UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA INDUSTRIAL

PROYECTO FIN DE CARRERA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

INTERFAZ GRÁFICA PARA SINTONÍA DE CONTROLADORES FRACCIONARIOS MEDIANTE ALGORITMO DE EVOLUCIÓN DIFERENCIAL

AUTORA: TAMARA MARTÍN MORENO

TUTORA: CONCEPCIÓN ALICIA MONJE MICHARET

DIRECTOR: FERNANDO MARTÍN MONAR

LEGANÉS, MAYO 2016

AGRADECIMIENTOS

Por fin ha llegado el momento de dar las gracias a todas aquellas personas que, directa o indirectamente, me han apoyado en la realización de este proyecto, con el que culmina el esfuerzo y trabajo dedicados a todos estos años de Ingeniería Industrial.

En primer lugar, quiero dar las gracias a mi tutora Concha por el apoyo y la paciencia que me ha mostrado en todo momento, así como por su ayuda y colaboración, sin las que me habría resultado imposible llegar hasta aquí. También quisiera agradecer a Fernando todo el tiempo dedicado a ayudarme con sus conocimientos, ideas y aportaciones. Me siento muy afortunada de haber podido realizar este proyecto con vosotros porque he aprendido muchísimo.

Gracias a vosotros dos, y a todos esos profesores que he tenido a lo largo de mi vida de estudiante, de los cuales me llevo algo más que sus conocimientos: a Cristina por enseñarme a lidiar con mis puntos débiles, a Ángel por "enseñarme a volar", a Mº José por sus "truculencias" y a Alberto "Spasic" por esas irónicas "curiosidades, amenidades y pasatiempos" que ahora tienen más sentido que nunca.

A mis amigos y compañeros de la universidad, muchas gracias por todos los momentos compartidos. Elsa, Lucía, Sara, Coral, Elena... habéis estado a mi lado cuando más lo necesitaba... Y por supuesto, Silvia, mi paso por la universidad no lo concibo sin ti, sin ese primer día en que nos conocimos cuando aún no teníamos ni idea de todo lo que nos iba a deparar esta andadura. A pesar de que las leyes educativas no nos permitieron seguir juntas, sabes que para mí siempre has sido y serás un gran apoyo, una gran amiga… mi "TCU".

A todas las personas que durante estos años han empezado a formar parte de mi vida, a las que perduran, y a las que ya no están, gracias porque de alguna manera me habéis aportado la fuerza y la motivación necesarias para llegar hasta aquí.

Por último, quiero agradecer a mi familia el esfuerzo que han hecho para que yo pudiera llegar hasta aquí. A Nuria, por ser mi referente y mi guía para dar los primeros pasos en esta carrera. A mi abuela, por ser mi ángel de la guarda. A mis padres, por los valores que me habéis inculcado, el apoyo y toda vuestra ayuda en mis momentos de desesperación, por ello, os dedico este trabajo. Especialmente a tí, mamá, por tus ánimos y tu insistencia, esto también es parte de ti.

Tamara

RESUMEN

Este proyecto final de carrera se centra en el desarrollo de una interfaz gráfica en MATLAB a través de su herramienta GUIDE, sobre la que se debe integrar el algoritmo de Evolución Diferencial para sintonizar controladores $PI^{\lambda}D^{\mu}$ fraccionarios.

De entre los distintos métodos de sintonía disponibles para el diseño de un controlador, se ha optado por elegir este algoritmo evolutivo porque permitirá crear una interfaz más versátil. Esto se debe a que dichos algoritmos realizan un ajuste basado en la optimización de una función objetivo, la cual mide el grado en que se cumplen varias especificaciones de diseño.

Esta nueva herramienta de simulación le ofrece al usuario las funcionalidades necesarias para llevar a cabo la sintonía de controladores fraccionarios tanto en el dominio del tiempo como de la frecuencia, así como la configuración de los diversos parámetros del filtro de optimización.

Finalmente, se incluirá un manual de ayuda para que el usuario pueda consultar todo lo referente a esta interfaz de sintonización fraccionaria.

Tanto la interfaz como su manual de ayuda se elaborarán en inglés con vistas a que esta herramienta software pueda tener una difusión de ámbito internacional.

Palabras clave: controladores $PI^{\lambda}D^{\mu}$ fraccionarios, Evolución Diferencial (DE), Interfaz Gráfica de Usuario (GUI), MATLAB.

ABSTRACT

This work focuses on the development of a graphical user interface in MATLAB through its GUIDE toolbox, on which the Differential Evolution algorithm should be integrated to tune fractional $PI^{\lambda}D^{\mu}$ controllers.

Among the different tuning methods available for the design of a controller, the evolutionary algorithm has been chosen because it will allow to create a more versatile method. This type of algorithm performs an adjustment based on the optimization of a target function. The flexibility of this fitness function makes it possible to meet multiple design specifications.

This new simulation tool provides the user with all the functionalities to carry out the time and frequency domain tuning of fractional controllers and to configure the different parameters of the optimization filter.

Finally, a help guide will be included so that the user could check the method performance regarding this fractional tuning interface.

Both the interface and the help guide will be developed in English with the aim of making this software tool internationally spread and well known.

Keywords: fractional $PI^{\lambda}D^{\mu}$ controllers, Differential Evolution (DE), Graphical User Interface (GUI), MATLAB.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Introducción: origen y motivación del proyecto

A pesar de que la idea de los operadores fraccionarios es tan antigua como la idea de los operadores de orden entero, ha sido en las últimas décadas cuando el uso de controladores fraccionarios se ha convertido cada vez más en el centro de importantes campos de investigación. El interés de estos controladores surge por su capacidad para resolver problemas de control relacionados con aplicaciones industriales que exigen mayor robustez y menor esfuerzo de implementación.

Por tanto, el auge de los controladores fraccionarios viene motivado por el gran conocimiento que se tiene hoy en día sobre el cálculo fraccionario y la creciente demanda de este tipo de controladores en multitud de campos de la ciencia y la ingeniería, entre los que se pueden destacar la ciencia de materiales, la teoría del caos y los fractales, la electrónica de dispositivos, la física teórica y la mecánica, etc. [1] De ahí radica la importancia de estudiar aspectos relevantes de los controladores fraccionarios, tales como el análisis, el diseño y la implementación de los mismos.

Dado que en la actualidad podemos encontrar innumerables problemas de control, existe una gran cantidad de métodos de sintonía disponibles para el diseño de un controlador. Entre todos ellos, este trabajo está basado en el algoritmo de Evolución Diferencial.

Con la intención de que el uso de este método sea utilizado de una manera rápida y cómoda, se propone la implementación de éste en una interfaz gráfica a través de funciones y objetos gráficos.

El origen de las interfaces gráficas se remonta a un período donde la mayoría de ordenadores, para su uso requerían un mínimo de conocimientos técnicos si se querían usar como algo más que una consola para jugar. Tras la creación de las interfaces se evitó que el usuario tuviese que aprender un entorno complejo, de esta manera, el uso del ordenador se incrementó en la sociedad. Tanto es así, que en tiempos actuales podemos decir que las interfaces gráficas poseen un papel relevante en el mundo tecnológico [2][3].

En función del uso que se les quiera dar, existen múltiples lenguajes para crearlas. Después de hacer un estudio sobre estos lenguajes, podemos destacar cuales son los principales: C++, Java, MATLAB. Cualquiera de los tres resulta muy potente y de gran importancia en la actualidad, pero tras evaluar las necesidades del método a implementar y debido a que dicho método está implementado en MATLAB, se opta por esta opción.

Se trata de un programa muy utilizado en el mercado gracias a su lenguaje de alto nivel y su entorno interactivo para el cálculo numérico [4]. También se

caracteriza por tener gran variedad de aplicaciones entre la que destacamos GUIDE, aplicación usada para la creación de interfaces gráficas.

Como resultado de la unión de estas dos ideas generales, surge este proyecto final de carrera, cuyo fin es la implementación de un método de sintonía de controladores $PI^{\lambda}D^{\mu}$ fraccionarios basado en el algoritmo de Evolución Diferencial mediante una interfaz gráfica en MATLAB.

1.2. Objetivos

El proyecto fin de carrera tiene como objetivo principal el diseño y la creación de una interfaz gráfica en MATLAB mediante la herramienta GUIDE, sobre la que se debe integrar el algoritmo de Evolución Diferencial para sintonizar controladores $PI^{\lambda}D^{\mu}$ fraccionarios.

Esta nueva herramienta se ha creado con la finalidad de que el usuario pueda interactuar con ella y le permita:

- Simular el ajuste de un controlador de orden fraccional en el dominio del tiempo o de la frecuencia.
- Elegir los parámetros de configuración del filtro de optimización dependiendo del dominio en el que se desee realizar el ajuste.
- Visualizar los resultados a través de gráficas.
- Almacenar datos, gráficas e imágenes del programa.
- Abrir un manual de ayuda para el usuario.

1.3. Planificación del proyecto

Este proyecto comenzó a desarrollarse el 21 de Septiembre del 2015. Inicialmente se planificaron las distintas etapas en que se divide el proyecto, las cuales se describen a continuación:

- La primera etapa está dedicada al estudio de la herramienta GUIDE. A partir de otros proyectos realizados anteriormente y de búsquedas en internet, conseguimos conocer los conceptos básicos de esta herramienta para emplearla posteriormente.

- La segunda etapa está dedicada al estudio del método de sintonización a implementar. En esta fase se adquieren todos los conocimientos sobre el método y sus posibles alternativas de implementación. Se debe citar especialmente el documento [5], donde se encuentra publicado todo lo referente al método de sintonización en el que se basa este proyecto.

- La tercera etapa, y podría decirse que la más importante, se centra en la implementación del método de sintonía en MATLAB, a través de su herramienta GUIDE. Resulta ser la fase más compleja por ser la encargada de cumplir los objetivos.

- La cuarta etapa está dedicada al análisis de los resultados obtenidos tras finalizar el trabajo y a realizar las pruebas de testeo de la sintonía. Esta fase se considera de gran importancia porque permitirá corregir los posibles errores que se puedan haber detectado anteriormente.

- El bloque final está dedicado a realizar la memoria del proyecto.

En la Tabla 1 aparece el diagrama del proyecto donde se estima el tiempo empleado en cada una de dichas fases.

#	Nombre de la tarea	Duración	Inicio	Final
1	Estudio de la herramienta GUIDE	39	21/09/2015	30/10/2015
2	Estudio del método de sintonía	25	02/11/2015	27/11/2015
3	Diseño de la interfaz	81	30/11/2015	19/02/2016
4	Pruebas y análisis de los resultados	46	22/02/2016	08/04/2016
5	Elaboración de la memoria del proyecto	206	21/09/2015	14/04/2016

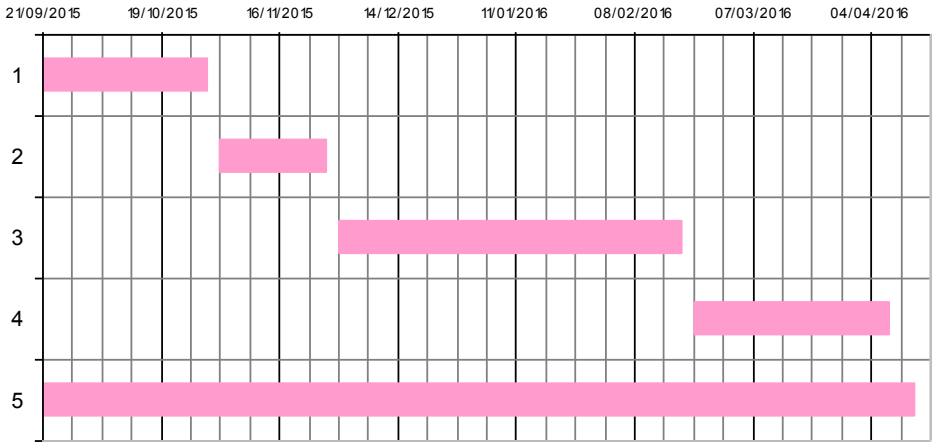

Tabla 1. Diagrama de Gantt

1.4. Estructura de la memoria

A continuación, se presenta un breve resumen de cada uno de los capítulos que componen la memoria de este proyecto fin de carrera.

- Capítulo 1 "**Introducción**": Se realiza una presentación acerca del contenido del documento incluyendo el origen y la motivación del proyecto, sus principales objetivos y una breve descripción de la estructura del documento.

- Capítulo 2 "**Sintonía de controladores fraccionarios mediante Evolución Diferencial**": Se describen detalladamente las características de un controlador fraccionario, el método de ajuste elegido y el algoritmo concreto para llevar a cabo la sintonía de dichos controladores.

- Capítulo 3 "**Diseño de la interfaz gráfica en MATLAB**": Se introducen brevemente los aspectos más importantes del entorno MATLAB y se describe más ampliamente su herramienta GUIDE para la creación de interfaces gráficas.

- Capítulo 4 "**Programación de la interfaz**": Se explica en detalle cómo se ha programado el diseño de la interfaz, describiendo su estructura general y las funciones utilizadas.

- Capítulo 5 "**Funcionamiento de la interfaz**": Se detalla paso a paso el funcionamiento de toda la interfaz gráfica, teniendo en cuenta ambos enfoques de sintonía (dominio del tiempo y de la frecuencia).

- Capítulo 6 "**Conclusiones y trabajos futuros**": Se realiza un análisis crítico sobre los resultados del trabajo y a continuación se comentan las posibles propuestas para la mejora del trabajo.

2. SINTONÍA DE CONTROLADORES FRACCIONARIOS MEDIANTE EVOLUCIÓN DIFERENCIAL

Este capítulo está estructurado en tres secciones, donde se recogen las bases teóricas en que se fundamenta este proyecto.

En la sección 2.1 se exponen los conceptos básicos más relevantes en control fraccionario. En la sección 2.2 se explica en qué consiste el algoritmo de Evolución Diferencial como método de sintonización para controladores fraccionarios. Y finalmente, en la sección 2.3, se especifica la estrategia concreta empleada para sintonizar controladores $PI^{\lambda}D^{\mu}$ fraccionarios. Por una parte, en la sección 2.3.1 se detalla la programación del algoritmo DE, y por su parte, en la sección 2.3.2 se definen y analizan las funciones de coste necesarias para llevar a cabo la sintonía en el dominio del tiempo o en el dominio de la frecuencia.

2.1. Introducción al control fraccionario

El control fraccionario ha ido incrementando su relevancia en Teoría de Control durante estos últimos años. Esto se ha debido a la aparición de artículos sobre el uso del cálculo fraccionario en disciplinas íntimamente relacionadas con la teoría de control, como son el tratamiento de señales, el ajuste de curvas y la estimación de parámetros, o el diseño y realización de filtros.

La herramienta matemática utilizada es el cálculo fraccionario, que puede definirse como una extensión natural de la matemática clásica que permite considerar la integración y la derivación de cualquier orden, no necesariamente entero.

Los aportes teóricos más tempranos sobre el cálculo fraccionario fueron realizados por Euler, Liouville, Abel o Laplace. Fue este último quien introdujo la noción del dominio de Laplace para describir con mayor comodidad la operación fraccional integro-diferencial. Donde la transformada de Laplace de la derivada/integral fraccional bajo condiciones iniciales nulas de orden α ($0 < \alpha < 1$) viene dada por:

$$\pounds\left\{{}_{a}D_{t}^{\pm\alpha} f(t)\right\} = s^{\pm\alpha} F(s). \qquad (1)$$

A partir de entonces, con el objetivo de resolver algunos de los problemas de control que aparecen en ciertas aplicaciones industriales, el control fraccionario ha llamado la atención de muchos investigadores [6].

El control fraccionario propone el uso de operadores y sistemas fraccionarios para obtener los controladores que hagan cumplir con las especificaciones de la planta, tal y como se muestra en el diagrama de bloques de la Figura 2.1.

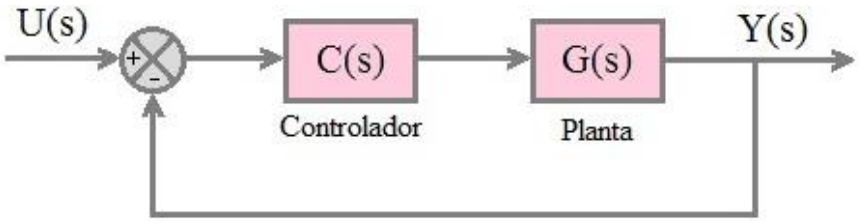

Figura 2.1. Diagrama de bloques de un sistema de control en circuito cerrado [5]

Donde G(s) es la función de transferencia de la planta o proceso a controlar; C(s) es el controlador de dicho proceso, cuyo objetivo es que el sistema de control cumpla unas especificaciones de funcionamiento dentro de una estructura con realimentación (considerando perturbaciones, incertidumbres y ruidos); U(s) son las entradas al sistema de control, sobre las que se puede actuar para conseguir el funcionamiento deseado; Y(s) son las salidas del sistema de control, cuyo valor o evolución se pretende regular.

El controlador PID, que engloba las tres acciones básicas de control (proporcional, integral y derivativa), es todavía el más utilizado en la industria de control de procesos debido a su simplicidad funcional y a la robustez que proporciona a los sistemas de control, y ha recibido una atención constante desde que Ziegler y Nichols presentaron sus métodos de sintonía en 1942 [7].

Así, en 1999 Podlubny [1] propone una generalización de los controladores clásicos PID definidos como $PI^{\lambda}D^{\mu}$, donde los órdenes de las partes integral (λ) y derivativa (μ) asumen valores fraccionales. De esta forma se consigue tratar con una clase más general de problemas de control en los cuales la naturaleza fraccionaria del controlador puede venir impuesta, bien por la naturaleza fraccionaria del propio sistema a controlar o bien por la especial naturaleza de las respuestas temporales o frecuenciales requeridas.

A continuación se analizarán los efectos de las acciones básicas de control en función de los valores que vaya tomando el parámetro μ en la siguiente expresión de un controlador:

$$C(s) = Ks^{\mu}, \qquad \mu \in [-1,1] \qquad (2)$$

- Acción proporcional ($\mu = 0$). Su efecto es como el de un amplificador con una ganancia ajustable, incitando la aparición de un error estacionario en el sistema.

- Acción integral ($\mu = -1$). Cuyos principales efectos son hacer más lenta la respuesta del sistema, disminuir la estabilidad relativa del mismo, y eliminar el error estacionario del sistema ante entradas para las que antes tenía un error finito. Dichos efectos se pueden ver en los distintos dominios de análisis. En el dominio del tiempo, los efectos sobre la respuesta transitoria se manifiestan en un decrecimiento del tiempo de subida y un aumento del tiempo de establecimiento y la sobreoscilación. En el dominio de la frecuencia dichos efectos se manifiestan en un

incremento de -20 dB/dec en las pendientes de la curva de magnitud y un descenso de $\pi/2$ en la curva de fase.

- Acción derivativa (μ =1). La acción derivativa, por su parte, aumenta la estabilidad del sistema y tiende a enfatizar los efectos de los ruidos y perturbaciones de alta frecuencia. En el dominio del tiempo, esto se manifiesta, principalmente, por una disminución tanto de la sobreoscilación como del tiempo de establecimiento. En el dominio de la frecuencia, como un adelanto constante de fase de $\pi/2$ y un aumento de 20 dB/dec en las pendientes de la curva de magnitud.

- En los casos en que se tengan órdenes fraccionarios de integración, es decir, $\mu \in (-1,0)$, y/o derivación, es decir, $\mu \in (0,1)$, la selección del valor de μ se traduce en una determinada ponderación de los efectos arriba comentados, tal como se muestra en la Figura 2.2.

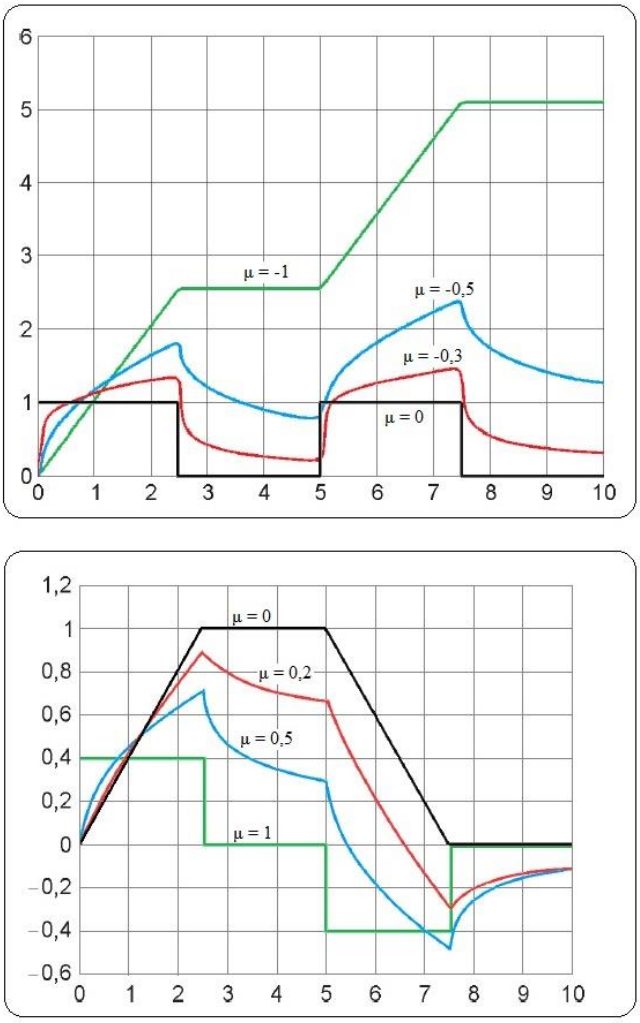

Figura 2.2. Efectos de las acciones integral sobre una onda cuadrada según el orden de integración (arriba) y derivativa sobre una función trapezoidal según el orden de derivación (abajo) [8]

La función de transferencia de un controlador $PI^{\lambda}D^{\mu}$ tiene esta expresión:

$$C(s) = k_p + k_i/s^{\lambda} + k_d s^{\mu}, \qquad (3)$$

Donde k_p, k_i y k_d son las ganancias proporcional, integral y derivativa de un PID convencional, respectivamente. Además aparecen dos parámetros más que son los órdenes fraccionales de las partes integral (λ) y derivativa (μ) del controlador fraccionario.

El uso de este tipo de controladores permite, frente al PID convencional, la consecución de hasta cinco especificaciones de diseño, ya que cuenta con cinco parámetros a sintonizar (k_p, k_i, k_d, λ, μ). La adicción de estos dos nuevos grados de libertad permite incrementar el número de especificaciones y obtener un control más robusto ante incertidumbres en la planta, rechazo al ruido u otros factores de diseño. Demostrando así que los controladores fraccionarios tienen mejores actuaciones de control [9].

Este hecho se puede ilustrar en la Figura 2.3, donde se observa que la acción de control de los controladores fraccionarios puede situarse en cualquier punto de la región sombreada, limitada por los órdenes máximos de integración y derivación. Mientras que la acción de control del PID convencional se reduce a uno de los cuatro puntos señalados en la gráfica, cuando λ = 1 y μ = 1.

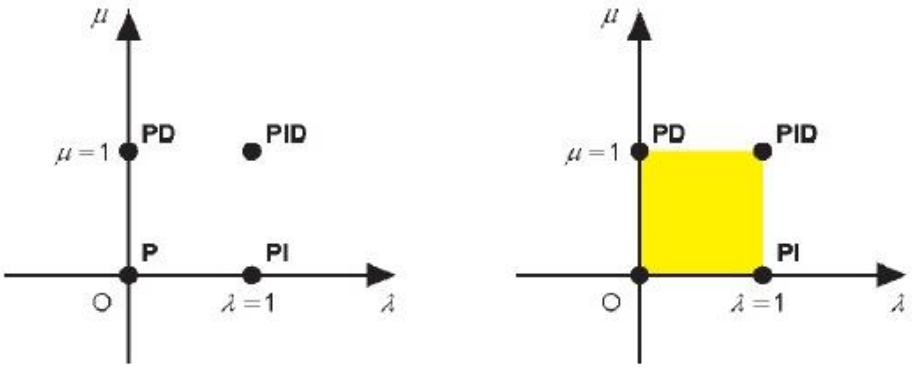

Figura 2.3. Región de trabajo de controladores de orden entero (izquierda) y orden fraccionario (derecha) [8]

Desde los trabajos de Ziegler y Nichols [7], se han propuesto muchos métodos de diseño y procedimientos de sintonía para los controladores PID. Teniendo en cuenta la posibilidad de seleccionar no sólo las constantes del controlador, sino también los órdenes de las acciones integral y derivativa, se pueden generalizar estos métodos y procedimientos de forma que puedan hacerse más flexibles y potentes, y permitan resolver una clase más amplia de problemas de control.

2.2. Introducción al algoritmo de Evolución Diferencial

Existen diversos métodos de sintonización para controladores fraccionarios [10], los cuales se pueden clasificar en tres grupos: analíticos, numéricos y basados en normas.

Los métodos análiticos sólo son asequibles para resolver sistemas de ecuaciones sencillos determinados tras establecer ciertas especificaciones. Las más empleadas son las propuestas por Vinagre [11] en relación a los márgenes de ganancia y de fase a sus respectivas frecuencias de cruce, y la propuesta por Chen [12] sobre la robustez ante variaciones en la ganancia.

El método basado en reglas permite calcular fácilmente los parámetros del controlador a partir de ciertas reglas de ajuste [13][14]. Sin embargo, este método sólo se puede aplicar en plantas cuya respuesta escalón sea en forma de S.

Los métodos numéricos o de optimización pretenden encontrar los parámetros del controlador que optimicen el valor de una función objetivo, la cual mide el grado en que se cumplen varias especificaciones de diseño.

En las últimas décadas se han desarrollado muchos algoritmos basados en la teoría de la evolución de Darwin, para solucionar problemas de optimización complejos (con alta dimensionalidad, multimodalidad, fuerte no linealidad, no diferenciabilidad, presencia de ruido y con funciones dependientes del tiempo). Estos algoritmos evolutivos se conocen como métodos de computación evolutiva, donde destacan los Algoritmos Genéticos o GA (Genetic Algorithms), Programación Genética o GP (Genetic Programming), Programación Evolutiva o EP (Evolutionary Programming) y Estrategias de Evolución o ES (Evolution Strategies).

El algoritmo de Evolución Diferencial o DE (Differential Evolution) es un método de optimización perteneciente a la categoría de computación evolutiva, que fue introducido por Storn y Price en 1996 [15]. El DE se aplica para la resolución de problemas complejos con funciones objetivo no diferenciables, no lineales y multimodales.

Al igual que otros algoritmos evolutivos, DE involucra una población de soluciones candidatas formada por un número fijo de vectores que pertenece al espacio de búsqueda D-dimensional del problema en consideración. Dichos vectores, después de ser inicializados de forma aleatoria, son sometidos a operaciones de mutación, recombinación y selección en cada iteración o generación. Lo que caracteriza a DE es el uso de vectores de prueba (como resultado de la recombinación) que compiten con los individuos de la población actual a fin de buscar el mejor individuo como solución del problema.

El algoritmo asume que la población está formada por *NP* individuos, donde cada uno es un vector de números reales de dimensión *D*, la cual se corresponde con el número de variables del problema a optimizar. Así se define un vector:

$$\mathbf{x_{i,g}} = \{x_{1i,g}, \dots, x_{ji,g}, \dots, x_{Di,g}\}, \qquad (4)$$

donde ***i*** es el índice del individuo de la población (i = 1,...,*NP*), ***j*** es el índice de la variable en el individuo (j = 1,...,*D*) y ***g*** es la generación correspondiente.

DE se compone básicamente de 4 pasos que se describen a continuación:

La **inicialización** se realiza aleatoriamente al principio de la ejecución de la búsqueda. Esta primera generación de población se crea considerando los valores mínimos y máximos de cada variable de decisión, los cuales han sido previamente definidos:

$$x_{ji,1} = x_j^{min} + rand(0,1) \cdot \left(x_j^{max} - x_j^{min}\right), \qquad (5)$$

para i = 1,...,*NP*, j = 1,...,*D*, y rand(0,1) un número aleatorio en el rango [0,1].

La **mutación** consiste en crear un vector mutado $\boldsymbol{v}_{i,g+1}$ (Figura 2.4) por cada vector objetivo $\boldsymbol{x}_{i,g}$, utilizando diferentes estrategias de acuerdo a las variantes de DE/*x*/*y*/*z*, que es la nomenclatura típica que se emplea en este algoritmo, donde: *x* indica el vector a ser mutado ("rand": vector de la población elegido al azar o "best": vector con el mejor valor de la función objetivo en la población actual); *y* indica el número de vectores de diferencia a considerar en la mutación (1 o 2); y *z* indica el tipo de recombinación utilizada ("bin": binomial o "exp": exponencial).

Entre las diez estrategias diferentes que existen [16], las más comúnmente utilizadas para generar $\boldsymbol{v}_{i,g+1}$ son:

1. DE/rand/1/bin: $\boldsymbol{v_{i,g+1}} = \boldsymbol{x_{r1,g}} + F \cdot \left(\boldsymbol{x_{r2,g}} - \boldsymbol{x_{r3,g}}\right) \qquad (6)$
2. DE/best/1/bin: $\boldsymbol{v_{i,g+1}} = \boldsymbol{x_{best}} + F \cdot \left(\boldsymbol{x_{r1,g}} - \boldsymbol{x_{r2,g}}\right) \qquad (7)$

donde *r1*, *r2*, *r3* son índices enteros aleatorios, generados en el rango [1,*NP*], mutuamente diferentes y distintos a i = 1,...,*NP*; $\boldsymbol{x}_{r1,g}$, $\boldsymbol{x}_{r2,g}$, $\boldsymbol{x}_{r3,g}$ son los vectores objetivo correspondientes a dichos índices; $\boldsymbol{x}_{best,g}$ es el vector con mejor valor de la función objetivo en la población de la generación g; y ***F*** es un factor de escala que controla la tasa de mutación en el rango [0.4, 1].

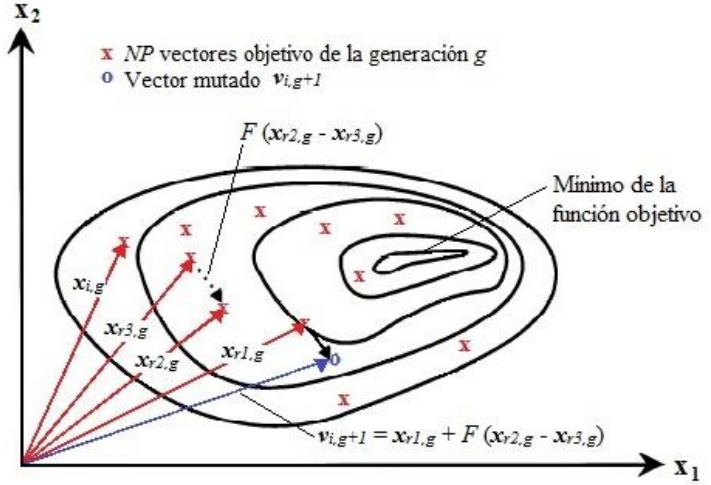

Figura 2.4. Proceso de mutación para el caso de una función objetivo bidimensional [15]

La **recombinación** (crossover, en inglés) consiste en comparar cada variable del vector mutado ya generado $\boldsymbol{v}_{i,g+1}$ con su correspondiente vector objetivo $\boldsymbol{x}_{i,g}$ para generar un vector de prueba $\boldsymbol{u}_{i,g+1} = \{u_{1i,g+1},...,u_{ji,g+1},...,u_{Di,g+1}\}$ (Figura 2.5) de la siguiente manera:

$$u_{ji,g+1} = \begin{cases} v_{ji,g+1}, & si(rand_j[0,1] \leq CR) \text{ ó } (j = j_{rand}) \\ x_{ji,g}, & en\ otro\ caso \end{cases}, \qquad (8)$$

para $i = 1,...,NP$, $j = 1,...,D$, $\boldsymbol{CR}$ es una constante que controla la tasa de recombinación en el rango [0,1] y j_{rand} es un índice entero aleatorio elegido en el rango [1,D] para asegurar que el vector de prueba sea la mejor combinación de los vectores objetivo y mutado (crossover no uniforme).

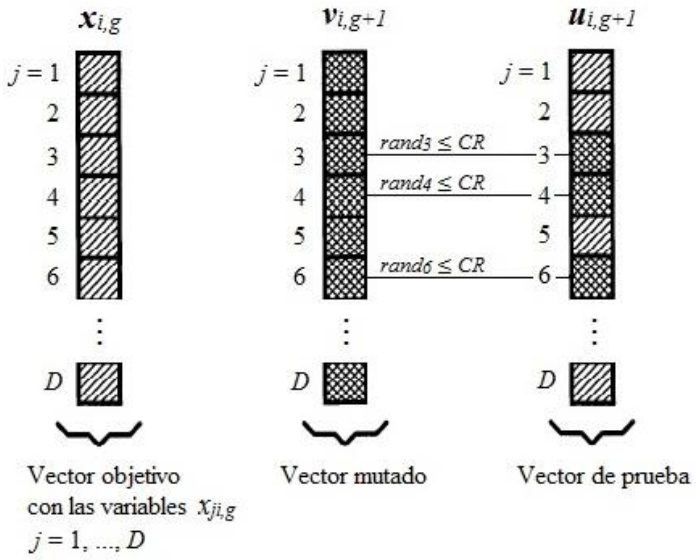

Figura 2.5. Proceso de recombinación [15]

Finalmente, la **selección** se realiza comparando el valor de la función objetivo de cada vector de prueba $f(\boldsymbol{u}_{i,g+1})$ con su correspondiente vector objetivo $f(\boldsymbol{x}_{i,g})$ en la población actual. El vector que pasará a formar parte de la siguiente generación será aquel que tenga el mejor valor de la función objetivo.

$$\boldsymbol{x}_{i,g+1} = \begin{cases} \boldsymbol{u}_{i,g+1}, & si\ f(\boldsymbol{u}_{i,g+1}) < f(\boldsymbol{x}_{i,g}) \\ \boldsymbol{x}_{i,g}, & en\ otro\ caso \end{cases}, \qquad (9)$$

Las últimas tres operaciones son repetidas de generación en generación hasta que sea satisfecho un criterio de detención específico (número de generaciones, tiempo transcurrido, o calidad de solución alcanzada, entre otras).

Por todo ello, el algoritmo DE es una versión mejorada de GA que resulta muy interesante de implementar en MATLAB debido a su facilidad de implementación, así como por su eficiencia y robustez en la resolución de problemas de optimización con múltiples objetivos.

2.3. Sintonía de controladores PI$^{\lambda}$D$^{\mu}$ fraccionarios mediante DE

En este apartado se propone una estrategia basada en el algoritmo DE para estimar los parámetros de un controlador PI$^{\lambda}$D$^{\mu}$ fraccionario. El desarrollo de este método se puede encontrar en el trabajo [5].

Como se acaba de explicar en el apartado anterior, DE es un método de optimización evolutiva basado en la minimización de una función objetivo, la cual suele estar contemplada en el dominio del tiempo. Sin embargo, el enfoque que se propone en este trabajo, también permite al usuario especificar múltiples restricciones en el dominio de la frecuencia.

A continuación se describe el algoritmo evolutivo que se aplica a la sintonización de los controladores PI$^{\lambda}$D$^{\mu}$ fraccionarios, así como las dos funciones de coste que se utilizarán para cumplir con las especificaciones de control en los dominios del tiempo y de la frecuencia, respectivamente.

2.3.1. Algoritmo de Evolución Diferencial

Aunque este algoritmo ya se ha descrito de forma global en la sección 2.2, en este apartado se pretende dar una explicación más detallada a partir del trabajo [5]. La actuación del algoritmo DE implementado se muestra en la Figura 2.6.

El algoritmo DE que se va a utilizar para sintonizar controladores PI$^{\lambda}$D$^{\mu}$ fraccionarios involucra a una población de *NP* soluciones candidatas, donde cada una se corresponde con un controlador fraccionario que tiene cinco parámetros a ajustar (tres ganancias y dos órdenes fraccionarios):

$$\boldsymbol{pop}_{i,k} = \left(k_p, k_i, k_d, \lambda, \mu\right), \qquad (10)$$

donde $\boldsymbol{pop}_{i,k}$ representa el elemento *i* en la iteración *k*.

La población inicial se genera aleatoriamente en intervalos predefinidos que limitan el espacio de búsqueda (línea 2, Figura 2.6). Además se debe calcular el valor de la función de coste para cada uno de los posibles controladores (línea 3), puesto que dicha función incluye las especificaciones a optimizar.

El proceso de optimización comienza en la línea 5 con un bucle que termina cuando se alcanza el límite superior de iteraciones o cuando se cumple una de las condiciones de convergencia.

El mecanismo evolutivo que incluye las etapas de mutación, recombinación y selección, se aplica a toda la población mediante un bucle (línea 6), para obtener la población de la siguiente generación (k+1).

Primero tiene lugar la etapa de mutación (línea 7), donde se genera un vector mutado por cada candidato de la población actual:

$$\boldsymbol{v}_{i,k} = \boldsymbol{pop}_{a,k} + F \cdot \left(\boldsymbol{pop}_{b,k} - \boldsymbol{pop}_{c,k}\right), \qquad (11)$$

donde $\boldsymbol{pop}_{a,k}$, $\boldsymbol{pop}_{b,k}$ y $\boldsymbol{pop}_{c,k}$ son tres de los elementos escogidos al azar en la iteración k y los índices a, b y c son mutuamente diferentes y distintos a i. F es el factor que controla la amplificación de las variaciones diferenciales entre [0.4, 1].

Después, la etapa de recombinación (líneas 8-11) incrementa la diversidad de la nueva generación mediante un vector de prueba $\boldsymbol{u}_{i,k}$ creado a partir de $\boldsymbol{v}_{i,k}$ y $\boldsymbol{pop}_{i,k}$ dependiendo de la probabilidad de recombinación (δ):

$$u_{ji,k} = \begin{cases} v_{ji,k}, & si\ p_{ji,k} < \delta \\ pop_{ji,k}, & en\ otro\ caso' \end{cases} \qquad (12)$$

donde $p_{ji,k}$ es un valor elegido al azar en el rango [0,1] para cada parámetro j del elemento i de la población en la iteración k.

Finalmente, la etapa de selección (líneas 12-17) determina el conjunto de la población de la siguiente generación (k+1) comparando el valor de la función de coste de los vectores $\boldsymbol{u}_{i,k}$ y $\boldsymbol{pop}_{i,k}$:

$$\boldsymbol{pop}_{i,k+1} = \begin{cases} \boldsymbol{u}_{i,k}, & si\ e_{i,k} < e_{i,k-1} \\ \boldsymbol{pop}_{i,k}, & en\ otro\ caso' \end{cases} \qquad (13)$$

donde $e_{i,k-1}$ es el valor de la función de coste del candidato actual y $e_{i,k}$ representa el valor de la función de coste del vector de prueba.

Tras la convergencia o cuando se alcanza un número máximo de iteraciones, el algoritmo devuelve el mejor elemento de la población, que se corresponde con el controlador fraccionario que minimiza la función de coste (líneas 19-20).

La función de coste se diseñará de acuerdo con los objetivos que se deben satisfacer en los dominios del tiempo y de la frecuencia.

```
1:  for i = 1: NP do
2:      pop_i,1 ← init_pop(pop_limits)          ▷Primera generación de población
3:      e_i,0 ← fitness(plant, pop_i,1)         ▷Cálculo de la función de coste
4:  end for
5:  for k = 1: max do
6:      for i = 1: NP do
7:          v_i,k = pop_a,k + F(pop_b,k - pop_c,k)        ▷Mutación
8:          for j = 1: D do
9:              u_ji,k = v_ji,k, ∀p_ji,k < δ              ▷Recombinación
10:             u_ji,k = pop_ji,k, ∀ p_ji,k ≥ δ
11:         end for
12:         e_i,k ← fitness(plant, pop_i,k)   ▷Nuevo cálculo de la función de coste
13:         if e_i,k < e_i,k-1 then                       ▷Selección
14:             pop_i,k+1 = u_i,k
15:         else
16:             pop_i,k+1 = pop_i,k
17:         end if
18:     end for
19:     ind_best ← min(e_k)
20:     bestmem ← pop_k(ind_best)
21:     if convergence = true then    ▷Se detiene la ejecución tras la convergencia
22:         exit(bestmem)                             ▷Devuelve la mejor estimación
23:     end if
24: end for
```

Figura 2.6. Algoritmo DE para sintonizar controladores $PI^{\lambda}D^{\mu}$ [5]

2.3.2. Funciones de coste

Según los enfoques tradicionales, para sintonizar un controlador $PI^{\lambda}D^{\mu}$ fraccionario, solo se podrían satisfacer cinco condiciones diferentes [17], que se corresponden con los cinco parámetros de ajuste.

Sin embargo, el algoritmo DE proporciona una gran flexibilidad en la sintonización de estos controladores ya que permite satisfacer múltiples especificaciones de diseño. De modo que, si dichas especificaciones son incluidas adecuadamente en una función objetivo, este método es capaz de ofrecer una buena solución de optimización.

En este caso, se definirán dos funciones de coste diferentes basándose en especificaciones de tiempo y frecuencia, respectivamente:

- La primera es una función que minimiza el error entre la salida y la entrada de referencia en el dominio del tiempo.
- La segunda es una función objetivo que permite introducir múltiples especificaciones en el dominio de la frecuencia, incluyendo robustez a variaciones en la ganancia de la planta.

Más adelante se discuten las ventajas e inconvenientes de cada opción.

2.3.2.1. Sintonía en el dominio del tiempo

Para diseñar un controlador en el dominio del tiempo, la variable que se suele considerar es el tiempo de respuesta, el cual se obtiene midiendo el error entre la salida y la entrada de referencia:

$$fitness(i) = \sum_{t=t_i}^{t_f} e_{TD}^2(t) = \sum_{t=t_i}^{t_f} [u(t) - y(t)]^2, \qquad (14)$$

donde $e_{TD}(t)$ es el error de la señal en el dominio del tiempo calculado en un intervalo finito en tiempo discreto, $u(t)$ es la señal de referencia (salida ideal que se desea obtener) e $y(t)$ es la salida de la planta.

La minimización de esta función de coste implica que el tiempo de respuesta será el más cercano posible al ideal (teniendo en cuenta la limitación de la acción de control). Por tanto, si el método de optimización tiene éxito, se espera que las características más típicas de este enfoque de control (tiempo de pico, tiempo de establecimiento, sobreoscilación) presentarán buenos valores.

Sin embargo, hay muchos otros factores que no están incluidos en este tipo de control (robustez a variaciones en la ganancia de la planta), los cuales pueden considerarse en el dominio de la frecuencia como se propone a continuación.

2.3.2.2. Sintonía en el dominio de la frecuencia

Para diseñar un controlador en el dominio de la frecuencia, se pueden cumplir muchas condiciones diferentes que, gracias al algoritmo DE, serán incluidas en la función objetivo.

Una de las especificaciones más comunes para los controladores PID es la robustez a variaciones en la ganancia de la planta [18], que puede obtenerse si se cumple la siguiente condición:

$$\left.\frac{d(\arg(F(s)))}{d\omega}\right|_{\omega=\omega_{cg}} = 0, \qquad (15)$$

donde ω_{cg} es la frecuencia de ganancia de cruce y $F(s) = C(s)\ G(s)$ es la función de transferencia de lazo abierto, siendo $G(s)$ la función de transferencia de la planta. Esto significa que la fase del sistema en lazo abierto será plana en ω_{cg} y casi constante en un intervalo alrededor de ω_{cg}. El sistema será más robusto ante cambios y la sobreoscilación de la respuesta es casi constante en un rango de ganancias (propiedad de iso-amortiguación de la respuesta en el tiempo).

Así, la función de coste vendrá dada por la pendiente de la fase de la respuesta en lazo abierto alrededor de ω_{cg}:

$$fitness(i) = \sum_{\omega=\omega_i}^{\omega_f} e_{FD}^2(\omega) = (\varphi(\omega) - (\varphi_m - \pi))^2, \qquad (16)$$

donde el error $e_{FD}(\omega)$ se calcula en un intervalo predefinido entre ω_i y ω_f alrededor de ω_{cg}, $\varphi(\omega)$ es la fase del sistema en una frecuencia ω. Con el método DE se puede fijar el intervalo de ganancias con una respuesta plana.

Además, en esta función de coste se incluyen más condiciones para satisfacer las mismas especificaciones que se describen en [17]:

- Intervalo de aceptación para la frecuencia de ganancia de cruce (ω_{cg}) y margen de fase mínimo (φ_m). Estas variables tienen una gran influencia en la robustez del sistema, y se calculan a través de las siguientes ecuaciones:

$$\left|C(j\omega_{cg})G(j\omega_{cg})\right|_{dB} = 0\ dB, \qquad (17)$$

$$\arg\left(C(j\omega_{cg})G(j\omega_{cg})\right) = -\pi + \varphi_m. \qquad (18)$$

- Rechazo de ruido a altas frecuencias ($\omega \geq \omega_t$). El objetivo es atenuar el ruido en frecuencias altas, de modo que la solución candidata se descarta si no se cumple la ecuación 20. El valor de la atenuación A viene dado por la siguiente función de sensibilidad complementaria:

$$\left|T(j\omega) = \frac{C(j\omega)G(j\omega)}{1 + C(j\omega)G(j\omega)}\right|_{dB} = A\ dB, \qquad (19)$$

$$\forall\omega \geq \omega_t\ rad/s \Rightarrow |T(j\omega_t)|_{dB} \leq A\ dB, \qquad (20)$$

- Sensibilidad a bajas frecuencias ($\omega \leq \omega_s$). El objetivo es garantizar un buen rechazo de perturbaciones de salida, de modo que la solución candidata se descarta si no se cumple la ecuación 22. El valor de la sensibilidad B viene dado por la siguiente función de sensibilidad:

$$\left|S(j\omega) = \frac{1}{1 + C(j\omega)G(j\omega)}\right|_{dB} = B\ dB, \qquad (21)$$

$$\forall \omega \geq \omega_s\ rad/s \Rightarrow |S(j\omega_s)|_{dB} \leq B\ dB, \qquad (22)$$

- Cancelación del error de estado estacionario. Esta condición siempre se va a cumplir en controladores fraccionarios porque el integrador fraccionario es tan eficiente como uno de orden entero [17].

En el dominio de la frecuencia es posible incluir muchas condiciones adicionales siempre que se puedan incorporar en la función de coste. Por tanto, el valor de la función de coste para este enfoque de control se calcula a través de la ecuación 16, pero se incluyen las restricciones para determinar valores que cumplan las distintas condiciones. Si el sistema de control no cumple una de estas condiciones, se descarta la posible solución.

3. DISEÑO DE LA INTERFAZ GRÁFICA EN MATLAB

Este capítulo está estructurado en cuatro secciones, donde se recogen los aspectos más importantes del entorno MATLAB. Y más ampliamente, en la sección 3.4, se describe su herramienta GUIDE para la creación de interfaces gráficas.

3.1. Introducción

MATLAB® es un lenguaje de alto nivel y un entorno interactivo para el cálculo numérico, la visualización y la programación. Mediante MATLAB, es posible analizar datos, desarrollar algoritmos y crear modelos o aplicaciones. El lenguaje, las herramientas y las funciones matemáticas incorporadas permiten explorar diversos enfoques y llegar a una solución antes que con hojas de cálculo o lenguajes de programación tradicionales, como pueden ser C/C++ o Java™ [19].

Así, es usado en una amplia variedad de aplicaciones, incluyendo procesamiento de señal e imagen, comunicaciones, diseño de control, biología computacional, análisis financiero, estadística, álgebra lineal, etc. Existen diversas toolboxes específicamente diseñadas para las distintas áreas de aplicación, que contienen funciones especiales para resolver los problemas propios de estas [20].

El nombre MATLAB deriva del término inglés 'matrix laboratory', ya que la principal característica de MATLAB es que permite manipular de manera sencilla y eficiente vectores y matrices [19].

El sistema MATLAB está constituido por estas partes principales [20]:

- Herramientas del escritorio y Entorno de desarrollo: esta parte de MATLAB es el conjunto de herramientas que facilitan el trabajo con archivos y funciones. Muchas de estas herramientas son interfaces gráficas de usuario. Éstas incluyen: el escritorio MATLAB y la ventana de comandos, un editor y un depurador de código, así como exploradores para visualizar el contenido de ayuda, el entorno de trabajo y las carpetas.

- Librería de funciones matemáticas: esta librería es una amplia colección de algoritmos computacionales que van desde funciones elementales como suma, seno, coseno y aritmética compleja, hasta funciones más sofisticadas como la matriz inversa, cálculo de los autovalores de una matriz, funciones de Bessel o la transformada rápida de Fourier.

- El lenguaje: el lenguaje MATLAB es un lenguaje de alto nivel construido con matrices y vectores, con sentencias de control de flujo, funciones, estructuras de datos, y funcionalidades de entrada y salida y programación orientada a objetos.

- Gráficos: MATLAB cuenta con amplias instalaciones para la visualización de vectores y matrices como gráficos, además permite guardar e imprimir cada uno ellos. Incluye un alto nivel de funciones para la visualización de datos en dos dimensiones y en tres dimensiones, gráficos de procesamiento de imágenes, animación y presentación. También contiene funciones de bajo nivel que permiten personalizar totalmente la apariencia de gráficos, así como construir interfaces gráficas de usuario para aplicaciones de MATLAB.

- Interfaces externas: La biblioteca de interfaz externa permite escribir programas en C y Fortran. Esta incluye instalaciones para llamar a rutinas de MATLAB (enlace dinámico), utilizar a MATLAB como motor de cálculo, y para la lectura y la escritura MAT-archivos.

A continuación se explican algunos aspectos fundamentales del lenguaje Matlab, para facilitar la interpretación de la programación de los apartados 2 y 4. También se detallan algunos aspectos generales sobre la herramienta GUIDE usada para crear la interfaz.

3.2. Variables en MATLAB

MATLAB permite asignar nombres variables a valores numéricos. Los nombres de las variables deben empezar con una letra seguida de una serie de caracteres alfanuméricos. Una apreciación sobre este lenguaje es que distingue entre letras mayúsculas y minúsculas. Los nombres de las variables pueden tener toda la longitud que se quiera, pero MATLAB sólo usará los primeros N caracteres e ignorará el resto. Es importante dentro asegurarse que dentro de una misma función dos variables no estén definidas con el mismo nombre.

La sentencia para la creación de la variable es de la forma:

$$variable = expresion$$

En "expresion" pueden aparecer principalmente: matrices, valores numéricos, otras variables y operadores.

Las **matrices numéricas** rectangulares están formadas por números reales o complejos. De este modo, un escalar en MATLAB es una matriz de dimensiones 1x1, y un vector es una matriz de una única fila o columna [19].

El siguiente ejemplo muestra la declaración de una matriz:

$$Matriz = [1\ 2\ 3;4\ 5\ 6]$$

$$Matriz = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Una **matriz de tipo estructura** es un conjunto heterogéneo de datos distribuidos en distintos grupos llamados campos. Para crear un nuevo campo, o modificar el valor de uno existente se utiliza la siguiente sentencia:

estructura.campoX = "nuevo_texto"

Dentro de las **variables de tipo numérico**, existen diferentes tipos, se pueden destacar datos de tipo flotante (float), entero (int), de doble precisión (double), etc…

También existen **variables de texto**. Para asignar el valor a la variable se debe introducir el texto entre comillas, quedando su declaración de la siguiente manera:

variable_de_texto = "texto"

Este tipo de variables son en realidad un vector fila, quedando almacenado cada carácter de texto en una componente del vector. Las variables de texto se pueden convertir en numéricas. En función de la precisión con la que la que se quiere realizar la conversión tenemos, por un lado la sentencia *str2num* y para una mayor precisión en su resultado tenemos la sentencia *str2double* .Para el proceso inverso donde se convierte una variable numérica en una variable de texto, solo existe una única sentencias *num2str*.

Por defecto, MATLAB utiliza la doble precisión para almacenar variables numéricas.

En función de la operación que se desee realizar tenemos por una parte, para expresar condición los operadores relacionales: igual "==", menor o igual "<=", mayor o igual ">=", menor "<", mayor ">", distinto "~=", y los operadores lógicos: AND "&", NOT "~" y OR "|" y por otra parte tenemos los operadores aritméticos ya conocidos.

3.2.1. Tipos de variables

En función del uso que se le quiera dar a la variable dentro del programa, MATLAB divide las variables en tres tipos de: locales, globales y persistentes [2].

Variables locales: se trata de todas las variables que son definidas dentro de cada función, estas no permanecen en memoria una vez que se llama a una nueva función.

Variables globales: se trata de variables que son compartidas por diferentes funciones, para ello en cada función donde se quiere usar, se debe declarar la variable con el prefijo GLOBAL. Para identificarlas mejor es recomendable escribirlas en letras mayúsculas.

Variables persistentes: se trata de variables que sólo pueden ser declaradas en funciones, y sólo tendrán acceso a ellas las funciones donde han sido declaradas. Cuando termina la ejecución de una función, MATLAB no borra estas variables de la memoria, así su valor persiste en las sucesivas llamadas a la función.

3.2.2. Alcance de una variable

MATLAB almacena las variables en un espacio de memoria llamado *workspace* o espacio de trabajo. Las variables creadas de manera interactiva y las creadas al ejecutar scripts se almacenan en el *espacio base de trabajo*. Pero las funciones no usan este *espacio base de trabajo*, porque cada función tiene su propio espacio de trabajo. A continuación se muestran algunas de las opciones para ampliar el alcance de una variable:

1. Usar variables globales

2. Pasar la variable a la función que necesita usarla, como un argumento extra en la llamada a la función.

3. Uso de funciones anidadas: el alcance de una variable local se extiende a todas las funciones que estén anidadas dentro de ella.

Para eliminar los valores de variables almacenadas en el espacio de trabajo, se emplea el comando *clear* seguido del nombre de la variable. Para eliminar los valores de todas las variables, se emplea el comando *clear all*.

3.3. Programación con MATLAB

La programación con MATLAB es muy similar a la realizada con otros lenguajes de programación. Podemos encontrar las sentencias habituales para el control condicional (sentencias *if*/ *else*/ *elseif*, y sentencias *switch*/ *case*/ *otherwise*) y el control por ciclos (*for*, *while*, *continue* y *break*).

3.3.1. Tipos de archivos

Aunque MATLAB puede usarse de un modo interactivo, su utilización más habitual consiste en la ejecución de secuencias de sentencias que deberán almacenarse en archivos. En este sentido trabaja de manera similar a cualquier lenguaje de programación [19].

Estos archivos se conocen como *M-File* y tienen extensión "*.m*". Son compilados de manera automática durante su primera ejecución. Hay dos tipos de ficheros *M-File*: los *ficheros script* y las *funciones*.

Los **ficheros script** consisten en una serie de sentencias y comandos de MATLAB. Cuando se ejecuta un fichero script, todas las variables utilizadas en el mismo quedan almacenadas en memoria durante la sesión actual de trabajo.

Las **funciones** deben siempre comenzar con la siguiente sintaxis:

function [argumentos de salida] = nombre_funcion (argumentos de entrada)

Otro requisito que debe cumplir la función, es que el nombre del archivo debe ser igual que el nombre de la función en la declaración de la primera línea. En el ejemplo de arriba, el archivo se llamaría "*nombre_funcion.m*".

La diferencia principal entre una función y un script es que las variables que se definen y manipulan dentro de la función sólo permanecen en memoria mientras dura la ejecución de la misma; por tanto, las únicas variables que permanecen en memoria en la sesión de trabajo son los argumentos de entrada y salida.

Para interrumpir la ejecución de un script o una función, se emplea la sentencia *return*.

Los archivos de tipo función pueden contener funciones adicionales con nombres diferentes, que se irán escribiendo unas a continuación de otras. La primera función es la *función principal* y las demás son **subfunciones**. Cada subfunción debe comenzar con su propia sentencia de declaración de función.

3.3.2. Sintaxis de las funciones

En la declaración de una función, si no hay argumentos de salida se pueden omitir los corchetes y el signo "=". Si sólo hay un argumento de entrada no es necesario corchetes, y si no hay argumentos de entrada no es necesario poner el paréntesis.

Si el número de argumentos de la función es variable, se pueden emplear las variables *varargin* y *varargout*.

function varargout = nombre_funcion (varargin)

La variable *varargin* es una matriz de celdas que contiene todos los argumentos opcionales de la función. Recoge todos los argumentos que se introducen al producirse la llamada a la función. La variable *varargout* es una matriz de celdas que recoge todos los argumentos de salida a los que se haya asignado valores dentro de la función.

Siguiendo con el ejemplo anterior, podríamos extraer las variables que nos interesan de la matriz *varargin* del siguiente modo.

[variable_1, variable_2] = varargin {[2,5]};

Y podemos asignar valores a las variables de la matriz *varargout* que consideremos.

varargout {1} = resultado;

También existe un modelo de función utilizada en la programación de la interfaz, su sentencia es la siguiente:

function pushbutton1_Callback(hObject,eventdata,handles)

Es un tipo de función usada para declarar el funcionamiento tras ejecutar una acción a través de un uicontrol, tienen los siguientes argumentos de entrada estándar:

hObject: se trata del componente de interfaz gráfica de usuario, por lo que se activó la devolución de llamada. Para un grupo de botones de *SelectionChangeFcn* devolución de llamada, *hObject* es la manija del botón de opción seleccionado o botón de alternar.

eventdata: secuencias de eventos provocados por las acciones del usuario, tales como selecciones.

handles: estructura de MATLAB que contiene todos los objetos de la interfaz gráfica de usuario, y también puede contener datos definidos por la aplicación.

3.4. GUIDE

Con la intención de crear una interfaz gráfica de usuario, del inglés *GUI* (*"Graphical user interface"*) se recurre a una aplicación propia de MATLAB denominada GUIDE (GUI Development Environment), es una herramienta interactiva que facilita el diseño y la construcción de esta.

Al comenzar el proceso de creación de una GUI mediante esta herramienta, aparece una figura que podemos ir poblando de controles y objetos por medio de un editor gráfico. GUIDE crea un archivo de código asociado a esta figura, que contiene los *callbacks* de la GUI y sus componentes. GUIDE guarda ambos, la figura (como un archivo *.fig*), y el código (como un archivo *.m*). Al abrir uno de ellos se abre el otro para arrancar la GUI.

3.4.1. Alternativa para construir un GUI

Podemos decir que existe una alternativa para crear una GUI en MATLAB, mediante programación.

Se crea crear un archivo de código en el que se definen todos los elementos y comportamientos de la GUI. Al ejecutar el archivo, se crea una figura en la que aparecen todos los elementos programados.

Normalmente, la figura no se guarda entre sesiones porque el programa crea una nueva cada vez que se arranca el programa.

Este procedimiento resulta ser una menos práctico ya que se deben definir todas las propiedades de la figura y sus controles, así como los *callbacks*, cosa que la aplicación GUIDE ya los tiene definidos internamente.

3.4.2. Componentes que forman la GUI

En una primera aproximación, se puede decir que una *GUI* está basada en objetos gráficos. El objeto principal es una figura. De hecho, se puede decir que en MATLAB, una *GUI* es en sí una figura. Las figuras pueden contener los siguientes objetos gráficos [19]:

- **Objetos *axes*:** definen una región dentro de la figura, en la cual se representarán imágenes, dibujos en dos o tres dimensiones, se escribirán textos, etc.

- **Objetos *uicontrol*:** abreviatura del término inglés, *"user interface control"*. Son cajas de texto y botones que permiten ejecutar una determinada acción, programada por el usuario, cuando se activan con el ratón.

- **Objetos *uimenu*:** abreviatura del término inglés *"user interface menu"*. Son los menús creados por el usuario que se añaden al menú ya existente en la parte superior de la figura.

- **Objetos *uicontextmenu*:** abreviatura del término inglés *"user interface context menu"*. Son menús que se activan con el botón derecho del ratón y que pueden asociarse a cualquiera de los objetos anteriores.

Para diseñar la GUI de una manera más organizada se recurre a utilizar paneles, los cuales agrupan los distintos objetos gráficos, otros paneles, etc., esto se hace mediante la **función *uipanel***.

3.4.3. Propiedades de los objetos gráficos

Los objetos gráficos tienen una serie de propiedades que determinan su aspecto en la pantalla. Algunas de las más importantes son su posición en la jerarquía de objetos gráficos (*parent*, *children*), su visibilidad (*on*, *off*), el color, la posición, etc. Si no se indican otros valores de manera explícita, estas propiedades tomarán los valores que el objeto gráfico tenga definidos por defecto.

A continuación se indican algunas funciones necesarias para interaccionar con las propiedades de los objetos gráficos [19]:

- get (h): donde "*h*" es el *handle* del objeto gráfico, permite obtener un listado de todas las propiedades del objeto gráfico.
- set (h): proporciona un listado por pantalla de todas las propiedades del objeto con los posibles valores que puede tomar cada propiedad.
- get (h,"propiedad"): permite conocer el valor de la propiedad "*propiedad*" del objeto "*h*".
- set (h,"propiedad"): permite conocer los valores que puede tomar la propiedad "*propiedad*" del objeto "*h*".
- set (h,"propiedad", nuevovalor): permite asignar el valor "*nuevovalor*", a la propiedad "*propiedad*" del objeto "*h*".

3.4.4. Uicontrols

Como se comentó antes, los objetos *uicontrol* son cajas de texto y botones que permiten ejecutar acciones programadas por el usuario, cuando se activan con el ratón. Distinguimos los siguientes tipos de *uicontrol* [19]:

- *Push button*: al hacer clic en ellos se ejecuta una determinada acción.
- *Toggle button*
- *Radio button*
- *Checkbox*
- *Edit text*: cajas de texto editables.
- *Static text*: cajas de texto no editables.
- *Slider*: permite ejecutar acciones dependientes de un parámetro numérico cuyo valor varía al trasladar el botón a lo largo de una barra rectangular por medio del ratón.
- *Listbox*
- *Popup menu*
- *Frame*: marco cuya función es agrupar grupos de botones.

Los *uicontrols* de tipo *Toggle button*, *Radio button* y *Checkbox* permiten realizar una acción al hacer clic en ellos. Esta acción se mantiene hasta que se vuelve a hacer clic de nuevo en el mismo. Suelen emplearse para elegir entre opciones.

Los de tipo *Listbox* y *Popup menu*, muestran una lista de alternativas que harán que se ejecute una determinada acción programada al seleccionar una de ellas con el ratón.

Se pueden crear *uicontrols* con la **función *uicontrol***, cuya sintaxis es la siguiente:

h1 = uicontrol (identificador_padre,"prop1","valor1","prop2","valor2",...)

donde *identificador_padre* es el identificador de la figura a la que se añade el *uicontrol*.

Las **propiedades** más relevantes que presentan los ***uicontrols*** son las siguientes:

- *BackgroundColor*: es el color del objeto.
- *Callback*: especifica la acción a realizar por MATLAB al actuar sobre el botón con el ratón.
- *Enable*: permite desactivar el *uicontrol* de modo que no sea posible actuar sobre él con el ratón. Toma valores *on* y *off*.
- *Position*: determina posición y tamaño del *uicontrol* dentro de la figura. Se define mediante un vector de cuatro elementos: [x y anchura altura]. La propiedad "*units*" establece las unidades en que se mide la posición: *inches*, *centimeters*, *normalized*, *points*, *pixels* y *characters*.
- *String*: es el texto que aparece en el *uicontrol*. El tamaño, tipo, color y grosor de la fuente, el ángulo de la letra, pueden modificarse mediante las propiedades *FontSize*, *FontName*, *ForegroundColor*, *FontWeight* y *FontAngle* respectivamente.
- *Style*: es el tipo de *uicontrol*. Anteriormente hemos visto los distintos tipos que podemos encontrar: *pushbutton*, *togglebutton*, *radiobutton*, *checkbox*, *edit*, *text*, *slider*, *frame*, *listbox* y *popupmenu*.
- *Value*: en los *uicontrols* de tipo *Checkbox*, *Radio Button* y *Toggle Button* toma el valor "1" al estar seleccionado, y el valor "0" en caso contrario.
- *TooltipString*: permite introducir un comentario que se mostrará en un recuadro amarillo al acercarnos con el ratón al *uicontrol*.

4. PROGRAMACIÓN DE LA INTERFAZ

En este capítulo se explica en detalle cómo se ha programado el diseño de la interfaz, y está estructurado en dos secciones: en la sección 4.1 se describe la estructura general de la interfaz, mientras que en la sección 4.2 se analizan cada una de las funciones utilizadas para su programación.

4.1. Introducción

Debido a la complejidad de un primer diseño de la interfaz y por los conocimientos previos de otros lenguajes, en este proyecto fin de carrera se opta por la opción programática, dejando a un lado la otra opción que está más encaminada a interfaces sencillas.

Para facilitar la explicación de la programación, se divide el programa en dos bloques generales, la Figura 4.1 muestra los archivos que componen cada uno de ellos.

Archivos generales	
main.m	fdt.jpg
uitabpanel.m	domains.jpg
maximize.m	pid.jpg
uc3m.jpg	Help.pdf

Archivos sintonía PI$^{\lambda}$D$^{\mu}$ dominio del tiempo (*tuning = 1*)	*Archivos sintonía PI$^{\lambda}$D$^{\mu}$ dominio de la frecuencia* (*tuning = 3*)
Control_Fraccionario.m	
cost.m	
aprox_pidfrac_to_pid	
Control_Fraccionario_Demo.mdl	

Figura 4.1. Estructura general del programa

4.2. Archivos generales

Entrando en el detalle de los archivos generales, se pueden distinguir en primer lugar el principal archivo del programa *main.m*. Se trata de un archivo compuesto por una función principal y unas subfunciones. En la Figura 4.2 se muestra un esquema general.

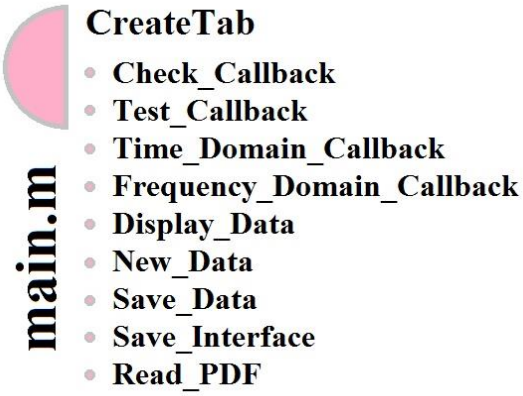

Figura 4.2. Esquema general de la función main

Con el objetivo de dar a la interfaz un aspecto más complejo, se recurre a dos funciones externas al programa, contenidas dentro de los archivos *uitabpanel.m* y *maximize.m*, ambas se detallan en el apartado 4.2.1 y 4.2.2.

Los últimos archivos tienen poca importancia a nivel de programación, son complemento para el diseño de la interfaz, serán llamados desde la función principal. Las figuras 4.3, 4.4, 4.5, 4.6 y 4.7 muestran su contenido:

Figura 4.3. Archivo *uc3m.jpg*

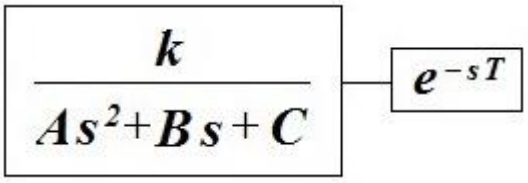

Figura 4.4. Archivo *fdt.jpg*

$$fitness(i) = \sum_{t=t_i}^{t_f} e_{TD}^2(t) \qquad fitness(i) = \sum_{w=w_i}^{w_f} e_{FD}^2(w)$$

Figura 4.5. Archivo *domains.jpg*

$$C(s) = k_p + \frac{k_i}{s^{\lambda}} + k_d s^{\mu}$$

Figura 4.6. Archivo *pid.jpg*

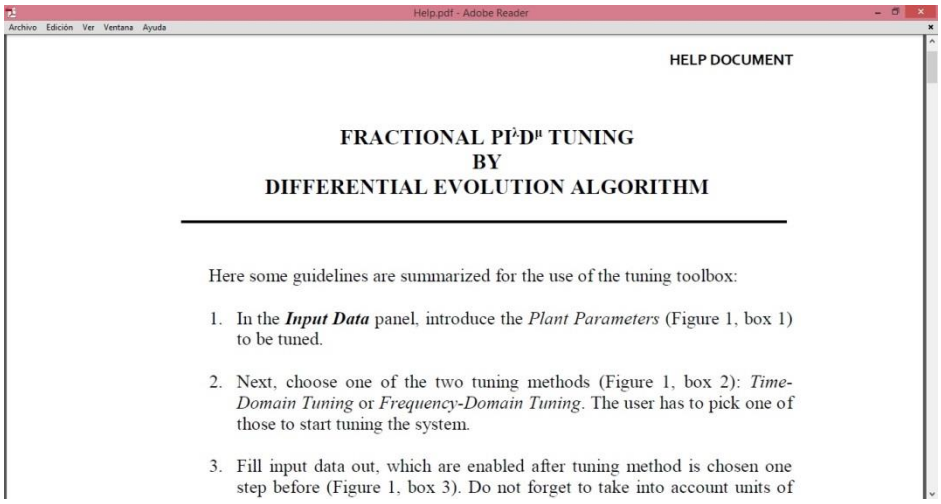

Figura 4.7. Archivo *Help.pdf*

4.2.1. Función uitabpanel

Con vistas a mejoras futuras del trabajo, se plantea crear diferentes pestañas dentro de la interfaz. Al no estar disponible esa opción en guide es necesario recurrir a una función externa uitabpanel. Esta permite crear pestañas y colocarlas en la posición deseada mediante el parámetro 'TabPosition'.

Esta función es llamada dentro de la función main, para poder crear esta pestaña previamente se debe haber creado una figura interfaz sobre la que se van a pintar . Nuestra interfaz solo tiene una pestaña FRACTIONAL TUNING [21].

```
uitabpanel(...
  'Parent',interface,...
  'TabPosition','lefttop',...
  'Units','pixels','Position',[0,0,1*WIDTH,0.93*HEIGHT],...
  'PanelBorderType','line','Title',{' FRACTIONAL TUNING ',},...
  'BackGroundColor',[0.078,0.169,0.549],'CreateFcn',@CreateTab);
```

4.2.2. Función maximize

Esta función debido a su utilidad, ha sido recuperada de otro proyecto [22], igualmente puede descargarse de su correspondiente enlace [23].

Al hacer la llamada a esta función se maximiza la figura que se le pasa como parámetro de entrada. La sintaxis de la función es la siguiente:

maximize (figure)

Esta función es llamada desde *main* (ver apartado 4.2.3) en dos ocasiones: para maximizar la pantalla temporal de espera y, una vez cargados todos los controles de la interfaz, para maximizar la figura correspondiente a la pantalla principal.

4.2.3. Función main

A continuación se explican las funciones que componen el archivo principal *main.m*, para ver la programación entera consultar el apartado A1.3.

La función *main* es la encargada de construir la interfaz, está compuesta de una pantalla de espera (*loading*) y una pantalla principal (*interface*), la cual es invocada a través de la función anidada *CreateTab* (ver apartado 4.2.4). A continuación se ofrecen las primeras líneas de código donde se muestran algunos aspectos importantes que forman la interfaz.

En primer lugar se declaran todas las variables globales del programa. Se pueden destacar las variables EJE y S al ser variables de tipo estructura.

```
function main
  global WIDTH
  global HEIGHT
  global loading
  global interface
  global EJE
  global S
```

A continuación a través de la siguiente sentencia *get(0)* se pueden obtener todos los características de fábrica de MATLAB. Es nuestro caso, solo nos interesa los valores de las dimensiones de la pantalla, por lo que se concreta la sentencia con el parámetro *ScreenSize*. Como resultado, se muestra el siguiente array con los valores de la pantalla en la que se está ejecutando el programa [dist_desde_izqda dist_desde_margen_inf ancho_pantalla largo_pantalla].

Las variables *WIDTH* y *HEIGHT* han sido declaradas globales porque serán empleadas más adelante para permitir que el tamaño de los elementos de la interfaz se ajuste al de la pantalla correctamente.

```
screensize = get(0,'ScreenSize');
WIDTH = screensize (3);
HEIGHT = screensize (4);
```

Debido a la cantidad de objetos gráficos que debe cargar MATLAB, se crea una pantalla de espera, a través de la figura *loading*, mientras que el programa carga toda la interfaz. Esta pantalla tendrá el tamaño anteriormente guardado y el color definido para todo el programa. El mensaje de espera *'Loading interface...'* se indica mediante un uicontrol de tipo *text*.

```
loading = figure(...
  'Name','','NumberTitle','off','Menubar','none',...
  'Units','pixels','Position',[0,0,WIDTH,HEIGHT],...
  'Color',[0.078,0.169,0.549]);
uicontrol(...
  'Parent',loading,...
  'Units','normalized','Position',[0.1,0.5,0.8,0.1],...
  'Style','text','String','Loading interface...',...
  'FontName','courier new','FontSize',32,'FontWeight','bold',...
  'ForeGroundColor',[1 1 1],'FontAngle','normal',...
  'BackGroundColor',[0.078,0.169,0.549]);
```

La figura se maximiza mediante la función *maximize* [23] (ver apartado 4.2.2).

```
maximize(loading)
```

Cuando todos los objetos gráficos están cargados, se superpone una nueva figura *interface*. Esta figura no será visible hasta que finalice la función *CreateTab* (ver apartado 4.2.4) y se ejecute la sentencia *set* incluida en las tres últimas líneas de código. Las otras dos sentencias permiten que la figura se adapte a la pantalla y se cierre la pantalla de espera.

```
interface = figure(...
  'Visible','off',...
  'Name','','NumberTitle','off','Menubar','none',...
  'Units','normalized','Position',[0,0,1,1]);
function CreateTab(varargin) [...]
end
set(interface,'Visible','on')
maximize(interface)
delete(loading)
end
```

4.2.4. Función CreateTab

Esta es la función donde se declaran todos los objetos gráficos de la interfaz, y además contiene todas las subfunciones que se muestran en la Figura 4.2.

A continuación se explica un ejemplo de los uicontrols más usados en esta función, para más detalle se recomienda ver el anexo 1 (apartado A1.3) donde se encuentra la programación completa.

Con la intención de que la interfaz quede ordenada se recurre a organizarlo mediante paneles y subpaneles. A continuación se muestran los paneles generales de entrada (*input*) y salida (*output*), que son paneles padre del resto de subpaneles.

```
E.input_panel = uipanel(...
  'Parent',hpanel(1),...
  'Units','normalized','Position',[0.01 0.01 0.43 0.98],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[0.82,0.863,0.922]);
```

```
E.output_panel = uipanel(...
  'Parent',hpanel(1),...
  'Units','normalized','Position',[0.46 0.25 0.14 0.65],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[1,1,1]);
```

Por su parte, la sintaxis de programación para cargar una imagen se verá a través del caso en que se insertan las funciones de coste correspondientes a cada dominio:

- Primero se declara el panel donde se quiere localizar.

```
E.domain_panel = uipanel(...
  'Parent',E.input_panel,...
  'Units','normalized','Position',[0.03,0.58,0.94,0.16],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[1,1,1]);
```

- A continuación se destinan unos ejes para poder dibujarlo.

```
axes(...
  'Parent',E.domain_panel,...
  'Units','pixel','Position',[0,0,514,85]);
```

- Finalmente se carga la imagen mediante la sentencia *imread* e *imshow*.

```
domains=imread('domains.jpg');
imshow(domains),axis off,hold on
```

Por último, se mencionan los dos tipos de uicontrol más utilizados a lo largo del programa: el primero es el uicontrol que solo muestra texto (tipo *text*) y el segundo es el uicontrol que permite introducir texto (tipo *edit*).

```
E.gain_text = uicontrol(...
  'Parent',E.plant_panel,...
  'Units','normalized','Position',[0.60,0.80,0.35,0.10],...
  'Style','text','String','Gain, k:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');

E.gain_data = uicontrol(...
  'Parent',E.plant_panel,'Enable','on',...
  'Units','normalized','Position',[0.82,0.80,0.10,0.14],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
```

4.2.5. Función Check_Callback

Esta función se crea para poder controlar el uicontrol de tipo *checkbox*. A través del parámetro de entrada *hObject* se le pasa a la función el valor correspondiente de la acción realizada, en este caso el *check* marcado. Inicialmente ambos aparecen por defecto con un valor 0, es decir, desmarcados.

```
%Checkbox time-domain
E.domain_T = uicontrol(...
  'Parent',E.domain_panel,'Enable','on',...
  'Units','normalized','Position',[0.05,0.77,0.45,0.15],...
  'Style','checkbox','String','Time-Domain Tuning',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'Callback',@Check_Callback);
%Checkbox frequency-domain
E.domain_F = uicontrol(...
  'Parent',E.domain_panel,'Enable','on',...
  'Units','normalized','Position',[0.52,0.77,0.45,0.15],...
  'Style','checkbox','String','Frequency-Domain Tuning',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'Callback',@Check_Callback);
```

Al marcar el *check E.domain_T* su valor cambia a 1, entrando en el primer bucle condicional *if*. Automáticamente se pondrá a 0 el *check E.domain_F* y, por la lógica del bucle, se activarán los uicontrols correspondientes a los parámetros generales del método de sintonización, mientras que los correspondientes al dominio de la frecuencia permanecerán desactivados. Además, se habilitará el uicontrol del botón que ejecuta el programa, desde el que se llamará a una nueva función *Time_Domain_Callback* (ver apartado 4.2.7).

```
function Check_Callback(hObject, eventdata, handles)
  set(E.new_button,'Enable','off');
  if hObject == E.domain_T
    if get(E.domain_T,'Value') == 1
        set(E.domain_F,'Value',0);
        set(E.size_data,'Enable','on');
        set(E.limit_data,'Enable','on');
        set(E.gain_L_data,'Enable','on');
        set(E.gain_U_data,'Enable','on');
        set(E.exp_L_data,'Enable','on');
        set(E.exp_U_data,'Enable','on');
        set(E.parameter_F_data,'Enable','on');
        set(E.parameter_CR_data,'Enable','on');
        set(E.interval_data,'Enable','off');
        set(E.minwcg_data,'Enable','off');
        set(E.maxwcg_data,'Enable','off');
        set(E.phase_data,'Enable','off');
        set(E.high_F_data,'Enable','off');
        set(E.high_F_dB_data,'Enable','off');
        set(E.sensitivity_data,'Enable','off');
        set(E.sensitivity_dB_data,'Enable','off');
        set(E.new_button,'Enable','off');
        E.run_button = uicontrol (...
          'Parent',E.input_panel,'Enable','on',...
          'Units','normalized',...
          'Position',[0.60,0.015,0.30,0.04],...
          'Style','pushbutton','String','Run',...
          'FontName','courier new','FontSize',12,...
          'Callback',@Time_Domain_Callback);
```

Para evitar que los uicontrols se queden activados si se desmarca el *check*, se utiliza una sentencia *elseif* que inhabilita de nuevo los uicontrols.

```
    elseif get(E.domain_T,'Value') == 0
        set(E.size_data,'Enable','off');
        set(E.limit_data,'Enable','off');
        set(E.gain_L_data,'Enable','off');
        set(E.gain_U_data,'Enable','off');
        set(E.exp_L_data,'Enable','off');
        set(E.exp_U_data,'Enable','off');
        set(E.parameter_F_data,'Enable','off');
        set(E.parameter_CR_data,'Enable','off');
        set(E.run_button,'Enable','off');
    end
```

En el caso de que se marque la otra opción disponible (*E.domain_F*) ocurrirá el mismo proceso, pero además se activarán los uicontrols correspondientes al dominio de la frecuencia y desde el botón se llamará a una función diferente (*Frequency_Domain_Callback*, ver apartado 4.2.8).

```
  elseif hObject == E.domain_F
    if get(E.domain_F,'Value') == 1
        set(E.domain_T,'Value',0);
        set(E.size_data,'Enable','on');
        set(E.limit_data,'Enable','on');
        set(E.gain_L_data,'Enable','on');
        set(E.gain_U_data,'Enable','on');
        set(E.exp_L_data,'Enable','on');
        set(E.exp_U_data,'Enable','on');
        set(E.parameter_F_data,'Enable','on');
        set(E.parameter_CR_data,'Enable','on');
        set(E.interval_data,'Enable','on');
        set(E.minwcg_data,'Enable','on');
        set(E.maxwcg_data,'Enable','on');
        set(E.phase_data,'Enable','on');
        set(E.high_F_data,'Enable','on');
        set(E.high_F_dB_data,'Enable','on');
        set(E.sensitivity_data,'Enable','on');
        set(E.sensitivity_dB_data,'Enable','on');
        set(E.new_button,'Enable','off');
        E.run_button = uicontrol (...
          'Parent',E.input_panel,'Enable','on',...
          'Units','normalized', ...
          'Position',[0.60,0.015,0.30,0.04],...
          'Style','pushbutton','String','Run',...
          'FontName','courier new','FontSize',12,...
          'Callback',@Frequency_Domain_Callback);
```

4.2.6. Función Test_Callback

Esta función se ha creado para ofrecer al usuario la opción de establecer todos los parámetros de entrada requeridos por la interfaz. La llamada a esta función se produce al marcar la casilla *Test Data*, momento en que el panel de entrada se carga automáticamente con unos valores de prueba determinados. Si la casilla se desmarca, los valores de prueba se borran.

```
E.test_button = uicontrol(...
  'Parent',E.input_panel,'Enable','on',...
  'Units','normalized','Position',[0.35,0.015,0.2,0.04],...
  'Style','checkbox','String','Test Data',...
  'FontName','courier new','FontSize',11,...
  'BackGroundColor',[0.82,0.863,0.922],'Callback',@Test_Callback);

function Test_Callback(hObject, eventdata, handles)
  if get(E.test_button,'Value') == 1
    %Set plant parameters
    set(E.gain_data,'String','3');
    set(E.A_data,'String','0.2');
    set(E.B_data,'String','1');
    set(E.C_data,'String','0');
    set(E.delay_data,'String','0');
    %Set general parameters
    set(E.size_data,'String','20');
    set(E.limit_data,'String','20');
    set(E.gain_L_data,'String','0');
    set(E.gain_U_data,'String','5');
    set(E.exp_L_data,'String','0');
    set(E.exp_U_data,'String','1');
    set(E.parameter_F_data,'String','0.5');
    set(E.parameter_CR_data,'String','0.5');
    %Set frequency parameters
    set(E.interval_data,'String','0.5');
    set(E.minwcg_data,'String','1');
    set(E.maxwcg_data,'String','50');
    set(E.phase_data,'String','10');
    set(E.high_F_data,'String','100');
    set(E.high_F_dB_data,'String','10');
    set(E.sensitivity_data,'String','0.01');
    set(E.sensitivity_dB_data,'String','10');
  elseif get(E.test_button,'Value') == 0
    %Reset all parameters
    set(E.gain_data,'String','');
    set(E.A_data,'String','');
    set(E.B_data,'String','');
    set(E.C_data,'String','');
    set(E.delay_data,'String','');
    set(E.size_data,'String','');
    set(E.limit_data,'String','');
    set(E.gain_L_data,'String','');
    set(E.gain_U_data,'String','');
    set(E.exp_L_data,'String','');
    set(E.exp_U_data,'String','');
    set(E.parameter_F_data,'String','');
    set(E.parameter_CR_data,'String','');
    set(E.interval_data,'String','');
    set(E.minwcg_data,'String','');
    set(E.maxwcg_data,'String','');
    set(E.phase_data,'String','');
    set(E.high_F_data,'String','');
    set(E.high_F_dB_data,'String','');
    set(E.sensitivity_data,'String','');
    set(E.sensitivity_dB_data,'String','');
  end
end
```

4.2.7. Función Time_Domain_Callback

Esta función permite conectar el programa principal con la función de sintonía de controladores fraccionarios correspondiente al dominio del tiempo, la cual es llamada desde la función *Check_Callback* (ver apartado 4.2.5).

La variable global *tuning* será la que permita seleccionar el tipo de sintonía, que para este caso tendrá el valor 1. Además, los valores introducidos en los diferentes uicontrols del panel de entrada se almacenarán en unas nuevas variables de tipo estructura a través de la sentencia *get*.

```
function Time_Domain_Callback(hObject, eventdata, handles)
  global tuning
    tuning = 1; %Option for time domain tuning
    E.gain_str = get(E.gain_data,'String');
    E.A_str = get(E.A_data,'String');
    E.B_str = get(E.B_data,'String');
    E.C_str = get(E.C_data,'String');
    E.delay_str = get(E.delay_data,'String');
    E.size_str = get(E.size_data,'String');
    E.limit_str = get(E.limit_data,'String');
    E.gain_U_str = get(E.gain_U_data,'String');
    E.exp_U_str = get(E.exp_U_data,'String');
    E.parameter_F_str = get(E.parameter_F_data,'String');
    E.parameter_CR_str = get(E.parameter_CR_data,'String');
```

El siguiente bucle evita que alguno de los campos de entrada se quede vacío, de modo que si cualquiera de las variables anteriores no contiene ningún valor automáticamente aparecerá un mensaje de error a través de la sentencia *msgbox*.

```
  if strcmp('',E.gain_str)||strcmp('',E.A_str)||...
     strcmp('',E.B_str)||strcmp('',E.C_str)||...
     strcmp('',E.delay_str)|| strcmp('',E.size_str)||...
     strcmp('',E.limit_str)|| strcmp('',E.gain_U_str)||...
     strcmp('',E.exp_U_str)|| strcmp('',E.parameter_F_str)||...
     strcmp('',E.parameter_CR_str)
    msgbox('Input data missing. Please check','Error','error');
```

Debido a que los datos de los uicontrols son de tipo caracter, antes de llamar a la función que realiza todas las operaciones, se deben transformar los datos a tipo entero. Esto se hace a través de la sentencia *str2num*.

```
  else
    S.gain = str2num(E.gain_str);
    S.A_constant = str2num(E.A_str);
    S.B_constant = str2num(E.B_str);
    S.C_constant = str2num(E.C_str);
    S.delay = str2num(E.delay_str);
    S.size = str2num(E.size_str);
    S.limit = str2num(E.limit_str);
    S.gain_U = str2num(E.gain_U_str);
    S.exp_U = str2num(E.exp_U_str);
    S.parameter_F = str2num(E.parameter_F_str);
    S.parameter_CR = str2num(E.parameter_CR_str);
```

A continuación se llega a una de las partes más importantes del programa, la llamada a la función externa *Control_Fraccionario* (ver apartado A1.1). Como datos de entrada se le pasan los anteriormente transformados junto con la variable de selección del dominio de sintonía, y como datos de salida la función devuelve valores que utilizaremos más tarde.

```
[S.out_P,S.out_I,S.out_D,S.out_orderI,S.out_orderD,...
S.out_frequency,S.out_phase,S.out_overshoot,S.out_peaktime,...
S.out_settlingtime,S.out_costvalue] = Control_Fraccionario(...
tuning,S.gain,S.A_constant,S.B_constant,S.C_constant,S.delay,...
S.size,S.limit,S.gain_U,S.exp_U,S.parameter_F,S.parameter_CR,...
EJE.iteration_plot,EJE.magnitude_bode_plot,EJE.phase_bode_plot);
```

Se observa que los tres últimos valores de entradas no están definidos dentro de esta función, pero sí dentro de la función *CreateTab*. Al pertenecer a una estructura *EJE* declarada como global al principio del programa, los valores se podrán utilizar sin problemas dentro del programa.

Una vez realizadas todas las operaciones, se obtienen los datos de salida de tipo entero. Para poder mostrar los datos sobre los uicontrols de salida se deben pasar los valores de tipo entero a string mediante la sentencia *num2str* con una precisión de cuatro caracteres para cada valor. Seguidamente se llama a la función *Display_Data* (ver apartado 4.2.9).

```
    E.conv_P = num2str(S.out_P,4);
    E.conv_I = num2str(S.out_I,4);
    E.conv_D = num2str(S.out_D,4);
    E.conv_orderI = num2str(S.out_orderI,4);
    E.conv_orderD = num2str(S.out_orderD,4);
    E.conv_frequency = num2str(S.out_frequency,4);
    E.conv_phase = num2str(S.out_phase,4);
    E.conv_overshoot = num2str(S.out_overshoot,4);
    E.conv_peaktime = num2str(S.out_peaktime,4);
    E.conv_settlingtime = num2str(S.out_settlingtime,4);
    E.conv_costvalue = num2str(S.out_costvalue,4);
    %Call subfunction
    Display_Data;
  end
end
```

4.2.8. Función Frequency_Domain_Callback

Esta función es muy similar a la anterior (ver apartado 4.2.7), solo que en este caso la sintonía se realiza en el dominio de la frecuencia. En adelante se detallan las diferencias existentes.

Además de la variable global *tuning*, que tomará el valor 3, esta función requiere de otra variable global *Fparam*, que será el array donde se guarden los parámetros correspondientes al dominio de la frecuencia.

```
function Frequency_Domain_Callback(hObject, eventdata, handles)
  global tuning Fparam
    tuning = 3; %Option for frequency domain tuning
    Fparam = zeros(1,8);
    E.gain_str = get(E.gain_data,'String');
    E.A_str = get(E.A_data,'String');
    E.B_str = get(E.B_data,'String');
    E.C_str = get(E.C_data,'String');
    E.delay_str = get(E.delay_data,'String');
    E.size_str = get(E.size_data,'String');
    E.limit_str = get(E.limit_data,'String');
    E.gain_U_str = get(E.gain_U_data,'String');
    E.exp_U_str = get(E.exp_U_data,'String');
    E.parameter_F_str = get(E.parameter_F_data,'String');
    E.parameter_CR_str = get(E.parameter_CR_data,'String');
    E.interval_str = get(E.interval_data,'String');
    E.minwcg_str = get(E.minwcg_data,'String');
    E.maxwcg_str = get(E.maxwcg_data,'String');
    E.phase_str = get(E.phase_data,'String');
    E.high_F_str = get(E.high_F_data,'String');
    E.high_F_dB_str = get(E.high_F_dB_data,'String');
    E.sensitivity_str = get(E.sensitivity_data,'String');
    E.sensitivity_dB_str = get(E.sensitivity_dB_data,'String');
```

Para que el proceso de sintonía en el dominio de la frecuencia se ejecute de manera satisfactoria, antes de llamar a la función externa *Control_Fraccionario* (ver apartado A1.1) se comprobará que ningún campo esté vacío y los datos se transformarán a valores de tipo entero.

```
  if strcmp('',E.gain_str)||strcmp('',E.A_str)||...
     strcmp('',E.B_str)||strcmp('',E.C_str)||...
     strcmp('',E.delay_str)||strcmp('',E.size_str)|| ...
     strcmp('',E.limit_str)||strcmp('',E.gain_U_str)|| ...
     strcmp('',E.exp_U_str)||strcmp('',E.parameter_F_str)|| ...
     strcmp('',E.parameter_CR_str)||...
     strcmp('',E.interval_str)||strcmp('',E.minwcg_str)||...
     strcmp('',E.maxwcg_str)||strcmp('',E.phase_str)||...
     strcmp('',E.high_F_str)||strcmp('',E.high_F_dB_str)||...
     strcmp('',E.sensitivity_str)||strcmp('',E.sensitivity_dB_str)
    msgbox('Input data missing. Please check','Error','error');
  else
    S.gain = str2num(E.gain_str);
    S.A_constant = str2num(E.A_str);
    S.B_constant = str2num(E.B_str);
    S.C_constant = str2num(E.C_str);
    S.delay = str2num(E.delay_str);
    S.size = str2num(E.size_str);
    S.limit = str2num(E.limit_str);
    S.gain_U = str2num(E.gain_U_str);
    S.exp_U = str2num(E.exp_U_str);
    S.parameter_F = str2num(E.parameter_F_str);
    S.parameter_CR = str2num(E.parameter_CR_str);
    S.interval = str2num(E.interval_str);
    S.minwcg =str2num(E.minwcg_str);
    S.maxwcg =str2num(E.maxwcg_str);
    S.phase = str2num(E.phase_str);
```

```
S.high_F = str2num(E.high_F_str);
S.high_F_dB = str2num(E.high_F_dB_str);
S.sensitivity = str2num(E.sensitivity_str);
S.sensitivity_dB = str2num(E.sensitivity_dB_str);
```

Cada uno de los valores correspondientes a los parámetros frecuenciales se guardarán en el array *Fparam*.

```
Fparam(1,1)= S.interval;
Fparam(1,2)= S.minwcg;
Fparam(1,3)= S.maxwcg;
Fparam(1,4)= S.phase;
Fparam(1,5)= S.high_F;
Fparam(1,6)= S.high_F_dB;
Fparam(1,7)= S.sensitivity;
Fparam(1,8)= S.sensitivity_dB;
```

En la llamada a la función externa lo único que cambia es que se incluye esta última variable como dato de entrada.

```
%Call to function Control_Fraccionario!!!!!
[S.out_P,S.out_I,S.out_D,S.out_orderI,S.out_orderD,...
S.out_frequency,S.out_phase,S.out_overshoot, S.out_peaktime,...
S.out_settlingtime,S.out_costvalue] = Control_Fraccionario(...
tuning,S.gain,S.A_constant,S.B_constant,S.C_constant,S.delay,...
S.size,S.limit,S.gain_U,S.exp_U,S.parameter_F,S.parameter_CR,...
EJE.iteration_plot,EJE.magnitude_bode_plot,EJE.phase_bode_plot,...
Fparam);
```

Los datos de salida y su posterior conversión no cambian con respecto a lo explicado en el apartado 4.2.7.

4.2.9. Función Display_Data

Esta es una función sencilla que permite la visualización de los resultados. Para ello vuelve a escribir los uicontrols de tipo *text* ya definidos en la función *CreateTab* (ver punto 4.2.4), con la diferencia de que ahora contienen una nueva propiedad *string* con la correspondiente variable de salida.

Dado que todos los uicontrols tienen la misma sintaxis, a continuación solo aparece uno de ellos.

```
function Display_Data ()
  E.out_P = uicontrol(...
    'Parent',E.pid_panel,'Enable','on',...
    'Units','normalized','Position',[0.55,0.66,0.35,0.10],...
    'Style','text','String',E.conv_P,'FontName','courier new',...
    'FontSize',10,'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
```

En la Figura 4.8 se relaciona cada uicontrol de esta función con su cuadro de salida correspondiente en la interfaz.

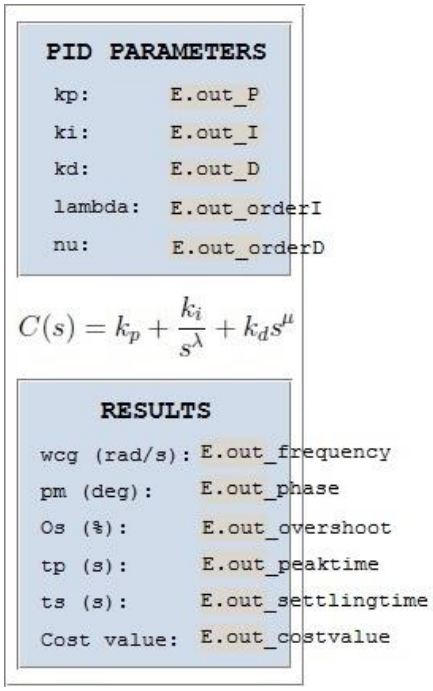

Figura 4.8. Panel de salida con sus uicontrols

Seguidamente se activará el botón *New Data* para permitir al usuario realizar un nuevo proceso. Cuando éste se activa, el resto de botones se desactivan hasta recibir la llamada del nuevo proceso.

```
  E.new_button = uicontrol (...
    'Parent',E.input_panel,'Enable','on',...
    'Units','normalized','Position',[0.10,0.015,0.2,0.04],...
    'Style','pushbutton','String','New Data',...
    'FontName','courier new','FontSize',12,'Callback',@New_Data);
  set(E.test_button,'Enable','off');
  set(E.run_button,'Enable','off');
  set(E.domain_T,'Enable','off');
  set(E.domain_F,'Enable','off');
end
```

4.2.10. Función New_data

Esta función se encarga de borrar todos los datos de la interfaz: para los uicontrol de tipo *edit* y *text* utiliza la sentencia *set*, y las gráficas se dibujan de nuevo vacías.

```
function New_Data(hObject, eventdata, handles)
  set(E.domain_T,'Enable','on','Value',0);
  set(E.domain_F,'Enable','on','Value',0);
  set(E.test_button,'Enable','on','Value',0);
  set(E.run_button,'Enable','off');
  set(E.size_data,'String','','Enable','off');
```

4.2.11. Función Save_Data

Esta función ofrece al usuario la opción de guardar todos los datos que pertenecen a la estructura *S* en un archivo de tipo *mat*. La llamada a esta función se produce al pulsar el botón *Save data (.mat)*, de forma que el usuario puede elegir el destino donde desea guardar los datos.

```
E.save_data_button = uicontrol(...
  'Parent',hpanel(1),'Enable','on',...
  'Units','normalized','Position',[0.47,0.03,0.12,0.05],...
  'Style','pushbutton','String','Save data(.mat)',...
  'FontName','courier new','FontSize',11,'Callback',@Save_Data);
function Save_Data(hObject, eventdata, handles)
  [filename,pathname] = uiputfile('*.mat');
  save(filename,'*S')
  clear *S;
end
```

4.2.12. Función Save_Interface

Esta función ofrece al usuario la opción de exportar (sentencia *hgexport*) una imagen de la figura que se está visualizando en el momento de pulsar el botón *Save screenshot*. La llamada a la función se produce al pulsar dicho botón, de forma que el usuario puede elegir el destino donde desea guardar la imagen.

```
E.save_screen_button = uicontrol(...
  'Parent',hpanel(1),'Enable','on',...
  'Units','normalized','Position',[0.47,0.09,0.12,0.05],...
  'Style','pushbutton','String','Save screenshot',...
  'FontName','courier new','FontSize',11,...
  'Callback',@Save_Interface);
function Save_Interface(varargin)
  [filename,pathname] = uiputfile('*.jpg');
  hgexport(gcf,filename,hgexport('factorystyle'),'Format','jpeg');
end
```

4.2.13. Función Read_PDF

Esta función se ha creado para ofrecer al usuario la opción de abrir un documento de ayuda a través de una sentencia propia de Matlab que abre el archivo situado entre paréntesis, siempre que sea de tipo pdf, txt, etc. La llamada a la función se produce al pulsar el botón *HELP(?)*.

```
E.help_button = uicontrol(...
  'Parent',hpanel(1),'Enable','on',...
  'Units','normalized','Position',[0.50,0.17,0.06,0.05],...
  'Style','pushbutton','String','HELP(?)',...
  'FontName','courier new','FontSize',11,...
  'BackGroundColor',[1,1,0.5],'Callback',@Read_PDF);
function Read_PDF (varargin)
  open('Help.pdf');
end
```

5. FUNCIONAMIENTO DE LA INTERFAZ

En este capítulo se trata todo lo referente al funcionamiento de la interfaz, y está estructurado en dos secciones: en la sección 5.1 se da una idea general de los requisitos necesarios para el uso de la interfaz, mientras que en la sección 5.2 se detalla su funcionamiento paso a paso considerando ambos enfoques de sintonía (dominio del tiempo y de la frecuencia).

5.1. Requisitos para su funcionamiento

Para poder hacer uso de la interfaz es imprescindible que el usuario tenga instalado el programa MATLAB en su equipo. Una vez abierto el programa, se deberá cargar la carpeta donde se localizan todos los archivos necesarios para el funcionamiento de la interfaz en el espacio *Current Folder*. A continuación basta con abrir el archivo principal *main.m* y ejecutarlo en el comando *Run*.

Además se debe apuntar que la interfaz ha sido programada en la versión R2014a de Matlab, compatible con todos los sistemas operativos que contenga las versiones posteriores a la del 2014 (esta última incluida). Si se usan versiones anteriores conviene asegurarse que éstas tienen todas las librerías necesarias.

Una vez creada toda la interfaz a partir de la programación explicada en los apartados 2 y 4, se pasa a detallar el funcionamiento de la misma. Para un buen uso de la interfaz se recomienda leer todos los pasos detenidamente además de tener unos conocimientos mínimos sobre control fraccionario y algoritmo DE.

5.2. Funcionamiento paso a paso

Antes de empezar el apartado debe aclararse que para un mejor entendimiento del funcionamiento de la interfaz, se analizarán ambos procesos de sintonía: en el dominio del tiempo y en el dominio de la frecuencia. A continuación se describe el funcionamiento en común, y más adelante se especificarán las diferencias.

Debido a la cantidad de archivos que la interfaz tiene que procesar tras inicializar el programa, se muestra una primera pantalla de espera con el mensaje que se ve en la Figura 5.1 para indicar que la interfaz se está cargando.

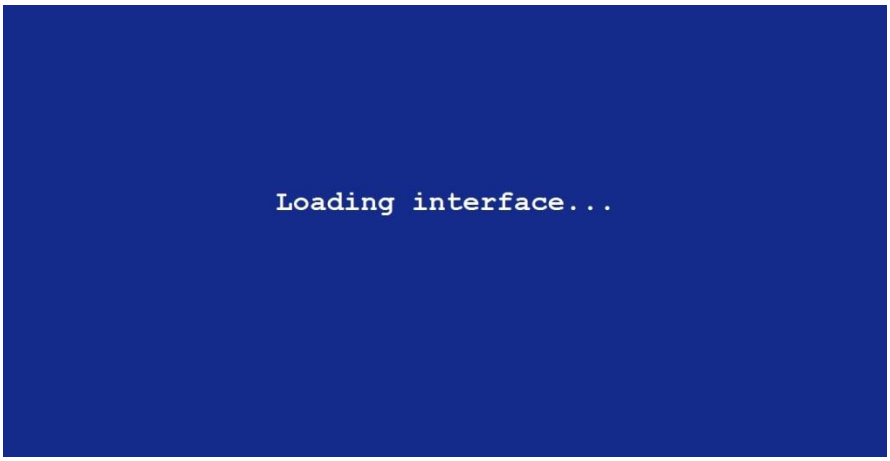

Figura 5.1. Pantalla de espera.

Pasados unos instantes, el programa comienza a funcionar mostrando la pantalla principal dentro de una pestaña. En la Figura 5.2 se muestra esta pantalla donde se encuentran localizados los diferentes elementos necesarios para que el usuario pueda sintonizar un controlador fraccionario sin dificultades.

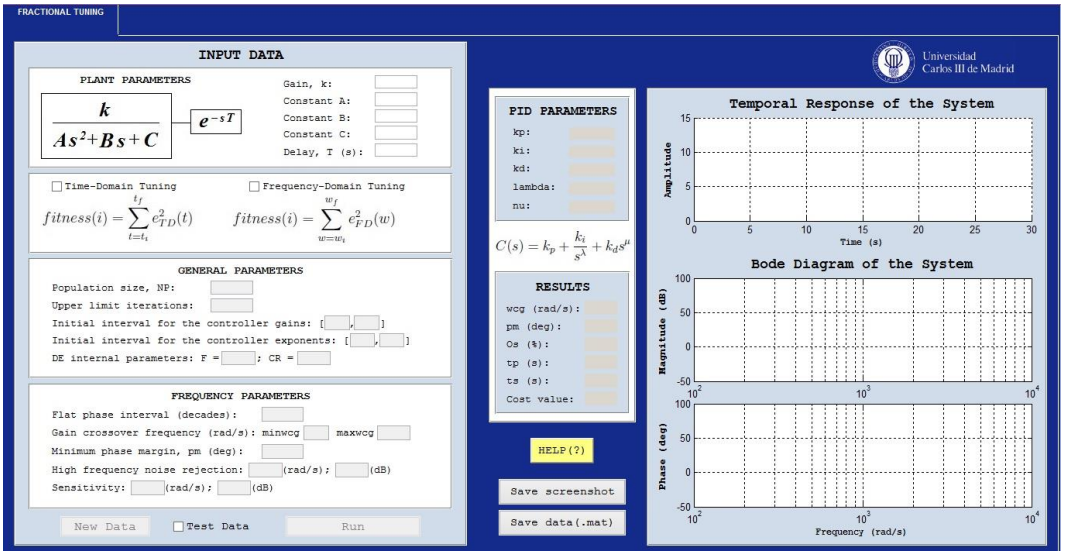

Figura 5.2. Pantalla principal del programa.

El primer paso se realiza sobre el panel de entrada (*Input Data*) donde se deben introducir los datos de entrada requeridos por la interfaz. Este panel dispone de una casilla *Test Data* que, cuando se selecciona, permite establecer unos valores de prueba para todos ellos (ver Figura 5.3).

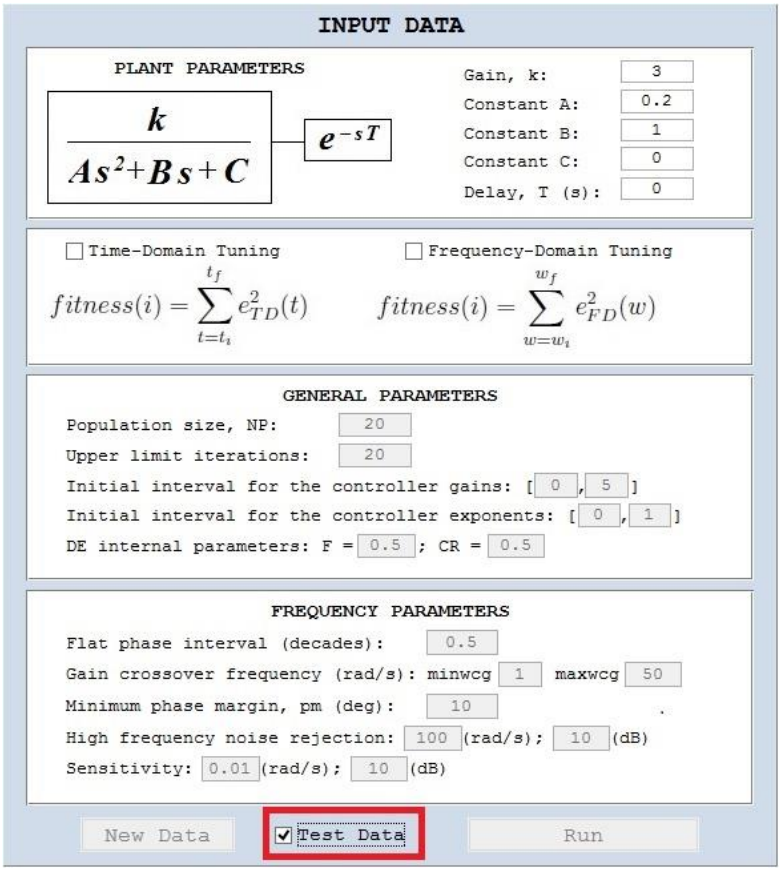

Figura 5.3. Opción *Test Data*.

Como se puede observar, existen tres tipos de datos de entrada diferentes.

Por un lado están los parámetros de la planta que se desea sintonizar (*Plant Parameters*). Para el caso de prueba propuesto, esta planta se define con:

- Una ganancia (*Gain, k*) de 3.
- Una constante propia del segundo orden de la planta (*Constant A*) de 0.2.
- Una constante propia del primer orden de la planta (*Constant B*) de 1.
- Una constante propia del orden cero de la planta (*Constant C*) nula.
- Un retardo (*Delay, T*) nulo.

Por otro lado están los parámetros generales (*General Parameters*) de configuración del filtro de optimización, cuyos valores de prueba son:

- Tamaño de la población (*Population size, NP*): 20.
- Límite superior de iteraciones (*Upper limit iterations*): 20.
- Intervalo inicial de las ganancias del controlador (*Initial interval for the controller gains*): [0, 5].
- Intervalo inicial de los exponentes del controlador (*Initial interval for the controller exponents*): [0, 1].
- Parámetros internos de DE (*DE internal parameters*): F=0.5 y CR=0.5.

Para el caso del dominio de la frecuencia, será necesario especificar otra serie de parámetros (*Frequency Parameters*), cuyos valores de prueba son:

- Intervalo de fase plana (*Flat phase interval*): 0.5 décadas.
- Frecuencia de ganancia de cruce (*Gain crossover frequency*) en el intervalo [1, 50] rad/s.
- Margen de fase mínimo (*Minimum phase margin*): 10º.
- Rechazo de ruido a alta frecuencia (*High frequency noise rejection*): a la frecuencia de 100 rad/s, una atenuación de 10 dB.
- Sensibilidad (*Sensitivity*): a la frecuencia de 0.01 rad/s, una atenuación de 10 dB.

A continuación, se debe elegir el dominio en el que se desea realizar la sintonización del controlador fraccionario: dominio del tiempo (*Time-Domain Tuning*) o dominio de la frecuencia (*Frequency-Domain Tuning*). Dependiendo del dominio seleccionado se habilitarán unos parámetros de entrada u otros, tal y como se explica en los apartados 5.2.1 y 5.2.2.

5.2.1. Funcionamiento en el dominio del tiempo

Estando habilitados los parámetros de la planta (*Plant Parameters*), cuando se selecciona la sintonía en el dominio del tiempo, se habilitan los parámetros generales (*General Parameters*) de configuración del algoritmo DE, así como el botón *Run*.

Una vez introducidos todos los datos cuyos campos están habilitados, se pulsa el botón *Run*, y si alguno de ellos está vacío, el programa lanza el mensaje de error que se muestra en la Figura 5.4.

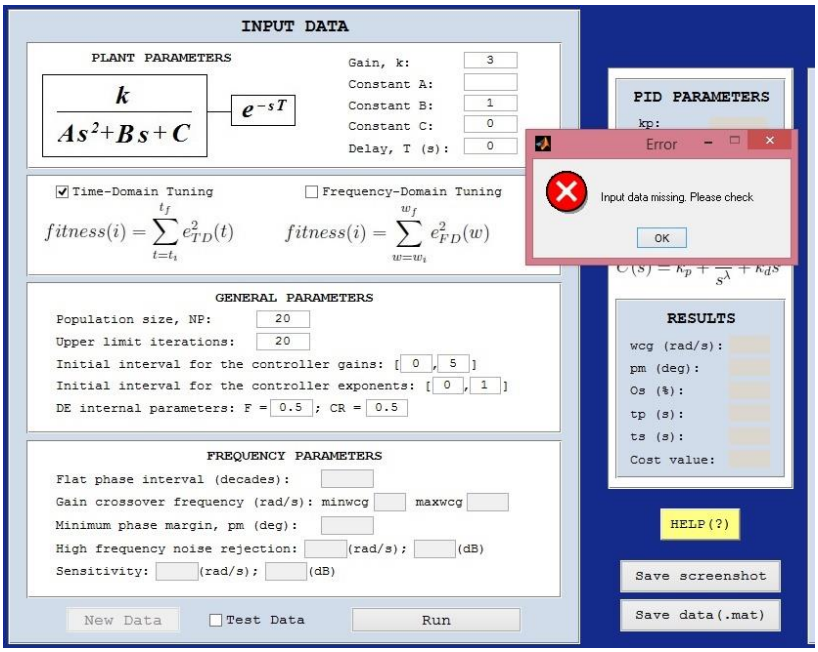

Figura 5.4. Ventana de error por falta de datos.

En caso contrario, comienza el proceso de sintonía. Para indicar al usuario el tiempo de espera restante hasta completar el proceso, se visualiza la ventana de espera que aparece en la Figura 5.5.

Figura 5.5. Ventana de espera

Una vez finalizado el proceso, los resultados se mostrarán en el panel de salida donde se distinguen dos paneles (ver Figura 5.6).

En el primer panel se obtienen los valores de los parámetros del controlador $PI^{\lambda}D^{\mu}$ fraccionario (*PID Parameters*). Para el caso de prueba, la Ecuación 3 propia de la función de transferencia del controlador tiene la siguiente expresión:

$$C(s) = 2.318 + 1.791 / s^{0.1}, \qquad (23)$$

En el segundo panel aparecen los datos de salida correspondientes al proceso de sintonía en el dominio elegido (*Results*). Como resultados del caso de prueba tenemos:

- Frecuencia de ganancia de cruce (*wcg*): 4.986 rad/s.
- Margen de fase (*pm*): 36.28 º.
- Sobreoscilación (*Os*): 39.64 %.
- Tiempo de pico (*tp*): 0.6 s.
- Tiempo de establecimiento (*ts*): 2.1 s.
- Valor de la función de coste (*Cost value*): 5.067. Es el valor que arroja la función de coste al finalizar el proceso de optimización.

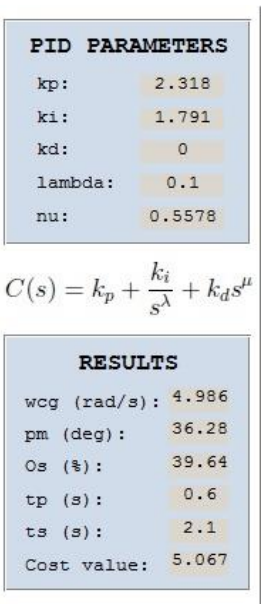

Figura 5.6. Datos de salida en el dominio del tiempo

Con el fin de facilitar la interpretación de los resultados, en un tercer panel se muestra la respuesta temporal de la última iteración del sistema (ver Figura 5.7).

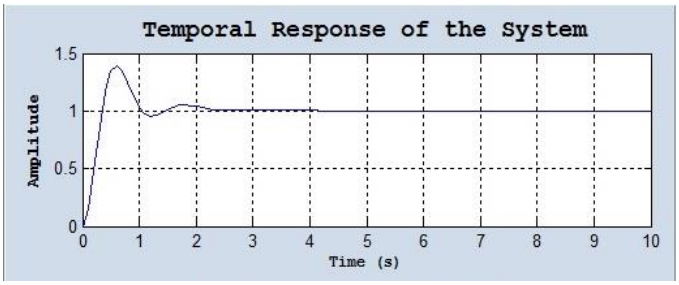

Figura 5.7. Respuesta temporal del sistema

5.2.2. Funcionamiento en el dominio de la frecuencia

En este apartado se explicarán las diferencias con respecto a la sintonía en el dominio del tiempo.

Cuando se selecciona la sintonía en el dominio de la frecuencia, además de habilitarse los parámetros generales (*General Parameters*) y el botón *Run*, también se habilitan los parámetros de configuración específicos de la frecuencia (*Frequency Parameters*).

El proceso de sintonía comienza al pulsar el botón *Run*. Si alguno de los campos habilitados estuviese vacío, aparecería el mensaje de error visto en la Figura 5.4. En caso contrario, se indicaría el tiempo de espera restante hasta completar el proceso (ver Figura 5.5). Así, los datos de salida obtenidos como resultado de la sintonía en el dominio de la frecuencia para los valores de prueba citados anteriormente se muestran en la Figura 5.8.

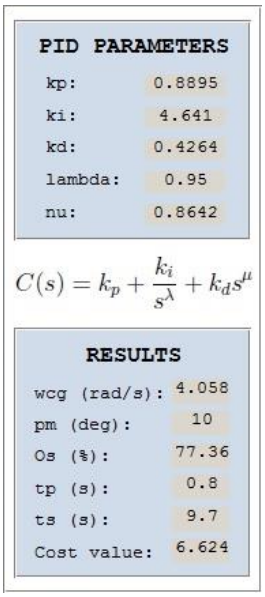

Figura 5.8. Datos de salida en el dominio de la frecuencia

La función de transferencia del controlador tiene la siguiente expresión:

$$C(s) = 0.8895 + 4.641 / s^{0.95} + 0.4264s^{0.8642}, \qquad (24)$$

En este caso, además de la respuesta temporal obtenida de la última iteración del sistema, también aparece el diagrama de Bode oportuno (ver Figura 5.9).

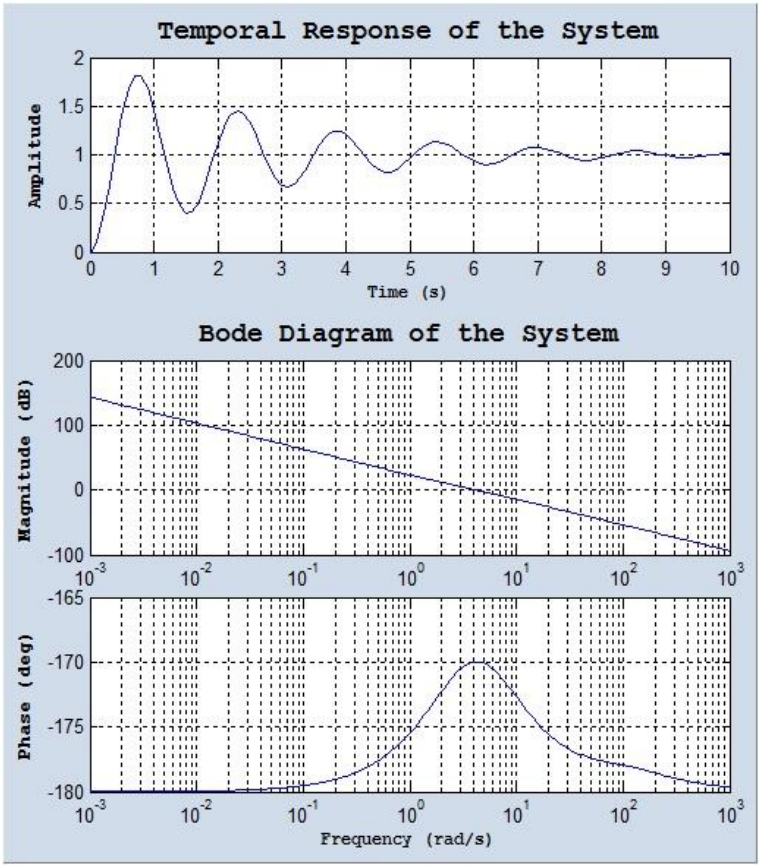

Figura 5.9. Respuesta temporal y diagrama de Bode

A la vista de los resultados arrojados por la interfaz, y observando la respuesta temporal, se aprecia un claro incremento de la sobreoscilación y del tiempo de establecimiento con respecto al dominio del tiempo. Este es un resultado lógico puesto que las limitaciones temporales no se han incluido en la función de coste basada en la frecuencia.

En cuanto al diagrama de Bode, se puede observar que el sistema sigue las especificaciones, la fase es lo más plana posible alrededor de la frecuencia de ganancia de cruce. El margen de fase es 10º y la frecuencia de cruce es 4.058 rad/s. El rechazo de ruido a alta frecuencia y las condiciones de sensibilidad también se satisfacen.

5.2.3. Otras opciones de funcionamiento general

Todas las gráficas del proceso se guardan automáticamente en un archivo denominado *graphs.fig* (ver Figura 5.10). De esta forma, el usuario las puede reutilizar y analizar en detalle.

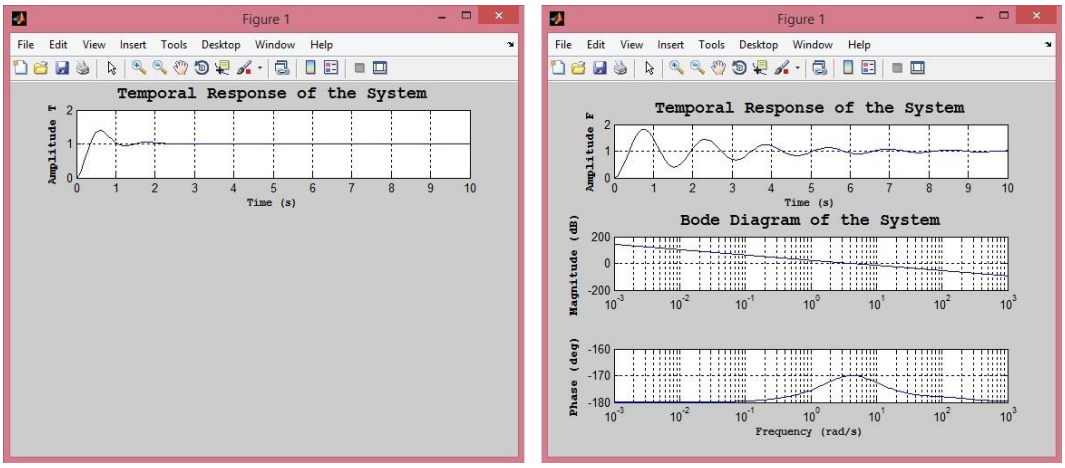

Figura 5.10. Archivo *graphs.fig* en el dominio del tiempo (izquierda) y en el dominio de la frecuencia (derecha)

La interfaz también ofrece la posibilidad de guardar una imagen de toda la pantalla a través del botón *Save screenshot*. Por su parte, el botón *Save data(.mat)* permite almacenar los datos de entrada y de salida en un fichero de tipo *mat*. En ambos casos el usuario elige el destino de estos archivos (ver Figura 5.11).

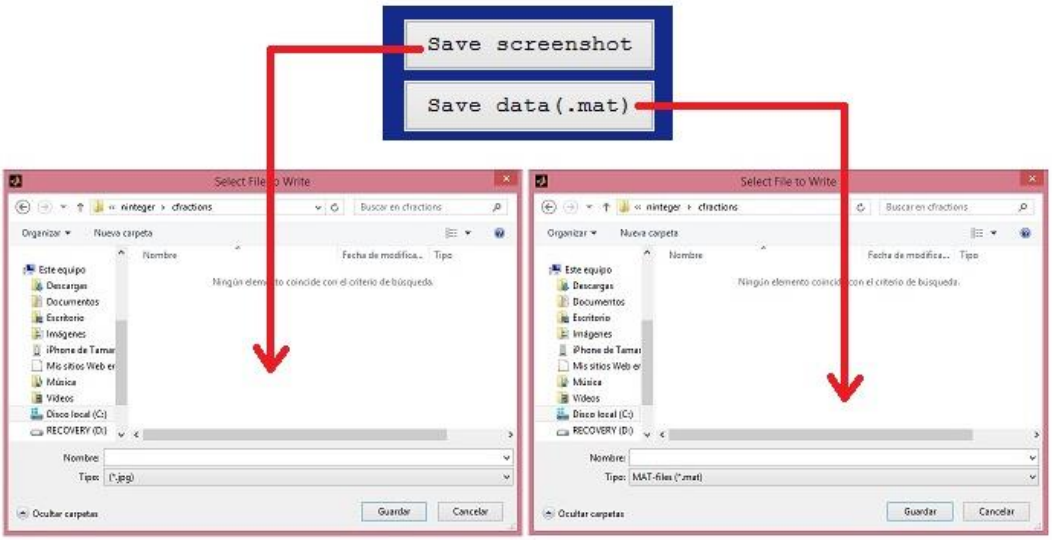

Figura 5.11. Botones *Save screenshot* y *Save data(.mat)*

Además existe un botón de ayuda *HELP(?)*, cuyo objetivo es resolver las dudas que el usuario pueda tener con respecto al funcionamiento de la interfaz. Al pulsarlo se abre automáticamente un documento en formato PDF que contiene las instrucciones.

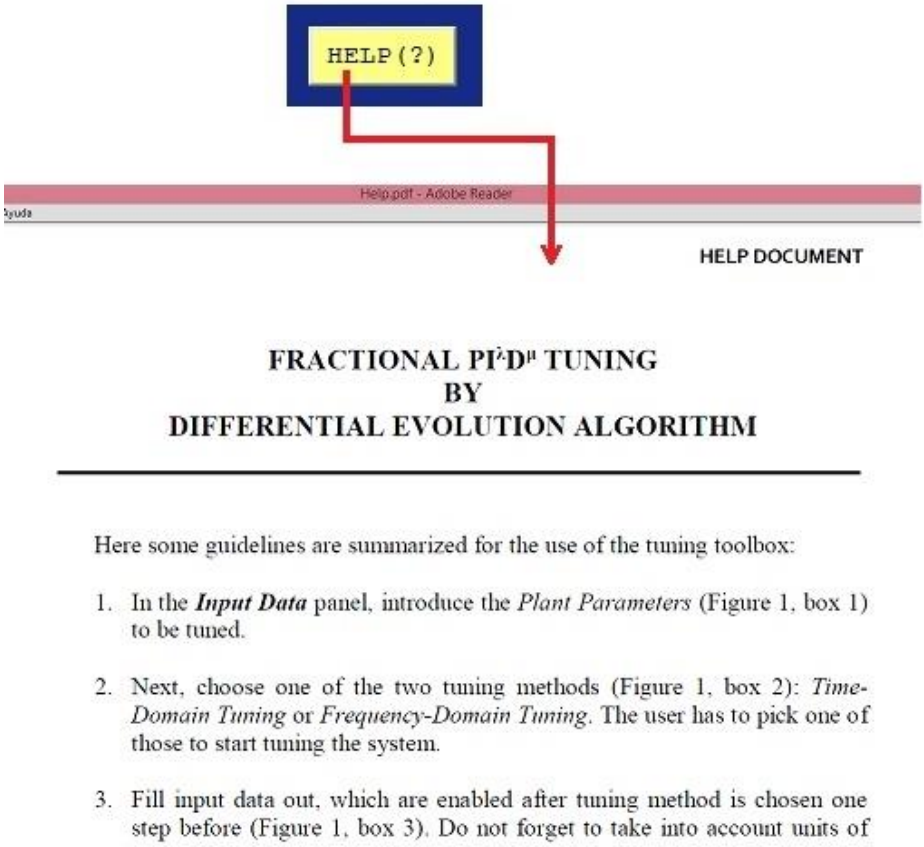

Figura 5.12. Botón *HELP(?)*

Por último mencionar que para iniciar un nuevo proceso, el usuario debe pulsar el botón *New Data*. De esta forma se borrarán todos los datos y gráficas del proceso anterior sin necesidad de cerrar y abrir de nuevo la interfaz.

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1. Conclusiones

Una vez llegados a este punto, debemos recordar que el objetivo principal de este trabajo era integrar el algoritmo de Evolución Diferencial para sintonizar controladores $PI^{\lambda}D^{\mu}$ fraccionarios sobre una interfaz gráfica a través de la herramienta GUIDE.

De los resultados experimentales obtenidos de dicha implementación, puede concluirse que tanto para el dominio del tiempo como de la frecuencia, el ajuste de los parámetros del controlador se realiza de manera satisfactoria, y por tanto, el objetivo principal puede darse por alcanzado.

Conviene mencionar que los resultados obtenidos para una misma planta, diferirán dependiendo de la estrategia de diseño elegida. Mientras que en el dominio del tiempo se optimiza la respuesta escalón, en el dominio de la frecuencia el sistema es más robusto a cambios en la ganancia de la planta.

Incluso siguiendo una misma estrategia y utilizando los mismos valores para el filtro de optimización, los resultados diferirán en cada simulación de una misma planta. Esto se debe a la naturaleza del propio método evolutivo, ya que utiliza poblaciones aleatorias en la estimación de los parámetros del controlador, pero siempre cumpliendo con las especificaciones de la función de coste.

Por otra parte, resulta interesante comentar algunas de las dificultades más reseñables encontradas a lo largo del trabajo:

- El punto fuerte del trabajo era realizar la conexión entre la interfaz y el método de sintonía. Aquí hubo bastantes problemas con la transferencia de datos de entrada (parámetros generales y frecuenciales) y de salida (sobreoscilación, tiempo de pico y tiempo de establecimiento), cuya solución es la que se desarrolla en esta memoria.

- Otro punto importante en la interfaz era la transferencia de las gráficas [24]. Para evitar la superposición de las mismas, se optó por guardarlas automáticamente en un archivo aparte según se generase el proceso. De esta manera se da la opción de poder trabajar con ellas una vez cerrada la interfaz, quedando a su vez cubierto el requisito de almacenamiento de gráficas.

- Finalmente se solucionaron las dificultades de comunicación entre la interfaz y el usuario. Así, para que el usuario tenga conocimiento sobre la ejecución del proceso y su tiempo restante, se ha incluido la ventana de espera de la Figura 5.5. Por su parte, la casilla *Test Data* permite establecer todos los datos de entrada de manera rápida, ofreciendo al usuario una orientación sobre la magnitud de los mismos.

A la vista de estos resultados, podríamos concluir que la interfaz está lista para su uso, y por tanto, el trabajo ha finalizado con éxito.

6.2. Trabajos futuros

A partir del trabajo presentado se pueden plantear ciertas mejoras e ideas que darían lugar al desarrollo de futuros trabajos:

- Depurar el código de la interfaz buscando alternativas para todo el proceso de conexión.

- Modificar la disposición de los datos de entrada a través de un botón que permita acceder a los parámetros generales y frecuenciales para cambiar sus valores sin necesidad de estar visibles todos ellos en la pantalla principal de la interfaz.

- Añadir un botón en la interfaz que le ofrezca al usuario la opción de guardar cada una de las gráficas de los distintos procesos de simulación.

- Agregar una nueva pestaña donde se incluya el documento de ayuda como un elemento más de la interfaz.

- Incorporar una opción en la interfaz que permita comparar los resultados obtenidos de las distintas estrategias de diseño para una misma planta.

- Implementar una nueva estrategia de diseño que combine ambas funciones de coste (tiempo y frecuencia) para ajustar un controlador que herede las ventajas de ambos enfoques.

- Usar diferentes pestañas para incorporar otros métodos de sintonía sobre la misma interfaz.

BIBLIOGRAFÍA

[1] Podlubny, I.: "*Fractional-order systems and $PI^{\lambda}D^{\mu}$ controllers*", IEEE Transactions on Automatic Control 44 (1999), pp. 208-214.

[2] Royo, J.: "*Diseño Digital*", Ediciones Paidós Ibérica (2004).

[3] Shneiderman, B.: "*Designing the user interface: Strategies for effective Human-computer interaction*", Addison-wesley (1998).

[4] Productos de MATLAB. Acceso en Septiembre de 2015. http://www.mathworks.com/products/matlab/index.html

[5] Martín, F., Monje, C.A., Moreno, L., Balaguer, C.: "*DE-based Tuning of $PI^{\lambda}D^{\mu}$ Controllers*"

[6] Monje, C.A., Chen, Y., Vinagre, B.M., Xue, D., Feliu-Batle, V.: "*Fractional-order Systems and Controls: Fundamentals and Applications*", Springer-Verlag London (2010).

[7] Ziegler, J.G., Nichols, N.B.: "*Optimum settings for automatic controllers*", Transactions of the ASME, Vol. 64 (1942), pp. 759-768.

[8] Vinagre, B.M., Monje, C.A.: "*Introducción al Control Fraccionario*", Revista Iberoamericana de automatica e Informatica Industrial, Vol. 3 (2006), pp. 5-23.

[9] Podlubny, I., Petrás, I., Vinagre, B. M., O'Leary, P., Dorcák, L.: "*Analogue realizations of fractional-order controllers*", Nonlinear Dynamics 29 (2002), pp. 281-296.

[10] Valério, D., Sá da Costa, J.: "*A review of tuning methods for fractional PIDs*", in Proceedings of the 4th IFAC Workshop on Fractional Differentiation and Its Applications, Badajoz, Spain (2005).

[11] Vinagre, B.M.: "*Modelling and control of dynamic systems characterized by integro-differential equations of fractional order*" (in Spanish), Ph.D. thesis, UNED, Madrid, Spain (2001).

[12] Chen, Y., Dou, H., Vinagre, B.M., Monje, C.A.: "*A robust tuning method for fractional order PI controllers*", in Proceedings of the 2nd IFAC Workshop on Fractional Differentiation and Its Applications, Porto, Portugal (2006), pp. 22-27.

[13] Valério, D., Sá da Costa, J.: "*Tuning of fractional PID controllers with Ziegler-Nichols type rules*". Signal Processing, Vol. 86 (2006), pp. 2771-2784.

[14] Valério, D., Sá da Costa, J.: "*Tuning rules for fractional PIDs*", in J.A.T. Machado, J. Sabatier, and O. Agrawal (eds.), Fractional calculus: theoretical

developments and applications in Physics and Engineering, Springer, Dordrecht (2007), pp. 463-476.

[15] Storn, R., Price, K.: "*Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces*", Journal of Global Optimization, Vol. 11 (1997), pp. 341-359.

[16] Estrategias mutación DE. Acceso en Septiembre de 2015. http://www.ICSI.Berkeley.edu/~storn/code.html

[17] Monje, C.A., Vinagre, B.M., Feliu-Batle, V., Chen, Y.: "*Tuning and auto-tuning of fractional order controllers for industry applications*", Control Engineering Practice, Vol. 16 (2008), pp. 798-812.

[18] Chen, Y., Moore, K.L.: "*Relay feedback tuning of robust PID controllers with iso-damping property*", IEEE Transactions on Systems, Man, and Cybernetics, Part B 35 (2005), pp. 23-31.

[19] Carrera Amuriza, A.R., Martínez Nebreda, M.: "*Introducción a MATLAB y a la creación de interfaces gráficas*", Servicio Editorial de la Universidad del País Vasco (2004).

[20] "*MATLAB 7 Getting Started Guide*", Mathworks (2010).

[21] Archivo uitabpanel.m (MATLAB Central). Acceso en Marzo de 2010 http://www.mathworks.com/matlabcentral/fileexchange/11546-uitabpanel

[22] Sabio Gallego, F.B.: "*Interfaz para la generación de trayectorias para las plataformas robóticas HOAP-3 Y RH-2*", Proyecto Fin de Carrera, Universidad Carlos III de Madrid (2011).

[23] Archivo maximize.m (MATLAB Central). Acceso en Marzo de 2010 http://www.mathworks.com/matlabcentral/fileexchange/25471-maximize

[24] Documentación de MATLAB. Acceso en Noviembre de 2015. http://www.mathworks.com/help/matlab/index.html

[25] Figueredo Pacheco, A.: "*Implementación en PLC de un método de autosintonía de controladores PID fraccionarios para servomotores de velocidad y posición*", Proyecto Fin de Carrera, Universidad Carlos III de Madrid (2009).

[26] De Lucas Barbosa, L.: "*Control fraccionario y adaptativo de un cilindro neumático*", Proyecto Fin de Carrera, Universidad Carlos III de Madrid (2011).

[27] Serrano Simón, T.: "*Identificación de sistemas dinámicos mediante test de relé modificado*", Trabajo Fin de Grado, Universidad Carlos III de Madrid (2014).

ANEXOS

ANEXO 1: Código programado

En esta sección se ofrece el código completo del proyecto fin de carrera, para que el lector pueda consultar con más detalle los apartados 2 y 4.

A1.1. Algoritmo de Evolución Diferencial

Se detalla toda la programación referente al algoritmo de Evolución Diferencial.

```
function [kp,ki,kd,lambda,nu,wcg,pm,Os,tp,ts,costvalue] =
Control_Fraccionario(tuning,k,A,B,C,delay,size,limit,gain_U,...
exp_U,factor,crossover,iteration_plot,magnitude_bode_plot,...
phase_bode_plot,Fparam)

%-------------------------------------------------------------------
addpath('ninteger');
%-------------------------------------------------------------------
% Options
cost_function=tuning;
mode_pid_approx=1;
% Transfer function of the process to be controlled
Gp=tf(k,[A B C]);
% DE variables
iter_max=limit;     %Number of iterations.
NP=size;            %Number of particles.
MAX_K_VAR=gain_U;   %Maximum value of the PID constants.
MAX_EXP_VAR=exp_U;  %Maximum value of the PID exponents.
F=factor;           %Differential mutation factor (0-2).
CR=crossover;       %Crossover constant (0-1).
D=5;                %Number of chromosomes(kp,kd,ki,lambda,nu).
D_exp=2;            %Number of chormosomes which are exponents.
% Some auxiliar variables
trial=zeros(1,D);
pob_aux=zeros(NP,D+1);
error=10000;
error_max=20000;
GLOBAL=zeros(iter_max,1);
MIN=zeros(iter_max,1);
MAX=zeros(iter_max,1);
%-------------------------------------------------------------------

% Start waiting bar.
WB=waitbar(0,'Program in execution. Please,wait...','Name','');

% The initial population is generated.
population=inicio_pob(NP,D,D_exp,MAX_K_VAR,MAX_EXP_VAR);

% Cost function evaluation for the initial population.
if cost_function == 1
  for i=1:NP
    population(i,1)=cost(Gp,population(i,2:(D+1)),...
                         mode_pid_approx,cost_function);
```

```
  end
elseif cost_function == 3
  for i=1:NP
    population(i,1)=cost(Gp,population(i,2:(D+1)),...
                         mode_pid_approx,cost_function,Fparam);
  end
end

% The DE algorithm estimates the best solution.
count=1;
imp_count=1;
while(count<=iter_max)
  waitbar(count/iter_max,WB); %Waiting bar
  for i=1:NP
    % MUTATION AND CROSSOVER
    % There different vectors and different from running index i.
    a=random('unid',NP);
    while((a==i)||(a==0))
      a=random('unid',NP);
    end
    b=random('unid',NP);
    while((b==i)||(b==a)||(b==0))
      b=random('unid',NP);
    end
    c=random('unid',NP);
    while((c==i)||(c==a)||(c==b)||(c==0))
      c=random('unid',NP);
    end
    for k=2:(D+1)
      cross_rand=random('unid',100);
      if(cross_rand < (100*CR))
        trial(1,(k-1))=population(c,k)+ F*(population(a,k)-...
                       population(b,k));
      else trial(1,(k-1))=population(i,k);
      end
    end
    % The Ks cannot be negative numbers.
    if trial(1,1)<0
      trial(1,1)=0;
    end
    if trial(1,2)<0
      trial(1,2)=0;
    end
    if trial(1,3)<0
      trial(1,3)=0;
    end
    % The exponents should be in the interval [0.2,0.95].
    % Not useful, the exponents get stocked at zero when using
    % this mechanism.
    if trial(1,4)<0.1
      trial(1,4)=0.1;
    end
    if trial(1,4)>0.95
      trial(1,4)=0.95;
    end
    if trial(1,5)<0.1
      trial(1,5)=0.1;
```

```
  end
  if trial(1,5)>0.95
    trial(1,5)=0.95;
  end
  % Cost function evaluation according to each hypothesis.
  % error_trial=cost(Gp,population(i,2:(D+1)));
  if cost_function == 1
    error_trial=cost(Gp,trial,mode_pid_approx,cost_function);
  elseif cost_function == 3
    error_trial=cost(Gp,trial,mode_pid_approx,cost_function,...
                     Fparam);
  end
  % SELECTION: The best individuals are chosen for the next
  % generation (between candidates and current population).
  % Thresholding band to deal with noisy measurements is
  % also implemented.
  if(error_trial<population(i,1)*1.00)
    for j=2:(D+1)
      pob_aux(i,j)=trial(1,j-1);
    end
    pob_aux(i,1)=error_trial;
  else
    for j=1:(D+1)
      pob_aux(i,j)=population(i,j);
    end
  end
end
population=pob_aux; %Population for the next generation
% DISCARDING - The 5% of worst individuals are discarded.
pob_orden=sortrows(population,1);
if NP<10
  ndisc=1;
else
  ndisc=int8(NP/20);
end
if NP>4 %Discarding is nonsense with less than 5 individuals
  for i=1:ndisc
    disc=random('unid',int8(NP/2));
    while disc==0
      disc=random('unid',int8(NP/2));
    end
    pob_orden(NP+1-i,:)=pob_orden(disc,:);
  end
end
population=pob_orden;
% The best, worst and global error are printed.
bestmem=population(1,2:(D+1));
error=population(1,1);
error_max=max(population(:,1));
error_global=sum(population(:,1));
if imp_count==4
  fprintf(1,'\n It: %f Best %f Worst: %f Global: %f \n
    Parameters: kp: %f kd: %f ki: %f nu: %f lambda: %f \n',
    count,error,error_max,error_global,bestmem(1),bestmem(2),...
    bestmem(3),bestmem(4),bestmem(5));
  imp_count=0;
end
GLOBAL(count)=error_global;
```

```
  MIN(count)=error;
  MAX(count)=error_max;
  imp_count=imp_count+1;
  count=count+1;
end
CONV.MIN=MIN;
CONV.MAX=MAX;
CONV.GLOBAL=GLOBAL;

% End waiting bar.
delete(WB)

% Output values.
kp=bestmem(1);
ki=bestmem(3);
kd=bestmem(2);
lambda=bestmem(5);
nu=bestmem(4);
costvalue=error;

% PID Calculation.
if mode_pid_approx==1
  PID=nipid(bestmem(1),bestmem(2),bestmem(4),bestmem(3),...
            bestmem(5),'crone',10,'crone');
else
  [Num,Den]=aprox_pidfrac_to_pid(bestmem);
  PID=tf(Num,Den);
end
steps=feedback(PID*Gp,1);
[~,pm,~,wcg] = margin(PID*Gp);
t = (0:0.1:10)';
y = step(steps,t);

% Overshoot.
yRP = y(length(y)); %Steady state value
Os = 100*(max(y) - yRP)/yRP;
% Peak time.
for i = 1:length(y)
  if y(i) == max(y)
    tp = t(i);
    break;
  end
end
% Settling time.
j=length(y);
while (y(j)>0.98*yRP) && (y(j)<1.02*yRP)
  j=j-1;
end
ts = t(j);

% PLOTS.
figura=figure;
  % Plot of the response when the input is a step, useful when
  % designing with the error of the step response.
if cost_function == 1
  subplot(3,1,1)
  plot(t,y),grid on;
```

```
    title('Temporal Response of the System','FontName',
      'courier new','FontSize',14,'FontWeight','bold');
    ylabel('Amplitude T','FontName','courier new',
      'FontSize',10,'FontWeight','bold');
    xlabel('Time (s)','FontName','courier new',
      'FontSize',9,'FontWeight','bold');
    grid on;
  plot(iteration_plot,t,y),grid on;
    title(iteration_plot,'Temporal Response of the System',
      'FontName','courier new','FontSize',14,'FontWeight','bold');
    ylabel(iteration_plot,'Amplitude','FontName','courier new',
      'FontSize',10,'FontWeight','bold');
    xlabel(iteration_plot,'Time (s)','FontName','courier new',
      'FontSize',9,'FontWeight','bold');
    grid(iteration_plot);
elseif cost_function == 3
  subplot(3,1,1)
  plot(t,y),grid on;
    title('Temporal Response of the System','FontName',
      'courier new','FontSize',14,'FontWeight','bold');
    ylabel('Amplitude F','FontName','courier new',
      'FontSize',10,'FontWeight','bold');
    xlabel('Time (s)','FontName','courier new',
      'FontSize',9,'FontWeight','bold');
    grid on;
  plot(iteration_plot,t,y),grid on;
    title(iteration_plot,'Temporal Response of the System',
      'FontName','courier new','FontSize',14,'FontWeight','bold');
    ylabel(iteration_plot,'Amplitude','FontName','courier new',
      'FontSize',10,'FontWeight','bold');
    xlabel(iteration_plot,'Time (s)','FontName','courier new',
      'FontSize',9,'FontWeight','bold');
    grid(iteration_plot);
  % Plot of the Bode Diagram.
  w=logspace(-3,3,200);
  [mg,ph]=bode(PID*Gp,w);
  bode_mgdB=zeros(1,200);
  bode_mgdB(:)=20*log10(mg(:));
  bode_ph=zeros(1,200);
  bode_ph(:)=ph(:);
  subplot(3,1,2)
  semilogx(w,bode_mgdB);
    title('Bode Diagram of the System','FontName','courier new',
      'FontSize',14,'FontWeight','bold');
    ylabel('Magnitude (dB)','FontName','courier new',
      'FontSize',10,'FontWeight','bold');
    grid on;
  semilogx(magnitude_bode_plot,w,bode_mgdB);
    title(magnitude_bode_plot,'Bode Diagram of the System',
      'FontName','courier new','FontSize',14,'FontWeight','bold');
    ylabel(magnitude_bode_plot,'Magnitude (dB)',
      'FontName','courier new','FontSize',10,'FontWeight','bold');
    grid(magnitude_bode_plot);
    grid on;
  subplot(3,1,3)
  semilogx(w,bode_ph);
    ylabel('Phase (deg)','FontName','courier new',
      'FontSize',10,'FontWeight','bold');
```

```
    xlabel('Frequency (rad/s)','FontName','courier new',
      'FontSize',9,'FontWeight','bold');
    grid on;
  semilogx(phase_bode_plot,w,bode_ph);
    ylabel(phase_bode_plot,'Phase (deg)','FontName','courier new',
      'FontSize',10,'FontWeight','bold');
    xlabel(phase_bode_plot,'Frequency (rad/s)','FontName',
      'courier new','FontSize',9,'FontWeight','bold');
    grid(phase_bode_plot);
end
saveas(figura,'graphs.fig')
set(figura,'visible','Off')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------
function pop=inicio_pob(NP,D,D_exp,MAX_K_VAR,MAX_EXP_VAR)
poppre=zeros(NP,D-D_exp);
cost=zeros(NP,1);
exponents=zeros(NP,D_exp);
for i=1:NP
    poppre(i,:) = MAX_K_VAR*rand(1,D-D_exp);
    exponents(i,:) = MAX_EXP_VAR*rand(1,D_exp);
end
pop=[cost poppre exponents];
%-------------------------------------------------------------------------
```

A1.2. Función de coste

Se explica la programación utilizada para el cálculo de la función de coste del sistema.

```
function[error]=cost(Gp,param,mode_pid_approx,option,Fparam)

%-------------------------------------------------------------------
% Initialization parameters
warning('OFF');
PHASE=zeros(11,1);
fr_w=0.2;
%-------------------------------------------------------------------

% PID Calculation.
if mode_pid_approx==1
  PID=nipid(param(1),param(2),param(4),param(3),param(5),...
            'crone',10,'crone');
else
  [Num,Den]=aprox_pidfrac_to_pid(param);
  PID=tf(Num,Den);
end
% Cost function options.
if option==1
  % OPTION 1: step response.
  M=feedback(PID*Gp,1);
  [Y]=step(M);
  error=(Y-1)'*(Y-1); %The squared error
end
if option==2
  % OPTION 2: frequency response.
  M=feedback(PID*Gp,1);
  [~,Pm,~,Wpm] = margin(PID*Gp);
  [Y]=step(M);
  if Pm>15 %Pm close to zero leads to oscilators
    if Wpm>1
      if Y(35)<0.8 %Y res is 0.0304sec
        for i=1:11
          Wint=10^((i-6)/30)*Wpm;
          %Phases around the phase margin, one decade interval
          [~,phase_i] = bode(PID*Gp,Wint);
          %Weighting the slope of the phases
          PHASE(i)=abs(phase_i-(Pm-180));
        end
        error=PHASE'*PHASE;
      else
        error=10000000;
      end
    else
      error=10000000;
    end
  else
    error=100000000000;
  end
end
```

```
if option==3
  % OPTION 3: frequency response: phase margin + plain.
  % Frequency parameters.
  fpinterval=Fparam(1); %Input: flat phase interval
  minwcg=Fparam(2); %Input: interval gain crossover frequency
  maxwcg=Fparam(3);
  minpm=Fparam(4); %Input: minimum phase margin (deg)
  high_F=Fparam(5); %Input: high frequency noise rejection (rad/s)
  high_F_dB=Fparam(6); %Input: high frequency noise rejection (dB)
  sensitivity=Fparam(7); %Input: sensitivity (rad/s)
  sensitivity_dB=Fparam(8); %Input: sensitivity (dB)
  % How to include the phase margin in the cost function.
  [~,Pm,~,Wpm] = margin(PID*Gp);
  M=feedback(PID*Gp,1);
  S=feedback(1,PID*Gp);
  [M_high,~]=bode(M,high_F);
  M_high_db=20*log10(M_high);
  [S_low,~]=bode(S,sensitivity);
  S_low_db=20*log10(S_low);
  [Y]=step(M);
  % Loop
  if Y(35)<0.8
    if M_high_db<-high_F_dB
      if S_low_db<-sensitivity_dB
        if Pm>minpm
          if Wpm>minwcg && Wpm<maxwcg
            A_dec=fpinterval;
            Wint=Wpm/10^(A_dec/2);
            [~,phase_i]=bode(PID*Gp,Wint);
            PHASE(1)=abs(phase_i-(Pm-180));
            for i=2:11
              Wint=Wint*10^(A_dec/10); %One decade interval
              %Phases around the phase margin
              [~,phase_i]=bode(PID*Gp,Wint);
              %Weighting the slope of the phases
              PHASE(i)=abs(phase_i-(Pm-180));
            end
            error=PHASE'*PHASE;
          else
            error=100000000000;
          end
        else
          error=100000000000;
        end
      else
        error=100000000000;
      end
    else
      error=100000000000;
    end
  else
    error=100000000000;
  end
end
```

```
if option==4
  % OPTION 4: mixture between step response and phase response.
  M=feedback(PID*Gp,1);
  [~,Pm,~,Wpm]=margin(PID*Gp);
  [Y]=step(M);
  size_Y=size(Y,1);
  error_st=((Y-1)'*(Y-1))/(size_Y);
  if Pm>30
    if Wpm>1 && Wpm<15
      for i=1:11
        Wint=Wpm+(i-6);
        if Wint>0
          %Phases around the phase margin
          [~,phase_i]=bode(PID*Gp,Wint);
          %Weighting the slope of the phases
          PHASE(i)=abs(phase_i-(Pm-180));
        else
          PHASE(i)=10000;
        end
      end
      error_fr=PHASE'*PHASE/11;
    else
      error_fr=100000000000;
    end
  else
    error_fr=100000000000;
  end
  error=(1-fr_w)*error_st+fr_w*error_fr;
end

end
```

A1.3. Programación interfaz

En este apartado se detalla toda la programación necesaria para la creación de la interfaz diseñada.

```
function main

%%%%%%%%%%%%%%%%%%%%-----Global variables-----%%%%%%%%%%%%%%%%%%%%
global WIDTH
global HEIGHT
global loading
global interface
global EJE
global S

%array = [left,bottom,width,height]
screensize = get(0,'ScreenSize');
WIDTH = screensize (3);
HEIGHT = screensize (4);

%%%%%%%%%%%%%%%%%%%%%-----LOADING FIGURE-----%%%%%%%%%%%%%%%%%%%%%
loading = figure(...
  'Name','','NumberTitle','off','Menubar','none',...
  'Units','pixels','Position',[0,0,WIDTH,HEIGHT],...
  'Color',[0.078,0.169,0.549]);
uicontrol(...
  'Parent',loading,...
  'Units','normalized','Position',[0.1,0.5,0.8,0.1],...
  'Style','text','String','Loading interface...',...
  'FontName','courier new','FontSize',32,'FontWeight','bold',...
  'ForeGroundColor',[1 1 1],'FontAngle','normal',...
  'BackGroundColor',[0.078,0.169,0.549]);
%Function to fit the program to the screen
maximize(loading)
%%%%%%%%%%%%%%%%%%%%%%%-----END LOADING-----%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%-----INTERFACE FIGURE-----%%%%%%%%%%%%%%%%%%%%
interface = figure(...
  'Visible','off',...
  'Name','','NumberTitle','off','Menubar','none',...
  'Units','normalized','Position',[0,0,1,1]);
uitabpanel(...
  'Parent',interface,...
  'TabPosition','lefttop',...
  'Units','pixels','Position',[0,0,1*WIDTH,0.93*HEIGHT],...
  'PanelBorderType','line','Title',{'  FRACTIONAL TUNING  ',},...
  'BackGroundColor',[0.078,0.169,0.549],'CreateFcn',@CreateTab);

%----------------------FUNCTION CREATE TAB-----------------------%
function CreateTab(varargin)
[htab,hpanel,hstatus] = varargin{[1,3,4]};
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------------------INPUT PANEL-------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E.input_panel = uipanel(...
  'Parent',hpanel(1),...
  'Units','normalized','Position',[0.01 0.01 0.43 0.98],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[0.82,0.863,0.922]);
%Input panel header
E.input_header = uicontrol(...
  'Parent',E.input_panel,...
  'Units','normalized','Position',[0.01,0.96,0.98,0.03],...
  'Style','text','String','INPUT DATA',...
  'FontName','courier new','FontSize',14,'FontWeight','bold',...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Center');

%%%%%%%%%%%%%%%%%%%----PLANT PARAMETERS PANEL----%%%%%%%%%%%%%%%%%%%
E.plant_panel = uipanel(...
  'Parent',E.input_panel,...
  'Units','normalized','Position',[0.03,0.75,0.94,0.20],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[1,1,1]);
E.plant_header = uicontrol(...
  'Parent',E.plant_panel,...
  'Units','normalized','Position',[0.01,0.85,0.48,0.10],...
  'Style','text','String','PLANT PARAMETERS',...
  'FontName','courier new','FontSize',11,'FontWeight','bold',...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
%Plant fdt image
axes(...
  'Parent',E.plant_panel,...
  'Units','pixel','Position',[10,5,271,96]);
fdt=imread('fdt.jpg');
imshow(fdt),axis off,hold on
%Text boxes
E.gain_text = uicontrol(...
  'Parent',E.plant_panel,...
  'Units','normalized','Position',[0.60,0.80,0.35,0.10],...
  'Style','text','String','Gain, k:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.constant_A_text = uicontrol(...
  'Parent',E.plant_panel,...
  'Units','normalized','Position',[0.60,0.62,0.35,0.10],...
  'Style','text','String','Constant A:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.constant_B_text = uicontrol(...
  'Parent',E.plant_panel,...
  'Units','normalized','Position',[0.60,0.45,0.35,0.10],...
  'Style','text','String','Constant B:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.constant_C_text = uicontrol(...
  'Parent',E.plant_panel,...
  'Units','normalized','Position',[0.60,0.28,0.35,0.10],...
  'Style','text','String','Constant C:',...
  'FontName','courier new','FontSize',10,...
```

```
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.delay_text = uicontrol(...
  'Parent',E.plant_panel,...
  'Units','normalized','Position',[0.60,0.10,0.35,0.10],...
  'Style','text','String','Delay, T (s):',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
%Data boxes
E.gain_data = uicontrol(...
  'Parent',E.plant_panel,'Enable','on',...
  'Units','normalized','Position',[0.82,0.80,0.10,0.14],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.A_data = uicontrol(...
  'Parent',E.plant_panel,'Enable','on',...
  'Units','normalized','Position',[0.82,0.62,0.10,0.14],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.B_data = uicontrol(...
  'Parent',E.plant_panel,'Enable','on',...
  'Units','normalized','Position',[0.82,0.45,0.10,0.14],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.C_data = uicontrol(...
  'Parent',E.plant_panel,'Enable','on',...
  'Units','normalized','Position',[0.82,0.28,0.10,0.14],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.delay_data = uicontrol(...
  'Parent',E.plant_panel,'Enable','on',...
  'Units','normalized','Position',[0.82,0.10,0.10,0.14],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%----DOMAIN CHOICE PANEL----%%%%%%%%%%%%%%%%%%%%
E.domain_panel = uipanel(...
  'Parent',E.input_panel,...
  'Units','normalized','Position',[0.03,0.58,0.94,0.16],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[1,1,1]);
axes(...
  'Parent',E.domain_panel,...
  'Units','pixel','Position',[0,0,514,85]);
domains=imread('domains.jpg');
imshow(domains),axis off,hold on
%Checkbox time-domain
E.domain_T = uicontrol(...
  'Parent',E.domain_panel,'Enable','on',...
  'Units','normalized','Position',[0.05,0.77,0.45,0.15],...
  'Style','checkbox','String','Time-Domain Tuning',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'Callback',@Check_Callback);
%Checkbox frequency-domain
E.domain_F = uicontrol(...
  'Parent',E.domain_panel,'Enable','on',...
  'Units','normalized','Position',[0.52,0.77,0.45,0.15],...
  'Style','checkbox','String','Frequency-Domain Tuning',...
```

```
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'Callback',@Check_Callback);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%-----GENERAL PARAMETERS PANEL-----%%%%%%%%%%%%%%%%
E.general_panel = uipanel(...
  'Parent',E.input_panel,...
  'Units','normalized','Position',[0.03,0.33,0.94,0.24],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[1,1,1]);
E.general_header = uicontrol(...
  'Parent',E.general_panel,...
  'Units','normalized','Position',[0.01,0.84,0.98,0.12],...
  'Style','text','String','GENERAL PARAMETERS',...
  'FontName','courier new','FontSize',11,'FontWeight','bold',...
  'TooltipString','Información sobre el botón',...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
%Text boxes
E.size_text = uicontrol(...
  'Parent',E.general_panel,...
  'Units','normalized','Position',[0.05,0.70,0.35,0.10],...
  'Style','text','String','Population size, NP:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.limit_text = uicontrol(...
  'Parent',E.general_panel,...
  'Units','normalized','Position',[0.05,0.55,0.35,0.10],...
  'Style','text','String','Upper limit iterations:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.interval_gain_text = uicontrol(...
  'Parent',E.general_panel,...
  'Units','normalized','Position',[0.05,0.40,0.90,0.10],...
  'Style','text','String','Initial interval for the gains:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.interval_exp_text = uicontrol(...
  'Parent',E.general_panel,...
  'Units','normalized','Position',[0.05,0.25,0.90,0.10],...
  'Style','text','String','Initial interval for the exp:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.internal_parameters_text = uicontrol(...
  'Parent',E.general_panel,...
  'Units','normalized','Position',[0.05,0.10,0.90,0.10],...
  'Style','text','String','DE internal parameters: F = ; CR =',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
%Data boxes
E.size_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.43,0.70,0.10,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.limit_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.43,0.55,0.10,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
```

```
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.gain_L_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.70,0.40,0.06,0.12],...
  'Style','edit','TooltipString','lower','FontName','courier new',
  'FontSize',10,'BackGroundColor',[1,1,1],...
  'HorizontalAlignment','Center');
E.gain_U_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.77,0.40,0.06,0.12],...
  'Style','edit','TooltipString','upper','FontName','courier new',
  'FontSize',10,'BackGroundColor',[1,1,1],...
  'HorizontalAlignment','Center');
E.exp_L_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.76,0.25,0.06,0.12],...
  'Style','edit','TooltipString','lower','FontName','courier new',
  'FontSize',10,'BackGroundColor',[1,1,1],...
  'HorizontalAlignment','Center');
E.exp_U_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.83,0.25,0.06,0.12],...
  'Style','edit','TooltipString','upper','FontName','courier new',
  'FontSize',10,'BackGroundColor',[1,1,1],...
  'HorizontalAlignment','Center');
E.parameter_F_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.455,0.10,0.08,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.parameter_CR_data = uicontrol(...
  'Parent',E.general_panel,'Enable','off',...
  'Units','normalized','Position',[0.635,0.10,0.08,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%----FREQUENCY PARAMETERS PANEL----%%%%%%%%%%%%%%%
E.frequency_panel = uipanel(...
  'Parent',E.input_panel,...
  'Units','normalized','Position',[0.03,0.07,0.94,0.25],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[1,1,1]);
E.frequency_header = uicontrol(...
  'Parent',E.frequency_panel,...
  'Units','normalized','Position',[0.01,0.84,0.98,0.12],...
  'Style','text','String','FREQUENCY PARAMETERS',...
  'FontName','courier new','FontSize',11,'FontWeight','bold',...
  'TooltipString','Información sobre el botón',...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
%Text boxes
E.interval_text = uicontrol(...
  'Parent',E.frequency_panel,...
  'Units','normalized','Position',[0.05,0.70,0.48,0.10],...
  'Style','text','String','Flat phase interval (decades):',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.crossover_F_text = uicontrol(...
```

```
  'Parent',E.frequency_panel,...
  'Units','normalized','Position',[0.05,0.55,0.90,0.10],...
  'Style','text','String','Gain crossover frequency (rad/s):',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.phase_text = uicontrol(...
  'Parent',E.frequency_panel,...
  'Units','normalized','Position',[0.05,0.40,0.48,0.10],...
  'Style','text','String','Minimum phase margin, pm (deg):',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.high_F_text = uicontrol(...
  'Parent',E.frequency_panel,...
  'Units','normalized','Position',[0.05,0.25,0.90,0.10],...
  'Style','text','String','High frequency noise rejection:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
E.sensitivity_text = uicontrol(...
  'Parent',E.frequency_panel,...
  'Units','normalized','Position',[0.05,0.10,0.90,0.10],...
  'Style','text','String','Sensitivity:      (rad/s);    (dB)',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Left');
%Data boxes
E.interval_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.55,0.70,0.10,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.minwcg_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.65,0.55,0.06,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.maxwcg_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.825,0.55,0.08,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.phase_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.55,0.40,0.10,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.high_F_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.52,0.25,0.08,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.high_F_dB_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.725,0.25,0.08,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.sensitivity_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.24,0.10,0.08,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
```

```
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
E.sensitivity_dB_data = uicontrol(...
  'Parent',E.frequency_panel,'Enable','off',...
  'Units','normalized','Position',[0.445,0.10,0.08,0.12],...
  'Style','edit','FontName','courier new','FontSize',10,...
  'BackGroundColor',[1,1,1],'HorizontalAlignment','Center');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%-----INPUT BUTTONS-----%%%%%%%%%%%%%%%%%%%%%%%%
E.new_button = uicontrol (...
  'Parent',E.input_panel,'Enable','off',...
  'Units','normalized','Position',[0.10,0.015,0.2,0.04],...
  'Style','pushbutton','String','New Data',...
  'FontName','courier new','FontSize',12);
%Checkbox test data
E.test_button = uicontrol(...
  'Parent',E.input_panel,'Enable','on',...
  'Units','normalized','Position',[0.35,0.015,0.2,0.04],...
  'Style','checkbox','String','Test Data',...
  'FontName','courier new','FontSize',11,...
  'BackGroundColor',[0.82,0.863,0.922],'Callback',@Test_Callback);
E.run_button = uicontrol (...
  'Parent',E.input_panel,'Enable','off',...
  'Units','normalized','Position',[0.60,0.015,0.30,0.04],...
  'Style','pushbutton','String','Run',...
  'FontName','courier new','FontSize',12);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%-----------------Subfunction to choose domain-------------------%
function Check_Callback(hObject, eventdata, handles)
  set(E.new_button,'Enable','off');

  if hObject == E.domain_T
    if get(E.domain_T,'Value') == 1
      set(E.domain_F,'Value',0);
      set(E.size_data,'Enable','on');
      set(E.limit_data,'Enable','on');
      set(E.gain_L_data,'Enable','on');
      set(E.gain_U_data,'Enable','on');
      set(E.exp_L_data,'Enable','on');
      set(E.exp_U_data,'Enable','on');
      set(E.parameter_F_data,'Enable','on');
      set(E.parameter_CR_data,'Enable','on');
      set(E.interval_data,'Enable','off');
      set(E.minwcg_data,'Enable','off');
      set(E.maxwcg_data,'Enable','off');
      set(E.phase_data,'Enable','off');
      set(E.high_F_data,'Enable','off');
      set(E.high_F_dB_data,'Enable','off');
      set(E.sensitivity_data,'Enable','off');
      set(E.sensitivity_dB_data,'Enable','off');
      set(E.new_button,'Enable','off');
      E.run_button = uicontrol (...
        'Parent',E.input_panel,'Enable','on',...
        'Units','normalized',...
        'Position',[0.60,0.015,0.30,0.04],...
```

```
        'Style','pushbutton','String','Run',...
        'FontName','courier new','FontSize',12,...
        'Callback',@Time_Domain_Callback);
    elseif get(E.domain_T,'Value') == 0
      set(E.size_data,'Enable','off');
      set(E.limit_data,'Enable','off');
      set(E.gain_L_data,'Enable','off');
      set(E.gain_U_data,'Enable','off');
      set(E.exp_L_data,'Enable','off');
      set(E.exp_U_data,'Enable','off');
      set(E.parameter_F_data,'Enable','off');
      set(E.parameter_CR_data,'Enable','off');
      set(E.run_button,'Enable','off');
    end
  elseif hObject == E.domain_F
    if get(E.domain_F,'Value') == 1
      set(E.domain_T,'Value',0);
      set(E.size_data,'Enable','on');
      set(E.limit_data,'Enable','on');
      set(E.gain_L_data,'Enable','on');
      set(E.gain_U_data,'Enable','on');
      set(E.exp_L_data,'Enable','on');
      set(E.exp_U_data,'Enable','on');
      set(E.parameter_F_data,'Enable','on');
      set(E.parameter_CR_data,'Enable','on');
      set(E.interval_data,'Enable','on');
      set(E.minwcg_data,'Enable','on');
      set(E.maxwcg_data,'Enable','on');
      set(E.phase_data,'Enable','on');
      set(E.high_F_data,'Enable','on');
      set(E.high_F_dB_data,'Enable','on');
      set(E.sensitivity_data,'Enable','on');
      set(E.sensitivity_dB_data,'Enable','on');
      set(E.new_button,'Enable','off');
      E.run_button = uicontrol (...
        'Parent',E.input_panel,'Enable','on',...
        'Units','normalized',...
        'Position',[0.60,0.015,0.30,0.04],...
        'Style','pushbutton','String','Run',...
        'FontName','courier new','FontSize',12,...
        'Callback',@Frequency_Domain_Callback);
    elseif get(E.domain_F,'Value') == 0
      set(E.size_data,'Enable','off');
      set(E.limit_data,'Enable','off');
      set(E.gain_L_data,'Enable','off');
      set(E.gain_U_data,'Enable','off');
      set(E.exp_L_data,'Enable','off');
      set(E.exp_U_data,'Enable','off');
      set(E.parameter_F_data,'Enable','off');
      set(E.parameter_CR_data,'Enable','off');
      set(E.interval_data,'Enable','off');
      set(E.minwcg_data,'Enable','off');
      set(E.maxwcg_data,'Enable','off');
      set(E.phase_data,'Enable','off');
      set(E.high_F_data,'Enable','off');
      set(E.high_F_dB_data,'Enable','off');
      set(E.sensitivity_data,'Enable','off');
      set(E.sensitivity_dB_data,'Enable','off');
```

```
      set(E.run_button,'Enable','off');
    end
  end
end
%--------------------------------------------------------------------%


%-----------------Subfunction to set test values-----------------%
function Test_Callback(hObject, eventdata, handles)
  if get(E.test_button,'Value') == 1
    %Set plant parameters
    set(E.gain_data,'String','3');
    set(E.A_data,'String','0.2');
    set(E.B_data,'String','1');
    set(E.C_data,'String','0');
    set(E.delay_data,'String','0');
    %Set general parameters
    set(E.size_data,'String','20');
    set(E.limit_data,'String','20');
    set(E.gain_L_data,'String','0');
    set(E.gain_U_data,'String','5');
    set(E.exp_L_data,'String','0');
    set(E.exp_U_data,'String','1');
    set(E.parameter_F_data,'String','0.5');
    set(E.parameter_CR_data,'String','0.5');
    %Set frequency parameters
    set(E.interval_data,'String','0.5');
    set(E.minwcg_data,'String','1');
    set(E.maxwcg_data,'String','50');
    set(E.phase_data,'String','10');
    set(E.high_F_data,'String','100');
    set(E.high_F_dB_data,'String','10');
    set(E.sensitivity_data,'String','0.01');
    set(E.sensitivity_dB_data,'String','10');
  elseif get(E.test_button,'Value') == 0
    %Reset all parameters
    set(E.gain_data,'String','');
    set(E.A_data,'String','');
    set(E.B_data,'String','');
    set(E.C_data,'String','');
    set(E.delay_data,'String','');
    set(E.size_data,'String','');
    set(E.limit_data,'String','');
    set(E.gain_L_data,'String','');
    set(E.gain_U_data,'String','');
    set(E.exp_L_data,'String','');
    set(E.exp_U_data,'String','');
    set(E.parameter_F_data,'String','');
    set(E.parameter_CR_data,'String','');
    set(E.interval_data,'String','');
    set(E.minwcg_data,'String','');
    set(E.maxwcg_data,'String','');
    set(E.phase_data,'String','');
    set(E.high_F_data,'String','');
    set(E.high_F_dB_data,'String','');
    set(E.sensitivity_data,'String','');
    set(E.sensitivity_dB_data,'String','');
```

```
  end
end
%--------------------------------------------------------------------%


%----------------Subfunctions to execute program-----------------%
function Time_Domain_Callback(hObject, eventdata, handles)
  global tuning
    tuning = 1; %Option for time domain tuning
    E.gain_str = get(E.gain_data,'String');
    E.A_str = get(E.A_data,'String');
    E.B_str = get(E.B_data,'String');
    E.C_str = get(E.C_data,'String');
    E.delay_str = get(E.delay_data,'String');
    E.size_str = get(E.size_data,'String');
    E.limit_str = get(E.limit_data,'String');
    E.gain_U_str = get(E.gain_U_data,'String');
    E.exp_U_str = get(E.exp_U_data,'String');
    E.parameter_F_str = get(E.parameter_F_data,'String');
    E.parameter_CR_str = get(E.parameter_CR_data,'String');
  %Check no empty boxes
  if strcmp('',E.gain_str)||strcmp('',E.A_str)||...
     strcmp('',E.B_str)||strcmp('',E.C_str)||...
     strcmp('',E.delay_str)|| strcmp('',E.size_str)||...
     strcmp('',E.limit_str)|| strcmp('',E.gain_U_str)||...
     strcmp('',E.exp_U_str)|| strcmp('',E.parameter_F_str)||...
     strcmp('',E.parameter_CR_str)
    msgbox('Input data missing. Please check','Error','error');
  else
    S.gain = str2num(E.gain_str);
    S.A_constant = str2num(E.A_str);
    S.B_constant = str2num(E.B_str);
    S.C_constant = str2num(E.C_str);
    S.delay = str2num(E.delay_str);
    S.size = str2num(E.size_str);
    S.limit = str2num(E.limit_str);
    S.gain_U = str2num(E.gain_U_str);
    S.exp_U = str2num(E.exp_U_str);
    S.parameter_F = str2num(E.parameter_F_str);
    S.parameter_CR = str2num(E.parameter_CR_str);
    %Call to function Control_Fraccionario!!!!!
    [S.out_P,S.out_I,S.out_D,S.out_orderI,S.out_orderD,...
    S.out_frequency,S.out_phase,S.out_overshoot,S.out_peaktime,...
    S.out_settlingtime,S.out_costvalue] = Control_Fraccionario(...
    tuning,S.gain,S.A_constant,S.B_constant,S.C_constant,...
    S.delay,S.size,S.limit,S.gain_U,S.exp_U,S.parameter_F,...
    S.parameter_CR,EJE.iteration_plot,EJE.magnitude_bode_plot,...
    EJE.phase_bode_plot);
    %Turn variables num to string
    E.conv_P = num2str(S.out_P,4);
    E.conv_I = num2str(S.out_I,4);
    E.conv_D = num2str(S.out_D,4);
    E.conv_orderI = num2str(S.out_orderI,4);
    E.conv_orderD = num2str(S.out_orderD,4);
    E.conv_frequency = num2str(S.out_frequency,4);
    E.conv_phase = num2str(S.out_phase,4);
    E.conv_overshoot = num2str(S.out_overshoot,4);
```

```
    E.conv_peaktime = num2str(S.out_peaktime,4);
    E.conv_settlingtime = num2str(S.out_settlingtime,4);
    E.conv_costvalue = num2str(S.out_costvalue,4);
    %Call subfunction
    Display_Data;
  end
end

function Frequency_Domain_Callback(hObject, eventdata, handles)
  global tuning Fparam
    tuning = 3; %Option for frequency domain tuning
    Fparam = zeros(1,8);
    E.gain_str = get(E.gain_data,'String');
    E.A_str = get(E.A_data,'String');
    E.B_str = get(E.B_data,'String');
    E.C_str = get(E.C_data,'String');
    E.delay_str = get(E.delay_data,'String');
    E.size_str = get(E.size_data,'String');
    E.limit_str = get(E.limit_data,'String');
    E.gain_U_str = get(E.gain_U_data,'String');
    E.exp_U_str = get(E.exp_U_data,'String');
    E.parameter_F_str = get(E.parameter_F_data,'String');
    E.parameter_CR_str = get(E.parameter_CR_data,'String');
    E.interval_str = get(E.interval_data,'String');
    E.minwcg_str = get(E.minwcg_data,'String');
    E.maxwcg_str = get(E.maxwcg_data,'String');
    E.phase_str = get(E.phase_data,'String');
    E.high_F_str = get(E.high_F_data,'String');
    E.high_F_dB_str = get(E.high_F_dB_data,'String');
    E.sensitivity_str = get(E.sensitivity_data,'String');
    E.sensitivity_dB_str = get(E.sensitivity_dB_data,'String');
  %Check no empty boxes
  if strcmp('',E.gain_str)||strcmp('',E.A_str)||...
     strcmp('',E.B_str)||strcmp('',E.C_str)||...
     strcmp('',E.delay_str)||strcmp('',E.size_str)|| ...
     strcmp('',E.limit_str)||strcmp('',E.gain_U_str)|| ...
     strcmp('',E.exp_U_str)||strcmp('',E.parameter_F_str)|| ...
     strcmp('',E.parameter_CR_str)||...
     strcmp('',E.interval_str)||strcmp('',E.minwcg_str)||...
     strcmp('',E.maxwcg_str)||strcmp('',E.phase_str)||...
     strcmp('',E.high_F_str)||strcmp('',E.high_F_dB_str)||...
     strcmp('',E.sensitivity_str)||strcmp('',E.sensitivity_dB_str)
    msgbox('Input data missing. Please check','Error','error');
  else
    S.gain = str2num(E.gain_str);
    S.A_constant = str2num(E.A_str);
    S.B_constant = str2num(E.B_str);
    S.C_constant = str2num(E.C_str);
    S.delay = str2num(E.delay_str);
    S.size = str2num(E.size_str);
    S.limit = str2num(E.limit_str);
    S.gain_U = str2num(E.gain_U_str);
    S.exp_U = str2num(E.exp_U_str);
    S.parameter_F = str2num(E.parameter_F_str);
    S.parameter_CR = str2num(E.parameter_CR_str);
    S.interval = str2num(E.interval_str);
    S.minwcg =str2num(E.minwcg_str);
```

```
    S.maxwcg =str2num(E.maxwcg_str);
    S.phase = str2num(E.phase_str);
    S.high_F = str2num(E.high_F_str);
    S.high_F_dB = str2num(E.high_F_dB_str);
    S.sensitivity = str2num(E.sensitivity_str);
    S.sensitivity_dB = str2num(E.sensitivity_dB_str);
    Fparam(1,1)= S.interval;
    Fparam(1,2)= S.minwcg;
    Fparam(1,3)= S.maxwcg;
    Fparam(1,4)= S.phase;
    Fparam(1,5)= S.high_F;
    Fparam(1,6)= S.high_F_dB;
    Fparam(1,7)= S.sensitivity;
    Fparam(1,8)= S.sensitivity_dB;
    %Call to function Control_Fraccionario!!!!!
    [S.out_P,S.out_I,S.out_D,S.out_orderI,S.out_orderD,...
    S.out_frequency,S.out_phase,S.out_overshoot,S.out_peaktime,...
    S.out_settlingtime,S.out_costvalue] = Control_Fraccionario(...
    tuning,S.gain,S.A_constant,S.B_constant,S.C_constant,...
    S.delay,S.size,S.limit,S.gain_U,S.exp_U,S.parameter_F,...
    S.parameter_CR,EJE.iteration_plot,EJE.magnitude_bode_plot,...
    EJE.phase_bode_plot,Fparam);
    %Turn variables num to string
    E.conv_P = num2str(S.out_P,4);
    E.conv_I = num2str(S.out_I,4);
    E.conv_D = num2str(S.out_D,4);
    E.conv_orderI = num2str(S.out_orderI,4);
    E.conv_orderD = num2str(S.out_orderD,4);
    E.conv_frequency = num2str(S.out_frequency,4);
    E.conv_phase = num2str(S.out_phase,4);
    E.conv_overshoot = num2str(S.out_overshoot,4);
    E.conv_peaktime = num2str(S.out_peaktime,4);
    E.conv_settlingtime = num2str(S.out_settlingtime,4);
    E.conv_costvalue = num2str(S.out_costvalue,4);
    %Call subfunction
    Display_Data;
  end
end
%-----------------------------------------------------------------%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------OUTPUT PANEL-------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E.output_panel = uipanel(...
  'Parent',hpanel(1),...
  'Units','normalized','Position',[0.46 0.25 0.14 0.65],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[1,1,1]);

%%%%%%%%%%%%%%%%%%%%----PID PARAMETERS PANEL----%%%%%%%%%%%%%%%%%%%%
E.pid_panel = uipanel(...
  'Parent',E.output_panel,...
  'Units','normalized','Position',[0.03,0.60,0.94,0.38],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[0.82,0.863,0.922]);
E.pid_header = uicontrol(...
  'Parent',E.pid_panel,...
  'Units','normalized','Position',[0.01,0.85,0.98,0.10],...
```

```
  'Style','text','String','PID PARAMETERS',...
  'FontName','courier new','FontSize',13,'FontWeight','bold',...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Center');
%----Proportional----%
E.out_P_text = uicontrol(...
  'Parent',E.pid_panel,...
  'Units','normalized','Position',[0.12,0.66,0.55,0.10],...
  'Style','text','String','kp:','TooltipString','Proportional',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_P = uicontrol(...
  'Parent',E.pid_panel,'Enable','on',...
  'Units','normalized','Position',[0.55,0.66,0.35,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Integral----%
E.out_I_text = uicontrol(...
  'Parent',E.pid_panel,...
  'Units','normalized','Position',[0.12,0.51,0.55,0.10],...
  'Style','text','String','ki:','TooltipString','Integral',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_I = uicontrol(...
  'Parent',E.pid_panel,'Enable','on',...
  'Units','normalized','Position',[0.55,0.51,0.35,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Derivative----%
E.out_D_text = uicontrol(...
  'Parent',E.pid_panel,...
  'Units','normalized','Position',[0.12,0.36,0.55,0.10],...
  'Style','text','String','kd:','TooltipString','Derivative',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_D = uicontrol(...
  'Parent',E.pid_panel,'Enable','on',...
  'Units','normalized','Position',[0.55,0.36,0.35,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Integrator order----%
E.out_orderI_text = uicontrol(...
  'Parent',E.pid_panel,...
  'Units','normalized','Position',[0.12,0.21,0.55,0.10],...
  'Style','text','String','lambda:','TooltipString','Intorder',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_orderI = uicontrol(...
  'Parent',E.pid_panel,'Enable','on',...
  'Units','normalized','Position',[0.55,0.21,0.35,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
```

```
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Differentiator order----%
E.out_orderD_text = uicontrol(...
  'Parent',E.pid_panel,...
  'Units','normalized','Position',[0.12,0.06,0.55,0.10],...
  'Style','text','String','nu:','TooltipString','Differorder',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_orderD = uicontrol(...
  'Parent',E.pid_panel,'Enable','on',...
  'Units','normalized','Position',[0.55,0.06,0.35,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%----RESULTS PANEL----%%%%%%%%%%%%%%%%%%%%%%%%%
E.results_panel = uipanel(...
  'Parent',E.output_panel,...
  'Units','normalized','Position',[0.03,0.02,0.94,0.43],...
  'ForeGroundColor',[1,1,1],'BackgroundColor',[0.82,0.863,0.922]);
E.results_header = uicontrol(...
  'Parent',E.results_panel,...
  'Units','normalized','Position',[0.01,0.85,0.98,0.10],...
  'Style','text','String','RESULTS',...
  'FontName','courier new','FontSize',13,'FontWeight','bold',...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Center');
%----Crossover frequency----%
E.out_frequency_text = uicontrol(...
  'Parent',E.results_panel,...
  'Units','normalized','Position',[0.07,0.70,0.60,0.08],...
  'Style','text','String','wcg (rad/s):','TooltipString','CF',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_frequency = uicontrol(...
  'Parent',E.results_panel,'Enable','on',...
  'Units','normalized','Position',[0.67,0.70,0.25,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Phase margin----%
E.out_phase_text = uicontrol(...
  'Parent',E.results_panel,...
  'Units','normalized','Position',[0.07,0.57,0.60,0.08],...
  'Style','text','String','pm (deg):','TooltipString','margin',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_phase = uicontrol(...
  'Parent',E.results_panel,'Enable','on',...
  'Units','normalized','Position',[0.67,0.57,0.25,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
```

```
  'HorizontalAlignment','Center');
%----Overshoot----%
E.out_overshoot_text = uicontrol(...
  'Parent',E.results_panel,...
  'Units','normalized','Position',[0.07,0.44,0.60,0.08],...
  'Style','text','String','Os (%):','TooltipString','Overshoot',..
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_overshoot = uicontrol(...
  'Parent',E.results_panel,'Enable','on',...
  'Units','normalized','Position',[0.67,0.44,0.25,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Peak time----%
E.out_peaktime_text = uicontrol(...
  'Parent',E.results_panel,...
  'Units','normalized','Position',[0.07,0.31,0.60,0.08],...
  'Style','text','String','tp (s):','TooltipString','Peaktime',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_peaktime = uicontrol(...
  'Parent',E.results_panel,'Enable','on',...
  'Units','normalized','Position',[0.67,0.31,0.25,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Settling time----%
E.out_settlingtime_text = uicontrol(...
  'Parent',E.results_panel,...
  'Units','normalized','Position',[0.07,0.18,0.60,0.08],...
  'Style','text','String','ts (s):','TooltipString','Settime',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_settlingtime = uicontrol(...
  'Parent',E.results_panel,'Enable','on',...
  'Units','normalized','Position',[0.67,0.18,0.25,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%----Cost value----%
E.out_costvalue_text = uicontrol(...
  'Parent',E.results_panel,...
  'Units','normalized','Position',[0.07,0.05,0.60,0.08],...
  'Style','text','String','Cost value:',...
  'FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.82,0.863,0.922],...
  'HorizontalAlignment','Left');
E.out_costvalue = uicontrol(...
  'Parent',E.results_panel,'Enable','on',...
  'Units','normalized','Position',[0.67,0.05,0.25,0.10],...
  'Style','text','FontName','courier new','FontSize',10,...
  'BackGroundColor',[0.859,0.843,0.808],...
  'HorizontalAlignment','Center');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%----GRAPHICS----%%%%%%%%%%%%%%%%%%%%%%%%
E.graphics_panel = uipanel(...
  'Parent',hpanel(1),...
  'Units','normalized','Position',[0.61 0.01 0.38 0.89],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[0.82,0.863,0.922]);
%----Iteration plot----%
EJE.iteration_plot =
axes('Parent',hpanel(1),'Position',[0.655,0.64,0.32,0.20]);
title(EJE.iteration_plot,'Temporal Response of the System',...
    'FontName','courier new','FontSize',14,'FontWeight','bold');
axis([0 30 0 15])
ylabel(EJE.iteration_plot,'Amplitude',...
    'FontName','courier new','FontSize',10,'FontWeight','bold');
xlabel(EJE.iteration_plot,'Time (s)',...
    'FontName','courier new','FontSize',9,'FontWeight','bold');
grid on;
%----Magnitude plot----%
EJE.magnitude_bode_plot =
axes('Parent',hpanel(1),'Position',[0.655,0.33,0.32,0.20]);
x = 1000;
y = -50:10:100;
semilogx(x,y);
title(EJE.magnitude_bode_plot,'Bode Diagram of the System',...
    'FontName','courier new','FontSize',14,'FontWeight','bold');
ylabel(EJE.magnitude_bode_plot,'Magnitude (dB)',...
    'FontName','courier new','FontSize',10,'FontWeight','bold');
grid on;
%----Phase plot----%
EJE.phase_bode_plot =
axes('Parent',hpanel(1),'Position',[0.655,0.085,0.32,0.20]);
x = 1000;
y = -50:10:100;
semilogx(x,y);
ylabel(EJE.phase_bode_plot,'Phase (deg)',...
    'FontName','courier new','FontSize',10,'FontWeight','bold');
xlabel(EJE.phase_bode_plot,'Frequency (rad/s)',...
    'FontName','courier new','FontSize',9,'FontWeight','bold');
grid on;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%-----OUTPUT CONTROLS-----%%%%%%%%%%%%%%%%%%%%
%Button to save experiment data in a file.mat
E.save_data_button = uicontrol(...
  'Parent',hpanel(1),'Enable','on',...
  'Units','normalized','Position',[0.47,0.03,0.12,0.05],...
  'Style','pushbutton','String','Save data(.mat)',...
  'FontName','courier new','FontSize',11,'Callback',@Save_Data);
%Button to save an screen image
E.save_screen_button = uicontrol(...
  'Parent',hpanel(1),'Enable','on',...
  'Units','normalized','Position',[0.47,0.09,0.12,0.05],...
  'Style','pushbutton','String','Save screenshot',...
  'FontName','courier new','FontSize',11,...
  'Callback',@Save_Interface);
%Help button
E.help_button = uicontrol(...
```

```
  'Parent',hpanel(1),'Enable','on',...
  'Units','normalized','Position',[0.50,0.17,0.06,0.05],...
  'Style','pushbutton','String','HELP(?)',...
  'FontName','courier new','FontSize',11,...
  'BackGroundColor',[1,1,0.5],'Callback',@Read_PDF);
%UC3M logo
axes(...
  'Parent',hpanel(1),...
  'Units','normalized','Position',[0.82,0.90,0.14,0.10]);
uc3m=imread('uc3m.jpg');
imshow(uc3m),axis off,hold on
%Fractional order PID image
axes(...
  'Parent',E.output_panel,...
  'Units','pixel','Position',[0,198,197,56]);
pid=imread('pid.jpg');
imshow(pid),axis off,hold on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%----------------------------CONNECTION----------------------------%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------------Subfunction to show screen results-----------------%
function Display_Data ()
  E.out_P = uicontrol(...
    'Parent',E.pid_panel,'Enable','on',...
    'Units','normalized','Position',[0.55,0.66,0.35,0.10],...
    'Style','text','String',E.conv_P,'FontName','courier new',
    'FontSize',10,'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_I = uicontrol(...
    'Parent',E.pid_panel,'Enable','on',...
    'Units','normalized','Position',[0.55,0.51,0.35,0.10],...
    'Style','text','String',E.conv_I,'FontName','courier new',
    'FontSize',10,'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_D = uicontrol(...
    'Parent',E.pid_panel,'Enable','on',...
    'Units','normalized','Position',[0.55,0.36,0.35,0.10],...
    'Style','text','String',E.conv_D,'FontName','courier new',
    'FontSize',10,'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_orderI = uicontrol(...
    'Parent',E.pid_panel,'Enable','on',...
    'Units','normalized','Position',[0.55,0.21,0.35,0.10],...
    'Style','text','String',E.conv_orderI,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_orderD = uicontrol(...
    'Parent',E.pid_panel,'Enable','on',...
    'Units','normalized','Position',[0.55,0.06,0.35,0.10],...
    'Style','text','String',E.conv_orderD,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
```

```
  E.out_frequency = uicontrol(...
    'Parent',E.results_panel,'Enable','on',...
    'Units','normalized','Position',[0.67,0.70,0.25,0.10],...
    'Style','text','String',E.conv_frequency,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_phase = uicontrol(...
    'Parent',E.results_panel,'Enable','on',...
    'Units','normalized','Position',[0.67,0.57,0.25,0.10],...
    'Style','text','String',E.conv_phase,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_overshoot = uicontrol(...
    'Parent',E.results_panel,'Enable','on',...
    'Units','normalized','Position',[0.67,0.44,0.25,0.10],...
    'Style','text','String',E.conv_overshoot,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_peaktime = uicontrol(...
    'Parent',E.results_panel,'Enable','on',...
    'Units','normalized','Position',[0.67,0.31,0.25,0.10],...
    'Style','text','String',E.conv_peaktime,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_settlingtime = uicontrol(...
    'Parent',E.results_panel,'Enable','on',...
    'Units','normalized','Position',[0.67,0.18,0.25,0.10],...
    'Style','text','String',E.conv_settlingtime,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.out_costvalue = uicontrol(...
    'Parent',E.results_panel,'Enable','on',...
    'Units','normalized','Position',[0.67,0.05,0.25,0.10],...
    'Style','text','String',E.conv_costvalue,'FontName',...
    'courier new','FontSize',10,...
    'BackGroundColor',[0.859,0.843,0.808],...
    'HorizontalAlignment','Center');
  E.new_button = uicontrol (...
    'Parent',E.input_panel,'Enable','on',...
    'Units','normalized','Position',[0.10,0.015,0.2,0.04],...
    'Style','pushbutton','String','New Data',...
    'FontName','courier new','FontSize',12,'Callback',@New_Data);
  set(E.test_button,'Enable','off');
  set(E.run_button,'Enable','off');
  set(E.domain_T,'Enable','off');
  set(E.domain_F,'Enable','off');
end
%-----------------------------------------------------------------%

%----------Subfunction to reset and enter new values------------%
function New_Data(hObject, eventdata, handles)
  set(E.domain_T,'Enable','on','Value',0);
```

```
  set(E.domain_F,'Enable','on','Value',0);
  set(E.gain_data,'String','','Enable','on');
  set(E.A_data,'String','','Enable','on');
  set(E.B_data,'String','','Enable','on');
  set(E.C_data,'String','','Enable','on');
  set(E.delay_data,'String','','Enable','on');
  set(E.size_data,'String','','Enable','off');
  set(E.limit_data,'String','','Enable','off');
  set(E.gain_L_data,'String','','Enable','off');
  set(E.gain_U_data,'String','','Enable','off');
  set(E.exp_L_data,'String','','Enable','off');
  set(E.exp_U_data,'String','','Enable','off');
  set(E.parameter_F_data,'String','','Enable','off');
  set(E.parameter_CR_data,'String','','Enable','off');
  set(E.interval_data,'String','','Enable','off');
  set(E.minwcg_data,'String','','Enable','off');
  set(E.maxwcg_data,'String','','Enable','off');
  set(E.phase_data,'String','','Enable','off');
  set(E.high_F_data,'String','','Enable','off');
  set(E.high_F_dB_data,'String','','Enable','off');
  set(E.sensitivity_data,'String','','Enable','off');
  set(E.sensitivity_dB_data,'String','','Enable','off');
  set(E.out_P,'String','','Enable','off');
  set(E.out_I,'String','','Enable','off');
  set(E.out_D,'String','','Enable','off');
  set(E.out_orderI,'String','','Enable','off');
  set(E.out_orderD,'String','','Enable','off');
  set(E.out_frequency,'String','','Enable','off');
  set(E.out_phase,'String','','Enable','off');
  set(E.out_overshoot,'String','','Enable','off');
  set(E.out_peaktime,'String','','Enable','off');
  set(E.out_settlingtime,'String','','Enable','off');
  set(E.out_costvalue,'String','','Enable','off');
  set(E.test_button,'Enable','on','Value',0);
  set(E.run_button,'Enable','off');

E.graphics_panel = uipanel(...
  'Parent',hpanel(1),...
  'Units','normalized','Position',[0.61 0.01 0.38 0.89],...
  'ForeGroundColor',[1,1,1],'BackGroundColor',[0.82,0.863,0.922]);
%----Iteration plot----%
EJE.iteration_plot =
axes('Parent',hpanel(1),'Position',[0.655,0.64,0.32,0.20]);
title(EJE.iteration_plot,'Temporal Response of the System',...
    'FontName','courier new','FontSize',14,'FontWeight','bold');
axis([0 30 0 15])
ylabel(EJE.iteration_plot,'Amplitude',...
    'FontName','courier new','FontSize',10,'FontWeight','bold');
xlabel(EJE.iteration_plot,'Time (s)',...
    'FontName','courier new','FontSize',9,'FontWeight','bold');
grid on;
%----Magnitude plot----%
EJE.magnitude_bode_plot =
axes('Parent',hpanel(1),'Position',[0.655,0.33,0.32,0.20]);
x = 1000;
y = -50:10:100;
semilogx(x,y);
```

```
title(EJE.magnitude_bode_plot,'Bode Diagram of the System',...
    'FontName','courier new','FontSize',14,'FontWeight','bold');
ylabel(EJE.magnitude_bode_plot,'Magnitude (dB)',...
    'FontName','courier new','FontSize',10,'FontWeight','bold');
grid on;
%----Phase plot----%
EJE.phase_bode_plot =
axes('Parent',hpanel(1),'Position',[0.655,0.085,0.32,0.20]);
x = 1000;
y = -50:10:100;
semilogx(x,y);
ylabel(EJE.phase_bode_plot,'Phase (deg)',...
    'FontName','courier new','FontSize',10,'FontWeight','bold');
xlabel(EJE.phase_bode_plot,'Frequency (rad/s)',...
    'FontName','courier new','FontSize',9,'FontWeight','bold');
grid on;
end
%--------------------------------------------------------------------%

%------------Subfunction to save data in a file.mat--------------%
function Save_Data(hObject, eventdata, handles)
  [filename,pathname] = uiputfile('*.mat');
  save(filename,'*S')
  clear *S;
end
%--------------------------------------------------------------------%

%-------------Subfunction to save an screen image----------------%
function Save_Interface(varargin)
  [filename,pathname] = uiputfile('*.jpg');
  hgexport(gcf,filename,hgexport('factorystyle'),'Format','jpeg');
end
%--------------------------------------------------------------------%

%-------------------Subfunction to show help---------------------%
function Read_PDF (varargin)
  open('Help.pdf');
end
%--------------------------------------------------------------------%
end
%-------------------------END CREATE TAB-------------------------%

set(interface,'Visible','on')
maximize(interface)
delete(loading)
%%%%%%%%%%%%%%%%%%%%%%%%-----END INTERFACE-----%%%%%%%%%%%%%%%%%%%%%%
end

%%%%%%%%%%%%%%%%%%%%%%%%-----END MAIN-----%%%%%%%%%%%%%%%%%%%%%%%%%%
```

ANEXO 2: Documento de ayuda

En este anexo se adjunta el documento de ayuda donde se detallan todos los pasos que el usuario debe seguir para el correcto uso de la interfaz. Se trata de un documento de tipo PDF que se abre a partir del botón de "*HELP*" incluido en la interfaz.

FRACTIONAL PID TUNING BY DIFFERENTIAL EVOLUTION ALGORITHM

Here some guidelines are summarized for the use of the tuning toolbox:

1. In the ***Input Data*** panel, introduce the *Plant Parameters* (Figure 1, box 1) to be tuned.

2. Next, choose one of the two tuning methods (Figure 1, box 2): *Time-Domain Tuning* or *Frequency-Domain Tuning*. The user has to pick one of those to start tuning the system.

3. Fill input data out, which are enabled after tuning method is chosen one step before (Figure 1, box 3). Do not forget to take into account units of each variable: (s) seconds; (rad) radians; (decades) decades; (deg) degrees; (dB) decibels.

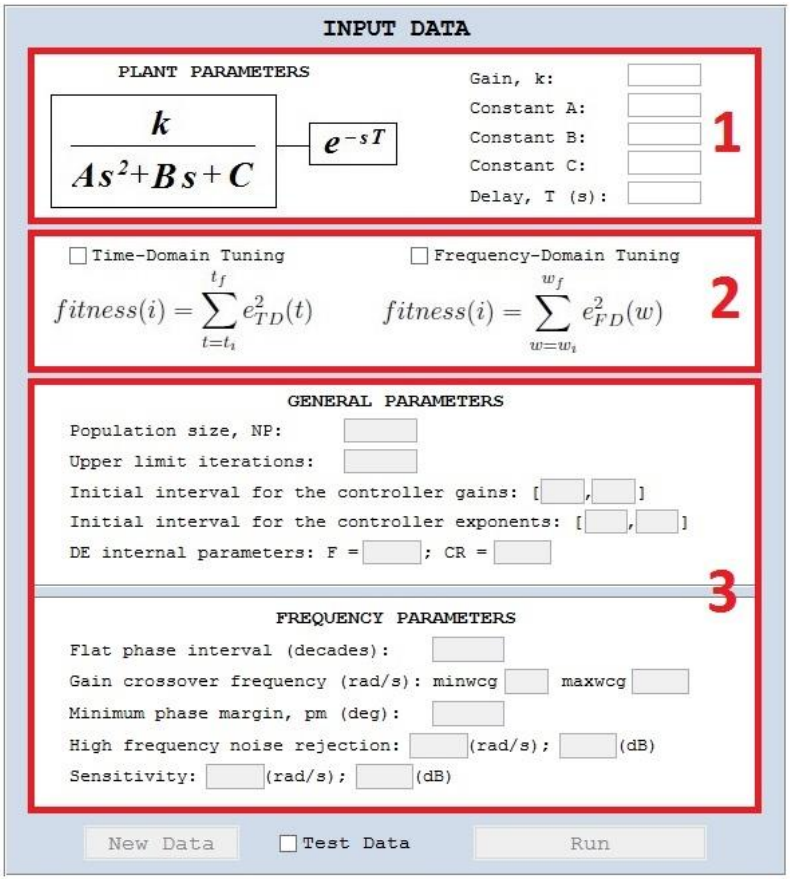

Figure 1. *Input Data* panel.

NOTE: The user can select *Test Data* box (Figure 2) to set all required input data with predefined values.

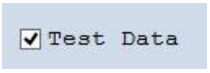

Figure 2. *Test Data* box.

4. Click on ***Run*** button to continue. If some of input data is incomplete, next error message will appear: *Input data missing. Please check.* (Figure 3)

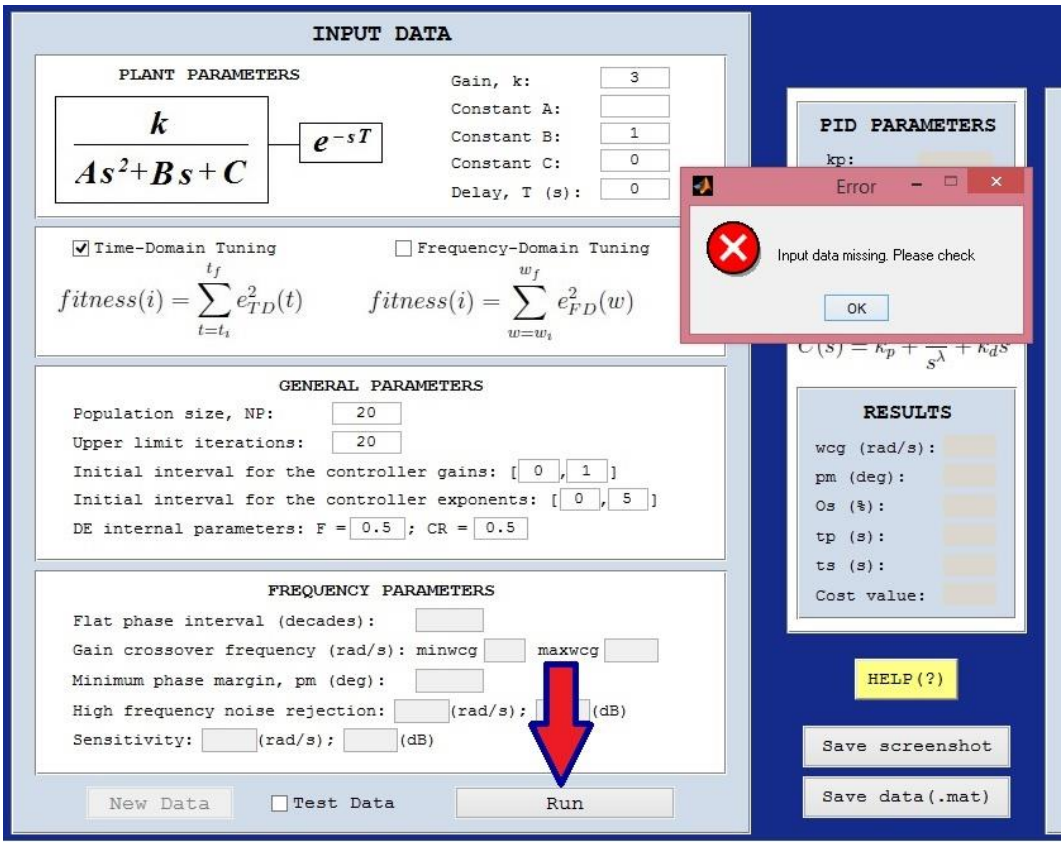

Figure 3. Error message.

5. Wait until the process is completed. Timeout is indicated in a new screen (Figure 4).

Figure 4. Waiting window.

6. Analyze the obtained results once the tuning is finished. The results are shown on *PID Parameters* and *Results* panels (Figure 5). Remember to take into account units of each variable: (deg) degrees; (%) per cent; (s) seconds.

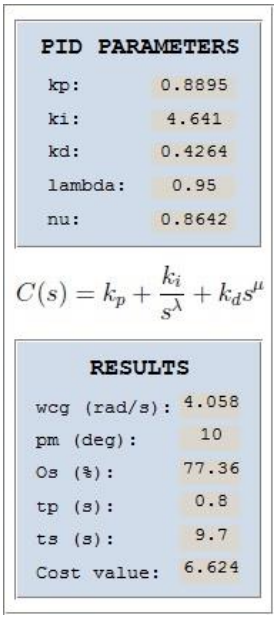

Figure 5. *Output Data* panel.

7. Observe the different graphs (Figure 6). The first one corresponds to temporal response of the system. Second and third ones are Bode's plots (magnitude and phase) obtained with the results.

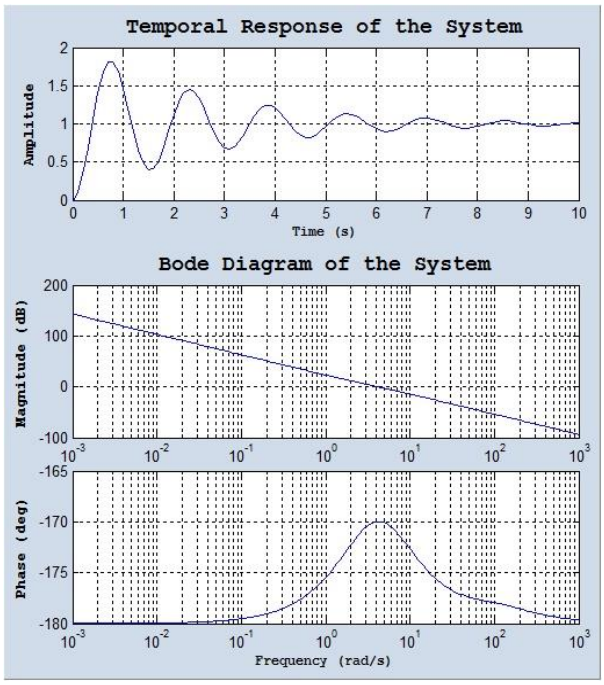

Figure 6. Graphs.

8. Save the results with the different buttons (Figure 7):
   a. *Save screenshot* button: it saves the image of the whole screen.
   b. *Save data(.mat)* button: it saves the inputs and outputs data in a *.mat* type file.

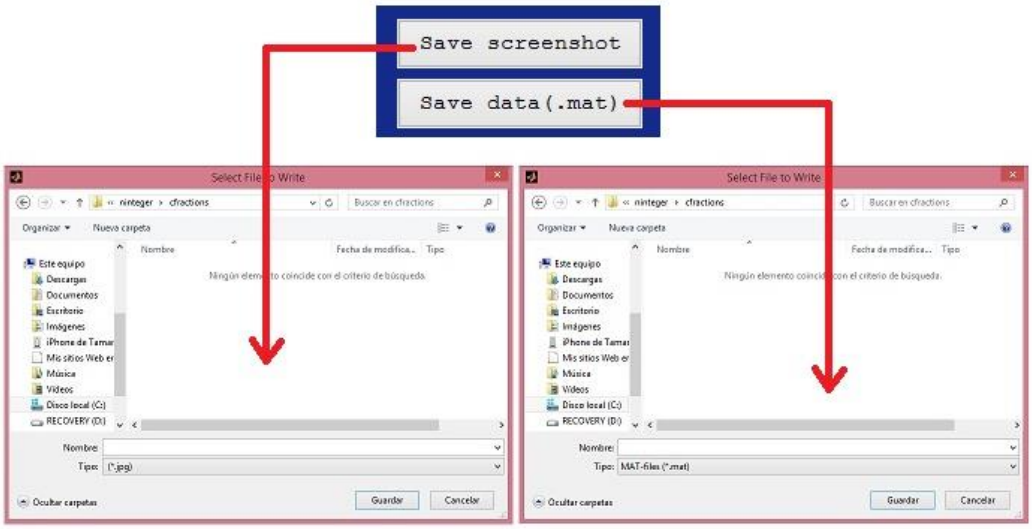

Figure 7. Save buttons.

Remember that every graph is saved automatically in a *.fig* file, as it is shown in Figure 8.

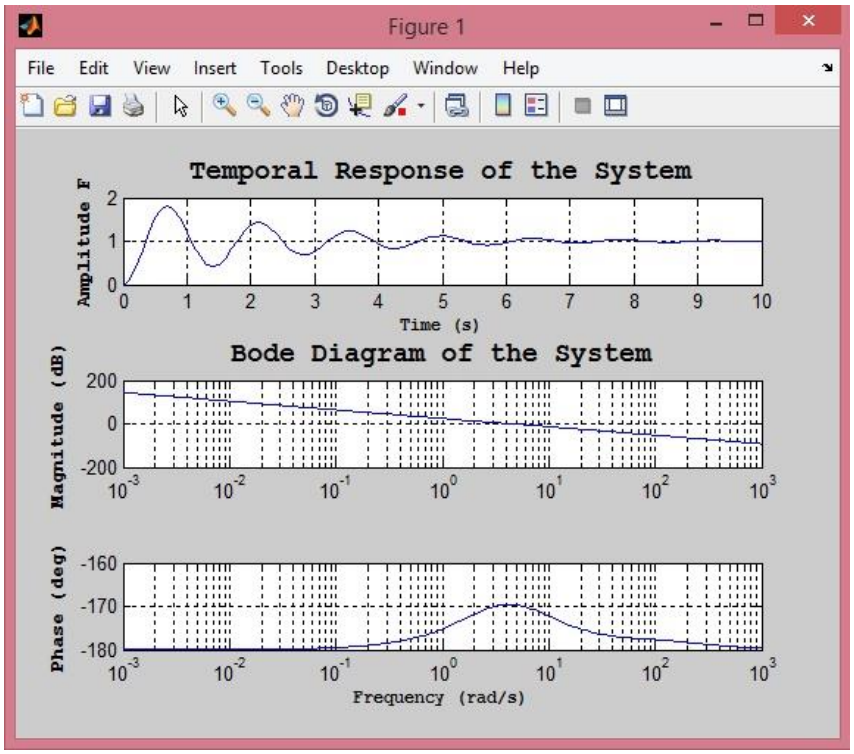

Figure 8. *Graphs.fig* file.

9. Start a new process with the ***New Data*** button (Figure 9) or close the program.

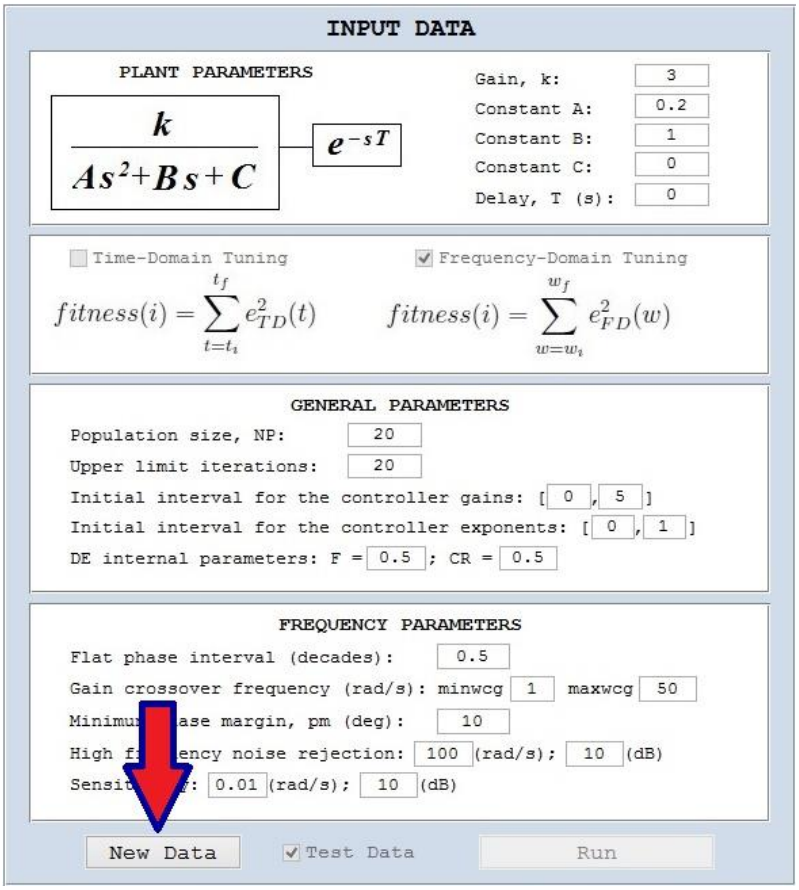

Figure 9. ***New Data*** **button.**