Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Implementación de controladores predictivos en LabVIEW

Autor: José Ramón Romero Gómez
Tutor: Daniel Limón Marruedo

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

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Implementación de controladores predictivos en LabVIEW

Autor:
José Ramón Romero Gómez

Tutor:
Daniel Limón Marruedo
Profesor Titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Implementación de controladores predictivos en LabVIEW

Autor: José Ramón Romero Gómez

Tutor: Daniel Limón Marruedo

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi familia

Agradecimientos

Este proyecto representa la culminación del esfuerzo y el trabajo realizado durante los años de estudio de este Grado. En primer lugar quiero dar las gracias a mis padres, no solo por haberme dado esta la oportunidad, sino por su incondicional apoyo. De igual forma quiero agradecer a toma mi familia su apoyo y confianza en mí para la realización de estos estudios. Agradecer también a mis amigos y a mis compañeros de clase haber sido una gran ayuda durante el transcurso de estos años.

Quiero agradecer personalmente a mi tutor Daniel Limón Marruedo por su tiempo, su dedicación y su cercanía. Todo ello ha hecho posible la realización de este Trabajo Fin de Grado. De igual forma, agradecer a Pablo Krupa su tiempo y ayuda en los inicios de este proyecto.

José Ramón Romero Gómez

Sevilla, 2016

Resumen

El objetivo de este proyecto es la implementación de un Control Predictivo Basado en Modelo en la plataforma de programación LabVIEW. Por ello, nos vamos a centrar en la resolución de problemas de optimización cuyo coste sea cuadrático.

Esta resolución se va a realizar mediante algoritmos de optimización tales como el ISTA y el FISTA. Estos algoritmos van a ser implementados tanto para problemas de optimización con matrices no diagonales como para problemas de optimización con restricciones de igualdad. La gran ventaja que se obtiene del uso de estos algoritmos es la facilidad de implementación que presentan para así poder ser aplicados en computadores de bajas prestaciones. El tiempo de convergencia de dichos algoritmos será comparado con la función *quadprog* de MATLAB.

El MPC tendrá asociado el Steady-State Target Optimization, que permitirá realizar el cálculo de la referencia alcanzable a partir de una referencia deseada. Este cálculo es también un problema de optimización pudiéndose obtener de forma alternativa una solución factible al SSTO sin resolver dicho problema de optimización.

De igual forma, el MPC cuenta con un observador de estado del cual a partir de las medidas de la planta estime el estado actual del sistema a controlar. Tanto el observador de estado como el SSTO tendrán asociados filtros con el fin de evitar perder la factibilidad del problema de optimización al producirse cambios bruscos en la referencia.

A modo de comprobación y para realizar la implementación completa del MPC, se simulará junto con un sistema. El sistema elegido para la simulación será la planta de los 4 tanques que se encuentra en el Departamento de Ingeniería de Sistemas y Automática, de la que se conocen tanto su modelo lineal como su modelo no lineal. Esta implementación en LabVIEW se hará mediante una DLL programada en lenguaje C que contenga tanto a la función que simula el controlador predictivo como a la función que simula el modelo no lineal de dicha planta.

Abstract

The aim of the project is to implement a Model Based Predictive Control on LabVIEW. Therefore, we will focus on solving optimization problems whose cost is quadratic.

Resolving will be performed by using optimization algorithms such as ISTA and FISTA. These optimization algorithms will be implemented for both optimization problems with non diagonal matrices and with matrices with equalily restrictions. The greatest advantage of using the previously mentioned algorithms is the ease of implementing them into low end computers. The convergence time of these algorithms will be later compared with the quadprog function of MATLAB.

The MPC will be associated with the Steady-State Target Optimization, which will allow the calculation of a reachable reference from a desired reference. Since this calculation is an optimization problem as well, there could be an alternative solution feasible to the SSTO without solving the optimization problem.

Likewise, the MPC has a state observer which can estimate the state of the given system from the measurement of the plant. Both the state observer and the SSTO will be associated to filters amied to avoid the loss of the feasibility of the optimization problem due to sudden changes in the reference.

A simulation system will be used to accomplish a full implementation and checking of the MPC. The simulation system that was chosen for this purpose is the quadruple tank system belonging to the Systems and Automatic Egineering Department, of whom the linear and nonlinear models are known. Implementing in LabVIEW wil be done through a DLL programmed in C programming language containing both the function simulating the predictive controller and the function simulating the non-linear model of such a plant.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

:	Tal que
$<$	Menor
$>$	Mayor
$\geq$	Menor o igual
$\leq$	Mayor o igual
s. a.	Sujeto a
$\in$	Perteneciente
Δ	Incremento
∇	Jacobiano
A^T	Transpuesta
I	Matriz identidad
$\|z\|$	Valor absoluto
z^*	Óptimo
x^0	Valor inicial
x_k	Valor en el instante de tiempo k
$\hat{x}$	Valor estimado

1 INTRODUCCIÓN A LA OPTIMIZACIÓN

El objetivo de la optimización consiste en calcular el conjunto de variables que minimizan (o maximizan) un determinado criterio satisfaciendo un conjunto de restricciones sobre las mismas. El criterio se debe elegir de manera adecuada para hacer que la solución sea única.

En este capítulo nos vamos a centrar en resolver problemas de optimización cuyo coste es cuadrático. Para la resolución de estos problemas van a cobrar gran importancia los algoritmos que se utilicen para resolverlos. Éstos deben ser sencillos de implementar en computadores de bajas prestaciones para así poder ser utilizados por un amplio abanico de equipos de control.

1.1 Problema 1

Sea z un número real, calcular z^*, la solución del problema

$$\min_z \quad \frac{1}{2}dz^2 + cz \tag{1.1}$$

$$s.a. \quad a \leq z \leq b \tag{1.2}$$

siendo $a < b$ y $d > 0$.

Solución

La función para la que buscamos la z mínima, se corresponde con la ecuación de una parábola definida tal que

$$y = \frac{1}{2}d(z - z_0)^2 + y_0 \tag{1.3}$$

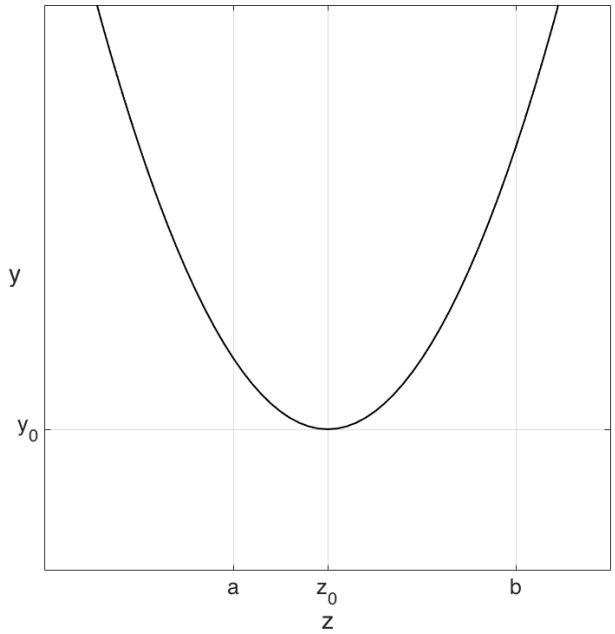

Figura 1-1. Parábola convexa.

Es fácil comprobar gráficamente que la z que hace mínima y será z_0 siempre que ésta esté comprendida entre los valores a y b. En caso contrario, tomará el valor del límite más cercano tal que

Tabla 1-1. Solución del problema 1.

Si $a \leq z_0 \leq b$	$z^* = z_0$
Si $z_0 > b$	$z^* = b$
Si $z_0 < a$	$z^* = a$

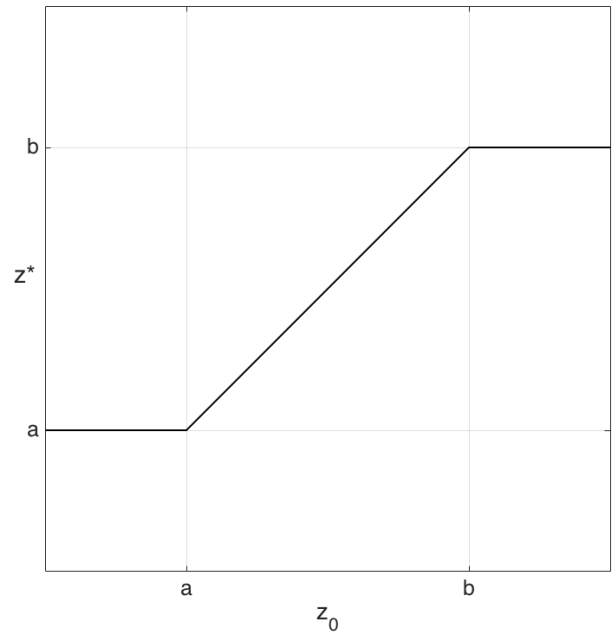

Figura 1-2. Solución del problema 1.

Como estamos buscando la solución mínina, al desarrollar la ecuación [1.3], se pueden eliminar todos los términos constantes que no sean dependientes de z.

$$y = \frac{1}{2}dz^2 + \frac{1}{2}dz_0^2 - dzz_0 + y_0 \quad [1.4]$$

$$y = \frac{1}{2}dz^2 - dzz_0 \quad [1.5]$$

Por lo tanto, igualando [1.5] con la función [1.1] que queremos minimizar, obtenemos

$$\frac{1}{2}dz^2 - dzz_0 = \frac{1}{2}dz^2 + cz \quad [1.6]$$

$$z_0 = -\frac{c}{d} \quad [1.7]$$

1.2 Problema 2

Sea $z \in \mathbb{R}^n$ un vector de números reales, calcular z^*, la solución del problema

$$\min_z \quad \frac{1}{2}z^T Dz + C^T z \quad [1.8]$$

$$s.a. \quad a_i \leq z_i \leq b_i, \quad i = 1, \dots, n \quad [1.9]$$

siendo $a_i < b_i$ y siendo $D \in \mathbb{R}^{n \times n}$ una matriz diagonal tal que $d_{ii} > 0$.

Solución

Se comprueba que la solución de este problema es similar a la del problema anterior. Desarrollando la ecuación [1.8] se obtiene

$$\frac{1}{2}z_1^2 d_{11} + c_1 z_1 + \cdots + \frac{1}{2}z_n^2 d_{nn} + c_n z_n \quad [1.10]$$

Por lo que nuestro problema se puede separar en n problemas distintos, siendo cada uno de ellos, similar al problema 1. La solución será $z^* \in \mathbb{R}^n$, cuyas componentes se corresponden con las soluciones individuales de cada problema.

Tabla 1-2. Solución del problema 2.

Si $a_i \leq z_{0_i} \leq b_i$	$z_i^* = z_{0_i}$
Si $z_{0_i} > b_i$	$z_i^* = b_i$
Si $z_{0_i} < a_i$	$z_i^* = a_i$

Donde

$$z_{0_i} = -\frac{c_i}{d_{ii}} \quad [1.11]$$

Como hemos podido comprobar, la minimización de funciones de coste simples no supone una gran dificultad matemática. El reto ahora es hacer el cálculo de estos problemas de optimización lo más rápido y eficiente posible para una dimensión $\mathbb{R}^n$. En función del tipo de matrices de las que se compone la función de coste y de las posibles restricciones adicionales, la dificultad del problema de optimización varía.

1.3 Problema de optimización con matrices no diagonales

El primer problema que agrava la dificultad del cálculo es el hecho de que la matriz hessiana anteriormente llamada D, ahora no sea diagonal. Esta matriz no diagonal pasa ahora a llamarse H quedando el problema de optimización con matrices no diagonales definido tal que

Sea $z \in \mathbb{R}^n$ un vector de números reales, calcular z^*, la solución del problema

$$\min_z \qquad \frac{1}{2} z^T H z + f^T z \tag{1.12}$$

$$s.a. \qquad a_i \leq z_i \leq b_i, \quad i = 1, \dots, n \tag{1.13}$$

siendo $a_i < b_i$, y $H \in \mathbb{R}^{n \times n}$ definida positiva tal que $H \geq 0$.

Nace por tanto la necesidad de nuevos algoritmos que permitan el cálculo de la z^* que minimiza la función de coste con matrices no diagonales. Además es interesante hacer estos algoritmos lo más rápido y eficientes posibles. La rapidez de convergencia de estos algoritmos se va a comparar con el método usado por la función *quadprog* de MATLAB.

1.3.1 Algoritmo ISTA

Sea definida la función de coste como

$$g(z) = \frac{1}{2} z^T H z + f^T z \tag{1.14}$$

y sea

$$L = \lambda_{max}(H) \tag{1.15}$$

se cumple que

$$z^T H z \leq L z^T z \tag{1.16}$$

para todo z. Si se deriva una aproximación local de $g(z)$ en torno a un punto z_k se obtiene

$$g(z_k + \Delta z) = \frac{1}{2} (z_k + \Delta z)^T H (z_k + \Delta z) + f^T (z_k + \Delta z) \tag{1.17}$$

Desarrollando los términos

$$g(z_k + \Delta z) = \frac{1}{2} \Delta z^T H \Delta z + z_k^T H \Delta z + \frac{1}{2} z_k^T H z_k + f^T z_k + f^T \Delta z \tag{1.18}$$

Y reagrupando

$$g(z_k + \Delta z) = \frac{1}{2} \Delta z^T H \Delta z + (f + H z_k)^T \Delta z + g(z_k) \tag{1.19}$$

Podemos aplicar la ecuación [1.16] tal que

$$g(z_k + \Delta z) \leq \frac{1}{2} L \Delta z^T \Delta z + (f + H z_k)^T \Delta z + g(z_k) \quad [1.20]$$

El algoritmo ISTA consiste en los siguientes pasos:

1. Inicializar $k \leftarrow 0$
2. Inicializar $z_0 \leftarrow 0$
3. Inicializar $FIN \leftarrow NO$
4. Repetir hasta que se cumpla $FIN = SI$
    a. Calcular Δz tal que

$$\min_{\Delta z} \quad \frac{1}{2} L \Delta z^T \Delta z + (f + H z_k)^T \Delta z$$

$$s.a. \quad a_i \leq z_{ki} + \Delta z_i \leq b_i, \quad i = 1, \dots, n$$

    b. Hacer $z_{k+1} \leftarrow z_k + \Delta z$
    c. Incrementar $k \leftarrow k + 1$
    d. Si se cumple $|\Delta z_i| \leq TOL$, entonces hacer $FIN \leftarrow SI$
5. Hacer $z^* \leftarrow z_k$

Se observa que en el punto 4.a. el algoritmo ISTA resuelve del problema 1 desarrollado en el capítulo anterior para $D \in \mathbb{R}^{n \times n}$ definido como

$$D = \begin{bmatrix} L & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & L \end{bmatrix} \quad [1.21]$$

Y para $C \in \mathbb{R}^n$ definido como

$$C = f + H z_k \quad [1.22]$$

Siendo la variable de decisión $\Delta z \in \mathbb{R}^n$.

Para la comparativa de tiempos de convergencia se generan aleatoriamente con MATLAB varios problemas de optimización que cumplan las especificaciones del problema. Se generan por tanto, aleatoriamente, los parámetros de la función de coste H y f así como los límites a y b entre los que se debe encontrar la solución. El problema es ahora resuelto tanto por la función *quadprog* incluida en MATLAB como por el algoritmo ISTA programado en MATLAB. Comparando los tiempos de convergencia de ambos métodos para los mismos problemas generados aleatoriamente se observa la rapidez de convergencia del nuevo algoritmo.

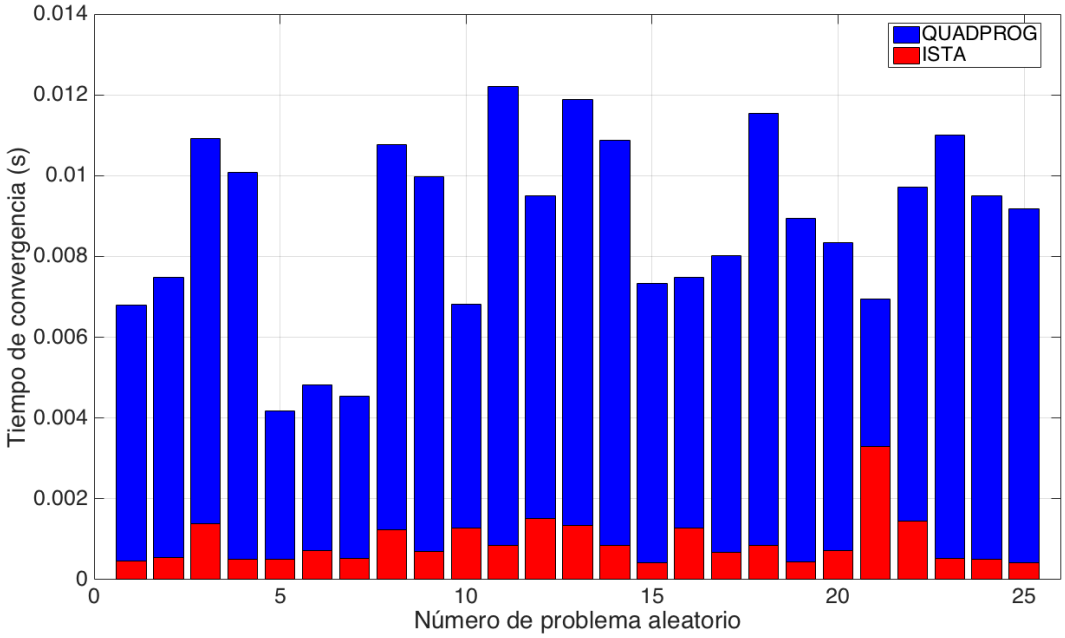

Figura 1-3. Comparativa de tiempos de convergencia QUADPROG – ISTA.

1.3.2 Algoritmo FISTA

Este segundo algoritmo consiste en una mejora del ISTA. El algoritmo FISTA basa su punto de linealización en un punto distinto de la solución. Dicho punto se denota como y_k.

El algoritmo FISTA consta de los siguientes pasos.:

1. Inicializar $k \leftarrow 1$
2. Inicializar $z_0 \leftarrow 0$
3. Inicializar $y_1 \leftarrow 0$
4. Inicializar $t_1 \leftarrow 1$
5. Inicializar $FIN \leftarrow NO$
6. Repetir hasta que se cumpla $FIN = SI$
   a. Calcular Δy tal que

$$\min_{\Delta y} \quad \frac{1}{2} L \Delta y^T \Delta y + (f + H y_k)^T \Delta y$$

$$s.a. \quad a_i \leq y_{ki} + \Delta y_i \leq b_i, \quad i = 1, \dots, n$$

   b. Hacer $z_k \leftarrow y_k + \Delta y$
   c. Hacer $t_{k+1} \leftarrow \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$
   d. Hacer $y_{k+1} \leftarrow z_k + \frac{t_k - 1}{t_{k+1}}(z_k - z_{k-1})$
   e. Incrementar $k \leftarrow k + 1$
   f. Si se cumple $|\Delta z_i| \leq TOL$, entonces hacer $FIN \leftarrow SI$
7. Hacer $z^* \leftarrow z_k$

Se observa de nuevo que en el punto 6.a. del algoritmo FISTA se vuelve a resolver el problema 1 para la variable de decisión Δy siendo $D \in \mathbb{R}^{n \times n}$ definido como

$$D = \begin{bmatrix} L & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & L \end{bmatrix} \quad [1.23]$$

Y $C \in \mathbb{R}^n$ definido como

$$C = f + Hy_k \quad [1.24]$$

Los puntos 6.c. y 6.d. del algoritmo FISTA son la gran diferencia con respecto al algoritmo ISTA. Estos puntos hacen que este algoritmo avance aún más rápido.

De nuevo, generando problemas aleatorios con MATLAB y comparando los tiempos de convergencia del algoritmo FISTA con la función *quadprog,* es fácil observar la clara ventaja que suponen estos algoritmos.

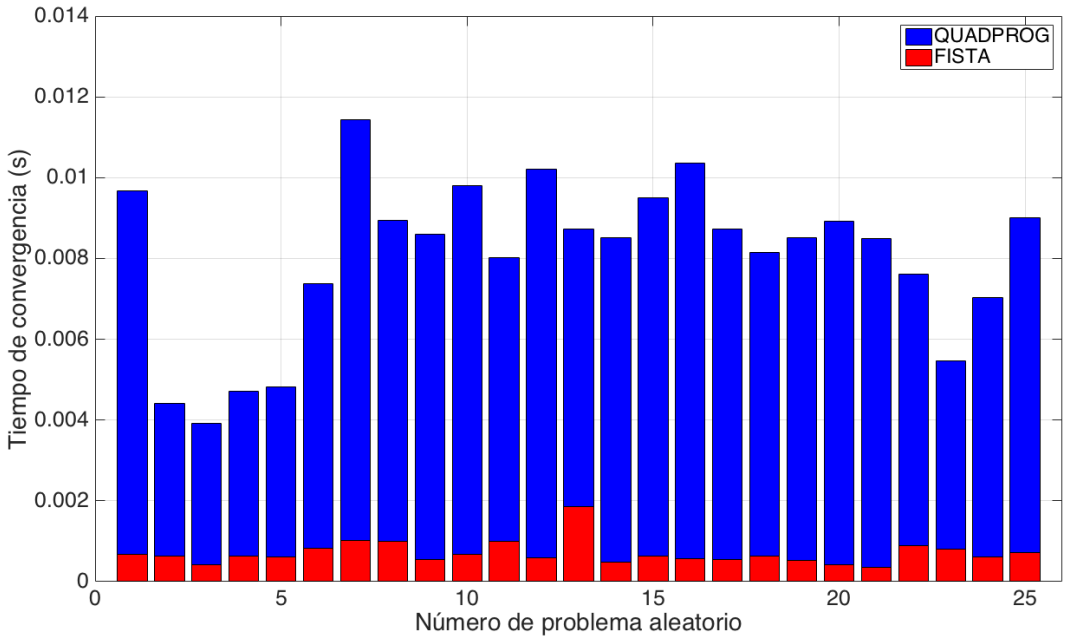

Figura 1-4. Comparativa de tiempos de convergencia QUADPROG – FISTA.

1.4 Problema de optimización con restricciones de igualdad

El segundo problema que agrava la dificultad del cálculo es la existencia de restricciones de igualdad. En el siguiente problema de optimización se añaden restricciones de igualdad para un problema de optimización cuya matriz D es diagonal. Dicho problema queda definido tal que

Sea $z \in \mathbb{R}^n$ un vector de números reales, calcular z^*, la solución del problema

$$\min_z \quad \frac{1}{2} z^T D z + f^T z \quad [1.25]$$

$$s.a. \quad LB \leq z \leq UB \quad [1.26]$$

$$Az = b \quad [1.27]$$

siendo $LB \in \mathbb{R}^n$, $UB \in \mathbb{R}^n$ y $LB_i < UB_i$. Y siendo $D \in \mathbb{R}^{n \times n}$ una matriz diagonal tal que $d_{ii} > 0$.

Siendo $A \in \mathbb{R}^{m \times n}$ y $b \in \mathbb{R}^m$, si se cumple que $m < n$, entonces la matriz $[A\ b]$ es de rango m y por tanto las restricciones son linealmente independientes.

Solución

Para resolver este problema se va a utilizar dualidad. Si definimos la función de coste del problema como

$$J(z) = \frac{1}{2} z^T D z + f^T z \quad [1.28]$$

Entonces, la función Lagrangiana del problema queda definida como

$$L(z, x) = J(z) - x^T (Az - b) \quad [1.29]$$

donde x es el multiplicador de Lagrange. Se define por tanto ahora la función

$$z^o(x) = \arg \min_{LB \le z \le UB} L(z, x) \quad [1.30]$$

y la función dual $g(x)$ es definida como

$$g(x) = L(z^o(x), x)) \quad [1.31]$$

Siendo z^* la solución óptima del problema y J^* el coste óptimo, se comprueba que

$$J^* = L(z^*, x) \ge g(x) \quad [1.32]$$

debido a que

$$Az^* - b = 0 \quad [1.33]$$

para todo x. La función dual es una cota inferior al problema de optimización, por lo que para calcular el coste óptimo J^* se busca la mejor cota inferior, es decir, se busca la x^0 que maximiza la función

$$x^0 = \arg \max_x g(x) \quad [1.34]$$

Si se define la función

$$g^* = g(x^0) \quad [1.35]$$

se tiene que

$$J^* \le g^* \quad [1.36]$$

Por lo que se demuestra que el problema posee la propiedad de dualidad fuerte y se cumple que

$$J^* = g(x^0) \quad [1.37]$$

$$z^* = z^o(x^0) \quad [1.38]$$

Pudiedo ser la ecuación [1.38] expresada de manera explícitamente directa tal que

$$z^* = \arg \min_{LB \le z \le UB} \frac{1}{2} z^T H z + (f - A^T x)^T z \quad [1.39]$$

1.4.1 Algoritmo FISTA

Para aplicar el algoritmo FISTA en este problema, basta con resolver el problema dual. Esto se basa en la desigualdad

$$g(x + \Delta x) \geq g(x) - \Delta x^T (Az(x) - b) - \frac{1}{2}\Delta x^T W \Delta x \quad [1.40]$$

siendo

$$W = AD^{-1}A^T \quad [1.41]$$

Al ser la matriz W invertible por ser D definida positiva tal que $D \geq 0$ y A de rango completo por filas, el óptimo del problema de optimización se alcanza en el punto en el que se anula el gradiente tal que

$$(Az(x) - b) + W\Delta x = 0 \quad [1.42]$$

Por lo que

$$\Delta x = -Q(Az(x) - b) \quad [1.43]$$

siendo

$$Q = W^{-1} \quad [1.44]$$

El algoritmo FISTA aplicado a este problema seguiría los siguientes pasos.

1. Inicializar $k \leftarrow 1$
2. Inicializar $x_0 \leftarrow 0$
3. Inicializar $y_1 \leftarrow 0$,
4. Inicializar $t_1 \leftarrow 1$,
5. Inicializar $FIN \leftarrow NO$
6. Repetir hasta $FIN = SI$
    a. Calcular $z(y_k)$ tal que
    $$arg \min_{LB \leq z \leq UB} \frac{1}{2} z^T D z + (f - A^T y_k)^T z$$
    b. Si se cumple $|Az(y_k) - b| \leq TOL$, entonces hacer $FIN = SI$
    c. Si no se cumple
        i. Hacer $\Delta y \leftarrow -Q(Az(y_k) - b)$
        ii. Hacer $x_k \leftarrow y_k + \Delta y$
        iii. Hacer $t_{k+1} \leftarrow \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$
        iv. Hacer $y_{k+1} \leftarrow x_k + \frac{t_k - 1}{t_{k+1}}(x_k - x_{k-1})$
        v. Incrementar $k \leftarrow k + 1$
7. Hacer $z^* \leftarrow z(y_k)$

De nuevo, se generan con MATLAB distintos problemas de optimización aleatorios y se compara el tiempo de

convergencia del algoritmo FISTA con la función *quadprog*. Una vez más, observamos que el algoritmo FISTA presenta una clara ventaja en tiempo de convergencia.

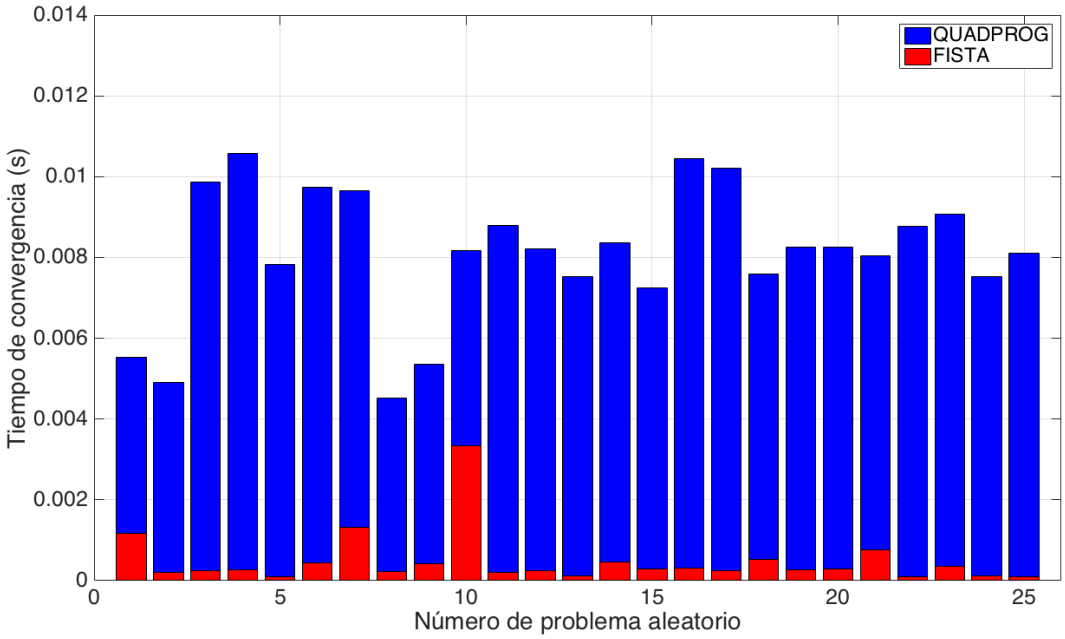

Figura 1-5. Comparativa de tiempos de convergencia con restricciones de igualdad.

2 INTRODUCCIÓN AL CONTROL PREDICTIVO

El concepto de Control Predictivo Basado en Modelo, en inglés, *Model Predictive Control,* se basa en el uso de un modelo dinámico para predecir el comportamiento del sistema y optimizar dicha predicción para poder realizar el control en tiempo real. Cobran gran importancia los modelos del sistema ya que el control óptimo del sistema depende del estado inicial del mismo.

2.1 Modelos de sistemas

El modelo dinámico de un sistema queda definido por las siguientes ecuaciones diferenciales

$$\frac{dX}{dt} = F(X, U) \quad [2.1]$$

$$Y = H(X, U) \quad [2.2]$$

$$X(0) = X^0 \quad [2.3]$$

donde $X \in \mathbb{R}^n$ representa el vector de estados del sistema, $Y \in \mathbb{R}^m$ el vector de entradas modificables del sistema, $Y \in \mathbb{R}^p$ el vector de salidas controlables del sistema y $X^0 \in \mathbb{R}^n$ el estado inicial del sistema.

2.1.1 Modelo dinámico lineal invariante con el tiempo

Este modelo es linealizable y además, se puede representar sin que varíe con el tiempo. Si linealizamos el sistema en un punto de funcionamiento (X_f, U_f) el modelo dinámico lineal invariante con el tiempo queda expresado como

$$\frac{dx}{dt} = Ax + Bu \quad [2.4]$$

$$y = Cx + Du \quad [2.5]$$

$$x(0) = x^0 \quad [2.6]$$

siendo $A \in \mathbb{R}^{n \times n}$

$$A = [\nabla_x F(X,U)]_{(X_f,U_f)} \quad [2.7]$$

la matriz de transición de estados y $B \in \mathbb{R}^{n \times m}$

$$B = [\nabla_u F(X,U)]_{(X_f,U_f)} \quad [2.8]$$

la matriz de entrada siendo $[A \quad B]$ el jacobiano de F.

La matriz $C \in \mathbb{R}^{p \times n}$ representa la matriz de salida y $D \in \mathbb{R}^{p \times m}$ la matriz que relaciona directamente la entrada modificable con la salida controlable.

El estado del sistema $x \in \mathbb{R}^n$, la entrada modificable $u \in \mathbb{R}^m$ y la salida controlable $y \in \mathbb{R}^m$ están expresados en variables incrementales con respecto al punto de funcionamiento sobre el que hemos linealizado de manera que

$$x = X - X_f \quad [2.9]$$

$$u = U - U_f \quad [2.10]$$

$$y = Y - X_f \quad [2.11]$$

siendo $x^0 \in \mathbb{R}^n$ el estado inicial del sistema expresado en variables incrementales.

2.1.2 Modelo dinámico lineal invariante con el tiempo discretizado

Este modelo dinámico lineal invariante con el tiempo se puede expresar discretizado en el tiempo. Si se muestrea para un cierto periodo T, el modelo dinámico lineal invariante en tiempo discreto se expresa como

$$x_{k+1} = Ax_k + Bu_k \quad [2.12]$$

$$y_k = Cx_k + Du_k \quad [2.13]$$

$$x_0 = x^0 \quad [2.14]$$

2.1.3 Restricciones en tiempo

Las entradas y salidas de los sistemas suelen estar limitadas. Es necesario por tanto, limitar el estado x y la actuación u del sistema. Estas restricciones se denominan restricciones en tiempo y se deben cumplir para todo k, quedando éstas expresadas como

$$x_k \in X \quad [2.15]$$

$$u_k \in U \quad [2.16]$$

siendo

$$X = \{x: \quad LB_x \leq x \leq UB_x\} \quad [2.17]$$

$$U = \{u: \quad LB_u \leq u \leq UB_u\} \quad [2.18]$$

donde los vectores LB_x y UB_x de dimensión $\mathbb{R}^n$ representan los límites inferiores (*Lower Bounds)* y superiores (*Upper Bounds)* de los n estados del sistema, y los vectores LB_u y UB_u de dimensión $\mathbb{R}^m$ representan los límites inferiores y superiores de las m entradas manipulables del sistema.

2.2 El controlador óptimo

El objetivo del controlador óptimo es que la salida controlable del sistema se estabilice en torno a una referencia $r \in \mathbb{R}^p$ tal que

$$y_k = r \quad [2.19]$$

Para ello, el sistema se debe estabilizarse en un punto de equilibrio en el que se cumpla

$$x_r = Ax_r + Bu_r \quad [2.20]$$

$$r = Cx_r + Du_r \quad [2.21]$$

Una vez definidos los siguientes cambios de variables

$$\bar{x}_k = x_k - x_r \quad [2.22]$$

$$\bar{u}_k = u_k - u_r \quad [2.23]$$

el objetivo del controlador óptimo pasa a ser

$$\bar{x} \to 0 \quad [2.24]$$

$$\bar{u} \to 0 \quad [2.25]$$

La función de coste cuadrático es definida como

$$L(\bar{x}, \bar{u}) = \bar{x}^T Q \bar{x} + \bar{u}^T R \bar{u} \quad [2.26]$$

donde $Q \in \mathbb{R}^n$ y $R \in \mathbb{R}^m$. Esta función mide el rendimiento del sistema a partir de la entrada manipulable u y el estado x del sistema. El coste de la trayectoria predicha para un horizonte de predicción N se puede expresar como

$$V_N(\bar{x}, \bar{u}) = \sum_{k=0}^{N} L(\bar{x}_k, \bar{u}_k) \quad [2.27]$$

Entonces, la ley de control óptima queda definida por

$$\min_{\bar{u}} \quad V_\infty(\bar{x}, \bar{u}) = \sum_{k=0}^{\infty} \bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k \quad [2.28]$$

$$\bar{x}_0 = \bar{x}^0 \quad [2.29]$$

$$\bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k \quad [2.30]$$

$$LB_{\bar{x}} \leq \bar{x}_k \leq UB_{\bar{x}} \quad [2.31]$$

$$LB_{\bar{u}} \leq \bar{u}_k \leq UB_{\bar{u}} \quad [2.32]$$

siendo

$$LB_{\bar{x}} = LB_x - x_r \quad [2.33]$$

$$UB_{\bar{x}} = UB_x - x_r \quad [2.34]$$

$$LB_{\bar{u}} = LB_u - u_r \quad [2.35]$$

$$UB_{\bar{u}} = UB_u - u_r \quad [2.36]$$

La ley de control óptima definida calcula el coste de la trayectoria del sistema para un horizonte de predicción infinito. El controlador óptimo tiene como función minimizar esta trayectoria con horizonte infinito.

2.2.1 Ley de control del MPC

El controlador óptimo no tiene una solución explícita y en la práctica solo puede resolverse para ciertos casos como para sistemas sin restricciones. La ley de control del Control Predictivo Basado en Modelo se basa en la ley de control óptima para un horizonte de predicción $N < \infty$. El cálculo de la trayectoria del sistema para un horizonte de predicción N definido tiene asociada una restricción adicional llamada restricción terminal. Esta restricción implica que el estado terminal del sistema x_N tiene que estar comprendido dentro de unos límites $X_F \in \mathbb{R}^n$. Por tanto, la ley de control del Control Predictivo Basado en Modelo para un horizonte de predicción N finito quedaría expresada como

$$\min_{\bar{u}} \sum_{k=0}^{N-1} \left[\bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k\right] + \bar{x}_N^T P \bar{x}_N \quad [2.37]$$

$$\bar{x}_0 = \bar{x}^0 \quad [2.38]$$

$$\bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k \quad [2.39]$$

$$LB_{\bar{x}} \leq \bar{x}_k \leq UB_{\bar{x}} \quad [2.40]$$

$$LB_{\bar{u}} \leq \bar{u}_k \leq UB_{\bar{u}} \quad [2.41]$$

$$\bar{x}_N \in X_F \quad [2.42]$$

siendo $P \in \mathbb{R}^n$ y la solución del problema

$$\bar{u}^* = \begin{bmatrix} \bar{u}_0^* \\ \vdots \\ \bar{u}_{N-1}^* \end{bmatrix} \quad [2.43]$$

2.2.1.1 Ley de control

De la solución óptima del problema de optimización se deriva la ley de control gracias a la estrategia de horizonte deslizante. Así se aplica sólo la acción de control óptima actual, recalculándose el problema de optimización en el instante siguiente para el nuevo estado del sistema.

El MPC solo aplica la actuación $\bar{u}_0^*$ calculada en el instante de tiempo $k = 0$. Para el siguente instante de tiempo calcula de nuevo la solución del problema usando el resto de la solución actual $\bar{u}_F^*$ como solución inicial del problema. De nuevo, aplica únicamente $\bar{u}_0^*$ y recalcula la solución del problema así sucesivamente.

Al realizar el control en bucle cerrado los resultados son mejores ya que, al realimentar el controlador predictivo, obtiene el estado real del sistema para el siguiente instante de tiempo k.

2.3 Control Predictivo Basado en Modelo que converge a cero

La ley de control del Control Predictivo Basado en Modelo para un sistema lineal discreto invariante con el tiempo que converge a cero de manera que $r = 0$ se puede expresar como

$$\min_{u} \sum_{k=0}^{N-1} \bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k \quad [2.44]$$

$$\bar{x}_0 = \bar{x}^0 \quad [2.45]$$

$$\bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k \quad [2.46]$$

$$LB_{\bar{x}} \leq \bar{x}_k \leq UB_{\bar{x}} \quad [2.47]$$

$$LB_{\bar{u}} \leq \bar{u}_k \leq UB_{\bar{u}} \quad [2.48]$$

$$\bar{x}_N = 0 \quad [2.49]$$

2.3.1 Correspondencia con el problema de optimización con matrices no diagonales

Este problema se puede expresar como el problema de optimización con restricciones linealmente independientes resuelto en el capítulo anterior, quedando éste expresado como

$$\min_{z} \quad z^T H z \quad [2.50]$$

$$Ez = b \quad [2.51]$$

$$LB_z \leq z \leq UB_z \quad [2.52]$$

Siendo $z \in \mathbb{R}^{N*(n+m)}$, de la forma

$$z = \begin{bmatrix} \bar{x}_0 \\ \bar{u}_0 \\ \vdots \\ \bar{x}_{N-1} \\ \bar{u}_{N-1} \end{bmatrix} \quad [2.53]$$

Siendo $H \in \mathbb{R}^{N*(n+m) \times N*(n+m)}$, de la forma

$$H = \begin{bmatrix} Q & 0 & \cdots & 0 & 0 \\ 0 & R & & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & & Q & 0 \\ 0 & 0 & \cdots & 0 & R \end{bmatrix} \quad [2.54]$$

Siendo $E \in \mathbb{R}^{n*(N+1) \times N*(n+m)}$, de la forma

$$E = \begin{bmatrix} I & 0 & 0 & 0 & 0 & & 0 & 0 \\ A & B & -I & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & A & B & -I & & 0 & 0 \\ & & \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & 0 & 0 & \cdots & A & B \end{bmatrix} \quad [2.55]$$

Siendo $b \in \mathbb{R}^{n*(N+1)}$, de la forma

$$b = \begin{bmatrix} \bar{x}^0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad [2.56]$$

Siendo $LB_z \in \mathbb{R}^{N*(n+m)}$, de la forma

$$LB_z = \begin{bmatrix} LB_{\bar{x}} \\ LB_{\bar{u}} \\ \vdots \\ LB_{\bar{x}} \\ LB_{\bar{u}} \end{bmatrix} \quad [2.57]$$

Siendo $UB_z \in \mathbb{R}^{N*(n+m)}$, de la forma

$$UB_z = \begin{bmatrix} UB_{\bar{x}} \\ UB_{\bar{u}} \\ \vdots \\ UB_{\bar{x}} \\ UB_{\bar{u}} \end{bmatrix} \quad [2.58]$$

Ejemplo 1

A modo de comprobación, creamos con MATLAB un sistema de dimensión $\mathbb{R}^{3\times2}$ constituido por 3 estados, 2 entradas modificables y 2 salidas controlables. Esto implica la creación de las matrices A, B, C y D, las matrices Q y R, los límites $LB_{\bar{x}}$, $LB_{\bar{u}}$, $UB_{\bar{x}}$ y $UB_{\bar{u}}$ así como el vector de estados iniciales $\bar{x}^0$, siendo estos

$$A = \begin{bmatrix} 0.8 & 0.1 & 0 \\ 0.1 & 0.7 & 0 \\ 0 & 0 & 0.85 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.5 & 0.1 \\ 1 & 0 \\ 0.1 & 0.7 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$Q = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 50 \end{bmatrix}$$

$$R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

$$LB_{\bar{x}} = \begin{bmatrix} -2 \\ -2 \\ -2 \end{bmatrix}$$

$$LB_{\bar{u}} = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$$

$$UB_{\bar{x}} = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

$$UB_{\bar{u}} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\bar{x}^0 = \begin{bmatrix} 1.68 \\ 1.56 \\ 1.6 \end{bmatrix}$$

A continuación se aplica en bucle cerrado el Control Predictivo Basado en Modelo resolviéndolo mediante el algoritmo de optimización FISTA para un horizonte de predicción $N = 10$ y durante 20 intervalos de muestreo. En cada intervalo de muestreo se toma, de la solución z^* del problema, el estado x_1 que resultaría de aplicar la actuación u_0 al sistema estando el sistema en el estado x_0. Este estado x_1 será el nuevo estado x_0 en el siguiente intervalo de muestreo. Al representar el vector x_0 para cada intervalo de nuestreo se observa la convergencia de los estados del sistema a cero.

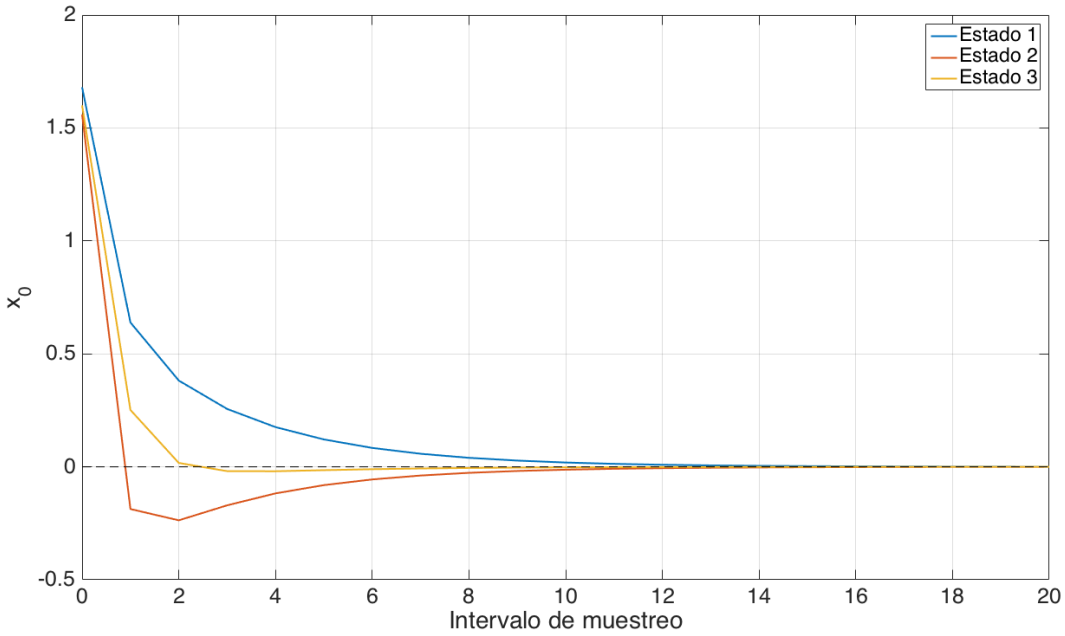

Figura 2-1. Control Predictivo Basado en Modelo que converge a cero.

2.4 Control Predictivo Basado en Modelo que converge a una referencia

Para un sistema lineal discreto invariante con el tiempo, la ley de control óptima para una convergencia en torno a una referencia r se puede expresar como

$$\min_u \sum_{k=0}^{N-1} (x_k - x_r)^T Q (x_k - x_r) + (u_k - u_r)^T R (u_k - u_r) \quad [2.59]$$

$$x_0 = x^0 \quad [2.60]$$

$$x_{k+1} = A x_k + B u_k \quad [2.61]$$

$$LB_x \leq x_k \leq UB_x \quad [2.62]$$

$$LB_u \leq u_k \leq UB_u \quad [2.63]$$

$$x_N = x_r \quad [2.64]$$

2.4.1 Correspondencia con el problema de optimización con restricciones de igualdad

De nuevo, este problema de minimización puede volver a expresarse como el problema de optimización con restricciones de igualdad desarrollado en el capítulo anterior.

$$\min_{z} \quad z^T H z + f^T z \tag{2.65}$$

$$Ez = b \tag{2.66}$$

$$LB_z \leq z \leq UB_z \tag{2.67}$$

Siendo $z \in \mathbb{R}^{N*(n+m)}$ de nuevo de la forma

$$z = \begin{bmatrix} x_0 \\ u_0 \\ \vdots \\ x_{N-1} \\ u_{N-1} \end{bmatrix} \tag{2.68}$$

Siendo $H \in \mathbb{R}^{N*(n+m) \times N*(n+m)}$, de la forma

$$H = \begin{bmatrix} Q & 0 & \cdots & 0 & 0 \\ 0 & R & & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & & Q & 0 \\ 0 & 0 & \cdots & 0 & R \end{bmatrix} \tag{2.69}$$

Siendo $f \in \mathbb{R}^{N*(n+m)}$, de la forma

$$f = \begin{bmatrix} -Qx_r \\ -Ru_r \\ \vdots \\ -Qx_r \\ -Ru_r \end{bmatrix} \tag{2.70}$$

Siendo $E \in \mathbb{R}^{n*(N+1) \times N*(n+m)}$, de la forma

$$E = \begin{bmatrix} I & 0 & 0 & 0 & 0 & & 0 & 0 \\ A & B & -I & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & A & B & -I & & 0 & 0 \\ & & \vdots & & & \ddots & \vdots & \\ 0 & 0 & 0 & 0 & 0 & \cdots & A & B \end{bmatrix} \tag{2.71}$$

Siendo $b \in \mathbb{R}^{n*(N+1)}$, de la forma

$$b = \begin{bmatrix} x^0 \\ 0 \\ \vdots \\ x_r \end{bmatrix} \tag{2.72}$$

Siendo $LB_z \in \mathbb{R}^{N*(n+m)}$, de la forma

$$LB_z = \begin{bmatrix} LB_x \\ LB_u \\ \vdots \\ LB_x \\ LB_u \end{bmatrix} \tag{2.73}$$

Siendo $UB_z \in \mathbb{R}^{N*(n+m)}$, de la forma

$$UB_z = \begin{bmatrix} UB_x \\ UB_u \\ \vdots \\ UB_x \\ UB_u \end{bmatrix} \quad [2.74]$$

Ejemplo 2

Aplicamos de nuevo sobre un sistema de dimensión $\mathbb{R}^{3\times2}$ creado con MATLAB el Control Predictivo Basado en Modelo en bucle cerrado resuelto mediante el algoritmo de optimización FISTA. En este caso, a las matrices A, B, C y D, las matrices Q y R, los límites LB_x, LB_u, UB_x y UB_u y el vector de estados iniciales x^0 deben añadirse ahora, los vectores x_r y u_r.

Las ecuaciones [2.20] y [2.21] pueden ser expresadas tal que

$$\begin{bmatrix} A-I & B \\ C & D \end{bmatrix} \begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix} \quad [2.75]$$

De manera que el cálculo de los vectores x_r y u_r se realizaría tal que

$$\begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} A-I & B \\ C & D \end{bmatrix} \backslash \begin{bmatrix} 0 \\ r \end{bmatrix} \quad [2.76]$$

Por lo tanto, las matrices y vectores genearados son

$$A = \begin{bmatrix} 0.8 & 0.1 & 0 \\ 0.1 & 0.7 & 0 \\ 0 & 0 & 0.85 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.5 & 0.1 \\ 1 & 0 \\ 0.1 & 0.7 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$Q = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 50 \end{bmatrix}$$

$$R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

$$LB_x = \begin{bmatrix} -20 \\ -20 \\ -20 \end{bmatrix}$$

$$LB_u = \begin{bmatrix} -30 \\ -30 \end{bmatrix}$$

$$UB_x = \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix}$$

$$UB_u = \begin{bmatrix} 30 \\ 30 \end{bmatrix}$$

$$x^0 = \begin{bmatrix} 1.68 \\ 1.56 \\ 1.6 \end{bmatrix}$$

$$r = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$x_r = \begin{bmatrix} 2 \\ 1 \\ 11.7333 \end{bmatrix}$$

$$u_r = \begin{bmatrix} 0.1 \\ 2.5 \end{bmatrix}$$

El procedimiento es similar al anterior. Para un horizonte de predicción $N = 10$ y durante 20 intervalos de muestreo, se toma, de la solución z^* del problema, el estado x_1 que resultaría de aplicar la actuación u_0 al sistema estando el sistema en el estado x_0. Este estado x_1 será el nuevo estado x_0 en el siguiente intervalo de muestreo. Al representar el vector x_0 para cada intervalo de nuestreo se observa la convergencia de los estados del sistema a cero.

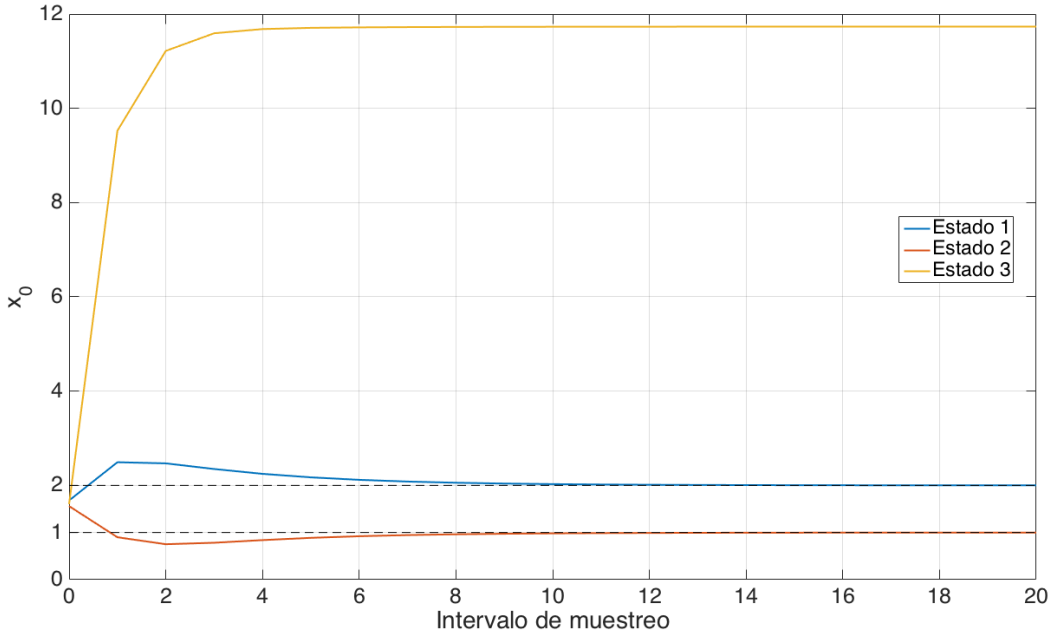

Figura 2-2. Control Predictivo Basado en Modelo que converge a una referencia.

Se observa cómo con 2 entradas modificables sólo se pueden fijar a voluntad 2 señales. Estas señales se corresponden en este ejemplo con 2 estados, no pudiendo ser fijados los 3. El número de grados de libertad de las referencias es igual al número de entradas.

Para fijar a voluntad los 3 estados es necesario modificar las matrices B, C y D, la matriz R, y el vector x_r, siendo estos

$$B = \begin{bmatrix} 0.5 & 0.1 & 0.25 \\ 1 & 0 & 0.5 \\ 0.1 & 0.7 & 0.35 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$R = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

$$r = \begin{bmatrix} 1 \\ 1.2 \\ 2 \end{bmatrix}$$

$$x_r = \begin{bmatrix} 1 \\ 1.2 \\ 2 \end{bmatrix}$$

Representando de nuevo el vector x_0 para cada intervalo de nuestreo se observa ahora la convergencia hacia la referencia de todos los estados.

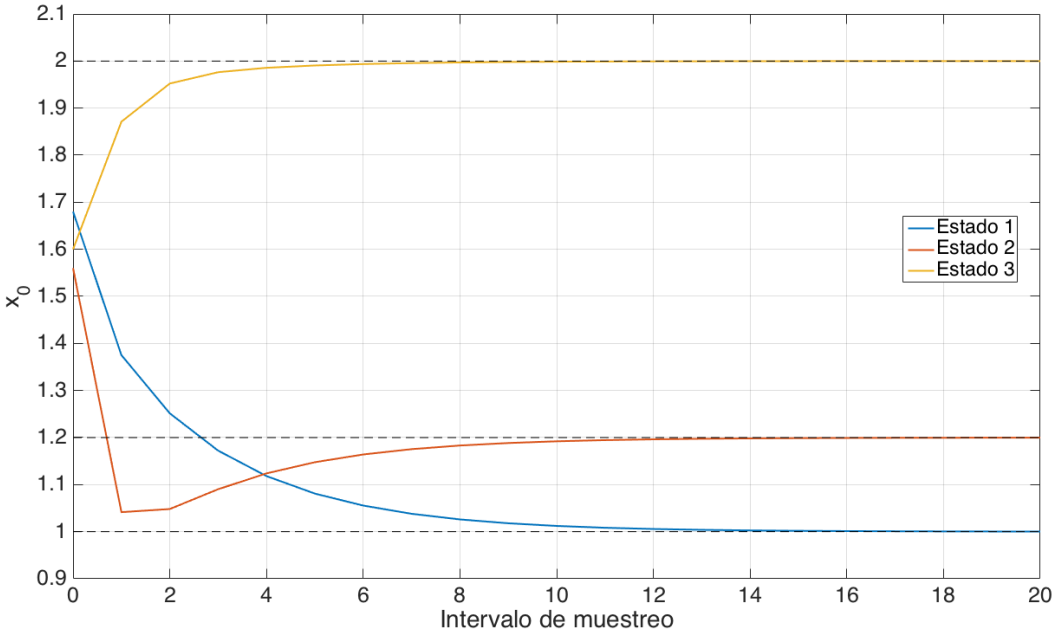

Figura 2-3. Control Predictivo Basado en Modelo que converge a una referencia para $p = m$.

2.5 Steady-State Target Optimization

Para una referencia deseada r, el estado x_r y la actuación u_r que satisface dicha convergencia calculados mediante la ecuación [2.76] pueden no estar dentro de los límites premitidos por el sistema. Aparece por tanto, el concepto de referencia alcanzable y_r definida tal que

$$y_r = Cx_r + Du_r \qquad [2.77]$$

estando x_r y u_r comprendidos dentro de los límites del sistema y pudiendo ser

$$y_r \neq r \qquad [2.78]$$

El cálculo de la referencia alcanzable a partir de la referencia deseada r se convierte ahora también en un problema de optimización con el fin de minimizar esta diferencia. Este nuevo problema de optimización se puede expresar como

$$\min_{x_r,u_r}(r - Cx_r - Du_r)^T Q_r(r - Cx_r - Du_r) \quad [2.79]$$

$$(A - I)x_r + Bu_r = 0 \quad [2.80]$$

$$LB_x \leq x_r \leq UB_x \quad [2.81]$$

$$LB_u \leq u_r \leq UB_u \quad [2.82]$$

Este optimizador de puntos de equilibrio es conocido con el nombre de Steady-State Target Optimization (SSTO). Si la matriz

$$\begin{bmatrix} A - I & B \\ C & D \end{bmatrix} \quad [2.83]$$

es de rango completo, existe una forma alternativa de obtener una referencia alcanzable, es decir, una solución factible al SSTO sin resolver el problema de optimización.

El cálculo de los vectores x_r y u_r optimizados se realiza mediante

$$\begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} L_x \\ L_u \end{bmatrix} r\lambda \quad [2.84]$$

siendo

$$\begin{bmatrix} L_x \\ L_u \end{bmatrix} = \begin{bmatrix} A - I & B \\ C & D \end{bmatrix} \backslash \begin{bmatrix} 0 \\ I \end{bmatrix} \quad [2.85]$$

Se deben cumplir por tanto las desigualdades

$$LB_x \leq L_x \lambda r \leq UB_x \quad [2.86]$$

$$LB_u \leq L_u \lambda r \leq UB_u \quad [2.87]$$

Si definimos el vector $c \in \mathbb{R}^{2*(n+m) \times m}$ como

$$c = \begin{bmatrix} L_x \\ -L_x \\ L_u \\ -L_u \end{bmatrix} \quad [2.88]$$

y el vector $d \in \mathbb{R}^{2*(n+m)}$ como

$$d = \begin{bmatrix} UB_x \\ -LB_x \\ UB_u \\ -LB_u \end{bmatrix} \quad [2.89]$$

las desigualdades [2.86] y [2.87] pasan a estar expresadas tal que

$$c\ \lambda r \leq d \quad [2.90]$$

Por tanto, se debe cumplir

$$c_i \lambda r \leq d_i \qquad \forall i \tag{2.91}$$

Para aquellos casos en los que esto no se cumpla, debemos calcular

$$\lambda_j \leq \frac{d_j}{c_j r} \tag{2.92}$$

de manera que

$$\lambda = \min_{j \in J_n} \left(\frac{d_j}{c_j r} \right) \tag{2.93}$$

siendo

$$J_n = \{i: \quad c_i r > d_i \quad \forall i\} \tag{2.94}$$

2.5.1 Filtro del SSTO

La factibilidad del problema de optimización puede perderse si la referencia r cambia bruscamente. Este cambio produce una variación en x_r y por tanto en la resctricción terminal x_N. Como paso previo al SSTO es conveniente utilizar un filtro para evitar estos cambios bruscos. Por lo tanto, la referencia de entrada al SSTO será una $r_{filtrada}$ calculada a partir de

$$r_{filtrada_{k+1}} = (1 - \rho) r_{filtrada_k} + \rho \hat{r} \tag{2.95}$$

siendo $\rho \in [0, 1]$ y $\hat{r}$ la referencia de entrada al filtro del SSTO.

2.5.2 Ejemplo de aplicación sobre la planta de los 4 tanques

Con el fin de comprobar el correcto funcionamiento del SSTO y el MPC realizaremos en MATLAB una simulación basada en el siguiente diagrama de bloques.

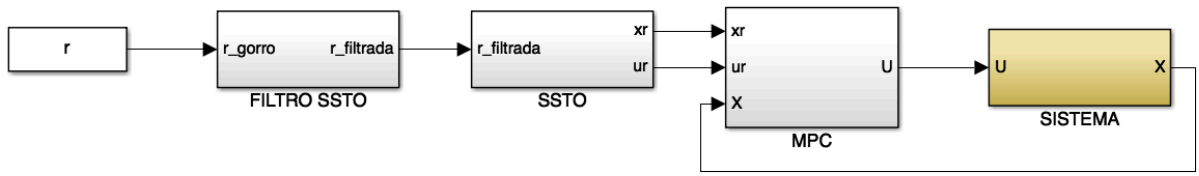

Figura 2-4. Diagrama de bloques SSTO-MPC

Un aspecto a tener en cuenta es que el MPC realiza los cálculos en variables incrementales, mientras que la salida controlable U calculada debe expresarla en valores absolutos y el estado X del sistema que toma como entrada también está expresado en valores absolutos. Este cambio de variable queda expresado tal que

$$U = u_0 + U_f \tag{2.96}$$

$$X = x_0 + X_f \tag{2.97}$$

siendo $X_f \in \mathbb{R}^n$ y $U_f \in \mathbb{R}^m$ el estado y la actuación correspondientes al punto de funcionamiento sobre el que se linealiza el sistema. De igual manera, la referencia de entrada r resulta de

$$r = Ref - X_f \tag{2.98}$$

siendo $Ref \in \mathbb{R}^p$ la referencia de entrada en valores absolutos.

Como sistema para realizar nuestra simulación utilizaremos el modelo lineal de la planta de los 4 tanques que se encuentra en el laboratorio del Departamento de Ingeniería de Sistemas y Automática. Esta planta se basa en el modelo de planta de 4 tanques diseñada por Karl Henrik Johansson.

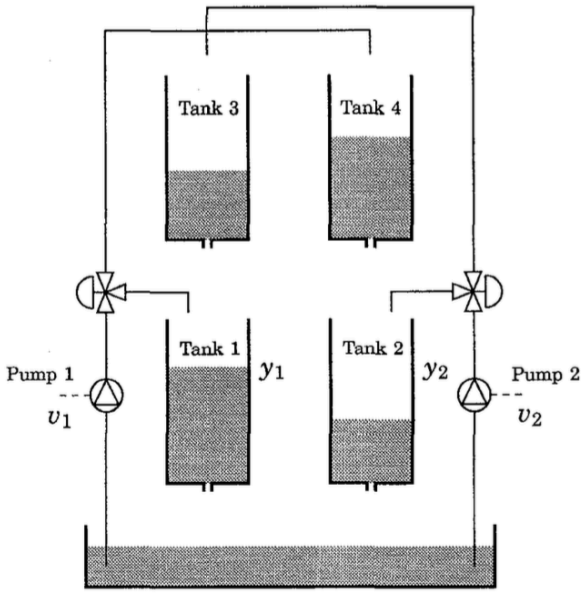

Figura 2-5. Planta de Johansson

La planta de los 4 tanques tiene como características $n = 4$, $m = 2$, y $p = 2$ por lo que al cumplirse $p = m$ permite el cálculo de x_r y u_r mediante el SSTO de la manera anteriormente explicada. Este modelo lineal será también el modelo del sistema que tenga implementado el MPC.

A continuación se muestran los puntos de funcionamiento de la planta de los 4 tanques. La linealización de la planta se hará a partir del primero de ellos y dicho estado de funcionamiento será el estado inicial desde el que comenzará la simulación.

Tabla 2-1. Puntos de funcionamiento de la planta de los 4 tanques

U_f	X_f
$\begin{bmatrix}1.7602\\1.8072\end{bmatrix}$	$\begin{bmatrix}0.5954\\0.6615\\0.5384\\0.7681\end{bmatrix}$
$\begin{bmatrix}1.9480\\2.0000\end{bmatrix}$	$\begin{bmatrix}0.7292\\0.8102\\0.6594\\0.9408\end{bmatrix}$
$\begin{bmatrix}1.4802\\1.5197\end{bmatrix}$	$\begin{bmatrix}0.4210\\0.4678\\0.3807\\0.5432\end{bmatrix}$

Al realizar la simulación en MATLAB con un cambio de referencia en el intervalo de muestreo número 100, se observa cómo las salidas controlables del sistema convergen a la referencia indicada en ambos casos. La primera referencia a alcanzar por el sistema es el segundo punto de funcionamiento indicado anteriormente. La segunda referencia no se corresponde con ningún punto de funcionamiento siendo ésta también alcanzable.

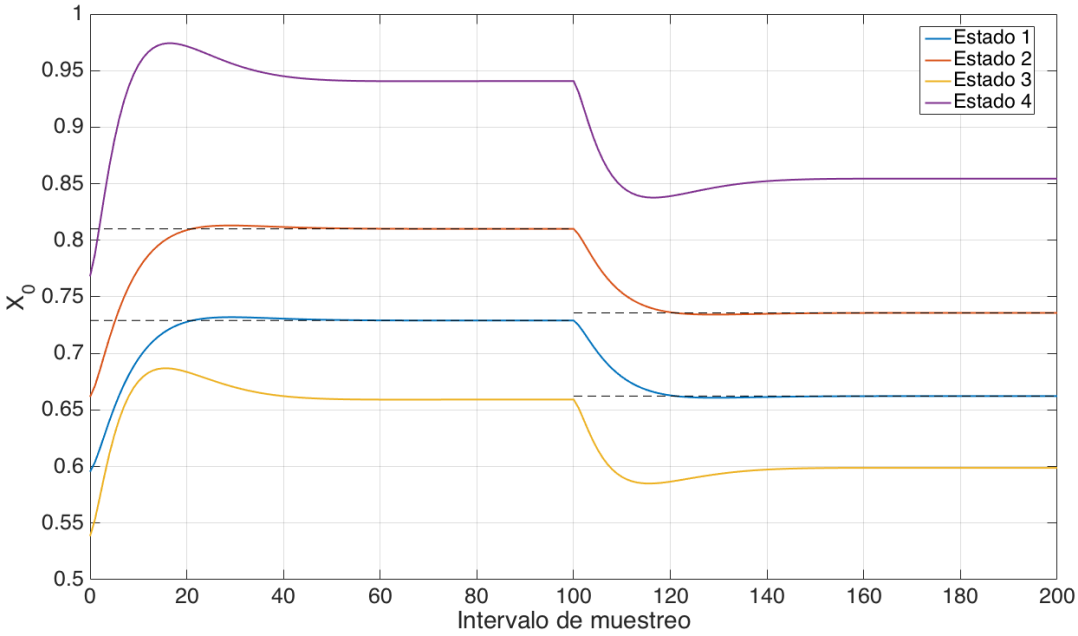

Figura 2-6. Simulación con el modelo lineal de la planta de los 4 tanques.

2.6 Observador de estado

En la mayor parte de las plantas las variables que forman el estado del sistema x no se pueden medir. Solo se dispone de la medida de ciertas variables de la planta. Por ello es necesario un observador de estado, que a partir de las medidas de la planta y la entrada u calculada por el MPC, estime el estado actual del sistema. La salida a la que el observador tiene acceso es de la forma

$$y_k = Cx_k + Du_k + d_k \qquad [2.99]$$

Siendo $d_k \in \mathbb{R}^p$ la perturbación en la salida en el instante k. Esta perturbación puede referirse tanto a una perturbación en la medida de la salida del sistema como a una perturbación real en la salida que no dependa del sistema.

$$\begin{bmatrix}\hat{x}_{k+1}\\ \hat{d}_{k+1}\end{bmatrix} = A_0\begin{bmatrix}\hat{x}_k\\ \hat{d}_k\end{bmatrix} + B_0u_k + L_0\left(y_k - C\hat{x}_k - Du_k - \hat{d}_k\right) \qquad [2.100]$$

Siendo $A_0 \in \mathbb{R}^{(n+m)\times(n+m)}$, de la forma

$$A_0 = \begin{bmatrix}A & 0\\ 0 & I\end{bmatrix} \qquad [2.101]$$

Siendo $B_0 \in \mathbb{R}^{(n+m)\times m}$, de la forma

$$B_0 = \begin{bmatrix}B\\ 0\end{bmatrix} \qquad [2.102]$$

Siendo $L_0 \in \mathbb{R}^{(n+m)\times m}$, de la forma

$$L_0 = \begin{bmatrix}L_{\hat{x}}\\ L_{\hat{d}}\end{bmatrix} \qquad [2.103]$$

y calculado tal que

$$L_0: |\lambda_i(A_0 - L_0 C_0)| < 1 \quad [2.104]$$

Siendo $C_0 \in \mathbb{R}^{m \times (n+m)}$, de la forma

$$C_0 = [C \quad I] \quad [2.105]$$

2.6.1 Filtro del observador de estado

Al igual que el SSTO, el observador tiene asociado un filtro con el fin de evitar cambios bruscos al producirse cambios en la perturbación del sistema. Por lo tanto, la perturbación medida por el observador estará definidapor una d_r calculada a partir de

$$d_{r_{k+1}} = (1 - \alpha) d_{r_k} + \alpha \hat{d} \quad [2.106]$$

siendo $\alpha \in [0, 1]$ y $\hat{d}_r$ la perturbación estimada.

2.6.2 Ejemplo de aplicación sobre la planta de los 4 tanques

Realizamos en MATLAB una simulación para comprobar el buen funcionamiento del MPC, el SSTO y el observador de estado conjuntamente según el siguiente diagrama de bloques.

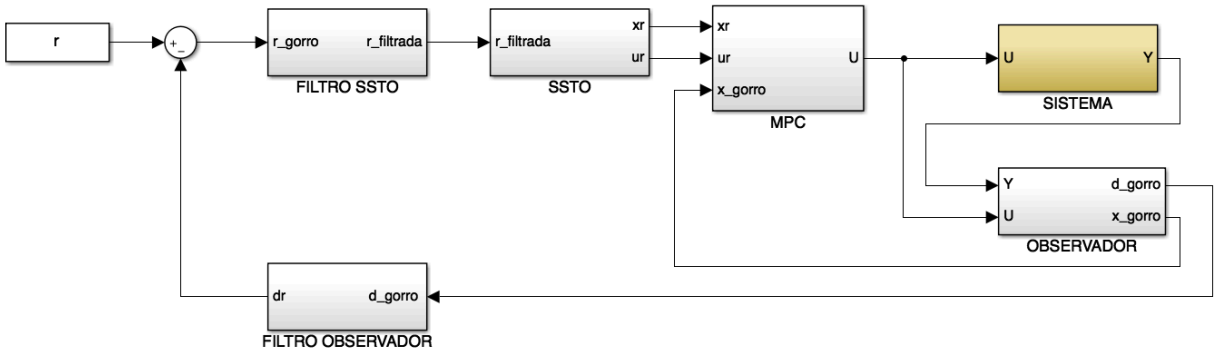

Figura 2-7. Diagrama de bloques SSTO-MPC-Observador

La perturbación estimada a la salida del sistema d_r debe ser realimentada hacia la referencia de manera que

$$\hat{r} = r - d_r \quad [2.107]$$

Para para poder generar una perturbación a la salida nuestro sistema va a ser el modelo no lineal de la planta de los 4 tanques de Johansson. El modelo de la planta que tiene el MPC es el linealizado en torno a un estado X_f. Esto hace que exista una diferencia entre el sistema que vamos a controlar y el modelo contenido en el MPC cuando el estado del sistema sea distinto al del punto de linealización X_f. Esta diferencia será interpretada por el observador como $\hat{d}$.

En la simulación realizada en MATLAB haremos un cambio de referencia en el intervalo de muestreo número 500. De igual manera, cada 200 intervalos de muestreo, añadiremos una perturbación aleatoria a la salida controlable del sistema.

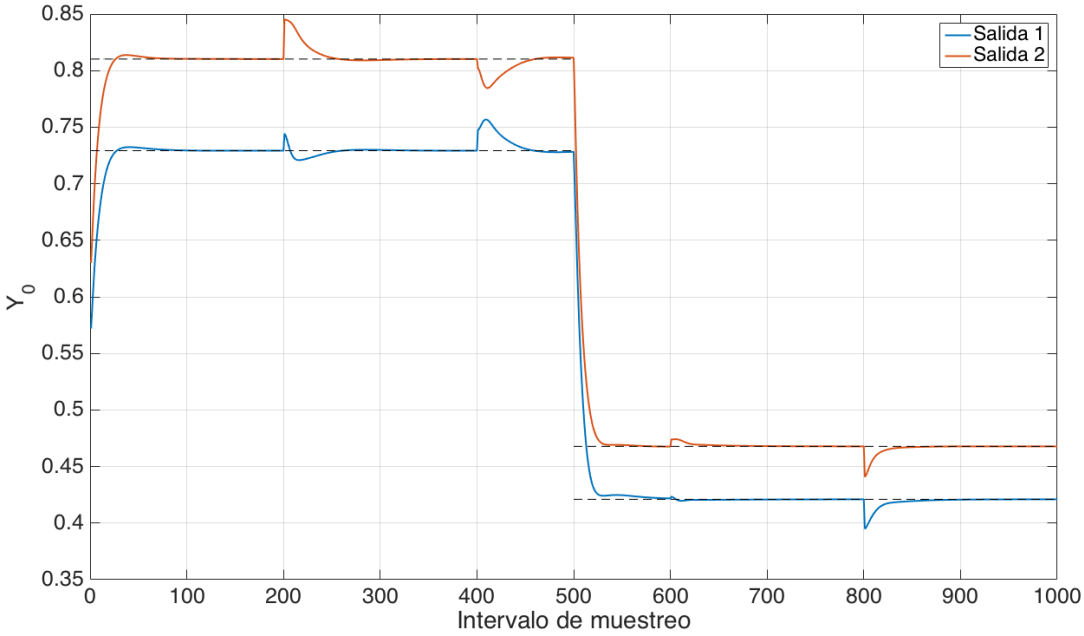

Figura 2-8. Simulación con el modelo no lineal de la planta de los 4 tanques

Se puede observar cómo el sistema converge a la referencia con exactitud a pesar de que el MPC está diseñado para un sistema distinto al que está controlando. Además, es capaz de llevar la salida controlable a la referencia aunque ésta tenga una perturbación adicional. Es interesante observar cómo el estado del sistema X y el estado del sistema estimado por el observador $\hat{X}$ no coinciden. Esto se debe a la perturbación que afecta a la salida controlable Y ya sea por una perturbación o por tratarse de un modelo no lineal. El observador estima la perturbación que afecta a la salida $\hat{d}$ y el estado en el que tendría que estar el sistema para contrarrestar esa perturbación estimada y encontrarse en la referencia deseada. La perturbación estimada a la salida del sistema es también restada a la referencia de entrada para conseguir llevar al sistema a un estado distinto en el que junto a la perturbación, alcance la referencia correctamente.

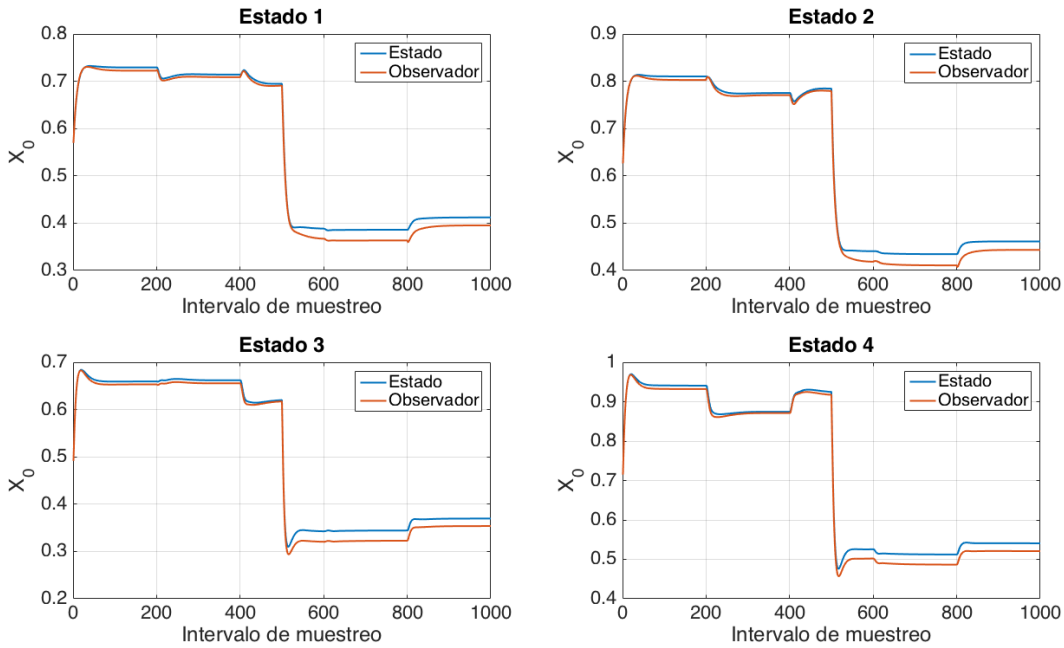

Figura 2-9. Estado real y observado para la simulación con el modelo no lineal

El diagrama de bloques de la simulación puede ser también expresado de una manera más compacta incluyendo dentro del bloque del MPC el observador, el SSTO y sus correspondientes filtros. Además se

incluyen los cambios de variables absolutas a variables incrementales y viceversa para el correcto funcionamiento conjunto del MPC y el sistema.

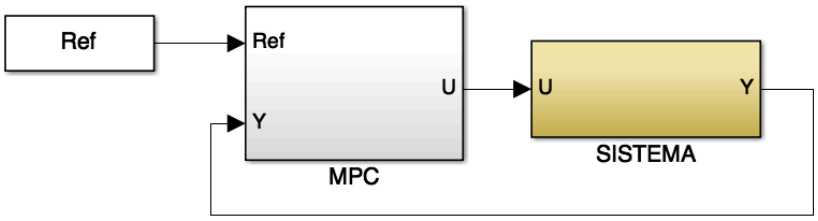

Figura 2-10. Diagrama de bloques MPC

3 IMPLEMENTACIÓN EN LABVIEW

LabVIEW es una plataforma para el diseño de sistemas especializada en adquisición de datos, instrumentación y control, y automatización industrial. El lenguaje de programación usado por LabVIEW es visual y se basa en el fujo de datos. La ejecución del mismo se determina a través de la estructura de los diagramas de bloques.

3.1 Programación del Control Predictivo Basado en Modelo

Para implementar el Control Predictivo Basado en Modelo en LabVIEW utilizaremos una DLL. Se trata de una biblioteca de enlace dinámico que tiene como función almacenar código que puede ser ejecutado. En esta biblioteca estará incluida nuestra función *MPC,* programada en lenguaje C, que contiene al controlador predictivo.

Por comodidad, a través de un *script* de MATLAB se genera un archivo de texto *.txt* en el que se definen todas las variables y constantes necesarias para la cabecera de la función *MPC*. De manera que al cambiar los parámetros modificables del Control Predictivo Basado en Modelo en el *script* de MATLAB, se genere automáticamente la cabecera correcta de la función para los parámetros indicados.

Además dicho *script* se encargará también de realizar cálculos previos con los parámetros modificables del controlador de manera que la función *MPC* solo tenga que realizar aquellos cálculos que dependan exclusivamente de las entradas de dicha función. De esta manera se evita que la función repita ciertos cálculos que siempre dan el mismo resultado cada vez que se ejecute dicha función. Del mismo modo, para hacer más rápida la ejecución de la función, algunas matrices y vectores que sigan un patrón de repetición o contengan una gran cantidad de ceros será definidas para que tengan una dimensión menor.

Las matrices Q y R serán redefinidas como los vectores $Q \in \mathbb{R}^n$ y $R \in \mathbb{R}^m$.

La matriz E pasa a estar definida como $E \in \mathbb{R}^{n \times n+m}$ tal que

$$E = [A \quad B \quad -I] \tag{3.1}$$

Para usar la matriz E de manera traspuesta se definen dos matrices adicionales $ET1 \in \mathbb{R}^{n+m \times 2*n}$ tal que

$$ET1 = \begin{bmatrix} I & A \\ 0 & B \end{bmatrix} \tag{3.2}$$

y $ET2 \in \mathbb{R}^{n+m \times 2*n}$ definida como

$$ET2 = \begin{bmatrix} -I & A \\ 0 & B \end{bmatrix} \quad [3.3]$$

El vector f basta con estar definido como $f \in \mathbb{R}^{n+m}$

$$f = \begin{bmatrix} -Qx_r \\ -Ru_r \end{bmatrix} \quad [3.4]$$

Los vectores UB_x, LB_x, UB_u y LB_u pasan a estar definidos por los vectores $edge$ y $newz$ que representan un cálculo intermedo del Control Predictivo Basado en Modelo que se repite para cada ejecución del mismo. Siendo $edge \in \mathbb{R}^{n+m}$ definido tal que

$$edge = \begin{bmatrix} \frac{UB_x - LB_x}{2} \\ \frac{UB_u - LB_u}{2} \end{bmatrix} \quad [3.5]$$

y $newz \in \mathbb{R}^{n+m}$ como

$$newz = \begin{bmatrix} \frac{UB_x + LB_x}{2} \\ \frac{UB_u + LB_u}{2} \end{bmatrix} \quad [3.6]$$

3.2 Aspectos a tener en cuenta antes de la implementación

La principal diferencia entre las simulaciones que se van a realizar en LabVIEW y las programadas hasta ahora en MATLAB es que éstas son secuenciales mientras que en LabVIEW las tareas de control y simulación o muestreo de la planta se ejecutan en paralelo de acuerdo a la precedencia impuesta por LabVIEW. Esto hace que no se puedan comparar directamente dos simulaciones iguales realizadas en MATLAB y en LabVIEW. Además, es interesante hacer un estudio previo sobre el muestreo en tiempo real que se va a realizar en LabVIEW.

3.2.1 Simulación secuencial previa

En primer lugar se va a comparar la correcta implementación en lenguaje C de la función *MPC*. Para ello, se realiza una simulación secuencial en LabVIEW de manera que esta simulación sea similar a una simulación programada en MATLAB.

Dentro de una estructura *While Loop* que se repite cada 500 milisegundos se añade una estructura *Flat Sequence* en la que se simularán de forma secuencial la planta de los 4 tanques y el controlador predictivo. En el primer subdiagrama de esta estructura se incluye un bloque *Call Library Function* que cargará la función

$$PLANTA(U, X, Y)$$

contenida en una DLL. Esta función simulará el modelo no lineal de la planta de los 4 tanques para un tiempo de muestreo. A la salida de la misma, se tomarán como datos la salida controlable Y del sistema.

En el segundo subdiagrama de la estructura *Flat Sequence* se incluye otro bloque *Call Library Function* que llama a la función

$$MPC(U, Y, Ref)$$

contenida en la misma DLL anterior. Para esta simulación previa, el estado estimado por el observador $\hat{X}$, la perturbación estimada por el observador $\hat{d}$, la referencia filtrada $r_{filtrada}$ y la perturbación estimada filtrada d_r serán definidos como *stactic* dentro de la función y no serán accesibles.

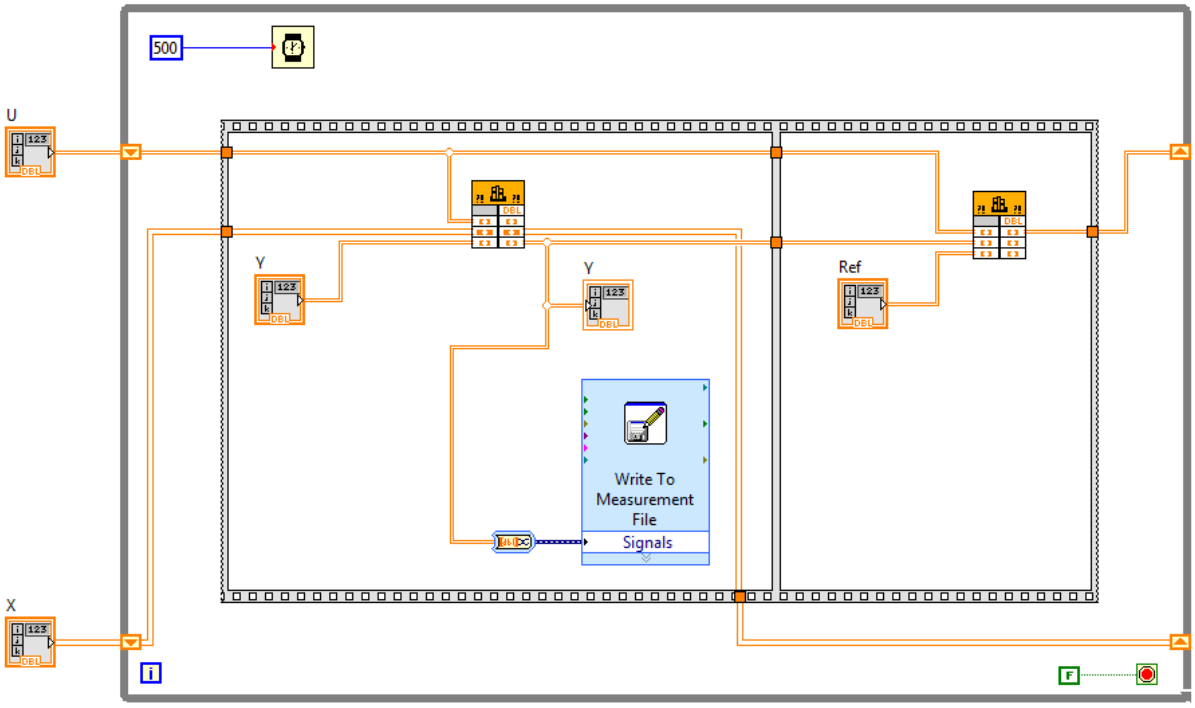

Figura 3-1. *Block Diagram* de la simulación secuencial previa en LabVIEW

Como se puede observar a simple vista, los resultados de ambas simulaciones coindicen notablemente para ambas salidas.

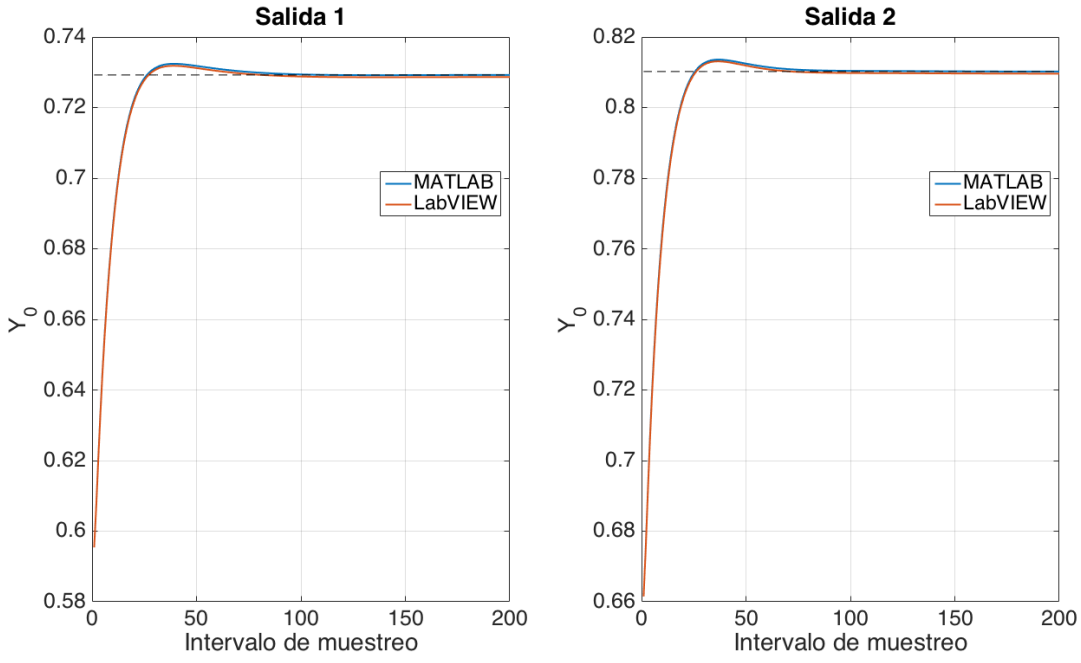

Figura 3-2. Comparativa de las simulaciones secuenciales MATLAB – LabVIEW

Para una mejor comparativa, se muestra la diferencia en valor absoluto entre los resultados obtenidos de la simulación programada en MATLAB y los resultados obtenidos de la simulación secuencial en LabVIEW.

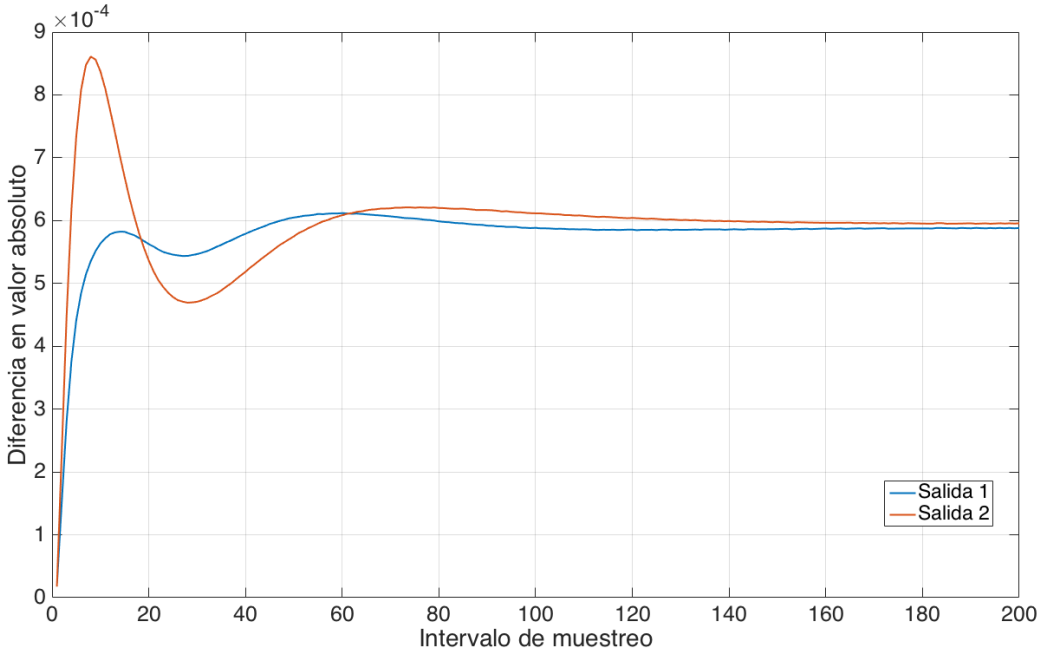

Figura 3-3. Diferencia entre las simulaciones secuenciales MATLAB – LabVIEW

La pequeña diferencia que se observa se debe a la precisión con la que realizan los cálculos MATLAB y LabVIEW. Mientras que en MATLAB las variables necesarias para el MPC son deducidas unas a partir de otras y acarreando todos los decimales, en LabVIEW todas las variables han sido definidas en la cabecera de la función con un máximo de 4 cifras decimales.

3.2.2 Muestreo en paralelo

Antes de implementar por completo el Control Predictivo Basado en Modelo en LabVIEW es conveniente hacer un estudio sobre el muestreo de la planta de los 4 tanques en paralelo. Para ello, se simula únicamente el modelo no lineal de la planta de los 4 tanques de la siguiente manera:

- Cada 100 ms se ejecuta la función $PLANTA(U, X, Y)$ que ejecuta el modelo no lineal de la planta una vez y se toma como dato la lectura de la salida controlable Y.
- Cada 300 ms se incrementa la entrada modificable U en 0.01.
- Cada 5000 ms se toma como dato muestreado la lectura de la salida controlable Y.

Para esta simulación programada en MATLAB y en LabVIEW, el muestreo se realiza idealmente en los instantes de tiempo

$$[1 \quad 51 \quad 101 \quad 151 \quad 201 \quad \cdots]$$

Es decir, se muestrea la planta de los 4 tanques una vez cada 50 veces que se ejecuta la función que contiene su modelo no lineal.

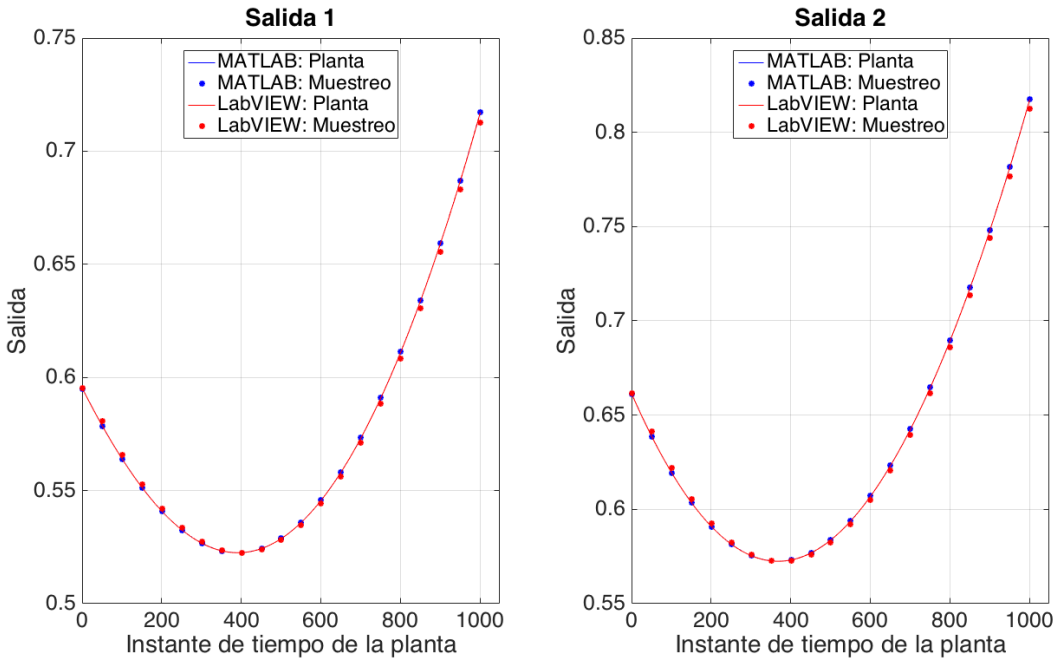

Figura 3-4. Muestreo ideal de la planta de los 4 tanques

Si observamos el muestreo realizado en MATLAB, coincide con la lectura de la planta para esos instantes de tiempo. Sin embargo, para el muestreo realizado en LabVIEW, los valores muestreados no coindicen con los instantes de tiempo indicados.

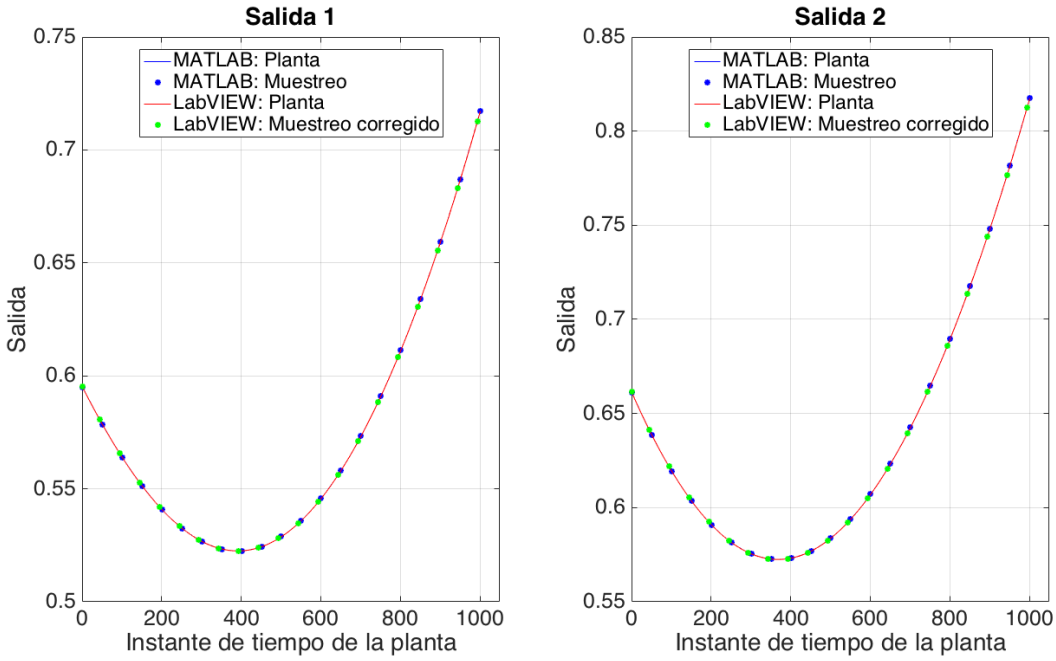

Figura 3-5. Muestreo real de la planta de los 4 tanques

Tras un análisis exhaustivo sobre los datos muestreados en LabVIEW, se observa que los valores muestreados de la planta coinciden con la planta simulada en los instantes de tiempo

$$[1 \quad 44 \quad 94 \quad 144 \quad 194 \quad \cdots]$$

Se deduce por tanto que en paralelo no se realiza correctamente el muestreo de la planta durante los primeros instantes de tiempo, pero sin embargo, tras varios muestreos incorrectos, sí realiza correctamente el muestreo cada 50 veces que se ejecuta el modelo no lineal de la planta.

Como solución a este problema, se propone hacer que la simulación de la planta de los 4 tanques parta siempre

de un punto de funcionamiento manteniendo la referencia en dicho punto. En ese caso la toma de datos en los primeros instantes de muestreo se hará de forma correcta ya que mientras se mantenga la referencia, el valor de la salida no varía al tratarse de un punto de funcionamiento de la planta.

3.3 Control Predictivo de la planta de los 4 tanques

Para la simulación completa del Control Predictivo Basado en Modelo en LabVIEW, se implementará el controlador dentro de una estructura *While Loop*. Este bucle se ejecutará cada 5000 milisegundos siendo este el tiempo de muestreo del controlador. Para hacer que la simulación sea en tiempo real para el tiempo de muestreo indicado se incluye la estructura *Wait Until Next ms Multiple* dentro del bucle *while*. Esto permitirá que se ejecute el bucle en tiempo real para cada tiempo de muestreo independiente de la duración de la ejecución del contenido del bucle.

La función *MPC* contenida en el archivo *.dll* será llamada durante la simulación mediante el bloque *Call Library Function Node*. En este bloque es necesario configurar la ruta donde se encuentra la DLL, el nombre de la función y el prototipo de la función, siendo ésta

$$MPC(U, Y, Ref, Obs, rfiltrada, dgorro, dr)$$

Siendo el argumento U la actuación calculada por el controlador, el argumento Y la salida muestreada de la planta, Ref la referencia alcanzable, Obs el estado estimado por el observador, $rfiltrada$ la referencia filtrada, $dgorro$ la perturbación estimada por el observador y dr la perturbación estimada filtrada. Estos últimos 4 argumentos deben ser inicializados junto a la entrada U inicial. El tipo de datos de todos ellos serán *array data pointer* con una resolución *double* de 8 *byte*.

La entrada modificable U y salida controlable Y de la planta utilizadas en la función *MPC* se definirán mediante estructuras *Global Variable*. Esta estructura hace que dichas variables sean de tipo global. La referencia por su parte, estará contenida dentro del bucle *while* de manera que podrá ser modificada en línea durante el transcurso de la simulación.

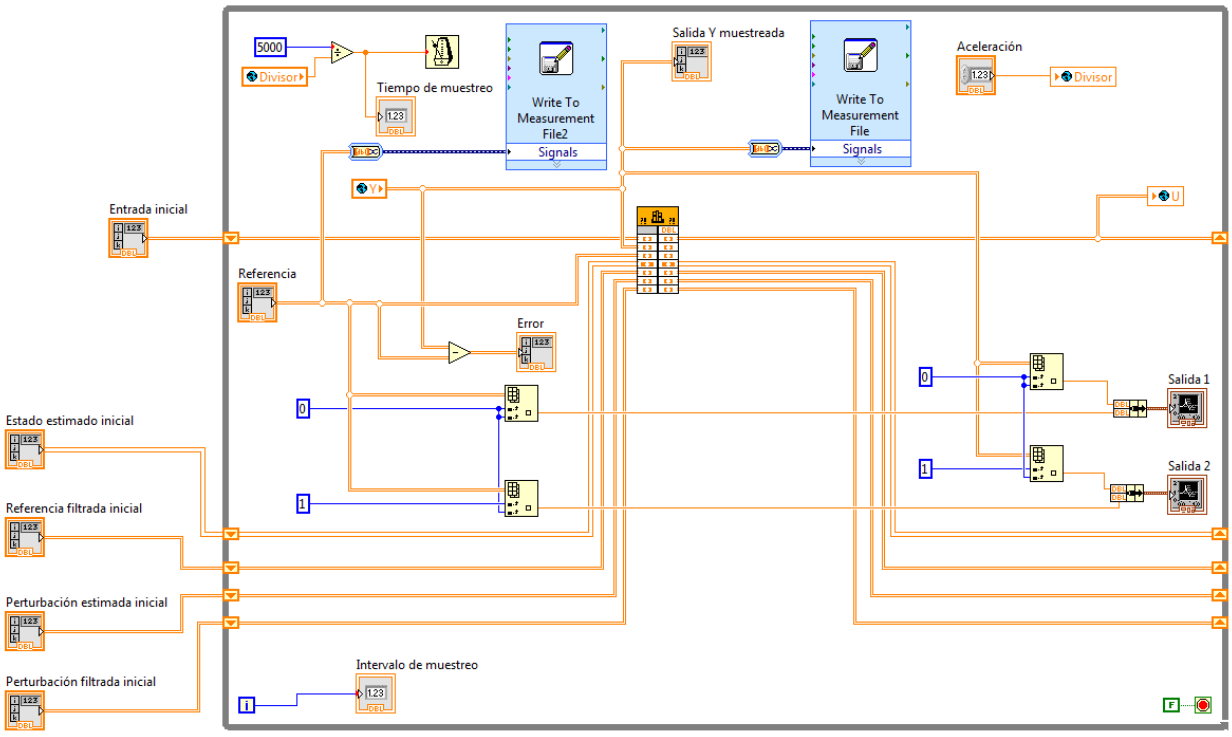

Figura 3-6. *Block Diagram* del Control Predictivo Basado en Modelo

Para simular la planta de los 4 tanques se llama a través de un bloque *Call Library Function Node* a la función *PLANTA* dentro de otra estructura *While Loop*. Esta estructura se ejecutará cada 100 milisegundos siendo éste el tiempo de integración de la planta. La función está contenida en la misma DLL que el controlador predictivo siendo ésta de la forma

$$PLANTA(U, X, Y)$$

Siendo el argumento U la actuación calculada por el controlador y el argumento Y la salida muestreada de la planta. El estado real de la planta X no será en ningún momento accesible por parte del controlador debiendo éste ser inicializado correctamente. El tipo de datos de todos los argumentos serán de nuevo de tipo *array data pointer* con una resolución *double* de 8 *byte*. La actuación calculada por el MPC y la salida de la planta serán de nuevo definidas dentro de una estructura *Global Variable* que convertirá a dichas variables en globales.

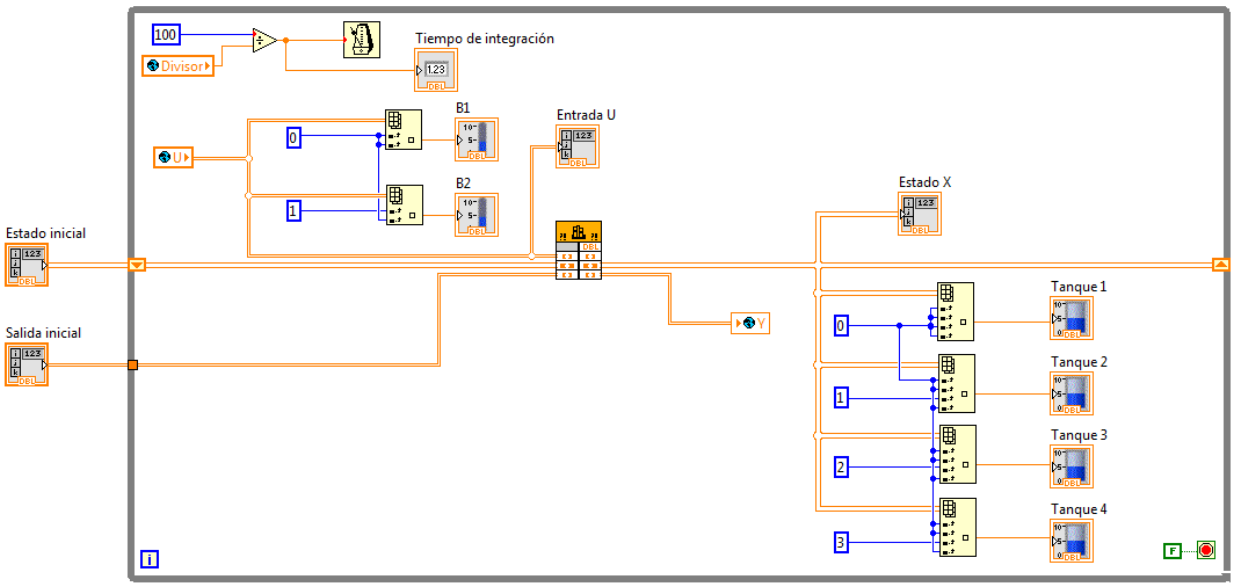

Figura 3-7. *Block Diagram* del modelo no lineal de la planta de los 4 tanques

Esta simulación programada en LabVIEW ejecutará por tanto una simulación de la planta de los 4 tanques con un controlador predictivo en tiempo real. Mientras que el modelo no lineal de la planta se ejecuta cada 100 milisegundos, el controlador muestreará la salida de la planta cada 5000 milisegundos y calculará la entrada U correspondiente para alcanzar la referencia *Ref*.

Esta simulación en LabVIEW presenta una ventaja ya que si el tiempo de integración de la planta y el tiempo de muestreo del controlador se eligen proporcionales a 100 ms y 5000 ms respectivamente los resultados de la simulación no varían. Es decir, si elegimos como tiempo de integración 20 ms y como tiempo de muestreo 1000 ms obtendremos los mismos datos que para la simulación original solo que la ejecución de la misma tarda 5 veces menos.

Durante el transcurso de la simulación, esta aceleración de la misma será configurable a través de una variable global. El tiempo de muestreo y el tiempo de integración de la planta se dividen entre esta variable global de manera que esta variable indica cuantas veces es más rápida la simulación con respecto a la simulación original.

En el *Front Panel* de LabVIEW son configurables antes del inicio de la simulación el estado inicial del sistema, la entrada inicial y siendo la salida inicial una entrada necesaria para el correcto funcionamiento de la DLL pero que no produce ninguna alteración en el resultado de la simulación. El estado estimado incicial del observador, la referencia filtrada inicial, la perturbación estimada inicial y la perturbación filtrada inicial son también configurables para su inicialización. La referencia por su parte, será modificable en cualquier momento una vez que la simulación esté en marcha.

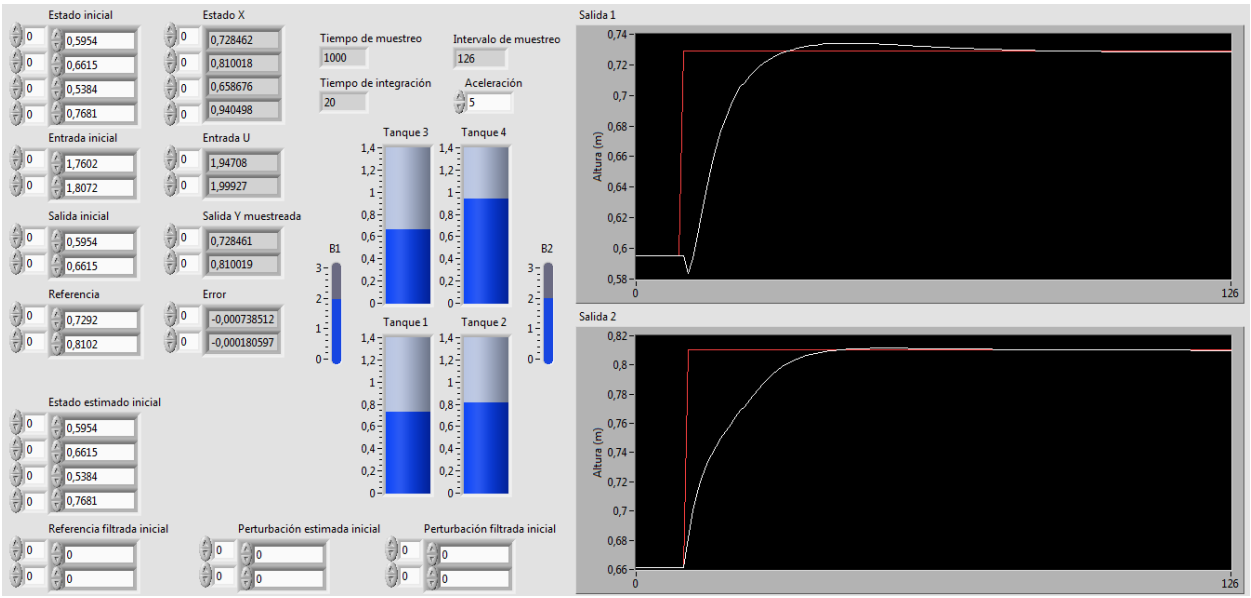

Figura 3-8. *Front Panel*

Se muestran a continuación los resultados de una simulación con 3 cambios de refencia partiendo de un punto de funcionamiento para inicializar correctamente el muestreo de la planta.

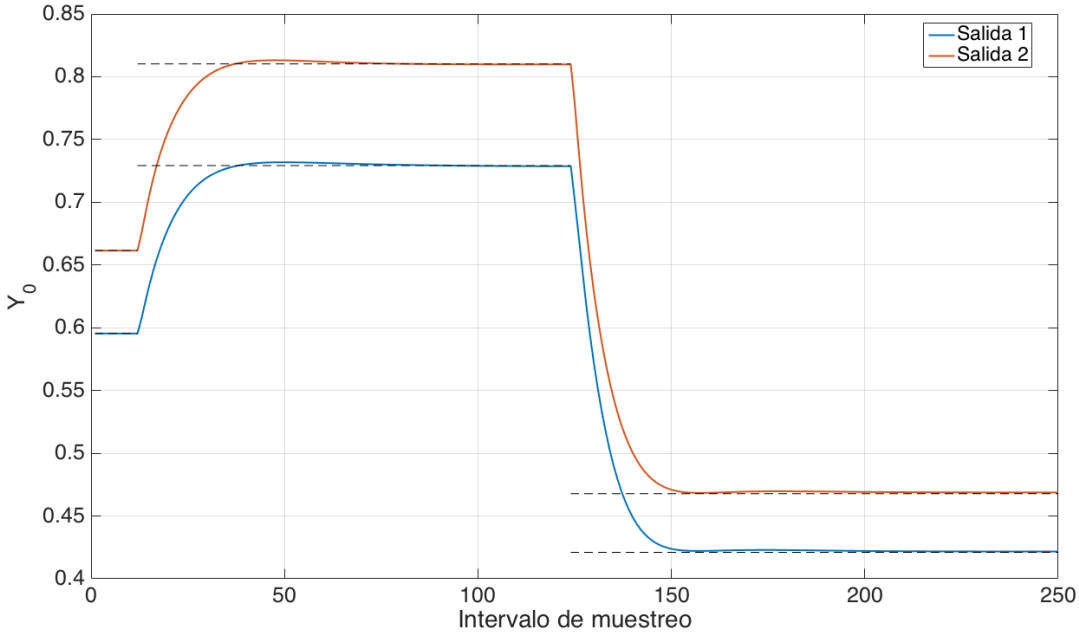

Figura 3-9. Simulación MPC – Modelo no lineal de la planta de los 4 tanques

REFERENCIAS Y BIBLIOGRAFÍA

[1] Alvarado, I., «Model Predictive Control for Tracking Constrained Linear Systems», 2007.

[2] Johansson, K.H., «The Quadruple Tank Process: A Multivariable Laboratory Process with an Adjustable Zero», 2000.

[3] Limón, D., «Ejercicios de introducción a la optimización», 2015.

[4] Rawlings, J.B., Mayne, D.Q., «Model Predictive Control: Theory and Desing», 2009.

GLOSARIO

DLL: Dynamic-Link Library 29

FISTA: Fast Iterative Shrinkage-Thresholding Algorithm 6

ISTA: Iterative Shrinkage-Thresholding Algorithm 4

LabVIEW: Laboratory Virtual Instrumentation Engineering Workbench 29

MATLAB: MATrix LABoratory 4

MPC: Model Predictive Control 14

SSTO: Steady-State Target Optimization 22

ANEXO A: SCRIPTS DE MATLAB

A1. Comparativa de tiempos de convergencia con matrices no diagonales

```
clc; clear; close all;
%% Datos iniciales
n = 5;
TOL = 1e-8;
error = 1e-4;
repeticiones = 25;

time = zeros(repeticiones,3);
k=1;
while k <= repeticiones
    %% Generación del problema
    T = rand(n);
    H = T'*T + 2*eye(n);
    f = 2*rand(n,1) - ones(n,1);
    LB = -rand(n,1);
    UB = 2*rand(n,1);

    edge = (UB - LB)./2;
    newz = (UB + LB)./2;
    %% quadprog(H,f,A,b,Aeq,beq,LB,UB,X0,OPTIONS)
    tic
    sol1 = quadprog(H,f,[],[],[],[],LB,UB);
    time(k,1) = toc;

    %% ALGORITMO ISTA
    landa = max(eig(H));
    H2 = eye(n)*landa;
    zk = zeros(n,1);
    incremento = Inf;

    tic
    while abs(incremento) > TOL
        f2 = (f + H*zk)';
        % Problema 1
        z = -f2/H2;
        zaux = min(abs(z' - newz),edge).*sign(z' - newz);
        incremento = zaux + newz;
        %%%%%%%%%%%%%
        zk = zk + incremento;
    end
    time(k,2) = toc;
    %% ALGORITMO FISTA
    z0 = zeros(n,1);
    y1 = zeros(n,1);
    incrementoz = Inf;
    t1 = 1;

    tic
    while incrementoz > TOL
        f2 = (f + H*y1)';
        % Problema 1
        z = -f2/H2;
        zaux = min(abs(z' - newz),edge).*sign(z' - newz);
        incrementoy = zaux + newz;
        %%%%%%%%%%%%%
        z1 = y1 + incrementoy;
        t2 = 0.5*(1 + sqrt(1 + 4*(t1^2)));
```

```
        y1 = z1 + (t1 - 1)/(t2)*(z1 - z0);
        incrementoz = abs(z1 - z0);
        z0 = z1;
        t1 = t2;
    end
    time(k,3) = toc;

    if abs(z0 - sol1) < error & abs(zk - sol1) < error
        k = k + 1;
    end
end
%% Resultados gráficos
text = {'ISTA','FISTA'};
for k=1:2
    figure(k)
    hold on
    hf = subplot(1,1,1);
    bar(time(1:repeticiones,1),'b');
    bar(time(1:repeticiones,k+1),'r');
    legend('QUADPROG',text{k},'Location','best');
    xlabel('Número de problema aleatorio');
    ylabel('Tiempo de convergencia (s)','Rotation',90);
    hf.XLim = [0 repeticiones+1];
    hf.FontSize = 25;
    hf.Box = 'on';
    grid on
end
```

A2. Comparativa de tiempos de convergencia QUADPROG – FISTA con restricciones de igualdad

```
clc; clear; close all;
%% Datos iniciales
n = 5;
TOL = 1e-5;
error = 1e-4;
repeticiones = 25;

time = zeros(repeticiones,2);
k=1;
while k <= repeticiones
    %% Generación del problema
    D = diag(5*rand(n,1));
    A = rand(n-1,n);
    b = zeros(n-1,1);
    f = 2*rand(n,1) - ones(n,1);
    LB = -rand(n,1);
    UB = 2*rand(n,1);

    edge = (UB - LB)./2;
    newz = (UB + LB)./2;
    %% quadprog(H,f,A,b,Aeq,beq,LB,UB,X0,OPTIONS)
    tic
    z0 = quadprog(D,f,[],[],A,b,LB,UB);
    time(k,1) = toc;

    %% ALGORITMO FISTA
    x0 = zeros(n-1,1);
    y1 = zeros(n-1,1);
    t1 = 1;
    fin = 0;
    Q = inv(A*inv(D)*A');
```

```
    tic
    while fin==0
        f2 = (f - A'*y1)';
        % Problema 1
        z = -f2/D;
        zaux = min(abs(z' - newz),edge).*sign(z' - newz);
        zyk = zaux + newz;
        %%%%%%%%%%%%
        if abs(A*zyk - b) < TOL
            fin = 1;
        else
            incrementoy = -Q*(A*zyk - b);
            x1 = y1 + incrementoy;
            t2 = 0.5*(1 + sqrt(1 + 4*(t1^2)));
            y1 = x1 + (t1 - 1)/(t2)*(x1 - x0);

            x0 = x1;
            t1 = t2;
        end
    end
    time(k,2) = toc;

    if abs(z0 - zyk) < error
        k = k + 1;
    end
end
%% Figura QUADPROG - FISTA
figure(1)
hold on
hf = subplot(1,1,1);
bar(time(1:repeticiones,1),'b');
bar(time(1:repeticiones,2),'r');
legend('QUADPROG','FISTA','Location','best');
xlabel('Número de problema aleatorio');
ylabel('Tiempo de convergencia (s)','Rotation',90);
hf.XLim = [0 repeticiones+1];
hf.FontSize = 25;
hf.Box = 'on';
grid on
```

A3. Simulación del modelo no lineal de la planta de los 4 tanques

```
clear; clc; close all;
% Puntos de funcionamiento
Uf1 = [1.7602; 1.8072];
Uf2 = [1.948; 2];
Uf3 = [1.4802; 1.5197];
Xf1 = puntoFunc(Uf1);
Xf2 = puntoFunc(Uf2);
Xf3 = puntoFunc(Uf3);
%% Parámetros modificables
n = 4;
m = 2;
% Tiempo de muestreo y tiempo de integración
Tm = 5;
h = 0.1;
% Punto de linealización
Uf = Uf1;
Xf = Xf1;
% Referencia alcanzable
Ref1 = Xf2;
Ref2 = Xf3;
```

```
% Estado inicial
X = Xf;
% Entrada inicial
U = Uf;
% Observador y perturbación inicial
observador = X - Xf;
perturb = zeros(m,1);
dgorro = zeros(m,1);
dr = zeros(m,1);
rfiltrada = zeros(m,1);
% Parámetros de los filtros
rho = 0.8;
alpha = 0.8;
% Constantes modificables para el MPC
N = 20;
TOL = 1e-7;
repeticionesmax = 2000;
Q = 1*eye(n);
R = 1e-4*eye(m);
Q0 = eye(n+m);
R0 = 100*eye(m);

%% Parametización del modelo
[A,B,C,D] = modelo4tanques(Xf,5);
r = Ref1(1:m) - Xf(1:m);
r2 = Ref2(1:m) - Xf(1:m);
% Cálculo de los límites
LBx = 0.2 - Xf;
UBx = 1.2 - Xf;
LBu = 0 - Uf;
UBu = 3 - Uf;
% Cálculo de los parámetros del observador
A0 = eye(n+m);
A0(1:n,1:n) = A;
B0 = [C,eye(m)];
Lxd = dlqr(A0',B0',Q0,R0)';
%% Generar parámetros para MPC
% Generar límites
LB = zeros(N*(m+n),1);
UB = zeros(N*(m+n),1);
H = zeros(N*(m+n));
k=0;
while k <= (N-1)*(m+n)
    for i=1:n
        LB(k+i,1) = LBx(i,1);
        UB(k+i,1) = UBx(i,1);
        H(k+i,k+i) = Q(i,i);
    end
    for j=1:m
        LB(k+i+j,1) = LBu(j,1);
        UB(k+i+j,1) = UBu(j,1);
        H(k+i+j,k+i+j) = R(j,j);
    end
        k = k + i + j;
end
% Generar fronteras
edge = (UB-LB)./2;
newz = (UB+LB)./2;
% Generar matriz E
E = kron(eye(N),[A B]);
j=0;
for i=1:n:n*N-n
    j = j+1;
```

```
    E(i:i+n-1,((j-1)*(n+m)+(m+n+1)):((j-1)*(n+m)+(n+m+1)+n-1)) = -eye(n);
end
E = [eye(n) zeros(n,n*N-n+m*N); E];
% Generar vector b
b = zeros(n*(N+1),1);
% Generar Qfista
Qfista = inv(E*inv(H)*E');
%% Parámetros para el SSTO
L = [A-eye(n) B;C D]\[zeros(n,m);eye(m)];
c = [L;-L];
d = [UBx;UBu;-LBx;-LBu];

%% Inicialización del sistema NO LINEAL
X = simula_NL_4Tanques(X,U,Tm,h)';
Y = C*X;
x = X - Xf;
y = C*x;
u = U - Uf;

estado(1,:) = X;
entrada(1,:) = U;
salida(1,:) = Y;

%% Bucle
for i=1:1000
    %% Observador
    observador2 = A*observador + B*u + Lxd(1:4,:)*(y - C*observador -
dgorro);
    dgorro = dgorro + Lxd(5:6,:)*(y - C*observador - dgorro);

    observador = observador2;

    % Filtro del observador
    dr = (1 - alpha)*dr + alpha*dgorro;

    %% Actualización de la referencia
    if i==500
        r = r2;
    end
    rk = r - dr;

    %% Filtro del SSTO
    rfiltrada = (1 - rho)*rfiltrada + rho*rk;

    % SSTO
    indice = c*rfiltrada > d;
    res = d./(c*rfiltrada);
    landa = min(res(indice));
    if size(landa)==[0 1]
        landa = 1;
    end
    resul = L*rfiltrada*landa;
    xr = resul(1:n);
    ur = resul(n+1:n+m);

    %% MPC
    % Generar vector f
    f=[];
    for k=1:N
        f = [f;-Q*xr;-R*ur];
    end
    % Actualización del estado actual
    b((n*N+1):(n*(N+1))) = xr(1:n);
```

```
    b(1:n) = observador(1:n);
    % Algoritmo FISTA
    x0 = zeros(n*(N+1),1);
    y1 = zeros(n*(N+1),1);
    t1 = 1;
    fin = 0;
    repeticiones = 0;
    while (fin==0)
        f2 = f - E'*y1;
        % Problema 1
        z0 = -f2'/H;
        zaux = min(abs(z0' - newz),edge).*sign(z0' - newz);
        z0fista = zaux+newz;
        %%%%%%%%%%%%%
        repeticiones = repeticiones + 1;
        if abs(E*z0fista - b) < TOL | repeticiones > repeticionesmax
            fin = 1;
        else
            incrementoy = -Qfista*(E*z0fista - b);
            x1 = y1 + incrementoy;
            t2 = 0.5*(1 + sqrt(1 + 4*(t1^2)));
            y2 = x1 + (t1 - 1)/(t2)*(x1 - x0);
            y1 = y2;
            x0 = x1;
            t1 = t2;
        end
    end
    u = z0fista(n+1:n+m);

    %% Sistema NO LINEAL
    U = u + Uf;
    X = simula_NL_4Tanques(X,U,Tm,h)';
    x = X - Xf;
    y = C*x;
    % Perturbación a la salida
    if rem(i,200)==0
       perturb = rand(m,1)/20;
    end
    y = y + perturb;
    Y = y + Xf(1:m);
    %% Toma de datos
    estado(i+1,:) = X;
    obs(i,:) = observador + Xf;
    entrada(i+1,:) = U;
    salida(i+1,:) = Y;
end
%% Representación gráfica
% Figura 1
figure(1)
for k = 1:4
    hf = subplot(2,2,k);
    plot(1:i,estado(1:i,k),'LineWidth',2); hold on;
    plot(1:i,obs(1:i,k),'LineWidth',2);
    legend('Estado','Observador')
    title(sprintf('Estado %d',k))
    xlabel('Intervalo de muestreo')
    ylabel('X_{0}')
    hf.FontSize = 22;
    hf.Box = 'on';
    grid on
end
% Figura 2
figure(2)
```

```
hf = subplot(1,1,1);
plot(1:i,salida(1:i,1),'LineWidth',2); hold on;
plot(1:i,salida(1:i,2),'LineWidth',2);
legend('Salida 1','Salida 2')
plot(1:499,Ref1(1)*ones(1,499),'k--',1:499,Ref1(2)*ones(1,499),'k--');
plot(500:1000,Ref2(1)*ones(1,501),'k--',500:1000,Ref2(2)*ones(1,501),'k--');
xlabel('Intervalo de muestreo')
ylabel('Y_{0}')
hf.FontSize = 25;
hf.Box = 'on';
grid on
```

ANEXO B: DLL

B1. Generador de la cabecera

```
%% Genera los datos para la DLL
clear; clc;
fid = fopen('generar_datos_c.txt','w');
% Puntos de funcionamiento
Uf1 = [1.7602; 1.8072];
Uf2 = [1.948; 2];
Uf3 = [1.4802; 1.5197];
Xf1 = puntoFunc(Uf1);
Xf2 = puntoFunc(Uf2);
Xf3 = puntoFunc(Uf3);
%% Parámetros modificables
n = 4;
m = 2;
h = 0.1; % Tiempo de integración en segundos
Tm = 5; % Tiempo de muestreo en segundos
% Punto de linealización
Uf = Uf1;
Xf = Xf1;
% Parámetros de los filtros
rho = 0.7;
alpha = 0.8;
% Constantes modificables para el MPC
N = 20;
TOL = 1e-7;
repeticionesmax = 2000;
% Matrices
Q = 5*ones(n,1);
R = 3*ones(m,1);
Q0 = eye(n+m);
R0 = 0.1*eye(m);

%% Parametización del modelo
[A,B,C,D] = modelo4tanques(Xf,Tm);
% Cálculo de los límites
LBx = 0.2 - Xf;
UBx = 1.2 - Xf;
LBu = 0 - Uf;
UBu = 3 - Uf;
% Generar límites
LB = [LBx;LBu];
UB = [UBx;UBu];
% Generar fronteras
edge = (UB - LB)./2;
newz = (UB + LB)./2;
% Cálculo de los parámetros del observador
A0 = eye(n+m);
A0(1:n,1:n) = A;
B0 = [C,eye(m)];
Lxd = dlqr(A0',B0',Q0,R0)';
% Generar matriz H
H = zeros(N*(m+n));
k=0;
while k <= (N-1)*(m+n)
    for i=1:n
        H(k+i,k+i) = Q(i);
    end
    for j=1:m
        H(k+i+j,k+i+j) = R(j);
```

```
    end
        k = k + i + j;
end
% Generar matriz E
E = kron(eye(N),[A B]);
j = 0;
for i=1:n:n*N-n
    j = j+1;
    E(i:i+n-1,((j-1)*(n+m)+(m+n+1)):((j-1)*(n+m)+(n+m+1)+n-1)) = -eye(n);
end
E = [eye(n) zeros(n,n*N-n+m*N); E];
% Generar E1 y E2
E1 = E(n+1:2*n,1:2*n+m);
ET1 = E(1:2*n,1:n+m)';
ET2 = E(n+1:3*n,n+m+1:2*(n+m))';
% Generar Qfista
Qfista = inv(E*inv(H)*E');
%% SSTO
L = [A-eye(n) B;C D]\[zeros(n,m);eye(m)];
c = [L;-L];
d = [UBx;UBu;-LBx;-LBu];

%% Generar archivo
fprintf(fid,'/* Cabecera fuera de la función */\n');
% Generar dimensiones
fprintf(fid,'#define n %d\n',n);
fprintf(fid,'#define m %d\n',m);
fprintf(fid,'#define N %d\n\n',N);
fprintf(fid,'/* Cabecera dentro de la función */\n');
% Generar Uf
[ii,~]=size(Uf);
fprintf(fid,'double Uf[%d]={',ii);
for i=1:ii-1
    fprintf(fid,'%.4f,',Uf(i));
end
fprintf(fid,'%.4f};\n',Uf(ii));
% Generar Xf
[ii,~]=size(Xf);
fprintf(fid,'double Xf[%d]={',ii);
for i=1:ii-1
    fprintf(fid,'%.4f,',Xf(i));
end
fprintf(fid,'%.4f};\n\n',Xf(ii));
% Generar A
[ii,jj]=size(A);
fprintf(fid,'double A[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',A(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n\n',A(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',A(i,jj));
    end
end
% Generar B
[ii,jj]=size(B);
fprintf(fid,'double B[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',B(i,j));
    end
```

```
    if i==ii
        fprintf(fid,'%.4f}};\n\n',B(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',B(i,jj));
    end
end
% Generar C
[ii,jj]=size(C);
fprintf(fid,'double C[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',C(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n\n',C(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',C(i,jj));
    end
end
% Generar edge
[ii,~]=size(edge);
fprintf(fid,'double edge[%d]={',ii);
for i=1:ii-1
    fprintf(fid,'%.4f,',edge(i));
end
fprintf(fid,'%.4f};\n',edge(ii));
% Generar newz
[ii,~]=size(newz);
fprintf(fid,'double newz[%d]={',ii);
for i=1:ii-1
    fprintf(fid,'%.4f,',newz(i));
end
fprintf(fid,'%.4f};\n\n',newz(ii));
% Generar Q
[ii,~]=size(Q);
fprintf(fid,'double Q[%d]={',ii);
for i=1:ii-1
    fprintf(fid,'%.4f,',Q(i));
end
fprintf(fid,'%.4f};\n',Q(ii));
% Generar R
[ii,~]=size(R);
fprintf(fid,'double R[%d]={',ii);
for i=1:ii-1
    fprintf(fid,'%.4f,',R(i));
end
fprintf(fid,'%.4f};\n\n',R(ii));
% Generar E1 y E2
[ii,jj]=size(E1);
fprintf(fid,'double E[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',E1(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n',E1(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',E1(i,jj));
    end
end
[ii,jj]=size(ET1);
fprintf(fid,'double ET1[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',ET1(i,j));
```

```
    end
    if i==ii
        fprintf(fid,'%.4f}};\n',ET1(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',ET1(i,jj));
    end
end
[ii,jj]=size(ET2);
fprintf(fid,'double ET2[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',ET2(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n\n',ET2(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',ET2(i,jj));
    end
end
% Generar Qfista
[ii,jj]=size(Qfista);
fprintf(fid,'double Qfista[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',Qfista(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n\n',Qfista(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',Qfista(i,jj));
    end
end
% Generar Lxd
[ii,jj]=size(Lxd);
fprintf(fid,'double Lxd[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',Lxd(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n\n',Lxd(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',Lxd(i,jj));
    end
end
% Generar L
[ii,jj]=size(L);
fprintf(fid,'double L[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
        fprintf(fid,'%.4f,',L(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n',L(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',L(i,jj));
    end
end
% Generar c
[ii,jj]=size(c);
fprintf(fid,'double c[%d][%d]={{',ii,jj);
for i=1:ii
    for j=1:jj-1
```

```
        fprintf(fid,'%.4f,',c(i,j));
    end
    if i==ii
        fprintf(fid,'%.4f}};\n',c(ii,jj));
    else
        fprintf(fid,'%.4f},\n\t{',c(i,jj));
    end
end
% Generar d
[ii,~]=size(d);
fprintf(fid,'double d[%d]={',ii);
for i=1:ii-1
    fprintf(fid,'%.4f,',d(i));
end
fprintf(fid,'%.4f};\n\n',d(ii));
% Generar parámetros MPC
fprintf(fid,'double TOL=%.10f;\n',TOL);
fprintf(fid,'int repeticionesmax=%d;\n\n',repeticionesmax);
% Generar parámetros filtros
fprintf(fid,'double rho=%.4f;\n',rho);
fprintf(fid,'double alpha=%.4f;\n\n',alpha);
% Generar observador
fprintf(fid,'double observador[%d];\n',n);
fprintf(fid,'double observador2[%d];\n',n);
fprintf(fid,'double dgorro2[%d];\n',m);
fprintf(fid,'double rk[%d];\n\n',m);
% Generar variables auxiliares
fprintf(fid,'int i,j;\n');
fprintf(fid,'double landa;\n');
fprintf(fid,'double xr[%d];\n',n);
fprintf(fid,'double ur[%d];\n',m);
fprintf(fid,'double aux1[%d];\n',2*(n+m));
fprintf(fid,'double aux2[%d];\n',n);
fprintf(fid,'double aux3[%d];\n',n);
fprintf(fid,'double f[%d];\n',n+m);
fprintf(fid,'double x0[%d];\n',n*(N+1));
fprintf(fid,'double y1[%d];\n',n*(N+1));
fprintf(fid,'double t1,t2;\n');
fprintf(fid,'int fin,repeticiones;\n');
fprintf(fid,'int ii,jj;\n');
fprintf(fid,'double f2[%d];\n',N*(n+m));
fprintf(fid,'double z0[%d];\n',N*(n+m));
fprintf(fid,'double z0fista[%d];\n',N*(n+m));
fprintf(fid,'double res[%d];\n',n*(N+1));
fprintf(fid,'double incrementoy[%d];\n',n*(N+1));
fprintf(fid,'double x1[%d];\n',n*(N+1));
fprintf(fid,'double zaux[%d];\n',n);
fprintf(fid,'int signo,tol;\n\n');
fprintf(fid,'double u[%d];\n',m);
fprintf(fid,'double r[%d];\n',m);
fprintf(fid,'double y[%d];\n',m);
fclose('all');
```

B2. *dll.h*

```
#ifndef _DLL_H_
#define _DLL_H_

#if BUILDING_DLL
# define DLLIMPORT __declspec (dllexport)
#else /* Not BUILDING_DLL */
# define DLLIMPORT __declspec (dllimport)
#endif /* Not BUILDING_DLL */
```

```
DLLIMPORT double MPC (double *, double *, double *, double *, double *, double *,
double *);
DLLIMPORT double PLANTA (double *, double *, double *);


#endif /* _DLL_H_ */
```

B3. *dllmain.c*

```
#include "dll.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Insertar cabecera generada en MATLAB */

/* ---------------------------------- */

DLLIMPORT double MPC (double *U, double *Y, double *Ref, double *Obs, double
*rfiltrada, double *dgorro, double *dr)
{
    /* Insertar cabecera generada en MATLAB */


    /* ---------------------------------- */

    for(i=0;i<n;i++)
    {
        observador[i]=Obs[i]-Xf[i];
    }
    for(i=0;i<m;i++)
    {
        u[i]=U[i]-Uf[i];
        y[i]=Y[i]-Xf[i];
        r[i]=Ref[i]-Xf[i];
    }

    /* Observador */
    /*observador2=A*observador+B*u+Lxd*(y-C*observador-dgorro);*/
    for(i=0;i<m;i++)
    {
        /* 2x4-4x1 y-C*observador-dgorro */
        aux3[i]=0;
        for(j=0;j<n;j++)
        {
            aux3[i]+=C[i][j]*observador[j];
        }
        aux3[i]=y[i]-aux3[i]-dgorro[i];
    }
    for(i=0;i<n;i++)
    {
        /* 4x4-4x1 A*observador */
        aux1[i]=0;
        for(j=0;j<n;j++)
        {
            aux1[i]+=A[i][j]*observador[j];
        }
        /* 4x2-2x1 B*u+Lxd*(y-C*observador-dgorro) */
        aux2[i]=0;
        for(j=0;j<m;j++)
        {
            aux2[i]+=B[i][j]*u[j]+Lxd[i][j]*aux3[j];
        }
        observador2[i]=aux1[i]+aux2[i];
        observador[i]=observador2[i];
        Obs[i]=observador[i]+Xf[i];
    }
```

```
    /*dgorro2=dgorro+Lxd*(y-C*observador-dgorro);*/
    for(i=0;i<m;i++)
    {
        /* 2x2-2x1 Lxd*(y-C*observador-dgorro) */
        aux1[i]=0;
        for(j=0;j<m;j++)
        {
            aux1[i]+=Lxd[i+n][j]*aux3[j];
        }
        dgorro2[i]=dgorro[i]+aux1[i];
        dgorro[i]=dgorro2[i];

        /* Filtro del observador */
        dr[i]=(1-alpha)*dr[i]+alpha*dgorro[i];

        /* Actualizacion de la referencia */
        rk[i]=r[i]-dr[i];

        /* Filtro del SSTO */
        rfiltrada[i]=(1-rho)*rfiltrada[i]+rho*rk[i];
    }

    /* SSTO */
    landa=1;
    for(i=0;i<2*(n+m);i++)
    {
        /* 12x2-2x1 c*rfiltrada */
        aux1[i]=0;
        for(j=0;j<m;j++)
        {
            aux1[i]+=c[i][j]*rfiltrada[j];
        }
        if (aux1[i]>d[i])
        {
            aux2[0]=d[i]/aux1[i];

            if (aux2[0]<landa)
            {
                landa=aux2[0];
            }
        }
    }
    for(i=0;i<n+m;i++)
    {
        /* 6x2-2x1 L*rfiltrada*landa */
        aux1[i]=0;
        for(j=0;j<m;j++)
        {
            aux1[i]+=L[i][j]*rfiltrada[j];
        }
        aux1[i]=aux1[i]*landa;
    }
    for(i=0;i<n;i++)
    {
        xr[i]=aux1[i];
    }
    for(i=0;i<m;i++)
    {
        ur[i]=aux1[i+n];
    }

    /* MPC */
    /* Generar f */
    /* 6x1 f=[-Q*xr;-R*ur]; */
    for(i=0;i<n;i++)
    {
        f[i]=-Q[i]*xr[i];
```

```
    }
    for(i=0;i<m;i++)
    {
        f[i+n]=-R[i]*ur[i];
    }

    /* Algoritmo FISTA */
    for(i=0;i<n*(N-1);i++)
    {
        x0[i]=0;
        y1[i]=0;
    }

    t1=1;
    fin=0;
    repeticiones=0;

    /* BUCLE */
    while (fin==0)
    {
        /* Calculo de f2 */
        /* 120x1 f2=f-E'*y1 */
        for(i=0;i<n+m;i++)
        {
            /* 6x8-8x1 ET1*y1 */
            aux1[i]=0;
            for(j=0;j<2*n;j++)
            {
                aux1[i]+=ET1[i][j]*y1[j];
            }
            f2[i]=f[i]-aux1[i];
        }
        for(ii=n+m,jj=n;ii<N*(n+m);ii+=n+m,jj+=n)
        {
            for(i=0;i<n+m;i++)
            {
                /* 6x8-8x1 ET2*y1 */
                aux1[i]=0;
                for(j=0;j<2*n;j++)
                {
                    aux1[i]+=ET2[i][j]*y1[jj+j];
                }
                f2[ii+i]=f[i]-aux1[i];
            }
        }

        /* Calculo de z0 */
        /* z0=-f2'/H; */
        for(ii=0;ii<N*(n+m);ii+=n+m)
        {
            for(i=0;i<n;i++)
            {
                z0[ii+i]=-f2[ii+i]/Q[i];
            }
            for(j=0;j<m;j++)
            {
                z0[ii+i+j]=-f2[ii+i+j]/R[i];
            }
        }

        /* Calculo de z0fista*/
        /* zaux=min(abs(z0'-newz),edge).*sign(z0'-newz)+newz*/
        for(ii=0;ii<N*(n+m);ii+=n+m)
        {
            for(i=0;i<n+m;i++)
            {
                signo=0;
```

```
                aux1[i]=z0[ii+i]-newz[i];
                if (aux1[i]<0)
                {
                    aux1[i]=-aux1[i];
                    signo=1;
                }

                if (aux1[i]>edge[i])
                {
                    zaux[i]=edge[i];
                }
                else
                {
                    zaux[i]=aux1[i];
                }
                if (signo)
                {
                    z0fista[ii+i]=-zaux[i]+newz[i];
                }
                else
                {
                    z0fista[ii+i]=zaux[i]+newz[i];
                }
            }
        }

        /* Comprobacion fin de bucle */
        repeticiones++;
        tol=0;
        /* E*z0fista-b */
        for(i=0;i<n;i++)
        {
            /* 4x4 E*z0fista-b */
            res[i]=z0fista[i]-observador[i];
            /* 4x6-6x1 E*z0fista-b */
            aux1[i]=0;
            for(j=0;j<n+m;j++)
            {
                aux1[i]+=E[i][j]*z0fista[(n+m)*(N-1)+j];
            }
            res[n*N+i]=aux1[i]-xr[i];
        }
        for(ii=n,jj=0;ii<n*N;ii+=n,jj+=n+m)
        {
            for(i=0;i<n;i++)
            {
                /* 4x10-10x1 E*z0fista-b */
                aux1[i]=0;
                for(j=0;j<2*n+m;j++)
                {
                    aux1[i]+=E[i][j]*z0fista[jj+j];
                }
                res[ii+i]=aux1[i];
            }
        }

        /* abs(E*z0fista-b)<TOL */
        for(i=0;i<n*(N+1);i++)
        {
            if (res[i]<0)
            {
                if (-res[i]<TOL)
                {
                    tol++;
                }
            }
            else
            {
```

```
                if (res[i]<TOL)
                {
                    tol++;
                }
            }
        }

        /* if abs(E*z0fista-b)<TOL | repeticiones>repeticionesmax */
        if ((tol==n*(N+1)) || (repeticiones>repeticionesmax))
        {
            fin=1;
        }
        else
        {
            /* incrementoy=-Qfista*(E*z0fista-b) */
            for(i=0;i<n*(N+1);i++)
            {
                /* 84x84-84x1 */
                incrementoy[i]=0;
                for(j=0;j<n*(N+1);j++)
                {
                    incrementoy[i]+=-Qfista[i][j]*res[j];
                }
                /* x1=y1+incrementoy */
                x1[i]=y1[i]+incrementoy[i];
            }
            /* t2=0.5*(1+sqrt(1+4*(t1^2))) */
            t2=0.5*(1+sqrt(1+4*t1*t1));
            /* y1=x1+(t1-1)/(t2)*(x1-x0) */
            for(i=0;i<n*(N+1);i++)
            {
                y1[i]=x1[i]+((t1-1)/t2)*(x1[i]-x0[i]);
                x0[i]=x1[i];
            }
            t1=t2;
        }
    }
    for(i=0;i<m;i++)
    {
        U[i]=z0fista[n+i]+Uf[i];
    }

    return 1;
}

DLLIMPORT double PLANTA (double *U, double *X, double *Y)
{
    int i,j;
    double h = 0.1;
    /* Sistema NO LINEAL (LEER Y)*/
    X[0] = X[0] + h*((-1.3104e-4*sqrt(2*9.81*X[0])+9.2673e-
5*sqrt(2*9.81*X[2])+(0.3/3600)*U[0])/0.03);
    X[1] = X[1] + h*((-1.5074e-4*sqrt(2*9.81*X[1])+8.8164e-
5*sqrt(2*9.81*X[3])+(0.4/3600)*U[1])/0.03);
    X[2] = X[2] + h*((-9.2673e-5*sqrt(2*9.81*X[2])+(0.6/3600)*U[1])/0.03);
    X[3] = X[3] + h*((-8.8164e-5*sqrt(2*9.81*X[3])+(0.7/3600)*U[0])/0.03);

    for(j=0;j<4;j++)
    {
        if(X[j]<0)
        {
            X[j]=0;
        }
    }
    for(i=0;i<m;i++)
    {
        Y[i]=X[i];
```

```
    }

    return 1;
}


BOOL APIENTRY DllMain (HINSTANCE hInst     /* Library instance handle. */ ,
                       DWORD reason        /* Reason this function is being called. */
,
                       LPVOID reserved     /* Not used. */ )
{
    switch (reason)
    {
      case DLL_PROCESS_ATTACH:
        break;

      case DLL_PROCESS_DETACH:
        break;

      case DLL_THREAD_ATTACH:
        break;

      case DLL_THREAD_DETACH:
        break;
    }

    /* Returns TRUE on success, FALSE on failure */
    return TRUE;
}
```