En el caso de que las restricciones sean de carácter no lineal, será también necesario garantizar un enlace entre las condiciones KKT y el teorema (2.10), de tal forma que se siga cumpliendo que:

$$\begin{aligned} &\nabla f(x^o) + \nabla g(x^o)^T u^o + \nabla h(x^o)^T v^o = 0 \\ &g(x^o) \leqslant 0, \;\; h(x^o) = 0 \\ &g(x^o)^T u^o = 0, \;\; u^o \geqslant 0 \end{aligned} \tag{2.14}$$

Sin embargo, es necesario la adición de una nueva condición para garantizar que las direcciones limitantes en el punto solución pueden ser representadas mediante las

siguientes condiciones de como derivadas de la linealización de las restricciones activas que afectan al sistema.

$$\nabla h_i(x^o)^T d = 0, \;\; \nabla g_i(x^o)^T d \leqslant 0 \tag{2.15}$$

para todo valor de i perteneciente a la región de factibilidad del problema. Partiendo de aquí, y para poder garantizar en el enlace, es necesario una nueva cualificación de las restricciones que prueben que las direcciones limitantes de las restricciones activa no lineales pueden ser representadas por su linealizaciones en el punto solución. Para ello, es común utilizar la condición conocida como cualificación de restricciones linealmente independientes, por sus siglas en ingles, LICQ (Linear Independence Constraint Qualification), la cual se define de la siguiente manera.

Definición 18.
Dada un solución local del problema descrito por (2.1), y un determinado conjunto activo $A(x^o)$, se define LICQ mediante la independencia lineal de los gradientes de las restricciones.

$$\nabla g_i(x^o), \;\; \nabla h_i(x^o), \;\; i \in A(x^o) \tag{2.16}$$

Para hacer patente la necesidad de la cualificación de restricciones, es necesario el uso de los siguientes teoremas.

Teorema 9.
El conjunto de de direcciones limitantes de todas las secuencias factibles es un subconjunto del cono. Si además LICQ es aplicable a (2.15), entonces:

- El cono es equivalente al conjunto de direcciones limitantes de las secuencias factibles.
- las direcciones limitantes que satisfacen (2.15) con la norma de las direcciones igual a la unidad, una determinada secuencia de valor x^k pueden ser siempre encontrada de tal forma que:

$$\begin{aligned} h_i(x^k) &= t^k \nabla h(x^o)^T d = 0 \\ g_i(x^k) &= t^k \nabla g_i(x^o)^T d \leqslant 0, \; i \in A(x^o) \end{aligned} \tag{2.17}$$

donde t^K son valores pequeños positivos tales que $\lim t^k \longrightarrow 0$

Mediante el uso de este teorema, es posible extender el teorema 8 a problema restringido de forma no lineal. Combinando los resultados anteriores, se llega a la conclusión descrita en el siguiente teorema.

Teorema 10.
Si x^o es una solución local del problema general descrito por (2.1) y LICQ se cumple en esta solución, entonces, y tal y como se demuestra en [5]:

- Se satisfacen las condiciones KKT.
- Si $f(x)$ y $g(x)$ son convexas, $h(x)$ es lineal y las condiciones KKT se cumplen en un punto x^*, entonces x^* es una solución global de (2.1)

Además, el cumplimiento de LICQ conduce hacia otra propiedad de los multiplicadores del problema, dada por el siguiente teorema.

Teorema 11.
Dado un determinado punto solución del problema x^o, que satisface las condiciones KKT, y que pertenece a un conjunto activo $A(x^o)$ con multiplicadores u^o y v^o, entonces los multiplicadores son únicos en el punto solución, tal y como se demuestra en [5].

Hasta ahora se ha considerado únicamente LICQ por ser la más extendida. Sin embargo, existen en la literatura diversos métodos de cualificación de restricciones. en particular, es común la utilización de la cualificación de restricciones de Mangasarian Fromovitz (MFCQ), la cual viene dada por la siguiente definición.

Definición 19.
Dada una determinada solución a (2.1) x^o, y un conjunto activo $A(x^o)$, el MFCQ definido por una dependencia lineal de los gradientes de restricciones de igualdad y la existencia de una búsqueda en la dirección d tal que:

$$\nabla g_i(x^o)^T d < 0, \;\; \nabla h_i(x^o)^T d = 0. i \in A(x^o) \tag{2.18}$$

Entonces MFCQ es siempre satisfecho si LICQ es satisfecha. Además, la satisfacción de MFCQ conduce a multiplciadores u^o y v^o acotados, aunque no necesariamente únicos.

2.2.5 Condiciones de Segundo Orden

En este apartado se analizan las condiciones de segundo orden descritas por (2.8). Para ello, comenzamos definiendo dos conos para los siguientes sistemas lineales.

$$C_1(x^o) = \{d|\nabla h(x^o)^T d = 0.\nabla g_i(x^o)^T d \leqslant 0,\ \ i \in A(xo)\}$$
$$C_2(x^o,u^o) = \{d\nabla h(x^o)^T d = 0, \nabla g_i(x^o)^T d = 0, i \in A_s(x^o), \nabla g_i(x^o)^T d \leqslant 0,\ \ i \in A_w(x^o)\} \tag{2.19}$$

donde tal y como puede observarse $C_2(x^o,u^o) \subseteq C_1(x^o)$. Si se asume que se cumple LICQ entonces sabemos también que $C_1(x^o)$ representa el espacio de direcciones limitantes de las secuencias factibles. Con estas definiciones, es posible demostrar las siguientes propiedades.

Teorema 12.
Si se supone que x^o es una solución local de (2.1), se cumple LICQ, y u^o y v^o son multiplicadores que satisfacen las condiciones KKT, entonces:

$$d^T \nabla_{xx} L(x^o,u^o,v^o) \geqslant 0 \forall d \in C_2(x^o,u^o) \neq 0 \tag{2.20}$$

Ya que LICQ se cumple, el cono $C_1(x^o)$ contiene las direcciones limitantes de todas las secuencias factibles, y x^o es una solución de (3.1), se sabe que no puede haber secuencias factibles crecientes hasta llegar a la solución para valores de k lo suficientemente grande. Si ahora se asume que $\nabla d^T \nabla_{xx} L(x^o,u^o,v^o) < 0$ para algunas de las

direcciones limitantes $d \in C_2(x^o, u^o) \subseteq C_1(x^o)$ y que se puede construir una secuencia factible x^k asociada a las direcciones limitantes que satisfacen:

$$h_i(x^k) = t^k \nabla h_i(x^o)^T d = 0, g_i(x^k) = t^k \nabla g_i(x^o) d \leqslant 0, i \in A(x^o) \tag{2.21}$$

Aplicando el teorema de Taylor, y viendo que $\nabla L(x^o, u^o, v^o) = 0$ se obtienen los siguientes resultados:

$$\begin{aligned} & f(x^k) = L(x^k, u^o, v^o) \\ & = L(x^o, u^o, v^o) + t^k \nabla L(x^o, u^o, v^o)^T d \\ & + \frac{(t^k)^2}{2} d^T \nabla_{xx} L(x^o, u^o, v^o) d + o((t^k)^2) \\ & = f(x^o) + \frac{t^k}{2} d^T \nabla_{xx} L(x^o, u^o, v^o) d + o((t^k)^2) \\ & < f(x^o) + \frac{(t^k)^2}{4} d^T \nabla_{xx} L(x^o, u^o, v^o) d < f(x^o) \end{aligned} \tag{2.22}$$

Para un valor de k lo suficientemente grande. Esto contradice la asunción de que x^o es solución y que no hay secuencia factibles a dicha solución, comprobándose así lo expuesto en (2.20).

Teorema 13.
Si x^o y los multiplicadores satisfacen las condiciones KKT y:

$$d^T \nabla_{xx} L(x^o, u^o, v^o) > 0 \forall d \in C_2(x^o, u^o) \neq 0 \tag{2.23}$$

entonces x^o es una solución estricta de (2.1)

Que x^o sea un mínimo local estricto, significa que todas y cada una de las secuencias factibles han de cumplir que $f(x^k) > f(x^o)$ para un valor de k lo suficientemente grande. Ahora se consideran cualquiera de las secuencias factibles asociadas de las direcciones limitantes en los siguientes casos:

- $d \in C_2(x^o, u^o) \subseteq C_1(x^o)$. En este caso, para cualquier secuencia con dirección limitante , $d \in C_2(x^o, u^o)$, $\|d\| = 1$ y:

$$f(x^k) \geqslant f(x^o) + \frac{1}{2} d^T \nabla_{xx} L(x^o, u^o, v^o) d \left\| x^k - x^o \right\|^2 + o\left(\left\| x^k - x^o \right\|^2 \right) \tag{2.24}$$

- $d \in C_1(x^o) \setminus C_2(x^o, u^o)$. En este caso, $d^T \nabla_{xx} L(x^o, u^o, v^o)$ puede no ser positiva para $d \in C_1(x^o) \setminus C_2(x^o, u^o)$, por lo que es necesario desarrollar otra aproximación. Como $d \notin C_2(x^o, u^o)$ existe al menos una restricción con $\nabla g_i(x^o)^T d < 0, i \in A_s(x^o)$. Además se tiene que:

$$\begin{aligned} & h(x^k)^T v^o \\ & g_i(x^k) u_i^o = \nabla g_i(x^o)^T d \left\| x^k - x^o \right\| u_i^o + o\left(\left\| x^k - x^o \right\| \right) \end{aligned} \tag{2.25}$$

Con esto se tiene que,

$$L(x^k, u^o, v^o) = f(x^k + \nabla g_i(x^o)^T d \left\| x^k - x^o \right\| u_i^o + o(\left\| x^k - x^o \right\|) \tag{2.26}$$

Aplicando el teorema de Taylor y las condiciones KKT, se obtiene,

$$L(x^k, u^o, v^o) = f(x^o) + O\left(\left\| x^k - x^o \right\|^2 \right) \tag{2.27}$$

E igualando dichas expresiones,

$$f(x^k) - f(x^o) + \nabla g_i(x^o)^T d \left\| x^k - x^o \right\| u_i^0 = o\left(\left\| x^k - x^o \right\| \right) \tag{2.28}$$

Expresión la cual, para un valor de k lo suficientemente grande, puede ser expresada como:

$$f(x^k) - f(x^o) \geqslant -\frac{1}{2} \nabla g_i(x^o)^T d \left\| x^k - x^o \right\| u_i^o > 0 \tag{2.29}$$

Lo cual proporciona los resultados deseados para el segundo caso.

2.2.6 Propiedades del Hessiano

Las condiciones de optimalidad que han sido desarrolladas previamente puede que no sean fáciles de comprobar. Por ello, se han desarrollado alternativas para estas condiciones de optimalidad, como es la evaluación de la matriz Hessiana proyectada en un determinado subespacio. Considérese la siguiente definición.

Definición 20.
Sea A una matriz de dimensiones $n \times m$ cuyas columnas son de rango completo. A partir de ella, es posible definir un definir una matriz de espacio nulo, Z tal que $A^T Z = 0$. Para llevar a cabo la determinación de la matriz de espacio nulo, se realiza una factorización QR de A, tal y como se muestra a continuación.

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q^R \mid Q^N] \begin{bmatrix} R \\ 0 \end{bmatrix} = Q^R R \tag{2.30}$$

Donde $Q \in \mathbb{R}^{n \times n}$ es una matriz ortonormal y $Q^R \in \mathbb{R}^{n \times m}$,$Q^N \in \mathbb{R}^{n \times (n-m)}$ y $R \in \mathbb{R}^{m \times m}$ es una matriz triangular superior. A partir de esta representación es facil ver que $(Q^R)^T Q^R = I_m$, $(Q^N)^T Q^N = I_{n-m}$, $(Q^R)^T Q^N = 0$, $a^T Q^N = 0$, y que el vector $A^T d = 0$ puede ser representado como $d = Q^N q$ con $q \in \mathbb{R}^{n-m}$. Además, dada un matriz $W \in \mathbb{R}^{n \times n}$ tal que $d^T W d > 0 \forall A^T d = 0$, se tiene de $d^T W d = q^T((q^N)^T W Q^N) q$ que la matriz proyectada $(Q^N)^T W Q^N$ es definida positiva.

A partir de la proyección del Hessiano es posible derivar las condiciones necesarias introduciendo dos nuevos conjuntos a partir de subespacios lineales, los cuales se expresan como:

$$\begin{aligned} C_3(x^o) &= \{d \mid \nabla h(x^o)^T d = 0, \ \ \nabla g_i(x^o)^T d = 0 \\ C_4(x^o, u^o) &= \{d \mid \nabla h(x^o)^T d = 0, \nabla g_i(x^o)^T d = 0 \end{aligned} \tag{2.31}$$

A partir de estos conceptos se derivan las siguientes propiedades:

- Los subespacios previos han de satisfacer que $C_3(x^o) \subseteq C_4(x^o, y^o)$ y $C_2(x^o, u^o) \subseteq C_4(x^o, u^o)$.

- Usando $C_3(x^o)$ es posible definir una matriz A_3 a partir de las columnas de $\nabla g_i(x^o)$ y de la correspondiente matriz de espacio nulo Q_3^N, y de manera análoga, A_4 y Q_4^N a partir de $C_4(x^o, u_o)$. Con dichas matrices, todas las direcciones $d \in C_3(x^o)(d \in C_4(x^o, u^o))$ como $d = Q_3^N q(d = Q_4^N q)$

- Debido a que $d \in C_2(x^o, u^o) \subseteq C_4(x^o, u^o)$, si $d^T \nabla_{xx} L(x^o, u^o, v^o) d > 0$, $d \in C_4(x^o, u^o)$, entonces la condición de suficiencia descrita por (2.23) es también satisfecha. Por tanto, la definición de la proyección de la matriz Hessiana es una condición de suficiencia de segundo orden más fuerte que la presentada anteriormente.

- Debido a que $d \in C_3(x^o) \subseteq C_2(x^o, u^o)$, si se cumple (2.20), entonces también se ha de satisfacer que $d^T \nabla_{xx} L(x^o, u^o, v^o) d \geqslant 0$. Entonces la semidefinibilidad positiva de las proyección del Hessiano es condición necesaria de segundo orden (más débil).

- Si se asume complementariedad estricta, las condiciones de segundo orden pueden ser simplificadas como:

  - Condición necesaria de segundo orden: $(Q_3^N)^T \nabla_{xx} L(x^o, u^o, v^o) Q_3^N$ ha de ser definida positiva.
  - Condición suficiente de segundo orden: $(Q_3^N)^T \nabla_{xx} L(x^o, u^o, v^o) Q_3^N$

2.3 ALGORITMOS NUMÉRICOS EN PROBLEMAS NLP

En esta sección se llevará a cabo el análisis de algunos de los principales métodos numéricos utilizados para la resolución de problemas de optimización no lineales sujetos a restricciones, tanto de igualdad como de desigualdad. Para ello, se han considerado aquellos métodos utilizados actualmente por los solvers desarrollados para la implementación en un computador de tal labor, dejándose así de lado los métodos tradicionales que no son capaces de tratar con los complejos problemas de optimización que se presentan en la actualidad, como pudieren ser los métodos de optimización

de Newton para problemas con restricciones de igualdad, los cuales difícilmente eran capaces de tratar con restricciones de desigualdad.

El problema general de optimización no lineal ya presentado por (3.1) venía dado por:

$$\begin{aligned} &\text{mín } f(x) \\ &s.a \\ &\quad g(x) \leqslant 0 \\ &\quad h(x) = 0 \end{aligned} \tag{2.32}$$

Donde $f(x)$, $g(x)$, y $h(x)$ son funciones con primera y segunda derivadas continuas. Dicha formulación genérica del problema es conveniente que sea reformulada a la hora de derivar los algoritmos de programación no lineal utilizados, surgiendo así la formulación general del problema NLP, en la cual mediante la inclusión de variables de inactividad (slack variables), el problema puede ser expresado como:

$$\begin{aligned} &\text{mín } f(x) \\ &s.a \\ &\quad g(x) + s = 0 \\ &\quad s \geqslant 0 \\ &\quad h(x) = 0 \end{aligned} \tag{2.33}$$

Mediante esta nueva reformulación, las no linealidades han sido removidas de la función de desigualdad. De una manera más general, el problema puede ser expresado, acotándose los límites superiores e inferiores de las variables del problema como:

$$\begin{aligned} &\text{mín } f(x) \\ &s.a \\ &\quad c(x) = 0 \\ &\quad x_L \leqslant x \leqslant x_U \end{aligned} \tag{2.34}$$

Ambas alternativas de formulación del problema no se comportarán de igual manera a pesar de que las formulaciones sean equivalentes. Por este motivo, la elección de una u otra dependerá de la naturaleza del algoritmo utilizado para llevar a cabo la resolución del problema. A partir de ahora, y para proceder al desarrollo de los algoritmos presentados en esta sección, las condiciones KKT serán expresadas como:

$$\begin{aligned} &\nabla L(x^o,u^o,v^o) = \nabla f(x^o) + \nabla c(x^o)v^o - u_L^o + u_U^o = 0 \\ &c(x^o) = 0 \\ &0 \leqslant u_L^o \perp (x^o - x_L) \geqslant 0 \\ &0 \leqslant u_U^o \perp (x_U - x^o) \geqslant 0 \end{aligned} \tag{2.35}$$

Para poder lidiar con las condiciones de complementariedad, son dos las estrategias utilizadas en el diseño de algoritmos NLP. Mediante la estrategia conocida como de conjuntos activos, el algoritmo fuerza a la variable a tomar el valor de su límite, o el correspondiente multiplicador límite a valer cero. Por otro lado, en los métodos de punto interior o barrera las condiciones de complemeriedad son expresadas como:

$$\begin{aligned} U_L(x - x_L) &= \mu e \\ U_U(x_U - x) &= \mu e \end{aligned} \tag{2.36}$$

Donde $\mu > 0$, $e^T = [1,1,...,1]$, $U_L = diag(u_L)$, $U_U = diag(u_U)$. Si las variables x se mantiene estrictamente entre los límites de acotación y los multiplicadores u_L y u_U permanecen estrictamente positivos, las condiciones KKT son resultas directamente mediante un valor de μ fijo. Por lo tanto, mediante la resolución de una determinada secuencia de estas estaciones cuando $\mu \to 0$ se obtiene la solución del problema original de programación no lineal. En las secciones que siguen se llevará a cabo el desarrollo de tres estrategias derivadas de la formulación de las condiciones KKT descritas por (2.35), los métodos SQP, como estrategia de conjuntos activos, los métodos de punto interior que hace uso de funciones de barrera, y los métodos de conjuntos activos anidados, donde las variables son particionadas para tratar con las condiciones de complementariedad.

2.3.1 Métodos SQP

Uno de los métodos de más utilizados por los solver actuales debido a su amplio rango de aplicabilidad y al hecho de que hereda las buenas cualidades en cuanto a rápida convergencia de los métodos de Newton son los métodos SQP (Sequential Quadratic Programming). Estos métodos se derivan a partir de las condiciones KKT y la asunción de que los límites activos del problema son conocidos de antemano.

Supónganse los conjuntos $A_L = \{i_1, i_2, ..., i_j\}$ donde $x^o_{i_j} = X_{L_{i_j}}$ y $A_U = \{i'_1, i'_2, ..., i'_j\}$ donde $x^o_{i'_j} = x_{U_{i'_j}}$ así como los multiplicadores u_{A_L} y u_{A_U}correspondientes a dichos conjuntos y las matrices E_L y E_U que determinan el valor de las variables en los límites donde:

$$\{E_L\}_{ij} = \begin{cases} 1 \; si \; i = i_j \\ 0 \text{ de otra forma} \end{cases} \tag{2.37}$$

definiéndose E_U de igual manera. Estas nuevas matrices hacen se puedan expresar las nuevas condiciones KKT como:

$$\begin{aligned} &\nabla L(x, u_L, u_U, v) = \nabla f(x) + \nabla c(x)v - E_L u_{A_L} + E_U u_{A_U} = 0 \\ &c(x) = 0 \\ &E_U^T x = E_U^T x_U \\ &E_L^T x = E_L^T x_L \end{aligned} \tag{2.38}$$

Aplicando el método de Newton al sistema, se obtiene el siguiente sistema de ecuaciones lineales para una determinada iteración:

$$\begin{bmatrix} \nabla_{xx} L(x^k, u_A^k, v^k) & \nabla c(x^k) & -E_L & E_U \\ \nabla c(x^k)^T & 0 & 0 & 0 \\ -E_L^T & 0 & 0 & 0 \\ E_U^T & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_v \\ d_{u_{A_L}} \\ d_{u_{A_U}} \end{bmatrix} = - \begin{bmatrix} \nabla L(x^k, u_A^k, v^k) \\ c(x^k) \\ E_L^T(x_L - x^k) \\ E_U^T(x^k - x_U) \end{bmatrix} \tag{2.39}$$

O de manera equivalente

$$\begin{bmatrix} \nabla_{xx}L(x^k,u_A^k,v^k) & \nabla c(x^k) & -E_L & E_U \\ \nabla c(x^k)^T & 0 & 0 & 0 \\ -E_L^T & 0 & 0 & 0 \\ E_U^T & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ \bar{v} \\ \bar{u}_{A_L} \\ \bar{u}_{A_U} \end{bmatrix} = - \begin{bmatrix} \nabla f(x^k) \\ c(x^k) \\ E_L^T(x_L - x^k) \\ E_U^T(x^k - x_U) \end{bmatrix} \tag{2.40}$$

donde $\bar{u}_{A_L} = u_{A_L}^k + d_{u_{A_L}}$, $\bar{u}_{A_U} = u_{A_U}^k + d_{u_{A_U}}$ y $\bar{v} = v^k + d_v$. Este sistema correspondiente a las condiciones KKT de primer orden, y siendo esto la base de los métodos SQP, puede ser expresado como un problema de programación cuadrática como:

$$\begin{aligned} &\min_{d_x} \nabla f(x^k)^T d_x + \frac{1}{2} d_x^T W^k d_k \\ &s.a \\ &\quad c(x^k) + \nabla c(x^k)^T d_x = 0 \\ &\quad E_U^T(x^k + d_x) = E_U^T x_U \\ &\quad E_L^T(x^k + d_k) = E_L^T x_L \end{aligned} \tag{2.41}$$

Donde $W^k = \nabla_{xx}L(x^k,u^k,v^k) = \nabla_{xx}f(x^k) + \sum_{j=1}^m \nabla_{xx}c_j x^k v_j^k$. Si se permite la selección de límites activos locales a x^k, es posible expresar el problema de programación cuadrática QP para cada iteración k como:

$$\begin{aligned} &\min_{d_x} \nabla f(x^k)^T d_x + \frac{1}{2} d_x^T W^k d_x \\ &s.a \\ &\quad c(x^k) + \nabla c(x^k)^T d_x = 0 \\ &\quad x_L \leqslant x^k + d_x \leqslant x_U \end{aligned} \tag{2.42}$$

Para llevar a cabo la resolución de (2.34), la solución del problema de programación cuadrática conduce a la selección del conjunto activo con multiplicadores cuadráticos, $\bar{u}_L$, $\bar{u}_U$ y $\bar{v}$, los cuales son actualizados en cada iteración. En el caso de que (2.42) tenga solución $d_x = 0$, es aplicable el siguiente teorema.

Teorema 14.
Si el problema (3.42) tiene solución, los gradientes de las restricciones activas vienen dados por $A(x^k) = [\nabla c(x^k) \mid E_L \mid E_U]$ que tiene columnas linealmente independientes (cumpliendo LICQ), y la proyección de W^k en el espacio nulo $A(x^k)^T$ es definido positivo, entonces $x^k, \bar{v}, \bar{u}_L, \bar{u}_U$ cumple las condiciones KKT de (2.34) si y sólo si $d_x = 0$, tal y como se prueba en [5].

Por tanto, sustituyendo el problema de programación cuadrática con paso de Newton, se desarrolla el algoritmo SQP como una extensión del método de Newton para sistemas sujetos a restricciones de igualdad.

2.3.2 Full Space SQP

El algoritmo básico para llevar a cabo la implementación del método SQP es el conocido como FSSQP (Full-Space SQP). Para poder llevar a cabo la resolución del problema SQP, es necesario que el problema de programación cuadrática derivado tenga solución única en cada una de las iteraciones. El problema principal que se encuentra es que debido a que la formulación se realiza en función de la matriz Hessiana en cada una de las iteraciones, no es posible garantizar la no singularidad, la cual sí se encuentra siempre que el problema sea linealmente independiente de las restricciones activa y el Hessiano reducido sea definido positivo.

Para tratar con la no factibilidad del problema QP, es necesaria una estrategia alternativa para hallar la solución al problema NLP, o al menos converger hacia un punto donde se minimize la no factibilidad de manera local. Para esto, son dos las estrategias seguidas:

- Incluir una fase conocida como fase de restauración para encontrar dicho punto donde el problema es de mayor factibilidad.
- Formular el problema QP relajando las restricciones (técnica conocida como modo elástico), mediante la cual se crea una región de factibilidad que garantiza la existencia de solución.

Para llevar a cabo esta relajación, supóngase un función de penalización de norma uno de la forma:

$$\phi(x,\rho) = f(x) + \rho \|c(x)\|_1 \tag{2.43}$$

Mediante el uso de esta expresión, el problema de programación cuadrática puede ser expresado como:

$$\begin{aligned} &\text{mín}\, \nabla f(x^k)^T d_x + \frac{1}{2} d_x^T W^k d_x \\ &+ \rho \sum_j | \, c_j x^k + \nabla c_j(x^k)^T d_x \, | \\ &s.a \\ &\quad x_L \leqslant x^k + d_x \leqslant x_U \end{aligned} \tag{2.44}$$

Mediante la introducción de dos nuevas variables no negativas al problema, s y t, el problema puede ser reformulado, obteniéndose así el conocido como problema QP de norma uno:

$$\begin{aligned} &\min_{d_x} \nabla f(x^k) d_x + \frac{1}{2} d_x^T W^k d_x + \rho \sum_j (s_j + t_j) \\ &s.a \\ &\quad c(x^k) + \nabla c(x^k)^T d_x = s - t \\ &\quad x_L \leqslant x^k + d_x \leqslant x_U \\ &\quad s,t \geqslant 0 \end{aligned} \tag{2.45}$$

donde, debido a que $d_x = 0$, el punto $s_j = \text{máx}(0, c_j(x^k))$, $t_j = -\text{mín}(0, c_j(x^k))$ es un punto factible de (2.45), y eligiendo W^k como una matriz acotada definida positiva, es siempre posible generar una solución al problema QP. Además, se cumple que la solución encontrada puede ser expresada como decremento de la dirección de una función de mérito de la norma uno expresada por (2.43), lo cual permite que el algoritmo SQP sólo tenga que proceder a realizar una reducción de la función ϕ.

Otra forma de expresar el problema de manera alternativa y que se traduce en un problema QP de menor tamaño es:

$$\begin{aligned} &\min f(x) + \bar{M}\bar{\chi} \\ &s.a \\ &\quad c(x)(1-\bar{\chi}) = 0 \\ &\quad x_L \leqslant x \leqslant x_U,\ \bar{\chi} \in [0,1] \end{aligned} \tag{2.46}$$

Donde $\bar{M}$ es un peso arbitrariamente grande que penaliza la variable escalar $\bar{\chi}$. Eligiendo $\bar{M}$ lo suficientemente grande facilita encontrar una solución del problema (2.46) con $\bar{\chi} = 0$ cuando existe solución factible para (2.34). Para este problema (2.46) es posible escribir el problema QP como:

$$\begin{aligned} &\min_{d_x} \nabla f(x^k)^T d_x + \frac{1}{2} d_x^T W^k d_k + M\chi \\ &s.a \\ &\quad c(x^k)(1-\chi) + \nabla c(x^k)^T d_x = 0 \\ &\quad x_L \leqslant x^k + d_k \leqslant x_U,\ \chi \geqslant 0 \end{aligned} \tag{2.47}$$

Donde $\chi = \frac{\bar{\chi}-\bar{\chi}^k}{1-\bar{\chi}^k}$ y $M = (1-\bar{\chi}_k)\bar{M}$, asumiendo que $\bar{\chi}_k < 1$. Para garantizar que la solución encontrada es única cuando $\chi > 0$, el problema QP (2.47) puede ser modificado como:

$$\begin{aligned} &\min_{d_x} \nabla f(x^k)^T d_x + \frac{1}{2} d_x^T W^k d_x + M(\chi + \chi^2/2) \\ &s.a \\ &\quad c(x^k)(1-\chi) + \nabla c(x^k)^T d_x = 0 \\ &\quad x_L \leqslant x^k + d_x \leqslant x_U,\ \chi \geqslant 0 \end{aligned} \tag{2.48}$$

Debido a que el objetivo se encuentra acotado por una región de factibilidad con Hessiano definido positivo, el problema QP tiene así solución única. Además, si se

puede encontrar solución para $d_x \neq 0$, el algoritmo SQP procede a llevar la reducción de $\phi(x,\rho)$ tal y como describe el siguiente teorema.

Teorema 15.

Considérese el problema QP descrito por (2.48) donde W^k es definida positiva. Si el problema QP tiene solución con $\chi < 1$, entonces d_x es decreciente en la dirección de la función de mérito $\phi_p(x,\rho) = f(x) + \rho \|c(x)\|_p$ con $p > 1$ para todo valor de $\rho > \bar{\rho}$ eligiéndose $\bar{\rho}$ lo suficientemente grande, tal y como se demuestra en [5].

Mediante esta caracterización del problema QP, y las propiedades que confiere la función de penalización, se establece el siguiente algoritmo, conocido como algoritmo de búsqueda en línea, para la resolución del problema.

Algoritmo 1.

Elegir constantes $\eta \in (0,0{,}5)$, tolerancias de convergencia $\epsilon_1, \epsilon_2 > 0$. Elegir un punto inicial $x_{init}, v_{init}, u_{L_{init}}, u_{U_{init}}$ y un valor inicial de penalización $\rho^{-1} > 0$

Para $k > 0$, mientras $\|d_x\| > \epsilon_1$ y máx$(\|\nabla L(x^k, u_L^k, u_U^k, v^k)\|, \|c(x^k)\|) > \epsilon_2$:

- **Paso 1:** Evaluar $f(x^k)$, $\nabla f(x^k, c(x^k)$, $\nabla c(x^k)$ y los términos del Heassiano.
- **Paso 2:** Resolver el problema de programación QP para calcular el paso de búsqueda d_x. Si $\chi = 1$, parar. Las restricciones son inconsistentes y no es posible progresar.
- **Paso 3:** Establecer $\alpha^k = 1$.
- **Paso 4:** Actualizar $\rho^k = max\{\rho^{k-1}, \|\bar{v}\|_q + \epsilon\}$
- **Paso 5:** Comprobar si se cumple la condición de búsqueda en línea:

$$\phi_p(x^k, \alpha^k d_x, \rho^k) < \phi(x^k, \rho^k) + \eta \alpha^k D_{d_x} \phi_p(x^k, \rho^k) \tag{2.49}$$

- **Paso 6:** Si (2.49) no se satisface, elegir un nuevo valor de α^k y volver al paso 5. Si no, establecer:

$$\begin{aligned} x^{k+1} &= x^k + \alpha^k d_x,\ v^{k+1} = \bar{v} \\ u_L^{k+1} &= \bar{u}_L, u_U^{k+1} = \bar{u}_U \end{aligned} \tag{2.50}$$

Con esta relajación del problema QP, el problema se adapta a muchas de las propiedades propias de un problema de optimización sujeto a restricciones de igualdad. A partir de aquí, son aplicables los siguientes teoremas.

Teorema 16.

Supóngase que las secuencias de valores x^k y $x^k + d_k$ seguidas por el algoritmo (1) se encuentran contenidas en una región cerrada acotada y convexa con $f(x)$ y $c(x)$ uniformemente derivables en primer y segundo grado. Entonces, si W^k es definida positiva y uniformemente acotada, LICQ se cumple, todos los problemas QP en cada iteración son resolubles con $\chi < 1$ y $\rho \|\bar{v}\| + \epsilon$ para un valor $\epsilon > 0$, entonces todos los puntos límite de x^k cumplen con las condiciones KKT.

Teorema 17.

Supóngase que x^o es una solución local del problema (2.34) que satisface LICQ, cumple las condiciones de suficiencia de segundo orden, y es estrictamente complementario de los límites de los multiplicadores.

Asúmase también que las iteraciones generadas por el algoritmo (1) convergen hacia x^o. Bajo estas condiciones, existe una determinada vecindad donde la solución del problema QP (2.48) produce $\chi = 0$ y el conjunto activo de (2.48) es igual que el de (2.34)

Si el conjunto activo permanece invariable en una vecindad en torno al punto solución, es posible aplicar los mismos resultados de convergencia local que en un problema de optimización sujeto a restricciones de igualdad. Si se lleva a cabo una redefinición de v como $v^T = [u_L^T, u_U^T, v^T]$, estos resultados pueden resumirse como:

- Si el conjunto $W^k = \nabla_{xx} L(x^k, v^k)$. Aplicando el algoritmo (1) con x^k y v^k suficientemente cercanas a x^o y v^o, existe una constante C tal que:

$$\left\| \begin{bmatrix} x^k + d_x - x^o \\ v^k + d_v - v^o \end{bmatrix} \right\| \leqslant C \left\| \begin{bmatrix} x^k - x^o \\ v^k - v^o \end{bmatrix} \right\|^2 \tag{2.51}$$

Es decir, la convergencia de x^k y v^k es cuadrática.

- Si W^k es aproximado mediante una actualización cuasi-Newton B^k, entonces x^k converge a x^o a una tasa de convergencia superlineal, es decir,

$$\lim_{k\to\infty} = \frac{\left\|x^k + d_x - x^o\right\|}{\left\|x^k - x^o\right\|} = 0 \tag{2.52}$$

si y sólo si se cumple que

$$\lim_{k\to\infty} \frac{\left\|Z(x^k)^T(B^k - \nabla_{xx}L(x^o, v^o))d_x\right\|}{\|d_x\|} = 0 \tag{2.53}$$

sonde $Z(x)$ es la representación ortonormal del espacio nulo de $A(x) = [\nabla c(x^k) \mid E_L \mid E_U]$ en la vecindad de x^o.

- Si W^k se aproximada mediante actualización de cuasi-Newton B^k cumpliéndose que

$$\lim_{k\to\infty} \frac{\left\|Z(x^k)^T(B^k - \nabla_{xx}L(x^o, v^o))Z(x^k)Z(x^k)^T d_x\right\|}{\|d_x\|} = 0 \tag{2.54}$$

entonces x^k converge a x^o con una tasa de dos pasos superlineales. Gracias a esta propiedad se proporciona la tasa de convergencia típica de la mayoría de los problemas NLP.

Ahora es necesario unir las propiedades de convergencia global y local debido a la necesidad de tomar pasos completos en la vecindad de la solución. Sin embargo, el efecto Maratos ocurre debido a las restricciones de desigualdad [5]. Para ello, es necesario el uso de correcciones de segundo orden así como métodos de búsqueda en línea relacionados para lidiar con este problema y poder dar pasos completos cerca de la solución.

Para la resolución de problemas con restricciones de desigualdad resulta de especial interés la estrategia de búsqueda en línea conocida como *Watchdog*. Dicha estrategia se basa en la ignorancia de la condición de decremento de Armijo y tomar una secuencia de pasos con $\alpha = 1$. El algoritmo resultante de la aplicación de la estrategia Watchdog al algoritmo (1) se describe a continuación.

Algoritmo 2.
Elíjase una constante $\eta \in (0{,}0{,}5)$ y unas tolerancias de convergencia $\epsilon_1, \epsilon_2 > 0$. Elíjase un punto inicial x_{init}, v_{init} con un valor inicial $\rho^{-1} > 0$ del parámetro de penalización.

Para $k > 0$, mientras que $\max(\left\|x^k, u_L^k, u_U^k, v^k\right\|, \left\|c(x^k)\right\|) > \epsilon_2$

- **Paso 1:** Evaluear $f(x^k)$, $\nabla f(x^k)$, $c(x^k)$, $\nabla c(x^k)$ y los términos del Hessiano.
- **Paso 2:** Resolver el problema QP para calcular el paso de búsqueda d_x. Si $\chi = 0$ parar. Las restricciones con inconsistencias no se puedes seguir progresando.
- **Paso 3:** Actualizar $\rho^k = \max\{\rho^{k-1}, \|\bar{v}\|_q \epsilon\}$
- **Paso 4:** Si $i \in \{1, i_w\}$:
  - Establecer $x^{k+i} = x^{k+i-1} + d_x^i$, donde d_x^i es la solución del problema QP en el instante anterior.
  - Evaluar $f(x^{k+i})$, $\nabla f(x^{k+i})$, $c(x^{k+i})$, $\nabla c(x^{k+i})$
  - Si $\phi_p(x^{k+i}, \rho^k) \leqslant \phi_p(x^k, \rho^k) + \eta D_{d_x}\phi_p(x^k, \rho^k)$ aceptar el paso, establecer $k = k + i - 1$, $v^{k+1} = \bar{v}, u_L^{k+1} = \bar{u}_L, u_U^{k+1} = \bar{u}_U$ y volver al paso 1.
- Restablecer $x^{k+i_w} = x^k, k = k + i_w$ y establecer $\alpha^k = 1$
- Comprobar la condición de búsqueda:

$$\phi_p(x^k, \alpha^k d_x, \rho^k) \leqslant \phi_p(x^k, \rho^k) + \eta \alpha^k D_{d_x}\phi_p(x^k, \rho^k). \tag{2.55}$$

- Si (2.55) no se cumple, elegir un nuevo valor de α^k y volver al paso 6.
- Establecer $x^{k+1} = x^k + \alpha^k d_k$, $v^{k+1} = \bar{v}$, $u_L^{k+1} = \bar{u}_L$, $u_U^{k+1} = \bar{u}_U$

2.3.3 Large Scale SQP

Los algoritmos descritos previamente presentan la ventaja de requerir pocas evaluaciones de funciones y sus gradientes para converger, razón por la cual se encuentran ampliamente extendidos. Por otra parte, los algoritmos utilizados para la resolución de problemas QP requieren de una gran cantidad de evaluaciones de funciones algebraicas, haciendo que el coste computacional sea alto para problemas NLP de gran envergadura. Además, debido a las desigualdades, la elección de un adecuado conjunto activo puede ser muy costosa, razón por la cual puede ser conveniente recurrir a métodos de punto interior desarrollados en secciones venideras. En este apartado se llevará a cabo una distinción entre algoritmos para problemas de pequeño y gran tamaño, es decir, problemas con pocos y con muchos grados de libertad.

Problemas con pocos grados de libertad

Los problemas de optimización NLP con pocos grados de libertad se dan frecuentemente en problemas de optimización en tiempo real con modelos estacionarios, problemas de estimación paramétrica, etc. A continuación se describe un procedimiento que puede disminuir considerablemente el coste computacional del problema QP desarrollado en cada iteración.

Primeramente, se comienza caracterizando la solución d_x al problema (2.42) a través de dos componentes, de la forma:

$$d_x = Y^k p_Y + Z^k p_Z \tag{2.56}$$

Donde Z^k es una matriz de rango completo y dimensiones $n \times (n-m)$ que se expande en el espacio nulo de $(\nabla c(x^k))^T$ e Y^k es cualquier matriz tal que $[Y^k \mid Z^k]$ sea no singular, lo cual conduce a pasos normales y tangenciales $Y^k p_Y$ y $Z^k p_Z$ respectivamente. Sustituyendo (2.56) en la ecuación linealizada $c(x^k + \nabla c(x^k)^T d_x) = 0$ el paso normal

puede ser extraído de la expresión:

$$c(x^k)(\nabla(x^k))^T Y^k p_Y = 0 \tag{2.57}$$

Despejando de esta expresión, el paso p_Y queda expresado como:

$$p_Y = -[\nabla c(x^k)^T Y^k]^{-1} c(x^k) \tag{2.58}$$

de tal forma que d_x queda expresada como:

$$d_x = -Y^k[\nabla c(x^k)^T Y^k]^{-1} c(x^k) + Z^k p_Z \tag{2.59}$$

Para determinar la componente tangencial, sustituyendo (2.88) en (2.42) se obtiene el siguiente problema QP de espacio reducido.

$$\begin{aligned} &\min_{p_Z}((Z^k)^T \nabla f(x^k) + \omega^k)^T p_Z + \frac{1}{2} p_Z^T \bar{B}^k p_Z \\ &s.a \\ &\quad x_L \leqslant x^k + Y^k p_Y + Z^k p_Z \leqslant x_U \end{aligned} \tag{2.60}$$

donde $\omega^k = (Z^k)^T W^k Y^k p_Y$ y $\bar{B}^k = (Z^k)^T W^k Z^k$. Para garantizar la factibilidad de la región, el problema (2.60) puede ser modificado, en aras de incluir el modo elástico como:

$$\begin{aligned} &\min_{p_Z}((Z^k)^T \nabla f(x^k) + \omega^k)^T p_Z + \frac{1}{2} p_Z^T \bar{B}^k p_Z + M(\chi + \chi^2/2) \\ &s.a \\ &\quad x_L \leqslant x^k + (1-\chi) Y^k p_Y + Z^k p_Z \leqslant x_U \end{aligned} \tag{2.61}$$

Problema el cual podrá ser resuelto si $\bar{B}^k$ es definida positiva, lo cual puede ser comprobado mediante descomposición de Cholesky. El problema QP tangencial requiere por otro lado que el Hessiano proyectado sea de curvatura positiva, lo cual es condición menos restrictiva que la asunción de espacio nulo.

Además, la solución del problema (2.61) puede ser costosa si existen demasiadas restricciones límite. Para la resolución del problema QP, el solver a menudo realiza una búsqueda en el espacio de variables primales. Primero, se encuentra un punto factible y mediante las iteraciones del problema se reduce el objetivo cuadrático hasta que se satisfacen las condiciones de optimalidad. Sin embargo, las regiones de factibilidad de (2.60) y (2.61) pueden ser difíciles de explorar si el problema se encuentra mal condicionado. Para ello, se ha llevado a cabo el desarrollo de solvers duales, los cuales no permanecen en la región factible de (2.61), sino que encuentran el mímino no restringido del objetivo cuadrático y posteriormente actualizan la solución añadiendo sucesivamente desigualdades que violan las restricciones. De esta forma el problema QP permanece factible de manera dual y el solver QP termina cuando el problema primal es factible. Se describe a continuación el conocido como algoritmo de espacio reducido como modificación del algoritmo (1).

Algoritmo 3.
Elegir una constante $\eta \in (0,0,5)$, tolerancias de convergencia ϵ_1, ϵ_2, un punto inicial $x_{init}, v_{init}, y_{L_{init}}, u_{U_{init}}$ y un valor inicial del parámetro de penalización $\rho^{-1} > 0$.

Para $k \geqslant 0$, mientras que $\|d_x\| > \epsilon_1$ y máx$(\left\|\nabla L(x^k, u_L^k, u_U^k, v^k\right\|, \left\|c(x^k)\right\|) > \epsilon_2$:

- **Paso 1:** Evaluar $f(x^k)$, $\nabla f(x^k)$, $c(c^k)$, $\nabla c(x^k)$ y los términos del Hessiano.
- **Paso 2:** Calcular el paso normal p_Y
- **Paso 3:** Calcular el paso tangencial p_Z así como los límites de los multiplicadores límite $\bar{uL}$ y $\bar{u_U}$ del problema QP. Si $\chi = 1$ parar. Las restricciones son inconsistentes y no es posible progresar.
- **Paso 4:** Calcular el paso de búsqueda $d_x = Z^k p_Z + Y^k p_Y$ así como los multiplica-

dores para las restricciones de igualdad:

$$\bar{v} = -((Y^k)^T \nabla c(x^k))^{-1} (Y^k)^T (\nabla f(x^k) + \bar{u}_U - \bar{u}_L) \tag{2.62}$$

- **Paso 5:** Establecer $\alpha^k = 1$.
- **Paso 6:** $\rho^k = \max\{\rho^{k-1}, \|\bar{v}\|_q + \epsilon\}$.
- **Paso 7:** Comprobar la condición de búsqueda en línea

$$\phi(x^k + \alpha^k d_x, \rho^k) \leqslant \phi_p(x^k, \rho^k) + \eta \alpha^k D_{d_x} \phi_p(x^k, \rho^k) \tag{2.63}$$

- **Paso 8:** Si (2.63) no se satisface. Elegir un nuevo α^k y volver a 7. Si sí se satisface, establecer $x^{k+1} = x^k + \alpha^k d_x$, $v^{k+1} = \bar{v}$, $u_L^{k+1} = \bar{u}_L$, $u_U^{k+1} = \bar{u}_U$.

Problemas con muchos grados de libertad

Los problemas de optimización con muchos grados de libertad se encuentran frecuentemente en problemas de estimación de estado, reconciliación de datos, y problemas de optimización dinámica. Para ellos, es necesario considerar el problema Full Space SQP, el cual explota la escasez de las matrices Hessiana y Jacobiana. Para la construcción de algoritmos SQP que traten problemas de gran escala , son dos los aspectos fundamentales a considerar. En primer lugar, el algoritmo ha de garantizar que la matriz KKT puede ser factorizada. En segundo lugar, es necesario desarrollar un algoritmo QP eficiente que explote la escasez del problema, especialmente en los procesos de actualización de los conjuntos activos.

Todo ello hace que la solución de un problema QP de gran escala sea por lo general un problema complicado, debido en gran parte a la dificultad a la hora de elegir un conjunto activo adecuado. En este sentido, sistemas lineales de la forma de (2.40) son resueltos basándose en la factibilidad de las variables primales y duales, es decir, las variables de paso d_x dentro de los límites y los multiplicadores límite u_L y u_U no nega-

tivos. En estos métodos, un elemento clave es la adición y eliminación de restricciones activas y el recálculo de variables.

Considérese el sistema KKT (2.40) , el cual puede ser representado como $Kz = r$, donde $z^T = [d_x^T, \bar{v}^T, \bar{u}^T]$ y r el término de la derecha de (2.40). Si se actualiza el conjunto activo mediante la adición de restricciones lineales, denotadas como $\hat{e}_i^T d_x = b_i$ con su correspondiente multiplicador escalar $\bar{u}_i$, entonces el sistema (2.40) es aumentado, dando lugar al siguiente sistema lineal:

$$\begin{bmatrix} K & \hat{e}_i \\ \hat{e}_i^T & 0 \end{bmatrix} \begin{bmatrix} z \\ \bar{u}_i \end{bmatrix} = \begin{bmatrix} r \\ b_i \end{bmatrix} \tag{2.64}$$

donde $\hat{e}_i^T = [e_i^T, 0, ..., 0]$ tiene la misma dimensión que z. Utilizando el método complemento de Schur, es posible llevar a cabo una factorización previa de K y computar la solución del sistema aumentado a partir de:

$$\begin{aligned} \bar{u}_i &= -(\hat{e}_i^T K^{-1} \hat{e}_i)^{-1} (b_i - \hat{e}_i^T K^{-1} r) \\ z &= K^{-1}(r - \hat{e}_i \bar{u}_i) \end{aligned} \tag{2.65}$$

y computar la solución del sistema aumentado mediante el uso también del método complemento de Schur.

2.3.4 Métodos de Punto Interior

En esta sección se describen los conocidos como métodos de punto interior para la resolución de problemas NLP. Considérese el problema NLP dado por la expresión:

$$\begin{aligned} &\min_x f(x) \\ &s.a \\ &\quad c(x) = 0, x > 0 \end{aligned} \tag{2.66}$$

Para poder tratar con los límites inferiores no negativos de la variable x, se desarrolla una barrera logarítmica y se considera la solución de una secuencia del problema NLP con restricciones de igualdad de la forma:

$$\begin{aligned} &\text{mín}\, \varphi_{\mu_l}(x) = f(x) - \mu_l \sum_{i=1}^{n_x} ln(x_i) \\ &s.a \\ &\quad c(x) = 0,\ x > 0 \end{aligned} \tag{2.67}$$

Donde l es el contador de la secuencia y $\lim_{l\to\infty} \mu_l = 0$. La barrera logarítmica se encuentra no acotada en el punto $x = 0$, y por lo tanto el camino hallado por el algoritmo debe alojarse en una región de punto estrictamente positivos. Además, si $c(x) = 0$ satisface LICQ, entonces la solución de (2.67) satisface las siguientes condiciones de primer orden:

$$\begin{aligned} &\nabla f(x) + \nabla c(x)v - \mu X^{-1} e = 0 \\ &c(x) = 0 \end{aligned} \tag{2.68}$$

Donde $X = \text{diag}\{x\}$, $e = [1,1,...,1]^T$ y el vector solución $x(\mu) > 0$. A las ecuaciones (3.28) se las conoce como condiciones primales de optimalidad, para denotar la ausencia de multiplicadores para las restricciones de desigualdad. Una de las principales ventajas de la utilización de barrera es el hecho de que no es necesario tomar ninguna decisión acerca de los valores de los conjuntos activos. El siguiente teorema relaciona la secuencia de soluciones de (2.67) con la solución del problema NLP (2.66).

Teorema 18.

Considérese el problema descrito por (2.66), con $f(x)$ y $c(x)$ diferanciables al menos dos veces, y un punto x^o un minimizador restringido de manera local del problema (2.66). Considérese además que la región de factibilidad del problema (2.66) es de interior estricto y que se cumplen las siguientes condiciones de suficiencia de optimalidad en x^o:

- x^o es un punto KKT.
- Se cumple LICQ.

- Se cumple complementariedad estricta para x^o y para los multiplicadores límite u^o que satisfacen las condiciones KKT.
- Para $L(x,v) = f(x) + c(x)^T v$ existe $\omega > 0$ tal que $q^T \nabla_{xx} L(x^o, v^o) q \geqslant \omega \|q\|^2$ para los multiplicadores de las restricciones de igualdad v^o que satisfacen las condiciones KKT y todos los espacios nulos de restricciones activas distintos de cero.

Si ahora se resuelve la secuencia de problemas (2.67) con $\mu_l \to 0$ entonces:

- Existe una subsecuencia de minimizadores x_{μ_l} de la función de barrera que convergen a x^o.
- Para cada una de las subsecuencias convergentes, la correspondiente secuencia de aproximaciones de los multiplicadores de barrera $\mu X^{-1} e$ y $v(\mu)$ se encuentra acotado y converge a los multiplicadores u^o y v^o que satisfacen las condiciones KKT para x^o.
- Existe una función vectorial continuamente diferenciable $x(\mu)$ de los minimizadores de (2.67) en una vecindad de $\mu = 0$.
- $\lim_{\mu \to 0^+} x(\mu) = x^o$.
- $\|x(\mu_l) - x^o\| = O(\mu_l)$.

Los gradientes de la función de barrera se encuentran no acotados en los límites de las restricciones, lo cual conduce a superficies solución muy mal condicionadas para $\varphi(x)$. Esto se puede observar también a partir del Hessiano del problema (3.67):

$$W(x,v) = \nabla^2 f(x) + \sum_{j=1}^{m} \nabla^2 c_j(x) v_j + \mu X^{-2} \tag{2.69}$$

De esta forma, ya que debido al comportamiento extremadamente no lineal de la función de barrera, la obtención directa de la solución a problemas de barrera suele ser complicada. Además, para el subconjunto de variables que son cero en la solución de (3.66), el Hessiano se vuelve no acotado conforme μ tiende a cero.

Por otro lado, si se considera la cantidad $q^T W(x^o, v^o)q$ con direcciones q pertenecientes al espacio nulo del Jacobiano del conjunto de restricciones activas, entonces los términos no acotados ya no se encuentran presentes y los condicionantes del Hessiano reducido no se ven afectados por los términos de barrera.

Por ello, se hace patente la necesidad de modificar las condiciones primales de optimalidad para dar lugar al conocido como sistema primal-dual. En éste, se define un conjunto de nuevas variables duales junto a la ecuación $Xu = \mu e$ y se reemplaza la contribución de la barrera para dar lugar a:

$$\begin{aligned} &\nabla f(x) + \nabla c(x)v - u = 0 \\ &Xu = \mu e \\ &c(x) = 0 \end{aligned} \tag{2.70}$$

Esta sustitución y linealización simplifica los términos no lineales de la barrera y conduce a un relajación de las condiciones KKT. Por tanto, se puede ver esta modificación de la barrera como la aplicación de un método homotópico a las ecuaciones primal-duales con el parámetro homotópico μ, donde los multiplicadores u se corresponden con los multiplicadores KKT para las restricciones acotadas de (2.66) conforme $\mu \to 0$. De esta forma, $\mu = 0$ así como $x, u \geqslant 0$ son las condiciones KKT del problema original descrito en (2.66). Esto define el paso fundamental de los métodos de punto interior.

Ahora se considera una estrategia basada en métodos de Newton para la resolución de (2.66) a través del sistema primal-dual. Primeramente, utilizando los elementos de las ecuaciones primar-duales de (2.70), se define el error de las condiciones de optimalidad del problema de punto interior como:

$$E_\mu(x, v, u) = \text{máx}\{\|\nabla f(x) + \nabla c(x)v - u\|_\infty, \|c(x)\|_\infty, \|Xu - \mu e\|_\infty\} \tag{2.71}$$

Para comprobar que el algoritmo de barrera que será desarrollado ha terminado se ha de cumplir que:

$$E_o(\tilde{x}_o, \tilde{v}_o, \tilde{u}_o) \leqslant \epsilon_{tol} \tag{2.72}$$

Donde ϵ_{tol} es la tolerancia establecida por el diseñador. A continuación se describe un algoritmo donde el problema de la barrera es resuelto de manera aproximada en un lazo interno con un valor fijo de μ.

Algoritmo 4.
Elegir constantes $\epsilon_{tol} > 0$, $k_\epsilon > 0$, $k_\mu \in (0,1)$ y $\theta_\mu \in (1,2)$. Elegir un punto inicial $x_{init} > 0$, v_{init}, u_{init} y un valor inicial μ_{init} como parámetro de barrera.

Para $l > 0$ mientras que $E_o(\tilde{x}_{\mu_l}, \tilde{v}_{\mu_l}, \tilde{u}_{\mu_l}) \leqslant \epsilon_{tol}$:

- Encontrar una solución aproximada del problema de barrera $\tilde{x}_{\mu_l}, \tilde{v}_{\mu_l}, \tilde{u}_{\mu_l}$ tal que se satisfaga que:

$$E_{\mu_l}(\tilde{x}(\mu_l), \tilde{v}(\mu_l), \tilde{u}(\mu_l)) \leqslant k_\epsilon \mu_l \tag{2.73}$$

- Establecer un nuevo parámetro de barrera:

$$\mu_{l+1} = \text{máx}\left\{\frac{\epsilon_{tol}}{10}, \text{mín}\left\{k_\mu \mu_l, \mu_l^{\theta_\mu}\right\}\right\} \tag{2.74}$$

El parámetro de barrera es así decrementado en un ratio superlineal, lo cual gobierna el ratio de convergencia del algoritmo. Por otro lado, es importante percatarse de que la actualización llevada a cabo en (2.74) no permite que μ sea muy pequeña para evitar así dificultades numéricas para la resolución del problema. Valores típicos de las constante suelen ser $k_\epsilon = 10$, $k_\mu = 0{,}2$ y $\theta_\mu = 1{,}5$.

Sistema Primal-Dual

Para la resolución del problema descrito por (2.67) para un determinado valor de μ_l, se considera un método de Newton de búsqueda lineal aplicado a las ecuaciones primal-duales descritas por (2.70), con k elegido como contador de iteración.

Dada un determinada iteración en un punto (x^k, v^k, u^k) con $x^k, u^k > 0$, y direcciones de búsqueda (d_x^k, d_v^k, d_u^k) se obtiene la siguiente linealización de (2.70):

$$\begin{bmatrix} W^k & \nabla c(x^k) & -I \\ \nabla c(x^k)^T & 0 & 0 \\ U^k & 0 & X^k \end{bmatrix} \begin{bmatrix} d_x^k \\ d_v^k \\ d_u^k \end{bmatrix} = - \begin{bmatrix} \nabla f(x^k) + \nabla c(x^k) v^k - u^k \\ c(x^k) \\ X^k u^k - \mu_l e \end{bmatrix} \tag{2.75}$$

Donde $W^k = \nabla_{xx} L(x^k, v^k)$, $X = \text{diag}\{x\}$, $U = diag\{u\}$. El sistema sistema anterior puede ser simplificado, mediante la resolución primera del sistema lineal:

$$\begin{bmatrix} W^k + \sum^k & \nabla c(x^k) \\ \nabla c(x^k)^T & 0 \end{bmatrix} \begin{bmatrix} d_x^k \\ d_v^k \end{bmatrix} = - \begin{bmatrix} \nabla \varphi_\mu(x^k) + \nabla c(x^k) v^k \\ c(x^k) \end{bmatrix} \tag{2.76}$$

Donde $\sum^k = (X^k)^{-1} U^k$. Por otro lado, el vector d_u^k se obtiene como:

$$d_u^k = \mu_l (X^k)^{-1} e - u^k - \Sigma^k d_x^k \tag{2.77}$$

Para que se mantenga la no singularidad del sistema, es posible llevar a cabo una modificación de (2.76) como:

$$\begin{bmatrix} W^k + \Sigma^k \text{`} \delta_w I & \nabla c(x^k) \\ \nabla c(x^k)^T & -\delta_A I \end{bmatrix} \begin{bmatrix} d_x^k \\ d_v^k \end{bmatrix} = - \begin{bmatrix} \nabla \varphi_\mu(x^k) + \nabla c(x^k) v^k \\ c(x^k) \end{bmatrix} \tag{2.78}$$

Donde δ_w y δ_A deben ser actualizadas para asegurar que (2.78) tiene la inercia adecuada. Para las direcciones de búsqueda de (2.78) y (2.77), se puede aplicar una búsqueda en línea con pasos de tamaño $\alpha^k, \alpha_u^k \in (0,1]$ para la obtención de las siguientes iteraciones:

$$\begin{aligned} x^{k+1} &= x^k + \alpha^k d_x^k \\ v^{k+1} &= v^k + \alpha^k d_v^k \\ u^{k+1} &= u^k + \alpha_u^k d_u^k \end{aligned} \tag{2.79}$$

Dado que x y u son ambos positivos en la solución al problema de barrera, y $\varphi_\mu(x)$ sólo es definida para valores positivos de x, esta propiedad ha de mantenerse para todas las iteraciones. Por ello, se aplica la conocida como regla del paso en el límite:

$$\begin{aligned} \alpha_{max}^k &= \text{máx}\left\{\alpha \in (0,1] : x^k + \alpha d_x^k \geqslant (1-\tau_l)x^k\right\} \\ \alpha_u^k &= \text{máx}\left\{\alpha \in (0,1] : u^k + \alpha d_u^k \geqslant (1-\tau_l)u^k\right\} \end{aligned} \tag{2.80}$$

Con el parámetro τ_l igual a:

$$\tau_l = \text{máx}\left\{\tau_{min}, 1-\mu_l\right\} \tag{2.81}$$

donde τ_l toma típicamente valores cercanos a 1, y $\tau_l \to 1$ cuando $\mu_l \to 0$.

2.3.5 Resolución mediante Regiones de Confianza

En este apartado se aplicarán los métodos de región de confianza pertenecientes a la familia de métodos de Newton para la resolución del problema NLP descrito por (3.67).

Considérense dos problemas de región de confianza separados para el cálculo de

los pasos normal y tangencial, de tal forma que $d_x = p_N + p_T$. Ambos pasos son de carácter escalar, y una determinada región de confianza se aplica a estos pasos escalados para conseguir pasos más grandes para los elementos del vector lejos de los límites, $x \geqslant 0$, y pasos más pequeños para elementos cercanos a los límites. Estos pasos escalados pueden ser definidos como:

$$\begin{aligned} \tilde{d}_x &= (X^k)^{-1} dx \\ \tilde{p}_N &= (X^k)^{-1} p_N \\ \tilde{p}_T &= (X^k)^{-1} p_T \end{aligned} \tag{2.82}$$

Donde la región de confianza es aplicada directamente a $\tilde{p}_N$ y $\tilde{p}_T$. El subproblema de región de confianza para el paso normal puede ser expresado como:

$$\begin{aligned} &\min_{p_N} \left\| c(x^k) + \nabla c(x^k)^T p_N \right\|_2^2 \\ &s.a \\ &\quad \|\tilde{p}_N\|_2 \leqslant \chi_N \Delta \\ &\quad \tilde{p}_N \geqslant \frac{-\tau}{2} e \end{aligned} \tag{2.83}$$

Por otro lado, es posible definir un subproblema que calcule la dirección normal y tangencial de manera combinada, $d_x = d_N + d_T$ directamente. Debido a que el problema normal ya ha reducido la violación de las restricciones de igualdad, el subproblema para el paso tangenical puede ser expresado como:

$$\begin{aligned} &\min_{d_x} (\nabla f(x^k) - \mu (X^k)^{-1} e) d_x + \frac{1}{2} d_x^T (W^k + \Sigma^k) d_x \\ &s.a \\ &\quad \left\| (X^k)^{-1} d_x \right\| \leqslant \Delta \\ &\quad \nabla c(x^k)^T d_x = \nabla c(x^k)^T p_N \\ &\quad d_x \geqslant -\tau x^k \end{aligned} \tag{2.84}$$

Por otro lado, el correspondiente problema, esta vez escalado viene dado por:

$$\begin{aligned}
&\min_{d_x}(\nabla f(x^k)X^k - \mu e)^T \tilde{d}_x^T + \frac{1}{2}\tilde{d}_x^T(X^k W^k X^k + X^k \Sigma^k X^k)\tilde{d}_x \\
&s.a \\
&\|\tilde{d}_x\| \leqslant \Delta \\
&\nabla c(x^k)^T X^k \tilde{d}_x = \nabla c(x^k)^T X^k \tilde{p}_N \\
&\tilde{d}_x \geqslant -\tau e
\end{aligned} \tag{2.85}$$

En este problema, al igual que en el subproblema para el paso normal, si la desigualdad del paso escalado es violada por la solución aproximada, es posible aplicar un procedimiento de recuperación de d_x.

2.3.6 Métodos Anidados

En esta sección se analizan los una serie de métodos conocidos como método anidados, lo cuales a pesar de no encontrarse tan extendidos resultan de especial interés en problemas NLP con objetivos y restricciones no lineales, en los cuales es importante mantenerse siempre lo más cerca posible de la región de factibilidad en la secuencia de iteraciones.

El problema a considerar, primeramente se lleva a cabo una partición de las variables, con el objetivo de tratar con las partes restringidas del problema:

$$\begin{aligned}
&\min f(x) \\
&s.a \\
&\quad c(x) = 0 \\
&\quad x_L \leqslant x \leqslant x_U
\end{aligned} \tag{2.86}$$

En este problema, las variables se dividen es tres categorías, a saber:

- Variables No básicas

- Variables Básicas
- Variables Superbásicas

Las variables no básicas son aquellas que se establecen en el límite en el punto de solución óptima, mientras que las variables básicas pueden ser determinadas a partir de $c(x) = 0$ una vez que se han fijado el resto de variables. En los problemas de carácter lineal, la partición de variables en básicas y no básicas es suficiente para garantizar la convergencia del algoritmo. Sin embargo, en problemas donde $f(x)$ o $c(x)$ son no lineales, es necesario la partición en variables superbásicas, las cuales, al igual que las variables no básicas, conducen el algoritmo de optimización, pero su valor óptimo no se encuentra en los límites. Para el desarrollo de estos métodos, considérense las condiciones KKT de primer orden:

$$\begin{aligned} &\nabla_x L(x^o, u^o, v^o) = \nabla f(x^o) + \nabla c(x^o)^T v^o - u_L^o + u_U^o = 0 \\ &c(x^o) = 0 \\ &0 \leqslant u_L^o \perp (x^o - x_L) \geqslant 0 \\ &0 \leqslant u_U^o \perp (x_U - x^o) \geqslant 0 \end{aligned} \tag{2.87}$$

Ahora, las variables son reordenadas como básicas, no básicas y superbásicas, de tal forma que $x = [x_B^T, x_S^T, x_N^T]$, partición la cual se deriva de la información local y la cual puede cambiar a lo largo del proceso iterativo. Así mismo, se particiona el Jacobiano de las restricciones como:

$$\nabla c(x)^T = [A_B(x) \mid A_S(x) \mid A_N(x)] \tag{2.88}$$

Y el gradiente de la función objetivo como:

$$\nabla f(x)^T = [f_B(x)^T \mid f_S(x)^T \mid f_N(x)^T] \tag{2.89}$$

A partir de estas definiciones, las correspondientes condiciones KKT pueden ser escritas como:

$$\begin{aligned} & f_S(x^o) + A_S(x^o)^T v^o = 0 \\ & f_N(x^o) + A_N(x^o)^T v^o - u_L^o + u_U^o = 0 \\ & f_B(x^o) + A_B(x^o)^T c^o = 0 \\ & c(x^o) = 0 \\ & x_{N,i}^o = x_{N,L,i} \;\; x_{N,i}^o = x_{N,U,i} \end{aligned} \tag{2.90}$$

Las ecuaciones descritas por (2.90) también proporcionan información acerca de los gradientes reducidos, lo cual da idea de cómo x_B cambia con respecto a x_S. Considerando x_B como función implícita de x_S, y diferenciando $c(x) = 0$ se tiene:

$$0 = A_B dx_B + A_S dx_S \Longrightarrow \left(\frac{dx_B}{dx_S}\right)^T = -(A_B)^{-1} A_S \tag{2.91}$$

Y el gradiente reducido puede ser descrito como:

$$\begin{aligned} & \frac{df(x_B(x_S), x_S)}{dx_S} = f_S + \left(\frac{dx_B}{dx_S}\right) f_B \\ & = f_S - A_S^T (a_B)^{-T} f_B = f_S A_S^T v \end{aligned} \tag{2.92}$$

Además, se define la matriz Z_p que pertenece al espacio nulo de $\nabla c(x)^T$ como:

$$Z_P = \begin{bmatrix} -A_B^{-1} A_S \\ I \\ 0 \end{bmatrix} \tag{2.93}$$

Así pues el Hessiano reducido puede ser expresado como $Z_P^T = \nabla_{xx} L(x, v) Z_P$. A partir de estos conceptos, el método de optimización por anidamiento considera la solución de los subproblemas acotados en el espacio de las variables superbásicas, des-

crito por:

$$\begin{aligned} &\min_{x_S} f(x_B(x_S), x_S, x_N) \\ &s.a \\ &\quad x_{S,L} \leqslant x_S \leqslant x_{S,U} \end{aligned} \tag{2.94}$$

Un algoritmo básico para la resolución del problema siguiendo la estrategia de anidamiento, conocido como algoritmo GRG (Generalized Reduced Gradient),es:

Algoritmo 5.
Establecer $l = 0$, seleccionar los parámetros del algoritmo incluyendo la tolerancia de convergencia ϵ_{TOL}y el tamaño del paso mínimo α_{min}. Elegir un punto de inicio factible llevando a cabo primeramente la partición del problema y reordenando x en los conjuntos indexados B^0, S^0 y N^0 para las variables básicas, superbásicas, y no básicas respectivamente x_S^o, x_N^o, x_B^o.

Para $l \geqslant 0$, y mientras $\|\nabla_x L(x^l, v^l, u^l)\| > \epsilon_{TOL}$:

- **Paso 1:** Determinar x^{l+1} del problema (2.94). Para $k \geqslant 0$, mientras que $\left\| \frac{df(x_B(x_S^{k,l}), x_S^{k,l})}{dx_S} \right\| \geqslant \epsilon_{TOL}$, en cada una de las iteraciones de (3.94), seguir los siguientes pasos:
  - Intentar resolver $c(x_B, x_S(\alpha), x_N^l) = 0$ para encontrar $x_B(\alpha)$.
  - Si no se encuentra solución debido a que A_B se vuelve mal condicionada, ir al paso 3.
  - Si $x_B(\alpha) \notin [x_B, L, x_B, U]$ o $f(x_B(\alpha), x_S(\alpha), x_N^l)$ no es significativamente más pequeño que $f(x_B^{k,l}, x_S^{k,l}, x_N^l)$, reducir el tamaño de α. Si $\alpha \leqslant \alpha_{min}$ para, con fallo de la búsqueda en línea.
  - Si no, establecer $x_B^{k+1,l} = x_B(\alpha)$ y establecer $x_S^{k+1,l} = x_S(\alpha)$.
- **Paso 2:** en la iteración x^{l+1} calcular la estimación de los multiplicadores v^{l+1} y u_L^{l+1} y u_U^{l+1}.

- **Paso 3:** Actualizar los conjuntos B^{l+1}, S^{l+1}, N^{l+1} y la repartición de variables para los siguientes casos:

  Si x_B no puede ser encontrada debido a fallo de convergencia, reparticionar los conjuntos B y S. Si $x_B^{k,l} \notin [x_B, L, x_B, U]$, reparticionar los conjuntos B y N. Si u_L^{l+1} o u_U^{l+1} tienen elementos negativos, reparticionar S y N. Si x_S tiene elementos en los límites, reparticionar S y N.

Una vez que el algoritmo ha encontrado la correcta partición de variables, ésta procede a actualizar x_S utilizando gradientes reducidos en una iteración tipo Newton en una búsqueda en línea.

Por otro lado, una actualización efectiva de la partición es esencial para los casos en los que el paso 3 falla y asegurar una eficiente actuación del algoritmo. Seguir el algoritmo (5) e ignorar las variables no básicas conduce a la toma de pasos horizontales de las variables superbásicas. Cada uno de los pasos requiere la existencia de solución a la ecuación de restricción , tomando pasos verticales para las variables básicas. En algún momento, el algoritmo terminará en un punto donde la matriz A_B es singular. En la mayoría de las implementaciones, la repartición de los conjuntos B y S en el paso 3 es gobernada por la actualizaciones heurísticas que pueden conducir a una actuación eficiente, pero sin tener bien definidas las condiciones de convergencia.

Por otro lado, la repartición entre los conjuntos S y N puede a veces ser tratada a través de una solución eficiente del problema (2.94) y liberando sistemáticamente las variables acotadas. De manera alternativa, es posible considerar un subproblema con restricciones límite ligeramente más grandes dado por:

$$\begin{aligned} &\min_{x_S, x_N} f(x_B(x_S, x_N), x_S x_N) \\ &s.a \\ &\quad x_{S,L} \leqslant x_S \leqslant x_{S,U} \\ &\quad x_{N,L} \leqslant x_N \leqslant x_{N,U} \end{aligned} \tag{2.95}$$

Donde las variables básicas y no básicas son consideradas de manera conjunta en el paso 1 del algoritmo (5).

Métodos de Proyección de Gradiente para Problemas con Restricciones Límite

El primer paso del algoritmo (5) requiere la resolución del problema (2.94) para x_S, o el problema (2.95). Ambos problemas pueden ser representados mediante el siguiente problema NLP con restricciones límites:

$$\begin{aligned} &\min_{z} f(z) \\ &s.a \\ &\quad z_L \leqslant z \leqslant z_U \end{aligned} \tag{2.96}$$

Para resolver (2.96) se han de introducir una nueva serie de conceptos. Las restricciones de desigualdad necesitan ser tratadas mediante el concepto de proyección de gradiente. Se define el operador de proyección para el vector z como el punto más cercano a z en la región factible Ω, es decir:

$$Z_P = P_\Omega(z) = arg\min_{y \in \Omega} \|y - z\| \tag{2.97}$$

Si la región de factibilidad es definida simplemente por los límites de z, $P(z)$ puede ser expresado como:

$$P(z) = \begin{cases} z_i \; si \; z_{L,i} < z_i z_{U,i} \\ z_{L,i} \; si \; z_i \leqslant z_{L,i} \\ z_{U,i} \; si \; z_{U,i} \leqslant z_i \end{cases} \tag{2.98}$$

Los métodos de proyección de gradientes aúnan ambos proyección de gradiente y pasos proyectados de Newton. A partir de aquí, es posible definir las nuevas iteracio-

nes basadas en gradiente proyectado como:

$$z(\alpha) = P(z - \alpha \nabla f(z)) \tag{2.99}$$

donde α es el parámetro que define el tamaño del paso. Se define el punto de Cauchy como el primer minimizador local de $f(z(\alpha))$ con respecto a α, es decir $z_c = z(\alpha_c)$ donde $\alpha_c = arg\,\text{mín}\, f(z(\alpha))$. Considérese el siguiente teorema.

Teorema 19.
Sea $f(z)$ continuamente diferenciable. Un punto $z^o \in \Omega$ es estacionario para el problema (2.96) si y sólo si:

$$z^o = P(z^o - \alpha \nabla f(z^o)) \forall \alpha \geqslant 0 \tag{2.100}$$

Estos conceptos conducen al siguiente algoritmo de proyección de gradiente.

Algoritmo 6.
Elegir un punto inicial z^o, una tolerancia de convergencia ϵ_{TOL}, y un parámetro de restricción de tamaño de paso $\beta < 1$.

Para $k \geqslant 0$, con iteraciones z^k y valores $f(z^k)$, y $\nabla f(z^k)$, mientras que $\|z^o - P(z^o - \alpha \nabla f(z^o))\| > \epsilon_{TOL}$:

- **Paso 1:** Encontrar el entero más pequeño $j \geqslant 0$ para el cual $\alpha(\beta)^k$ y $z(\alpha) = P(z^k - \alpha \nabla f(z^k))$ satisface el teorema de desigualdad de Armijo,

$$f(z(\alpha)) \leqslant f(z^k) + \delta \nabla f(z^k)^T (z(\alpha) - z^k) \tag{2.101}$$

Donde $\delta \in (0, 1/2]$

- **Paso 2:** Establecer $z^{k+1} = z(\alpha)$ y evaluar $f(z^{k+1})$ y $\nabla f(z^{k+1})$

Este algoritmo es globalmente convergente, pero debido a que se basa única y exclusivamente en información del gradiente, converge de manera lenta. Para acelerar la tasa de convergencia, podría pensarse en la mejora del algoritmo mediante la utilización de un paso tipo Newton $p_N = -(\bar{B}^k)^{-1}\nabla f(z^k)$, donde $\bar{B}^k$ es una matriz definida positiva que aproxima $\nabla_{zz} f(z^k)$. Sin embargo, una vez que ha sido proyectado dentro de unos límites, el paso no ha de satisfacer necesariamente las condiciones de descenso y puede conducir a un fallo de búsqueda.

En contrapartida, si el conjunto activo es conocido en la solución, es posible reordenar las variables y reparticionar como $z^k = [(z_I^k)^T, (z_A^k)^T]^T$, donde $z_{A,i}^k$ se encuentra en sus límites y $z_{I,i}^k \in (z_{I,L,i}, z_{I,U,i})$. Si se define la matriz de proyección como $Z_I(z^k) = [I_{nI} \mid 0]^T$ y el Hessiano modificado como:

$$B_R^k = \begin{bmatrix} Z_I^T \bar{B}^k Z_I & 0 \\ 0 & I \end{bmatrix} \tag{2.102}$$

Entonces el paso modificado puede ser escrito como:

$$z(\alpha) = P(z^k - \alpha(B_R^k)^{-1}\nabla f(z^k)) \tag{2.103}$$

Cuando nos encontramos cerca de z^o el conjunto activo correcto es conocido, lo cual conduce a pasos de gradiente proyectado para z_A e información de segundo orden para z_I. Por otro lado, lejos de la solución la repartición de z debe ser relajada y el conjunto de variables activas deber ser sobrestimado. Esto se puede realizar mediante la definición de un conjunto activo ε en la iteración k donde $\varepsilon^k > 0$, y donde z_A^k viene

dado por:

$$z_{U,A,i} - z^k_{A,i} \leqslant \varepsilon^k \;||\; Z^k_{A,i} - z_{L,A,i} \leqslant \varepsilon^k \tag{2.104}$$

Donde z_I incluye las restantes n_I variables. Las cantidades B_R y $z(\alpha)$ son ahora definidas con respecto a esta nueva repartición de variables. Además, con la siguiente elección de ε^k se obtiene una importante propiedad por la cual el conjunto activo en la solución permanece fijado para todo $\left\|z^k - z^o\right\| \leqslant \delta$ para $\delta > 0$ si se mantiene complementaridad estricta en z^o. Definiendo

$$\varepsilon^k = \text{mín}\left[\left\|z^k - P(z^k - \nabla f(z^k))\right\|, \min_i (z_{U,1} - z_{L,i})/2\right] \tag{2.105}$$

La repartición de variables permanece sin cambios en una determinada vecindad en torno al punto solución. Debido a esto, las proyecciones de Newton y cuasi-Newton pueden ser desarrolladas para garantizar una rápida convergencia. El algoritmo resultante se describe a continuación.

Algoritmo 7.
Elegir un punto inicial z_{init}, tolerancia de convergencia ϵ_{TOL} y un parámetro de restricción de tamaño de paso $\beta < 1$.

Para $k \geqslant 0$ con iteración z^k y valores $f(z^k)$ y $\nabla f(z^k)$, y $\bar{B}^k$, mientras que $\|z^o - P(z^o - \alpha\nabla f(z^o))\| > \epsilon_{TOL}$:

- **Paso 1:** Evaluar ε^k de (2.105), determinar el conjunto activo ε, y reordenar la repartición de variables z^k de acuerdo con (2.104). Calcular B^k_R.
- **Paso 2:** Encontrar el entero más pequeño $j \geqslant 0$ para el cual $\alpha = \beta^j$ y $z(\alpha)$ sea evaluada por (2.103) y satisface la desigualdad de Armijo:

$$f(z(\alpha)) \leqslant f(z^k) + \delta\nabla f(z^k)^T(z(\alpha) - z^k) \tag{2.106}$$

donde $\delta \in (0, 1/2]$.

- **Paso 3:** Establecer $z^{k+1} = z(\alpha)$ y evaluar $z(z^{k+1})$, $\nabla z(z^{k+1})$, y $\bar{B}^{k+1}$

Las propiedades de convergencia del algoritmo (7) pueden resumirse como:

1. El algorimto es globalmente convergente. Cualquier punto límite es un punto estacionario.
2. Si el punto estacionario z^o satisface las condiciones de estricta complementariedad, entonces el conjunto activo y la matriz asociada Z_I permanecen constantes para todo valor de $k > k_{init}$ con k_{init} suficientemente grande.
3. Si se utiliza Hessiano exacto y $Z_I^T \nabla_{zz} f(z^o) Z_I$ es definida positiva, entonces la convergencia global es cuadrática.

2.4 Métodos Simultáneos de Optimización Dinámica

2.4.1 Introducción

En las formulaciones tradicionales de controladores MPC tradicionales basados en sistemas y modelos lineales, o en su defecto, linealizados, los modelos de predicción podían obtenerse mediante el uso de álgebra lineal a partir de la descripción del sistema en su modelo en espacio de estado. En el caso de sistemas no lineales, será necesario obtener de alguna forma un modelo de predicción a partir del cual sea posible, dado el estado actual del sistema, predecir el estado siguiente del mismo. Para esta labor, han sido muchas las alternativas utilizadas tradicionalmente, como el uso de métodos de Runge-Kutta para el cálculo del estado del sistema en el siguiente punto de muestreo o temporal considerado.

En esta sección se presentan una serie de métodos, basados en optimización di-

námica, desde métodos secuenciales hasta métodos basados en *Multiple-Shooting* y en los cuales no se requiere la utilización de un solver DAE (Ecuaciones Algebraico-Diferenciales) empotrado. Para ello, se considerarán problemas de optimización dinámica multiperíodo donde cada uno de los períodos considerados son representados como elementos finitos en el tiempo, formados por funciones polinómicas definidas a trozos. El uso de este tipo de metodología hace necesaria la utilización de un determinado método de discretización equivalente a los métodos de Runge-Kutta. La conocida como formulación a gran escala para problemas NLP permite realizar un buen trato de la estructura y escasez del problema, así como el desarrollo de una serie de estrategias que permiten tratar el problema de manera flexible y eficiente. Además, este tipo de metodologías permite tratar con los problemas de convergencia que presenta la utilización de solvers DAE, y los cálculos de sensibilidad por parte del solver son sustituidos por evaluaciones directas de gradientes y Hessianos del problema formulado.

Por otro lado, la explotación de la estructura de la matriz KKT conduce al desarrollo de eficientes algoritmos de gran escala. Sin embargo, pueden presentarse problemas como la elección de un adecuado método de discretización y la selección del adecuado número de elementos finitos, así como la longitud de éstos.

En la siguiente sección se presentan los conocidos como métodos de colocación, así como la descripción de sus propiedades y su relación con los tradicionales métodos Runge-Kutta.

2.4.2 Métodos de Colocación

Considérese el sistema de ecuaciones algebraico-diferenciales dado por:

$$\begin{aligned} &\frac{dz}{dt} = f(z(t), y(t), u(t), p) \\ &g(z(t), y(t), u(t), p) = 0 \\ &z(0) = z_0 \end{aligned} \tag{2.107}$$

Como ya se ha comentado, será necesario realizar la discretización de las variables

continuas, estado, salida y señal de control, las cuales dependen del tiempo. Para llevar a cabo la discretización de sistemas como el descrito en (2.107) se tienen en cuenta las siguientes características para proporcionar una eficiente formulación NLP tal y como se describe en [5]:

- Debido a que los programas no lineales requieren una solución iterativa de las condiciones KKT, no hay ninguna ventaja computacional con respecto al uso de una discretización ODE explícita.
- Debido a que la formulación NLP necesita tratar con discontinuidades, es preferible el uso de un método de paso fijo, ya que no depende de perfiles suaves que se extienden sobre los pasos de tiempo anteriores.
- Las discretizaciones implícitas de alto orden proporcionan perfiles precisos con relativamente pocos elementos. Como resultado, no se requiere de la utilización de un excesivo número de elementos, lo cual es importante en problemas con gran número de estados y señales de control.

Teniendo esto en cuenta, se consideran las siguientes representaciones polinomiales a trozos para los estados y señales de control.

2.4.3 Representación Polinomial de Soluciones ODE

Considérese la ecuación diferencial ODE dada por :

$$\begin{aligned} \frac{dz}{dt} &= f(z(t),t) \\ z(0) &= z_0 \end{aligned} \tag{2.108}$$

para el desarrollo de los métodos de colocación, donde se resuelven ecuaciones diferenciales en puntos de tiempo determinados. Para la variable de estado, se considera una aproximación polinomial de orden $K+1$ a lo largo de un único elemento finito. Este polinomio , denotado como $z^K(t)$ puede ser representado, entre otras, como una

serie de potencias:

$$z^K(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + ... + \alpha_K t^K \tag{2.109}$$

Sin embargo, para el desarrollo de la formulación NLP, se prefieren representaciones basadas en interpolación por polinomios de Lagrange, debido a que los coeficientes de los polinomios tienen los mismo límites que las variables. Eligiendo $K+1$ puntos de interpolación en i elementos, el estado en un determinado elemento i viene dado por:

$$\begin{aligned} & t = t_{i-1} + h_i \tau \\ & z^K(t) = \sum_{j=0}^{K} l_i(\tau) z_{ij} \\ & \text{para } t \in [t_{i-1}, t_i],\ \tau \in [0,1] \\ & \text{donde } l_k(\tau) = \prod_{k=0,\neq j}^{K} \frac{\tau - \tau_k}{\tau_j - \tau_k},\ \tau_0 = 0,\ \tau_j < \tau_{j+1} \end{aligned} \tag{2.110}$$

y h_i es la longitud del elemento i. Esta representación polinómica tiene la propiedad de que $z^k(t_{ij}) = z_{ij}$, donde $t_{ij} = t_{i-1} + \tau_j h_i$. De manera equivalente, el tiempo derivativo del estado puede ser representado como un polinomio de Lagrange con K puntos de interpolación. Ésto conduce a la representación Runge-Kutta para el estado diferencial:

$$z^K(t) = z_{i-1} + h_i \sum_{j=1}^{K} \Omega_j(\tau) \dot{z}_{ij} \tag{2.111}$$

donde z_{i-1} es un coeficiente que representa el estado diferencia al comienzo del elemento i, $\dot{z}_{ij}$ representa la derivada temporal $\frac{dz^K(t_{ij})}{d\tau}$, y $\Omega_j(\tau)$ es un polinomio de orden K que satisface:

$$\Omega_j(\tau) = \int_0^{\tau} l_j(\tau') d\tau',\ t \in [t_{i-1}, t_i],\ \tau \in [0,1] \tag{2.112}$$

Para determinar los coeficientes polinómicos que aproximan la solución del problema DAE, es necesario sustituir el polinomio en (2.108) y reforzar las ecuaciones algebraicas en la interpolación τ_k. Esto conduce a las siguientes ecuaciones de colocación:

$$\frac{dz^K}{dt}(t_{ik}) = f(z^K(t_{ik}), t_{ik}) \tag{2.113}$$

Para las representaciones polinómicas (2.110) y (2.111), es conveniente normalizar el tiempo a lo largo de cada elemento, escribir el estado en función de τ, y aplicar $\frac{dz^K}{d\tau} = h_i \frac{dz^K}{dt}$. Para el polinomio de Lagrange (2.110), las ecuaciones de colocación se convierten en:

$$\sum_{j=0}^{K} z_{ij} \frac{dl_j(\tau_k)}{d\tau} = h_i f(z_{ik}, t_{ik}) \tag{2.114}$$

Mientras que las ecuaciones de colocación por Runge-Kutta vienen dadas por:

$$\begin{aligned} \dot{z}_{ik} &= f(z_{ik}, t_{ik}) \\ z_{ik} &= z_{i-1} + h_i \sum_{j=1}^{K} \Omega_j(\tau_k) \dot{z}_{ij} \end{aligned} \tag{2.115}$$

2.4.4 Métodos de Colocación con Polinomios Ortonormales

Una vez que se han determinado cada uno de los puntos de colocación τ_k, las ecuaciones (2.114) y (2.115) son única y exclusivamente ecuaciones algebraicas que pueden ser incorporadas a la formulación del problema NLP. Lo único que queda por hacer pues es la determinación de τ_k que permite una aproximación más precisa de las variables de estado. La solución a (2.113) puede ser expresada implícitamente como:

$$z(t_i) = z(t_{i-1}) + \int_{t_i-1}^{t_i} f(z(t), t) dt \tag{2.116}$$

Donde la solución numérica de esta integral viene dada por la fórmula cuadrática:

$$z(t_i) = z(t_{i-1}) + \sum_{j=1}^{K} w_j h_i f(z(t_{ij}), t_{ij}),\ t_{ij} = t_{i-1} + h_i \tau_j \tag{2.117}$$

La elección óptima de los puntos de interpolación τ_j y los pesos de cuadratura ω_j conduce a $2K$ grados de libertad, con el resultado de que (2.117) proporciona una solución exacta de (2.116) siempre y cuando $f(z(t), t)$ sea polinomial en t de orden $2K$. Dicha solución óptima, viene dada por el siguiente teorema.

Teorema 20.
La fórmula de cuadratura dada por (2.117) proporciona una solución exacta de la integral (2.116) si $f(z(t), t)$ es polinómica en t de orden $2K$ y τ_j son las raíces de un polinomio de orden K, $P_K(\tau)$, con la siguiente propiedad:

$$\int_0^1 P_j(\tau) P_{j'}(\tau) d\tau = 0,\ j = 0, ..., K-1,\ j' = 1, ..., K \tag{2.118}$$

Tal y como se demuestra en [5], se justifica la elección de los puntos de colocación τ_j como las raíces de $P_K(\tau)$, el polinomio cambiado de Gauss-Legendre que cumple la propiedad de ortogonalidad (2.118). Este polinomio pertenece a la clase más general de polinomios de Gauss-Jacobi que satisfacen que:

$$\int_0^1 (1-\tau)^\alpha \tau^\beta P_j(\tau) P_{j'}(\tau) d\tau = 0 \tag{2.119}$$

Usando estos polinomios, el teorema (20) puede ser adecuadamente modificado para permitir una cuadratura exacta de $f(\tau)$ con grado menor o igual que $2K - 1 - \alpha - \beta$. Los polinomios de Gauss-Jacobi de grado K pueden ser escritos como:

$$P_K^{(\alpha,\beta)} = \sum_{j=0}^{K} (-1)^{K-j} (\tau)^k \gamma_j \tag{2.120}$$

donde:

$$\begin{aligned}\gamma_0 &= 1\\ \gamma_j &= \frac{(K-j+1)(K+j+\alpha+\beta)}{j(j+\beta)}, / j = 1,...,K\end{aligned} \tag{2.121}$$

Con este resultado, se tiene que:

$$z^K(t_i) = z^K(t_i - 1) + h_i \sum_{j=1}^{K} \Omega_j(1) f(z_{ij}) \tag{2.122}$$

Debido a que los polinomios de Gauss-Jacobi conducen a aproximaciones de orden superior, se eligen las raíces como los puntos de colocación de (2.114) o (2.115). Esto conduce a los siguientes casos:

- $\alpha = 0, \beta = 0$ conduce a la colocación de Gauss-Legendre.
- $\alpha = 1, \beta = 0$ conduce a la colocación de Gauss-Radau.
- $\alpha = 1, \beta = 1$ conduce a la colocación de Gauss-Lobatto.

En el presente trabajo se utilizarán (debido a su implementabilidad posterior) colocaciones de Gauss-Legendre y Radau gracias a su compatibilidad con formulaciones NLP y sus buenas propiedades de estabilidad.

Hasta ahora, las ecuaciones de colocación (2.114) y (2.115) han sido escritas a los largo de un único elemento con un $z^K(t_{i-1})$ determinado. Si únicamente se usa un elemento a lo largo de todo el dominio temporal y K es grande, entonces $z^K(t_0) = z_0$. Para múltiples elementos, con $N > 1$, se está reforzando la continuidad del estado a través de los límites elementales. Mediante interpolación de Lagrange, esto puede ser

escrito como:

$$\begin{aligned} z_{1+1,0} &= \sum_{j=0}^{K} l_j(1) z_{ik},\ i = 1,...,N-1 \\ z_f &= \sum_{j=0}^{K} l_j(1) z_{Nj},\ z_{1,0} = z_0 \end{aligned} \tag{2.123}$$

y mediante el criterio de Runge-Kutta, las condiciones de continuidad vienen dadas por:

$$\begin{aligned} z_i &= z_{i-1} + h_i \sum_{j=1}^{K} \Omega_j(1) \dot{z}_{ij},\ i = 1,...,N-1 \\ z_f &= z_{N-1} + h_N \sum_{j=1}^{K} \Omega_j(1) \dot{z}_{Nj} \end{aligned} \tag{2.124}$$

2.4.5 Formulación NLP y Solución

Mediante la discretización por colocación de las ecuaciones de estado, se considera ahora el problema de optimización dinámica multiperíodo. Con cualquiera de las representaciones de las ecuaciones de estado diferenciales (2.114) y (2.115) las variables de control y los estados algebráicos pueden ser también representados mediante interpolación de Lagrange como:

$$\begin{aligned} u(t) &= \sum_{j=1}^{K} \bar{l}_j(\tau) u_{ij} \\ y(t) &= \sum_{j=1}^{K} \bar{l}_j(\tau) y_{ij} \\ \text{donde } \bar{l}_j(\tau) &= \prod_{k=1,\neq j}^{K} \frac{(\tau - \tau_k)}{(\tau_j - \tau_k)} \end{aligned} \tag{2.125}$$

Y las ecuaciones de colocación para las DAEs pueden ser escritas como:

$$\begin{aligned}
&\sum_{j=0}^{K} \dot{l}_j(\tau_k) z_{ij} - h_i f(z_{ik}, y_{ik}, u_{ik}, p) = 0,\ i \in \{1,...,N\},\ k \in \{1,...,K\} \\
&g(z_{ik}, y_{ik}, u_{ik}, p) = 0\ i \in \{1,...,N\},\ k \in \{1,...,K\} \\
&donde\ \ \dot{l}(\tau) = \frac{dl_j(\tau)}{d\tau}
\end{aligned} \tag{2.126}$$

La formulación del problema NLP para un problema de control óptimo puede ser escrita como, tal y como se muestra en [5] como:

$$\begin{aligned}
&\text{mín}\, \Phi(z_f) \\
&s.a \\
&\quad \sum_{j=0}^{K} \dot{l}_j(\tau_k) z_{ij} - h_i f(z_{ik}, y_{ik}, u_{ik}, p) = 0 \\
&\quad g(z_{ik}, y_{ik}, u_{ik}, p) = 0 \\
&\quad z_L \leqslant z_{ik} \leqslant z_U,\ u_L \leqslant u_{ik} \leqslant u_U \\
&\quad y_L \leqslant y_{ik} \leqslant y_U,\ p_L \leqslant p \leqslant p_U \\
&\quad z_{i+1,0} = \sum_{j=0}^{K} l_j(1) z_{ij},\ i = 1,...,N-1 \\
&\quad z_f = \sum_{j=0}^{K} l_j(1) z_{Nj},\ z_{1,0} = z(t_0) \\
&\quad h_E(z_f) = 0
\end{aligned} \tag{2.127}$$

2.4.6 Ajuste de Elementos Finitos

Una cuestión importante a la hora de llevar a cabo la transcripción del problema es la adecuada elección de los pasos temporales h_i. Si h_i es fijado en el problema NLP, el problema resultante es menos no lineal y por lo tanto más fácilmente resoluble. De hecho, para un sistema lineal DAE, fijar h_i conduce a un problema de programación no lineal linealmente restringido. Por otro lado, tratar h_i como variables en (2.127), proporciona una serie de ventajas. Suponiendo que los valores de h_i permanecen pequeños, los elementos finitos variables pueden localizar puntos de ruptura en el problema de

control óptimo así como segmentos con límites activos. Además, debido a que los métodos de Runge-Kutta necesitan que los estados permanezcan suaves únicamente a lo largo de un elemento finito, son únicamente necesarias una serie de pequeñas consideraciones para extender el problema cuando nos encontramos con elementos finitos variables.

- El número de elementos finitos N debe ser elegido lo suficientemente grande. Esta estimación suele ser llevada a cabo mediante ensayo prueba y error a través de la comparación de los perfiles de los estados numéricamente integrados para distintos valores de las variables de decisión p y $u(t)$.

- Si se asume que los perfiles de los estados son lo suficientemente suaves a lo largo del dominio de interés, el error global de la aproximación polinómica $e(t) = z(t) - z^K(t)$, puede ser estimado a partir de:

$$\max_{t\in[0,t_f]} \|e(t)\| \leqslant C_1 \max_{i\in[1,\dots,N]} (\|T_i(t)\|) + O(h_i^{K+2}) \tag{2.128}$$

  Donde C_1 es una constante computacional y $T(t)$ puede ser computada a partir del polinomio solución. Estudio a cerca de cual debería ser la elección de $T_i(t)$ ha sido llevado a cabo en Russell y Christensen [16]. En particular, se suele utilizar:

$$T_i(t) = \frac{d^{K+1}z(t)}{dt^{K+1}} h^{K+1} \tag{2.129}$$

  Esta cantidad puede ser estimada a partir de la discretización de $z^K(t)$ a lo largo de una vecindad de cada elemento i.

- De manera alternativa, es posible obtener una estimación del error a partir de:

$$T_i(t) = \frac{dz^K(t)}{d\tau} - h_i f(z^K(t), y(z^K(t)), u_i(t), p) \tag{2.130}$$

  Esta estimación residual puede ser calculada directamente a partir e la discretización de la DAE considerada. En ella, $T(t_{ik}) = 0$ en los puntos de colocación, pero

seleccionando un punto de no colocación $t_{nc} = t_{i-1} + h_i \tau_{nc}$, $\tau_{nc} \in [0,1]$ conduce a $\|e_i(t)\| \leqslant \bar{C} \|T_i(t_{nc})\|$ donde la constante $\bar{C}$ viene dada por:

$$\begin{aligned} \bar{C} &= \frac{1}{A} \int_0^{\tau_{nc}} \prod_{j=1}^{K} (s - \tau_j) ds \\ A &= \prod_{j=1}^{K} (\tau_{nc} - \tau_j) \end{aligned} \tag{2.131}$$

Con esta estimación, N suficientemente grande, y un error de tolerancia definido ϵ, los valores adecuados de h_i pueden ser determinados añadiendo las restricciones a (2.127):

$$\begin{aligned} &\sum_{i=1}^{N} h_i = t_f,\ h_i \geqslant 0 \\ &\bar{C} \|T_i(t_{nc})\| \leqslant \epsilon \end{aligned} \tag{2.132}$$

- Por otro lado, la formulación de elementos finitos presentada en (2.132) conduce a un aumento de la dificultad del problema NLP, siendo necesario llevar a cabo una cuidadosa inicialización. Una primera aproximación es elegir un numero suficientemente grande de elementos finitos y una valor de paso temporal nominal $\bar{h_i}$ por prueba y error y resolver (2.127) con estos valores fijos. Usando esta solución para inicializar el elemento variable, es posible relajar h_i y añadir la siguiente restricción a (2.127).

$$\sum_{i=1}^{N} h_i = t_f,\ h_i \in [(1-\gamma)\bar{h}_i, (1+\gamma)\bar{h}_i] \tag{2.133}$$

con constante $\gamma \in [0, 1/2]$ y resolver el problema NLP resultante.

Todo lo mostrado anteriormente referente a los métodos de colocación se muestra con el objetivo de tener referencia a cerca de los métodos utilizados, ya que, tal y como se ha comentado previamente, la selección adecuada del número de elementos finitos y puntos de colocación, es un proceso que suele ser llevado mediante prueba y error, tal y como se hará en el presente trabajo.

Parte II

Implementación Práctica

Pyomo: Modelado de Problemas de Optimización en Python

3.1 Introducción

A lo largo de las secciones anteriores, se han descrito los conocimientos teóricos necesarios para la comprensión del desarrollo de los controladores predictivos implementados en el presente trabajo. En el Capítulo 1 se analizaron los fundamentos de diseño de controladores predictivos no lineales en tiempo continuo así como el diseño de controladores predictivos no lineales con estabilidad garantizada atendiendo a la teoría de estabilidad de sistemas dinámicos de Lyapunov. Por otro lado, el Capítulo 2 se centra en los algoritmos desarrollados para la resolución del problema de optimización presente en todo controlador predictivo, así como en los métodos numéricos utilizados para la integración del modelo de predicción en sistemas no lineales.

Partiendo de estos conocimientos, se presenta en ente capítulo la herramienta utilizada para la implementación de los controladores. Si bien es cierto que dicha implementación podría ser llevada a cabo sin el conocimiento de la teoría desarrollada anteriormente, una correcta comprensión de la misma ayudará a realizar una correcta sintonía y ajuste de los controladores, lo cual conducirá a un mejor desempeño funcional y computacional.

La herramienta utilizada será Pyomo, un paquete open source en lenguaje Python para el modelado y solución de problema matemáticos, especialmente pensado para la resolución problemas que involucran algún proceso de optimización. La idea que subyace detrás de Ppyomo no es nueva, y han sido numerosos los lenguajes específicos desarrollados en los últimos tiempos para la resolución de este tipo de problemas de programación matemática, como pueden ser lenguajes tipo AMPL, AIMMS o GAMS. La principal ventaja que presenta Pyomo es que se encuentra desarrollado como pa-

quete embebido en un lenguaje de alto nivel en Python, con todas las ventajas que ello conlleva.

Así pues, Pyomo (Python Optimization Modeling Objects) es una aplicación desarrollada para el modelado, definición y solución de problemas de optimización mediante el uso de lenguaje Python. Python se trata de un lenguaje de alto nivel de sencilla sintaxis y una clara orientación a objetos, lo cual será un incentivo más en el uso de esta herramienta, tal y como se comprobará a lo largo de este capítulo. A continuación se describen alguna de las características principales de Pyomo:

1. **Paquete Open Source**

   Uno de las principales motivos que ha conducido al desarrollo de Pyomo es la necesidad de crear una herramienta adecuada para la resolución de problemas de optimización basada en un lenguaje de código abierto. A pesar de que son numerosos los solvers matemáticos de código abierto para este tipo de problemas, como podrían ser a modo de ejemplo los incluidos en COIN-OR, son pocas las herramientas de código abierto para el modelado del problema.

2. **Capacidad de Adaptación**

   Una de las principales desventajas que presentan los lenguajes de modelado de carácter comercial que desempeñan funciones similares a Pyomo es su limitada capacidad de customización de sus componentes o de los propios procesos de optimización. Debido al carácter de código abierto de Pyomo, el usuario tiene la capacidad de no sólo llevar a cabo modificaciones del código existente, así como de los componentes y características ya integradas, sino también la creación de nuevos componentes, herramientas, o futuras mejoras, lo cual abre un amplio campo para la comunidad investigadora. De esta manera, el diseñador puede personalizar el software a su gusto para adaptarlo a sus necesidades, y prototipar nuevas capacidades que pudieren incluso ser integradas en un futuro en la versión básica.

3. **Integración de Solvers**

   Las hasta ahora existentes herramientas de modelado de problemas de optimización pueden ser clasificadas de manera general en dos categorías. Por un lado

están las herramientas de modelado fuertemente acopladas en las cuales se tiene acceso directo a la utilización de un determinado solver matemático, y por otro lado las herramientas de modelado débilmente acopladas en los que se hace uso de ejecutables externos. Pyomo permite la utilización de ambas vertientes. Los objetivos de diseño de Pyomo han llevado a la distinción entre la formulación del modelo y la ejecución de la optimización, siendo ambos procesos completemente independientes. De esta forma, Pyomo utiliza un lenguaje de alto nivel para la formulación de un determinado problema que puede ser resuelto mediante la utilización de un optimizador escrito en lenguajes de más bajo nivel. Todo ello proporciona las ventajas de los lenguajes de alto nivel a la hora de llevar a cabo la descripción del problema, y las características de los lenguajes de bajo nivel de ser eficientes a la hora de llevar a cabo cálculos numéricos.

4. **Instanciación de Modelos**

   Una de las principales y deseables característica de Pyomo es la definición de modelos abstractos de manera similar a la forma en que trabajan AMPL y AIMMS. En un modelo abstracto se lleva a cabo la separación de la declaración del modelo y de los datos utilizados para la creación de una instancia de modelo específica. Dicha separación proporciona una gran capacidad de modelo flexible. Para ello, Pyomo utiliza una representación simbólica de los datos, variables, restricciones, y objetivos de optimización, para después proceder a generar instancias del modelo previamente creado mediante la llamada a fuentes de datos externos, funcionalidad la cual es parecida a la desarrollada por herramientas de modelado también de código abierto como PuLP y PyMathProg.

5. **Lenguaje de Modelado Flexible**

   Otro de los principales objetivo de Pyomo es el uso directo de lenguaje de programación en alto nivel para soportar la definición de modelos de optimización. De esta forma, Pyomo es similar a herramientas como FlopC++ y OptimJ, los cuales soportan modelado en C++ y Java respectivamente. Estos lenguajes de programación para el desarrollo de modelos de optimización tiene numerosas ventajas:

   - Extensibilidad y Robustez: el uso de lenguajes de programación de alto nivel ampliamente extendidos proporciona fundamentos robustos para el desarrollo y resolución de modelos de optimización. El lenguaje Pyhon en par-

ticular ha sido ampliamente utilizado para una extensa variedad de aplicaciones y contextos e implementado en una gran variedad de plataformas.

- Documentación: lenguajes como Java y Python se encuentran ricamente documentados, existiendo una gran comunidad de soporte detrás de ellos que proporciona ayuda el resto de los usuarios.
- Librerías: el lenguaje de programación Python sea probablemente (debido a ser el lenguaje más ampliamente utilizado) el lenguaje que cuenta con un mayor número de bibliotecas y librerías desarrolladas por la comunidad, lo cual simplifica enormemente el trabajo del desarrollador, al encontrar funciones de todo tipo ya implementadas previamente.

Descritas las principales razones por las cuales se justifica el desarrollo de aplicaciones para la resolución de problemas de optimización en un lenguaje de alto nivel como Pyhon, y antes de proceder a describir las características de la herramienta en sí, se listan a continuación los principales beneficios del uso de Pyomo a modo de propuesta de valor de la herramienta:

1. **Licencia de código abierto:** el uso de una herramienta de licencia libre, tal y como se ha comentado, suele ser de gran importancia a la hora de llevar a cabo el desarrollo de aplicaciones en la vida real. Además los lenguajes de código abierto son por lo general más customizables y proporcionan una mayor flexibilidad al usuario a la hora de llevar a cabo el diseño de aplicaciones.

2. **Embebido en lenguaje de alto nivel:** la mayoría de las herramientas AML comerciales se encuentran embebidas en lenguajes propietarios., lo cual normalmente reducen las capacidades que si proporcionan los lenguajes de código abierto, como pueden ser funciones, clases, y estructuras predefinidas.

3. **Acceso a multitud de funcionalidades externas:** debido a que Pyomo se encuentra embebido en Pyhon a modo de extensión del lenguaje, el usuario puede inmediatamente acceder a un amplio número de potentes librerías,como podría ser Scipy, Numpy, Matplotlib, Pyro, así como una gran número de bases de datos.

4. **Modelos concretos y abstractos de programación matemática:** tal y como ya se ha comentado, la separación entre modelos y datos es extremadamente útil en la práctica, debido a que un mismo modelo puede ser instanciado para la resolución de distintos problemas.

5. **Modelos de programas y solvers no lineales:** mientras que muchos AMLs de carácter comercial proporcionan la capacidad de trabajar con modelos y solvers no lineales, son pocas las herramientas de código abierto que tienen esta capacidad.

6. **Ayuda integrada y computación distribuida:** Pyomo proporciona ayuda integrada para realizar computación distribuida proporcionando capacidades para ambos Python y la librería Pyro. Esta capacidad es única en el contexto de los AMLs de código abierto, y no es común encontrarla en AMLs de carácter comercial.

7. **Ayuda integrada para la obtención de datos externos:** Pyomo proporciona un rico rango de interfaces para la extracción de datos de estructuras externas, como pueden ser documentos de texto o bases de datos, lo cual resulta de gran importancia a la hora de desarrollar herramientas de optimización para problemas de la vida real.

8. **Extensibilidad mediante uso de arquitecturas software basadas en componentes:** Pyomo y Coopr se encuentran diseñadas de manera modular basada en componentes, además, el diseñador puede diseñar extensiones sin que esto afecte al diseño del núcleo de la aplicación. Esta capacidad de ampliar las funcionalidades del núcleo es única entre los AMLs comerciales de código abierto.

3.2 FUNDAMENTOS DE PYOMO

Pyomo es una herramienta que se fundamenta en la creación de modelos a modo de objetos, construidos mediante la incorporación de diversos componentes. De esta forma, podría describirse un modelo en Pyomo como la suma de todos y cada uno

de los componentes declarados y asociados al mismo. Sin entrar en detalles acerca de cada uno de los elementos, se muestra a continuación un diagrama en el cual se puede ver la estructura clásica de un modelo en Pyomo.

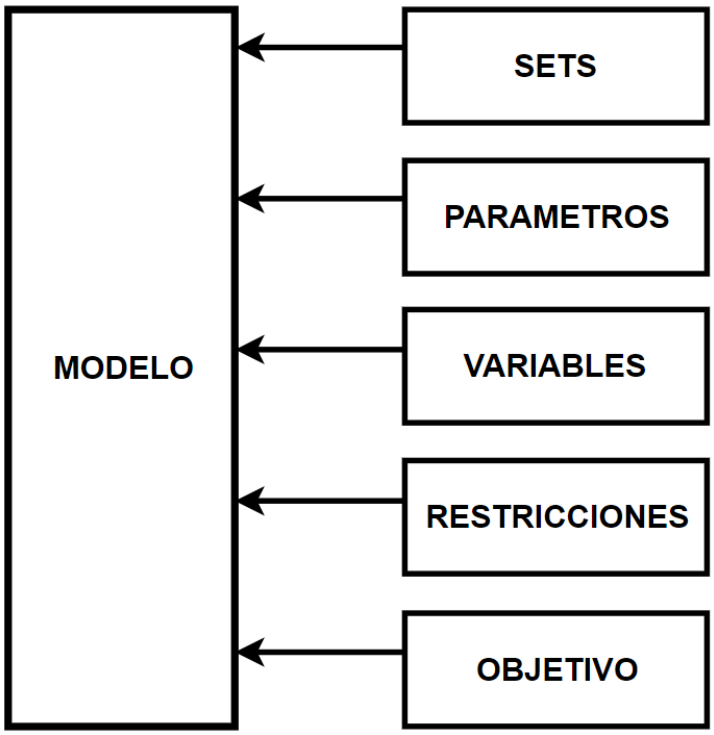


Figura 3.1: Diagrama típico de modelo en Pyomo

En el diagrama anterior se puede observar cómo la definición de un modelo matemático de optimización en Pyomo se llevará a cabo mediante la creación primeramente del modelo objeto y mediante la adición posterior de una serie de elementos de modelado del mismo, los cuales serán descritos con detenimiento posteriormente. El modelo de por sí sólo no es suficiente para la resolución del problema, sino que de alguna forma éste ha de ser pasado a un determinado solver encargado de resolver el problemas de optimización. De manera general, en todo programa para la resolución del un determinado modelo a optimizar, podrán encontrarse los siguientes elementos:

- **Objetos de Modelado:** elementos utilizados para llevar a cabo la descripción del modelo a optimizar, y mediante las cuales se definirán todas las características del sistema sobre el cual se desea trabajar.

- **Objetos de Optimización:** elementos mediante los cuales se lleva a cabo la configuración de la optimización. Dichos elementos es necesario que se ajusten en función de la naturaleza del modelo que se desea optimizar.
- **Interfaz con Solvers:** en todo proceso de optimización será necesaria la llamada a un determinado solver comercial o de código abierto en el cual se implementen los algoritmos necesarios para llevar a cabo la resolución en sí del problema a optimizar.
- **Transformaciones del Modelo:** en determinadas ocasiones, será necesario que el programa realice una serie de transformaciones sobre el modelo para que este pueda ser resuelto. Es el caso, por ejemplo, de la discretización que se ha de llevar a cabo para la integración del modelo mediante la aplicación de métodos de colocación, cuando este se encuentra descrito en función de las ecuaciones dinámicas que definen el sistema.
- **Extensiones de Modelado:** A veces la naturaleza del problema exige la utilización de funciones o elementos no definidos por el núcleo principal de Pyomo y es necesario recurrir a extensiones del lenguaje en la cuales sí se implementen tales funciones.

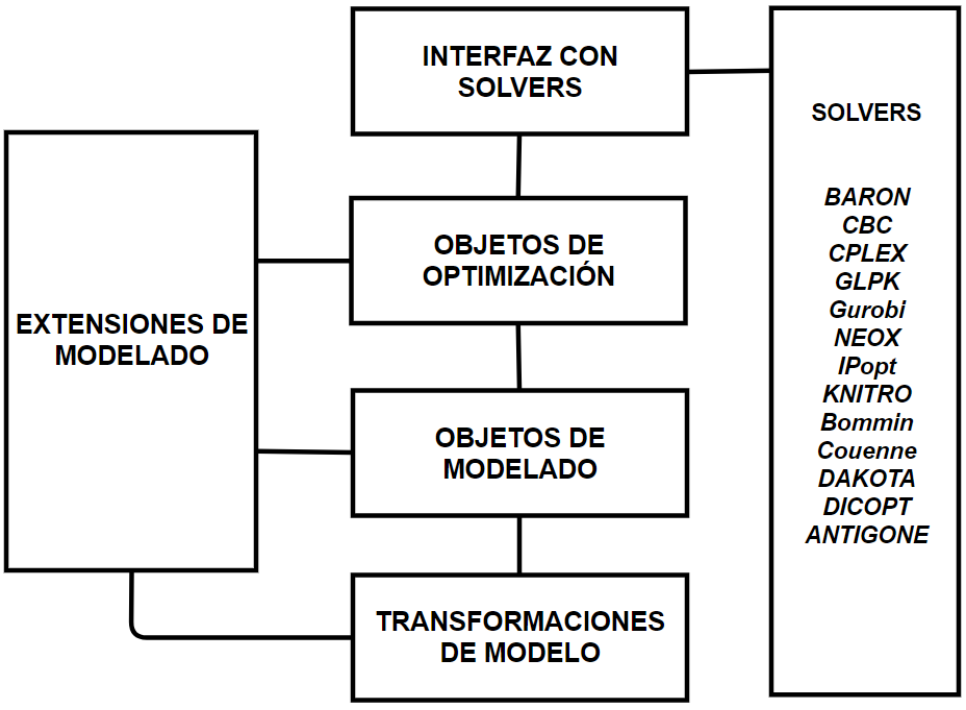


Figura 3.2: Elementos de un programa de optimización

3.3 Modelos y Elementos Básicos en Pyomo

Tal y como previamente se ha comentado, a la hora de crear un determinado modelo en Pyomo será primero necesaria la creación del objeto modelo y posteriormente la inclusión de cada uno de los elementos que llevan a cabo la descripción del sistema que representa. En esta sección se comienza comentando los tipos de modelos que pueden ser creados en Pyomo así como sus principales diferencias. Seguidamente, se analizarán todos y cada uno de los elementos básicos principales cuya inclusión será necesaria en la mayoría de los modelos.

3.3.1 Modelos en Pyomo

Existen principalmente dos tipos de modelos en Pyomo, los modelos concretos y los modelos abstractos. La diferencia principal entre ambos es que, mientras que en un modelo concreto todos los datos necesarios se encuentran incluidos en el mismo fichero en el cual se realiza la definición del modelo, en una modelo abstracto los datos se encuentran en ficheros externos los cuales han de ser llamados cuando se lleva a cabo la resolución del problema. De esta forma, se utilizará un modelo concreto cuando se desee resolver un problema particular mientras que se utilizará un modelo abstracto cuando se desee resolver un problema general del cual se realizarán numerosas instancias con distintos datos.

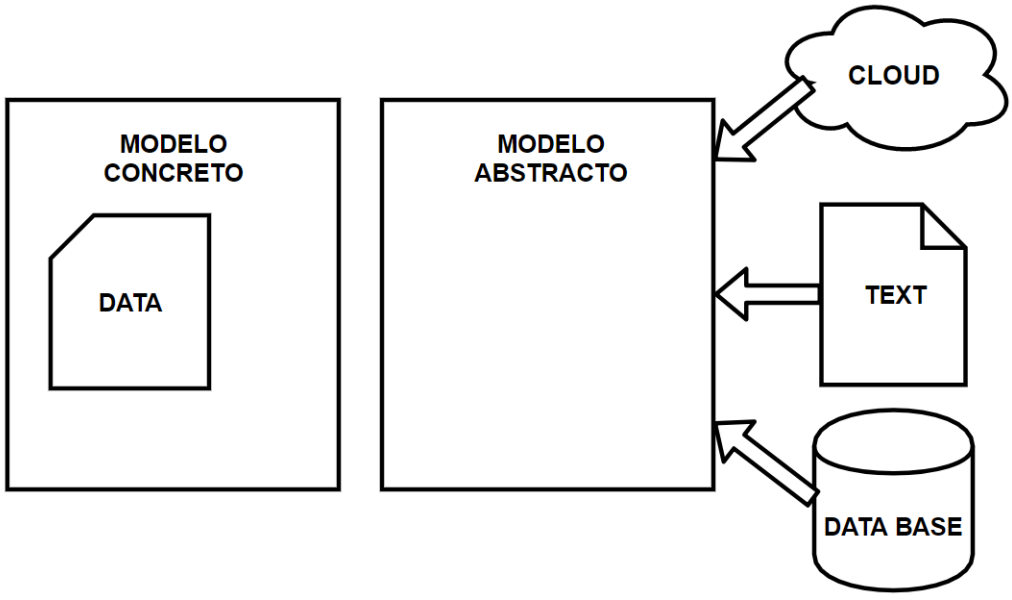


Figura 3.3: Modelos concretos y abstractos

De esta forma, en un modelo abstracto los datos utilizados son representados de manera simbólica, y su valor será asignado cuando se realice una llamada al archivo que los contiene durante el proceso de resolución del problema. A continuación se muestra como se lleva a cabo la creación de un modelo en Pyomo, tanto en el caso de que este sea concreto como abstracto. El primer paso en cualquier programa desarrollado mediante el uso del Pyomo es la llamada e la librería que contiene el paquete . Mediante el comando siguiente se importan todos los paquetes contenidos en el núcleo básico de Pyomo.

```
from pyomo.environ import *
```

Una vez que se ha declarado la librería en la cual se encuentran contenidos los elementos básicos de Pyomo, se puede declarar el modelo, bien concreto o bien abstracto, mediante los siguientes comandos.

```
model=ConcreteModel()
```

```
model=AbstractModel()
```

En el caso en el que el modelo sea concreto, simplemente será necesario declarar el solver que ha de ser utilizado para su resolución y llevar a cabo la misma mediante el comando oportuno. En el caso en el que el modelo sea abstracto, será necesario llevar a cabo una instanciación del mismo y la asignación del fichero del cual se leerán los datos pertinentes. Todo ello se realizará mediante un único comando, en el cual se indicará que se está creado una instancia del modelo, el fichero externo del cual se estarán leyendo los datos, y la extensión del mismo.

```
instancia=model.create("fichero")
```

3.3.2 Elementos Básicos en Pyomo

Una vez que se ha creado el objeto modelo en Pyomo, concreto o abstracto, el siguiente paso es el de agregar todos y cada uno de los elementos necesarios a dicho modelo para la descripción del problema de optimización, sistema a controlar, y del problema a resolver en sí. Estos elementos básicos que será necesario añadir al modelo son:

- **Sets**
- **Parámetros**
- **Variables**
- **Función Objetivo**
- **Restricciones**

A continuación de describirán en detalle cada uno de estos elementos listados anteriormente, comentando su función dentro del problema de optimización así como la forma en la que son declarados y así anexados al modelo.

Sets

Estos elementos son aquellos utilizados en Pyomo cuando se desea llevar a cabo la indexación de una determinada variable. Para su definición, es necesario llevar a cabo una inicialización de los mismos. Existe numerosas formas de realizar dicha inicialización. La más básica de ellas, es la utilización de lista, sets, o tuplas de python, tal y como se muestra a continuación:

```
model.A=Set(initialize=[1,2,3])
model.B=Set(initialize=set([1,2,3]))
model.C=Set(initialize=(1,2,3))
```

Para la inicialización de los sets en Pyomo, también es posible utilizar elementos generadores, como **range()** o bucles *for*, tal y como se muestra a continuación:

```
model.A=Set(initialize=range(3))
model.B=Set(initialize=(i for i in model.A)
```

Otra forma de llevar a cabo la inicialización mediante el uso de arrays de la librería numpy.

```
index=numpy.array([1,2,3])
model.A=Set(initialize=index)
```

Por último, existe la posibilidad, de utilizar funciones de Python que devuelvan datos nativos, técnica la cual será utilizada también para la declaración de la función objetivo y de las restricciones del problema, tal y como se verá más delante.

```
def index(model):
        return [1,2,3]
model.A=Set(initialize=index)
```

En los casos mostrados anteriormente se llevaba a cabo una inicialización directa de los Sets a la vez que estos se definían. Sin embargo, es posible llevar a cabo la inicialización de manera indirecta cuando el valor que ha de tomar se encuentra relacionado con el número de elementos de una serie de parámetros contenidos en un fichero externo. De esta forma, simplemente sería necesario indicar que el elemento agregado se trata de un set sin tener que llevar a cabo su inicialización.

```
model.A=Set()
```

A parte del comando utilizado para la inicialización del valor del set, existen una serie comandos adicionales que puede ser utilizados para su correcta definición, entre los cuales destacan:

- **doc:** utiliza un string para describir el set.
- **dimen:** Dimensión de cada uno de los miembros del set.
- **filter:** función booleana utilizada para indicar si debería o no asignarse un nuevo miembro al set.
- **ordered:**indicador booleano de que el set se encuentra ordenado.
- **validate:** función booleana que valida los nuevos datos añadidos.
- **virtual:** indicador booleano de que el set nunca tendrá elementos. No es muy usual.
- **within:**se trata de un super-set del set que se declara. Utilizado para validación.

Además de estas funciones, es posible llevar a cabo operaciones tanto de carácter booleano como de carácter algebraico:

```
model.A=Set()
model.B=Set()

model.C=model.A | model.B #Union
model.D=model.A & model.B  #Intersecion
model.E=model.A-model.B #Diferencia
model.F=model.A^model.B #Or-exclusiva
model.G=model.A*model.B #Producto
```

Por último, es posible especificar el domino en el cual se han de encontrar los elementos pertenecientes al Set. Los indicadores para expresar tales dominios serán también utilizados en parámetros y variables. A continuación se muestra una lista de tales indicadores.

- **Any:** cualquier valor es posible.

- **Reals:** valores en punto flotante
- **PositiveReals:** valores en punto flotante positivos.
- **NonPositiveReals:** valores en punto flotante no positivos.
- **NegativeReals:** valores en punto flotante negativos.
- **PercentFraction:** valores en punto flotante en el intervalor [0,1].
- **UnitInterval:** otra forma de indicar el anterior.
- **Integers:** valores enteros.
- **PositiveIntegers:** valores enteros positivos.
- **NonPositiveIntegers:** valores enteros no positivos.
- **NegativeIntegers:** valores enteros negativos.
- **NonNegativeIntegers:** valores enteros no negativos.
- **Boolean:** Valores booleanos.
- **Binary:** otra forma de indicar el anterior.

A modo de ejemplo, podría indicarse que los elementos de un determinado set han de pertenecer al conjunto de enteros positivos como:

```
model.A=Set(within=PositiveIntgers)
```

Parámetros

Son parámetros aquellos elementos utilizados durante el problema de optimización cuyo valor no es utilizado como variable de decisión del mismo, y que permanecen constantes durante todo el proceso de optimización para el valor de un índice o índices. Por ello, a la hora de llevar a cabo la declaración de un parámetro, será necesario indiciar de qué índice o indices depende.

```
model.A=Set(initialize=(1,2,3))
model.B=Param(model.A)
```

En el caso en que el que valor del parámetro permanezca constante a lo largo de todo el proceso de optimización sin depender de ningún índice, son dos las alternativas existentes. Bien su declaración como parámetros sin dependencia de índice, o bien mediante una constante de Python al uso. De esta forma se puede tener:

```
model.A=Param()
A=10
```

Siendo ésta ultima la opción más común en el caso de utilizarse modelos concretos en los cuales se utilizan parámetros constantes. Sin embargo, en modelos abstractos, en los cuales los datos son tomados de ficheros externos, se suele llevar a cabo la declaración conjunta de sets y parámetros, tal y como se muestra a continuación:

Data File

```
set A:= a b c
param B:= a 12 b 9 c 56
```

Es posible llevar a cabo la definición de múltiples parámetros que dependen de un mismo set. En este caso, se podría haber optado por la siguiente sintaxis, tanto en el archivo principal de programación como en el fichero externo de datos.

```
model.A=Set()
model.P1=Param(model.A)
model.P2=Param(model.A)
model.P3=Param(model.A)
```

Data File

```
set A:= d e f

param: B C D :=

d 1 2 3
e 4 5 6
f 7 8 9
```

De igual manera a lo que ocurría con los sets, existen una serie de opciones de comando para la configuración del parámetro, entre las cuales destacan:

- **default:** Ausenta cualquier otro tipo de especificación.
- **doc:** String par ala descripción del parámetro.
- **initialize:** Función para la inicialización del parámetro.
- **validate:** Función booleana con argumentos que corresponden al valor prospectivo del parámetro, los indices del parámetro, y el modelo.
- **within:** Especificación del dominio de los valores del parámetro.

Por último, también es posible utilizar una función de Python. A modo de ejemplo, supóngase que se desea crear un parámetro consistente en una matriz diagonal, donde cada uno de los elementos de la matriz es el cuadrado de uno de los dos índices utilizados para su indexación en función de filas y columnas, y el resto de elementos toma el valor cero.

```
def A1_init(model,i,j):
        if i==j:
                return i*i
        else:
                return 0.0
model.A1=Param(model.A, model.A,initialize=A_init)
```

Variables

Otro de los elementos clave, junto con sets y parámetros en pyomo, son las variables. Se definen como variables aquellos elementos a los cuales se les dará valor a lo largo del proceso de optimización. Dichas variables son declaradas, pueden ser opcionalmente restringidas a un conjunto de valores, y se les pueden ser asignados un conjunto de valores iniciales.

De esta forma, todos y cada uno de los elementos que sean declarados como variables serán utilizados como toma de decisión durante el proceso de optimización, siendo esto una de las principales ventajas de Pyomo, y lo cual se conoce como optmización multiobjetivo. Algunas de las directivas que se pueden incluir durante la declaración de variables son:

- **bounds:**Función que restringe superior e inferiormente los valores que puede tomar la variable durante el proceso de optimización.
- **domain:** Conjunto de valores que la variable puede tomar.
- **initialize:** Función que da un valor inicial a la variable en definida.
- **within:** Igual que la función domain.

A continuación de muestra un ejemplo de declaración genérica de variable.

```
model.A=Var(within=NonNegativeReals,bounds=(3,10),initialize=4)
```

Donde se ha declarado una variable, restringida al conjunto de valores de los números reales no negativos, en el intervalo (3,10), y tomando un valor inicial de 4.

Obetivo

Se define objetivo en Pyomo como la función que se desea optimizar, y la cual sirve como índice de desempeño del problema de optmimización. De esta forma, se trata de una función que devuelve un valor el cual se intenta minimizar o maximizar, con respecto al conjunto de variables. Aunque existen diversas maneras de llevar a cabo la declaración de la función objetivo, la más recomendada de todas ellas es el uso de una función genérica de Python que después es incluida como elemento del modelo a modo de objetivo. A continuación se muestra una ejemplo de declaración de función objetivo.

```
def FuncObjetivo(model):
        return 2*model.A[1}+5*model.A[3]
model.obj=Objective(rule=FuncObjetivo)
```

Por defecto, y si no se indica lo contrario, durante el proceso de optimización se llevará a cabo la minimización del objetivo. Además, este tipo de declaración resulta de especial interés cuando se desea que el objetivo sea dependiente de un determinado set de valores, además de para llevar a cabo la inicialización de las variables (aunque normalmente esto se realizará en la declaración de restricciones). A modo de ejemplo, se muestra el siguiente fragmento de código donde se intenta maximizar un determinado objetivo, y se inicializa una variable dentro de la misma función.

```
def FuncObjetivo(model,i):
        if i==0:
                return model.A[1]==A_init
        else:
                return 2*model.A[1}+5*model.A[3]
model.obj=Objective(rule=FuncObjetivo, sense=maximize)
```

Restricciones

Se definen restricciones como aquellos elementos utilizados para restringir, por lo general, del valor de las variables del problema. Sin embargo, también se utilizarán restricciones, tal y como se verá en secciones posteriores, para definir el comportamiento dinámico de un sistema.

Existen muchas formas de llevar a cabo la inclusión de restricciones en el sistema. Una de ellas ya ha sido mostrada, y es que es posible restringir los valores que puede tomar una determinada variables durante su proceso de declaración. Sin embargo, la mayoría de las restricciones se especifican utilizando expresiones de igualdad o desigualdad que se crean utilizando una regla, dentro de una función de Python, tal y como ocurría en la declaración del objetivo del problema. Se muestra a continuación un ejemplo de ellos.

```
def Conrule(model):
        return (model.A[1]+model.B[2]==4)
model.con_1=Constraint(rule=Conrule)
```

En lugar de expresiones de igualdad o desigualdad, las restricciones pueden ser también expresadas utilizando una tupla de la forma (lb,expr,ub) donde por ejemplo lb puede ser establecida a None, lo cual es interpretado como $lb \leqslant expr \leqslant ub$. De esta forma, las variabless sólo tienen que aparecer en la expresión intermedia. Por ejemplo, las dos declaraciones siguientes tienen el mismo significado.

```
model.x=Var()

def ruleA(model):
        return model.x >= 19
```

```
model.const=Constarint(rule=ruleA)

def ruleA(model):
        return (19,model.x,None)
model.const=Constarint(rule=ruleA)
```

3.3.3 Aplicación de Solvers

Una vez que se ha definido el problema de optimización, es decir, se ha descrito por completo el modelo que se desea optimizar, incluyéndose función objetivo a optimizar y restricciones impuestas al sistema, el único paso que queda por dar para la resolución del problema es la aplicación del solver matemático encargado de resolver el problema.

Una de las principales ventajas que presenta el uso de esta herramienta es que no se encuentra ligada a ningún solver en concreto, y es el usuario quien libremente puede seleccionar el solver más adecuado en función de los requerimientos del sistema y de la naturaleza del problema a resolver. El primer paso a dar es la llamada a la librería que lo contiene. En este caso sería la llamada a la librería SolverFactory, incluida en el paquete de optimización de la biblioteca genérica, después de la llamada al entorno de Pyomo en el cual se encuentran incluidas todas las bibliotecas de funciones generales.

```
import pyomo.environ as *
from pyomo.opt import SolverFactory
```

Una vez que se han incluido las bibliotecas pertinentes en las cuales se encuentra el solver a utilizar, el siguiente paso es la creación del objeto de optimización, en el cual además se designa el solver a utilizar. En este caso de ha seleccionado a modo de ejemplo el solver glpk para la resolución de problemas de programación lineal.

```
opt = pyo.SolverFactory('glpk')
```

Definido el solver a ser utilizado para le resolución del problema, existen dos alternativas. La primera de ellas es resolver directamente el modelo creado, y la segunda la creación primeramente de una instancia de dicho modelo y la posterior resolución de ésta. Para ello, simplemente es necesario invocar la función de resolución del objeto de optimización previamente creado, haciendo referencia al modelo o instancia que se desea resolver, tal y como se muestra a continuación.

```
instance=model.create_instance()
results=opt.solve(instance)
```

3.4 Optimización Dinámica en Pyomo

Hasta el momento se han descrito algunos de los elementos fundamentales del paquete básico de Pyomo. En este apartado, se describen las características de una de las diversas extensiones de la herramienta, conocida como Pyomo DAE. Esta extensión de modelado permite a los usuarios la incorporación al modelo a optimizar de ecuaciones algebraico diferenciales (DAE). Los componentes de modelado en esta extensión permiten la representación de ecuaciones diferenciales ordinarias o parciales. Estas ecuaciones diferenciales no han de ser escritas en un formato particular, y los componentes son lo suficientemente flexibles como para representar derivadas de alto orden o derivadas parciales mixtas. En la extensión Pyomo.DAE también se incluyen las transformaciones del modelo que utilizan métodos de discretización simultánea para transformar el modelo DAE en un modelo algebraico. Por último, Pyomo.DAE incluye utilidades para el uso simultáneo de modelos DAE y la inicialización de problemas de optimización dinámica.

En la extensión Pyomo.DAE son tres los nuevos elementos de modelado introducidos, a saber:

- **Sets continuos**
- **Variables derivativas**
- **Integrales continuas**

Tal y como se mostrará a continuación, cualquier ecuación diferencial puede ser descrita mediante el uso y combinación de estas nuevas componentes de modelado de manera conjunta con las variables y restricciones incluidas en el paquete básico.

Sets continuos

Este componente es utilizado para definir dominios continuos (por ejemplo temporales). Su declaración a componente set convencional incluido en la versión básica y puede ser utilizado para indexar elementos como variables y restricciones. Cualquier número de sets continuos puede ser utilizado para indexar cualquier componente, y un mismo componente puede ser indexado simultáneamente por sets y sets continuos de manera arbitraria.

En la versión actual, todos los modelos que incluyan sets continuos deben ser previamente discretizados para poder ser resueltos. Además, como mínimo cada set continuo debe ser inicializado con sendos valores numéricos que representen los límites inferior y superior del dominio continuo. El usuario ha de encargarse también de especificar los puntos adicionales dentro del dominio que serán utilizados como elementos finitos en el proceso de discretización.

A continuación se muestran ejemplos de declaración se sets continuos.

```
from pyomo.environ import *
from pyomo.dae import *
```

```
model=ConcreteModel()

model.t=ContinuousSet(bounds=(0,5))
model.x=ContinuousSet(initialize=[0,1,2,3])
```

Variables derivativas

Las variables derivativas son utilizadas en un modelo para representar la derivada de una determinada variable previamente definida con respecto a un determinado set (obligatoriamente un set de carácter continuo). De esta forma, a la hora de declarar una variable derivativa se ha de especificar la variable que se esta derivando así como el set continuo con respecto al cual se deriva.

A continuación se muestra un ejemplo de declaración de variables derivativas en un modelo de Pyomo. En cada caso, la variable que está siendo diferenciada es el único argumento que se utiliza en la declaración, y se usa la palabra clave *wrt* para especificar el tipo de derivada con respecto al set continuo.

```
from pyomo.environ import *
from pyomo.dae import *

model=ConcreteModel()
model.s=Set(initialize=['a','b'])
model.t=ContinuousSet(bounds=(0,5))
model.l=ContinuousSet(bounds=(-10,10))

model.x=Var(model.t)
model.y=Var(model.t,model.s)
model.z=Var(model.t,model.l)

model.dxdt=DerivativeVar(model.x,wrt=model.t)
model.dydt2=DerivativeVar(model.y,wrt=(model.t,model.t))
```

Declaración de Ecuaciones Diferenciales

La forma en la que se lleva a cabo la declaración de ecuaciones diferenciales e Pyomo es en forma de restricciones, es decir, se restringe el sistema a que se comporte dinámicamente según la ecuación diferencial considerada. A continuación se muestra un ejemplo de declaración de ecuación diferencial.

```
from pyomo.environ import *
form pyomo.dae import *

model=ConcreteModel()
model.s=Set(initialize=['a','b'])
model.t=ContinuousSet(bounds=(3,6))
model.l=ContinuousSet(bounds=(-129,49)))

model.x = Var(model.s, model.t)
model.y = Var(model.t, model.l)
model.dxdt = DerivativeVar(model.x, wrt=model.t)
model.dydt = DerivativeVar(model.y, wrt=model.t)
model.dydl2 = DerivativeVar(model.y, wrt=(model.l, model.l))

#Declaracion de ODE

def _ode_rule(m, s, t):
        if t == 0:
                return Constraint.Skip
        else:
                return m.dxdt[s, t] == m.x[s, t]**2
model.ode = Constraint(model.s, model.t, rule=_ode_rule)

#Declaracion de PDE

def _pde_rule(m, t, l):
        if t == 0 or l == m.l.first() or l == m.l.last():
                return Constraint.Skip
```

```
        else:
                return m.dydt[t, l] == m.dydl2[t, l]
model.pde = Constraint(model.t, model.l, rule=_pde_rule)
```

Declaración de integrales

Los elementos integrales pueden ser utilizados para representar la integral a lo largo de un dominio o set continuo. Una vez que se ha llevado a cabo la discretización del set continuo, cualquier integral en el modelo será convertida en ecuaciones algebraicas mediante el uso de la regla del trapecio. Se espera que en futuras mejores se utilicen métodos más avanzados de integración. A continuación se muestra un ejemplo sencillo de declaración de integral, nuevamente mediante el uso de una función al uso de Python.

```
model = ConcreteModel()
model.time = ContinuousSet(bounds=(0,10))
model.X = Var(model.time)
model.scale = Param(initialize=1E-3)

def _intX(m,t):
    return m.X[t]
model.intX = Integral(model.time,wrt=model.time,rule=_intX)

def _obj(m):
    return m.scale*m.intX
model.obj = Objective(rule=_obj)
```

Los argumentos posicionales proporcionados a la integral deben incluir los índices necesarios para evaluar la expresión integral. La expresión integral es definida como función y provista a la palabra clave rule como argumento. Por último, el usuario debe especificar el set continuo utilizado a lo largo del cual la integral será evaluada.

Una vez que la integral ha sido declarada, puede ser utilizada como un elemento expresión de Pyomo y puede ser incluida como restricción o función objetivo. Si una integral es definida mediante múltiples argumentos posicionales, es decir, múltiples sets de indexación, el último componente sera indexado por todos esos sets a excepción del set continuo que tomó la integral. En otras palabras, los set continuos especificados con la palabra clave wrt serán eliminados del conjunto sets de indexación incluso si estos han de ser especificados como argumento posicional. Se muestra a continuación un ejemplo de integral doble a lo largo de dos sets continuos. Además, la expresión es también indexada por un set convencional.

```
model = ConcreteModel()
model.t1 = ContinuousSet(bounds=(0, 10))
model.t2 = ContinuousSet(bounds=(-1, 1))
model.s = Set(initialize=['A', 'B', 'C'])

model.X = Var(model.t1, model.t2, model.s)

def _intX1(m, t1, t2, s):
    return m.X[t1, t2, s]
model.intX1 = Integral(model.t1,model.t2, model.s,wrt=model.t1,
rule=_intX1)

def _intX2(m, t2, s):
    return m.intX1[t2, s]
model.intX2 = Integral(model.t2, model.s, wrt=model.t2,
rule=_intX2)

def _obj(m):
    return sum(m.intX2[k] for k in m.s)
model.obj = Objective(rule=_obj)
```

Transformaciones de Discretización

Antes que cualquier modelo en Pyomo el cual contenga variables derivativas o integrales pueda ser enviado al solver matemático ha de pasar por un proceso de discretización. Mediante estas transformaciones es posible aproximar cualquier derivada o integral en el modelo mediante el uso de métodos numéricos. Los métodos numéricos utilizados actualmente por Pyomo discretizan los dominios continuos del problema e introducen restricciones de igualdad que aproximan las derivadas e integrales en los puntos de discretización seleccionados. En la extensión Pyomo.DAE se incluyen dos familias de métodos de discretización, mediante diferencias finitas y mediante puntos de colocación.

Las transformaciones mediante diferencias finitas incluyen multitud de métodos distintos. Uno de los más utilizados en el método Backward Euler. Las ecuaciones de discretización por este método se muestran a continuación.

$$\begin{aligned} &Dado \\ &\frac{dx}{dt} = f(t,x),\ x(t_0) = x_0 \\ &Sediscretizatyxcomo \\ &x(t_0 + kh) = x_k \\ &x_{k+1} = x_k + hf(t_{k+1}, x_{k+1}) \\ &t_{k+1} = t_k + 1 \end{aligned} \tag{3.1}$$

Donde h es el paso entre los puntos de discretización o el tamaño de cada elemento finito. Estas ecuaciones se generan automáticamente como restricciones cuando se aplica el método de discretización al modelo. Existen multitud de opciones de discretización que pueden ser especificadas mediante le uso de palabras clave como argumentos de la función de transformación. Algunas de estas palabras clave para la aplicación de métodos de diferencias finitas son:

- **nfe:** hace referencia al número de elementos finitos a incluir en la discretización. Toma un valor de 10 por defecto.
- **wrt:** Indica el set continuo con respecto al cual se debe de aplicar la discretización.

- **scheme:** Indica el método de diferencias finitas a aplicar. Las opciones disponibles son BACKWARD, CENTRAL o FORWARD. El método utilizado por defecto es el método BACKWARD.

Si el número existente de elementos finitos en el set continuo es inferior al número deseado, se añadirán por defecto nuevos puntos de discretización. Si el usuario especifica un número de elementos finitos inferior al numero de punto ya incluidos en el set contio entonces el método de transformación ignorará el número especificado y aplicará el mayor de ellos.

A continuación se muestra un ejemplo de aplicación de un método de diferencias finitas. Además, se muestra como es posible añadir restricciones al modelo discretizado.

```
# Discretizacion del modelo

discretizer = TransformationFactory('dae.finite_difference')
discretizer.apply_to(model,nfe=20,wrt=model.time,
scheme='BACKWARD')

#Adicion de restriccion al modelo discretizado

def _sum_limit(m):
        return sum(m.x1[i] for i in m.time) >=30
model.con_sum_limit = Constraint(rule=_sum_limit)

#Resolucion del modelo discretizado

solver = SolverFactory('ipopt')
results = solver.solve(model)
```

La otra familia de métodos de discretización disponibles en Pyomo son los conocidos como métodos de colocación, de los cuales ya se ha hablado en capítulos anteriores. Estas transformaciones utilizan colocaciones ortogonales para discretizar las ecuaciones diferenciales del modelo. Actualmente, existe dos tipos de métodos de colocación implementados, aquellos basados en polinomios de Gauss-Radau y los basados en polinomios de Gauss-Legendre. Las opciones de configuración de los métodos de colocación son iguales a aquellas para los métodos por diferencias finitas, con la diferencia de que se utilizan esquemas distintos y se incluye la opción de selección del número de puntos de colocación.

- **scheme:** Permite la selección del métodos por polinomios LAGRANGE-RADAU o LAGRANGE-LEGENDRE.
- **ncp:** Permite la selección del número de puntos de colocación entre cada elemento finito. El valor por defecto es de 3.

A continuación se muestra un código a modo de ejemplo de discretización mediante el uso de puntos de colocación.

```
#Discretizacion del modelo

discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(model,nfe=20,ncp=6,
scheme='LAGRANGE-RADAU')

#Adicion de restriccion al modelo discretizado

def obj_rule(m):
        return sum((m.x[i]-m.x_ref)**2 for i in m.time)
model.obj = Objective(rule=obj_rule)

#Resolucion del modelo discretizado

solver = SolverFactory('ipopt')
results = solver.solve(model)
```

3.5 Ejemplos Básicos en Pyomo

Una vez que se han definidos los elementos principales de Pyomo, se procede ahora, antes de mostrar los controladores predictivos no lineales propuestos en el presente trabajo, una serie de ejemplos para la mejor comprensión y asimilación de los conceptos previamente desarrollados.

Primeramente, se mostrarán ejemplos de problemas que hacen uso única y exclusivamente del núcleo de Pyomo, y que no requieren de ninguna extensión de modelado. Seguidamente, se llevarán a cabo el desarrollo de ejemplos que hacen uso de la extensión que permite la introducción de ecuaciones algebraico-diferenciales, lo cual será de vital importancia en el desarrollo de los controladores NMPC.

3.5.1 Problema de la Mochila (Modelo Concreto)

Supóngase que se desea llevar a cabo el transporte de una serie de objetos, pertenecientes a un conjunto A, y cada uno de los cuales tiene asociado un determinado peso y un determinado beneficio, tal y como se muestra en la siguiente tabla.

Objeto	**Peso (w)**	**Beneficio (b)**
Martillo	5	8
Llave	7	3
Pala	4	6
Toalla	3	11

Cuadro 3.1: Datos Problema Mochila

El objetivo del problema de optimización es el de seleccionar un subconjunto de objetos a transportar tal que se minimice el peso del equipaje y se maximice el valor del contenido transportado, teniéndose en cuenta que existe un máximo peso a poder ser transportado. La simbología utilizada será la siguiente.

Simbolo	Significado
A	Conjunto de objetos disponibles
b_i	Beneficio del objeto i
w_i	Peso del objeto i
W_{max}	Máximo peso que puede ser transportado
x_i	Variable discreta de selección

Cuadro 3.2: Simbología problema de la mochila

Matemáticamente el problema puede ser expresado como:

$$\begin{aligned} &\max_x \sum_{i \in A} b_i x_i \\ &s.a \\ &\quad \sum_{i \in A} w_i x_i \leqslant W_{max} \\ &\quad x_i \in \{0,1\} \; \forall i \in A \end{aligned} \tag{3.2}$$

El código para la resolución del problema se muestra a continuación.

```
from pyomo.environ import *

# Datos del problema
A = ['martillo', 'llave', 'pala', 'toalla']
b = {'martillo':8, 'llave':3, 'pala':6, 'toalla':11}
w = {'martillo':5, 'llave':7, 'pala':4, 'toalla':3}
W_max = 14

#Creacion del modelo
model = ConcreteModel()

#Declaracion de variables
model.x = Var(A,within=Binary)

#Funcion objetivo
model.value = Objective(
```

```
        expr=sum(b[i]*model.x[i] for i in A),
        sense=maximize)

#Restricciones del problema
model.weight = Constraint(
        expr=sum(w[i]*model.x[i] for i in A) <=W_max)

#Seleccion del solver para problemas LP
opt=SolverFactory('glpk')

#Resolucion del problema
result_obj=opt.solve(model,tee=True)

#model.pprint()
```

3.5.2 Problema de la dieta (Modelo abstracto)

El objetivo del problema de la dieta es seleccionar una serie de alimentos tal que se satisfagan las necesidades alimentarias básicas a un mínimo coste de los alimentos. Este problema puede ser formulado como un problema de programación lineal, en el cual las restricciones son el número de calorías, vitaminas, minerales, grasas, socio y y colesterol contenidas en la dieta. En esta ocasión se optará por la utilización de un modelo abstracto en lugar de concreto, de tal forma que los datos serán proporcionados por un fichero externo. La simbología utilizada será la siguiente.

Simbolo	Significado
F	Set de alimentos
N	Set de nutrientes
c_i	Coste de servir el alimento i
a_{ij}	Cantidad de nutriente j en alimento i
N_{min_j}	Nivel mínimo de nutriente j
N_{max_j}	Nivel máximo nutriente j
V_{max}	Volumen máximo de alimento a consumir
V_i	Volumen de alimento servido de i
x_i	Número de alimento i servidos

El problema puede ser matemáticamente expresado como:

$$
\begin{aligned}
&\text{mín} \sum_{i \in F} c_i x_i \\
&s.a \\
&\quad N_{min_j} \leqslant \sum_{i \in F} a_{ij} x_i \leqslant N_{max_j} \; \forall j \in N \\
&\quad \sum_{i \in F} V_i x_i \leqslant V_{max} \\
&\quad x_i \geqslant 0 \; \forall i \in F
\end{aligned}
\tag{3.3}
$$

El código para la resolución del problema se muestra a continuación. Como puede apreciarse, no se ha incluido en ninguna parte los datos del problema, y estos serán incluidos cuando se realice la llamada a la resolución del mismo a través del paso de un fichero de datos.

```
from pyomo.environ import *
infinity = float('inf')

#Creacion del modelo
model = AbstractModel()

#Sets del problema, alimentos y nutrientes
model.F = Set()
model.N = Set()

#Parametros del problema
model.c    = Param(model.F, within=PositiveReals)
model.a    = Param(model.F, model.N, within=NonNegativeReals)
model.Nmin = Param(model.N, within=NonNegativeReals,
default=0.0)
model.Nmax = Param(model.N,
within=NonNegativeReals, default=infinity)
model.V    = Param(model.F, within=PositiveReals)
model.Vmax = Param(within=PositiveReals)
```

```
#Declaracion de variables
model.x = Var(model.F, within=NonNegativeIntegers)

#Funcion objetivo
def cost_rule(model):
    return sum(model.c[i]*model.x[i] for i in model.F)
model.cost = Objective(rule=cost_rule)

#Restricciones del problema
def nutrient_rule(model, j):
    value = sum(model.a[i,j]*model.x[i] for i in model.F)
    return model.Nmin[j] <= value <= model.Nmax[j]
model.nutrient_limit = Constraint(model.N, rule=nutrient_rule)


def volume_rule(model):
    return sum(model.V[i]*model.x[i] for i in model.F)
    <= model.Vmax
model.volume = Constraint(rule=volume_rule)
```

Descrito el modelo, será necesario de alguna manera hacer una llamada al solver encargado de la resolución del problema, así como la inclusión de los datos del mismo. La forma más común de llevar a cabo este procedimiento es mediante comando por consola del sistema, de tal forma que simultáneamente se invocan los archivos que contienen tanto modelos como datos y el solver a utilizar. Suponiendo que el modelo se encuentra almacenado en un fichero de Python denominado *dieta.py*, los datos del mismo en un fichero de texto *dieta.dat*, y se desea utilizar el solver para problemas LP glpk, el comando de resolución por consola sería:

```
pyomo solve --solver=glpk dieta.py dieta.dat
```

3.5.3 Función de Rosenbrock (Problema No Lineal)

En este ejemplo se llevará a cabo la optimización de una función de carácter no lineal. El objetivo principal será el de encontrar el mínimo de la conocida como función de Rosenbrock dado un determinado punto inicial. Debido a la naturaleza no lineal de la función objetivo a ser optimizada, será necesario recurrir a un solver no lineal, en nuestro caso, debido a que será el utilizado en los futuros problemas de control NMPC, el solver Ipopt, el cual se basa en los métodos de punto interior desarrollados en secciones anteriores.

El problema matemático puede ser expresado como:

$$\begin{aligned} &\min_{x,y} f(x,y) = (1-x)^2 + 100(y-x^2)^2 \\ &x_0 = 1{,}5,\ y_0 = 1{,}5 \end{aligned} \tag{3.4}$$

El código para la implementación y resolución del problema puede ser expresado como:

```
from pyomo.environ import *

#Creacion del modelo
model=ConcreteModel()

#Declaracion de variables
model.x=Var(initialize=1.5)
model.y=Var(initialize=1.5)

#Funcion objetivo
def rosenbrock(m):
        return (1-m.x)**2+100*(m.y-m.x**2)**2
model.obj=Objective(rule=rosenbrock,sense=minimize)

#Seleccion del solver para problemas LP
```

```
opt=SolverFactory('ipopt')

#Resolucion del problema
results=opt.solve(model,tee=True)
```

3.5.4 Ejemplo Básico DAE

En este ejemplo se intenta ilustrar la manera en la cual se introducen ecuaciones diferenciales en un problema de optimización. En este caso en concreto, el único objetivo será el de introducir la forma en la cual dichas expresiones son incluidas en el modelo en forma de restricciones y aplicar el método de colocación para la resolución de las mismas. Por esta misma razón, no se tendrá en cuenta ninguna función objetivo (dummy objective).

Supóngase así un sistema dado por la expresión:

$$\begin{aligned} &\frac{dz}{dt} = z^2 - 2z + 1 \\ &z(0) = -3 \end{aligned} \tag{3.5}$$

El ejercicio consiste en la resolución de la ecuación utilizando métodos de colocación a lo largo de un único elemento finito con $t \in [0,1]$. El código solución para la implementación del problema se muestra a continuación.

```
from pyomo.environ import *
from pyomo.dae import *

#Creacion del modelo
m=ConcreteModel()
```

```
#Set de tiempo continuo
m.t=ContinuousSet(bounds=(0,1))

#Declaracion de variables
m.z=Var(m.t)

#Declaracion de variables derivativas
m.dzdt=DerivativeVar(m.z,wrt=m.t)

#Objetivo dummy
m.obj=Objective(expr=1)

#Definicion DAE como restriccion
def _zdot(m,t):
    return m.z[t]==m.dzdt[t]**2-2*m.z[t]+1
m.zdot=Constraint(m.t,rule=_zdot)

#Condiciones iniciales
def _init_con(m):
    return m.z[0]==-3
m.init_con=Constraint(rule=_init_con)

#Discretizacion del modelo
discretizer=TransformationFactory('dae.collocation')

#Condiguracion metodo de colocacion
discretizer.apply_to(m,nfe=1,ncp=3,scheme='LAGRANGE-RADAU')
```

3.5.5 Control Óptimo

En este ejemplo se lleva a cabo por primera vez el control de un determinado sistema dinámico. Para ello, se implementará el control óptimo de un sistema aleatorio a

partir del uso de las ecuaciones dinámicas que definen el comportamiento del mismo. En todo problema de control óptimo el objetivo principal es el de llevar a cabo la obtención de una trayectoria óptima para una determinada ley de control de tal forma que se conduzca el sistema a su punto de equilibrio, es decir, que las variables controladas alcancen su punto estacionario, y que el sistema permanezca estable. Para este ejemplo, el problema de optimización utilizado vendrá dado por las siguientes expresiones:

$$
\begin{aligned}
&\text{mín } x_3(t_f) \\
&s.a \\
&\quad \dot{x}_1 = x_2 \\
&\quad \dot{x}_2 = -x_2 + u \\
&\quad \dot{x}_3 = x_1^2 + x_2^2 + 0{,}005u^2 \\
&\quad x_2 - 8(t - 0{,}5)^2 + 0{,}5 \leqslant 0 \\
&\quad x_1(0) = 0, x_2(0) = -1, \\
&\quad x_3(0) = 0, t_f = 3
\end{aligned}
\tag{3.6}
$$

El código para la implementación del problema se muestra a continuación.

```
from pyomo.environ import *
from pyomo.dae import *
import matplotlib.pyplot as plt

#Creacion del modelo
m=ConcreteModel()
m.tf=Param(initialize=4)
m.t=ContinuousSet(bounds=(0,m.tf))

#Declaracion de variables
m.u=Var(m.t,initialize=0)
m.x1=Var(m.t)
m.x2=Var(m.t)
m.x3=Var(m.t)
```

```
#Declaracion de variables derivativas
m.dx1dt=DerivativeVar(m.x1, wrt=m.t)
m.dx2dt=DerivativeVar(m.x2, wrt=m.t)
m.dx3dt=DerivativeVar(m.x3, wrt=m.t)

#Funcion objetivo
m.obj=Objective(expr=m.x3[m.tf])

#Restricciones dinamicas del sistema
def _x1dot(m,t):
    return m.dx1dt[t]==m.x2[t]
m.x1dot=Constraint(m.t,rule=_x1dot)

def _x2dot(m,t):
    return m.dx2dt[t]==-m.x2[t]+m.u[t]
m.x2dot=Constraint(m.t,rule=_x2dot)

def _x3dot(m,t):
    return m.dx3dt[t]==m.x1[t]**2+\
    m.x2[t]**2+0.005*m.u[t]**2
m.x3dot=Constraint(m.t,rule=_x3dot)

def _con(m,t):
    return m.x2[t]-8*(t-0.5)**2+0.5 <=0
m.con=Constraint(m.t,rule=_con)

#Condiciones iniciales
def _init(m):
    yield m.x1[0]==0
    yield m.x2[0]==-1
    yield m.x3[0]==0
m.init_conditions=ConstraintList(rule=_init)

#Seleccion de metodo de colocacion
```

```
TransformationFactory('dae.collocation').apply_to
(m, nfe=7,ncp=6,scheme='LAGRANGE-RADAU')

#Resolucion
results=SolverFactory('ipopt').solve(m)
```

En la siguiente figura se muestran los resultados obtenidos tanto para la variable de control u como para sendas variables controladas x_1 y x_2, viendo como efectivamente el sistema tiende a estabilizarse alcanzando su estado estacionario.

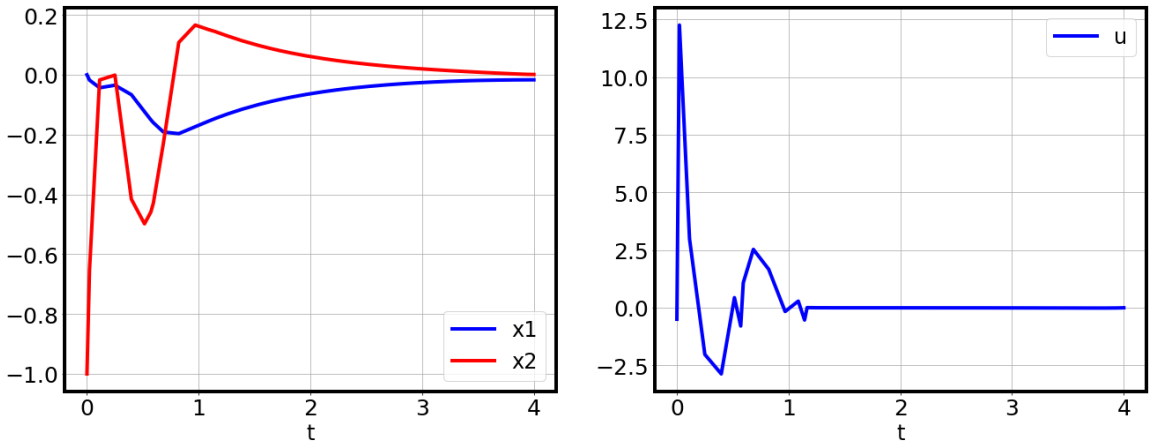


Figura 3.4: Resultados del problema de control óptimo

3.5.6 Estimación Paramétrica

En este ejemplo se lleva a cabo un proceso de estimación paramétrica a partir de las medidas tomadas de una determinada variable. De esta forma, se intentará obtener el valor de la trayectoria de dicha variable usando sólo y exclusivamente las medidas tomadas. El procedimiento llevado a cabo es similar al seguido en un proceso de identificación por mínimos cuadrados, donde se lleva a cabo la optimización del error cuadrático de las medidas con respecto a su valor real desconocido. El problema de

optimización a resolver puede ser expresado matemáticamente como:

$$
\begin{aligned}
&\text{mín} \sum_{t_i} (x_1(t_i) - x1_{meas}(t_i))^2 \\
&s.a \\
&\quad \frac{dx_1}{dt} = x_2 \\
&\quad \frac{dx_2}{dt} = 1 - 2x_2 - x_1 \\
&\quad -1{,}5 \leqslant p_1, p_2 \leqslant 1{,}5 \\
&\quad x_1(0) = p_1,\ x_2(0) = p2
\end{aligned}
\tag{3.7}
$$

El código para la resolución de este problema se muestra a continuación:

```
from pyomo.environ import *
from pyomo.dae import *
import matplotlib.pyplot as plt

#Medidas
measurements = {1:0.264, 2:0.594, 3:0.801, 5:0.959}

#Creacion de modelo
model=ConcreteModel()
model.t=ContinuousSet(initialize=measurements.keys()
,bounds=(0,6))

#Declaracion de variables
model.x1=Var(model.t)
model.x2=Var(model.t)
model.p1=Var(bounds=(-1.5,1.5))
model.p2=Var(bounds=(-1.5,1.5))

#Declaracion de variables derivativas
model.x1dot=DerivativeVar(model.x1,wrt=model.t)
```

```
model.x2dot=DerivativeVar(model.x2)

#Condiciones iniciales
def _init_conditions(model):
    yield model.x1[0]==model.p1
    yield model.x2[0]==model.p2
model.init_conditions=ConstraintList(rule=_init_conditions)

#Restriccioes
def _x1dot(model,i):
    return model.x1dot[i]==model.x2[i]
model.x1dotcon=Constraint(model.t,rule=_x1dot)

def _x2dot(model,i):
    return model.x2dot[i]==1-2*model.x2[i]-model.x1[i]
model.x2dotcon=Constraint(model.t,rule=_x2dot)

#Funcion objetivo
def _obj(model):
    return sum((model.x1[i]-measurements[i])**2
    for i in measurements.keys())
model.obj=Objective(rule=_obj)

#Discretizacion del sistema
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(model,nfe=8,ncp=5)

#Seleccion de solver y resolucion
results=SolverFactory('ipopt').solve(model, tee=True)
```

A continuación se muestran los resultados obtenidos para la estimación del valor de la variable x_1 así como los valores discretos de las medidas tomadas, viendo como el modelo calculado efectivamente se ajusta.

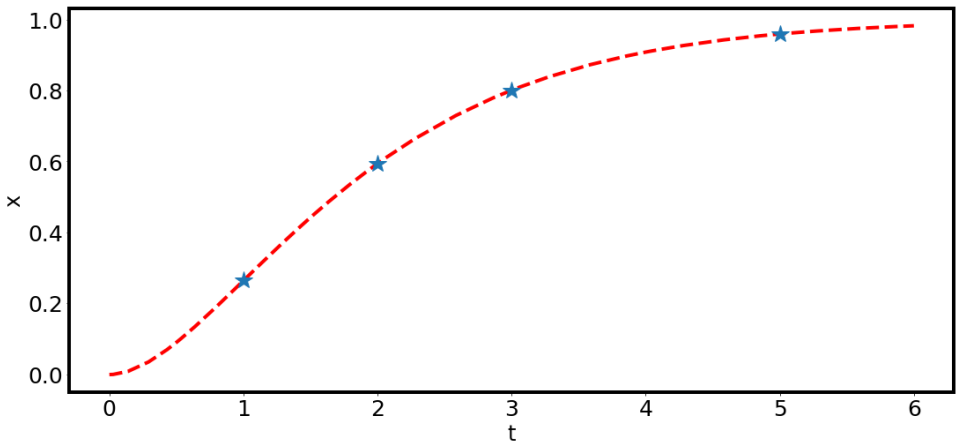


Figura 3.5: Resultados del problema de estimación paramétrica

3.5.7 Ecuación en Derivadas Parciales

En este ejemplo se llevará a cabo la resolución de una ecuación diferencial en derivadas parciales elegida aleatoriamente. El sistema que se desea resolver viene dado matemáticamente como:

$$\begin{aligned} & \pi^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \\ & s.a \\ & \quad u(x,0) = sin(\pi x) \\ & \quad u(0,t) = 0 \\ & \quad \pi e^{-t} + \frac{\partial u}{\partial x}(1,t) = 0 \end{aligned} \tag{3.8}$$

Se muestra a continuación el código para la resolución del problema.

```
from pyomo.environ import *
from pyomo.dae import *
import matplotlib.pyplot as plt

#Creacion del modelo
```

```
m=ConcreteModel()

#Declaracion de sets y parametros
m.pi=Param(initialize=3.1416)
m.t=ContinuousSet(bounds=(0,2))
m.x=ContinuousSet(bounds=(0,1))

#Declaracion de variables
m.u=Var(m.x,m.t)

#Declaracion de variables derivativas
m.dudx=DerivativeVar(m.u,wrt=m.x)
m.dudx2=DerivativeVar(m.u,wrt=(m.x,m.x))
m.dudt=DerivativeVar(m.u,wrt=m.t)

#Restricciones
def _pde(m,i,j):
    if i == 0 or i ==1 or j==0:
        return Constraint.Skip
    return m.pi**2*m.dudt[i,j]==m.dudx2[i,j]
m.pde=Constraint(m.x,m.t,rule=_pde)

def _initcon(m,i):
    if i == 0 or i == 1:
        return Constraint.Skip
    return m.u[i,0]== sin(m.pi*i)
m.initcon=Constraint(m.x,rule=_initcon)

#Condiciones limite
def _limite_inf(m,j):
    return m.u[0,j]==0
m.limite_inf=Constraint(m.t,rule=_limite_inf)

def _limite_sup(m,j):
    return m.pi*exp(-j)+m.dudx[1,j]==0
```

```
m.limite_sup=Constraint(m.t,rule=_limite_sup)

#Seleccion metodo de discretizacion
discretizer=TransformationFactory('dae.finite_difference')
discretizer.apply_to(m,nfe=25,wrt=m.x,scheme='BACKWARD')
discretizer.apply_to(m,nfe=20,wrt=m.t,scheme='BACKWARD')

#Seleccion de solver y resolucion
solver=SolverFactory('ipopt')
results=solver.solve(m,tee=True)
```

3.6 Implementación de Controladores NMPC en Pyomo

En este apartado se llevará a cabo la implementación de controladores predictivos mediante el uso de la herramienta Pyomo, lo cual es el objetivo principal de este trabajo. De esta forma, nos centraremos en analizar el uso de de dicha herramienta más que en el proceso de diseño y sintonía del controlador, lo cual es de sobra conocido, y de los que ya se ha hablado en el capítulo dedicado al diseño de controladores predictivos no lineales.

Para la resolución del problema de optimización descrito, es necesario la utilización de un determinado solver que lleve a cabo tal labor. En los controladores predictivos desarrollados, debido a su gran versatilidad y buenas prestaciones, y principalmente, a que se trata de un software de libre distribución, se ha optado por la utilización del solver IPOPT, basado en métodos de punto interior como los descritos en el capítulo dedicado a algoritmos numéricos de optimización.

En la siguiente sección se describe el proceso y la metodología seguida para el desarrollo de controladores predictivos a partir de la implementación del modelo en Pyomo donde se define la función objetivo a optimizar así como las restricciones dinámicas que definen el comportamiento del sistema.

3.6.1 Metodología de desarrollo

Para llevar a cabo la implementación de controladores predictivos, el procedimiento seguido será el siguiente:

- En primer lugar, se llevará a cabo de declaración y definición de todas y cada una de las constantes utilizadas por el modelo cuyo valor será fijo a lo largo del proceso de optimización.
- Hecho esto, se realizará la definición del modelo en Pyomo del problema de optimización a resolver, incluyéndose así las variables dinámicas del problema a ser controladas, así como cada una de sus derivadas, las variables que serán utilizadas como señales de control, la función objetivo a ser optimizada, y las restricciones dinámicas del sistema que definen el comportamiento de cada una de las variables controladas. Además, será necesario definir como set continuo el intervalo temporal del que depende cada una de las variables dinámicas descritas anteriormente. Este tiempo continuo ha de ser indicado en la declaración de las mismas, así como el valor inicial de éstas, el cual será el punto de partida de la trayectoria predicha.
- Una vez que se ha llevado a cabo la descripción del problema a resolver a modo de modelo de Pyomo, será necesario proceder a llevar a cabo la resolución de dicho problema, para lo cual se ha de aplicar primeramente un método de integración del sistema, principalmente método de colocación como los descritos en la sección dedicada a métodos simultáneos de optimización dinámica, la selección del solver que se ha de utilizar, y la aplicación de dicho solver al modelo descrito.
- Mediante la descripción del modelo en Pyomo, la discretización dinámica del mismo, y la aplicación de un determinado solver, se obtendrá como salida la trayectoria predicha para cada variable a lo largo del intervalo de tiempo establecido. El procedimiento a seguir a partir de aquí será el de realizar dicha predicción en bucle para un determinado número de pasos, de tal forma que en cada una de las iteraciones del bucle se actualice para la siguiente iteración el valor inicial de las variables como el primer valor de la trayectoria predicha, tomando éste como los primeros npc puntos de la trayectoria, donde npc es el número de puntos de colocación del proceso de discretización (podría haberse elegido otro).

- En cada una de las iteraciones se irá guardando el punto inicial seleccionado en cada una de ellas, de tal forma que la trayectoria descrita por el sistema controlado mediante el controlador NMPC será la descrita por dichos puntos iniciales obtenidos en cada una de las predicciones.

En el siguiente diagrama se muestra de manera gráfica el proceso descrito anteriormente.

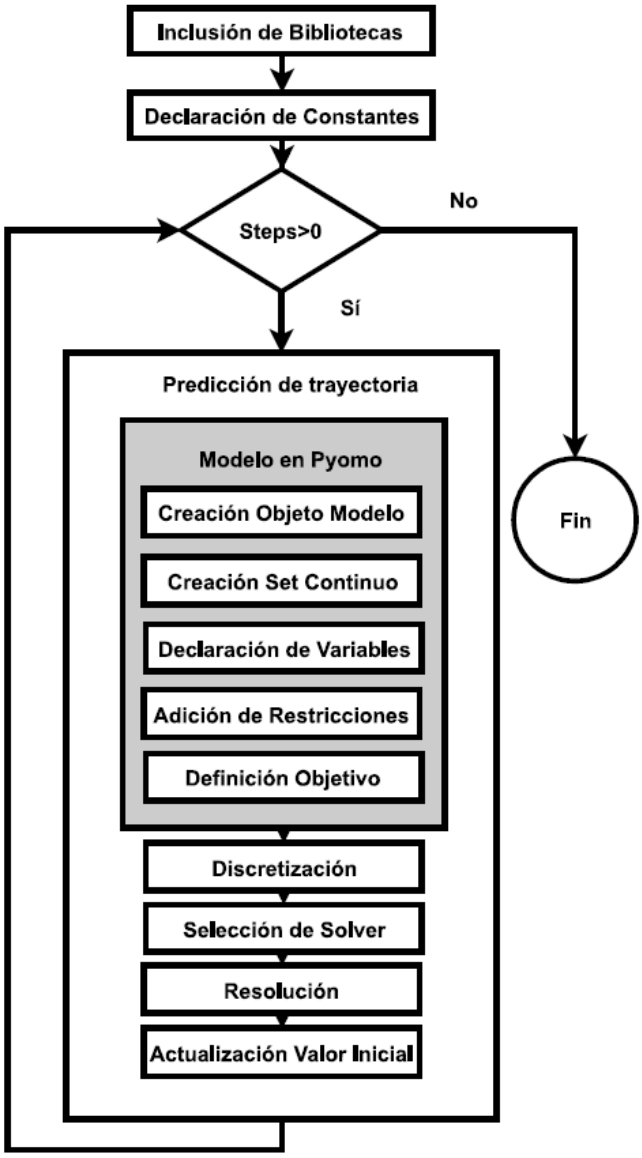


Figura 3.6: Proceso de diseño de controladores NMPC en Pyomo

Siguiendo esta metodología, se desarrollan a continuación los ejemplos de controladores predictivos no lineales en tiempo continuo mediante Pyomo desarrollados en el presente trabajo.

3.6.2 Ejemplo Planta de Cuatro Tanques

Descripción del sistema

El primero de los ejemplos de controlador predictivo NMPC en Pyomo desarrollado será aplicado a la planta de cuatro tanques propuesta en [6], y la cual ha sido ampliamente utilizada debido a su interés como problema de control multivariable. En la siguiente imagen se muestra un esquema de la planta.

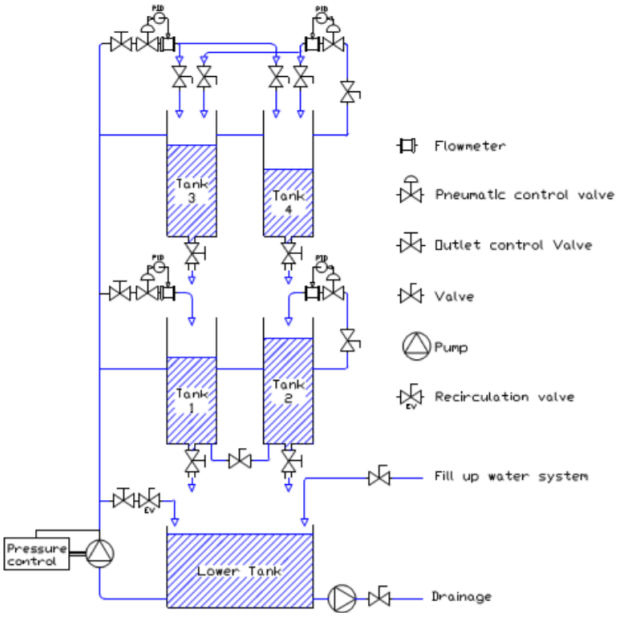


Figura 3.7: Esquema de la planta de cuatro tanques

Como puede observarse, la planta cuenta con cuatro depósitos de almacenamiento de agua, dos superiores y dos inferiores, y los cuales desaguan en un quinto tanque a modo de recipiente sumidero. Partiendo de dicho tanque, sendas bombas son encargadas de impulsar los caudales q_a y q_b, los cuales pasan a través de dos válvulas de tres vías encargadas de realizar la división del caudal entre dos conducciones separadas, una parte es enviada por una de las ramas y el resto por la otra (fracciones γ_a y γ_b.

Así, se tiene que un determinado caudal $\gamma_a q_a$ entra al tanque 1 y el caudal $(1-\gamma_a)q_a$ en el tanque cuatro. De igual manera, un caudal $\gamma_b q_b$ entra al tanque dos y el caudal $(1-\gamma_b)q_b$ en el tanque tres. El tanque cuatro descarga sobre el tanque tres, el tanque tres lo hace sobe el tanque uno, y estos dos tanques inferiores sobre el tanque sumidero.

El modelo de la planta puede ser descrito mediante la utilización de las siguientes ecuaciones diferenciales de primeros principios.

$$\begin{aligned} A\frac{dh_1}{dt} &= -a_1\sqrt{2gh_1} + a_3\sqrt{2gh_3} + \frac{\gamma_a q_a}{3600} \\ A\frac{dh_2}{dt} &= -a_2\sqrt{2gh_2} + a_4\sqrt{2gh_4} + \frac{\gamma_b q_b}{3600} \\ A\frac{dh_3}{dt} &= -a_3\sqrt{2gh_3} + (1-\gamma_b)\frac{q_b}{3600} \\ A\frac{dh_4}{dt} &= -a_4\sqrt{2gh_4} + (1-\gamma_a)\frac{q_a}{3600} \end{aligned} \tag{3.9}$$

donde h_i es la altura en el tanque i en metros y a_i la sección del orificio de descarga del tanque i en metros cuadrados. La sección de todos los tanques es la misma y toma un valor de A metros cuadrados. La unidad de tiempo utilizada en el modelo es el segundo. En la siguiente tabla se resumen los datos del problema:

Constante	**Valor**
A	0.03
a_1	$1{,}3104x10^{-4}$
a_2	$1{,}5074x10^{-4}$
a_3	$9{,}2673x10^{-4}$
a_4	$8{,}8164x10^{-5}$
γ_a	0.3
γ_b	0.4
$h_{i_{min}}$	0.2
$h_{i_{max}}$	1.2
$q_{a,b_{min}}$	0
$q_{a,b_{max}}$	2.5

Cuadro 3.3: Parámetros planta cuatro tanques

Implementación en Pyomo

En este primer problema, por ser el primero de los controladores predictivos desarrollados, se detallarán todas y cada una de las partes que componen el código desarrollado para su correcta comprensión. El primer paso en cualquier problema desarrollado en Python es la importación de las bibliotecas y librerías de funciones que serán utilizadas, tal y como se muestra a continuación.

```
#Libreria de funciones matematicas
import math

#Librerias de Pyomo
from pyomo.environ import *
from pyomo.dae import *

#Libreria de funciones para tratamiento de arrays
import numpy as np
```

Una vez que se han incluidos todas las librerías a ser utilizadas, se llevará a cabo la creación de una función, dentro de la cual se incluirán todos los procesos necesarios para la implementación del control predictivo, es decir, modelo en Pyomo, discretización del sistema, aplicación del solver matemático, y actualización de valores iniciales en bucle. Dicha función tomará como argumentos los valores iniciales de las variables del sistema, el horizonte de predicción en cada una de las predicciones, y el número de pasos que dará el sistema.

```
def cuatro_tanques(steps, hor,h1_init,h2_init,h3_init,h4_init,
gammaA_init,gammaB_init,qa_init,qb_init):
```

Definida la función se tendrá que ser llamada cada vez que se desee simular el proceso de optimización, se definirán, fuera del bucle, las constantes del modelo cuyo

valor permanece inalterado durante el proceso de optimización. Además, se incluyen los valores de las variables cuando el sistema ha alcanzado el estado estacionario, es decir, los valores del punto de equilibrio o referencia (el cual ha de ser previamente calculado).

```
#Parametros

#Gravedad
g=9.81

#Orificios
a1=1.3105e-4
a2=1.5074e-4
a3=9.2673e-5
a4=8.8164e-5

#Area de los tanques
 A=0.03

#Punto de equilibrio

#Alturas
h1_o=0.313788
h2_o=0.405569
h3_o=0.237628
h4_o=0.558387

#Aperturas
gammaA_o=0.3
gammaB_o=0.4

#Caudales
qa_o=1.5
qb_o=1.2
```

Por último, antes de proceder a implementar el bucle de control, se llevará a cabo la declaración de los vectores en los cuales se almacenarán los datos históricos para su posterior graficación. En cada una de las iteraciones del blucle, se almacenará en dichos vectores los valores de la trayectoria predicha en el primer paso de la misma de cada una de las variables, tal y como previamente se comentó.

```
#Vectores para almacenamiento de datos historicos

h1_final=[]
h2_final=[]
h3_final=[]
h4_final=[]

gammaA_final=[]
gammaB_final=[]

qa_final=[]
qb_final=[]

#Inclusion del primer valor inicial

h1_final.append(h1_init)
h2_final.append(h2_init)
h3_final.append(h3_init)
h4_final.append(h4_init)

gammaA_final.append(gammaA_init)
gammaB_final.append(gammaB_init)

qa_final.append(qa_init)
qb_final.append(qb_init)
```

Hecho esto, se está ya en condiciones de llevar a cabo la implementación del bucle de control. Para ello, se ha de seleccionar el número de pasos en el tiempo que se desean dar, el cual será igual al número de procesos de optimización que se realizarán.

```
#Implementacion del bucle
while (steps>0):
```

Dentro del bucle de control, el primer paso a dar es la implementación del modelo en Pyomo del sistema sobre el cual se desea realizar el control. Se muestra a continuación dicho proceso. En el modelo, primeramente se llevará a cabo declaración del mismo así como las variables y variables derivativas del mismo, tal y como se muestra a continuación.

```
#Creacion del modelo
m=ConcreteModel()

#Set de tiempo continuo
m.t =ContinuousSet(bounds=(0,hor))

#Variables del sistema

m.h1=Var(m.t,bounds=(0.2,1.2))
m.h2=Var(m.t,bounds=(0.2,1.2))
m.h3=Var(m.t,bounds=(0.2,1.2))
m.h4=Var(m.t,bounds=(0.2,1.2))

m.h1sqrt=Var(m.t,bounds=(0.2,1.2))
m.h2sqrt=Var(m.t,bounds=(0.2,1.2))
m.h3sqrt=Var(m.t,bounds=(0.2,1.2))
m.h4sqrt=Var(m.t,bounds=(0.2,1.2))

m.gammaA=Var(m.t,bounds=(0.3,0.3)
 ,initialize=gammaA_init)
m.gammaB=Var(m.t,bounds=(0.4,0.4)
 ,initialize=gammaB_init)

m.qa=Var(m.t,bounds=(0,3),initialize=qa_init)
m.qb=Var(m.t,bounds=(0,3),initialize=qb_init)
```

```
#Variables derivativas

m.h1_dot=DerivativeVar(m.h1,wrt=m.t)
m.h2_dot=DerivativeVar(m.h2,wrt=m.t)
m.h3_dot=DerivativeVar(m.h3,wrt=m.t)
m.h4_dot=DerivativeVar(m.h4,wrt=m.t)
```

Una vez que se ha llevado a cabo la definición de las variables del sistema, el siguiente paso es el de definir el comportamiento dinámico de las variables controladas, en este caso, de las alturas de cada uno de los tanques. Para esto, las ecuaciones dinámicas que rigen el comportamiento del sistema, dadas por (4.9) serán definidas como restricciones del mismo, es decir, se ha de llevar a cabo la optimización de la función objetivo dado que el comportamiento de las variables del sistema ha de venir dado por dichas expresiones. Debido a que en las expresiones aparecen en términos que hacen referencia a las raíces cuadradas de las variables, se ha llevado a cabo la inclusión también de restricciones para la definición de nuevas variables cuadráticas que evitan el uso de los términos radicales, lo cual disminuye el costo computacional.

```
#Restricciones para la definicion de variables
cuadraticas

def _h1sqrt(m,t):
                if t == 0:
                        return Constraint.Skip
                return m.h1sqrt[t]**2-m.h1[t] == 0
m.h1sqrtcon = Constraint(m.t, rule=_h1sqrt)

def _h2sqrt(m,t):
                if t == 0:
                        return Constraint.Skip
                return m.h2sqrt[t]**2-m.h2[t] == 0
m.h2sqrtcon = Constraint(m.t, rule=_h2sqrt)
```

```
def _h3sqrt(m,t):
                if t == 0:
                        return Constraint.Skip
                return m.h3sqrt[t]**2-m.h3[t] == 0
m.h3sqrtcon = Constraint(m.t, rule=_h3sqrt)

def _h4sqrt(m,t):
                if t == 0:
                        return Constraint.Skip
                return m.h4sqrt[t]**2-m.h4[t] == 0
m.h4sqrtcon = Constraint(m.t, rule=_h4sqrt)

#Restricciones para la definicion del comportamiento
 dinamico de las alturas

#Comportamiento dinamico de h1
def d_h1(m,t):
    if t==0:
        return m.h1[0]==h1_init
    else:
        return m.h1_dot[t]==(1/A)*(-a1*sqrt(2*g)
        *m.h1sqrt[t]+a3*sqrt(2*g)*m.h3sqrt[t]
        + m.gammaA[t]*m.qa[t]*(1/3600))
m.h1con=Constraint(m.t,rule=d_h1)

#Comportamiento dinamico de h2
def d_h2(m,t):
    if t==0:
        return m.h2[0]==h2_init
    else:
        return m.h2_dot[t]==(1/A)*(-a2*sqrt(2*g)
        *m.h2sqrt[t]+a4*sqrt(2*g)*m.h4sqrt[t]
        +m.gammaB[t]*m.qb[t]*(1/3600))
m.h2con=Constraint(m.t,rule=d_h2)
```

```
#Comportamiento dinamico de h3
def d_h3(m,t):
    if t==0:
        return m.h3[0]==h3_init
    else:
        return m.h3_dot[t]==(1/A)*(-a3*sqrt(2*g)
        *m.h3sqrt[t]+(1-m.gammaB[t])*m.qb[t]*(1/3600))
m.h3con=Constraint(m.t,rule=d_h3)

#Comportamiento dinamico de h4
def d_h4(m,t):
    if t==0:
        return m.h4[0]==h4_init
    else:
        return m.h4_dot[t]==(1/A)*(-a4*sqrt(2*g)
        *m.h4sqrt[t]+(1-m.gammaA[t])*m.qa[t]*(1/3600))
m.h4con=Constraint(m.t,rule=d_h4)
```

Por último, además de las restricciones anteriores se añadirán sendas restricciones para indicar que las variables de control relativas a la apertura de las válvulas siempre han de tomar un valor constante. Esto es así debido a que su modificación es manual y ha de ser llevada a cabo por el usuario, por lo cual se establecerá que el valor de dicha apertura sea constante ya que no es posible llevar a cabo una modificación continua de su valor.

```
#Restriccion del comportamiento de las valvulas

def _gammaA(m,t):
    return m.gammaA[t]==0.3
m.gammaAcon=Constraint(m.t,rule=_gammaA)

def _gammaB(m,t):
    return m.gammaB[t]==0.4
m.gammaBcon=Constraint(m.t,rule=_gammaB)
```

Por último, el único elemento que queda por añadir al modelo en Pyomo es la función objetivo que se desea optimizar. En este caso, se ha seleccionado como función objetivo un coste cuadrático en el cual se penaliza el cuadrado de la desviación de cada variable con respecto a su valor de referencia, tanto para variables controladas como variables de control, multiplicadas por sendas matrices de ponderación Q y R previamente diseñadas. En este caso, debido al buen funcionamiento proporcionado, se ha optado por la no inclusión de coste terminal, algo que sí se hará en ejemplos posteriores más susceptibles a inestabilizarse.

```
def _obj(m):
    return sum(Q[1]*(m.h1[t]-h1_o)**2
               +Q[2]*(m.h2[t]-h2_o)**2
               +Q[3]*(m.h3[t]-h3_o)**2
               +Q[4]*(m.h4[t]-h4_o)**2
               +R[1]*(m.qa[t]-qa_o)**2
               +R[2]*(m.qb[t]-qb_o)**2 for t in m.t)
m.obj=Objective(rule=_obj,sense=minimize)
```

Definidos ya todos y cada uno de los elementos que componen el modelo en Pyomo, lo único que queda por hacer es llevar a cabo la discretización del modelo, selección del solver, y aplicación del mismo. Se muestra a continuación dicho proceso.

```
#Discretizacion del modelo
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(m,nfe=15,ncp=col,
scheme='LAGRANGE-LEGENDRE')

#Seleccion de solver
solver=SolverFactory('Ipopt')

#Resolucion
results = solver.solve(m,tee=True)
```

Finalmente, una vez que se ha llevado a cabo la resolución del modelo para el cálcu-

lo de la trayectoria predicha en un determinado instante, se ha de actualizar el valor inicial para la resolución del problema en el siguiente instante temporal, almacenar dicho valor en el vector de almacenamiento de históricos, y disminuir el número de pasos a dar en el bucle. Dicho proceso se muestra a continuación.

```
#Actualizacion valor inicial

h1_init=m.h1[col].value
h2_init=m.h2[col].value
h3_init=m.h3[col].value
h4_init=m.h4[col].value

gammaA_init=m.gammaA[col].value
gammaB_init=m.gammaB[col].value

qa_init=m.qa[col].value
qb_init=m.qb[col].value

#Almacenamiento de datos

h1_final.append(value(m.h1[col]))
h2_final.append(value(m.h2[col]))
h3_final.append(value(m.h3[col]))
h4_final.append(value(m.h4[col]))

gammaA_final.append(value(m.gammaA[col]))
gammaB_final.append(value(m.gammaB[col]))

qa_final.append(value(m.qa[col]))
qb_final.append(value(m.qb[col]))

#Disminucion del numero de pasos

steps=steps-1
```

Una vez que el código anterior es ejecutado, y para la sintonía llevada a cabo, se han obtenido las siguientes trayectorias a lo largo del intervalo de tiempo establecido y para un determinado valor inicial seleccionado, viéndose como efectivamente tanto variables de control como variables controladas tienden a su punto de equilibrio.

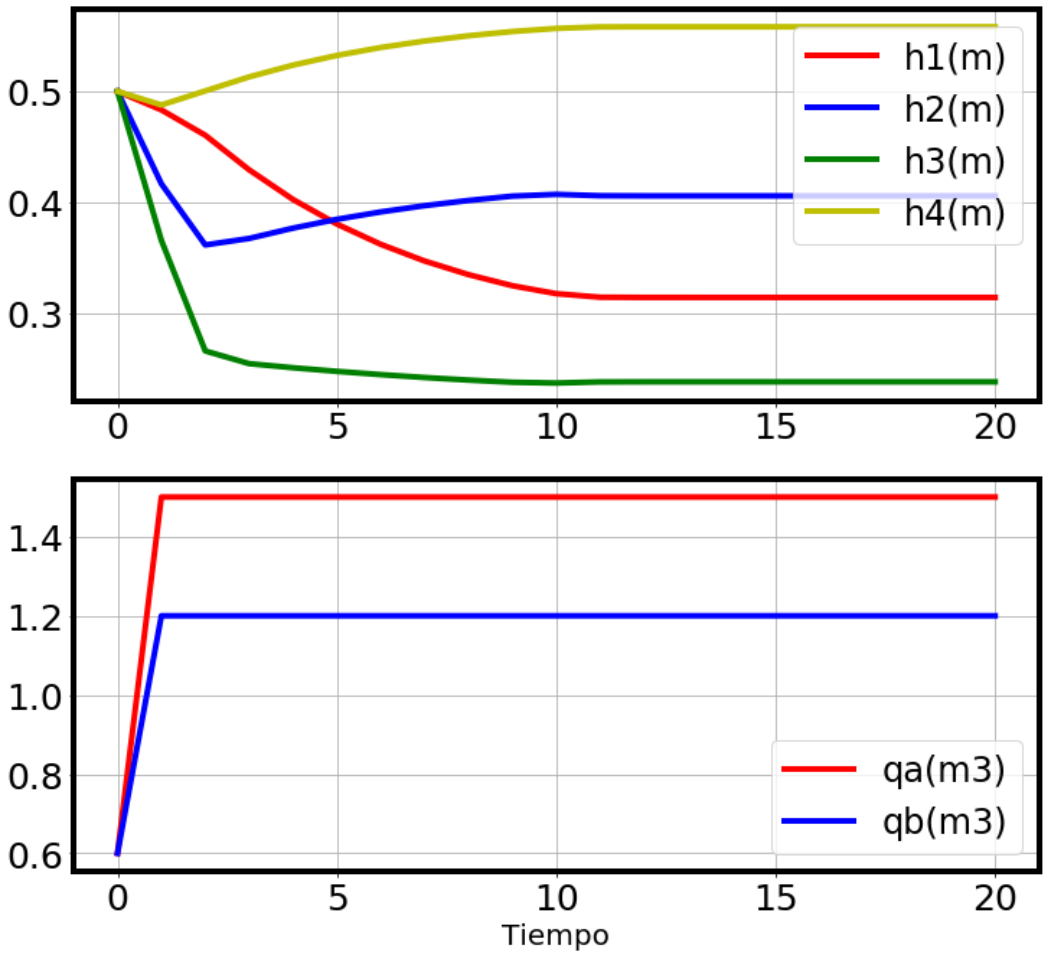


Figura 3.8: Resultados planta cuatro tanques (Ejemplo 1)

Se puede ver como todas y cada una de las variables controladas ha alcanzado el punto de referencia o de equilibrio previamente calculado y especificado. El controlador es pues capaz de conducir el sistema hasta dicho punto haciendo además que la variable de control alcance su valor de referencia. Para que el controlador funcione bien, el sistema ha de ser conducido hasta dicho punto independientemente del valor inicial de las variables. A continuación de muestra como esto efectivamente ocurre,

habiéndose elegido un punto de inicio aleatorio para cada una de las variables controladas así como para las variables de control.

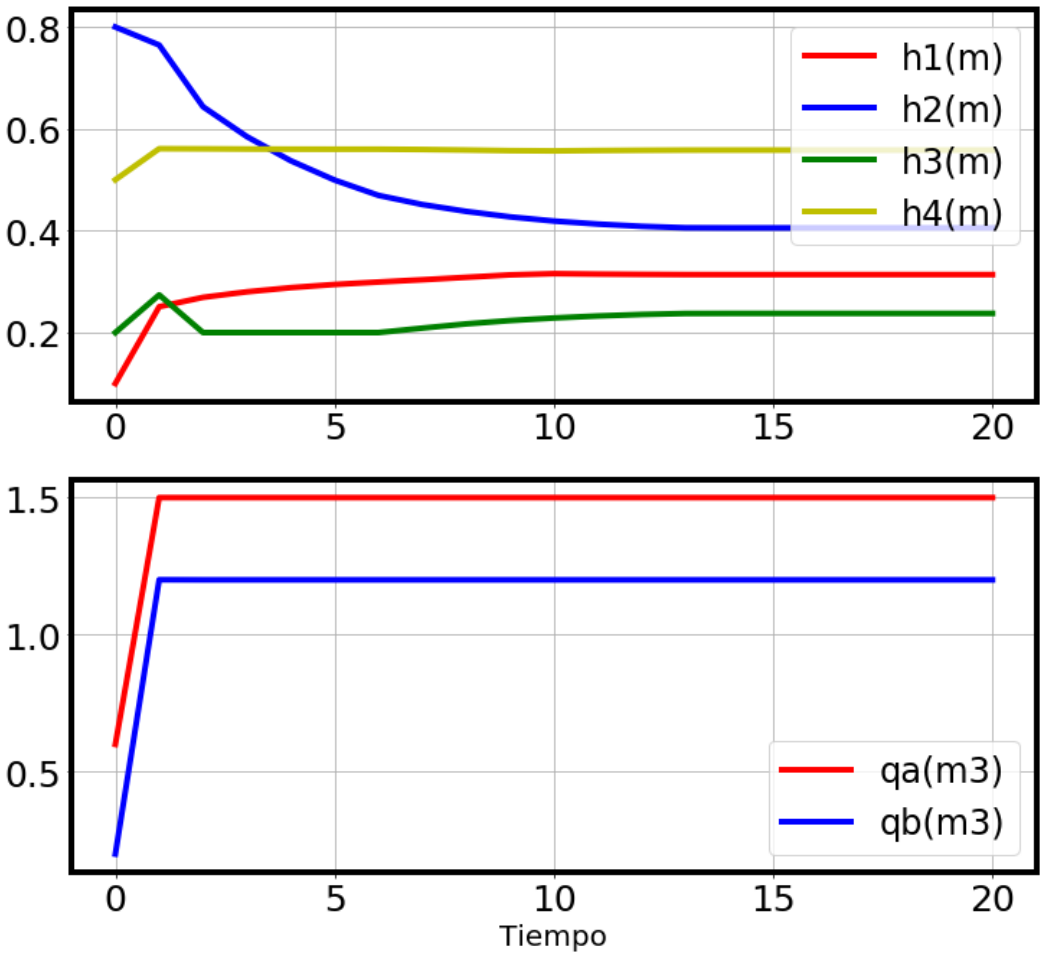


Figura 3.9: Resultados planta cuatro tanques (Ejemplo 2)

Como puede observarse en las gráficas anteriores, el optimizador encuentra que el punto de operación óptimo es aquel en el que la señal de control toma como valor su punto de referencia, por lo cual intenta conducir el mismo lo más rápidamente posible hasta éste. En ocasiones, este comportamiento puede no ser el deseado o el más adecuado, y es preferible que el sistema sea más lento a consta de señales de control más suaves. Para ello son dos las alternativas a la hora de llevar a cabo la sintonía del controlador, un cambio de valores en las matrices de ponderación de la función objetivo que penalice el la variación de la señal de control, o la modificación del tiempo de

integración del sistema.

3.6.3 Ejemplo Tanque Reactor Continuamente Agitado

Descripción del sistema

En este ejemplo se llevará a cabo el diseño de un controlador para un tanque reactor continuamente agitado, tal y como el descrito en [15]. Los reactores continuamente agitados, de ahora en adelante, CSTR, son uno de los problemas más extendidos en al ámbito de la ingeniería química, y han sido ampliamente utilizados como ejemplo de sistema a controlar mediante el uso de técnicas de control predictivo. En concreto, se utilizará un reactor CSTR no isotérmico en el cual un determinado reactivo A es convertido e un determinado producto B a través de de una reacción elemental de segundo orden. Dicho reactivo es introducido en el reactor a través de un flujo de entrada con una determinada concentración C_A, un determinado flujo volumétrico F, y una determinada temperatura T. Se asume que el contenido del reactor se encuentra uniformemente distribuido. El calor exotérmico producido durante el proceso se controla a través de una coraza de refrigereción por la que circula un determinado refrigerante a temperatura Tc.

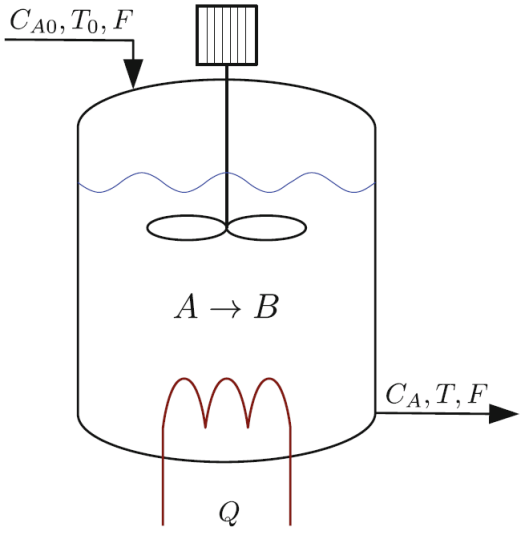


Figura 3.10: Esquema Reactor CSTR

Mediante la aplicación de ecuaciones de primeros principios, y asunciones estándar de modelado, como densidad de fluido y capacidad calorífica constantes, denotadas respectivamente ρ y C_p respectivamente, y variación de dependencia de Arrhenius de la reacción de la temperatura, la evolución del sistema puede ser descrita mediante

las siguientes ecuaciones diferenciales ordinarias, las cuales describen la variación de concentración y temperatura en el CSTR.

$$\begin{aligned}\frac{dC_A}{dt} &= \frac{F}{V_R}(C_{AF} - C_A(t)) - k_0 e^{-E/RT(t)}(C_A(t)) \\ \frac{dT}{dt} &= \frac{F}{V_R}(T_F - T(t)) - \frac{-\Delta H k_0}{\rho C_p} e^{-E/RT(t)} C_A^2 + \frac{UA}{\rho C_p V_R} * (T_c(t) - T(t))\end{aligned} \tag{3.10}$$

El objetivo principal del diseño del controlador es el de controlar la cantidad de concentración del reactivo A. Para ello, se tendrá como variable de control la temperatura del refrigerante T_c, la cual modificará la temperatura del tanque acelerándose o desacelerándose la reacción y en consecuencia la concentración de reactivo y producto.

Concepto	**Símbolo**	**Valor**
Densidad del flujo	ρ	1000
Calor específico	C_p	0.239
Entalpía	ΔH	$5x10^4$
Constante de Arrhenius	ER	8750
Velocidad de la reacción	k_0	$7{,}2x10^{10}$
Coeficiente de transferencia de calor	UA	$5x10^4$
Volumen	V	100
Temperatura de entrada del flujo	T_f	350
Concentración molar	C_f	1
Flujo de entrada	F	1000

Cuadro 3.4: Parámetros planta reactor CSTR

Implementación de Pyomo

Descrito ya el sistema a ser controlado, así como definidas las ecuaciones dinámicas que rigen su comportamiento, se está en disposición de llevar a cabo la implementación del modelo en Pyomo y posterior aplicación de la estrategia de control predictivo. La metodología y pasos seguido para dicho desarrollo serán los mismos que los llevados a cabo en el ejemplo anterior, por lo que simplemente se mostrarán las distintas partes del código.

Importación de librerías.

```
#Libreria para funciones matematicas
import math

#Librerias de Pyomo
from pyomo.environ import *
from pyomo.dae import *

#Libreria de funciones para tratamiento de arrays
import numpy as np
```

Definición de la función global para la resolución del problema.

```
def ReactorNMPC(steps,hor,CA_init,T_init,Tc_init):
```

Declaración de valores constantes.

```
    #Densidad del liquido
    rho=1000

    #Calor especifico medio
    Cp=0.239

    #Entalpia
    HR=5*(10**4)

    #Constante de Arrhenius
    E_R=8750
```

```
#Constante de velocidad de reaccion
K_0=7.2*(10**10)

#Coeficiente de transferencia de calor
UA=5*(10**4)

#Volumen
V=100

#Concentracion molar
CAf=1

#Temperatura de entrada del flujo
Tf=350

#Caudal de entrada
q_in=1000

#Punto de equilibrio

#Concentracion de equilibrio
CAref=0.92

#Temperatura de equilibrio
Tref=346.80

#Temperatura de quilibrio dle refrigerante
Tcref=331.54
```

Vectores para el almacenamiento de datos históricos e inclusión del primer valor inicial.

```
#Vectores para almacenamiento de datos hitoricos
```

```
CA_final=[]
T_final=[]
Tc_final=[]

#Inclusion del primer valor inicial

CA_final.append(value(CA_init))
T_final.append(value(T_init))
Tc_final.append(value(Tc_init))
```

Implementación del bucle.

```
#Implementacion del bucle
while (steps > 0):
```

Definición del modelo en Pyomo.

```
    #Creacion del modelo
    m=ConcreteModel()

    #Set de tiempo continuo
    m.t = ContinuousSet(bounds=(0,horizonte))

    #Variables del sistema

    m.CA=Var(m.t,bounds=(0,1))
    m.T=Var(m.t,bounds=(280,370))
    m.Tc=Var(m.t,bounds=(280,370),initialize=Tc_init)

    #Variables derivativas

    m.CAdot=DerivativeVar(m.CA,wrt=(m.t))
```

```
m.Tdot=DerivativeVar(m.T,wrt=(m.t))
```

Definición de restricciones dinámicas que definen el comportamiento del sistema.

```
        #Comportamiento de CA
def d_CA(m,l):
            if l==0:
                return m.CA[0]==CA_init
            else:
                return m.CAdot[l]==(q_in/V)
                *(CAf-m.CA[l])-K_0
                *exp(-E_R/(m.T[l]))*(m.CA[l])**2
m.CAdotcon=Constraint(m.t,rule=d_CA)

#Comportamiento de T
def d_T(m,l):
             if l==0:
                 return m.T[0]==T_init
             else:
                 return m.Tdot[l]==(q_in/V)
                 *(Tf-m.T[l])+((-HR*K_0)/(rho*Cp))
                 *exp(-E_R*(m.T[l]))*m.CA[l]
                 +(UA/(V*rho*Cp))*(m.Tc[l]-m.T[l])
m.Tdotcon=Constraint(m.t,rule=d_T)
```

A continuación se realiza la definición de la función objetivo a ser optimizada. En este caso, además del coste de penalización de las variables de estado del sistema así como de la señal de control, se ha incluido un término de penalización del coste terminal para garantizar la estabilidad del sistema.

```
#Funcion objetivo
def _obj(m):
```

```
        return sum((Q1*(m.CA[l]-CAref)**2
        +Q2*(m.T[l]-Tref)**2
        +R1*(m.Tc[l]-Tcref)**2) for l in m.t)
        +P1*(m.CA[horizonte]-CAref)**2
        +P2*(m.T[horizonte]-Tref)**2
    m.obj=Objective(rule=_obj,sense=minimize)
```

Discretización del modelo y aplicación del solver.

```
#Discretizacion del modelo
discretizer=TransformationFactory('dae.collocation')
discretizer.apply_to(m,nfe=fin,ncp=col,
scheme='LAGRANGE-RADAU')

#Seleccion del solver
solver=SolverFactory('ipopt')

#Resolucion
results = solver.solve(m,tee=True)
```

Almacenamiento del primer valor y actualización del valor inicial, y disminución del número de iteraciones a dar por el bucle.

```
#Actualizacion del valor inicial
CA_init=m.CA[ren].value
T_init=m.T[ren].value
Tc_init=m.Tc[ren].value

#Almacenamiento del primer valor
CA_final.append(value(m.CA[ren]))
T_final.append(value(m.T[ren]))
Tc_final.append(value(m.Tc[ren]))
```

```
#Disminucion del numero de pasos
steps=steps-1
```

Para que el comportamiento del sistema sea el adecuado, es necesario llevar a cabo una buena selección del número de elementos finitos y puntos de colocación durante el proceso de discretización. Por lo general, se hará coincidir el número de elementos finitos con el horizonte de predicción, lo cual hará que cada uno de dichos elementos finitos tenga longitud temporal unitaria. En cuanto al número de elementos de colocación, un mayor número de éstos hará que la trayectoria discretizada se asemeje más a la trayectoria real continua, teniendo esto como inconveniente que se incrementará considerablemente el coste computacional. Por ello, es necesario el encontrar una solución de compromiso. Se muestran a continuación los resultados obtenidos para un horizonte de predicción de 10, y tanto un número de elementos finitos como puntos de colocación igual al horizonte.

Nuevamente se lleva a cabo el control del sistema para un punto inicial distinto para comprobar el correcto funcionamiento del controlador.

3.6.4 Ejemplo Reactor Múltiple

En esta ocasión se llevará a cabo el control de un sistema formado por dos rectores y un separador como el descrito en [8]. Un determinado flujo de reactivo A puro es añadido a uno de los reactores y convertido en producto B mediante una reacción de primer orden. Dicho producto se pierde debido a una reacción paralela también de primer orden dando lugar al producto C. Por otro lado, existe un separador en el cual se lleva a cabo la un proceso de destilación. El destilado del separador es dividio y redirigido al primer reactor. En la siguiente figura se muestra el esquema de la planta.

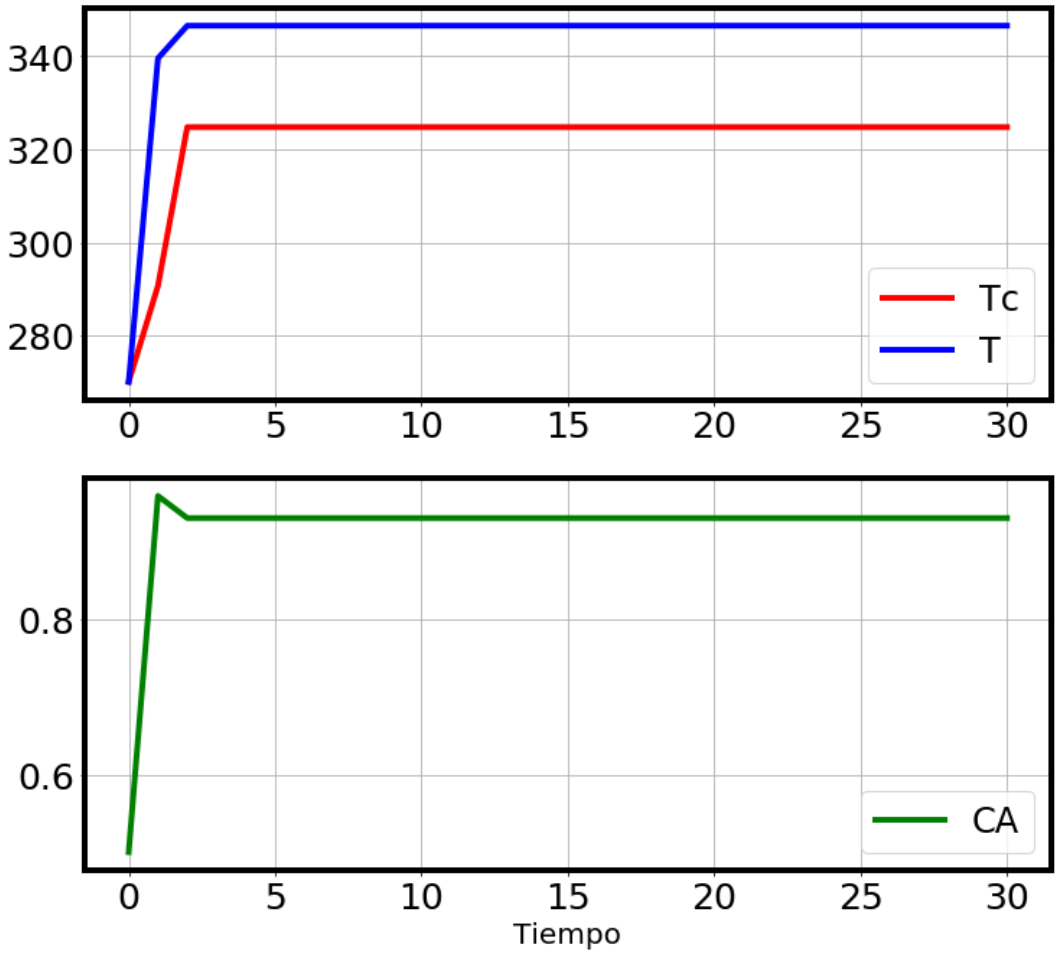


Figura 3.11: Resultados planta CSTR(Ejemplo 1)

La ecuaciones dinámicas que definen el comportamiento del sistema son:

$$\frac{dH_1}{dt} = \frac{1}{\rho A_1}(F_{f1} + F_R - F_1)$$
$$\frac{dx_{A1}}{dt} = \frac{1}{\rho A_1 H_1}(F_{f1}x_{A0} + F_R x_{AR} - F_1 x_{A1}) - k_{A1}x_{A1}$$
$$\frac{dx_{B1}}{dt} = \frac{1}{\rho A_1 H_1}(F_R x_{BR} - F_1 x_{B1}) + k_{A1}x_{A1} - k_{B1}x_{B1}$$
$$\frac{dT_1}{dt} = \frac{1}{\rho A_1 H_1}(F_{f1}T_0 + F_R T_R - F_1 T_1) - \frac{1}{C_p}(k_{A1}x_{A1}\Delta H_A + k_{B1}\Delta H_B + \frac{Q_1}{\rho A_1 C_p H_1}$$
$$\frac{dH_2}{dt} = \frac{1}{\rho A_2}(F_{f2} + F_1 - F_2)$$
$$\frac{dx_{A2}}{dt} = \frac{1}{\rho A_2 H_2}(F_{f2}x_{A0} + F_1 x_{A1} - F_2 x_{A2}) - k_{A1}x_{A1}$$
$$\frac{dx_{B2}}{dt} = \frac{1}{\rho A_2 H_2}(F_1 x_{B1} - F_2 x_{B2}) + k_{A2}x_{A2} - k_{B2}x_{B2}$$
$$\frac{dT_2}{dt} = \frac{1}{\rho A_2 H_2}(F_{f2}T_0 + F_1 T_R - F_2 T_2) - \frac{1}{C_p}(k_{A2}x_{A2}\Delta H_A + k_{B2}\Delta H_B + \frac{Q_2}{\rho A_2 C_p H_2}$$
$$\frac{dH_3}{dt} = \frac{1}{\rho A_3}(F_2 - F_D - F_R - F_3)$$
$$\frac{dx_{A3}}{dt} = \frac{1}{\rho A_3 H_3}(F_2 x_{A2} - F_D x_{A3} - F_R x_{AR} - F_3 x_{A3})$$

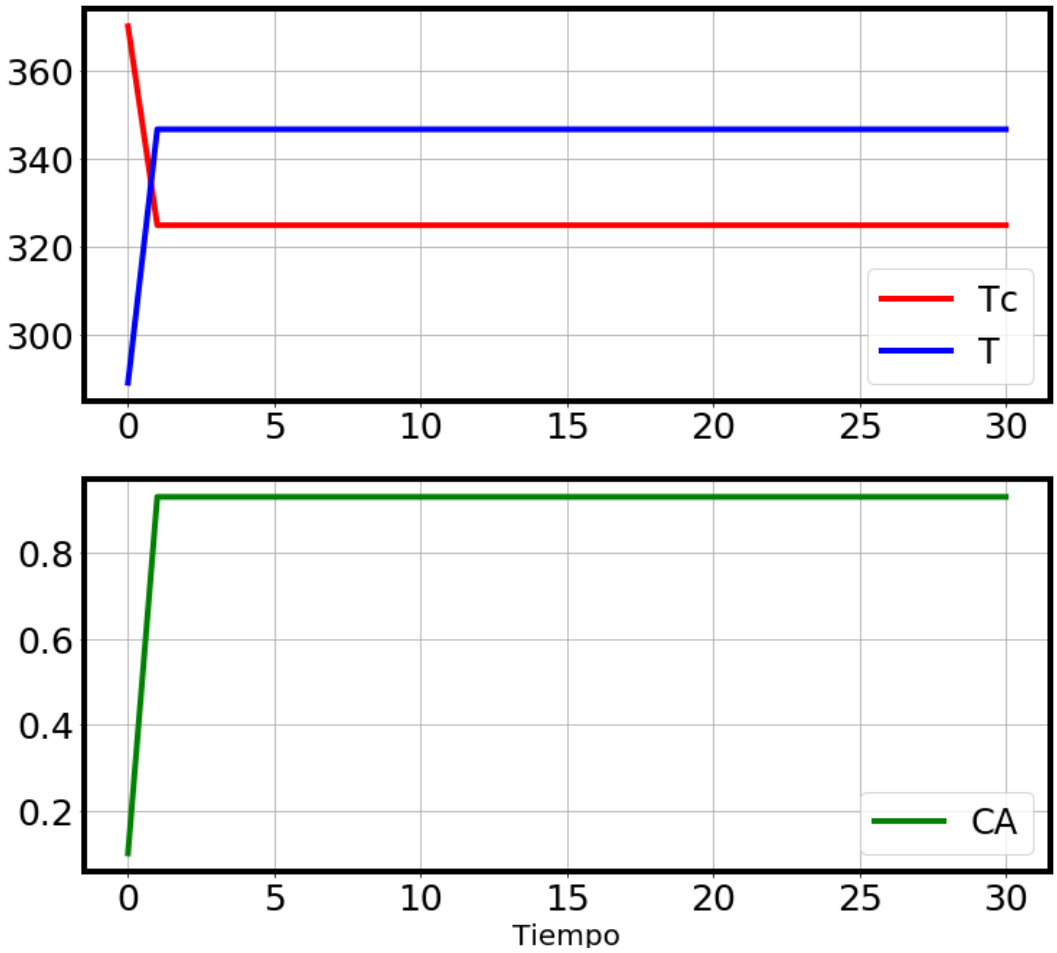


Figura 3.12: Resultados planta CSTR(Ejemplo 2)

donde

$$F_i = k_{vi} H_i \tag{3.12}$$

$$k_{Ai} = k_A exp\left(- \frac{E_A}{RT_i} \right) \tag{3.13}$$

$$k_{Bi} = k_B exp\left(- \frac{E_B}{RT_i} \right) \tag{3.14}$$

$$F_D = 0{,}01 F_R \tag{3.15}$$

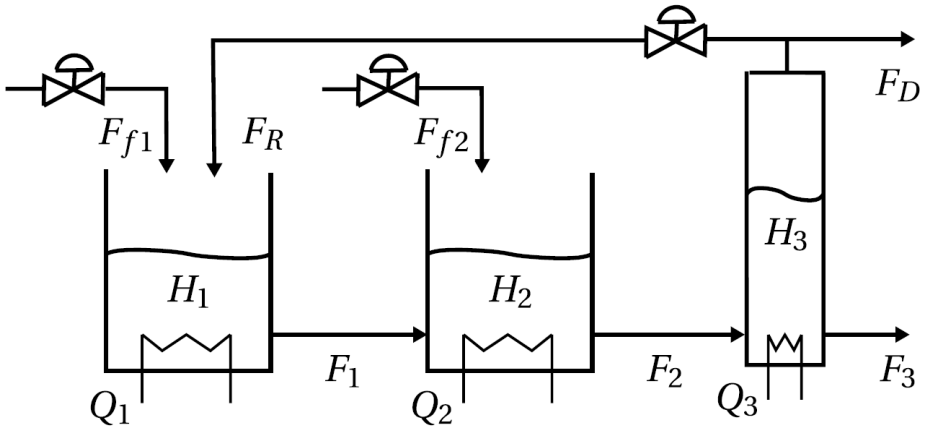


Figura 3.13: Esquema del reactor múltiple

$$x_{AR} = \frac{\alpha_A x_{A3}}{\bar{x}_3} \quad x_{BR} = \frac{\alpha_B x_{B3}}{\bar{x}_3} \tag{3.16}$$

$$\bar{x}_3 = \alpha_3 x_{A3} + \alpha_B x_{B3} + \alpha_C x_{C3} \quad x_{C3} = 1 - x_{A3} - x_{B3} \tag{3.17}$$

En la tabla siguiente se muestran los valores de los parámetros y constantes utilizados.

Símbolo	**Valor**	**Símbolo**	**Valor**
A_1	3	k_A	0.02
A_2	3	k_B	0.018
A_3	1	E_A/R	-100
ρ	0.15	E/RB	-150
C_p	25	ΔH_A	-40
k_{v1}	2.5	ΔH_B	-50
k_{v2}	2.5	α_A	3.5
k_{v3}	2.5	α_B	1.1
x_{A0}	1	α_C	0.5
T_0	313		

Cuadro 3.5: Parámetros planta reacror múltiple

Implementación en Pyomo

Descritas las ecuaciones dinámicas que definen el comportamiento de las variables del sistema, se muestra a continuación el código para la implementación del control en Pyomo.

Inclusión de librerías.

```
#Libreria para tratamiento de arrays
import numpy as np

#Librerias de Pyomo
from pyomo.environ import *
from pyomo.dae import *

#Libreria para funciones matematicas
import math
```

Definición de la función global para la resolución del problema.

```
def Mi_reactor_serie_primer(steps,hor):
```

Declaración de valores constantes y parámetros y puntos de referencia del sistema.

```
    #Definicion de parametros
    A1=3
    A2=3
    A3=1

    rho=0.15

    Cp=25


    kv1=2.5
    kv2=2.5
```

```
kv3=2.5


xA0=1
T0=313

kA=0.02
kB=0.018

EA_R=-100
EB_R=-150

D_HA=-40
D_HB=-50

alfa_A=3.5
alfa_B=1.1
alfa_C=0.5

#Punto de referencia
H1_ref=26.82
xA1_ref=0.92
xB1_ref=0.05
T1_ref=314.19

H2_ref=28.36
xA2_ref=0.92
xB2_ref=0.074
T2_ref=314.37

H3_ref=2.81
xA3_ref=0.76
xB3_ref=0.2
T3_ref=316.40
```

```
Ff1_ref=8.33
Q1_ref=10

Ff2_ref=0.5
Q2_ref=10

FR_ref=66.2
Q3_ref=10
```

En este problema se ha considerado que el punto inicial del sistema coincide con el punto de referencia, por lo que los valores de ambos son idénticos. A continuación se muestran los vectores para el almacenamiento de datos históricos y la adición del primer valor inicial.

```
#Almacenamiento de valores
H1_final=[]
xA1_final=[]
xB1_final=[]
T1_final=[]

H2_final=[]
xA2_final=[]
xB2_final=[]
T2_final=[]

H3_final=[]
xA3_final=[]
xB3_final=[]
T3_final=[]

#Inclusion punto inicial
H1_final.append(value(H1_init))
xA1_final.append(value(xA1_init))
xB1_final.append(value(xB1_init))
```

```
    T1_final.append(value(T1_init))

    H2_final.append(value(H2_init))
    xA2_final.append(value(xA2_init))
    xB2_final.append(value(xB2_init))
    T2_final.append(value(T2_init))

    H3_final.append(value(H3_init))
    xA3_final.append(value(xA3_init))
    xB3_final.append(value(xB3_init))
    T3_final.append(value(T3_init))
```

Implementación del bucle.

```
    #Bucle de ejecucion
    while (steps>0):
```

Creación del modelo en Pyomo y declaración de variables.

```
#Creacion de modelo
        m=ConcreteModel()
        m.t = ContinuousSet(bounds=(0,hor))

        #Declaracion de variables
        m.H1=Var(m.t,bounds=(2,40))
        m.xA1=Var(m.t,bounds=(0,1))
        m.xB1=Var(m.t,bounds=(0,1))
        m.T1=Var(m.t,bounds=(270,350))

        m.H2=Var(m.t,bounds=(2,40))
        m.xA2=Var(m.t,bounds=(0,1))
        m.xB2=Var(m.t,bounds=(0,1))
```

```
m.T2=Var(m.t,bounds=(270,350))

m.H3=Var(m.t,bounds=(2,40))
m.xA3=Var(m.t,bounds=(0,1))
m.xB3=Var(m.t,bounds=(0,1))
m.T3=Var(m.t,bounds=(270,350))

m.Ff1=Var(m.t,bounds=(0,10))
m.Q1=Var(m.t,bounds=(0,50))

m.Ff2=Var(m.t,bounds=(0,10))
m.Q2=Var(m.t,bounds=(0,50))

m.FR=Var(m.t,bounds=(0,75))
m.Q3=Var(m.t,bounds=(0,50))

#Declaracion de variables derivativas
m.H1_dot=DerivativeVar(m.H1,wrt=m.t)
m.xA1_dot=DerivativeVar(m.xA1,wrt=m.t)
m.xB1_dot=DerivativeVar(m.xB1,wrt=m.t)
m.T1_dot=DerivativeVar(m.T1,wrt=m.t)

m.H2_dot=DerivativeVar(m.H2,wrt=m.t)
m.xA2_dot=DerivativeVar(m.xA2,wrt=m.t)
m.xB2_dot=DerivativeVar(m.xB2,wrt=m.t)
m.T2_dot=DerivativeVar(m.T2,wrt=m.t)

m.H3_dot=DerivativeVar(m.H3,wrt=m.t)
m.xA3_dot=DerivativeVar(m.xA3,wrt=m.t)
m.xB3_dot=DerivativeVar(m.xB3,wrt=m.t)
m.T3_dot=DerivativeVar(m.T3,wrt=m.t)
```

Definición de las restricciones dinámicas que restringen el comportamiento del sistema.

```
#Restricciones dinamicas
def d_H1(m,l):
        if l==0:
            return m.H1[0]==H1_init
        else:
            return m.H1_dot[l]==(1/(rho*A1))
            *(m.Ff1[l]+m.FR[l]-kv1*m.H1[l])
m.H1con=Constraint(m.t,rule=d_H1)


def d_xA1(m,l):
        if l==0:
            return m.xA1[0]==xA1_init
        else:
            return m.xA1_dot[l]==(1/(rho*A1*m.H1[l]))
            *(m.Ff1[l]*xA0+m.FR[l]*(alfa_A*m.xA3[l])
            /(alfa_A*m.xA3[l]+alfa_B*m.xB3[l]
            +alfa_C*(1-m.xA3[l]-m.xB3[l]))
            -kv1*m.H1[l]*m.xA1[l])
            -kA*exp(-EA_R/m.T1[l])*m.xA1[l]
m.xA1con=Constraint(m.t,rule=d_xA1)


def d_xB1(m,l):
        if l==0:
            return m.xB1[0]==xB1_init
        else:
            return m.xB1_dot[l]==(1/(rho*A1*m.H1[l]))
            *(m.FR[l]*(alfa_B*m.xB3[l])
            /(alfa_A*m.xA3[l]+alfa_B*m.xB3[l]
            +alfa_C*(1-m.xA3[l]-m.xB3[l]))
            -kv1*m.H1[l]*m.xB1[l])
            +kA*exp(-EA_R/m.T1[l])
            *m.xA1[l]-kB*exp(-EB_R/m.T1[l])*m.xB1[l]
m.xB1con=Constraint(m.t,rule=d_xB1)
```

```
def d_T1(m,l):
        if l==0:
            return m.T1[0]==T1_init
        else:
            return m.T1_dot[l]==(1/(rho*A1*m.H1[l]))
            *(m.Ff1[l]*T0+m.FR[l]*m.T3[l]-kv1
            *m.H1[l]*m.T1[l])-(1/Cp)
            *(kA*exp(-EA_R/m.T1[l])*m.xA1[l]
            *D_HA+kB*exp(-EB_R/m.T1[l])
            *m.xB1[l]*D_HB)
            +(m.Q1[l]/(rho*A1*Cp*m.H1[l]))
m.T1con=Constraint(m.t,rule=d_T1)


def d_H2(m,l):
        if l==0:
            return m.H2[0]==H2_init
        else:

            return m.H2_dot[l]==(1/(rho*A2))
            *(m.Ff2[l]+kv1*m.H1[l]-kv2*m.H2[l])
m.H2con=Constraint(m.t,rule=d_H2)


def d_xA2(m,l):

        if l==0:
            return m.xA2[0]==xA2_init
        else:
            return m.xA2_dot[l]==(1/(rho*A2*m.H2[l]))
            *(m.Ff2[l]*xA0+kv1*m.H1[l]
            *m.xA1[l]-kv2*m.H2[l]
            *m.xA2[l])-kA*exp(-EA_R/m.T2[l])*m.xA2[l]
m.xA2con=Constraint(m.t,rule=d_xA2)
```

```
def d_xB2(m,l):
        if l==0:
            return m.xB2[0]==xB2_init
        else:
            return m.xB2_dot[l]==(1/(rho*A2*m.H2[l]))
            *(kv1*m.H1[l]*m.xB1[l]-kv2
            *m.H2[l]*m.xB2[l])+kA*exp(-EA_R/m.T2[l])
            *m.xA2[l]-kB*exp(-EB_R/m.T2[l])*m.xB2[l]
m.xB2con=Constraint(m.t,rule=d_xB2)


def d_T2(m,l):

        if l==0:
            return m.T2[0]==T2_init
        else:
            return m.T2_dot[l]==(1/(rho*A2*m.H2[l]))
            *(m.Ff2[l]*T0+kv1*m.H1[l]
            *m.T1[l]-kv2*m.H2[l]*m.T2[l])-(1/Cp)
            *(kA*exp(-EA_R/m.T2[l])*m.xA2[l]
            *D_HA+kB*exp(-EB_R/m.T2[l])
            *m.xB2[l]*D_HB)
            +(m.Q2[l]/(rho*A2*Cp*m.H2[l]))
m.T2con=Constraint(m.t,rule=d_T2)


def d_H3(m,l):

        if l==0:
            return m.H3[0]==H3_init
        else:

            return m.H3_dot[l]==(1/(rho*A3))
```

```
                *(kv2*m.H2[l]-0.01
                *m.FR[l]-m.FR[l]
                -kv3*m.H3[l])
m.H3con=Constraint(m.t,rule=d_H3)


def d_xA3(m,l):

        if l==0:
            return m.xA3[0]==xA3_init
        else:
            return m.xA3_dot[l]==(1/(rho*A3*m.H3[l]))
            *(kv2*m.H2[l]*m.xA2[l]
            -(0.01*m.FR[l]+m.FR[l])
            *(alfa_A*m.xA3[l])/(alfa_A*m.xA3[l]
            +alfa_B*m.xB3[l]+alfa_C
            *(1-m.xA3[l]-m.xB3[l]))-kv3*m.H3[l]
            *(alfa_A*m.xA3[l])/(alfa_A*m.xA3[l]
            +alfa_B*m.xB3[l]
            +alfa_C*(1-m.xA3[l]-m.xB3[l])))
m.xA3con=Constraint(m.t,rule=d_xA3)


def d_xB3(m,l):
        if l==0:
            return m.xB3[0]==xB3_init
        else:
            return m.xB3_dot[l]==(1/(rho*A3*m.H3[l]))
            *(kv2*m.H2[l]*m.xB2[l]
            -(0.01*m.FR[l]+m.FR[l])
            *(alfa_B*m.xB3[l])
            /(alfa_A*m.xA3[l]+alfa_B*m.xB3[l]
            +alfa_C*(1-m.xA3[l]-m.xB3[l]))
            -kv3*m.H3[l]*(alfa_B*m.xB3[l])
            /(alfa_A*m.xA3[l]
```

```
                +alfa_B*m.xB3[l]+alfa_C
                *(1-m.xA3[l]-m.xB3[l])))
m.xB3con=Constraint(m.t,rule=d_xB3)


def d_T3(m,l):
        if l==0:
            return m.T3[0]==T3_init
        else:
            return m.T3_dot[l]==(1/(rho*A3*m.H3[l]))
            *(kv2*m.H2[l]*m.T2[l]
            -(0.01*m.FR[l]+m.FR[l])
            *m.T3[l]-kv3*m.H3[l]*m.T3[l])
            +(m.Q3[l]/(rho*A3*Cp*m.H3[l]))
m.T3con=Constraint(m.t,rule=d_T3)
```

Definición de la función objetivo, en la cual además se ha incluido coste terminal para garantizar la estabilidad.

```
#Funcion objetivo
def _obj(m):


    return sum(Q1*(m.xA1[l]-xA1_ref)**2
    +Q2*(m.H1[l]-H1_ref)**2
    +Q3*(m.H2[l]-H2_ref)**2
    +Q4*(m.H3[l]-H3_ref)**2
    +Q5**(m.T1[l]-T1_ref)**2
    +Q6**(m.T2[l]-T2_ref)**2
    +Q7*(m.T3[l]-T3_ref)**2
    +R1*(m.Ff1[l]-Ff1_ref)**2
    +R2*(m.Q1[l]-Q1_ref)**2
    +R3*(m.Ff2[l]-Ff2_ref)**2
    +R4*(m.Q2[l]-Q2_ref)
```

```
        +R5*(m.FR[l]-FR_ref)**2
        +R6*(m.Q3[l]-Q3_ref)**2) for l in m.t)
        +P1*(m.xA1[hor]-xA1_ref)**2
        +P2*(m.H1[hor]-H1_ref)**2
        +P3*(m.H2[hor]-H2_ref)**2
        +P4*(m.H3[hor]-H3_ref)**2
        +P5**(m.T1[hor]-T1_ref)**2
        +P6*(m.T2[hor]-T2_ref)**2
m.obj=Objective(rule=_obj,sense=minimize)
```

Discretización del modelo y aplicación del solver.

```
#Discretizacion
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(m,nfe=hor,ncp=col
,scheme='LAGRANGE-LEGENDRE')

#Seleccion de solver y aplicacion
solver=SolverFactory('Ipopt')
results = solver.solve(m,tee=True)
```

Almacenamiento del primer valor inicial de la trayectoria predicha, actualización del valor inicial y disminución del número de iteraciones.

```
#Almacenamiento dle primer valor

H1_final.append(value(m.H1[col]))
xA1_final.append(value(m.xA1[col]))
xB1_final.append(value(m.xB1[col]))
T1_final.append(value(m.T1[col]))

H2_final.append(value(m.H2[col]))
```

```
        xA2_final.append(value(m.xA2[col]))
        xB2_final.append(value(m.xB2[col]))
        T2_final.append(value(m.T2[col]))

        H3_final.append(value(m.H3[col]))
        xA3_final.append(value(m.xA3[col]))
        xB3_final.append(value(m.xB3[col]))
        T3_final.append(value(m.T3[col]))

        #Actualizacion del valor inicial
        H1_init=m.H1[1].value
        H2_init=m.H2[1].value
        H3_init=m.H3[1].value

        xA1_init=m.xA1[1].value
        xA2_init=m.xA2[1].value
        xA3_init=m.xA3[1].value

        xB1_init=m.xB1[1].value
        xB2_init=m.xB2[1].value
        xB3_init=m.xB3[1].value

        T1_init=m.T1[1].value
        T2_init=m.T2[1].value
        T3_init=m.T3[1].value

        #Disminucion de iteraciones
        steps=steps-1
```

Se muestran a continuación los resultados obtenidos para las concentraciones en cada uno de los tanques.

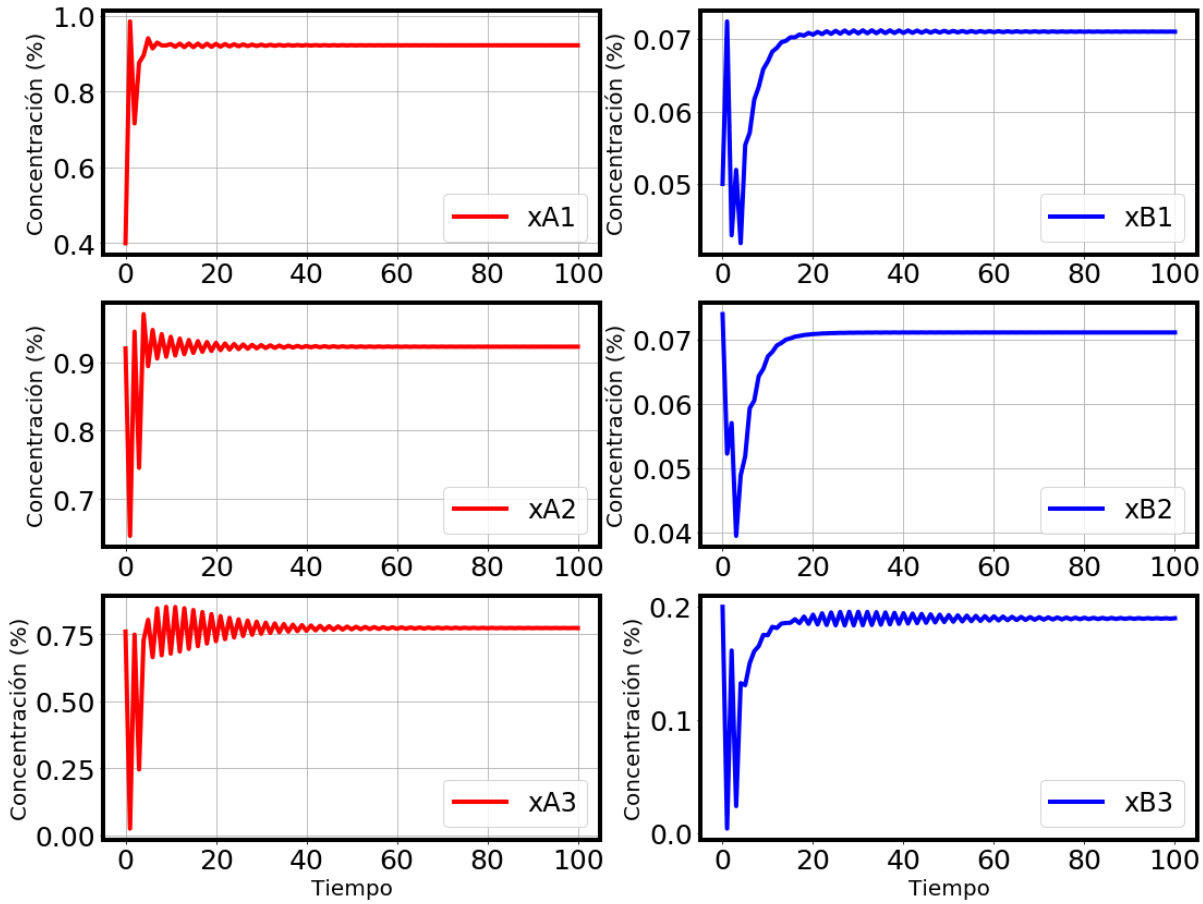


Figura 3.14: Resultados planta reactor múltiple

Parte III

Parte III: Para-digmas de futuro

Introducción al Control Predictivo Económico

4.1 Concepto General de EMPC

En esta sección se realiza una breve introducción al concepto de control predictivo económico basándose en las descripciones generales realizadas en [7], ya que se tiene pensado que las herramientas presentadas en el presente trabajo puedan ser utilizadas en futuros proyectos para la implementación de estrategias económicas de control predictivo.

Uno de los problemas en el ámbito industrial a la hora de llevar a cabo la implementación de una determinada estrategia de control predictivo (o cualquier otra metodología de control) es la elección del punto de operación óptimo en el cual debe de operar la planta con el objetivo de maximizar los beneficios económicos obtenidos. En los últimos años son numerosas las líneas de investigación en este ámbito que han surgido en la academia. En la mayoría de las industrias, llevar a cabo la operación de una planta por un largo período de tiempo manteniéndose un mismo punto de operación es prácticamente imposible, ya que existen numerosos condicionantes que hacen que la optimalidad de operación sea perdida.

Una de las alternativas a la hora de llevar el control de una determinada planta teniendo en cuenta que se han de alcanzar los máximos beneficios económicos posibles es el empleo de consignas solución de un determinado problema de control óptimo en línea. Es decir, las acciones de control sobre las entradas manipuladas son calculadas mediante la formulación y resolución de un determinado problema de optimización dinámica. En la actualidad, esto que hasta hace poco era impensable debido a la gran carga computacional que suponía la resolución del problema en un tiempo aceptable, es posible gracias a la facilidad de las plataformas y algoritmos matemáticos modernos de solventar complejos problemas de optimización dinámica.

Estos problemas de optimización calculan el punto de operación óptima de la planta. Son numerosos los aspectos que se han de tener en cuenta, como pueden ser, a modo de ejemplo, los valores de mercado, las reservas disponibles, la demanda futura prevista, el precio de materias primas y de la energía, etc. Para solventar este problema, tradicionalmente se ha utilizado, especialmente en la industria química, una estructura jerárquica como la mostrada en la siguiente figura, en la cual el punto de operación de la planta es suministrado al controlador procedente de una etapa previa en la cual se analizan los datos citados anteriormente.

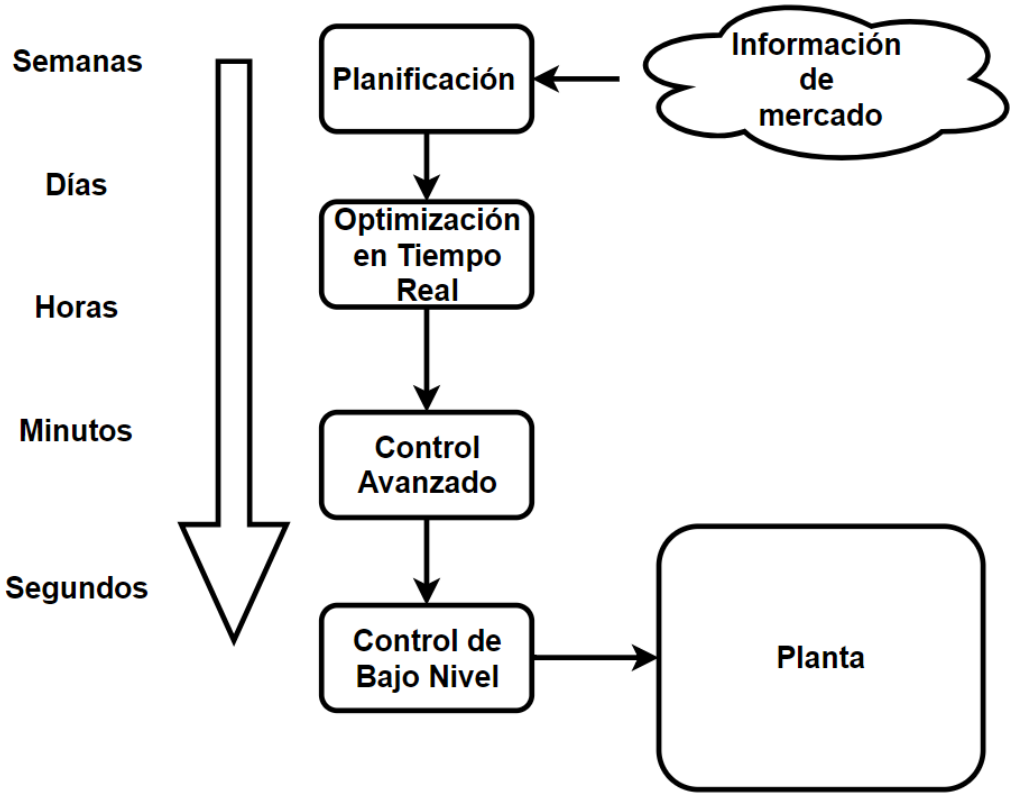


Figura 4.1: Jerarquía de planificación económica

En la figura anterior se puede ver cómo dicha estructura se encuentra jerarquizada mediante un determinada escala temporal, en la cual se refleja el tiempo de ejecución de cada una de las capas del diagrama.

En la parte superior, y teniendo el mayor tiempo de ejecución, se encuentra la etapa de planificación. En dicha etapa, se recogen los valores económicos de mercado y se

lleva a cabo una planificación de la producción que ha de tener la planta. Esta planificación se encuentra normalmente fuera del ámbito de la ingeniería de control, por cual no resulta de interés en el presente trabajo.

En la siguiente capa se encuentra la conocida como etapa de optimización en tiempo real (RTO), y es la capa en la cual se lleva a cabo la optimización del proceso que se ha de realizar. En dicha etapa, en la cual se tiene como entrada la planificaicón llevada a cabo en la capa anterior, se realiza un proceso de optimización que devuelve el estado estacionario o punto de operación al cual ha de trabajar la planta. Dicho punto de operación óptimo es pasado hasta la etapa conocida como de control avanzado. Son numerosas las alternativas de control avanzado presentes en la academia, entre las cuales de encuentra el control predictivo, ampliamente extendido y utilizado.

Por lo general, la salida de la etapa de control avanzado será la entrada de la etapa de control a bajo nivel, en la cual se encuentran los controladores de bajo nivel (como los PID) que interactúan directamente con los actuadores de la planta. En el pasado, esta etapa de bajo nivel era obligatoria, debido a que la etapa de control avanzado no era capaz de funcionar a la frecuencia a la cual requieren los actuadores de la planta. Sin embargo, es cada vez más frecuente encontrar actuadores en plantas controlados directamente mediante metodologías de control avanzado, debido al enorme avance que se ha producido tanto en plataformas de cómputo como en métodos de optimización matemática que aceleran enormemente dicho proceso.

Esta estructura ha ofrecido hasta la fecha buenos resultados, y su utilización se ha extendido a la mayor parte de las industrias productivas. Sin embargo, presenta la problemática de la gran diferencia de escala temporal entre capa y capa. Cada vez es más la necesidad de desarrollar estrategias de control que tengan en cuenta en tiempo real las informaciones de mercado, debido a los enormes beneficios económicos que esto supone. Ante esta necesidad es que surge el concepto de optimización económica dinámica. En concreto, existe la necesidad de llevar a cabo el desarrollo e implementación de algoritmos que permitan integrar lo máximo posible las capas de la estructura jerárquica.

Con el objetivo de llevar a cabo dicha integración de los procesos de optimización económica y los procesos de control así como mejorar la actuación del proceso conseguida mediante la variación dinámica con la información de mercado de los puntos de operación del sistema, es decir, sin forzar el sistema a trabajar en un determinado punto de operación estático e invariante, surge el concepto de control predictivo económico (EMPC). Esta metodología tiene como paradigma principal la incorporación de una función de coste general como parte de la función objetivo a optimizar por el controlador predictivo. Es decir, que tenga en cuenta no sólo información relativa al proceso llevado a cabo por la planta sino la información que antes era única y exclusivamente conocida en la etapa de optimización en tiempo real.

Siguiendo con los desarrollos llevados a cabo hasta el momento, supóngase un sistema dinámico cuya evolución temporal viene dada por la ecuación diferencial siguiente:

$$x^{+} = f(x,u,w) \tag{4.1}$$

donde x^{+} es el estado del sistema en el siguiente estado de muestreo (caso discreto) o instante temporal (caso continuo), x denota el conjunto de variables que conforman el estado del sistema, u es el conjunto de entradas manipulables o variables de control, y w hace referencia a las posibles perturbaciones que puedan afectar al sistema. Supóngase que el conjunto admisible que acota las variables de control $u \in U$ es un conjunto compacto, que el vector de perturbaciones se encuentra restringido a tomar valores dentro de un conjunto $w \in W$ tal que $\mid w \mid \leqslant \theta$ donde $\theta > 0$ acota la norma del vector de perturbaciones, y que la función $f \times U \times W \to X$ es localmente Lipschitz en $X \times U \times W$. El sistema cuenta además con una determinada función $l_e : X \times U \to R$ la cual es continua a lo largo de su dominio, y que refleja el coste económico del proceso. Dicha función será utilizada como coste de etapa del control predictivo y será conocida de ahora en adelante como coste de etapa económico.

Además, el sistema puede encontrarse sujeto a una serie de restricciones, como las que ya han sido extensamente tratadas en secciones anteriores, y que serán resumidas

como sigue:

$$(x,u) \in Z \tag{4.2}$$

donde se asume que Z es un conjunto compacto. Con todo ello, es posible encontrar una serie de valores óptimos del estado y de la señal de control como:

$$(x_s^*,u_s^*) = arg \min_{x_s,u_s \in Z} l_e(x_s,u_s) \tag{4.3}$$

Como puede observarse, el control predictivo económico no es más que un control predictivo el cual utiliza como función objetivo en el proceso de optimización términos que hacen referencia al coste de producción. Con todo lo dicho anteriormente, el problema general de optimización en un problema de control económico puede ser formulado (en su versión en tiempo continuo) como:

$$\begin{aligned} &\min_{u \in U} \int_{t_{init}}^{t+\Delta t} l_e(x(t),u(t))dt + V(x(t+\Delta t)) \\ &s.a \\ &\quad x^+(t) = f(x(t),u(t)) \\ &\quad (x(t),u(t)) \in Z \; \forall t \in (t_{init},t+\Delta t) \\ &\quad x(t+\Delta t) \in X_f \end{aligned} \tag{4.4}$$

o en su versión discreta como:

$$\begin{aligned} &\min_{u \in U} \sum_{j=k}^{N-1} l_e(x(j),u(j)) + V(x(N)) \\ &s.a \\ &\quad x(j+1) = f(x(j),u(j)) \\ &(x(j),u(j)) \in Z, \; j = k,k+1,...,k+N-1 \\ &x(k+N) \in X_f \end{aligned} \tag{4.5}$$

donde X_f hace referencia a la región terminal del estado para garantizar estabilidad. Dentro del contexto del control predictivo económico, la actuación del sistema

suele hacer referencia a la actuación en lazo cerrado del mismo. Considerando un intervalo finito de operación de longitud t_f, la actuación media se define como:

$$\bar{J}_e = \frac{1}{t_f}\int_0^{tf} l_e(x(t),u(t))dt \tag{4.6}$$

Debido a multitud de razones, como la no convexidad del coste de etapa, las no linealidades del sistema, etc, puede ser óptimo operar la planta con respecto al coste económico de etapa de manera variante en el tiempo, es decir, la estrategia de control óptima puede que no sea de operación estática. Por esta razón, suele considerarse la actuación económica media de la planta bajo la política definida por el EMPC a lo largo de la trayectoria de operación. De esta forma se tiene la siguiente definición.

Definición 21.
Se dice que un sistema está siendo operado de manera óptima en estado estacionario con respecto al coste $l_e(x,u)$ si para cualquier solución que satisfaga $(x(t),u(t)) \in Z$ se tiene que:

$$Av(l_e(x,u)) \subseteq [l_e(x_s^*,u_s^*),\infty] \tag{4.7}$$

donde:

$$Av := \bar{v} \in R^n : \exists t_n \to \infty : \lim_{n\to\infty} \frac{\Sigma_{k=0}^{t_{n-1}}}{t_n} = \bar{v} \tag{4.8}$$

Además, si $Av(l_e(x,u)) \subseteq (l_e(x_s^*,u_s^*),\infty$ ó $\lim_{k\to\infty} \mid x(k) - x_s^* \mid = 0$ se dice que el sistema se encuentra operado en estado estacionario de manera económica subóptima.

Como se puede observar, el control predictivo económico no es más que una variante del control predictivo convencional don la función objetivo a ser optimizada contiene un término de coste económico en el cual se tienen en cuenta información de

mercado externa al propio sistema de control. Debido a esto, es posible aplicar a estos sistemas los principios de estabilidad estudiados en [7], así como los aspectos relativos a la robustez del sistema. Un análisis amplio y detallado de la aplicación de dichos conceptos al control predictivo económico se lleva a cabo en [7], donde se analizan las principales técnicas de estabilización de controladores predictivos económicos.

Se espera que el control predictivo económico sea ampliamente estudiado en los próximos años debido a la fuerte repercusión que puede tener, especialmente en grandes empresas productivas, altamente dependientes de los valores de mercado. Sin embargo, debido a que en la actualidad no se encuentra aún extendido, se ha incluido en esta sección para abrir la puerta a futuros estudios y trabajos en este ámbito.

Control Predictivo Basado en Datos

5.1 Introducción

Como su propio nombre indica, el control predictivo MPC tiene como fundamento la utilización de un determinado modelo del sistema que permita hacer predicciones sobre el mismo. De manera general, el control predictivo se puede resumir como la predicción de las futuras salidas del sistema y señales de control a partir de la optimización de una función de coste. Dicha función de coste dependerá por lo general tanto del valor del estado del sistema como de la señal de control, tanto en el instante inicial como en una determinada franja temporal futura. Por ello, será necesaria la utilización de un modelo que, a partir del valor actual del estado y de las señales de control, sea capaz de predecir el comportamiento futuro del sistema.

Hasta el momento, los modelos que han sido utilizados en el presente trabajo se han basado en la utilización de ecuaciones de primeros principios, de tal manera que el estado futuro de las variables era conocido mediante el análisis de sus derivadas. Este enfoque a pesar de dar muy buen resultado en sistemas poco complejos y fáciles de modelar, difícilmente puede ser utilizado en muchos de los sistemas reales, debido a la complejidad (o imposibilidad) de obtener modelos de primeros adecuadamente. Típicamente lo que se ha hecho para solventar este problema es la de llevar a cabo la búsqueda del modelo del sistema mediante la identificación del mismo fuera de línea. Esta metodología presenta la desventaja de ser estática, es decir, el modelo es invariante en el tiempo, y depende del ensayo que haya sido llevado a cabo bajo unas determinadas características.

Frente a este tipo de metodologías, han surgido en los últimos años numerosas alternativas basadas en el uso de datos y técnicas de *machine learning* para la obtención de modelos empíricos que no requieran del conocimiento de los principios fundamentales que rigen el comportamiento de la planta, sino sólo y exclusivamente de datos

históricos de valores de entradas de la planta y sus correspondientes salidas. Es decir, se es capaz de predecir el estado futuro de la planta sin la necesidad de conocer el modelo físico en sí de su comportamiento.

Existen multitud de técnicas basadas en datos para poder llevar a cabo tal labor. Por ejemplo, la utilización de modelos de regresión, especialmente modelos de regresión gaussiana y basados en métodos Kernel tal y como los utilizados en [9]. Como alternativa a este tipo de modelos basados en regresión, han surgido numerosas técnicas basadas en *deep learning* y especialmente en la utilización de perceptrones multicapa también conocidos como redes neuronales, los cuales se basan en el entrenamiento de un sistema que asemeja el comportamiento del sistema al de neuronas biológicas, tal y como se hace en [10].

En el presente trabajo, se presentará además como alternativa a las metodologías anteriores, una estrategia de control predictivo basado en datos mediante la utilización de la técnica basada en interpolación de Lipschitz conocida como inferencia kinky, de manera similar al desarrollo llevado a cabo en [11]. El desarrollo seguido en el presente capítulo será el siguiente: primeramente se comenzará llevando a cabo un breve resumen de las técnicas basadas en procesos gaussianos y redes neuronales para la estimación. Seguidamente, se presentará la conocida como inferencia kinky y su aplicación al control MPC. Por último se mostrará un ejemplo en el cual se llevarán a cabo una serie de ensayos sobre un determinado sistema para la obtención de datos y seguidamente se implementará un controlador predictivo que hará uso de esos datos en lenguaje de programación Python, tal y como ha sido utilizado en los ejemplos de controladores realizados previamente en el presente trabajo. A partir de aquí se verán diversas formas de llevar a cabo modelos de predicción, aplicándose posteriormente uno de ellos al control predictivo.

5.2 Predicción mediante Procesos Gaussianos

5.2.1 Regresión Bayesiana

Una de las formas más comunes de llevar a cabo la predicción de la salida de un sistema en función de la entrada del mismo son las conocidas como métodos de re-

gresión. Cualquier método basado en regresión tiene como objetivo el encontrar un determinado regresor, así como un vector paramétrico asociado al mismo tales que, conociendo la entrada de sistema, se pueda predecir el valor de su salida. Por tanto, el problema básico a resolver durante un proceso de regresión es el de encontrar el mejor regresor y su vector paramétrico asociado, a partir de la información procedente de parejas de datos históricas de entrada y salida. El proceso de regresión puede a su vez dividirse en dos etapas. La primera de ellas es la de elegir cuál será el regresor más adecuado a utilizar. Este problema es de gran complejidad, debido a que son infinitas las posibles alternativas, y a que en principio no se tiene ninguna información de los datos a analizar. La segunda es, tras la selección del resgresor a ser utilizado y del conjunto de datos de entrada y salida ya conocidos, llevar a cabo el cálculo del vector de parámetros asociado al mismo.Considérese por ejemplo un caso en el cual los datos se encuentran distribuidos en el espacio bidimensional de manera lineal, podría tenerse que la salida del sistema vendría dada por la expresión:

$$y = w_0 + w_1 x \tag{5.1}$$

donde $y \in \mathbb{R}$ es la salida del sistema, $x \in \mathbb{R}$ la entrada del mismo y en este caso regresor del sistema, y $[w_0 \;\; w_1]$ el vector paramétrico del mismo. De igual manera, en el caso en el que los datos se encuentren distribuidos de manera cuadrática en el espacio bidimensional, se tendría que:

$$y = w_0 + w_1 x + w_2 x^2 \tag{5.2}$$

donde en este caso el vector paramétrico del sistema, a partir de ahora θ, sería expresado como $\theta = [w_0 \;\; w_1 \;\; w_2]$ y el regresor del sistema, de ahora en adelante γ, sería $\gamma = [x \;\; x^2]$. Por lo general, en cualquier proceso de regresión se tiene como objetivo encontrar una función de la forma:

$$y = f(\gamma \mid \theta) \tag{5.3}$$

En el caso de regresión convencional el cálculo del regresor viene dado sólo y exclusivamente por la experiencia del diseñador, mientras que el vector paramétrico puede ser encontrado mediante cálculos analíticos y algebraicos dependientes del conjunto de datos, como puede ser la utilización de mínimos cuadrados.

Otro enfoque distinto a la hora de llevar a cabo el proceso de regresión es el enfoque probabilístico, que hace uso del teorema de Bayes, dando lugar así a la conocida como regresión bayesiana. En dicha regresión bayesiana la selección del mejor regresor se realiza nuevamente basándose en la experiencia del diseñador. Sin embargo, a la hora de llevar a cabo el cálculo del vector paramétrico se considera dicho vector como una variable aleatoria la cual sigue una distribución conocida a priori. Es decir, este tipo de regresión se utiliza cuando ya se tiene en poder cierto conocimiento acerca del parámetro a estimar. De esta forma, se intentará encontrar una expresión que relacione la información a priori con la probabilidad de que el vector paramétrico tenga un determinado valor dado un conjunto de datos conocidos. Aplicando el teorema de Bayes se tiene:

$$p(\theta \mid X) = \frac{p(\theta)p(X \mid \theta)}{p(X)} \tag{5.4}$$

donde X es el conjunto de datos, $p(\theta)$ es la probabilidad a priori del parámetro θ, es decir, sin ser conocedores de los datos,$p(X \mid \theta)$ es la probabilidad de que se dé el conjunto de datos X dado un determinado vector paramétrico θ, y $p(X)$ es el normalizador para asegurar que $p(\theta \mid X)$ se integra a la unidad. Este término es conocido como probabilidad a posteriori, ya que nos informa de cuán probable es que se dé un determinado valor del parámetro siendo conocido un conjunto de datos. La regresión bayesiana tiene como objetivo generar la distribución a posteriori dado una distribución a priori y un conjunto de datos.

A modo de ejemplo, supóngase que se tiene un determinado conjunto de datos pasados X, los cuales llevan asociados una distribución de parámetros desconocidos θ. Supóngase además que se tiene un nuevo dato de entrada cuya probabilidad desea ser calculada. El primer paso es el de determinar una probabilidad a priori $p(\theta)$ y utilizar el conjunto de datos X y el nuevo dato de entrada como:

$$p(x, X, \theta) = p(\theta)p(X \mid \theta)p(x \mid \theta) \tag{5.5}$$

Esta expresión, conocida como probabilidad de unión, se puede utilizar para estimar la probabilidad de un nuevo dato de entrada dado un conjunto de datos históricos

como:

$$p(x \mid X) = \frac{\int_\theta p(x, X, \theta) d\theta}{p(X)} = \frac{\int_\theta p(\theta) p(X \mid \theta) p(x \mid \theta) d\theta}{p(X)} = \int_\theta p(\theta \mid X) p(x \mid \theta) d\theta \quad (5.6)$$

Supóngase ahora que cada una de las instancias del problema anterior es una variable multimodal de K estados distintos de tal forma que $x_i^t = 1$ si la instancia t se encuentra en el estado i y $x_j^t = 0 \ \forall j \neq i$. De esta forma, los parámetros son tratados como probabilidades de estados, $q = [q_1, q_2, ..., q_k]$ cumpliéndose que $\sum_i q_i = 1$. La probabilidad de cada muestra puede ser escrita como:

$$p(X \mid q) = \prod_{t=1}^{N} \prod_{i=1}^{K} q_i^{x_i^t} \quad (5.7)$$

Una de las distribuciones más utilizadas como distribuciones a priori es la distribución de Dirichlet, la cual puede ser expresada como:

$$Dir(q \mid \alpha) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1) ... \Gamma(\alpha_K)} \prod_{i=1}^{K} q_i^{\alpha_i - 1} \quad (5.8)$$

donde $\alpha = [\alpha_1, ..., \alpha_K]^T$ y $\alpha_0 = \sum_i \alpha_i \alpha_i$ son los parámetros de la distribución a priori y los cuales reciben el nombre de hiperparámetros y donde $\Gamma(x)$ es la función de distribución Gamma definida como:

$$\Gamma(x) = \int_0^\infty u^{x-1} e^{-1} du \quad (5.9)$$

Con ello, dada la probabiliad a priori del problema, se puede derivar la probabilidad a posteriori como:

$$p(q \mid X) \propto p(X \mid q) p(q \mid \alpha) \propto \prod_i q_i^{\alpha_i + N_i - 1} \quad (5.10)$$

donde $N_i = \sum_{t=1}^{N} x_i^t$. Se puede así ver como la probabilidad a posteriori tiene la misma forma que la probabilidad a priori, la cual recibe el nombre de probabilidad

conjugada. Combinando ambas se tiene que la probabilidad a posteriori puede ser expresada como:

$$p(q \mid X) = \frac{\Gamma(\alpha_0 + N)}{\Gamma(\alpha_1 + N_1 ... \Gamma(\alpha_K + N_K} \prod_{i=1}^{K} q_i^{\alpha_i + N_i - 1} = Dir(q \mid \alpha + n) \tag{5.11}$$

donde $n = [N_i, ..., N_K]$ y $\sum_i N_i = N$

5.2.2 Concepto de Kernel

En la sección anterior se ha visto cómo es posible a partir de los datos del conjunto y de cierta información a priori, llevar a cabo el cálculo del mejor vector paramétrico. Sin embargo todavía queda por resolver el problema de seleccionar un adecuado regresor que se ajuste bien al conjunto de datos. De esta forma, supóngase un problema en el que se tiene una serie de datos multivariables en el espacio tridimensional, donde cada muestra está formada por tres características x_1, x_2, x_3. A partir de aquí, son infinitas las posibilidades a la hora de elegir regresor. Se muestran a continuación una serie posibilidades a modo de ejemplo:

$$\begin{aligned}
\gamma &= [x_1 \;\; x_2 \;\; x_3] \\
\gamma &= [x_1 + x_2 \;\; x_2^2 \;\; x_1 x_3] \\
\gamma &= [sin(x_1) \;\; ln(x_1^{x_2}) \;\; tanh(x_3 - x_2)]
\end{aligned}$$

La pregunta es, ¿es posible determinar de entre las infinitas posibilidades cuál es la mejor elección del regresor? La respuesta es que sí, y para dar solución a este problema surgieron los métodos Kernel, los cuales se presentan a continuación.

La idea de Kernel se basa en el hecho de que es posible llevar a cabo el mapeo de un determinado conjunto de características pertenecientes a un conjunto de datos en un espacio de distinta dimensionalidad al espacio original. Esta técnica es muy común encontrarla en diversos métodos de *machine learning* en los cuales antes de llevar a cabo el estudio de un conjunto de datos estos son mapeados en espacios de distinta dimensionalidad. Por ejemplo, supóngase que un conjunto de datos en espacio $\mathbb{R}$ se

desea mapear en una espacio $\mathbb{R}^2$. Dicho mapeo $\varphi : \mathbb{R} \rightarrow \mathbb{R}^2$ podría ser expresado como:

$$\varphi(x) = (x, x^2) \tag{5.12}$$

De esta forma, dado un determinado conjunto X, es posible llevar a cabo la selección de una función φ tal que el conjunto de datos se mapee en un nuevo espacio F tal que $\varphi : X \rightarrow F$, el cual generalmente será un conjunto perteneciente a $\mathbb{R}^n$. La clave del éxito de este tipo de transformación es el de saber elegir una adecuada función φ para cada problema en concreto. Se denomina espacio de características al conjunto de todas y cada una de las funciones φ. El término kernel es usado en este contexto para describir productos internos en el espacio de características. De esta forma, dado una determinada función φ y un determinado dominio X perteneciente a un determinado espacio de Hilbert, se define Kernel como una determinada función $K(x_i, x_j) = \langle \varphi(x_i), \varphi(x_i') \rangle$.

En resumen, y sin entrar en conceptos matemáticos que se salen del ámbito del presente trabajo, se definirá la función kernel K como una función $K(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \rightarrow R$ definida positiva, tal que para un conjunto de n datos la matriz K, cada uno de sus elementos $K_{i,j} = K(x_i, x_j)$ es una matriz semidefinida positiva que cumple con las propiedades de una matriz de covarianza.

Se demuestra que si la función $K(,)$ satisface las propiedades anteriores, entonces existe un determinado regresor γ tal que:

$$K(x_i, x_j) = \gamma(x_i)^T \gamma(x_j) \tag{5.13}$$

Esto es evidente si primeramente se define γ y a partir de aquí se calcula $K(\cdot, \cdot)$. Sin embargo, es posible llevar a cabo el cálculo de $K(\cdot, \cdot)$ sin la necesidad de ser conocedores de γ. En conclusión, un kernel es una herramienta matemática que en cualquier problema que involucre el uso de un regresor evita la necesidad de tener que llevar a cabo la selección del mismo. Mientras que la selección de un regresor puede ser un problema complejo y que requiere de mucha experiencia y conocimiento de los datos, dadas las infinitas posibilidades, existen multitud de funciones kernel ya diseñadas que cumplen con la función deseada independientemente del problema que se desea

resolver.

De entre todas las funciones kernel que han sido desarrolladas, la más ampliamente extendida, y la cual se muestra a continuación a modo de ejemplo, es la conocida como función RBF (Radial Basis Function), la cual se basa en un regresor infinito dimensional, y que viene dada por la expresión:

$$K(\cdot,\cdot) = a \ \exp\left\{ -\frac{1}{b}\left\|x_i - x_j\right\|^2 \right\} \tag{5.14}$$

5.2.3 Definición de Proceso Gaussiano

Con lo expuesto anteriormente, se tienen todos los elementos para la definición del concepto de proceso gaussiano. Un proceso gaussiano se define como una regresión bayesiana kernelizada. Supóngase una determinada función $f(x) \in \mathbb{R}$ y un determinado dato de entrada $x \in \mathbb{R}^d$. Supóngase además una determinada función kernel entre dos puntos x_i y x_j, $K(x_i,x_j)$. entonces, se dice que $f(x)$ es un proceso gaussiano e $y(x)$ un proceso de ruido añadido si:

$$y \mid f \sim N(f,\sigma^2 I), \ \ f \sim N(0,K) \Leftrightarrow N(0,\sigma^2 I + K) \tag{5.15}$$

donde $y = (y_1,...,y_n)^T$ y K es de tamaño $n \times n$ con $K_{ij} = K(x_i,x_j)$. Es decir, se toma un determinado valor en $\mathbb{R}^d$, y se mapea dicho punto en $\mathbb{R}$, de tal forma que la función de mapeo $f(x)$ pertenece a una normal de media cero y varianza la función kernel. Se muestra a continuación un ejemplo de utilización de proceso gaussiano.

Imagínese que se tiene un conjunto de n observaciones de pares de entrada y salida de un sistema $D = (x_i,y_i)_{i=1}^N$ y se desea predecir el valor de un nuevo valor de entrada y_0 dado una entrada x_0. La distribución de unión del problema puede ser expresada como:

$$\begin{bmatrix} y_0 \\ y \end{bmatrix} = N\left(0,\sigma^2 I + \begin{bmatrix} x_0^T x_0 & (Xx_0)^T \\ Xx_0 & XX^T \end{bmatrix}\right) \tag{5.16}$$

Se desea entonces predecir y_0 dado el conjunto D y el nuevo dato de entrada x_0. La

probabilidad de que se dé un valor de salida y_0 dado un determinado conjunto D y un dato de entrada x_0 puede ser expresada como:

$$y_0 \mid D, x_0 \sim N(\mu_0, \sigma_0^2) \tag{5.17}$$

donde:

$$\begin{aligned} \mu_0 &= (Xx_0)^T (XX^T)^{-1} y \\ \sigma_0^2 &= \sigma^2 - x_0^T x_0 - (Xx_0)^T (XX^T)^{-1} (Xx_0) \end{aligned} \tag{5.18}$$

En este ejemplo simplemente se lleva a cabo un proceso de regresión bayesiana. Para transformar el problema anterior en un problema definido como proceso gaussiano, simplemente se han de reemplazar los productos presentes en la varianza de la distribución normal que define la probabilidad de que se dé la salida y_0.

Considárese una determinada función kernel $K(x, D_n) = [K(x, x_1), ..., K(x, x_n$ donde K_n es una matriz kernel de dimensión n x n. Entonces se tiene que:

$$y(x) \mid D_n \sim N(\mu(x), \Sigma(x)) \tag{5.19}$$

donde:

$$\begin{aligned} \mu(x) &= K(x, D_n) K_n^{-1} y \\ \Sigma(x) &= \sigma^2 + K(x, x) - K(x, D_n) K_n^{-1} K(x, D_n)^T \end{aligned} \tag{5.20}$$

De esta forma, se ve como es posible predecir el valor de la salida del sistema ante una determinada entrada mediante la utilización única y exclusivamente del valor de entrada, el conjunto de datos histórico, y una determinada función kernel seleccionada por el usuario, sin necesidad de conocer el valor de ningún regresor γ.

5.3 Predicción mediante Redes Neuronales

5.3.1 Concepto de Neurona Artificial

Una de las técnicas más utilizadas y que más desarrollo ha experimentado en los últimos años son las redes de neuronas artificiales. Las redes de neuronas artificiales son redes formadas por la unión de múltiples neuronas y mediante las cuales se intenta reproducir el comportamiento de una red neuronal biológica. Las neuronas son pues las unidades básicas de procesado de información. En la siguiente figura se muestra el modelo básico de una neurona artificial.

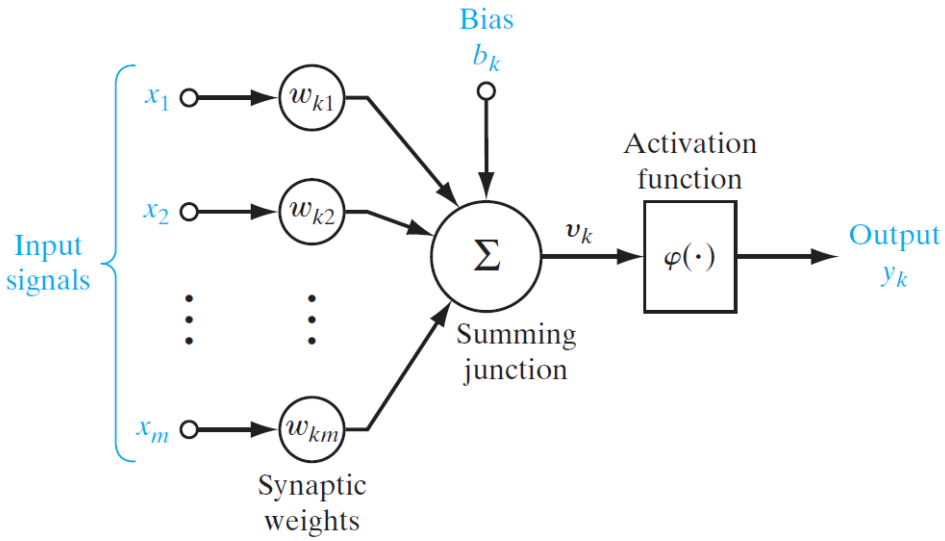


Figura 5.1: Modelo de una neurona artificial

En la neurona de la imagen anterior se pueden apreciar los siguientes elementos básicos:

- Un conjunto de sinapsis, o hilos de conexión, cada uno de los cuales está caracterizado por un peso. Cada entrada x_j conectada a la entrada de una determinada sinapsis j de la neurona k es multiplicada por el peso w_{kj}, donde k hace referencia a la neurona en concreto y j a la sinapsis de dicha neurona.
- Un elemento aditivo en el cual se realiza la suma de todas y cada una de las entradas de la neurona multiplicada por sus correspondientes pesos.
- Una función de activación que limita la amplitud de la salida de la neurona. La función de activación es también conocida como función límite, ya que limita

el rango de amplitudes permisibles de la señal de salida a una serie de valores finitos.

- En la neurona de la imagen también se incluye un determinado sesgo b, el cual tiene el efecto de incrementar o disminuir la entrada de la función de activación.

En términos matemáticos, la función que relaciona la salida de la neurona con el conjunto de señales de entrada puede ser escrita como:

$$\begin{aligned} u_k &= \sum_{j=1}^{m} w_{kj} x_j \\ y_k &= \phi(u_k + b_k) \end{aligned} \tag{5.21}$$

donde $x_1, ..., x_m$ son las señales de entrada, $w_{k1}, ..., w_{km}$los respectivos pesos sinápticos de cada una de las entradas de la neurona k, b_k es el sesgo de la neurona k, $\phi(\cdot)$ la función de activación, e y_k la salida de la neurona. La adición del sesgo tiene el efecto de aplicar una transformación afín a la salida u_k del modelo lineal, tal que:

$$v_k = u_k + b_k \tag{5.22}$$

En concreto, dependiendo de si el sesgo es positivo o negativo, la relación entre el campo local inducido, v_k de la neurona k y la combinación lineal u_k es modificada tal y como se muestra en la siguiente (5.2).

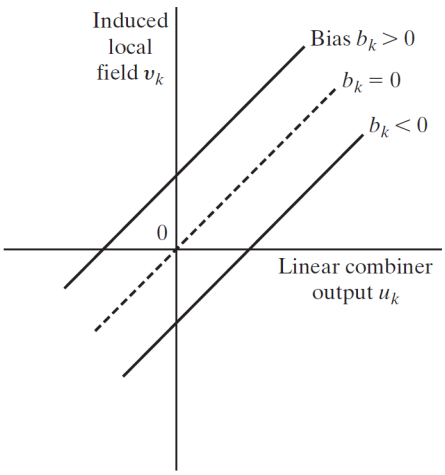


Figura 5.2: Transformación afín producida por el sesgo

Este sesgo es un parámetro externo de la neurona k. De manera equivalente, se pueden reformular las ecuaciones (5.21) y (5.22) como:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{5.23}$$

y

$$y_k = \phi(v_k) \tag{5.24}$$

En estas expresiones se ha añadido una nueva sinapsis, siendo su entrada:

$$x_0 = +1 \tag{5.25}$$

y su peso:

$$w_{k0} = b_k \tag{5.26}$$

De esta forma, se puede reformular el modelo de la neurona de tal forma que el sesgo es considerado una entrada más de la neurona. Esto tiene como consecuencia que la neurona tiene una nueva entrada fija de valor $+1$ y que se añade un nuevo peso sináptico de valor igual a b_k.

Tipos de función de activación

La función de activación $\phi(v)$ determina la salida de la neurona en términos del campo local inducido v. Existen dos funciones de activación principales.

Función Umbral

Este tipo de función se define como:

$$\phi(v) = \begin{cases} 1 \text{ si } v \geqslant 0 \\ 0 \text{ si } v < 0 \end{cases} \tag{5.27}$$

De esta forma, la salida de la neurona será:

$$y_k = \begin{cases} 1 \text{ si } v \geqslant 0 \\ 0 \text{ si } v < 0 \end{cases} \tag{5.28}$$

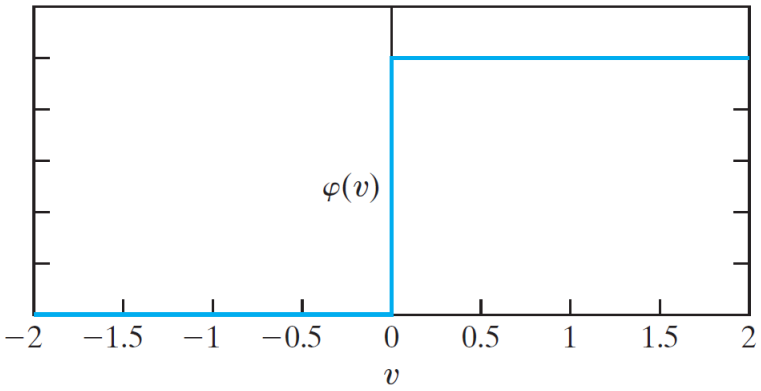


Figura 5.3: Representación de la función umbral

Función Sigmoide

Es con diferencia el tipo de función más utilizado en redes neuronales. Se define como una función estrictamente creciente que exhibe un comportamiento balanceado entre función lineal y no lineal. Un ejemplo de función sigmoide es la función logística definida como:

$$\phi(v) = \frac{1}{1 + \exp(-av)} \tag{5.29}$$

donde a es el parámetro de pendiente de la función sigmoide. Medinate la variación de dicho parámetro se consiguen funciones de mayor o menor pendiente.

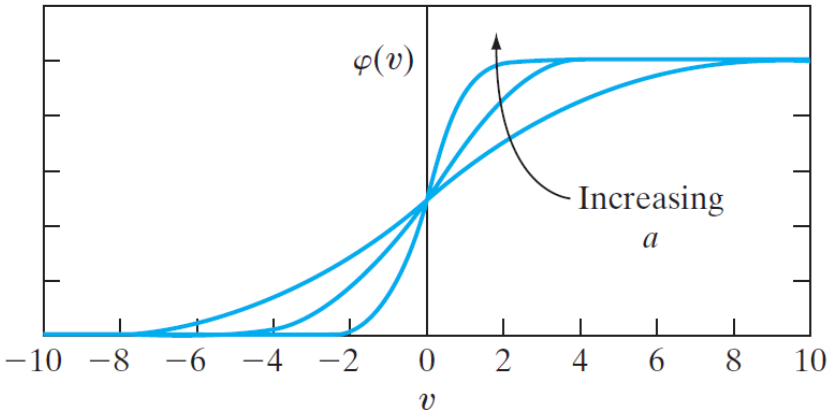


Figura 5.4: Representación de la función sigmoide

5.3.2 Arquitecturas de Redes Neuronales

El problema principal a la hora de llevar a cabo el diseño de cualquier tipo de red neuronal es el cálculo de los pesos sinápticos de cada una de las entradas. A dicho proceso de aprendizaje de los pesos se le conoce como entrenamiento de la red neuronal. Se denomina red neuronal a un determinado número de neuronas conectadas entre sí y que trabajan de manera cooperativa. Existen numerosas maneras en las cuales se puede organizar una red neuronal, lo cual está íntimamente ligado al algoritmo a utilizar durante el proceso de entrenamiento. En general, existen tres clases de arquitecturas de redes neuronales.

Redes Monocapa Feedforward

En una red neuronal, las neuronas se encuentran organizadas en forma de etapas o capas. En la arquitectura de red más sencilla, se tiene una única capa de nodos que reciben directamente las entradas del sistema, que conectan directamente con la salida o salidas del mismo; y siempre en un único sentido de avance de la señal desde las entradas hasta las salida. Es decir, la transmisión de señal es estrictamente de tipo feedforward. Este tipo de redes se denomina monocapa, ya que existe una única capa de nodos en la cual se lleva a cabo la computación de la señal, lo cual es la característica necesaria para que una determinada etapa se considere capa. En la imagen siguiente

se muestra la estructura de este tipo de redes.

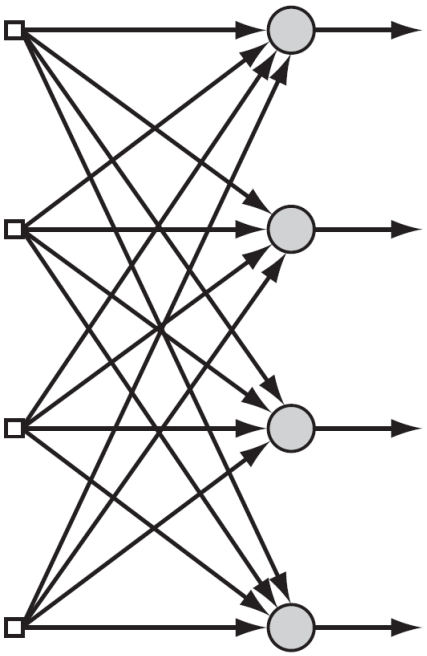


Figura 5.5: Estructura de Red Neuronal Monocapa Feedforward

Redes Multicapa Feedforward

Este segundo tipo de arquitectura de red se distingue del anterior debido a la presencia de más etapas de computación o capas entra el capa de entrada y la salida de la red neuronal, conocidas como capas ocultas de la red. El término ocultas hace referencia a que estas capas no son vistas ni por las entradas ni por las salidas de la red. La función de estas capas es la de intervenir de alguna manera entre la salida de la red y el conjunto de entradas. Mediante la adición de capas, la red neuronal es capaz de llevar a cabo cálculos de mayor orden.

Los nodos fuente de la capa de entrada proporcionan los respectivos elementos de patrones de activación, los cuales constituyen las señales de entrada aplicadas a las neuronas de la segunda capa. Las señales de salida de la segunda capa de la red son utilizadas como señales de entrada de la tercera capa, y así sucesivamente con el resto de capas hasta llegar a la capa de salida. Normalmente, cada una de las capas de la red tiene como señales de entrada aquellas procedentes únicamente de la capa anterior. El conjunto de señales de salida de la última capa o capa de salida constituyen la

respuesta general de una determinada red neuronal antes ante un determinado patrón de señales de activación. Se muestra a continuación la estructura de una red con 10 señales de entrada, de cuatro nodos, salida de dos nodos.

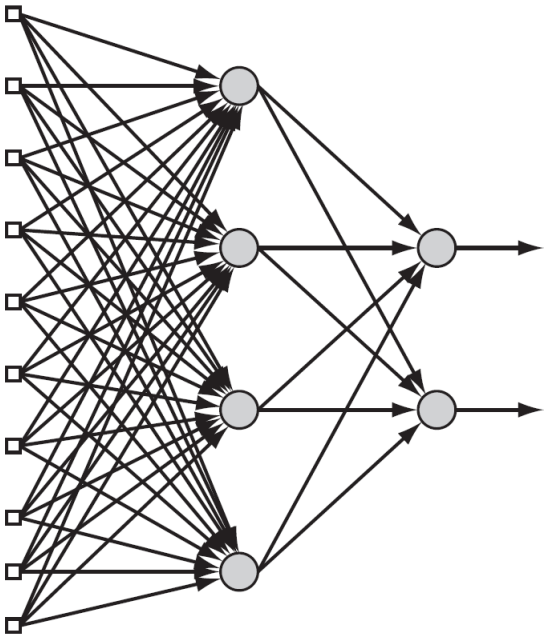


Figura 5.6: Estructura de Red Neuronal Multicapa Feedforward

Redes Recurrentes

La diferencia entre las redes feedforward y la redes recurrentes es que en las segundas existe al menos un lazo de realimentación. Por ejemplo, una red recurrente podría consistir en una capa simple de neuronas en la cual cada una de las neuronas realimenta su señal de salida a todas y cada una de las entradas de cada neurona.

La presencia de estos lazos de realimantación, ya se traten de redes monocapa o multicapa, tiene un profundo impacto en la capacidad de aprendizaje de la red y en la actuación de la misma. Además, los lazos de realimentación implican el uso de ramas particulares compuestas por elementos de retraso temporal (denotados como z^{-1}, en el dominio frecuencial) lo cual conlleva un comportamiento dinámico no lineal , asumiendo que la neurona contiene unidades no lineales. Se muestra a continuación un ejemplo de red neuronal recurrente.

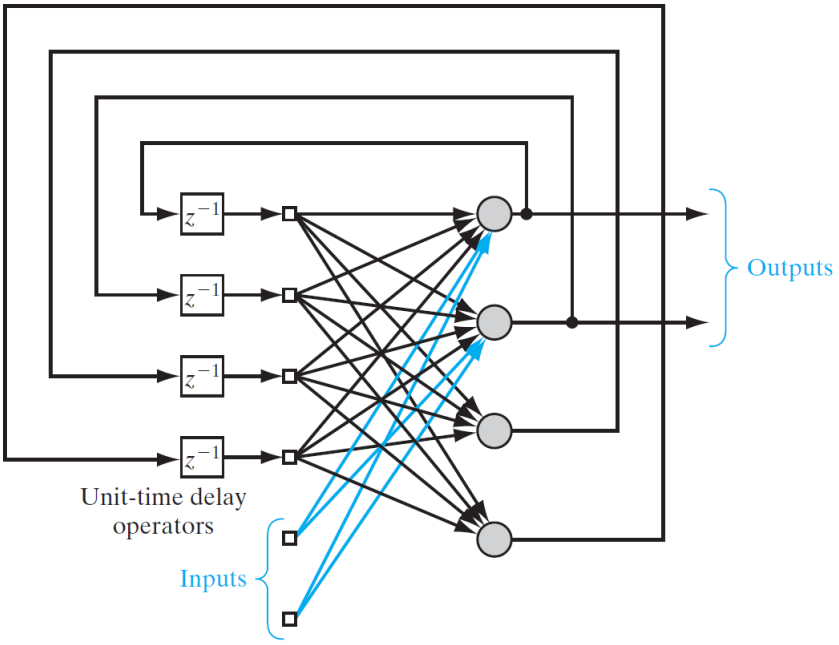


Figura 5.7: Estructura de Red Neuronal Recurrente

5.3.3 Aprendizaje de Redes Neuronales

Como previamente se ha comentado, uno de los factores claves en el diseño de cualquier red neuronal es el proceso de entrenamiento y aprendizaje de la misma, que no es nada más y nada menos que el cálculo de sus pesos sinápticos. En general, existen dos categorías de procesos de aprendizaje en redes neuronales, el aprendizaje supervisado o no supervisado.

Aprendizaje Supervisado

Al aprendizaje supervisado también se le conoce como aprendizaje con maestro. De manera conceptual, se puede pensar en este tipo de aprendizaje como un aprendizaje guiado en el cual se tiene cierta información y conocimiento acerca del entorno o problema a resolver, siendo dicho conocimiento materializado como una serie de datos de entrada y salida a modo de ejemplos. Supóngase ahora que tanto la red neuronal como el maestro se encuentran expuestos a un proceso de aprendizaje dado por un vector de entrenamiento de entrada.

El objetivo del maestro es el de proporcionar a la red neuronal cuál ha de ser la respuesta de la misma ante una determinada entrada. De hecho, la respuesta deseada representa la actuación óptima que la red puede llevar a cabo. Los parámetros de la red

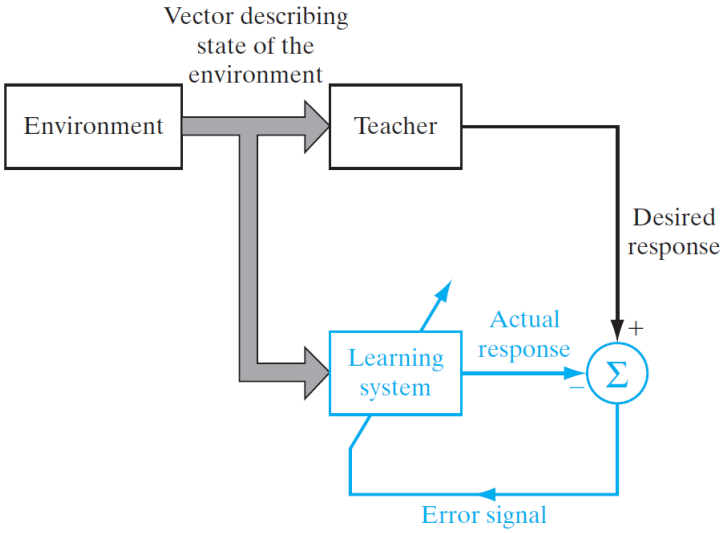


Figura 5.8: Proceso de Aprendizaje Supervisado

son así ajustado mediante combinación de la influencia del vector de entrenamiento y de la señal de error, tal y como se puede apreciar en la figura anterior. Así, se define la señal de error como la diferencia entre el comportamiento deseado provisto por el maestro y la respuesta actual de la red. El ajuste de los parámetros es llevado a cabo de manera iterativa paso a paso con el objetivo de que el comportamiento de la red sea idéntico al comportamiento del maestro después de un cierto tiempo de entrenamiento.

Este proceso de entrenamiento no deja de ser el tradicional aprendizaje mediante prueba y error. En la figura anterior se ve como el proceso de aprendizaje supervidado constituye un sistema con realimentación en lazo cerrado, sin embargo, el entorno se encuentra fuera de dicho lazo. Como medida del buen desempeño del sistema, se podría pensar en términos del error cuadrático medio, o la suma de errores cuadráticos a los largo del período de entrenamiento. Para que el sistema sea capaz de mejorar su comportamiento a lo largo del tiempo y por lo tanto aprender del maestro, el punto de operación de la red ha de moverse sucesivamente hacia un punto de mínimo error. Un proceso de aprendizaje supervisado ha de ser capaz de hacer esto mediante cierta información acerca del gradiente del error del comportamiento actual del sistema. El gradiente del error en cualquier instante es un vector que apunta en la dirección del mayor descenso. De hecho, en el caso de aprendizaje supervisado a través de ejemplos, el sistema puede utilizar la estimación instantánea del vector gradiente. La utilización de estas estimaciones conlleva el movimiento del punto de operación, que generalmen-

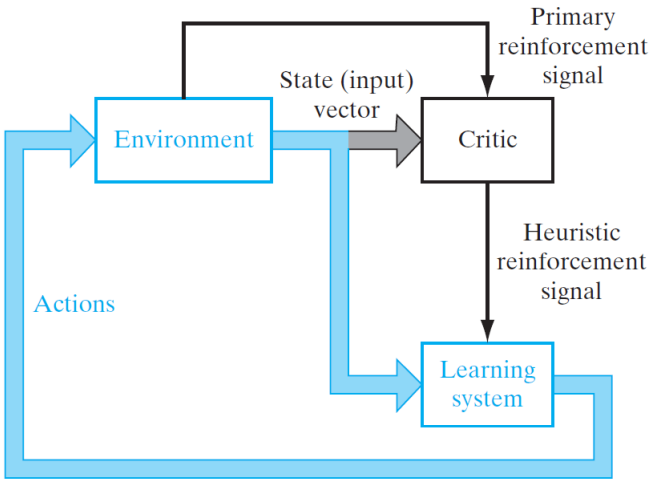


Figura 5.9: Proceso de Aprendizaje por refuerzo

te se mueve de manera aleatoria.

Aprendizaje No Supervisado

A diferencia del aprendizaje supervisado, en el cual había un maestro que conducía el proceso, en el paradigma de aprendizaje no supervisado no existe dicha figura del maestro. Esto quiere decir que no existe ejemplos previos de los cuales la red pueda aprender. Partiendo de aquí, se describe la principal categoría de aprendizaje no supervisado.

En el aprendizaje por refuerzo, el aprendizaje de un conjunto entrada-salida se lleva a cabo mediante la continua interacción con el entorno con el objetivo de minimizar un determinado índice de desempeño. En la figura siguiente se muestra el diagrama de una forma de aprendizaje por refuerzo en torno a un bloque crítico que convierte una señal de refuerzo primaria recibida del entorno en una señal de refuerzo de mayor calidad denominada señal de refuerzo heurística, ambas de carácter escalar.

El objetivo del aprendizaje por refuerzo es minimizar una función de coste definida como la esperanza del coste acumulativo de las acciones tomadas a lo largo de una secuencia de pasos en lugar de simplemente el coste inmediato. Puede ocurrir que ciertas acciones tomadas previamente en la secuencia de pasos temporales sean las mejores determinantes del comportamiento del sistema.

5.3.4 Aproximación de Funciones mediante Redes Neuronales

Existen multitud de aplicaciones en las cuales se puede hacer uso de redes de neuronas artificiales, como pueden ser problemas de reconocimiento de patrones, problemas de clasificación, problemas de control, etc. De entre todos ellos, y debido a su relación con la técnica que será aplicada en el presente trabajo, la aplicación que nos interesa es la de su uso como aproximación de funciones desconocidas.

Considérese un determinado sistema no lineal desconocido cuya relación entrada salida puede ser descrita como:

$$y = f(x) \tag{5.30}$$

donde el vector x es la entrada del sistema e y el vector de salida. La función $f(\cdot)$ es completamente desconocida. Para poder encontrar un modelo basado en redes neuronales que asemeje el comportamiento de la función, se dispone de un conjunto de datos obtenidos mediante ensayos del sistema:

$$Y = \{x_i, y_i\}_{i=1}^{N} \tag{5.31}$$

El objetivo es el de diseñar una red neuronal que aproxime el comportamiento de la función $f(\cdot)$ de tal forma que la función $F(\cdot)$ implementada mediante la red neuronal describa la relación entrada-salida del sistema lo más similar posible a la función real $f(\cdot)$. Se puede llevar a cabo la acotación del error cometido por la red como:

$$\|F(x) - f(x)\| < \epsilon \tag{5.32}$$

donde ϵ es un valor real positivo lo más pequeño posible. Dado que el tamaño del conjunto de datos de valor N es lo suficientemente grande y la red se encuentra equipada con un número adecuado de parámetros libres (pesos sinápticos), entonces el error de aproximación ϵ puede hacerse lo más pequeño posible para que el comportamiento de la red emule lo suficientemente bien el comportamiento real del sistema.

5.4 Predicción mediante Inferencia KINKY

5.4.1 Inferencia kinky

Vistas ya las principales técnicas utilizadas en al literatura para la resolución del problema de predecir la salida de una determinada función basándonos única y exclusivamente en la entrada de la misma y en una serie de datos históricos, se presenta a continuación una nueva técnica de *machine learning* para solventar este problema.

El método presentado se conoce como inferencia kinky, y tiene como objetivo llevar a cabo la inferencia de la salida de una función ante una determinada entrada y a partir de una serie de datos históricos de entradas y salidas. En esta sección, debido al interés práctico del presente trabajo y la utilización del lenguaje de programación Python para la resolución del problema de control predictivo, se presenta la técnica de manera resumida basándonos en el trabajo desarrollado en [14]. Par una mayor comprensión de la técnica y profundización en los fundamentos matemáticos de este método se remite al lector a [15] donde se presenta por primera vez y se desarrolla de manera extensiva.

El método conocido como inferencia kinky se asemeja a una clase de regresión no paramétrica. El método es utilizado para aprender un mapa $f : W \to Z$. Se asume que dicho mapa tiene continuidad de Hölder, con constante L_D y una constante exponencial $0 < \alpha \leqslant 1$, de tal forma que $\forall w_1, w_2 \in W$ se cumple que:

$$\|f(w_1) - f(w_2)\| \leqslant L_D \|w_1 - w_2\|^{\alpha} \tag{5.33}$$

Nótese que si $\alpha = 1$ se da un caso particular de continuidad de Hölder conocida como continuidad de Lipschitz. El conjunto de datos está formado por una serie de pares de entrada y salida de la función que describen el comportamiento que ha de emular el mapa f.

$$D := \{(w_i, \tilde{f}(w_i)) \mid_i = 1, ..., N_D \tag{5.34}$$

donde $\tilde{f}$ es la observación con ruido de f. El subconjunto formado única y exclusivamente por los datos de entrada de ensayos del sistema puede ser denotado como $W_D = \text{Porj}_W(D)$.

El objetivo es el de, dado una determinada entrada nueva q, llevar a cabo la estimación del valor de $f(q)$. Dicho valor, puede ser calculado como:

$$\hat{f}_j(q;\theta,D) = \frac{1}{2}\min_{i=1,\dots,N_D}(\tilde{f}_{i,j} + L_D\|q - w_i\|^{\alpha}) + \frac{1}{2}\max_{i=1,\dots,N_D}(\tilde{f}_{i,j} - L_D\|q - w_i\|^{\alpha}) \quad (5.35)$$

donde $\tilde{f}_j(q:\theta,D)$ es la componente j de $\tilde{f}$, $\tilde{f}_{i,j}$ es la componente j del valor del mapa para el dato i en D, w_i es la entrada correspondiente, N_D el número de datos del conjunto y θ los parámetros $\{L_D,\alpha\}$.

El valor de la constante de Hölder y la contante exponencial son en principio desconocidas. En este trabajo y para llevar a cabo su cálculo, se sigue el método POKI [16], mediante el cual se optimizan ambos parámetros para minimizar el error de predicción, y no ser necesario asumir ningún tipo de distribución a priori. En el presente trabajo se utilizarán dos conjuntos de datos, uno para llevar a cabo el diseño del modelo D^{cond} y otro para la validación del mismo. Si se agrupan los parámetros como $\theta = (L_D,\alpha)$ estos pueden ser calculados como:

$$\theta := arg\min_{\theta}\frac{1}{|W^{val}|}\sum_{w\in W^{val}}\left\|\tilde{f}(w) - \hat{f}(w;\theta,D)\right\| \quad (5.36)$$

donde W^{val} es el conjunto de datos de validación.

El objetivo del uso del método de inferencia kinky será el de, dado un determinado sistema, calcular el valor de la salida del mismo en el siguiente instante de muestreo conocido el estado de la planta en el instante actual (o también el estado actual y los valores pasados del estado) y la señal de entrada del sistema, mediante el uso de un modelo de predicción basado en datos históricos de ensayo de la planta. Así, dado un determinado conjunto de datos formado por valores de trayectorias de las señales de entrada y salida de un sistema:

$$D : \{(w_k, z_k) \mid k = 1,\dots,N_D\} \quad (5.37)$$

donde $w_k = (x(k), u(k)) \in R^n$ con $n_w = n_x + n_u$ y $z_k = y(k+1)$, el modelo de predicción puede ser calculado a partir de D mediante el uso de inferencia kinky y puede ser escrito como $z_k = \tilde{f}(w_k, \theta, D)$. Con un poco de abuso de notación, esto puede ser expresado como:

$$\hat{y}(k+1) = \hat{f}(x(k), u(k); \theta, D) \tag{5.38}$$

5.5 Implementación en Python

En esta sección se llevará a cabo la implementación de un control predictivo basado en datos mediante el uso de un modelo basado en inferencia kinky. El proceso de diseño del controlador se dividirá a su vez en dos etapas bien diferenciadas. La primera de ellas es la de llevar a cabo una serie de ensayos sobre el sistema para la obtención del set de datos, tanto de diseño como de validación, proceso el cual, basado en trabajos previos, será llevado a cabo mediante el uso de Matlab. Obtenido ya el conjunto de datos a utilizar, dichos datos serán exportados a la plataforma de desarrollo de Python, donde, una vez diseñado el algoritmo que implementa la inferencia kinky, será utilizado para el desarrollo de un control predictivo basado en datos.

El sistema utilizado sobre el cual se llevará a cabo la implementación de la estrategia de control es un tanque de almacenamiento de agua, el cual tiene una única entrada, el caudal de agua de entrada al tanque, y una única salida, el nivel o altura del fluido en el interior del tanque. Dicho tanque tiene un orificio en su base, a través del cual el agua cae por efecto de la gravedad. De esta forma, la ecuación dinámica que define el comportamiento del tanque es:

$$\frac{dh}{dt} = (q - k_d\sqrt{h})/A \tag{5.39}$$

donde q es el caudal de entrada del tanque en metros cúbico por segundo, k_d el coeficiente de descarga, h el nivel de líquido en el mismo, y A el área de la base. Esta ecuación, una vez muestreada, y tomando un tiempo de integración T, puede ser

expresada como:

$$h(k+1) = h(k) + T(q(k) - k\sqrt{h(k)})/A \tag{5.40}$$

donde k_d es el instante actual de muestreo. Dicho sistema ha de ser ensayado para poder registrar la salida del mismo en función de la entrada. Existen numerosas alternativas a la hora de elegir la señal de entrada con la cual el sistema será ensayado. De entre todas ellas, se ha optado por la utilización de una señal chirp, o señal seno de frecuencia variable, es decir, una señal de la forma:

$$x(t) = sin(\phi(t)) \tag{5.41}$$

donde $\phi(t)$ es una función dependiente del tiempo, que puede ser de distinto carácter, por lo general, una recta de pendiente positiva, es decir, se generará una señal seno cuya frecuencia irá aumentando con el tiempo. Se muestra a continuación la señal utilizada.

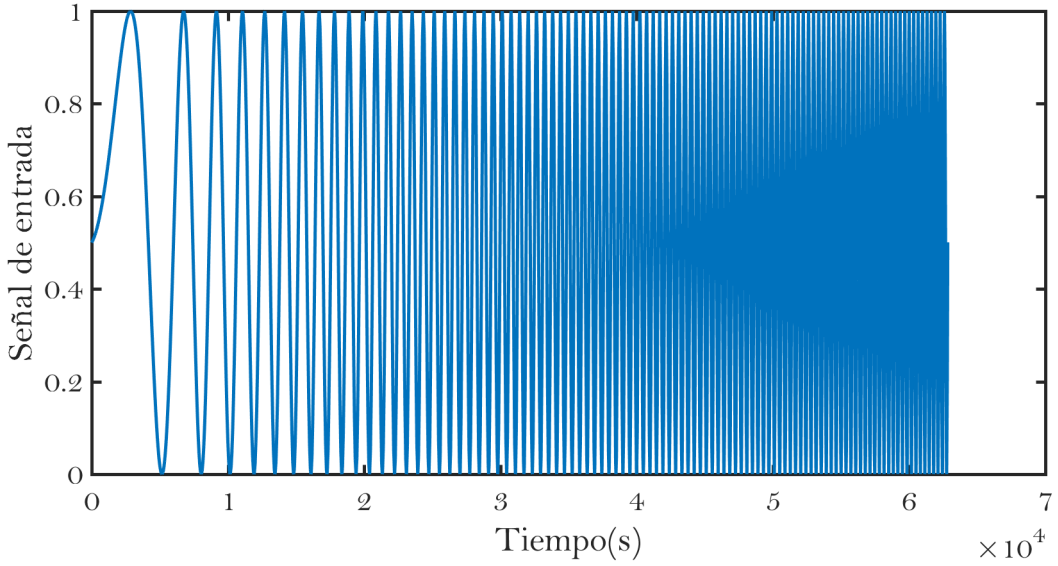


Figura 5.10: Señal chirp de ensayo del sistema

Ante dicha señal de entrada se ha registrado la siguiente señal de salida, a la que además se le ha añadido cierta cantidad de ruido gaussiano.

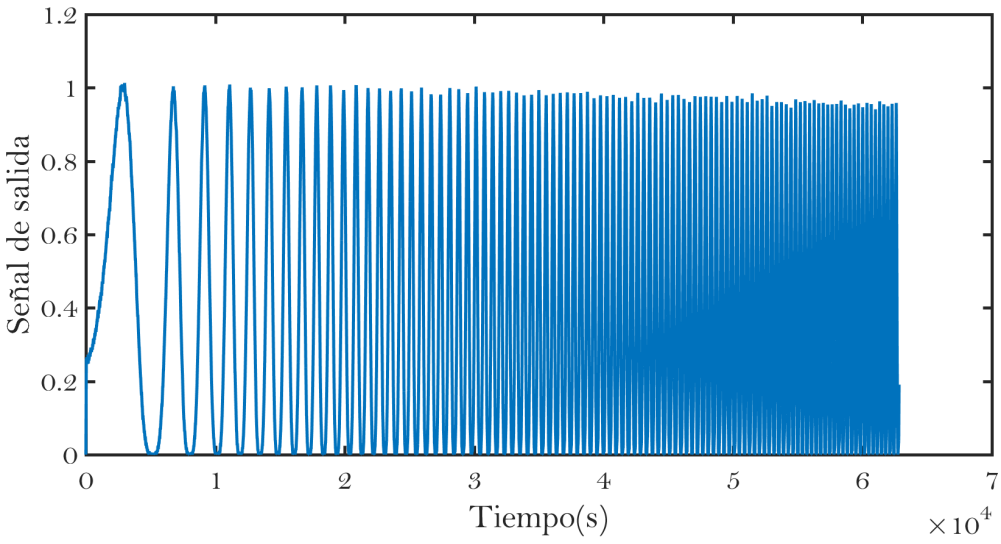


Figura 5.11: Señal de salida del sistema ante ensayo con chirp

Vistas las señales de entrada y de salida del sistema, resulta de especial interés llevar a cabo el cálculo de la característica estática del sistema, por un doble motivo. El primero de ellos para ver cómo los datos obtenidos se agrupan en torno a dicha característica, y en segundo lugar para saber cuáles han de ser los posibles puntos de equilibro del sistema seleccionado.

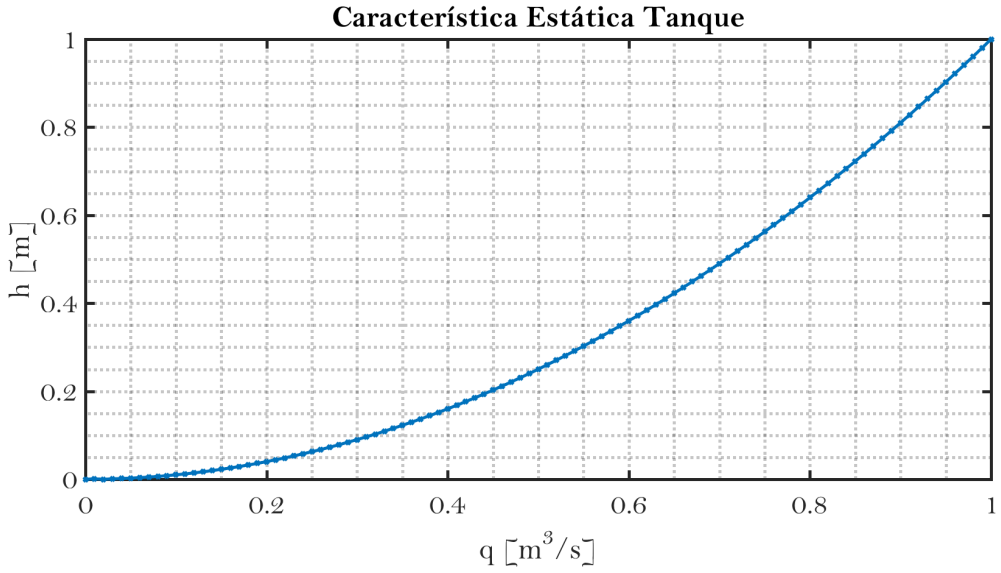


Figura 5.12: Característica estática del sistema

Si ahora se representan los puntos de entrada del sistema con respecto a los puntos de salida se puede ver cómo estos se agrupan siempre en torno a la curva característica, tal y como se puede apreciar en la siguiente imagen.

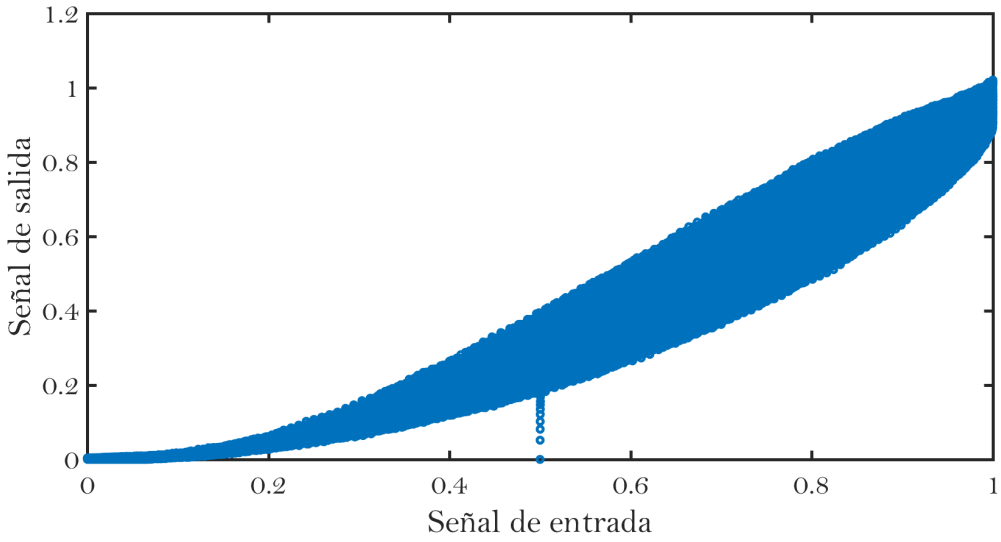


Figura 5.13: Datos en torno a la característica estática

Estos datos generados serán los utilizados para llevar a cabo el control predictivo basado en datos. Para llevar a cabo la validación del buen comportamiento del sistema se utilizará una señal distinta, en este caso una señal PRBS, señal de secuencia binaria pseudoaleatoria. Se muestra en la siguiente gráfica un ejemplo de señal de entrada PRBS al sistema así como la salida generada.

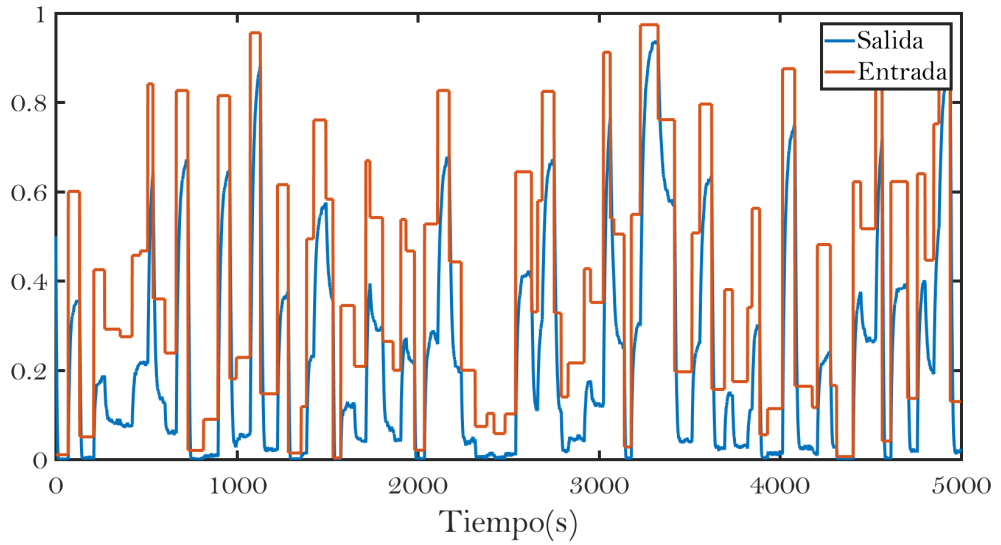


Figura 5.14: Entrada y salida mediante señal PRBS

Una vez que sen han llevado a cabo los pertinentes ensayos y se han obtenido los datos que será utilizados, D^{val} y D^{test}, se está en disposición de exportar dichos datos desde Matlab hasta Python. Sin embargo, debido al gran número de datos que conforman el conjunto, y con el objetivo de reducir el coste computacional, se ha optado

por llevar a cabo un proceso de esquilmado, mediante el cual se reducirá el número de puntos mediante la eliminación de puntos muy cercanos entre sí. Se muestra a continuación la importación de datos desde Python, tanto del conjunto de datos de diseño como de los datos de prueba.

```
#Importacion de datos para test
y_test= sio.loadmat('y_t')
u_test= sio.loadmat('u_t')

y_t=y_test['y_t']
u_t=u_test['u_t']

#Importacion de datos para modelo
y_esquilmada= sio.loadmat('y_esquilmada')
u_esquilmada= sio.loadmat('u_esquilmada')

y_red=y_esquilmada['y_esquilmada']
u_red=u_esquilmada['u_esquilmada']
```

Importados ya los datos, es posible llevar a cabo el diseño de la función que implementa la inferencia kinky, es decir, una función que toma como argumentos el estado actual del sistema y la entrada del mismo y devuelve el valor estimado en el instante siguiente. Se muestra a continuación dicha función.

```
def KINKY(q,y_init):

    f_w_hist=[]

    w_query=np.column_stack((q,y_init))
    w_ant=w
```

```
    superior=[]
    inferior=[]

    LD=1.008663
    alpha=1

    for i in range(1,len(w)):
        w_dif=w_query-w_ant[i-1]
        val_sup=y_red[i]+LD*LA.norm(w_dif,np.inf)**alpha
        val_inf=y_red[i]-LD*LA.norm(w_dif,np.inf)**alpha

        superior.append(val_sup)
        inferior.append(val_inf)

    lim_sup=min(superior)
    lim_inf=max(inferior)

    f_w=(1/2)*(lim_sup+lim_inf)
    f_w_hist.append(f_w)
    y_init=f_w

    f_w_float=f_w[0]

    return f_w_float
```

Dado que el sistema es de primer orden, se ha elegido $n_a = n_b = 1$, de tal forma que se construye w_k como (y_k, u_k). Además se han obtenido los valores de L_D y α usando el método POKI con ensayos de validación tal y como se muestra a continuación.

Antes de proceder a realizar el diseño del controlador mediante el uso de la función anterior como modelo de predicción, es conveniente comprobar que efectivamente el modelo se comporta como el sistema real. Para ello, se hará uso del conjunto de validación y se comprobará que los datos devueltos con la función KINKY() coinciden con

los datos de salida del sistema ensayado.

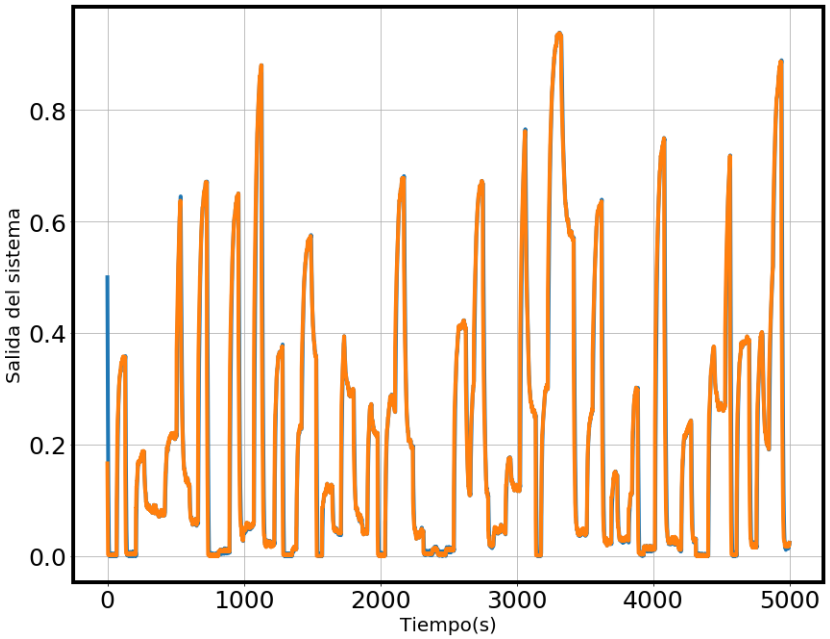


Figura 5.15: Salida de PRBS usando modelo kinky

Puede observarse como la salida obtenida mediante el uso del modelo basado en inferencia kinky coincide prácticamente a la perfección con la salida real del sistema.

Una vez que se ha comprobado el correcto funcionamiento de la función que lleva a cabo la predicción mediante inferencia kinky, son tres las funciones que aún se han de definir. Por un lado, será necesario llevar a cabo la implementación de una función que simule el comportamiento real del sistema para que esta pueda ser usada como simulador de éste ante cada una de las salidas propuestas por el controlador. Por otro lado, será necesario llevar a cabo una función en la cual se lleve a cabo el cálculo del coste de la trayectoria, función la cual será la encargada de llamar a la función de inferencia kinky a modo de modelo de predicción. Por último, será necesario definir una función en la cual se lleva a cabo la optimización de la función de coste. Se muestran a continuación dichas funciones.

Simulador del sistema.

```
def simulador(x,u):

    A=1
    K=1
    Tm=0.1

    #Calculo del valor de salida
    x=x+Tm*(u-K*math.sqrt(x))/A

    return x
```

Función de coste.

```
def Coste(us,ys,y_ref,u_ref):
    Lvect=[]

    R=1
    Q=10

    y_actual=ys

    for i in range(0,len(us)):

        #Calculo del coste de una etapa
        L= Q*(y_actual-y_ref)**2+R*(us[i]-u_ref)**2

        #Prediccion mediante kinky
        y_actual=KINKY(us[i],y_actual)
        Lvect.append(L)

    J=sum(Lvect)
```

```
    return J
```

Nótese que en este MPC no se ha utilizado ni coste ni restricción terminal por lo que la estabilidad no está garantizada, ya que no era el objetivo principal de este apartado.

Función de optimización.

```
def solver(y,y_ref,u_ref,u0):

    #Seleccion de metodo SQP
    solution=minimize(Coste,u0,args=(y,y_ref,u_ref)
    ,method='SLSQP')
    u=solution.x

    return u
```

Por último, una vez que se han definido todas y cada una de las funciones necesarias, sólo queda llevar a cabo la implementación del bucle de control del sistema, para lo cual simplemente se ha de llamar constantemente al solver y actualizar los valores iniciales del problema de optimización para la siguiente iteración tal y como se ha hecho en los controladores implementados con anterioridad.

```
#Tiempo de simulacion
time=50

#Almacenamiento valores historicos

u_hist=[]
y_hist=[]

y_mpc=0
```

```
y_ref=0.4624
u_ref=0.68

while(time>0):

    u0=np.array([0,0,0,0])

    #Llamada al solver
    u_aux=solver(y_mpc,y_ref,u_ref,u0)
    u_mpc=u_aux[1]

    #Llamada al simulador
    y_mpc=simulador(y_mpc,u_mpc)

    u_hist.append(u_mpc)
    y_hist.append(y_mpc)

    time=time-1
```

Ejecutando el código anterior, y habiendo tomado como valor de referencia un punto de la característica estática, se han obtenido los siguientes resultados, tanto de la entrada como salida del sistema.

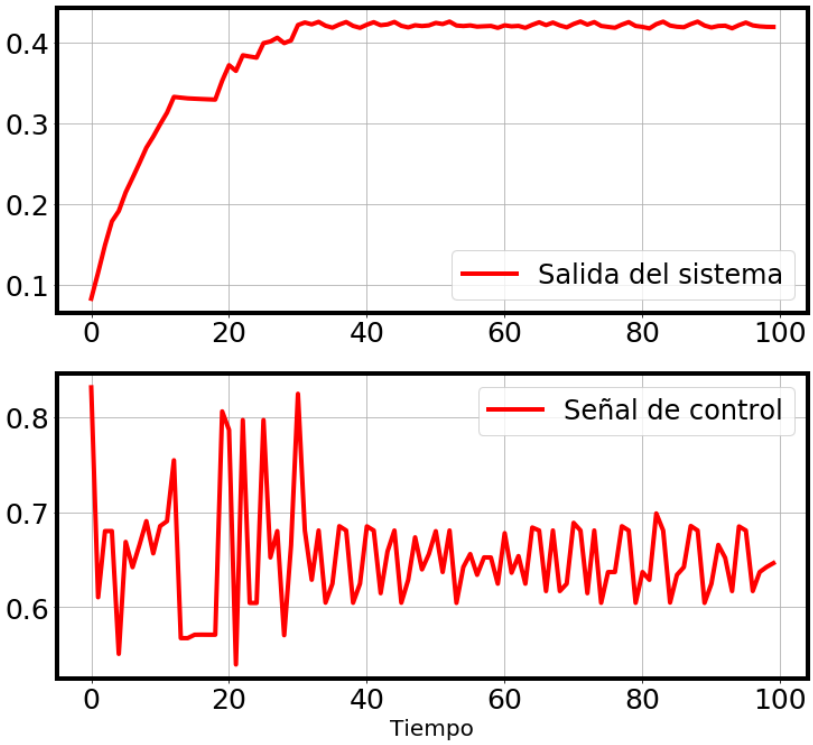


Figura 5.16: Señal de control y salida del sistema controlado

Conclusiones del Trabajo y Mejoras de Futuro

A lo largo de este trabajo se realizado una revisión general del control predictivo no lineal así como de diversos métodos y algoritmos de optimización de problemas no lineales sujetos a restricciones. Estos conocimientos han sido utilizados para explicar los principios de funcionamiento de una herramienta en lenguaje de programación Python para la resolución de problemas de optimización, Pyomo, la cual ha sido utilizada para la implementación de controladores predictivos no lineales en tiempo continuo basados en las ecuaciones dinámicas que definen el comportamiento del sistema, habiéndose comprobado efectivamente su buen funcionamiento para este tipo de problemas.

Además, se intentó llevar a cabo la utilización de esta herramienta para la resolución de problemas basados en datos con modelo de caja negra, en los cuales no se conocía nada acerca del sistema y sólo se disponía de datos fruto de ensayos sobre la planta. Tras intentarlo, y tras consultar con los desarrolladores de la herramienta, se comprobó que actualmente esto no es posible, ya que Pyomo no está preparado aún para la utilización de modelos de caja negra. La modificación del núcleo del programa para la implementación de esta funcionalidad es un proceso que, aunque posible, es tedioso y se sale del ámbito de estudio del presente trabajo.

Tras ello, y en aras de futuros estudios e investigaciones en el ámbito del control predictivo no lineal, se ha llevado a cabo la implementación de un control predictivo mediante el uso nuevamente de lenguaje Python y basado en la técnica de aprendizaje automatizado conocida como inferencia kinky, basándose en trabajos previos llevados a cabo en el departamento. Para ello, primeramente se ha realizado una introducción al control predictivo no lineal económico, en datos, y con modelos de predicción basados en datos, como son redes neuronales y procesos gaussianos.

Cabe decir que todo el trabajo realizado ha sido con vistas a futuros trabajos en el ámbito del control predictivo económico, para lo cual podrían ser utilizadas algunas de las técnicas y herramientas aquí presentadas. Así mismo, la realización del presente trabajo ha tenido como fin último servir de iniciación a la investigación en el campo del control predictivo, así como base para la realización de una futura tesis doctoral en la cual se pretende ampliar todo lo aquí expuesto. Algunas de las posibles mejoras podrían ser, por ejemplo, la modificación de la herramienta para su mejor adecuación a problemas de control predictivo, el uso de plataformas hardware que permitan el procesamiento en paralelo, el uso de las librerías de Pyomo para trabajar con sistemas estocásticos sujetos a restricciones, etc.

REFERENCIAS

[1] Lars Grüne, Jürgen Pannek**.** *Nonlinear Model Predictive Control, Theory and Algprithms. Second Edition.* Springer, Communications and Control Engineering, 2016. *ISBN* 9783319460246.

[2] David Q. Mayne, Jambes B. Rawlings. Model Predictive Control, Theory and Desing**.** First Ediction**.** Nob Hill Publishing, LLC, 2015. ISBN 9780975937709.

[3] D. Limón. *Control predictivo de sistemasno lineales con restricciones: estabilidad y robustez.* Tesis Doctoral, Universidad de Sevilla, 2002.

[4] M. Vidyasagar. *Nonlinear Systems Analisys. Second Edition.* Prentice-Hall, Englewood OHTs, 1993. ISBN 0136234631.

[5] Lorentz T. Biegler. *Nonlinear Programming, Concepts, Algorithms and Applications to chemical Processes.* MOS-SIAM Series on Optimization, 2010. ISBN 9780898717020.

[6] Daniel Limón Marruedo, Ignacio Alvarado Aldea, Teodoro Álamo Cantarero, Merio Pereira Martín. *Operación Óptima de la Planta de Cuatro Tanques.* CEA, Comité Español de Automática, 2014.

[7] Matthew Ellis, Jinfeng Liu, Panagiotis D. Christofides. *Economic Model Predictive Contro, Theroy, Formulations and Chrmical Processes Applications.* Springer, Advances in Inductrial Control, 2017. ISBN 9783319411071.

[8] Brett T. Stewart, Aswin N. Venkat, James B. Rawlings, Stephen J. Wright, Gabriele Pannocchia. *Cooperative distributed model predictive control.* ELSEVIER, Systems and Control Letters, 2010.

[9] Jus Kocijan1, Roderick Murray-Smith, Carl Edward Rasmussen, Bojan Likar. *Predictive control with Gaussian process models*. Computer as a Tool. The IEEE Region 8 (Vol. 1, pp. 352-356). IEEE, 2003.

[10] Krzysztof Patan, Piotr Witczak. *Robust model predictive control using neural networks.* IEEE International Symposium on Intelligent Control (ISIC), 2014.

[11] D Limon, J Calliess, JM Maciejowski - IFAC-PapersOnLine. *Learning-based Nonlinear Model Predictive Control.* IFAC-PapersOnLine, 2017.

[12] Alex Smola, S.V.N. Vishwanathan. *Introduction to Machine Learning.* Cambridge University Press, 2010, ISBN 0521825830.

[13] Simon Haykin. *Neural Networks and Leaning Machines. Third Edition.* Pearson Prentice Hall. 2009. ISBN 10987654321.

[14] Jan-Peter Calliess. *Conservative decision-making and inference in uncertain dynamical systems.* Tesis Doctoral, University of Oxford, 2014.

[15] R. Rajeswari, M. Shyamalagowri. *Model Predictive Control Design for Nonlinear Process Control Reactor Case Study: CSTR (Continuous Stirred Tank Reactor).* Journal of Electrical and Electronics Engineering (IOSR-JEEE), 2013.

[16] Calliess, Jan-Peter**.** *Lipschitz optimisation for lipschitz interpolation*. *American Control Conference (ACC), 2017.* IEEE, 2017.