industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de una Interfaz Gráfica de Usuario para aplicaciones docentes

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: **Pablo Casas Mateo**
Director: José Luis Muñoz Lozano
Codirector: Juan Domingo González Teruel

Cartagena, junio de 2021

Universidad Politécnica de Cartagena

Agradecimientos

A mi director José Luís Muñoz Lozano y codirector Juan Domingo Rodríguez, por toda la ayuda y apoyo durante todo el camino de este trayecto, su preocupación e implicación por que saliese bien, y por hacer más sencilla toda esta experiencia durante el periodo COVID-19.

Y a mi familia y amigos, por todo el apoyo emocional tanto en los buenos momentos como en los malos, ya que sin ellos no podría haber llegado donde estoy ahora mismo.

INDICE

INDICE DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

La regulación automática, o teoría de control, es una rama de la ingeniería que trata con sistemas dinámicos, que cambian a lo largo del tiempo, como podría ser un tanque con un flujo de entrada y otro de salida, o la temperatura del motor de un automóvil.

Esta rama de la ingeniería estudia estos sistemas tratándolos como bloques, obteniendo una señal de entrada, modificándola según el bloque en cuestión y así obteniendo una salida. Este bloque representará todas aquellas acciones que van a modificar la señal de entrada, como podrían ser actuadores o reguladores. Estas acciones vendrán representadas mediante unas funciones matemáticas específicas para cada situación, y las llamamos **funciones de transferencia**. Una vez se realizan las modificaciones de la señal de entrada a través de estas funciones de transferencia, tendremos la señal de salida. Si estas señales de salida deben ajustarse a una referencia previa, habría que añadir a nuestro sistema un controlador que modifique el valor de la señal de entrada para obtener la respuesta deseada. [1]

Por otra parte, la programación es algo importante dentro del mundo de la ingeniería, y es necesario que todo buen ingeniero sepa programar o tenga una buena base para seguir aprendiendo. Entre todos los programas en el mercado, MATLAB es de los que más resaltan y encajan en nuestro proyecto por varias razones, entre las que destacan:

- Su gran adaptación al mundo de la ingeniería
- Su programación con comandos familiares y fáciles de recordar
- La variedad de herramientas y opciones que tiene, orientadas al entorno científico y de la ingeniería
- La posibilidad de crear aplicaciones de manera muy visual utilizando herramientas como GUIDE o APP DESIGNER.

Objetivos

En la propuesta de este proyecto, los objetivos principales de este trabajo eran:

1. Estudio de las características de las Interfaces Gráficas de Usuario (GUI)

2. Análisis de las herramientas que proporciona MATLAB para el desarrollo de una GUI y elección de la herramienta más adecuada para las características de este trabajo.

3. Definición y diseño de una GUI para tareas de enseñanza/aprendizaje en el ámbito del control de procesos.

4. Desarrollo y evaluación de la GUI.

Durante la realización de este proyecto, los objetivos que se han seguido han sido los siguientes:

Por un lado, va a servir como guía para aquellos estudiantes u otro tipo de personas que quieran realizar programas con la herramienta GUIDE de MATLAB y no la conozcan. Esta guía constará de una explicación detallada de cómo manejarse dentro de la interfaz de MATLAB y de cómo se utilizan cada una de las opciones dentro de la herramienta GUIDE.

Por otro lado, realizar un estudio de la asignatura de Regulación Automática y determinar qué aspectos de la misma pueden ser interesantes introducir en una GUI destinada a tareas de enseñanza/aprendizaje en el ámbito del control de sistemas.

Por último, crear una interfaz gráfica sencilla e intuitiva que permita realizar los cálculos básicos en torno a la asignatura y facilitar el código con una estructura sencilla y bien definida, con comentarios claros y concisos que permitan a cualquier alumno o profesor que no tengan conocimientos avanzados de GUIDE seguirlo y entenderlo.

Estructura de la memoria

Esta memoria se ha dividido en 7 capítulos adicionales a esta introducción y un anexo:

En el segundo capítulo se introduce el concepto de GUI, se explica qué es y para qué sirve y se realiza una comparativa entre las dos herramientas de creación de aplicaciones que contiene MATLAB, se detallan los aspectos positivos y negativos de cada una de ellas y se concluye con la elección de GUIDE como herramienta de desarrollo.

En el tercer capítulo se ha recopilado toda la información necesaria para conocer la herramienta GUIDE y cómo funciona.

En el cuarto capítulo se muestra para qué sirve cada uno de los objetos existentes en la herramienta GUIDE y se hace una explicación detallada de cómo se usan.

En el quinto capítulo se ha analizado la asignatura de Regulación Automática, se ha hecho una síntesis de sus contenidos y se han buscado ideas que podrían ser útiles a la hora de implementar funciones en nuestro programa, teniendo en cuenta el objetivo de que la herramienta debe ser útil a la hora de ser usada en tareas de enseñanza y aprendizaje.

En el sexto capítulo entramos en la segunda parte del proyecto, en la creación de nuestra GUI, donde se explica qué tiene la GUI que hemos creado en cuanto a funciones y cómo se ha diseñado la aplicación.

En el séptimo capítulo, una vez finalizado nuestro programa, se muestran las pruebas para constatar que nuestra GUI funciona de manera correcta y no existe ningún tipo de fallo.

En el octavo capítulo se exponen las conclusiones a las que se ha llegado al finalizar el proyecto.

Por último, en el anexo se adjunta el código que hemos creado para realizar nuestro programa.

CAPÍTULO 2: GUIs Y DIFERENCIAS ENTRE GUIDE Y APP DESIGNER

¿Qué es una GUI?

GUI es una abreviación para *Graphic User Interface,* o interfaz gráfica de usuario. Pero bien, ¿qué es esto? Como su propio nombre indica, es un entorno interactivo gráfico donde podemos ejecutar líneas de código con un apoyo visual y hacer que nuestros códigos sean mucho más llamativos e interesantes.

Las GUIs son programas informáticos que actúan como interfaz de usuario, usando objetos e imágenes que veremos más adelante como herramientas de interacción, los cuales programaremos para que interactúen unos con otros. [2]

¿Para qué sirven las GUIs?

El objetivo principal de las GUIs es hacer más sencilla la comunicación entre una máquina/sistema operativo y un usuario. Si no existiesen, solo personas que se hubiesen especializado en el área de la informática serían capaces de utilizar un ordenador, por ejemplo, pero gracias a estas interfaces se puede reducir de manera considerable la complejidad del código por acciones predeterminadas asignadas a elementos visuales sencillos de comprender.

Un ejemplo podría ser el escritorio de tu ordenador. En el escritorio encontramos carpetas o accesos directos a programas (estos dos actuarían como objetos de la GUI), y cuando hacemos clic sobre ellos se ejecuta una parte del código que bien nos abre la carpeta o programa, nos permite cambiarle el nombre al acceso directo, o nos permite desplazarlo; esto dependerá de cómo se haya programado el objeto.

Características de las GUIs

Al ser programas informáticos, cada GUI tendrá unas características completamente distintas, pero bien, todas estas GUIs tendrán una serie de factores a tener en cuenta a la hora de crearlas.

Una buena GUI se caracteriza por: [3]

- Tiene que ser fácil de usar
- La curva de aprendizaje es acelerada y es fácil recordar su funcionamiento
- Disponen de elementos fácilmente reconocibles
- Deben ser capaces de facilitar y predecir las acciones más comunes del usuario
- Presentan un orden: la información que encontramos en la GUI está bien ordenada en menús, iconos, listas…
- Las operaciones tienen que ser rápidas, intuitivas y reversibles

Como desarrollar una GUI

Existen muchas herramientas que nos permiten crear GUIs, como GUIDE y APP DESIGNER, herramientas de las cuales se hablará a continuación, se hará una comparativa y se elegirá una.

La herramienta elegida, de la cual hablaremos en el próximo capítulo, nos brinda la oportunidad de crear una interfaz a nuestro gusto, y luego nos proporciona el código asociado a los elementos que hayamos situado en la GUI, código que deberemos modificar y ampliar para relacionar los elementos entre ellos. [3]

Ahora, se va a realizar una comparativa entre las 2 herramientas que tiene MATLAB para la creación de GUIs, GUIDE es la herramienta más antigua y APP DESIGNER es la opción más moderna. Se comentarán las diferencias que hay entre las herramientas en varios campos dentro de los programas y se finalizará con una conclusión de qué herramienta utilizaremos.

Extensión de los archivos

Al crear una aplicación, para que no se pierda, debemos tenerla guardada. Aquí encontramos la primera diferencia entre ambos métodos de diseño de aplicaciones, ya que, cuando guardas un programa en MATLAB GUIDE, encontramos que se crean dos archivos distintos, una figura con extensión *.fig* y luego por otra parte el código con extensión *.m*. Por el contrario, cuando realizamos el mismo proceso con MATLAB APP DESIGNER, nos encontramos que a la hora de salvar nuestra aplicación, se nos crea un solo archivo con extensión *.mlapp*. La ventaja sobre APP DESIGNER es que lo tenemos todo en un solo archivo, y a la hora de enviarlo o trasladarlo será más cómodo tener en cuenta un solo archivo que dos, pero esta ventaja no es demasiado significante. [4]

Acceso

La diferencia en el acceso a una herramienta u otra es mínima, ya que para acceder a GUIDE lo único que tenemos que hacer es escribir *GUIDE* en la ventana comandos y ya encontramos el atajo para abrir o crear archivos. Por otro lado, para entrar al entorno de trabajo con APP DESIGNER lo que debemos hacer es abrir el desplegable *NEW* en la barra de herramientas en la parte superior y hacer clic en el apartado *APP*. Un punto a favor sobre el entorno de APP DESIGNER es que, al abrir el entorno, este es un apartado independiente.

Aquí me gustaría comentar, más que el acceso pues su diferencia es mínima, es el acceso para tener o no una u otra opción. La herramienta GUIDE es la que puede ser utilizada en más dispositivos, ya que, al ser la versión antigua, es más fácil de encontrar. Esto significaría que no haría falta por parte del alumno, en este caso yo, que se desplazase desde su domicilio particular hasta las instalaciones de la universidad para poder emplear el Software MATLAB en sus versiones más nuevas. **Esta será la principal razón por la que trabajaremos con GUIDE.**

Inspector de propiedades y explorador de objetos

Aquí hablaremos sobre el inspector de propiedades y explorador de objetos, que son dos herramientas de trabajo que nos permiten ver, como su propio nombre indica, las propiedades de cada uno de los objetos que situemos en nuestra GUI.

Para entrar a estas opciones en GUIDE, tenemos dos opciones: presionar en un botón de arriba para entrar en el explorador de objetos, o bien hacer clic en cada uno de los componentes y seleccionar inspector de propiedades. En APP DESIGNER, nada más abrir el entorno de trabajo, nos encontramos a la derecha unos espacios donde se encuentran ya estas herramientas.

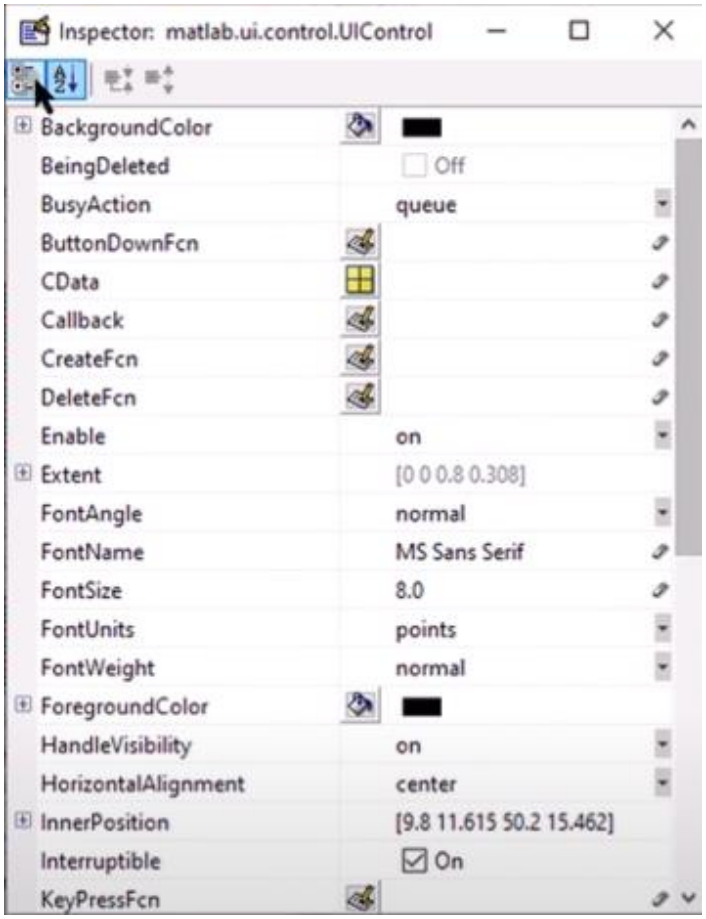

Figura 1: Inspector de propiedades en GUIDE

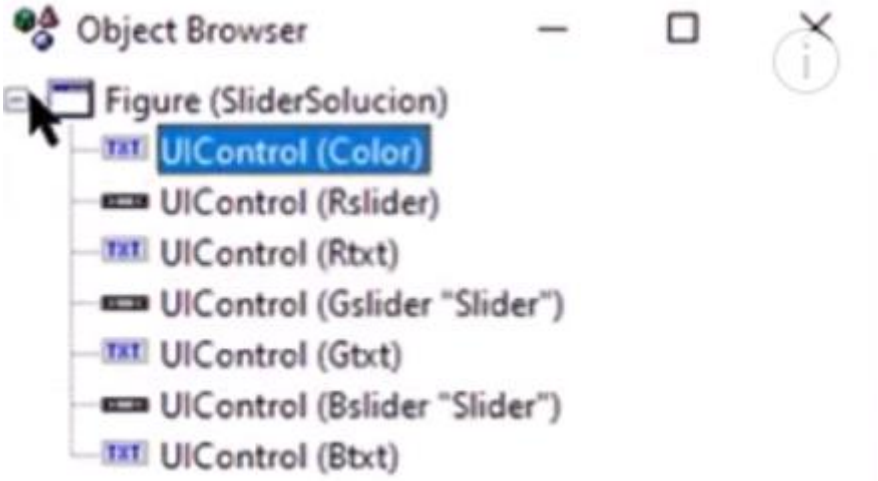

Figura 2: Explorador de objetos en GUIDE

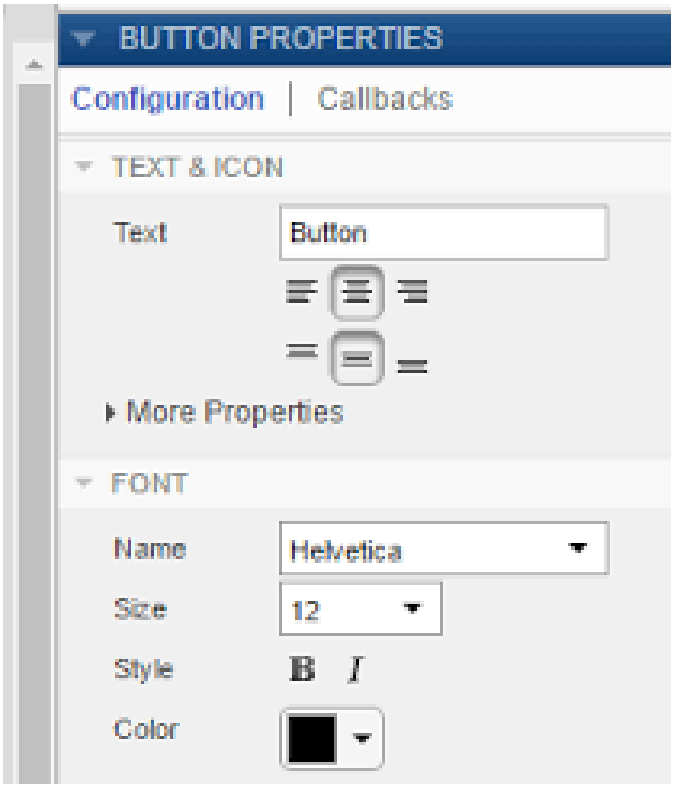

Figura 3: Inspector de propiedades en APP DESIGNER

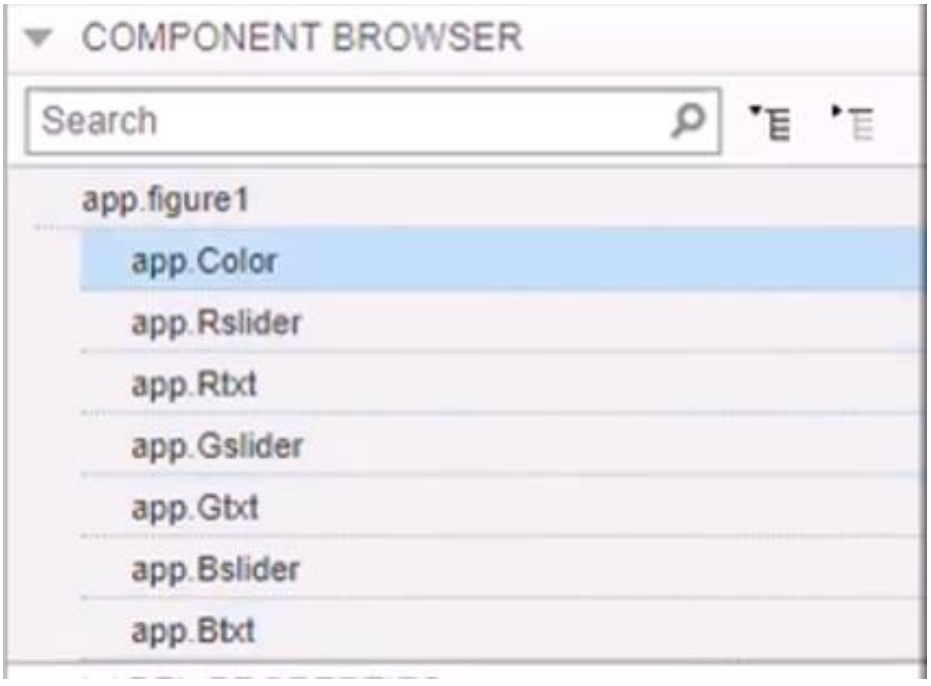

Figura 4: Explorador de objetos en APP DESIGNER

En las figuras anteriores se observan el inspector de propiedades y el explorador de objetos de ambas herramientas. Si hablamos del inspector de propiedades, vemos que ambas herramientas nos muestran unas opciones sencillas de entender y muy visuales, aunque en GUIDE aparece una lista sin estética y en APP DESIGNER se nos muestra todo más ordenado. Lo mismo podemos decir del explorador de objetos, que nos recoge cada uno de los componentes que tenemos en nuestra GUI. En definitiva, podemos ver cómo el uso de estas herramientas en APP DESIGNER está más ordenado y es más fácil de entender que en GUIDE, aunque no hay mucha diferencia y no será un factor relevante para la decisión final.

Estructura de código

A la hora de abrir el código en GUIDE, los archivos *.m* tienen una estructura similar a la de cualquier código, por lo que encontraremos una serie de comandos que pueden ser algo liosos de entender si no añadimos anotaciones y comentarios para ir facilitando su interpretación. Por la parte de APP DESIGNER, nos encontramos que el código ya viene mucho mejor estructurado. Para empezar, en APP DESIGNER encontramos partes del código que no podremos modificar, pues son rutinas que la interfaz hace para crear los componentes, por lo que es una gran cantidad de líneas de código que prácticamente podemos dejar a un lado. Después, cada vez que introduzcamos un objeto nuevo en la interfaz se crea un bloque específico, y a la parte izquierda de la pantalla encontraremos un buscador de código, donde podremos buscar los distintas funciones y variables globales.

Componentes clásicos

En cuanto a componentes clásicos, podemos mencionar que hay varios de ellos que se comparten entre GUIDE y APP DESIGNER, como pueden ser los componentes *AXES* y *SLIDER,* y el componente *PUSH BUTTON*, aunque este en APP DESIGNER se llama *BUTTON*, por lo que apenas hay diferencias entre ambos escenarios.

Luego, el componente *EDIT TEXT* que nos encontramos en GUIDE se desglosa en *EDIT FIELD (NUMERIC),* solo para números; *EDIT FIELD (TEXT),* solo para texto; y *TEXT AREA,* que sirve para múltiples filas de datos. En cuanto al texto estático, el único cambio que encontramos es que ahora se llama *LABEL.*

En cuanto a los demás componentes, son prácticamente iguales, con el único posible cambio en el nombre o la apariencia, o se les ha añadido alguna función extra, como, por ejemplo, el componente tabla que tiene más funciones. Lo que conocíamos en GUIDE como *BUTTON GROUP* que agrupaba todos los botones, en APP DESIGNER se separa en *RADIO BUTTON GROUP* para los botones radiales, y *TOGGLE BUTTON GROUP* para los demás.

Componentes nuevos

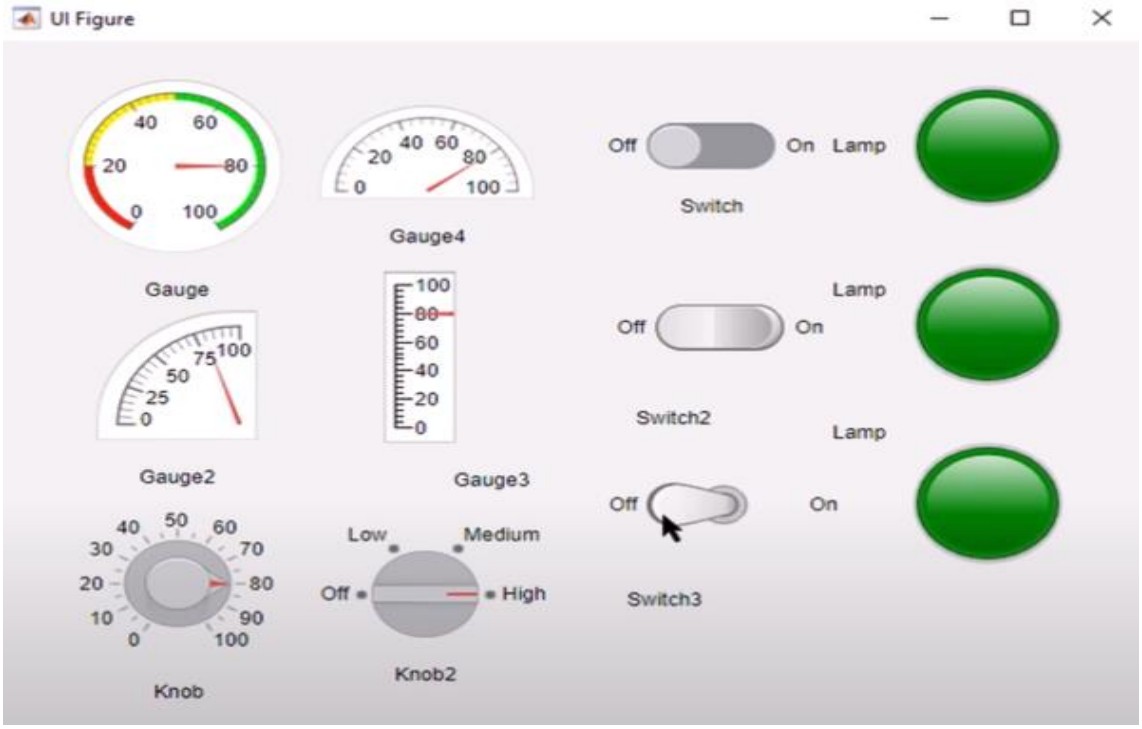

Figura 5: Instrumentación añadida en APP DESIGNER

En la figura 5 se pueden observar algunas de los instrumentos añadidos en APP DESIGNER que no encontramos, como unos calibres, tanto redondo, semicircular y un cuarto de circunferencia; otros elementos como unos botones más estéticos: unas lámparas que nos podrán servir para indicar la situación en la que nos encontremos, o unos mandos, discreto y continuo, para poder elegir el nivel de un cierto parámetro.

Instrumentación

Entorno de diseño	GUÍA [4]	Diseñador de aplicaciones
Calibre		✓
Calibre de 90 grados		✓
Calibre lineal		✓
Calibre semicircular		✓
Mando		✓
Perilla discreta		✓
Lámpara		✓
Cambiar		✓
Interruptor basculante		✓
Interruptor de palanca		✓
Indicador de velocidad aérea [3]		✓
Altímetro [3]		✓
Indicador de velocidad de ascenso [3]		✓
Indicador EGT [3]		✓
Indicador de rumbo [3]		✓
Horizonte artificial [3]		✓
Indicador de RPM [3]		✓
Indicador de giro [3]		✓

Figura 6: instrumentación APP DESIGNER

Por otro lado, en la figura 6 se puede ver la lista completa de herramientas que fueron añadidas a APP DESIGNER y que no podemos usar en GUIDE. En esta lista encontramos algunos elementos que no hemos visto en la figura anterior, y son herramientas que se usarían en el estudio o en la aplicación de un programa aeroespacial, como pueden ser indicadores de rumbo, RPM, giro o EGT, o un horizonte artificial y un altímetro. [5]

Migración de aplicaciones desde GUIDE a APP DESIGNER

Después de ver esta comparación entre GUIDE y APP DESIGNER, podemos sacar como conclusión que APP DESIGNER es una extensión más fácil de usar, más cómoda y con más futuro, ya que GUIDE tiene fecha de caducidad y acabará por extinguirse. Aun así, la idea de este proyecto es realizarlo mediante el uso de GUIDE, ya que, como se ha mencionado, el alumno no tiene a su disposición la herramienta APP DESIGNER para poder elaborarlo. Esto no debe suponer ningún problema, ya que la herramienta APP DESIGNER cuenta con un apartado denominado *MIGRACIÓN,* que nos serviría para poder trasladar el programa desde GUIDE a APP DESIGNER en un futuro si lo considerásemos oportuno. Este paso de migración es bastante sencillo, pues solo hay que buscar en GUIDE el botón ‘’migrar a APP DESIGNER’’ y el programa se encargará de adaptar el código y el escenario de trabajo a la nueva herramienta. [6]

Conclusión

En esta comparativa, hemos visto tanto aspectos positivos y negativos de ambas herramientas, y aunque se pueda ver que la herramienta APP DESIGNER, la herramienta más actual y nueva del MATLAB tiene más puntos positivos, más opciones añadidas y más aspectos para ser la elegida, para la realización de nuestro trabajo utilizaremos la herramienta GUIDE, como ya se ha comentado, por la disponibilidad del alumno que realiza el trabajo, su incompatibilidad para desplazarse a las instalaciones de la UPCT para utilizar el software proporcionado por la misma, y con la facilidad que nos presenta APP DESIGNER para migrar aplicaciones desde GUIDE si en un futuro nos resultase interesante.

CAPÍTULO 3: GUIDE

GUIDE

GUIDE es una herramienta de programación disponible en MATLAB que nos posibilita desarrollar GUIs.

GUIDE, después de haber creado toda la GUI a nuestro gusto, y de una manera que interpretemos que será fácil de usar para los usuarios últimos, genera de manera automática el código de programación en MATLAB que corresponde a esta interfaz. Este código podrá ser editado para cambiar el comportamiento de la aplicación que estemos creando. Esta edición nos sirve para poder relacionar los componentes que hayamos situado en la GUI y que la aplicación funcione correctamente.

Ejemplo: GUI suma de 2 números

Un ejemplo sencillo sería realizar un programa que hiciese una suma.

Primero, abriremos la herramienta GUIDE y crearemos un entorno de trabajo; una GUI vacía.

El siguiente paso sería ir introduciendo los elementos que queremos que salgan en nuestro programa, por ejemplo:

- cuadros de texto editable, donde escribiríamos en el programa finalizado los números que queremos sumar
- también podemos añadir un par de cuadros de texto fijo, que nos diga cuales son los nombres de las variables introducidas, por ejemplo, X e Y
- introduciremos el recuadro de texto fijo que supondrá el resultado de la suma
- un botón que nos permita dar paso a esta operación

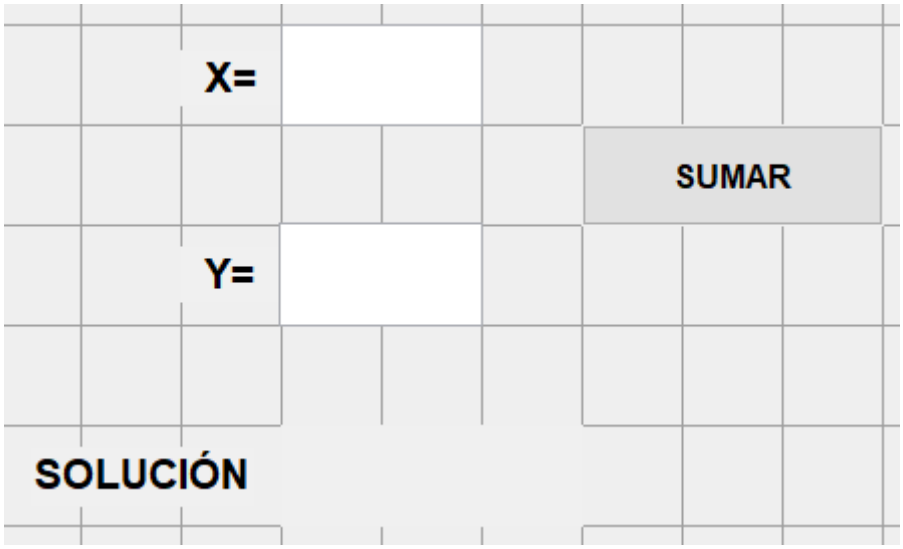

Figura 7: Ejemplo de GUI

En esta figura número 7 vemos la primera disposición de los objetos que tendrá la interfaz final de la calculadora. Lo importante es situar los objetos necesarios para luego poder programarlos.

Objetos

Los objetos son la base de nuestra GUI, pues sin ellos no tendríamos nada. Son los que se encargarán de transformar nuestros clics en la pantalla, mediante el código, en las acciones que esperamos de ellos. Por ejemplo, el más sencillo, el pushbutton hará que cuando hagamos clic sobre él se ejecuten una serie de líneas de código que acabarán por interactuar con otros objetos y darnos un resultado por pantalla.

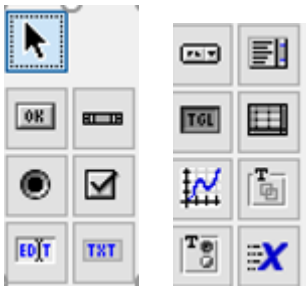

Figura 8: Objetos

En la figura anterior se observan los objetos disponibles para crear la interfaz, que son los siguientes:

- Pushbutton: se encuentra un recuadro donde leemos OK. Este objeto es, simplemente, un botón.

- Slider: se encuentra a la derecha del Pushbutton, y es una barra que se puede deslizar. Esto nos servirá para poner una escala de valores, con un máximo y un mínimo, que por ejemplo puede significar la frecuencia de una onda, y ver como varía esta onda en una gráfica.

- Radiobutton, Checkbox y ToggleButton: Los iconos de estos objetos son: un círculo con un punto, un cuadrado con una señal de verificación y un rectángulo con un escrito que pone TGL. Estos elementos pueden tener dos valores, 0 o 1. Cuando nos encontramos que no están activos, que no presentan la marca como que han sido presionados, representan un 0; cuando sí presentan dicha marca, actúan como un 1. Su funcionalidad es que tienen 2 estados, y que pueden realizar una función u otra según su estado. RadioButton y ToggleButton funcionan exactamente igual y la única diferencia entre ellos es la apariencia; sin embargo, mientras que podemos tener varios Checkbox activos a la vez, con RadioButton o ToggleTutton solo podemos tener uno. A continuación, se muestran en tres figuras distintas los tres objetos mencionados:

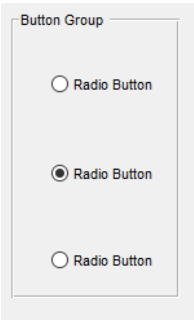

Figura 9: RadioButton

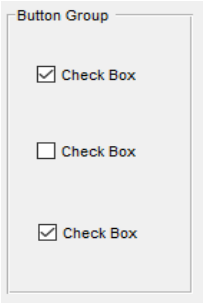

Figura 10: CheckBox

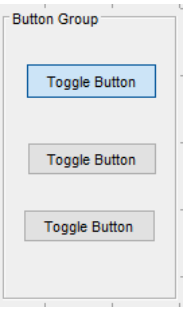

Figura 11: ToggleButton

- Edit Text: Lo encontramos como un cuadro donde leemos EDIT. Este objeto sirve como una casilla editable de nuestra aplicación. Por ejemplo, en la aplicación de sumar dos números, los recuadros donde escribimos los números son casillas de EDIT TEXT. Estas casillas se pueden programar para convertir texto en números y viceversa.

- Static Text: Lo encontramos a la derecha de EDIT text con un escrito que pone TXT. Esta casilla no será modificable por nosotros directamente, sino que cambiará su contenido en función del programa que hayamos creado, pero no es un objeto al que podamos acceder desde la aplicación y modificarlo.

- Pop-up menu y ListBox: Al igual que Radio, Check y Toggle Button, son dos objetos que actúan de manera muy similar. Los encontramos debajo de los botones Edit y Static text. Ambos objetos sirven como una lista de propiedades que podremos seleccionar para que cambie algo en nuestra aplicación. La principal diferencia entre ambos es que el Pop-up menu es una lista que podremos desplegar y esconder, mostrándonos inicialmente solo el primer elemento de la lista, lo que nos significará poco espacio en nuestra aplicación, y la ListBox es un cuadro donde nos aparecen todos los elementos, que si lo hacemos suficientemente grande nos mostrará toda la lista al completo, y si no es suficientemente grande, podremos recorrerla con una barra lateral.

- Tabla: Lo encontramos con un icono de una tabla a la derecha de botón Toggle. Es, esencialmente, una tabla, pero tiene muchas características, pues no solo podemos almacenar datos numéricos, podemos crear una tabla de casillas como las checkbox, también podemos hacer una tabla con contenido en forma de texto… También podemos añadir un menú desplegable a cualquiera de las columnas, pudiendo manejarse de la manera que queramos. Esta herramienta es muy grande, y se pueden hacer muchas cosas con ella, pero la investigaremos más a fondo conforme vayamos usándola.

- Axes: El objeto Axes lo encontramos en el botón con una gráfica dibujada. Como se trata de una gráfica, nos servirá para mostrar de manera visual una serie de parámetros o datos que hayamos recopilado. Hay opciones como tener varias gráficas, poder seleccionar cual queremos que nos dibujen, si queremos tener una estática y otra cambiando… Esta herramienta es muy amplia, al igual que la tabla, y la ampliaremos conforme vayamos dándole uso.

- Panel y Button Group: Estos componentes se ven iguales, hacen prácticamente lo mismo, y sirven básicamente para que nuestra aplicación quede más bonita estéticamente. Los encontramos como un cuadro, con una T en la parte superior, y teniendo en su interior dos cuadrados por parte del panel, y dos círculos por parte del Button Group. La principal diferencia es que en el Button Group solo podemos tener activa una de las opciones que hayamos añadido, mientras que en el panel puedes tener activas varias opciones a la vez, además de añadir pop-up menu o cuadros de texto editable.

- ActiveX Control: Esta herramienta la encontraremos abajo a la derecha en nuestro panel, con un icono de una X mayúscula. ActiveX es una herramienta que nos permitirá, entre muchas otras cosas, agregar una secuencia de video a nuestros programas. Se puede dar, como ejemplo, la programación de la curva de un péndulo. Podemos obtener las ecuaciones y todos los parámetros de este péndulo, pero algo muy interesante sería ver como se mueve este, y con la herramienta ActiveX se podría programar. Más adelante, si llegamos a usar esta herramienta, se profundizará más.

Inspector de propiedades

Cuando comenzamos a crear una GUI, situamos sobre el panel de trabajo objetos, como botones, pulsadores, botones correderos, textos editables y fijos, o un cuadro donde podría ir una gráfica. Todos estos elementos tendrán unas características estándar, un tamaño de letra, unos colores, un nombre con el que los encontraremos en el código… pero todos estos parámetros son editables, y los podemos encontrar en el inspector de propiedades. Hay varias formas de entrar al inspector de propiedades: presionando el botón derecho del ratón y seleccionando "Property Inspector" o "Inspector de propiedades"; haciendo doble clic sobre el elemento que queramos modificar; o bien seleccionando el icono de inspector de propiedades en la barra de herramientas. [7]

En este inspector encontramos muchos parámetros, y a continuación veremos algunos de los más utilizados:

- BackgroundColor: Elección del color del fondo.
- Enable: Podemos activar, desactivar o deshabilitar el objeto.
- FontAngle: Elección entre normal y cursiva.
- FontName: Elección de la fuente.
- FontSize: Elección del tamaño de la fuente.
- FontWeight: Elección entre normal y negrita.
- ForegroundColor: Elección del color del texto.
- String: En esta opción es donde escribimos qué texto llevará el objeto.
- Tag: En esta opción podremos cambiar el nombre que el objeto tendrá en el código (recomendable usar el mismo nombre que en STRING)
- Tooltip: En esta opción podremos escribir un texto que aparecerá cuando situemos nuestro cursor encima del objeto, sin llegar a usarlo.

Como se ha dicho, estos son los parámetros que suelen utilizarse más, y ser los que se modifican en prácticamente cada uno de los objetos, pero hay más, como podemos ver a continuación en la figura 12:

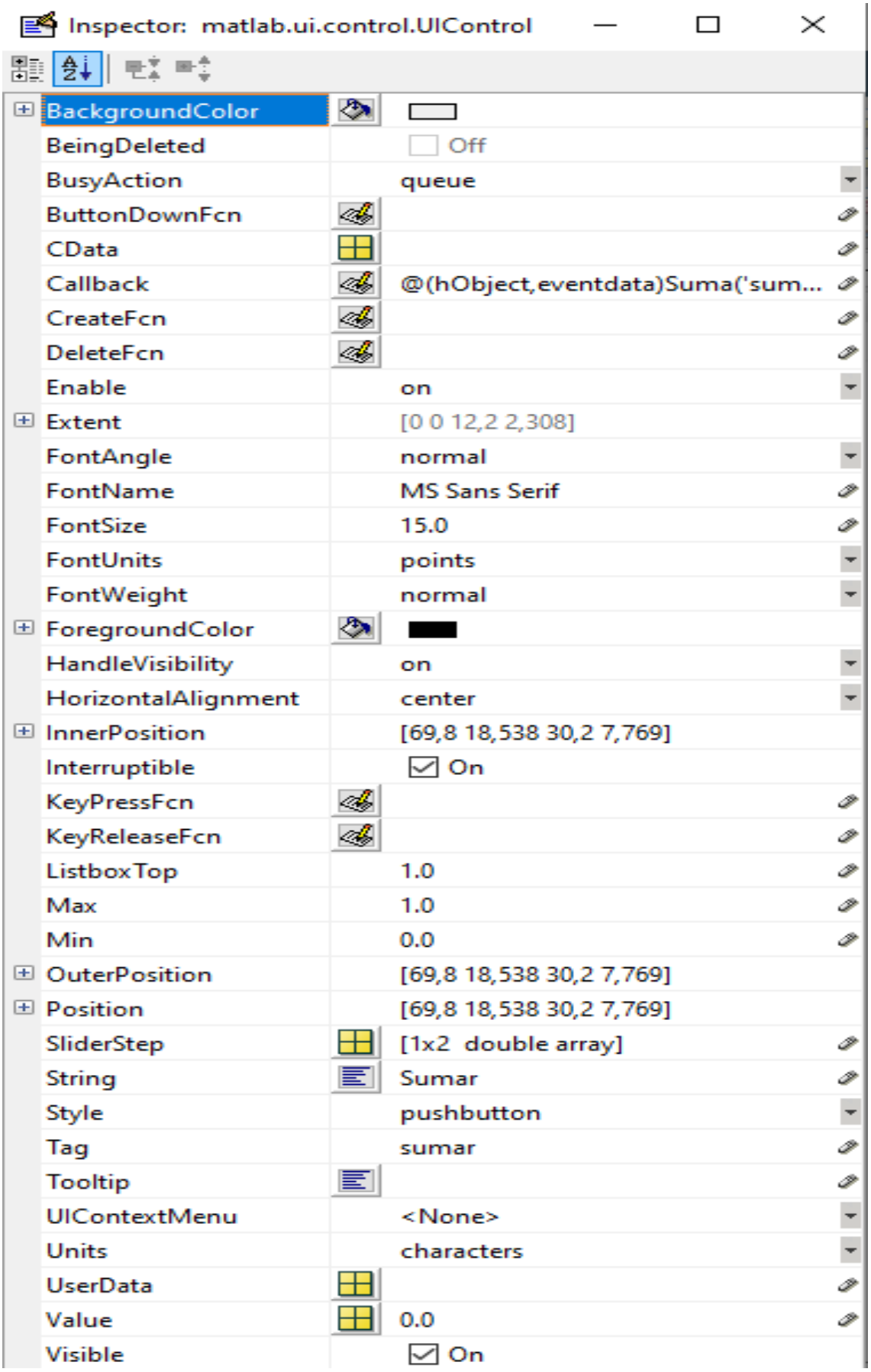

Figura 12: Inspector de propiedades

Funciones y callback

Una callback es una función ‘’A’’ que se usa como argumento de una función ‘’B’’. Cuando se llama o ejecuta la función ‘’B’’, esta ejecuta ‘’A’’.

Cada objeto de los que hemos visto anteriormente tiene una serie de funciones asociadas, y no todos los objetos tienen el mismo número de funciones. A cada función hay que escribirle el código de la acción que queremos que haga.

La mayoría de estos objetos tienen asociadas las siguientes 5 funciones:

- CreateFcn: se ejecuta durante la creación de un objeto una vez que MATLAB define todas las propiedades.

- Callback: es la función principal. Ejecuta las líneas de código asociadas al objeto. En el botón ‘’Sumar’’ del ejemplo, realiza la operación de suma de los dos sumandos.

- ButtonDownFcn: esta función actúa igual que la callback, pero solo ejecuta su código cuando el objeto o componente está inactivo o inhabilitado en su propiedad enable (la propiedad enable es el parámetro que indica si el componente funciona o no de manera normal). Una utilidad que puede tener esta función es tener, en un mismo objeto, 2 funciones distintas, aunque es más claro y sencillo crear 2 objetos distintos.

- DeleteFcn: esta función se ejecuta, bien cuando se cierra la GUI, o bien cuando se elimina el componente en cuestión mediante programación de la función delete.

- KeyPressFcn: esta es una de las funciones más interesantes, pues consiste en ejecutar la línea de código o la función del objeto que estemos utilizando sin tener que hacer clic sobre este, y asignarle una tecla del teclado. Por ejemplo, accionar un pushbutton cada vez que hagamos clic en la tecla “ç” del teclado.

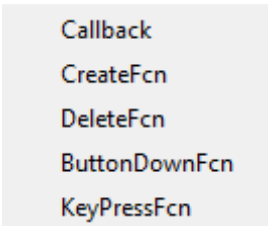

Figura 13: Funciones principales

En la figura número 13 encontramos una vista de cómo vemos las funciones asociadas a los objetos dentro del programa. Estas funciones las podremos ver si presionamos clic derecho sobre el objeto y desplegamos el menú callbacks.

Luego nos encontramos con unos objetos que son distintos en cuanto a las funciones, pues el texto estático solo usa 3 de estas, y la Tabla tiene, además de las 5 mencionadas, 2 adicionales. [8] [9]

Ejemplo: Código y programación

Ahora que conocemos un poco mejor la herramienta GUIDE podemos entrar más a fondo en el ejemplo que hemos visto anteriormente, el de la suma de dos números.

Como se puede observar en la Figura 7 tenemos la interfaz de nuestra GUI con un par de textos editables, 4 cajas de texto fijo y un PushButton. Con respecto a las cajas de texto fijo debemos explicar que hay 2 tipos distintos:

- 3 cajas de texto fijo que ya contienen texto (X=, Y= y Suma=): estos cuadros de texto fijo no hará falta modificarlos en el código posteriormente, pues su función es simplemente indicativa, para mostrarnos qué elementos tenemos a la derecha en los textos editables. Este cambio de nombre se puede realizar desde el inspector de propiedades, cambiando el parámetro ''String''.

- 1 caja de texto fijo sin texto: este cuadro no contiene información escrita de primeras, y nos mostrará al final el resultado de la suma. Esto hay que programarlo posteriormente en el cuadro de código.

En todos estos elementos es bueno cambiarles el nombre, pues por defecto vienen con nombres que simplemente los enumeran, como pushbutton_1 o EditText_2. Para ello, pulsamos el objeto que queramos, por ejemplo, el pushbutton, abrimos el inspector de propiedades y vamos al apartado que pone ''Tag'' (como se ha comentado, es recomendable que Tag y String coincidan, para que no haya opción a error). Una vez que cambiamos el Tag, este será el nuevo nombre con el cual modificaremos el código. En la siguiente figura, se muestra el inspector de propiedades con el nombre del objeto cambiado:

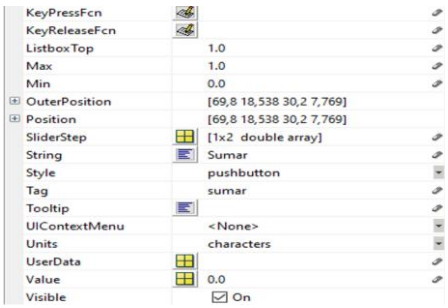

Figura 14: Cambio de nombre de un objeto

Una vez hemos cambiado estos parámetros en la GUI, es hora de ir al código para comenzar a programar y dar sentido a nuestro programa.

Como esta interfaz es muy sencilla, lo único que deberíamos hacer sería que cuando presionemos el botón ''Sumar'' se sumen los números que hayamos escrito en los cuadros de texto editable.

Cuando entramos al código, vemos que MATLAB ha creado 2 funciones por cada cuadro de texto editable: CreateFCN y Callback. La callback "X" nos servirá para obtener el dato que escribamos en el recuadro X, y lo mismo para Y.

```
function x_Callback(hObject, eventdata, handles)
% hObject    handle to x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of x as text
%        str2double(get(hObject,'String')) returns contents of x as a double


% --- Executes during object creation, after setting all properties.
function x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroun
    set(hObject,'BackgroundColor','white');
end
```

Figura 15: Callback y CreateFCN para variable X

En la figura 15 se observan las callbacks del objeto EditText_1 que se le ha cambiado el nombre a X. La primera línea que obtenemos de cada función es la declaración del tipo de función que vamos a usar (para el caso Callback, es que vamos a usar esta función, y después programaremos que queremos que haga).

Los términos entre paréntesis no se deben modificar, pues son partes de la función que nos servirán para poder declararla u obtener información de esta en otras funciones, pero explicaremos que son:

- hObject: es un identificador que solo contiene información de dicha función. Por ejemplo, en el caso de la callback para X, su hObject es un identificador que contiene la información en X.
- Handles: es también un identificador, pero este nos sirve para todo el código. Como veremos adelante, cuando tengamos que utilizar la información que tenemos en X para realizar la suma en la callback del pushbutton “Sumar”, identificaremos este valor mediante handles.
- Evendata: en nuestro caso no usaremos esta parte de la función, ya que, con nuestra versión, no está totalmente definido.

Una vez sabemos esto, lo que nos interesa a nosotros es decirle a la función ‘’Sumar’’ lo que debe hacer, que es coger la información que hayamos escrito en el recuadro “X”, en el recuadro “Y”, sumar ambos números y mostrarnos el resultado en el recuadro que pone “Suma”. Para ello, iremos a la parte del código donde encontraremos la Callback para el pushbutton ‘’sumar’’:

```
% --- Executes on button press in sumar.
function sumar_Callback(hObject, eventdata, handles)
x1=str2double(get(handles.x,'string'));

%Con el comando ''get'' cogemos la string que hayamos escrito en el cuadro
%x, y con el comando str2double lo que hacemos es transformar este string
%en un número.

y1=str2double(get(handles.y,'string'));

sumaxy=x1+y1;

set(handles.suma,'String',sumaxy);
%Con el comando ''set'' lo que hacemos es asignar, en el recuadro
%''resultado'', el valor que obtenemos en ''suma'', y le damos un formato
%string.
```

Figura 16: Función suma

La figura 16 nos muestra la sección del código que realiza la operación de sumar, y a continuación se va a detallar qué significa cada línea de este código:

- sumar_Callback(hObject, evendata, handles);: hace referencia al pushbutton “Sumar”, y su callback, es decir, cuando presionemos el botón, todo lo que contenga esta función se ejecutará.

- x1=str2double(get(handles.x, ’String’));: comando que utilizamos para guardar, en una variable que hemos llamado x1, el contenido que tenemos en la callback X que hemos visto anteriormente. Esta información, aunque sean números, GUIDE la interpreta como una cadena de texto, y mediante el prefijo str2double convertimos esta cadena de texto en el valor numérico que tiene y que GUIDE entiende. Como se puede observar, hemos usado el identificador handles.x, que nos dice que la información que buscamos está en el callback X, que no forma parte de la función “sumar”. Lo mismo para la variable Y.

- sumaxy=x1+y1;: se suman los valores que acabamos de obtener y se guardan en una variable denominada ‘’sumaxy’’.

- set(handles.suma, ‘String’, sumaxy);: con el prefijo ‘’set’’, lo que hacemos es asignar el valor en nuestra variable sumaxy a la variable externa ‘’suma’’, que corresponde al cuadro de texto fijo denominado de la misma manera (cuadro de texto que, si recordamos, era el único cuadro de texto fijo que no poseía ninguna información escrita en él).

CAPÍTULO 4: CÓMO SE CREA UNA GUI

Este capítulo se centrará en la explicación detallada y paso a paso de cómo se diseña una GUI. Ya vimos cuáles son los elementos disponibles, por lo que ahora empezaremos a usarlos y ver cómo se utilizan y programan cada uno de ellos.

Lo primero que haremos al abrir el programa MATLAB será decirle que queremos empezar a trabajar con una GUI, introduciendo la palabra *GUIDE* en la ventana de comandos. A continuación, se nos mostrará una pantalla donde nos da a elegir si crear una GUI nueva o si abrimos una ya existente. Seleccionaremos la opción de crear nueva GUI, y veremos un cuadro con varios tipos de GUIs predeterminadas, una con unos botones ya introducidos, otra con una gráfica y otra con un cuadro de diálogo, pero nosotros lo que haremos será empezar una GUI vacía, desde cero, por lo que seleccionaremos la opción *Blank GUI (Default).* Tras esto, ya tendremos disponible nuestra base de la GUI preparada para ser personalizada. En la siguiente figura se muestra la ventana inicial con la que nos encontramos a la hora de empezar a crear una GUI.

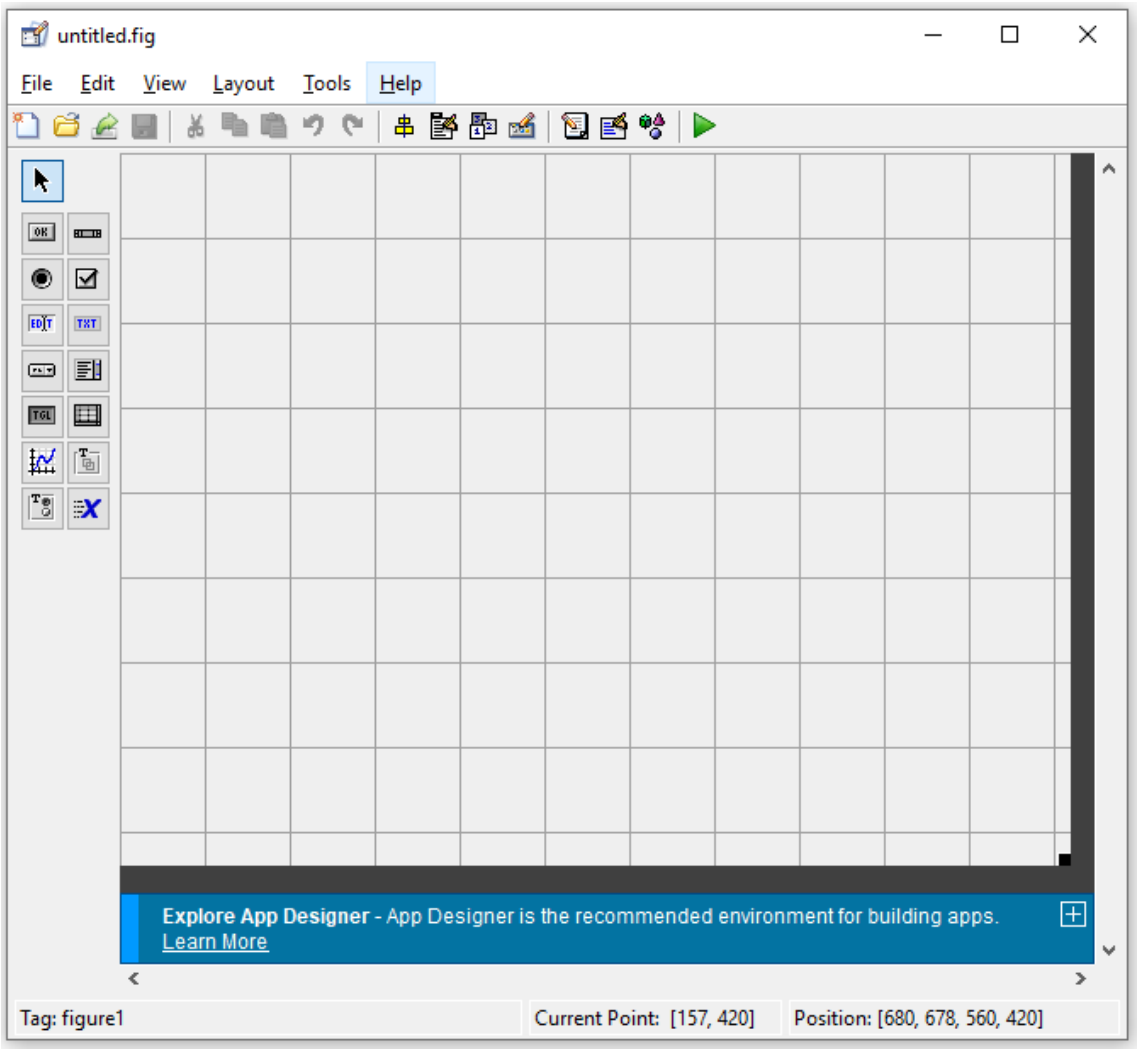

Figura 19: GUI en blanco

Si es la primera vez que usamos GUIDE y no estamos familiarizados con los objetos, podemos hacer que no aparezcan solo con un icono, sino también con el nombre de cada uno de ellos.

Para esto, haremos clic en *File*, seleccionaremos *Preferences* y marcaremos la casilla de *Show names in component palette.*

Si maximizamos la pantalla mostrada en la figura 19 podremos ver que la superficie en la que introducir los objetos es muy reducida, por lo que nos encontraríamos con una ventana en la que no se podrían introducir más de 2 objetos, por lo que necesitaremos un poco más de espacio. Para modificar el tamaño de la ventana de GUIDE lo que debemos hacer es presionar el cuadro negro situado en la parte inferior derecha de la pantalla y arrastrarlo, haciendo tan grande o pequeña la superficie como deseemos.

Ahora ya es momento de empezar a introducir los objetos. Para introducir cualquiera de estos objetos, lo único que debemos hacer es seleccionarlo e indicar el tamaño que queremos que tenga en la superficie de la GUI. Si retomamos el ejemplo de la calculadora que ya hemos visto, podemos observar que necesitamos, como mínimo, un botón que haga la función de iniciar la operación suma, un par de cuadros de texto editables para introducir los números a sumar, y un cuadro de texto fijo donde se muestre el resultado. Partiendo de una idea de lo que queremos hacer, debemos ir introduciendo los objetos básicos que llevará nuestra interfaz, aunque luego los modifiquemos.

Una vez tenemos lo básico, podemos introducir otros objetos para darle un poco de estética a la interfaz, y para ello podemos introducir cuadros de texto fijo en el que se nos indique que función tiene cada uno de los cuadros de texto editables, o qué es lo que aparecerá en el cuadro de texto fijo en el que se muestra la suma. Puede sonar redundante, pero para un comienzo es de gran ayuda tener todo bien detallado. Hay que tener claro que, de momento, no estamos uniendo ninguno de los objetos que introducimos, eso vendrá más adelante cuando comencemos a programar qué hace cada uno de ellos.

El ejemplo de la calculadora que suma se podría mejorar introduciendo las 4 operaciones básicas: suma, resta, multiplicación y división. Esto lo haríamos simplemente introduciendo, en lugar de un pushbutton, 4 distintos dentro de un buttongroup para que solo nos pueda ir mostrando un resultado cada vez.

Una vez tengamos la interfaz como queremos, es hora de empezar a relacionar los objetos que tenemos en la GUI mediante el código. Antes de empezar a buscar dónde programar y qué relaciones poner, debemos tener muy claro qué hace cada uno de los objetos, pues no todos se programan de la misma manera. Para un pushbutton encontraremos una zona del código donde debemos introducir qué pasará cuando presionemos el botón, pero para un edit text o static text no encontraremos una zona específica del código destinada a qué hará este objeto, sino que estos objetos deben programarse dentro de otros, por ejemplo, dentro de un pushbutton.

Ahora, se procederá a explicar cómo se programa cada uno de los objetos:

Pushbutton: en el código encontraremos una línea como la siguiente:

```
function Pushbutton_Callback(hObject, eventdata, handles)
```

y este será el comienzo donde empezaremos a programar nuestro botón. Debemos redactar qué queremos que haga cuando pulsemos el botón.

Edit text: la función de estos objetos es recoger la información escrita en un cuadro de texto. Su programación se reduce a la siguiente función:

```
X=get(handles.edit_text1, 'String')
```

Lo que hace esta función es coger el contenido del cuadro de texto editable edit_text1, con formato string, y guardarlo en la variable X. En el caso de la asignatura de regulación automática, como trabajamos con numerador y denominador en formas de polinomios, es necesario aplicarles una transformación mediante el código `eval(x)` que evalua lo que tenemos en X y eso lo guarda, por ejemplo, en lo que va a ser el numerador de nuestra función de transferencia. Para el caso de una suma, en el que simplemente queremos que lo que introducimos en el edit text se considere un número, podemos hacerlo más sencillo mediante el código:

```
N=str2double(get(handles.edit_text2,'String'))
```

Este Str2double lo que hace es identificar qué tenemos en el string edit_text2 y transformarlo en un número.

Static text: Para los textos estáticos es más sencillo, pues no queremos obtener nada de ellos, sino darles información, y esto lo haremos mediante el comando:

```
set(handles.static_text1,'String',variab)
```

Este comando lo que hará será introducir la información de la variable variab en el texto estático static_text1 en formato string.

Sliders: Un slider nos permite modificar el valor de una variable entre dos valores límite, en función de la posición de un cursor que podemos desplazar a lo largo de una barra. Antes de entrar en la parte de la programación, dentro del inspector de propiedades podemos introducir los valores máximo y mínimo entre los cuales irá cambiando el valor según movamos la barra, pero no es estrictamente necesario ya que también lo introduciremos dentro de la programación. Dentro del código, nos encontramos con un comando como el siguiente:

```
slider_OpeningFcn (hObject, evendata, handles, varargin)
```

debemos introducir, dentro de este comando, lo que harán los distintos sliders que introduzcamos, por ejemplo, un slider que varía de 0 a 1, comienza en 0, y se mueve 0.01 posiciones si lo desplazamos a través de las flechas, o 0.1 si desplazamos manualmente la barra, se programaría así:

```
set (handles.nombreslider, 'Min',0,'Max',1,'Sliderstep',[0.01 0.1]);
```

Según vayamos moviendo los slider, debemos guardar las modificaciones que hagamos con la barra, y simplemente se recogerán en una variable, por ejemplo, var:

```
handles.var=get(handles.nombreslider,'Value')
```

Si queremos mostrar este valor, simplemente lo mostramos en un static text como se ha visto anteriormente.

Radiobutton, CheckBox y ToggleButton: Como ya se explicó, este es uno de los 3 elementos que trabajan de forma binaria, bien toman el valor 0 y hacen algo, o bien toman el valor 1 y hacen otra cosa distinta, Pues bien, dentro de la Callback de cada uno de los elementos, debemos

programar qué hace este objeto en cualquiera de las dos posiciones, y esto se hace con un comando if-else, que puede programarse, por ejemplo:

`A=get(hObject, 'Value')`	aquí lo que hacemos es guardar el valor del radiobutton en una variable
`if A==1`	(Ahora programamos lo que hará la GUI si el radiobutton está marcado)
`else`	(ahora programamos lo que hará la GUI si el radiobutton no está marcado)

PopUp Menu y ListBox: estos objetos funcionan de manera similar a los 3 anteriores, pueden tomar varios valores y dependiendo del que seleccionemos realizarán una acción u otra, pero se diferencian en que los anteriores podían tomar solo 2 valores, y estos dos pueden tomar el número de valores que nosotros queramos. Al tratarse de una elección entre valores, podemos usar los comandos `switch-case` o realizar una cadena de `if elseif`, y en este ejemplo realizaremos una cadena de `if elseif`. Dentro de la Callback de cada uno de estos elementos, debemos recoger el valor de la posición dentro de la lista en la que estemos, siendo 1 el valor de la primera posición y n el valor de la enésima posición:

`C=get(hObject, 'Value')`	Ahora, debemos decir que pasa dentro de cada uno de los valores posibles que puede tomar:
`if C==1`	(Ahora programamos lo que hará la GUI si seleccionamos el primer elemento de la lista)
`elseif C==2`	(Ahora programamos lo que hará la GUI si seleccionamos el segundo elemento de la lista)
`elseif C==3`	(Ahora programamos lo que hará la GUI si seleccionamos el tercer elemento de la lista)
`...`	(así continuaríamos recopilando todos los casos que tengamos, y terminaríamos el código con la palabra end)
`end`	

Panel y ButtonGroup: estos objetos son un espacio reservado en la que se ponen los botones CheckBox, RadioButton y ToggleButton de nuestra GUI y queremos que haya relación entre ellos. Dentro del ButtonGroup, como solo podrá estar activa una de las opciones que tengamos, solo se introducirán RadioButton y ToggleButton, mientras que en el Panel se podrán introducir también los CheckBox. Se puede llegar a la conclusión que la funcionalidad de estos elementos es similar a la del PopUp Menu y la ListBox, pues entre varios valores se selecciona uno o varios de estos. Para la programación de estos objetos, lo que haremos será comprobar si el objeto con un Tag determinado está activo o no, y lo haremos nuevamente con una cadena `if elseif` de la siguiente manera:

```
X=get(hObject, 'Tag')
```

cuando tenemos guardado en X el Tag del elemento activo, lo que haremos será comparar los caracteres de este elemento con los distintos elementos que tenemos dentro del ButtonGroup

con la cadena if elseif. Imaginemos que tenemos 2 RadioButton y un ToggleButton, por lo que la programación quedaría de la siguiente forma:

```
if strcmp(X, 'radiobutton1')==1
```
(Ahora programamos lo que hará la GUI si seleccionamos el primer elemento del ButtonGroup)

```
elseif strcmp(X, 'radiobutton2')==1
```
(Ahora programamos lo que hará la GUI si seleccionamos el segundo elemento del ButtonGroup)

```
elseif strcmp(X, 'togglebutton1')==1
```
(Ahora programamos lo que hará la GUI si seleccionamos el tercer elemento del ButtonGroup)

```
...
```
(así continuaríamos recopilando todos los casos que tengamos, y terminaríamos el código con la palabra end)

```
end
```

Para el caso del panel, al poder trabajar con varios CheckBox activos al mismo tiempo, nos interesa que cada una de las operaciones sea independiente, por lo que cada uno de los CheckBox se programará de manera independiente al resto, para que no influya el estado de uno con el resto. Esta programación la haremos mediante el comando `if else`.

Esto lo podemos hacer bien de la misma manera que el anterior, comparando los Tag de los elementos activos, o bien los valores de los mismos, indicando lo que harán si están activos los CheckBox o si no lo están, y al estar en líneas independientes no interaccionan:

```
if strcmp(X, 'checkbox1')==1
```
(Ahora programamos lo que hará la GUI si el primer elemento del panel está activo)

```
else
```
(ahora programamos lo que haría si no está activo)

```
if strcmp(X, 'checkbox2')==1
```
(Ahora programamos lo que hará la GUI si el segundo elemento del panel está activo)

```
else
```
(ahora programamos lo que haría si no está activo)

```
if strcmp(X, 'checkbox3')==1
```
(Ahora programamos lo que hará la GUI si el tercer elemento del panel está activo)

```
else
```
(ahora programamos lo que haría si no está activo)

```
end
```

Tabla: Las tablas son los objetos más complejos. Podemos trabajar con este componente bien desde el editor de propiedades o bien, como se ha visto con los demás, desde el propio programa. En esta guía se mostrará la manera de trabajar desde el editor de propiedades porque

es la forma más sencilla de empezar a trabajar con las tablas. Una vez introducimos una tabla en nuestro espacio de trabajo, hacemos clic sobre el botón derecho del ratón sobre la tabla y seleccionamos la última opción donde pone *Table Property Editor…* Ahora tendremos cuatro opciones: columnas, filas, datos y colores. Comenzaremos por la introducción de un vector de datos que escribiremos en la ventana de comandos de la siguiente manera, por ejemplo:

```
M={VECTOR DE DATOS}
```

Donde 'VECTOR DE DATOS' es el vector donde tenemos los datos que queremos introducir en nuestra variable M, introduciéndolo separando los datos por comas.

Ahora, entrando en el editor de propiedades de la tabla, hacemos lo siguiente:

1- Entramos al apartado de Datos
2- Nos aparecerá una lista con las variables que hayamos creado en MATLAB, en nuestro caso, seleccionamos la variable M
3- De las 3 opciones que nos salen encima de las variables, seleccionamos la que pone *Change data value to the selected workspace variable below*
4- Aplicamos los cambios y salimos dándole al Ok.

Ahora, nuestra tabla tendrá los datos que hemos introducido y se habrá ajustado a las dimensiones de dicha matriz.

Podemos también cambiar los encabezados de las columnas para que quede claro qué trata cada una, y eso lo haremos mediante el editor de propiedades, siguiendo estos pasos:

1- Entramos al apartado Columnas
2- Seleccionamos la tercera opción que dice *Show names entered below as the column headers* (que es para cambiar los nombres de las columnas)
3- Modificamos debajo los nombres introduciendo los títulos que queramos
4- Aplicamos los cambios y salimos dándole al Ok.

En esta pestaña también podemos seleccionar si queremos que el contenido en esa columna sea editable o no, o el formato de la columna. Por ejemplo, si el vector de datos introducido es el siguiente:

```
M={'Pablo', 'Regulación', 8.5, [true]}
```

en nuestra primera columna que contiene el nombre del alumno nos interesa que esa columna tenga un formato Texto, el segundo podría ser una lista de opciones para elegir la materia, seleccionaríamos la opción *Choice list* y justo después nos abriría una ventana para redactar todas las opciones posibles; la tercera opción sería formato numérico pues consta de la nota, y la cuarta opción tendría formato lógico, seleccionando o no la casilla y así podríamos indicar si el alumno aprueba o no. En cuanto a las filas, al funcionar de igual manera que las columnas, se editan de la misma manera.

Axes: Este objeto es el que nos permite realizar las representaciones gráficas de todo aquello con lo que vayamos trabajando. Al igual que los Static Text y Edit Text, las gráficas no tienen por qué tener una función estrictamente para ellas como se puede ver en los pushbutton o los sliders, entre otros, sino que lo que debemos hacer es introducir la programación de estos Axes dentro de algún otro objeto, por ejemplo, un pushbutton. Lo que deberemos hacer será mandar

a realizar la gráfica de aquello que queremos dibujar, por ejemplo, si queremos realizar una gráfica de una función cos(x), deberemos programarlo de la siguiente manera:

`x=linspace(0,10,100);` (Aquí lo que hacemos es decir que x va de 0 a 10 y se dibuja con 100 puntos de precisión)

`plot(handles.axes1, x, cos(x))` (con el primer argumento, handles.axes1, indicamos que es en la gráfica 1 donde queremos que muestre el resultado del comando [en caso de solo tener una gráfica en nuestro espacio de trabajo no haría falta pasarle este argumento]; con el segundo argumento, indicamos que queremos dibujar en el dominio de x, y con el tercero, indicamos que queremos graficar es el coseno de x)

ActiveX Control: Esta herramienta no se va a explicar cómo se utiliza ya que, dada su función, que es la de crear objetos con movimiento, no tiene sentido para el tipo de aplicaciones que se verán en este proyecto.

Una vez conocidos de forma básica cómo funciona y cómo se programa cada uno de los objetos dentro de GUIDE, para comenzar un programa lo único que hay que hacer es ir uniendo mediante código cómo interactúan cada uno de estos objetos con el resto de ellos dentro de nuestra GUI.

Como ejemplo, en la figura 20 se muestra una mejora del ejemplo de una calculadora, donde se pueden introducir 2 números y realizar las 4 operaciones básicas: sumar, restar, multiplicar y dividir:

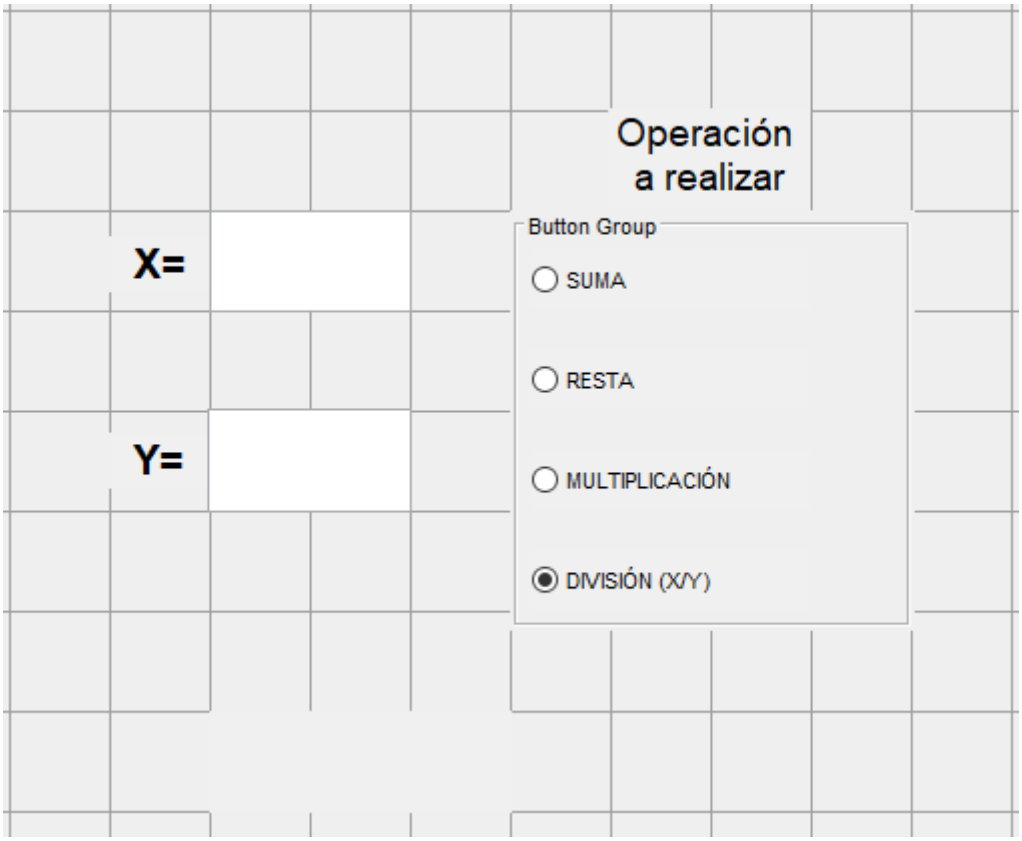

Figura 20: Interfaz de una calculadora

El código de esta interfaz es el siguiente:

```
function varargout = calculadora(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @calculadora_OpeningFcn, ...
                   'gui_OutputFcn',  @calculadora_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end


% --- Executes just before calculadora is made visible.
function calculadora_OpeningFcn(hObject, eventdata, handles, varargin)

%Establecemos las cuatro posibles operaciones como inactivas en un
inicio
set(handles.suma,'Value',0);
set(handles.resta,'Value',0);
set(handles.multiplicacion,'Value',0);
set(handles.division,'Value',0);

% Choose default command line output for calculadora
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes calculadora wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = calculadora_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;



function n1_Callback(hObject, eventdata, handles)
% hObject    handle to n1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function n1_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to n1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function n2_Callback(hObject, eventdata, handles)
% hObject    handle to n2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function n2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to n2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%Codigo del Buttongroup
function uibuttongroup1_SelectionChangedFcn(hObject, eventdata,
handles)

%Obtenemos el tag del RadioButton que está activo
Z=get(hObject, 'Tag')

%Obtenemos los números introducidos en los cuadros de texto editables
%Y los convertimos a número (ya que se introducen como STRING)
x=eval(get(handles.n1,'string'))
y=eval(get(handles.n2,'string'))

%Comprobación de qué RadioButton está activo y sus acciones
if strcmp(Z, 'suma')==1
res=num2str(x+y);
set(handles.resultado, 'string', res);
elseif strcmp(Z, 'resta')==1
res=num2str(x-y);
set(handles.resultado, 'string', res);
elseif strcmp(Z, 'multiplicacion')==1
res=num2str(x*y);
set(handles.resultado, 'string', res);
elseif strcmp(Z, 'division')==1
res=num2str(x/y);
set(handles.resultado, 'string', res);

end
```

Los puntos clave a programar en esta interfaz han sido:

- En la función `function calculadora_OpeningFcn(hObject, eventdata, handles, varargin)` se han introducido los valores que queremos que tengan las opciones de las operaciones en un inicio, y se han puesto todas a 0 para que ninguna esté seleccionada en un principio.

- En la función `function uibuttongroup1_SelectionChangedFcn(hObject, eventdata, handles)` se ha programado la obtención de los números introducidos en los cuadros de texto editables, la selección de las operaciones dentro del ButtonGroup y la impresión del resultado en el cuadro de texto fijo vacío.

- Como se puede observar, hay mucho más código, pero no es necesario hacer ninguna modificación en este pues contiene la creación y la llamada de algunos objetos con los que se puede interactuar, como las funciones Callback y las funciones CreateFcn. Estas líneas de código sirven para el correcto funcionamiento del programa, pero, en este caso, no nos interesa cambiarlos.

CAPÍTULO 5: SÍNTESIS DEL TEMARIO DE REGULACIÓN AUTOMÁTICA

En este capítulo se realizará un análisis de los contenidos de la asignatura de Regulación Automática y se identificarán las funcionalidades que puedan ser introducidas en nuestra GUI.

Debemos tener siempre en cuenta el objetivo de esta aplicación, y es su uso en un entorno de enseñanza y aprendizaje para complementar la asignatura de Regulación Automática, por lo que esta GUI se centrará en algunos de los contenidos de la asignatura, y para decidir cuáles, se va a realizar este análisis de la asignatura.

Al tratarse de un entorno de control, está claro que hablaremos de funciones de transferencia, sistemas de primer y segundo orden, polos y ceros, frecuencias amortiguada (wd) y no amortiguada (wn), y muchos más términos relacionados.

Pues bien, podemos empezar describiendo una GUI sencilla, en la que podamos introducir una función de transferencia, en formato [numerador] y [denominador], y que nos devuelva, por ejemplo:

- Polos y ceros (valores numéricos y representados en un gráfico)
- Frecuencias amortiguada y no amortiguada
- Gráfica de la respuesta del sistema

Para el caso de un sistema de 1er orden simple, nos interesaría tener la función de trasferencia de la siguiente manera:

$$\frac{X(s)}{F(s)} = \frac{K}{T \cdot s + 1} \qquad \{1\}$$

Donde:

- K es la ganancia del sistema
- T es la constante de tiempo
- -1/T es el polo del sistema

También podríamos encontrarnos con un sistema de 2º orden, donde podríamos pedir que nos simplificase la función de transferencia, y que nos la mostrase en el siguiente formato:

$$\frac{X(s)}{F(s)} = K \cdot \frac{w_n^2}{s^2 + 2\varsigma s + w_n^2} \qquad \{2\}$$

Donde:

- X(s) es la salida del sistema
- F(s) es la entrada del sistema
- K es la ganancia estática del sistema
- W_n es la frecuencia natural no amortiguada del sistema
- ς es el coeficiente de amortiguamiento

Con W_n y ς podemos calcular la frecuencia amortiguada (Wd):

$$W_d = W_n\sqrt{1 - \varsigma^2} \qquad \{3\}$$

Por otro lado, también podría ser interesante ver el caso contrario, cuando no conocemos la función de transferencia con la que se trabajará, pero si sabemos los parámetros característicos que debe tener, por ejemplo, con un sistema de 2º orden, donde debería el alumno introducir los valores de ganancia, frecuencia natural (Wn) y el factor de amortiguamiento ς, para que nos devuelva la función de transferencia correspondiente.

Asimismo, se puede dar el caso en que el alumno, en vez de algunos de estos parámetros como constante de tiempo, frecuencia natural o factor de amortiguamiento, sepa, o quiera establecer, donde estarán situados los polos y ceros de la función, por lo que, al tener dichos polos y ceros (por ejemplo, un cero C1 y dos polos P1 y P2), que nos devuelva una función de transferencia:

$$\frac{X(s)}{F(s)} = K \cdot \frac{(C1 + s)}{(P1 + s) \cdot (P2 + s)} \qquad \{4\}$$

Donde, si se realizan los ajustes necesarios, seríamos capaces de observar de manera sencilla una función de transferencia como la situada en la ecuación {1}.

Otra opción interesante podría ser la determinación por parte de la GUI de los parámetros del transitorio de la función de transferencia, pero del mismo modo también se podría añadir la opción de que, conociendo o estableciendo unos parámetros del transitorio, nos devolviese una función de transferencia acorde a nuestros requerimientos.

Como se puede observar, hay distintos ejercicios a plantear, y todos y cada uno de ellos interesante, y son varios puntos de vista desde los cuales se puede plantear la asignatura, por lo que sería conveniente tenerlos todos en cuenta e intentar abarcarlos todos ellos.

Teniendo en cuenta que esta herramienta que vamos a crear está enfocada a la ayuda en la asignatura de REGULACIÓN AUTOMÁTICA, sería interesante repasar los contenidos de esta asignatura, tema por tema, para poder saber qué funciones debe tener nuestra GUI, por lo que, a continuación, se irá sintetizando el contenido de la asignatura por temas, y empezaremos por el tema 2, ya que el primer tema es una breve introducción a la asignatura, y no tiene mucho interés verlo aquí. [10]

Análisis de la asignatura

TEMA 2

En este tema se ve, principalmente, que, aunque para nosotros sea más sencillo entender las funciones en el dominio del tiempo, a la hora de trabajar con ellas es más sencillo aplicarles la transformada de Laplace y así la complejidad matemática se reduce considerablemente, pasando así al dominio de la frecuencia. En este tema se ven todas las utilidades, propiedades y características de la transformada de Laplace, por lo que, en definitiva, y en relación con nuestra GUI, no tiene mucha relación.

En el dominio del tiempo, nos encontramos que tendríamos ecuaciones diferenciales, que como su propio nombre indica, son ecuaciones que contienen derivadas, y este es el motivo por el que es más sencillo trabajar en el dominio de la frecuencia, pues las expresiones se reducirán a fracciones con polinomios en el numerador y denominador.

Tras trabajar con nuestras expresiones en el dominio de la frecuencia, debemos aplicar la anti transformada para volver al dominio del tiempo. Para realizar esta anti transformada, uno de los primeros pasos a hacer es descomponer en fracciones simples las expresiones. Entonces, esta podría ser una herramienta interesante, pues ayudaría a los alumnos saltando algunos pasos, pues, aunque esto no sea complicado, puede llevar algo de tiempo.

TEMA 3

En el tema 3 se prepara a los alumnos para que sean capaces de realizar diagramas de influencia de sistemas físicos, de establecer los sistemas de ecuaciones diferenciales, trabajar con ellos para poder realizar un diagrama de bloques como el mostrado más adelante en la figura 18 y, lo más interesante, poder determinar la función de transferencia simplificado de un sistema físico. Este será nuestro punto de partida, pues la GUI, como ya se ha comentado, trabajará con la función de transferencia del sistema.

Como se ve en la asignatura, un sistema físico presenta una serie de ecuaciones que lo modelan, por ejemplo, un tanque con una entrada y una salida. A partir de ellas podremos analizarlo mediante sus ecuaciones características, y podremos entender cómo funciona mediante el diagrama de influencias.

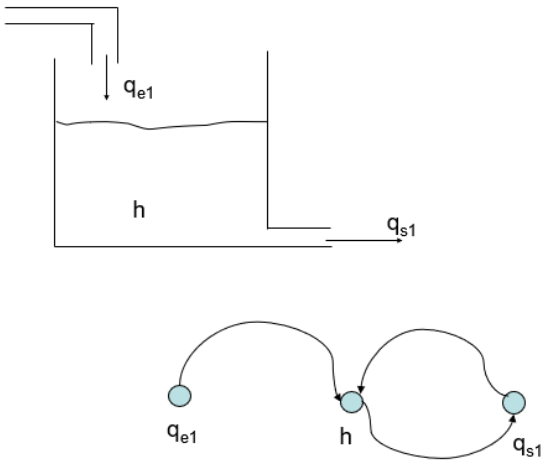

Figura 17: Sistema físico de intercambio de materia

Como podemos apreciar en la figura 17, la entrada afectará a la altura del líquido en el tanque, y esta altura influirá en el caudal de salida, pero este caudal de salida también afecta en la altura del líquido.

Otro tipo de sistemas conocidos podrían ser los sistemas eléctricos, mecánicos, electromecánicos o, simplemente, los sistemas con intercambio de materia, entre los cuales está el ejemplo del tanque.

Una vez tenemos claro cómo funciona nuestro sistema, es hora de realizar el diagrama de bloques, donde precisaremos las influencias matemáticas de cada una de las partes de nuestro sistema. Este diagrama, posteriormente, se simplificará para que quede como un solo bloque que relacione la entrada con la salida.

Como se ha comentado, en este tema se ven las ecuaciones y relaciones típicas en algunos sistemas físicos (eléctrico, mecánico, electromecánico o intercambio de materia), por lo que una opción que puede ser interesante estudiar podría ser la implementación de una función que ya tenga algunas de estas ecuaciones predefinidas, por lo que el alumno solo debería introducir los datos que tenga él en su problema a estudiar y el programa se encargaría de darle los resultados para su problema.

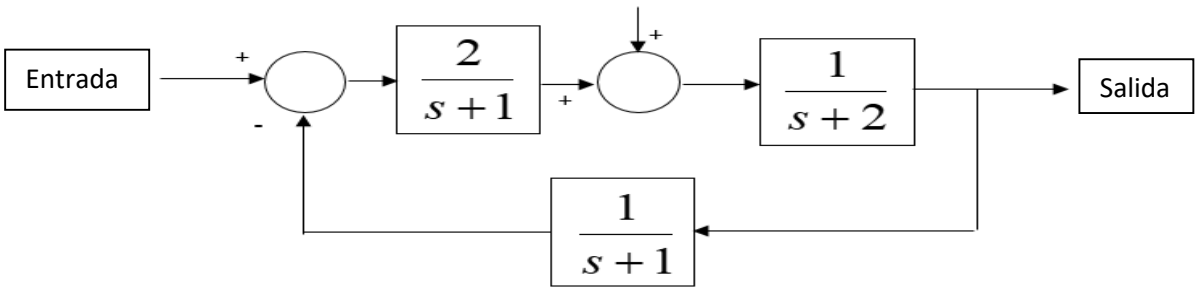

Figura 18: Diagrama de bloques

TEMA 4

Ahora es cuando se comienza a trabajar con la función de transferencia que en el tema 3 hemos arreglado y dejado simplificada. Aprendemos en este tema a determinar la estabilidad del sistema, su ganancia, estudiar el transitorio del sistema a partir de la función de transferencia. Algunas de las funciones que podríamos pedirle a la GUI sería que nos mostrase por pantalla la posición de los polos y ceros de la FDT, así podríamos identificar fácilmente la posición del polo o de los polos dominantes y distinguirlos del polo o polos adicionales y de los ceros, y, asimismo, poder conocer la respuesta de nuestra FDT como se enseña en esta unidad.

En este tema también se aprende un poco a distinguir entre los distintos tipos de comportamiento en régimen permanente (estable, limitadamente estable, marginalmente estable e inestable), y a como se representan gráficamente ante una determinada entrada, por lo que también podría ser interesante que nos mostrase por pantalla la respuesta de dicho sistema.

Cuando nos encontramos ante un sistema de segundo orden, podemos tener un sistema no amortiguado (un muelle sin rozamiento viscoso, por ejemplo) en el que la salida tendrá una forma sinusoidal y no tenderá a un valor fijo, sino que oscilará en torno a un valor.

También podemos tener un sistema subamortiguado, y podría ser interesante que, introduciendo los parámetros de nuestra función de transferencia y la entrada, que nos calculase los parámetros característicos del transitorio, como puede ser el tiempo de pico t_p, el pico de la salida y_p, la sobreoscilación M_p o el tiempo que tardará en llegar al régimen permanente t_s.

Lo visto anteriormente es para sistemas con 2 polos, pero se puede dar el caso que existan más polos, ceros o retardos en nuestro sistema, por lo que también puede ser interesante que se desarrollase una opción que nos permita ver el efecto de estos polos adicionales o ceros en la salida de nuestro sistema.

TEMA 5

En el tema 5 se introduce el concepto de *lugar de las raíces*, que indica de forma gráfica las raíces de la FDT cuando esta tiene un parámetro que puede variar (en la asignatura, este parámetro se conoce como MAN).

La herramienta del lugar de las raíces es muy útil para poder, de un simple vistazo, ver cómo será la respuesta de nuestra FDT en función del valor variable, pues en el tema 4 se ha visto que, según la posición de los polos y ceros en el plano complejo, la respuesta varía. Por esta razón, implementar esta funcionalidad en la GUI podría resultar muy beneficioso para el alumno, una herramienta que nos diga cómo las raíces del sistema varían con cada valor que toma el parámetro denominado MAN.

También se habla de los errores de un sistema, que aparecen cuando existe una realimentación para tener un control más preciso de nuestro sistema. Se distinguen 3 tipos de error:

- Error de posición: Se da cuando la entrada es un escalón. Error del sistema si se desea una salida constante.
- Error de velocidad: Se da cuando la entrada es una rampa. Error del sistema si se desea una salida proporcional al tiempo
- Error de aceleración: Se da cuando la entrada es una parábola. Error del sistema si se desea una salida que varía en forma de parábola (muy rápido el cambio)

Estos errores tienen una expresión, y finalmente se llega a que cada uno de estos valores depende de los bloques en la realimentación, y que se pueden simplificar por un parámetro constante: Kp es la constante de error de posición, Kv es la constante de error de velocidad y Ka es la constante de error de aceleración.

TEMA 6

El tema 6 está enfocado a los reguladores, y esto es algo que se añadirá, de forma posterior, al diagrama de bloques o a la función de transferencia.

Los reguladores nos sirven para controlar la salida de nuestra FDT, puesto que, una función de transferencia inicial puede presentar, por ejemplo, tiempos del transitorio muy altos, así como puede tener errores relativamente altos, por lo que, con estos añadidos, se podrían conseguir tiempos del transitorio más bajos y errores más pequeños o incluso nulos.

Los reguladores se introducen justo después de la señal de error, señal que se obtiene al comparar la señal de entrada y la señal realimentada. Estos reguladores pueden ser proporcionales, integrales o diferenciales, o cualquier combinación entre ellos, por lo que también se les llama PID.

Gracias a estos PID podemos controlar la variación de la señal de error, para que los cambios que se produzcan en la señal realimentada afecten más o menos a nuestro sistema. Esto significará que podemos hacer que la señal de error sea muy sensible a los cambios en la señal de realimentación o que, por el contrario, sea algo más difícil que esta se modifique.

TEMA 7

Por último, en este tema de la asignatura se aprende a trabajar con los diagramas de Bode, que son herramientas que nos permiten calcular de una manera muy sencilla, para señales senoidales y en régimen de la frecuencia, los parámetros que tendrá la salida, teniendo en cuenta que la entrada y la salida comparten la misma frecuencia, que existirá un desfase entre ambas señales, y que la señal estará amplificada o atenuada.

Opciones como estas podrían ser beneficiosas a la hora del desarrollo de una GUI de ayuda para el alumno, pues son herramientas que ayudarían a la comprensión y estudio de los sistemas que se tratan en la asignatura.

CAPÍTULO 6: CREACIÓN DE NUESTRA GUI

Es ahora cuando empezamos a trabajar directamente con MATLAB y comenzamos con el diseño de nuestra GUI. Se pretende en este capítulo realizar una redacción detallada de cómo se va creando esta interfaz.

El programa que estamos diseñando trata de una interfaz de ayuda para los alumnos que estudien regulación automática.

Finalmente se ha optado por desarrollar 3 funcionalidades, que se seleccionarán con 3 pestañas distintas. La primera pestaña servirá para la introducción de la función de transferencia, puesto que esto es lo primero con lo que hay que interactuar a la hora de trabajar con una función de transferencia. La segunda pestaña tendrá la función de representar la respuesta del sistema, ya que la evolución temporal de la salida de un sistema es una herramienta clave a la hora de trabajar con una función con parámetros definidos. Por último, la tercera pestaña contendrá la representación del lugar de las raíces, y esta es una herramienta fundamental a la hora de estudiar sistemas con un parámetro variable.

Comenzamos por presentar la primera pestaña de nuestro programa, donde se introduce la función de transferencia con la que se trabaja, escribiendo el numerador y denominador de la misma:

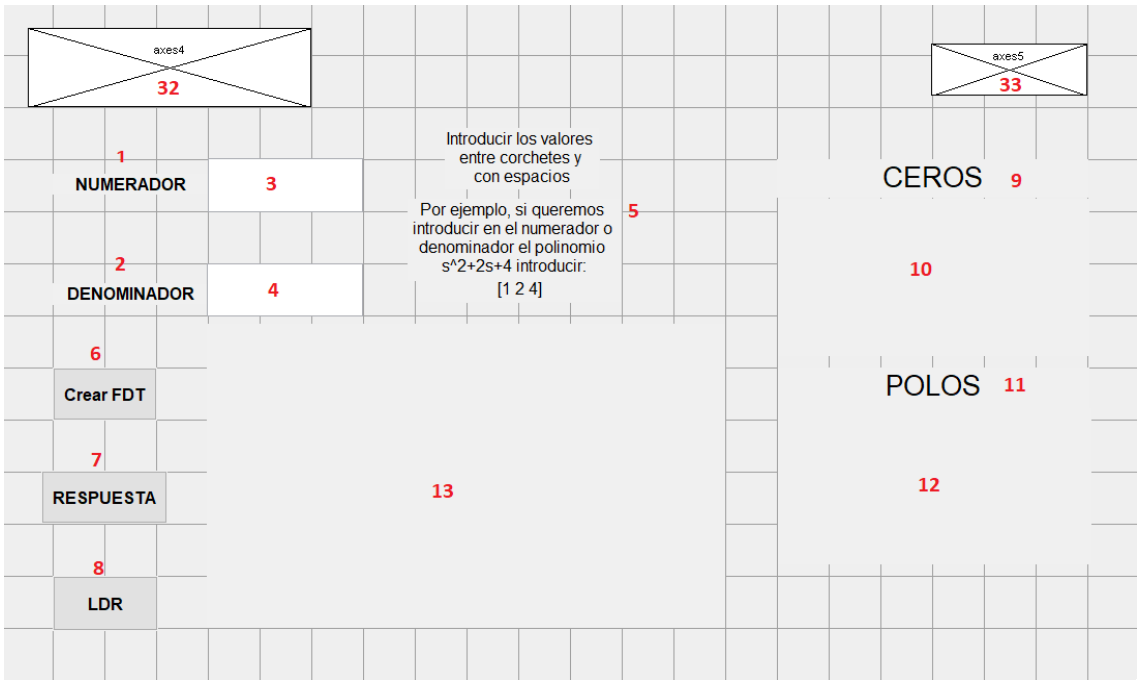

Figura 21: Pestaña principal GUI

En la figura 21 podemos observar la primera pestaña con distintos elementos que han sido enumerados, y ahora se irá describiendo uno a uno, por grupos, como se han ido personalizando.

Elementos 1, 2, 5, 9 y 11: STATIC TEXT. Estos cuadros de texto son puramente información escrita en nuestra pantalla, sirven para describir qué se mostrará o qué se debe introducir en los elementos adyacentes. Por ejemplo, en el cuadro de texto estático número 1 pone “NUMERADOR”, que nos indica que en el cuadro de texto editable 3 debemos introducir el

numerador de la función de transferencia, o el elemento 9, que indica que el cuadro de texto 10 nos mostrará los ceros de la función de transferencia. Para la personalización de este tipo de cuadro de texto estático, hacemos clic sobre ellos y abrimos el inspector de propiedades, donde se han modificado los parámetros “FONTSIZE” para cambiar el tamaño del texto; “FONTWEIGHT”, para decidir si ponerlos en negrita o normal; y “STRING”, para mostrar el texto que queremos que lleve cada cuadro. No es necesario cambiar el “TAG” de estos cuadros de texto ya que no se utilizarán en el código, pues no queremos que el texto que muestran se modifique.

Elementos 10, 12 y 13: STATIC TEXT. Estos cuadros son del mismo tipo que los anteriores, son cuadros de texto estático, pero no nos enseñarán información de primeras. Lo que haremos será introducirlos en el código de nuestro programa para que nos enseñen información relacionada con la función de transferencia introducida. El cuadro de texto 13 nos mostrará la función de transferencia introducida, lo que nos permitirá revisar fácilmente si hemos cometido errores al escribirla. Por otro lado, los cuadros 10 y 12 nos mostrarán los ceros y polos, respectivamente, de la función de transferencia con la que trabajamos. Al igual que el cuadro de texto estático anterior, modificaremos: “FONTSIZE” y “FONTWEIGHT” para que quede estéticamente agradable ver los resultados y eliminaremos “STRING” para que queden como cuadros en blanco de primeras. En este caso sí es interesante modificar el parámetro “TAG” para poder diferenciarlo de manera sencilla en nuestro código, ya que, de manera predeterminada, estos cuadros vienen con un nombre genérico “text#” donde # es el nº del cuadro generado en orden de creación.

Elementos 3 y 4: EDIT TEXT. Estos son los cuadros de texto editable donde podremos escribir la información sobre nuestra función de transferencia. En el cuadro 3 deberemos escribir el numerador de nuestra función y en el cuadro 4 el denominador. En el cuadro 5 nos explica cómo debemos introducirla para que el programa pueda entenderla, que es entre corchetes y separando los números por espacios. Las propiedades que hemos cambiado en los cuadros de texto editable han sido “FONTSIZE”, para que el texto que escribamos se vea bien; modificamos el ‘’TAG’’ para poder trabajar de forma cómoda con ellos en nuestro código; y hemos retirado el ‘’STRING’’, ya que no queremos que en estos cuadros se vea nada escrito de primeras.

Elementos 6, 7 y 8: PUSHBUTTON. Nos encontramos con 3 botones en nuestra GUI: el primero, llamado “Crear FDT”, hace exactamente eso, utilizar los valores introducidos en los cuadros de texto editable 3 y 4 para crear una función de transferencia y mostrarnos, en el cuadro 13 la fracción, y en los cuadros 10 y 12 los ceros y polos. Después, tenemos el botón de la respuesta, que nos conducirá a una segunda pantalla donde encontraremos la respuesta del sistema. Por último, tenemos un botón llamado LDR, y lo que hará será llevarnos a una tercera pestaña de la GUI para mostrarnos una gráfica con el lugar de las raíces en función de un valor variable. En cuanto a la personalización de estos botones, se modifican las propiedades “FONTSIZE” y “FONTWEIGHT” para que quede estéticamente agradable ver los botones, en el parámetro “STRING” escribimos el nombre del botón y en el parámetro “TAG” escribimos un nombre similar para poder identificarlo y programar con el de manera sencilla.

Elementos 32 y 33: Axes. Estos cuadros nos servirán como base para introducir los logotipos en nuestra GUI. En cuanto a las modificaciones en los parámetros de la gráfica, se cambiarán los “TAG” para diferenciarlos del resto y que no haya confusiones.

En la figura 22 seguimos con la segunda pestaña, que nos mostrará la respuesta del sistema ante 2 tipos de entradas unitarias que podremos seleccionar:

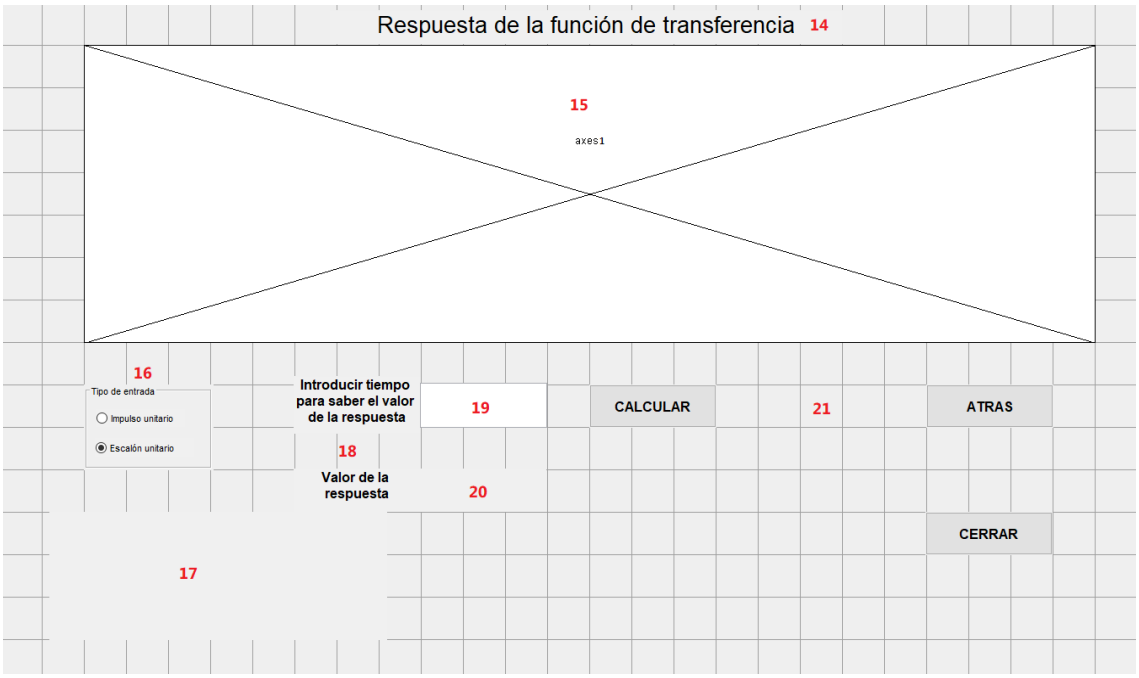

Figura 22: Segunda pestaña GUI

Elementos 14 y 18: STATIC TEXT. Como se ha visto en la pestaña anterior, estos cuadros de texto estáticos tienen como finalidad proporcionar información escrita en pantalla, siendo el elemento 14 el título del gráfico y el 18 son dos indicadores de qué debemos introducir en el cuadro 19 y lo que se nos mostrará en el cuadro 20.

Elementos 17 y 20: STATIC TEXT. Aquí nos volvemos a encontrar con el segundo tipo de cuadros de texto estático. Tenemos dos: el cuadro número 17, en el que se muestra la función de transferencia que hemos introducido, y el número 20, que muestra el valor de la respuesta de nuestro sistema ante una entrada de escalón unitario para un tiempo que hayamos introducido en el cuadro de texto editable número 19.

Elemento 15: AXES. En este gran cuadro blanco se nos mostrará la evolución temporal de la respuesta de nuestra función de transferencia ante la entrada unitaria que indicaremos en el elemento 16. En cuanto a las modificaciones en los parámetros de la gráfica, lo que podría ser interesante a efectos prácticos sería cambiar el “TAG”, pero yo he decidido dejar el que venía por defecto, que es axes1 ya que no vamos a tener muchos gráficos y se pueden identificar fácilmente mediante números.

Elemento 16: ButtonGroup. Aquí tenemos un ButtonGroup con dos RadioButton. Se ha escogido un ButtonGroup y no un panel porque interesa que sólo una de las dos opciones esté activa. La personalización de este objeto se separa en dos: primero, el propio ButtonGroup, donde se ha escogido el título que se le pone en el inspector de propiedades; y por otro lado los RadioButton, que se les ha puesto el nombre del tipo de entrada que representan, modificando el parámetro ‘’STRING’’ y el ‘’TAG’’ para identificarlos en la programación.

Elemento 19: EDIT TEXT. En este cuadro de texto editable podremos introducir un tiempo t para el que queramos conocer el valor de la respuesta de nuestro sistema ante una entrada en forma de escalón unitario.

Elementos 21: PUSHBUTTON. Nos encontramos igual que antes con 3 botones: el primero con la etiqueta ‘’Calcular’’, que nos servirá para que se nos muestre en el cuadro 20 el valor de la salida para el valor de tiempo que hayamos introducido en el cuadro 19; el segundo, donde tenemos el botón llamado “ATRAS”, que nos vuelve a llevar a la pestaña inicial para introducir unos nuevos valores para la función de transferencia; y, por último, un tercer botón que se llama “CERRAR”, que cierra el programa.

Por último, en la figura 23 tenemos nuestra tercera pestaña, en la que nos encontraremos con el lugar de las raíces:

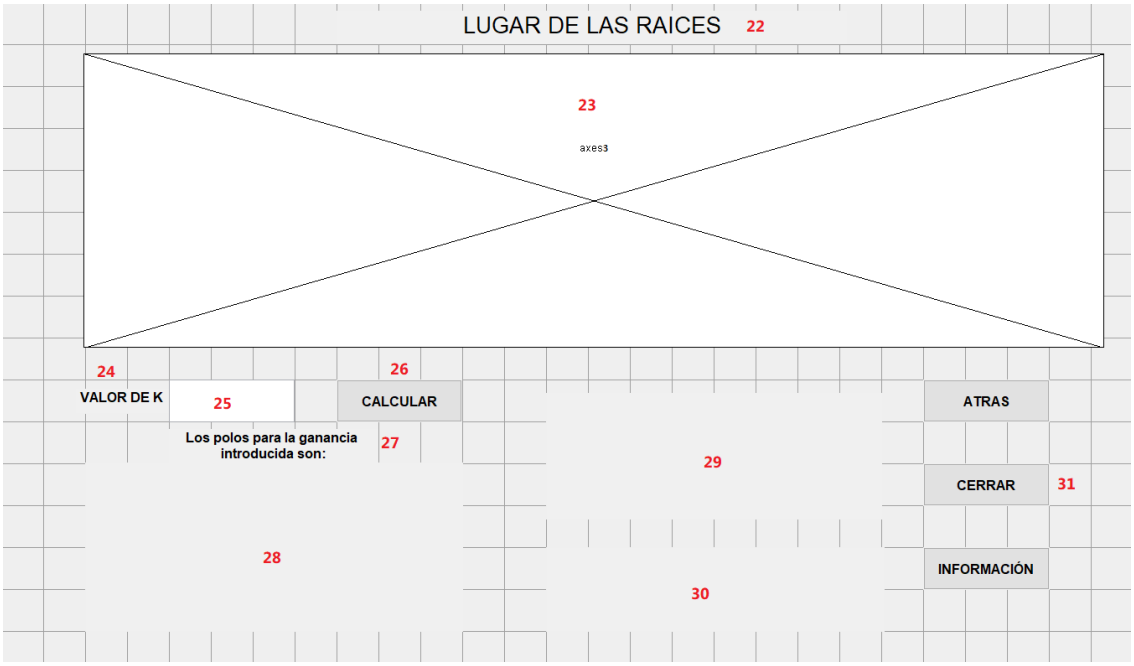

Figura 23: Tercera pestaña GUI

Elementos 22, 24 y 27: STATIC TEXT. De nuevo nos encontramos con estos cuadros de texto estáticos, que tienen como finalidad proporcionar información escrita en pantalla, siendo el elemento 22 el título del gráfico, y por otro lado tenemos el 24 y el 27 que nos indican qué debemos introducir en el cuadro 25 y qué nos aparecerá en el cuadro 28.

Elementos 28, 29 y 30: STATIC TEXT. Estos cuadros de texto estático vuelven a estar vacíos ya que se nos mostrará información en base a la función de transferencia con la que trabajemos. En el cuadro 28 se muestran los polos, cuando introducimos un valor en la variable K; en el cuadro 29 tendremos presente la función de transferencia que hemos introducido; y en el cuadro 30 tendremos un cuadro informativo que nos indica la función de esta función de transferencia en el caso de tener un parámetro variable, pues consideramos esta función de transferencia como parte de la ecuación 5.

Elemento 25: EDIT TEXT. En este cuadro de texto editable podremos introducir el valor de K para el que queramos conocer el valor de las soluciones de la ecuación 5.

Elemento 23: AXES. Este cuadro nos enseñará un gráfico con el lugar de las raíces correspondiente a la función de transferencia. El “TAG” que venía por defecto era axes1, y aunque los gráficos estén en pestañas separadas, que significa que están en códigos distintos, para evitar cualquier tipo de fallo posible se le ha modificado el nombre a axes2.

Elementos 26 y 31: PUSHBUTTON. Por último, nos encontramos con cuatro botones. El número 26 es un botón que, al presionarlo y habiendo escrito un valor en el cuadro de texto editable 25, nos muestra en el cuadro de texto 28 el valor de los polos. En cuanto al conjunto de botones 31, tenemos en primer lugar un botón llamado "ATRAS", que nos vuelve a llevar a la pestaña inicial para introducir unos nuevos valores para la función de transferencia. El segundo, llamado "CERRAR", cierra el programa. Por último, un botón llamado ''INFORMACIÓN'', nos muestra en el cuadro de texto 30 cómo se trabaja con esta función de transferencia cuando tenemos un parámetro variable.

Programa con varias pestañas

Como se puede observar, nuestro programa está compuesto por 3 pestaña que se mandan información de unas a otras.

Para crear un programa con más de una pestaña, lo que hay que hacer es crear las pestañas como GUIs independientes, tener los archivos .m y .fig de las pestañas en una misma carpeta en nuestro ordenador, y para cambiar de una a otra lo que tendremos que hacer será, en el código de un Pushbutton, escribir el nombre de la pestaña que queremos abrir y cerrar la pestaña actual con el comando "close (nombre de la pestaña)".

Por otro lado, para poder compartir la información entre las pestañas, lo que habrá que hacer será, en el código del programa, establecer como globales las variables que queremos que sean comunes para las distintas pestañas, y esto se hace mediante el comando "global (nombre de las variables que queramos globales)".

CAPÍTULO 7: PROBANDO LA GUI

Tras haber creado el programa, debemos asegurarnos de que está bien programado, que no hay errores y que funciona correctamente.

Este capítulo dividirá las pruebas en tres categorías: primero, se realizarán pruebas acerca de la introducción de datos en las distintas ventanas; segundo, se trabajará con la segunda pestaña del programa, la ventana de la respuesta, y se introducirán datos específicos para obtener las respuestas deseadas, como un sistema de primer grado, sistemas de segundo grado sobreamortiguado, subamortiguado y con ceros o polos adicionales, y por último un sistema inestable; el tercer y último apartado de pruebas sería trabajar con la ventana del lugar de las raíces de cada uno de los sistemas vistos previamente, analizando las gráficas que nos muestre el programa.

Introducción de datos

Comenzaremos esta prueba introduciendo datos en la primera pestaña de nuestro programa, lo primero que se solicitan son los datos de la función de transferencia.

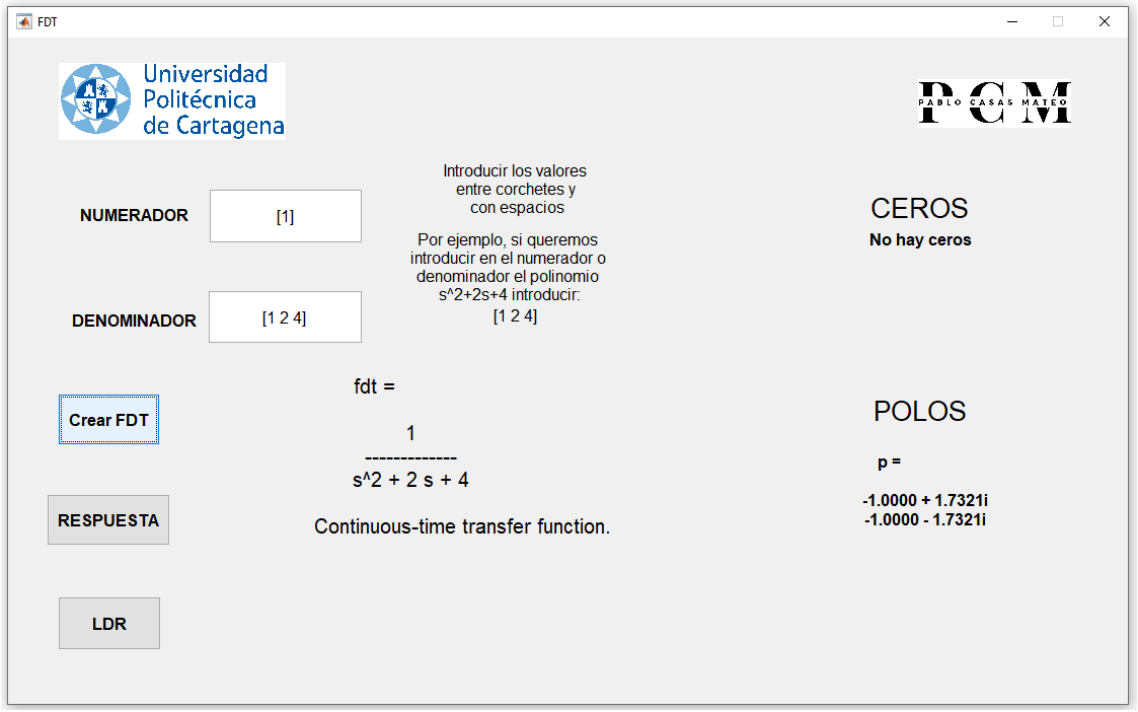

Figura 24: Introducción de datos en primera pestaña 1

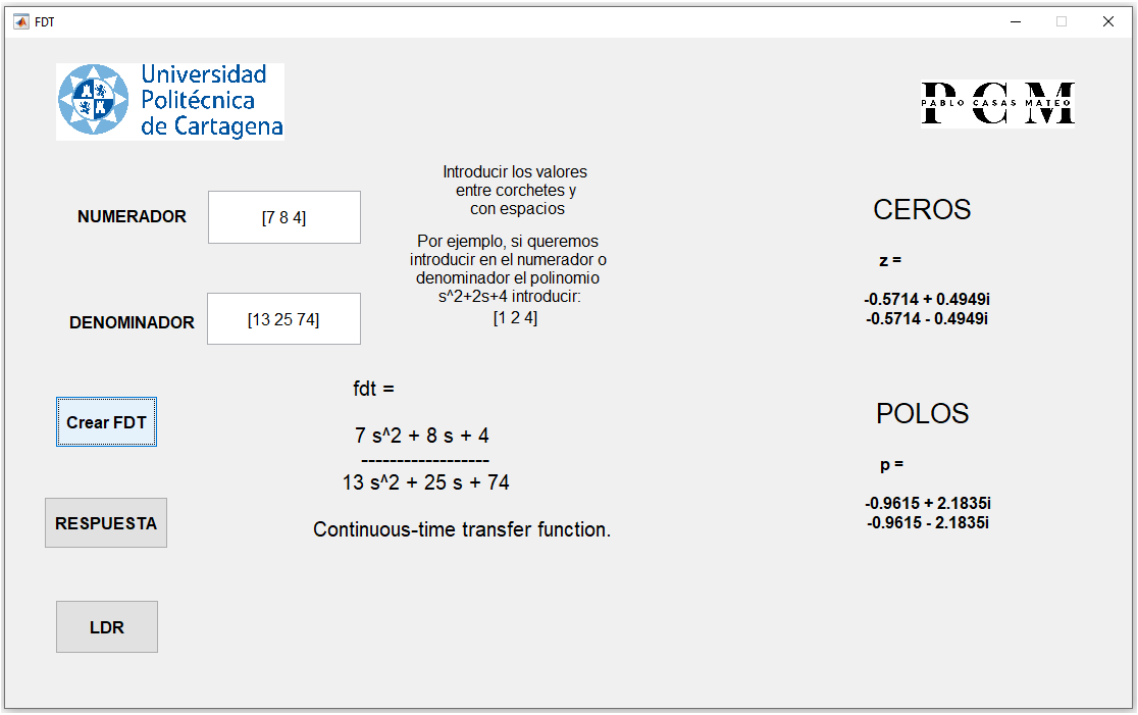

Figura 25: Introducción de datos en primera pestaña 2

Como se puede apreciar en las figuras 24 y 25, si se siguen los comentarios de la pantalla, la introducción de datos es sencilla y funciona correctamente. Ahora veremos lo que pasa si no se siguen estas indicaciones.

Figura 26: Introducción de datos en primera pestaña 3

Figura 27: Introducción de datos en primera pestaña 4

Cuando en los cuadros de introducción de datos se introducen letras (figura 26), el programa muestra un mensaje de error diciendo que los datos introducidos son incorrectos. Lo mismo sucede cuando no se introducen los valores no numéricos en general, sean solo letras o cualquier otro tipo de caracter como muestra la figura 27.

Respuesta

Habiendo introducido los valores en esta pestaña de manera correcta, y tomando como primer ejemplo un sistema de primer orden, pasamos a la pestaña de RESPUESTA, donde encontraremos la evolución temporal de la respuesta ante dos entradas distintas.

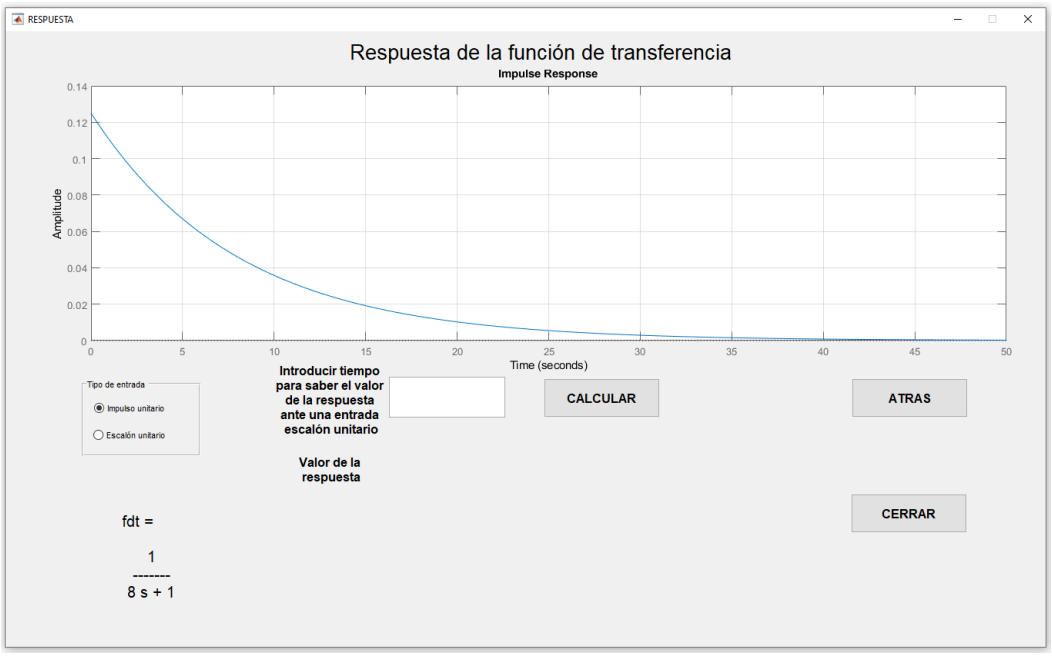

Figura 28: Respuesta ante impulso unitario fdt primer grado

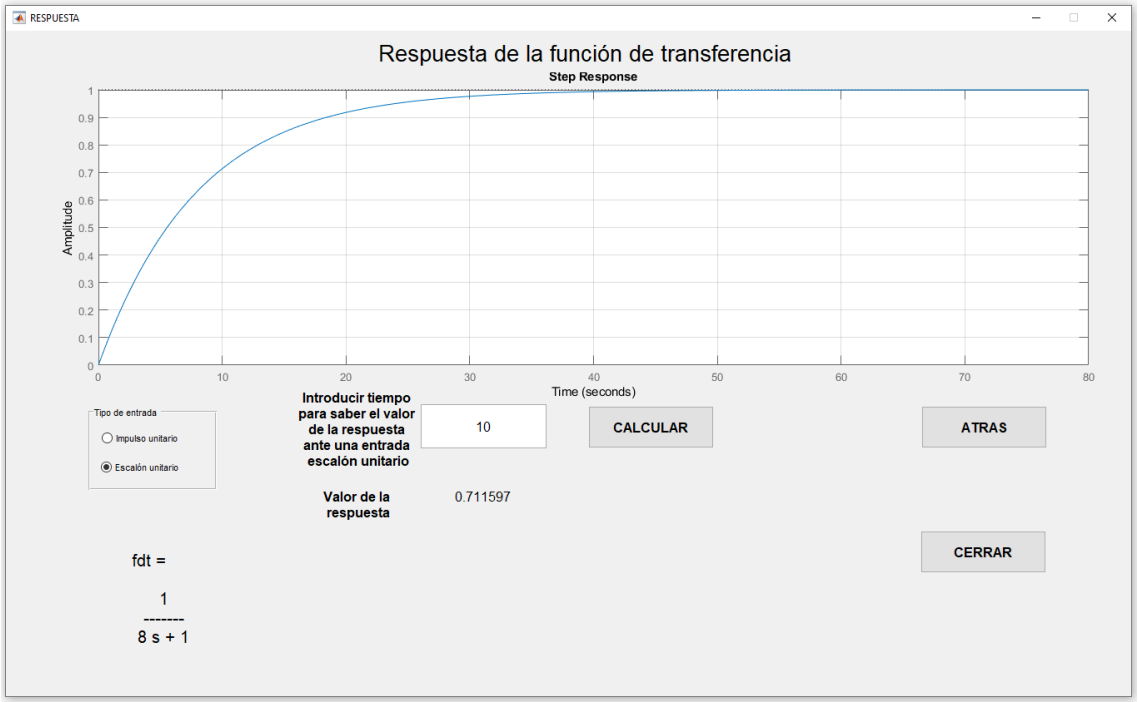

Figura 29: Respuesta ante escalón unitario fdt primer grado y t=10s

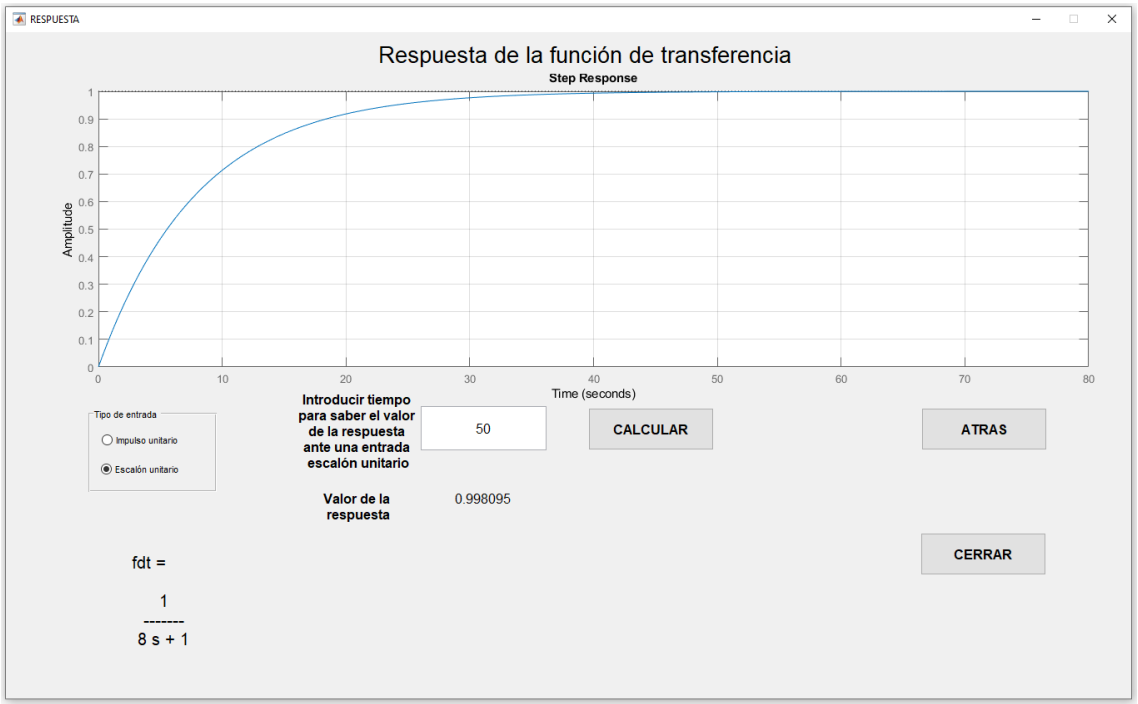

Figura 30: Respuesta ante escalón unitario fdt primer grado y t=50s

Figura 31: Respuesta ante escalón unitario fdt primer grado y t no numérico

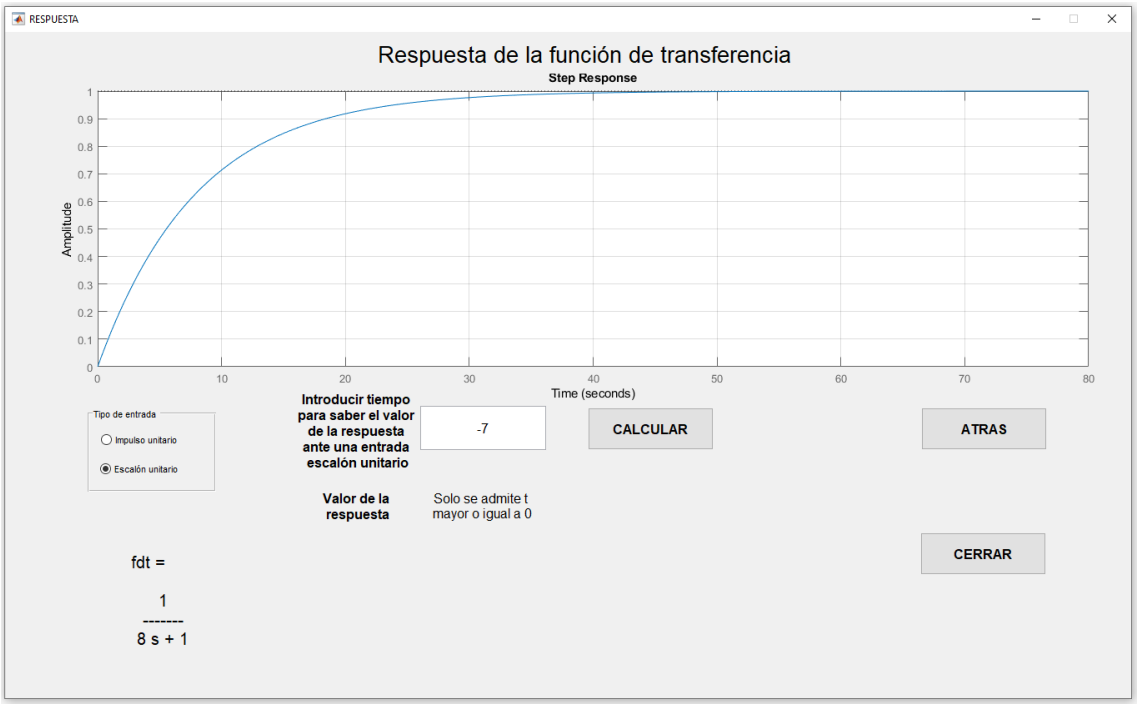

Figura 32: Respuesta ante escalón unitario fdt primer grado y t negativo

En las figuras 28, 29 y 30 vemos un correcto funcionamiento del sistema. Cuando seleccionamos la entrada en forma de impulso unitario, se nos representa en la gráfica la correspondiente evolución temporal, y vemos que cuando seleccionamos la entrada en forma de escalón unitario cambia la gráfica para representar la nueva respuesta.

Con la respuesta ante una entrada de escalón unitario podemos comprobar los valores para cada instante de tiempo t que queramos estudiar. En las figuras 29 y 30 se puede observar que los valores devueltos por el programa se ajustan perfectamente a los valores de la gráfica para dichos instantes de tiempo.

Después, podemos observar en la figura 31 que, al introducir un valor no numérico, el programa nos muestra un mensaje diciendo que solo se aceptan valores numéricos para la variable t. Por otro lado, si introducimos un valor de tiempo negativo como se puede observar en la figura 32, el programa nos devuelve un mensaje diciendo que solo acepta valores positivos para el tiempo.

Por último, en cuanto al rango del eje temporal, se ajusta de manera automática dependiendo del sistema con el que trabajemos. En este ejemplo se ajusta hasta un valor de 80 segundos, y es hasta aproximadamente este valor que podemos introducir en el programa para saber el valor exacto de la función, aunque suele estabilizarse antes. Por otro lado, en el último ejemplo veremos una fdt inestable, y se podrá observar que la escala es mucho mayor, tanto la del eje temporal como la del eje de ordenadas. Se explicará más detenidamente cuando lleguemos a dicho ejemplo.

Continuamos con un sistema de 2º orden subamortiguado.

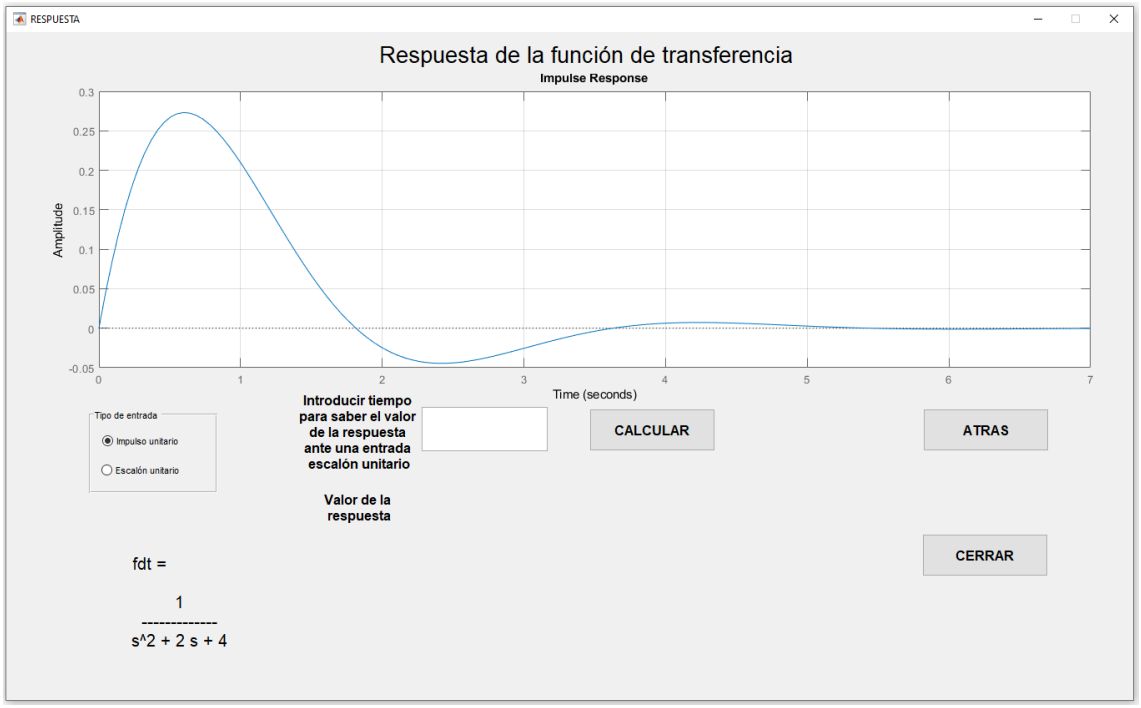

Figura 33: Respuesta ante impulso unitario sistema segundo grado subamortiguado

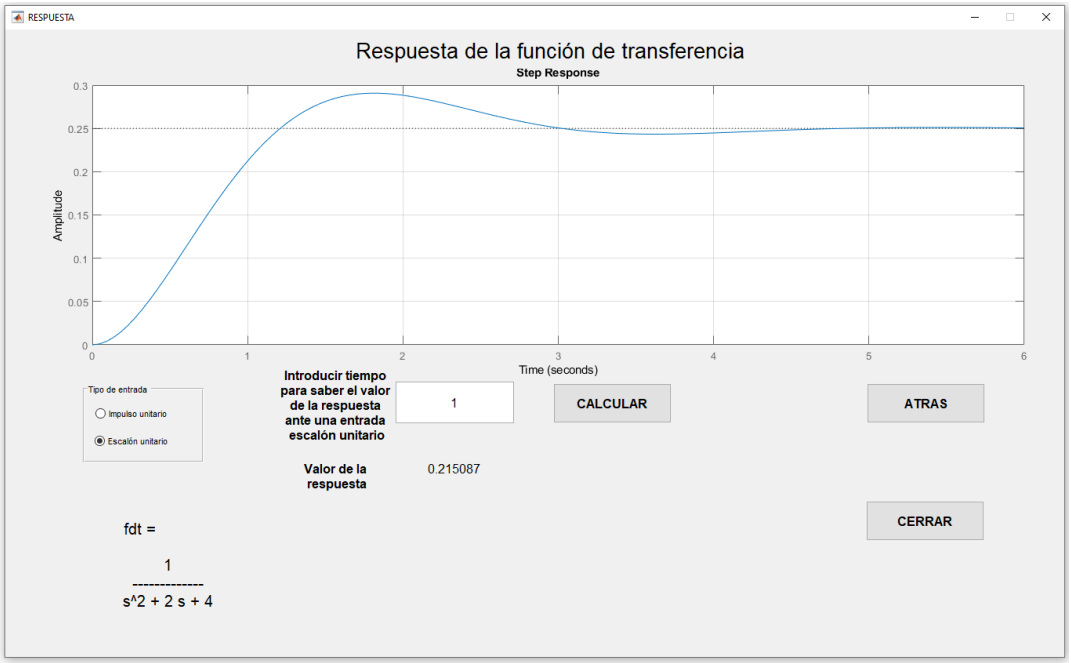

Figura 34: Respuesta ante escalón unitario sistema segundo grado subamortiguado y t=1s

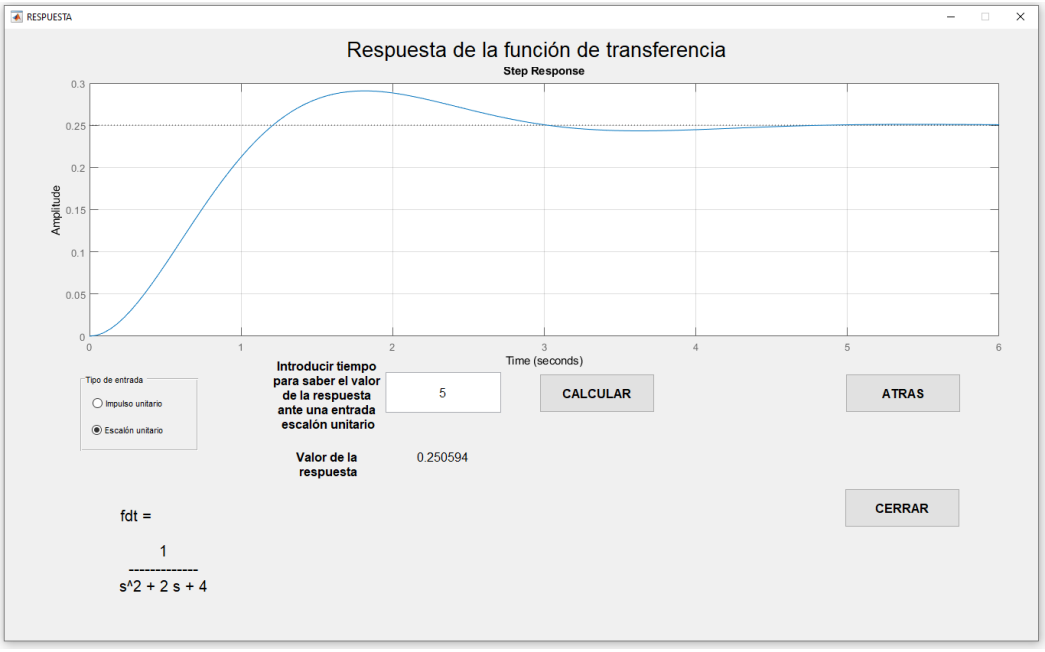

Figura 35: Respuesta ante escalón unitario sistema segundo grado subamortiguado y t=5s

Se comprueba en las figuras 33, 34 y 35 el correcto funcionamiento del programa a la hora de introducir el tipo de entrada deseado y los valores de tiempo para los que queremos conocer el valor preciso de la salida.

Como era de esperar, cuando seleccionamos la entrada en forma de escalón unitario se puede observar la forma característica de la respuesta cuando trabajamos con un sistema subamortiguado, pudiendo apreciar la sobreoscilación para un tiempo aproximado de 1,5 segundos.

Si observamos los valores de los ejes, son distintos a los del ejemplo anterior, y es por la razón que se ha comentado previamente. El sistema tarda ese tiempo aproximadamente en estabilizarse, por lo que no tendría sentido ver una gráfica de 100 segundos de tiempo si 90 de esos segundos se mantiene una línea prácticamente recta.

Seguimos con un sistema de 2º orden sobreamortiguado.

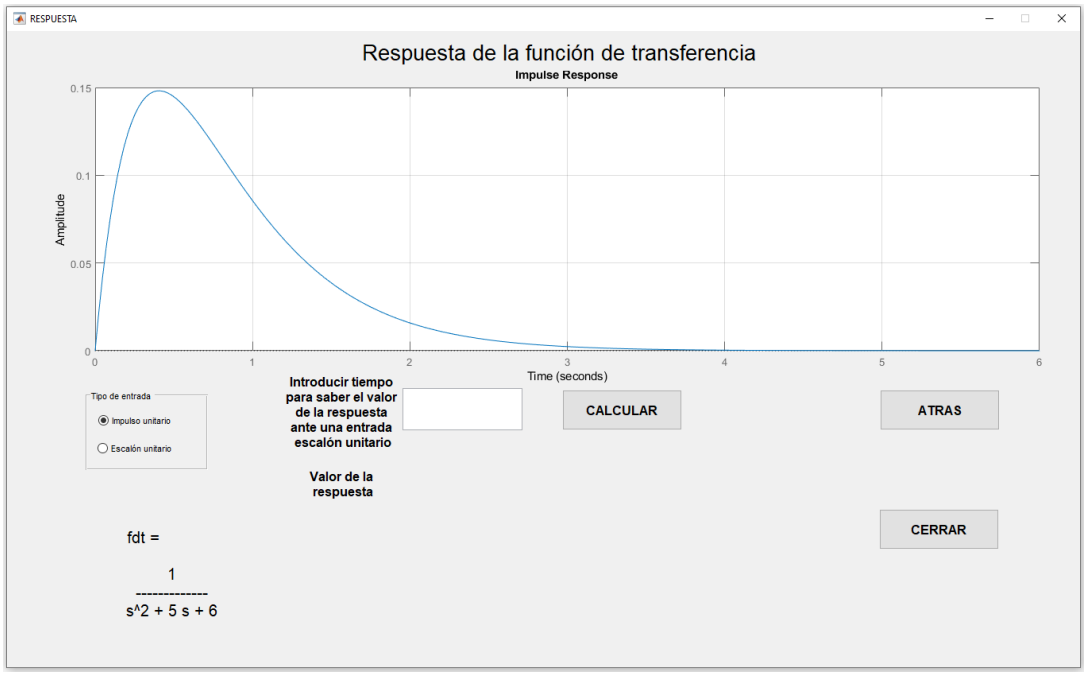

Figura 36: Respuesta ante impulso unitario sistema segundo grado sobreamortiguado

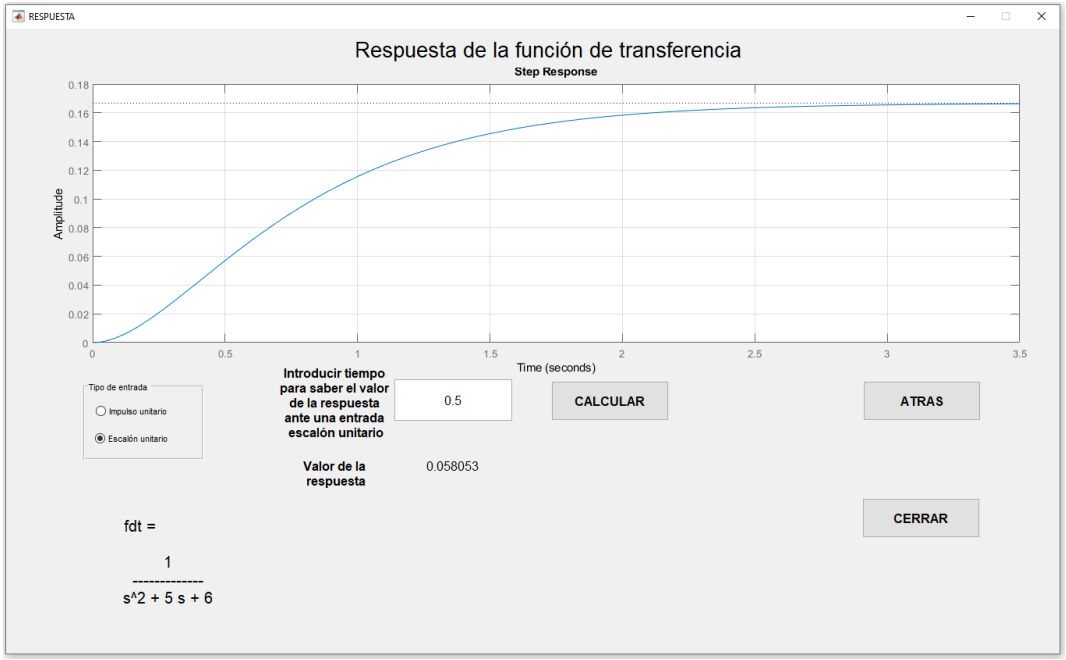

Figura 37: Respuesta ante escalón unitario sistema segundo grado sobreamortiguado y t=0.5s

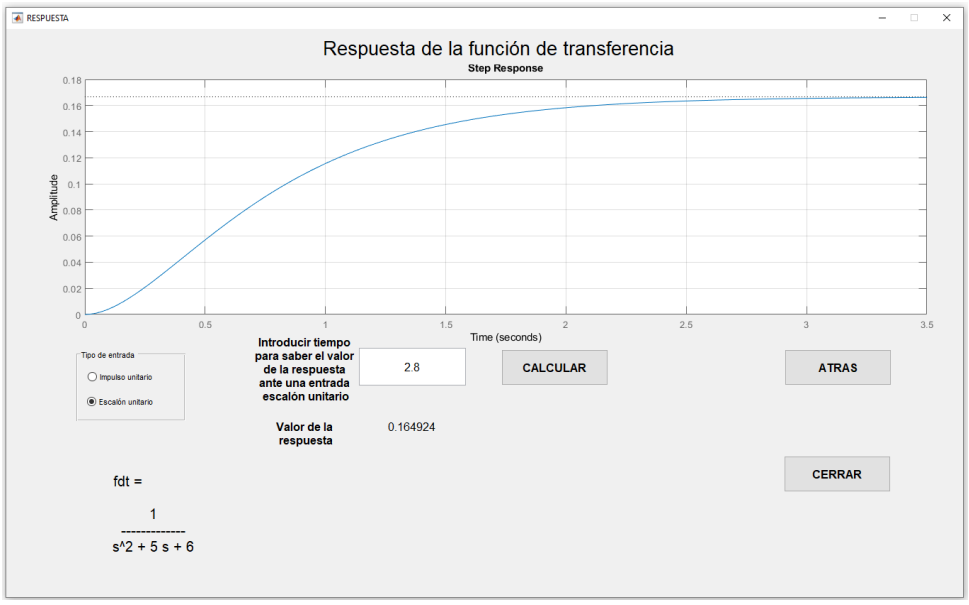

Figura 38: Respuesta ante escalón unitario sistema segundo grado sobreamortiguado y t=2.8s

Al igual que en los casos anteriores, se comprueba en las figuras 36, 37 y 38 el correcto funcionamiento del programa.

Se vuelve a observar de forma clara que la evolución temporal de la respuesta cuando aplicamos una entrada en forma de escalón unitario es como esperábamos, una curva que se acerca infinitamente a un valor sin llegar a este. Se hacen pruebas para dos valores de tiempo distintos, y se verifica que los valores que nos devuelve el programa corresponden a los que podemos ver en la gráfica.

Procedemos ahora a estudiar un sistema de 2º orden con ceros y polos adicionales.

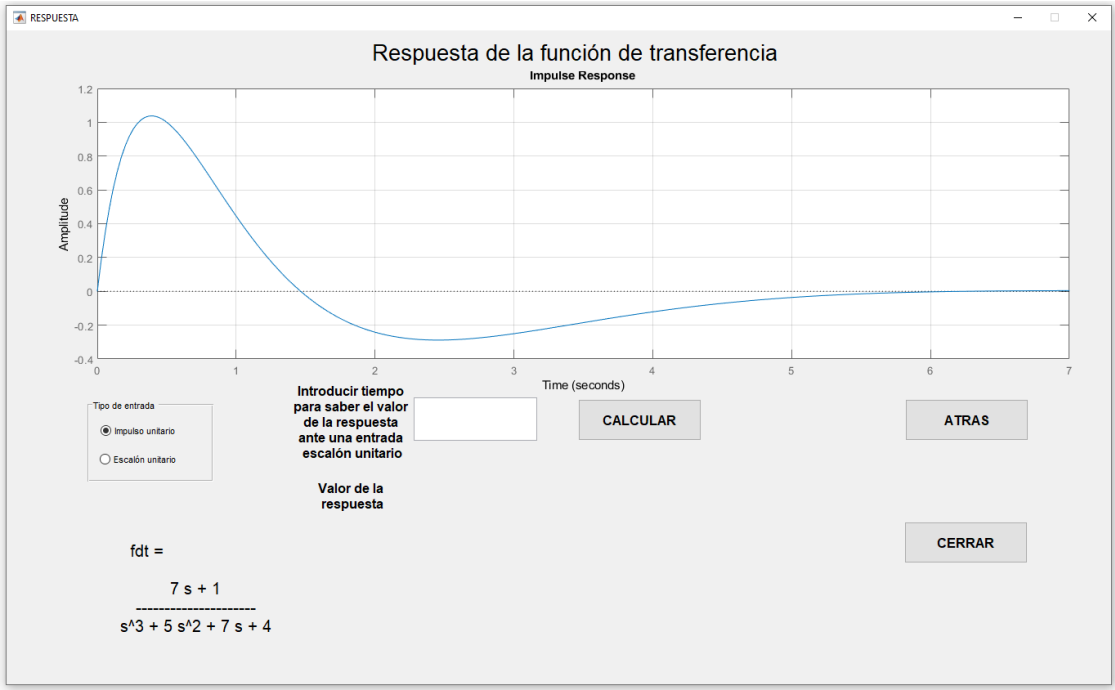

Figura 39: Respuesta ante impulso unitario sistema segundo grado con cero y polo adicional

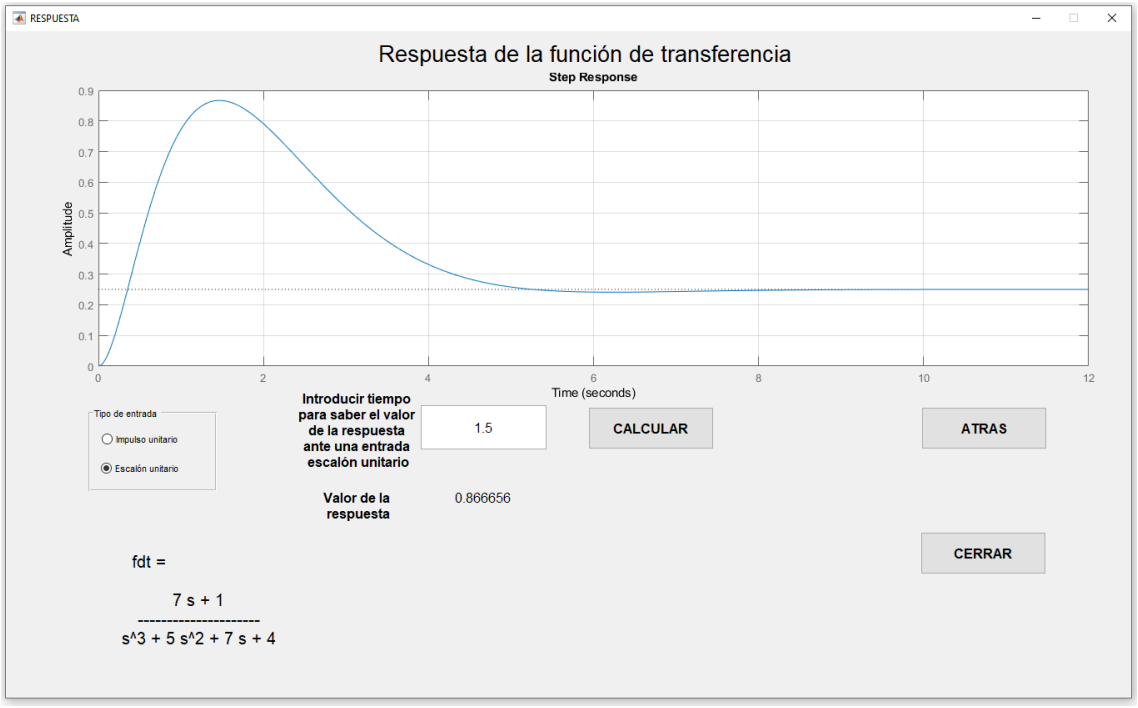

Figura 40: Respuesta ante escalón unitario sistema segundo grado con cero y polo adicional y t=1.5s

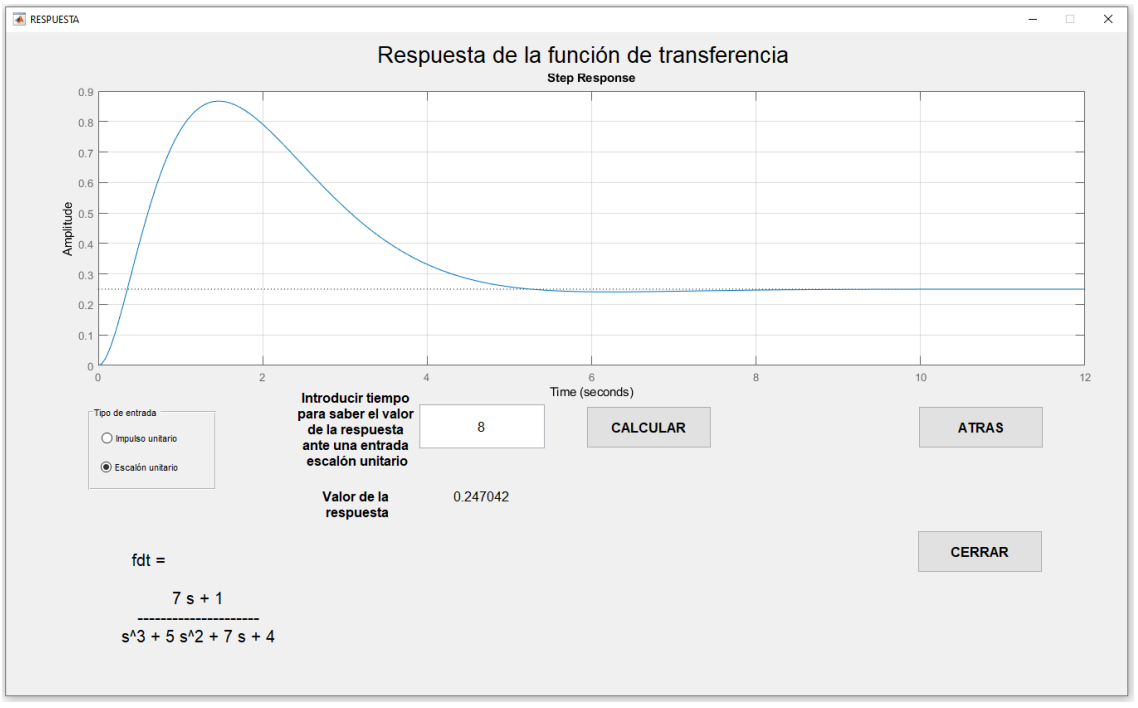

Figura 41: Respuesta ante escalón unitario sistema segundo grado con cero y polo adicional y t=8s

En este ejemplo mostrado en las figuras 39, 40 y 41 se puede ver el efecto de los polos y ceros en la respuesta del sistema, es que, ante su presencia, la sobreoscilación y el tiempo de estabilización de la respuesta aumentan. Podemos ver que el valor de la sobreoscilación en el ejemplo del sistema subamortiguado sobrepasaba un poco el valor en torno al cual se estabilizaría, obteniendo un valor de sobreoscilación de 0,2 aproximadamente; por el contrario, en este ejemplo tenemos un valor de sobreoscilación de 240% aproximadamente, por lo que

notamos un claro aumento. El efecto de este cero y polo adicional en el tiempo de estabilización es menor, pero seguimos notando que tarda más tiempo en llegar al régimen permanente que en el caso anterior.

Por último, veremos un caso de sistema inestable, para ver cómo se comporta el programa.

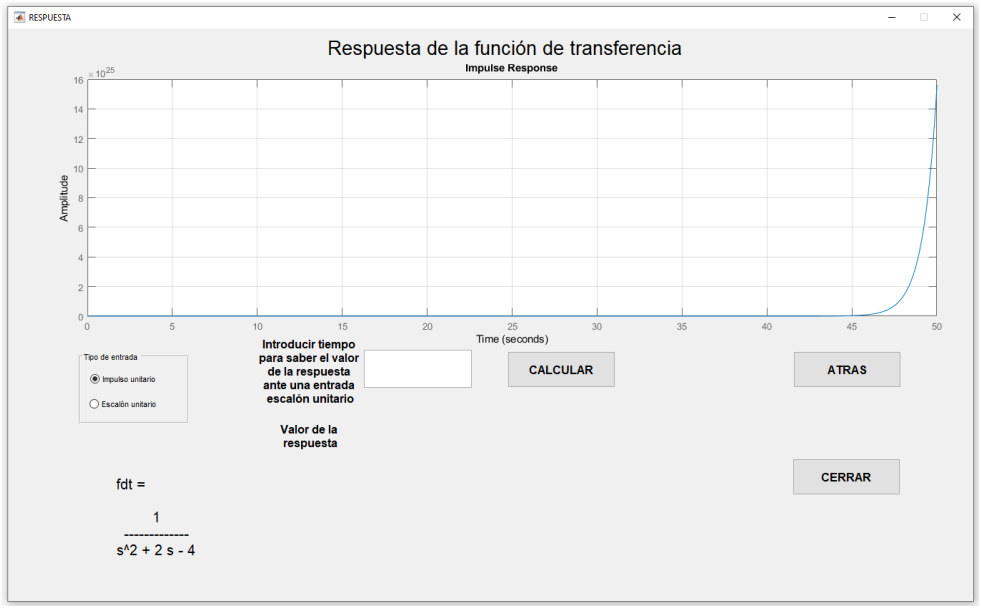

Figura 42: Respuesta ante impulso unitario sistema inestable

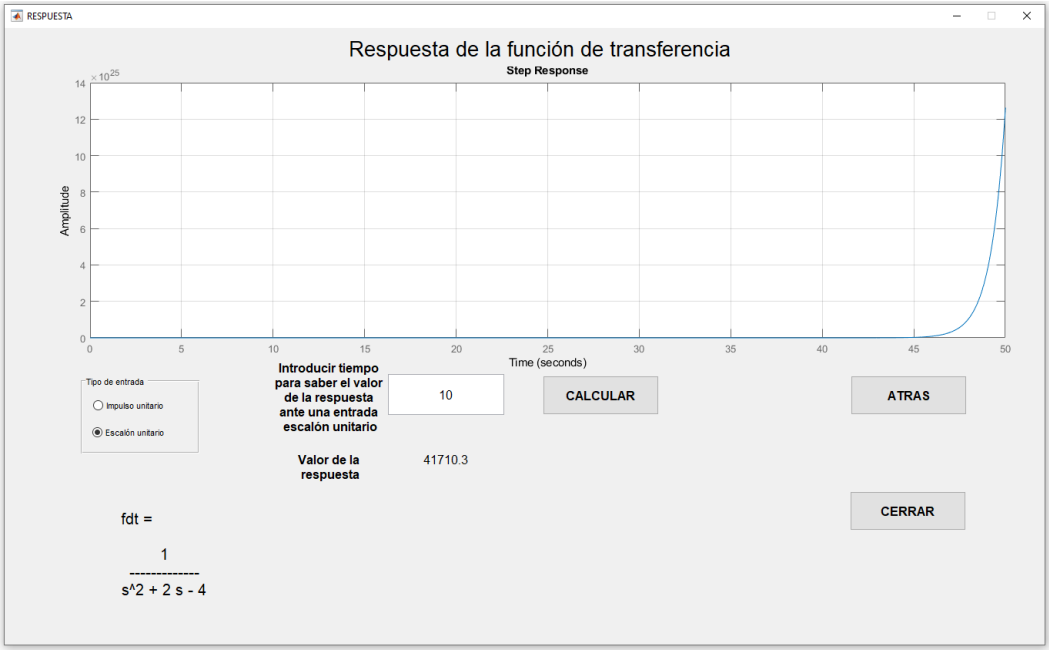

Figura 43: Respuesta ante escalón unitario sistema inestable y t=10s

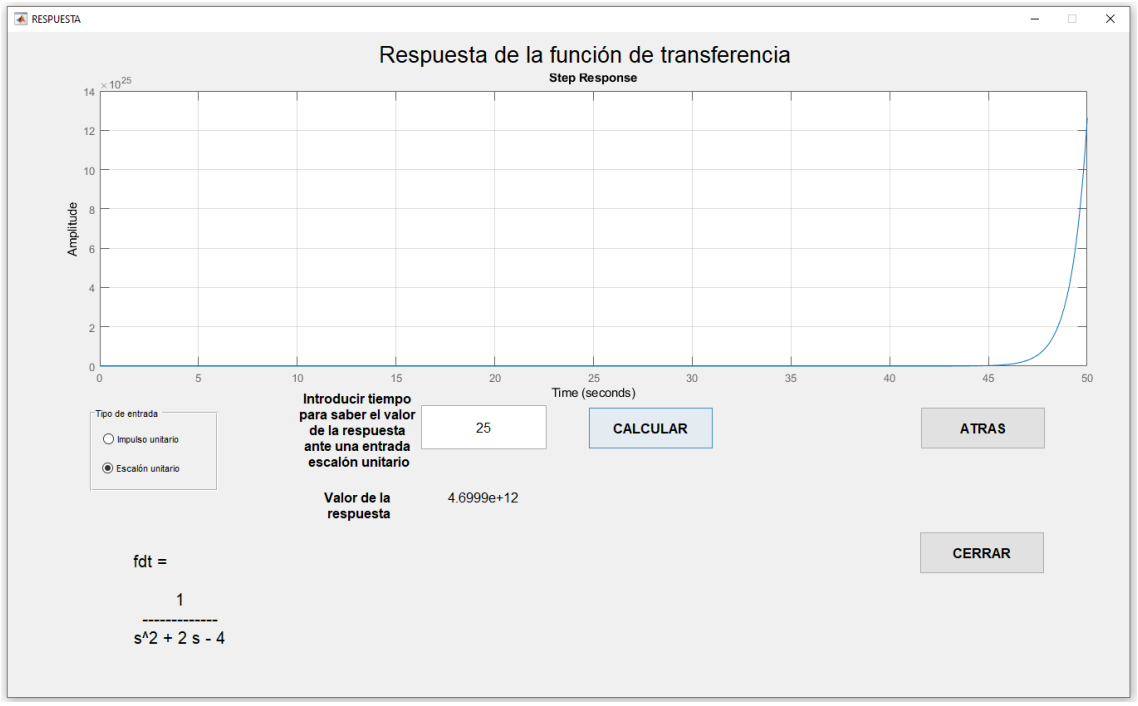

Figura 44: Respuesta ante escalón unitario sistema inestable y t=25s

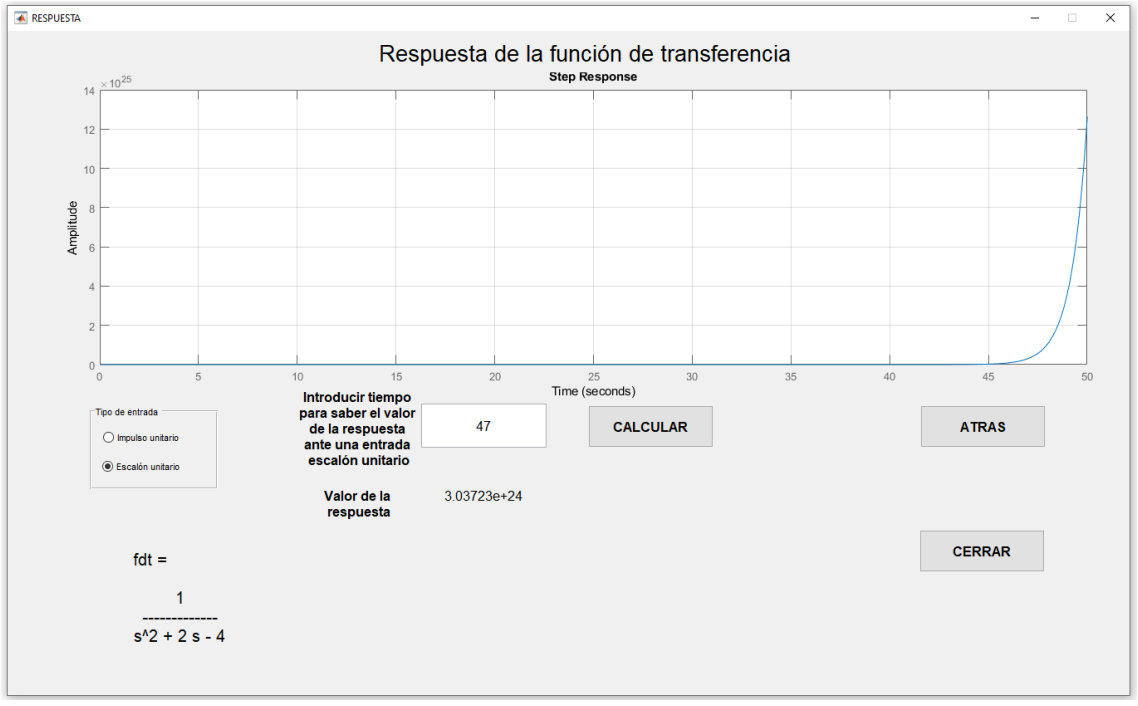

Figura 45: Respuesta ante escalón unitario sistema inestable y t=47s

En este último ejemplo de la pestaña de respuesta nos encontramos con un sistema inestable, pudiéndolo identificar fácilmente porque, cuanto mayor es el tiempo, no llega a estabilizarse en ningún valor, sino que se dispara al infinito. Lo primero que vemos en la figura 42 es la salida ante una entrada en forma de impulso unitario. No se aprecia diferencia entre los dos tipos de entrada, y esto es debido a que los valores de la salida son tan grandes para tiempos superiores a 45 segundos que no podemos apreciar el desarrollo de la salida antes de este tiempo.

Al ver las figuras 43 y 44 vemos que el programa nos devuelve, para t=10s y t=25s valores de señal de respuesta, pero al observar la gráfica parece que esta línea se sitúa en el valor 0. Esto se puede explicar si nos fijamos en los valores de los ejes, ya que vemos que el eje de ordenadas está multiplicado $x10^{25}$, y como los valores de respuesta que tenemos para esos valores de tiempo son extremadamente pequeños en comparación con la escala usada, su representación gráfica estará próxima a 0 y no será posible distinguirlos, aunque lo que suceda en realidad es que existe una oscilación continua que crece en amplitud a lo largo del tiempo. Por último, en la figura 45 si se puede apreciar el aumento de la curva para t=47s, y que el valor de la salida se aproxima al que nos devuelve el programa.

Por otro lado, como se comentó en el primer ejemplo, el rango de valores mostrado lo establece de manera automática el programa, y en este caso se aprecia que, como escoge un valor de escala de salida ampliado a $x10^{25}$, cuando en el tiempo se llega a ese rango de valores se acaba la gráfica.

Lugar de las raíces

Para finalizar este capítulo, se realizarán pruebas en la pestaña del lugar de las raíces con varios sistemas distintos.

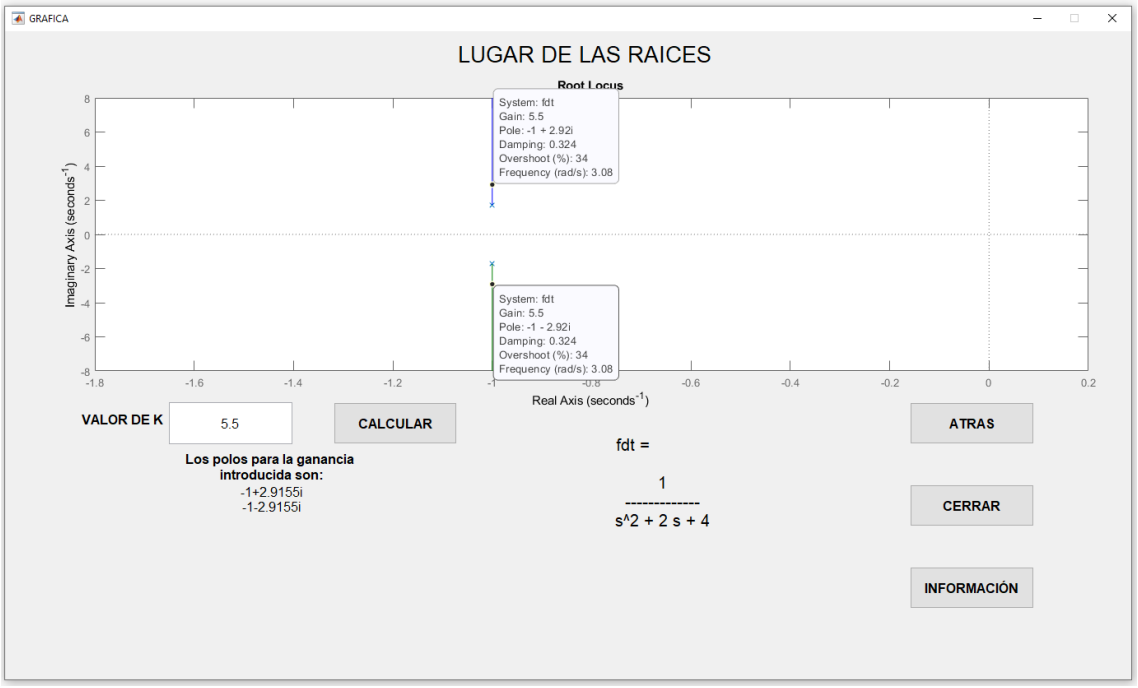

Figura 46: Lugar de las raíces primer sistema

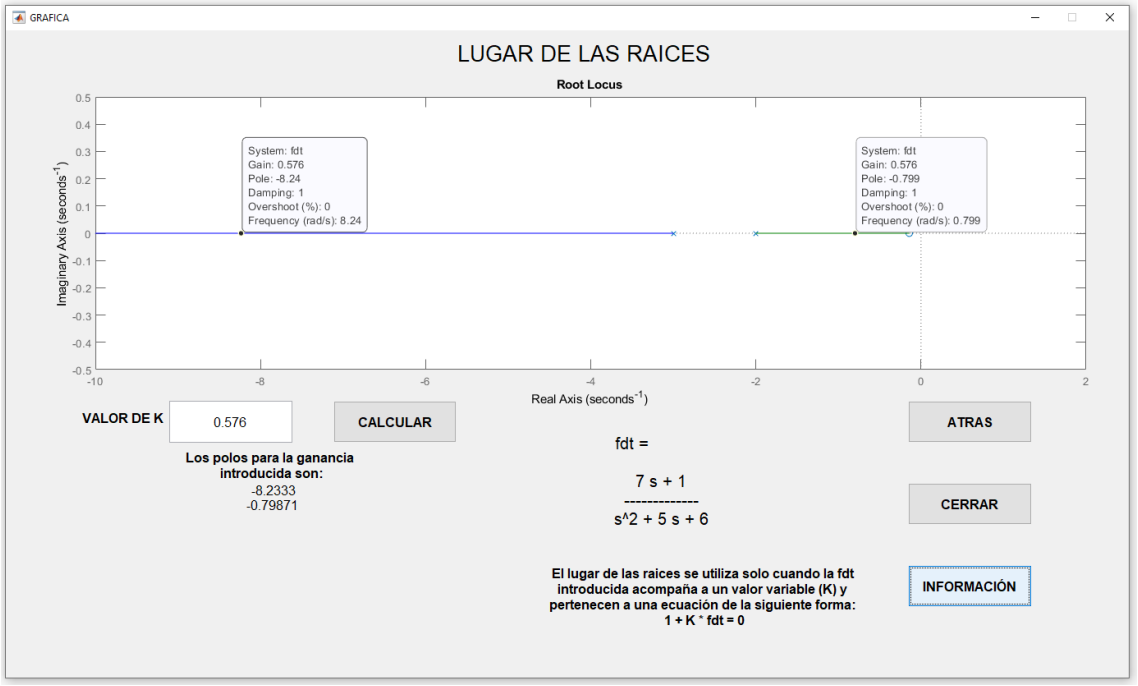

Figura 47: Lugar de las raíces segundo sistema y botón información

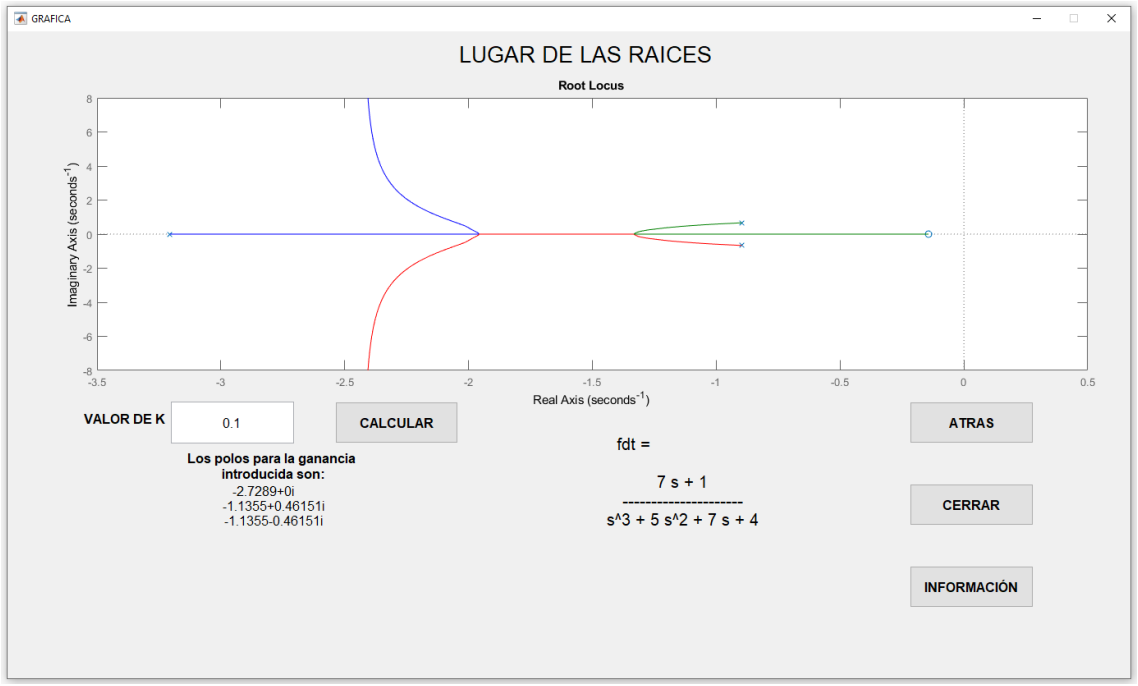

Figura 48: Lugar de las raíces tercer sistema

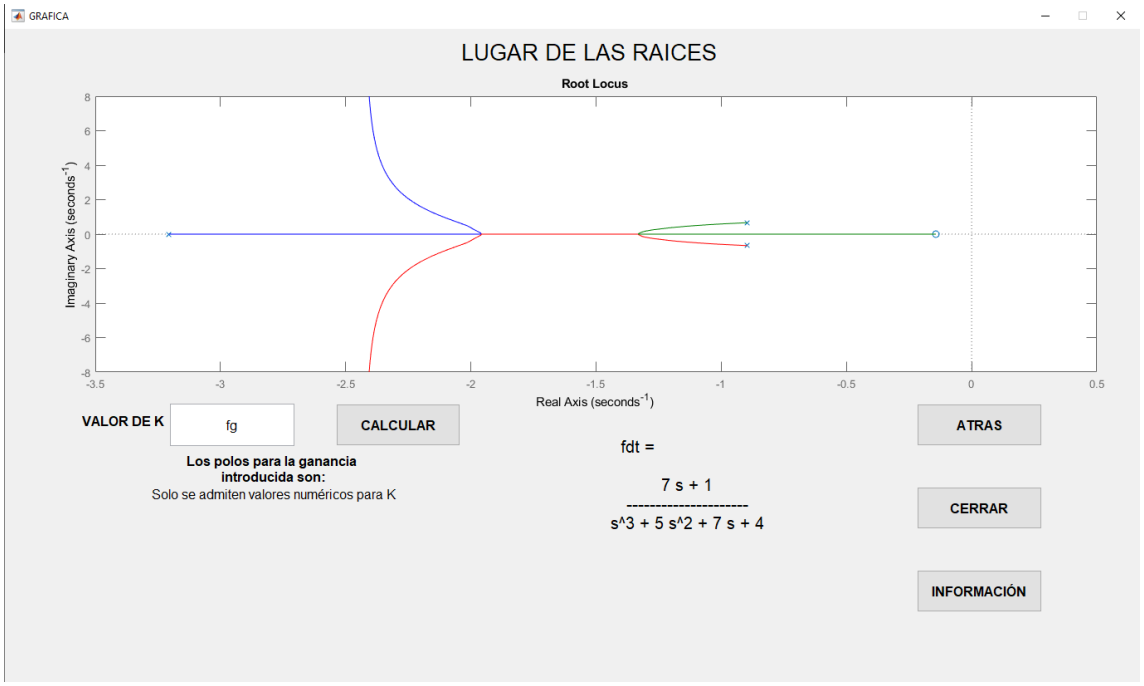

Figura 49: Lugar de las raíces tercer sistema introduciendo valores no numéricos

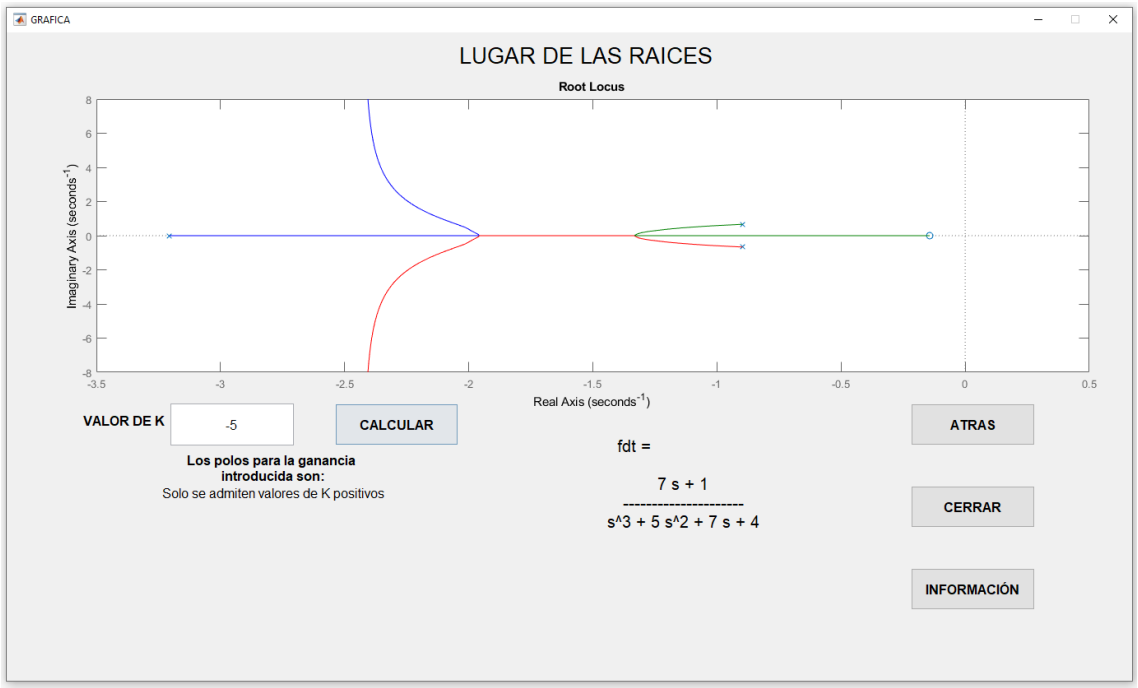

Figura 50: Lugar de las raíces tercer sistema introduciendo valores negativos

Nos encontramos con varias situaciones distintas:

Primero, en la figura 46 encontramos un sistema en el que sus polos son complejos conjugados, por lo que su lugar de las raíces tendrá una asíntota vertical.

A continuación, en la figura 47 tenemos dos polos reales negativos y un cero, por lo que uno de los polos irá desde su valor inicial hasta el cero, y el otro aumentará de manera infinita conforme aumente el valor de K teniendo una asíntota horizontal. En esta figura también hemos

comprobado que el botón de información funciona correctamente, mostrando una explicación de cuándo se utiliza la herramienta del lugar de las raíces.

La figura 48 nos muestra un sistema de segundo orden con un cero y un polo adicional. Como tenemos un cero, uno de los polos trazará su trayectoria desde su valor inicial hasta el cero, y como restan 2 polos, estos aumentarán su valor teniendo una asíntota vertical.

Por último, en la figura 49 nos encontramos que el valor de K introducido no es numérico, por lo que, como pasaba en las otras pestañas, el programa nos muestra un mensaje diciendo que solo se admiten valores numéricos para K. En la figura 50 observamos que cuando se introduce un valor de K negativo, el programa nos indica que solo se admiten valores de K positivos.

Concluye aquí el capítulo de pruebas, y podemos verificar que el programa funciona de manera correcta, pues no tiene errores y realiza todas las funciones programadas.

CAPÍTULO 8: CONCLUSIONES Y TRABAJOS FUTUROS

Para terminar, se expondrán las conclusiones a las que hemos llegado y se comentarán los posibles trabajos futuros que pueden surgir a partir de este proyecto.

Conclusiones

Se hará un resumen de cada una de las partes en las que está dividido este proyecto, y a qué conclusiones llegamos.

En el segundo capítulo se ha realiza una introducción al concepto de GUI y una comparación entre las dos herramientas que tiene MATLAB para crear aplicaciones: GUIDE y APP DESIGNER. Al ver los puntos positivos y negativos de cada una de estas aplicaciones, se puede concluir que APP DESIGNER es una mejor opción, pero debido a factores clave como la disponibilidad del alumno a la hora de realizar el trabajo, la incompatibilidad de realizar el proyecto desplazándose a las instalaciones de la universidad, y que en un principio se pretendía realizar una GUI que complementara otra realizada previamente en GUIDE, pero finalmente se ha optado por el diseño de una GUI orientada a tareas de enseñanza y aprendizaje, y es por eso que se acaba tomando la decisión de usar GUIDE.

En el tercer capítulo, cuando ya se ha concluido que se utilizará GUIDE como herramienta de trabajo, se ha realizado un análisis más profundo de esta herramienta para conocerla mejor, conocer los componentes que contiene y saber utilizarla.

A continuación, en el capítulo cuarto se procede a la explicación detallada del funcionamiento de cada uno de los objetos que encontramos dentro de la herramienta GUIDE, se explica cómo se utilizan, qué variantes tiene cada objeto, y cómo se programan. Está claro que esta explicación no es completa al 100% puesto que cada persona programa de una forma, y se pueden usar distintas funciones para realizar la misma acción, y en este proyecto se ha explicado sólo con una de estas variantes.

Durante el quinto capítulo, nos centramos en el análisis de la asignatura de Regulación Automática, y se han recopilado las ideas clave que puedan ser desarrolladas en una GUI como elementos de utilidad en la enseñanza y aprendizaje de la asignatura. Se hace un estudio tema por tema, y se recogen ideas como:

- Realizar una interfaz que calcule los polos, ceros, respuesta y lugar de las raíces de una función de transferencia dada.
- Posibilidad de insertar la función de transferencia de distintas maneras: introduciendo numerador y denominador, introduciendo los polos y ceros o introduciendo los valores característicos de la misma.
- Inserción de unos sistemas físicos predeterminados para mayor agilidad, por ejemplo, un sistema eléctrico, un sistema mecánico, o un sistema de intercambio de materia.
- Diseño de un regulador PID para el control de la salida del sistema

En el sexto capítulo se ha desarrollado nuestro programa, en el que se han acabado introduciendo las 3 funciones básicas a la hora de trabajar con una función de transferencia, que son la introducción de la misma, con su posterior cálculo de polos y ceros, la respuesta del sistema ante dos tipos de entrada distintas, y la representación del lugar de las raíces. Se han introducido estos aspectos pues son las herramientas esenciales a la hora de trabajar con un sistema, esté totalmente definido o dependa de un parámetro variable.

Por último, en el séptimo capítulo se han realizado distintas pruebas dentro del programa, y se ha verificado que este funciona de manera correcta y que no contiene errores.

En definitiva, se ha creado una guía de cómo crear una GUI desde cero, que puede ser fácilmente entendida tanto por gente del ámbito de la ingeniería como por aquellos que entienden menos, pues esta guía ha sido bien detallada, y se ha creado un programa sin errores como aplicación de cómo crear una GUI desde cero y que, a su vez, es funcional y podrá ser utilizada por alumnos.

Valoración personal

Este trabajo ha sido muy beneficioso en mi desarrollo personal. Además de aprender nuevas formas de programación, a usar mejor el programa MATLAB, o descubrir un entorno de creación de GUIs, este trabajo me ha dado la oportunidad de hacer una investigación por mi cuenta, prácticamente solo, y a buscar la forma de desenvolverme cuando han surgido problemas.

Trabajos futuros

Como hemos visto en el capítulo 5, la asignatura de Regulación Automática nos permite abordar y trabajar con los problemas desde distintos puntos de vista, partiendo de distintos comienzos o, simplemente, trabajando con los datos en distintas expresiones.

En próximos trabajos, se pueden introducir nuevas funciones a esta GUI, como la posibilidad de introducir la función de transferencia con la que se trabaje de distintas maneras (insertando los polos y ceros de la misma o introduciendo los valores característicos del transitorio), añadir sistemas físicos predeterminados, como podrían ser los sistemas eléctricos, sistemas mecánicos o sistemas de intercambio de materia, insertar PID para controlar la salida de nuestro sistema, o la posibilidad de que en la respuesta del sistema nos aparezcan los datos característicos del transitorio por ejemplo, en un sistema subamortiguado, que nos devolviese el tiempo de pico, el valor de pico de la salida, el tiempo de duración del transitorio o la sobreoscilación.

Por otro lado, como ya se vio que existe APP DESIGNER, una herramienta más completa y moderna en MATLAB, otra línea de trabajos futuros podría ser la extensión de este proyecto con esta herramienta, ya que podrá suponer nuevas y posibles mejoras.

ANEXO

Código FDT

```
function varargout = FDT(varargin)
% FDT MATLAB code for FDT.fig
%      FDT, by itself, creates a new FDT or raises the existing
%      singleton*.
%
%      H = FDT returns the handle to a new FDT or the handle to
%      the existing singleton*.
%
%      FDT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in FDT.M with the given input
arguments.
%
%      FDT('Property','Value',...) creates a new FDT or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before FDT_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to FDT_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help FDT

% Last Modified by GUIDE v2.5 27-Apr-2021 10:35:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @FDT_OpeningFcn, ...
                   'gui_OutputFcn',  @FDT_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before FDT is made visible.
function FDT_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to FDT (see VARARGIN)

%Insertar imagen logo upct

axes(handles.axes4);
bkgrnd=imread('logoupct.jpg');
imshow(bkgrnd);

%Insertar imagen logo PCM

axes(handles.axes5);
bkgrnd=imread('MiLogo.png');
imshow(bkgrnd);

%Recordar que para que se carguen las imágenes es necesario que estas
estén
%en la misma carpeta que los archivos .m y .fig del programa

% Choose default command line output for FDT
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes FDT wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = FDT_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in Boton_crear_fdt.
function Boton_crear_fdt_Callback(hObject, eventdata, handles)

%Vaciamos los campos de solucion polos y ceros

set(handles.solucion, 'string', '')
set(handles.polos,'string', '')
set(handles.ceros,'string', '')

%Variables globales para poder usarlas en todas las pestañas
global fdt fdtstring pstring zstring num numerador den denominador

%Recoger información de los edit_text y los introduce en las
%variables num (numerador) y den (denominador)
num=get(handles.Numerador_editable,'string')
den=get(handles.Denominador_editable,'string')
```

```
%Evaluamos las cadenas de texto num y den y se convierten en
%formato numérico
numerador=str2num(num)
denominador=str2num(den)

if (isempty(numerador) || isempty(denominador))
    %Si los datos están mal introducidos, nos dará error
    set(handles.solucion, 'string', 'Los datos se han introducido mal,
por favor, siga las instrucciones arriba indicadas')
else

%Creamos la función de transferencia
fdt=tf(numerador,denominador)

%Convierto la función de transferencia para poder plasmarla
%en la pantalla
fdtstring=evalc('fdt')

%Obtenemos los polos y ceros de la función de transferencia
%y los convertimos para poder plasmarlos en la pantalla
p=pole(fdt)
pstring=evalc('p')
z=zero(fdt)
zstring=evalc('z')

%Envio los resultados a los cuadros de texto estáticos en blanco
set(handles.solucion,'string', fdtstring)

if isempty(p)==1
    set(handles.polos,'string', 'No hay polos')
else set(handles.polos,'string', pstring)
end


if isempty(z)==1
    set(handles.ceros,'string', 'No hay ceros')
else set(handles.ceros,'string', zstring)
end
end



function Numerador_editable_Callback(hObject, eventdata, handles)
% hObject    handle to Numerador_editable (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Hints:get(hObject,'String') returns contents of Numerador_editable as
text
%        str2double(get(hObject,'String')) returns contents of
%Numerador_editable as a double


% --- Executes during object creation, after setting all properties.
function Numerador_editable_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Numerador_editable (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Denominador_editable_Callback(hObject, eventdata, handles)
% hObject    handle to Denominador_editable (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Hints: get(hObject,'String') returns contents of Denominador_editable
% as text
%       str2double(get(hObject,'String')) returns contents of
% Denominador_editable as a double


% --- Executes during object creation, after setting all properties.
function Denominador_editable_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Denominador_editable (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in ldr.
function ldr_Callback(hObject, eventdata, handles)
GRAFICA;
close FDT;
% hObject    handle to ldr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in respuesta.
function respuesta_Callback(hObject, eventdata, handles)
RESPUESTA;
close FDT;
% hObject    handle to respuesta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Código RESPUESTA

```
function varargout = RESPUESTA(varargin)
% RESPUESTA MATLAB code for RESPUESTA.fig
%      RESPUESTA, by itself, creates a new RESPUESTA or raises the
existing
%      singleton*.
%
%      H = RESPUESTA returns the handle to a new RESPUESTA or the
handle to
%      the existing singleton*.
%
%      RESPUESTA('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in RESPUESTA.M with the given input
arguments.
%
%      RESPUESTA('Property','Value',...) creates a new RESPUESTA or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before RESPUESTA_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to RESPUESTA_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help RESPUESTA

% Last Modified by GUIDE v2.5 27-Apr-2021 11:15:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RESPUESTA_OpeningFcn, ...
                   'gui_OutputFcn',  @RESPUESTA_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before RESPUESTA is made visible.
function RESPUESTA_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
%Variables globales
global fdt fdtstring pstring zstring

set(handles.impulso,'Value',0);
set(handles.escalon,'Value',0);

set(handles.fdtenrespuesta,'string', fdtstring);

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RESPUESTA (see VARARGIN)

% Choose default command line output for RESPUESTA
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes RESPUESTA wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = RESPUESTA_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in atras.
function atras_Callback(hObject, eventdata, handles)
FDT;
close RESPUESTA;
uiwait;
% hObject    handle to atras (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in cerrar.
function cerrar_Callback(hObject, eventdata, handles)
close RESPUESTA;
% hObject    handle to cerrar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function tiempo_Callback(hObject, eventdata, handles)
% hObject    handle to tiempo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of tiempo as text
%        str2double(get(hObject,'String')) returns contents of tiempo
as a double


% --- Executes during object creation, after setting all properties.
function tiempo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tiempo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in calcular.
function calcular_Callback(hObject, eventdata, handles)
global fdt
T=str2num(get(handles.tiempo,'string'))
[y,t]=step(fdt)

if (isempty(T))
    set(handles.respuesta, 'string', 'Solo se admiten valores
numéricos para t')
else
    if (T>=0)
    [minValue,closestIndex]=min(abs(t-T));
    set(handles.respuesta,'string',y(closestIndex))
    else
    set(handles.respuesta,'string', 'Solo se admite t mayor o igual a
0');
    end

end

% --- Executes when selected object is changed in uibuttongroup1.
function uibuttongroup1_SelectionChangedFcn(hObject, eventdata,
handles)
X=get(hObject, 'Tag')
global fdt
if strcmp(X, 'impulso')
    impulse(fdt); grid on; %creo cuadricula
    axes(handles.axes2);
elseif strcmp(X, 'escalon')
    step(fdt); grid on;
    axes(handles.axes2);
end
% hObject    handle to the selected object in uibuttongroup1
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Código LDR

```
function varargout = GRAFICA(varargin)
% GRAFICA MATLAB code for GRAFICA.fig
%      GRAFICA, by itself, creates a new GRAFICA or raises the
existing
%      singleton*.
%
%      H = GRAFICA returns the handle to a new GRAFICA or the handle
to
%      the existing singleton*.
%
%      GRAFICA('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in GRAFICA.M with the given input
arguments.
%
%      GRAFICA('Property','Value',...) creates a new GRAFICA or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before GRAFICA_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to GRAFICA_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GRAFICA

% Last Modified by GUIDE v2.5 28-Apr-2021 13:47:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GRAFICA_OpeningFcn, ...
                   'gui_OutputFcn',  @GRAFICA_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GRAFICA is made visible.
function GRAFICA_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
%Variables globales
global fdt fdtstring pstring zstring

%Creamos la gráfica del lugar de las raices
rlocus(fdt)

set(handles.fdtenldr,'string', fdtstring)

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GRAFICA (see VARARGIN)

% Choose default command line output for GRAFICA
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GRAFICA wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = GRAFICA_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in atras.
function atras_Callback(hObject, eventdata, handles)
FDT;
close GRAFICA;
uiwait;
% hObject    handle to atras (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in cerrar.
function cerrar_Callback(hObject, eventdata, handles)
close GRAFICA;
% hObject    handle to cerrar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in informacion.
function informacion_Callback(hObject, eventdata, handles)
set(handles.cuadro,'string', 'El lugar de las raices se utiliza solo
cuando la fdt introducida acompaña a un valor variable (K) y
pertenecen a una ecuación de la siguiente forma: 1 + K * fdt = 0')
```

```
% hObject    handle to informacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
informacion.



function k_Callback(hObject, eventdata, handles)
% hObject    handle to k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of k as text
%        str2double(get(hObject,'String')) returns contents of k as a
double


% --- Executes during object creation, after setting all properties.
function k_CreateFcn(hObject, eventdata, handles)
% hObject    handle to k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in calcular.
function calcular_Callback(hObject, eventdata, handles)
global fdt
%Recogemos el valor de la ganancia introducido
K=str2num(get(handles.k,'string'))
%Con Rlocus ya nos introduce la fdt en una ecuación de la forma:
%1+K*fdt=0
%y nos devuelve los valores de los polos y ceros para dicha ganancia
if (isempty(K))
    set(handles.resultado, 'string', 'Solo se admiten valores
numéricos para K')
else
if (K>0)
r=rlocus(fdt,K)
x=num2str(r)
set(handles.resultado, 'string', x);
else
    set(handles.resultado, 'string', 'Solo se admiten valores de K
positivos');
end

end
```

```
% if k<0 mostrar error como que no puedo introducir valores negativos

% hObject    handle to calcular (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: place code in OpeningFcn to populate axes2
```

BIBLIOGRAFÍA

[1] Wikipedia [en línea] [fecha de consulta: 10 mayo 2021] https://es.wikipedia.org/wiki/Regulaci%C3%B3n_autom%C3%A1tica

[2] Wikipedia [en línea] [fecha de consulta: 1 diciembre 2020] https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario#:~:text=La%20interfaz%20gr%C3%A1fica%20de%20usuario,acciones%20disponibles%20en%20la%20interfaz.

[3] Workana.com [en línea] [fecha de consulta: 8 diciembre 2020] https://www.workana.com/i/glosario/que-es-la-interfaz-grafica-de-usuario-gui/

[4] Youtube, canal de Tutoingeniero [en línea] [fecha de consulta: 5 noviembre 2020] https://www.youtube.com/watch?v=xYnifF6U268&t=2s&ab_channel=Tutoingeniero

[5] Mathworks.com [en línea] [fecha de consulta: 8 noviembre 2020] https://es.mathworks.com/products/matlab/app-designer/comparing-guide-and-app-designer.html

[6] Mathworks.com [en línea] [fecha de consulta: 9 noviembre 2020] https://es.mathworks.com/help/matlab/creating_guis/differences-between-app-designer-and-guide.html

[7] Youtube, canal de Tutoingeniero [en línea] [fecha de consulta: 1 diciembre 2020] https://www.youtube.com/watch?v=WaxeyVOY1vw&list=PLSrGuI5Xqm-42rhAO2CU1lxO_24KIvl3u&index=2&ab_channel=Tutoingeniero

[8] Youtube, canal de Tutoingeniero [en línea] [fecha de consulta: 2 diciembre 2020] https://www.youtube.com/watch?v=QV_7h3kec_8&list=PLSrGuI5Xqm-42rhAO2CU1lxO_24KIvl3u&index=3&ab_channel=Tutoingeniero

[9] Mathworks [en línea] [fecha de consulta: 2 diciembre 2020] https://es.mathworks.com/help/matlab/creating_guis/write-callbacks-using-the-guide-workflow.html

[10] Presentaciones de Jose Luís Muñoz Lozano [fecha de consulta: 1 marzo 2021]