Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIAL

TRABAJO FIN DE GRADO

HERRAMIENTA DE CÁLCULO DE MOTORES SÍNCRONOS, CURVA DE PAR Y CURVA EN V DE MORDEY

Alumno: Fuente Ares, Xabier

Director: Valverde Santiago, Víctor

Curso: 2018-2019

Fecha: En Bilbao, 17 de julio de 2019

DATOS BÁSICOS DEL TRABAJO FIN DE GRADO

- Alumno: Fuente Ares, Xabier.
- Director: Valverde Santiago, Víctor.
- Departamento: Ingeniería Eléctrica.

- Título del Trabajo: Herramienta de cálculo de motores síncronos, curva de par y curva en V de Mordey.
- Resumen: El objetivo principal de este trabajo es el desarrollo de una interfaz gráfica de usuario que nos permita realizar cálculo de motores síncronos, de la gráfica de par y de la gráfica en V de Mordey. Para el desarrollo de esta haremos uso de GUIDE, herramienta que nos proporciona MATLAB para diseñar interfaces gráficas. Junto con la aplicación, también se realizará un estado del arte sobre los motores síncronos.
- Palabras clave: motor síncrono, interfaz gráfica, curva de par, curva en V de mordey, situación de funcionamiento.

- Izenburua: Motor sinkronoen, par-kurbaren eta Mordeyren V-kurbaren kalkulu erreminta.
- Laburpena: Lan honen helburu nagusia motor sinkronoen, par- kurbaren eta Mordeyren V-kurbaren kalkulua egin ahal izateko interfaze grafikoa garatzea da. Interfazearen garapena burutzeko GUIDE erabiliko dugu, MATLABek eskaintzen digun erreminta interfaze grafikoak egiteko. Aplikazioaz gain, motor sinkronoen egungo egoeraren azterketa egingo da.
- Hitzgakoak: motor sinkronoa, interfaze grafikoa, par-kurba, Mordeyren V-kurba, funtzionamendu egoera.

- Title: Calculation tool of synchronous motors, motor torque graphic and Mordey's V graphic.
- Abstract: The main goal of this project is to develop a graphical interface that enable us to make calculations of synchronous motors, motor torque graphic and Mordey's V graphic. In order to develop it, we will use GUIDE, a tool provided by MATLAB to create graphical interfaces. Apart from the application, we will also make the state-of-the-art of the synchronous motors.
- Keywords: synchronous motor, graphical interface, motor torque graphic, Mordey's V graphic, operating condition.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Este documento contiene el desarrollo del Trabajo de Fin de Grado (TFG) titulado “Herramienta de cálculo de motores síncronos, curva de par y curva en V de Mordey”. Para presentar de manera aproximada los conceptos que se tratarán en este TFG, primero se va a desarrollar el contexto de trabajo, los objetivos y alcances del mismo y los beneficios (técnicos, económicos, sociales) que se espera conseguir mediante su realización.

A continuación, en el apartado correspondiente al estado del arte se estudiará en profundidad el funcionamiento y las características del motor síncrono, para la posterior realización de la interfaz gráfica. Para ello se analizarán diferentes situaciones de funcionamiento a las que se puede ver sometido el motor síncrono, además de estudiar toda la formulación necesaria para poder realizar los cálculos necesarios para la resolución de dichas situaciones. Las fórmulas estudiadas en este apartado serán las utilizadas para el posterior desarrollo de la aplicación software.

En lo que se respecta al análisis de alternativas disponibles para la realización de la interfaz, se presentan los diferentes programas informáticos de los que disponemos explicando las ventajas y desventajas de cada uno de ellos. Posteriormente, una vez analizados los diferentes softwares se justifica la elección del más adecuado para el desarrollo de nuestra herramienta de cálculo.

El objetivo principal es la programación de la interfaz gráfica que nos permita calcular diferentes aspectos de los motores síncronos, además de que nos permita representar la curva de par y la curva en V de Mordey. Para ello se explicará detalladamente como se ha desarrollado la herramienta de cálculo, explicando las diferentes partes de las que está compuesta. Además, posterior a la explicación del desarrollo de la interfaz, en el apartado de ejemplos de aplicación se presentarán varios ejemplos en los que se puede observar el funcionamiento de la aplicación.

Este Trabajo de Fin de Grado también incluye un presupuesto en el que se detalla la inversión inicial necesaria para realizar el proyecto y un diagrama de Gantt en el que se muestran las diferentes tareas llevadas a cabo durante el proyecto y el orden en el que se han ejecutado.

Finalmente, se analizan los resultados obtenidos tras el estudio de los motores síncronos y la realización de la interfaz gráfica. Después de interpretar dichos resultados, se redactan las conclusiones que recogen un resumen de toda la información tratada y del proceso de realización del proyecto.

2. CONTEXTO

Este Trabajo de Fin de Grado se ha llevado a cabo en la Escuela de Ingenieros de Bilbao (EIB) en el departamento de Ingeniería Eléctrica y trata sobre el desarrollo de una interfaz gráfica que nos permita calcular diferentes situaciones de funcionamiento de los motores síncronos, además de representar la curva de par y la curva en V de Mordey.

2.1. Sistema eléctrico de potencia

Se conoce con el nombre de Sistema Eléctrico de Potencia al conjunto de equipos e instalaciones que posibilitan el uso de la energía eléctrica a gran escala. Las partes fundamentales del Sistema Eléctrico de Potencia son la generación, el transporte, la distribución y el consumo. Aunque cada parte tiene sus propios niveles de tensión, todas ellas constituyen un sistema trifásico a frecuencia constante (50 Hz en Europa y 60 Hz en América) [1].

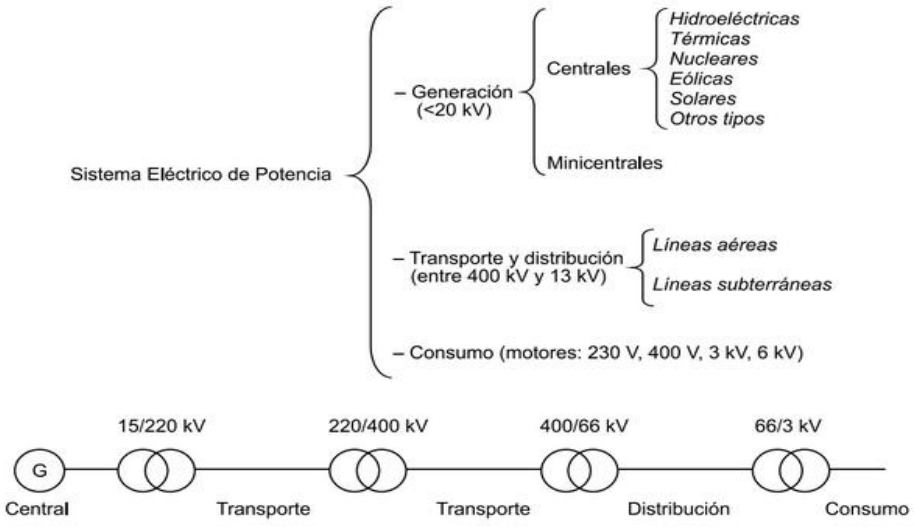

Figura 1. Partes de un sistema eléctrico de potencia *[1]* **.**

Las centrales eléctricas son las encargadas de la generación, para lo cual producen energía eléctrica haciendo uso de otro tipo de energías disponibles. Dependiendo de la manera mediante la cual las centrales generadoras produzcan la energía eléctrica, se pueden clasificar en nucleares, hidroeléctricas, térmicas, eólicas, etc. La energía eléctrica generada por estas centrales se consigue llevar hasta los consumidores mediante las redes de transporte y distribución. Las redes de transporte se encargan de transferir la energía desde las centrales de generación hasta los núcleos de reparto de carga principales o subestaciones. Las redes de distribución son las

encargadas de repartir la energía de estas subestaciones dentro de un núcleo de consumidores.

Finalmente, en los diferentes centros de consumo se recibe la energía generada en las centrales y se emplea en función de las características propias de cada centro (urbano, industrial, etc.). Este consumo puede realizarse a diferentes niveles de tensión de la red en función de la potencia demandada y los niveles de tensión presentes en la zona [1].

En consecuencia, para que el Sistema Eléctrico de Potencia pueda funcionar es necesario convertir diferentes formas de energía en energía eléctrica, transportar y distribuir esta energía hasta los centros de consumo y reconvertir la energía eléctrica a las formas de energía adecuadas para los diferentes tipos de consumo. Además, todo ello debe realizarse al nivel de tensión óptimo para cada etapa. Los elementos responsables de realizar las diferentes conversiones de energía y adecuar el nivel de tensión de cada etapa a su valor óptimo son las máquinas eléctricas de corriente alterna [1].

2.2. Clasificación de las máquinas eléctricas

Las máquinas eléctricas pueden clasificarse dependiendo de la movilidad de sus componentes. Esta clasificación deriva en dos tipos de máquinas:

- **Máquinas estáticas:** son aquellas que, en funcionamiento, tienen todas sus partes fijas. La máquina estática fundamental es el transformador [1].
- **Máquinas rotativas:** son aquellas que, en funcionamiento, tienen una parte fija y otra móvil. La parte fija se denomina estator y la parte móvil rotor [1].

Dentro de las máquinas rotativas, se puede hacer otra clasificación dependiendo de la función que realicen:

- **Motores:** su función es producir energía mecánica a partir del consumo de energía eléctrica. La energía mecánica se suele transferir en forma de par.
- **Generadores:** su función es producir energía eléctrica a partir de otras formas de energía que previamente se han convertido en energía mecánica.

Además, según trabajen con corriente continua o corriente alterna y tanto si desempeñan la función de generadores como si realizan la función de motores, las máquinas rotativas se clasifican en máquinas de corriente continua y máquinas de corriente alterna (síncronas o asíncronas) [1].

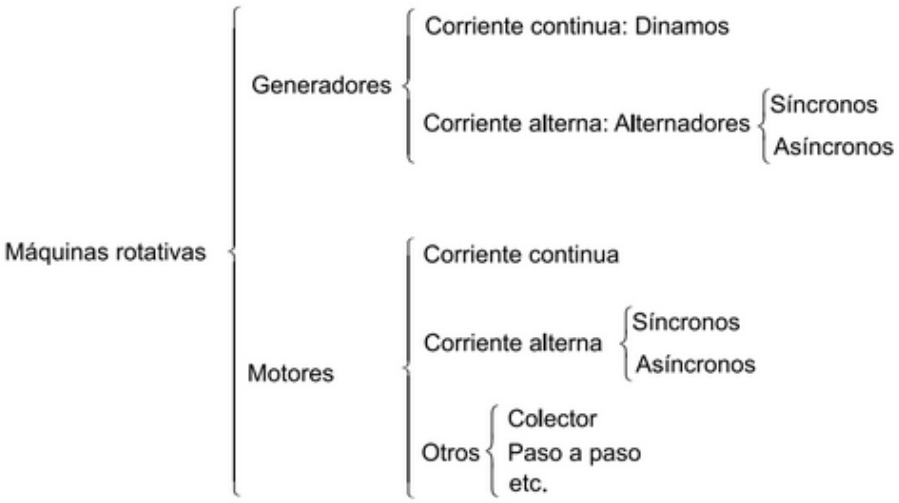

Figura 2. Clasificación de las máquinas rotativas *[1]*.

2.3. La máquina síncrona

Las máquinas síncronas son máquinas eléctricas cuya velocidad de rotación n está vinculada con la frecuencia f de la red de frecuencia alterna con la que trabaja, mediante la siguiente expresión:

$$n = \frac{60*f}{p} \quad (1)$$

siendo p el número de pares de polos [2].

Las máquinas síncronas, en función de la conversión de energía que realicen, mecánica en eléctrica o eléctrica en mecánica, puede trabajar como generadores (alternadores) o como motores [1]. En la práctica, en las instalaciones eléctricas es más frecuente su empleo como generadores, para producir energía eléctrica de corriente alterna a partir de fuentes primarias. En cuanto a su uso como motor, la máquina síncrona se emplea como motor en aquellos accionamientos industriales que requieren velocidades de transmisión constantes, teniendo la ventaja que nos ofrecen los motores síncronos de poder regular el factor de potencia con el que trabaja. Esta capacidad de regulación del factor de potencia es de gran importancia en ciertos tipos de industrias, ya que se evita la colocación de condensadores para la reducción de la potencia reactiva absorbida por la instalación [2].

En las máquinas síncronas, la parte móvil, el rotor, está recorrido por corriente continua y la parte fija, el estator, por corriente alterna. Además, en función de las condiciones de funcionamiento a las que vaya a estar sometida la máquina, el rotor puede ser de dos tipos:

- **Rotor de polos salientes:** son rotores de un diámetro considerable y se emplean cuando las velocidades de giro no son muy elevadas (<1000 rpm).
- **Rotor liso o cilíndrico:** tienen menor diámetro que los de polos salientes, pero mayor longitud. Se emplean para velocidades de giro elevadas (>1000 rpm).

2.4. El motor síncrono

Es el modo de funcionamiento mediante el cual la máquina síncrona produce energía mecánica a través del consumo de energía eléctrica. Para su funcionamiento es necesario conectar sus bobinados del estator a una red trifásica activa y debe alimentarse su bobinado rotórico con corriente continua. Este modo de funcionamiento es menos habitual que su aplicación como generador, pero es de gran utilización en aquellas aplicaciones que requieren velocidades de funcionamiento constantes. Además, el motor síncrono tiene la capacidad de regular su factor de potencia, pudiendo utilizarse como compensador de fase regulando la potencia reactiva que absorbe o cede a la red [1].

En cuanto a su modo de operación, la capacidad de la mayoría de los motores síncronos se sitúa entre los 150 KW y los 15MW, y giran a velocidades que oscilan entre 150 y 1800 rpm. Teniendo esto en cuenta, la mayoría de sus aplicaciones están relacionadas con actividades industriales pesadas, aunque también pueden aplicarse en otros ámbitos como dispositivos de control o relojes eléctricos [3].

Por último, es importante destacar que los motores síncronos no disponen de par de arranque. Esta situación provoca la necesidad de disponer de un método de arranque mediante el cual se pueda poner el motor en funcionamiento.

3. OBJETIVOS Y ALCANCE

El objetivo principal de este proyecto consiste en el desarrollo de una interfaz gráfica sencilla e intuitiva mediante la utilización de GUIDE, herramienta que nos proporciona MATLAB para el desarrollo de interfaces gráficas, que nos permita definir las curvas de comportamiento de un motor síncrono (Curva de par y curva en V de Mordey) en base a sus valores nominales y a los datos obtenidos en sus ensayos de vacío y cortocircuito.

Además del objetivo principal, durante el desarrollo del proyecto se presentan otros objetivos:

- Profundizar en el conocimiento sobre las máquinas síncronas y más en concreto sobre el motor síncrono.
- Analizar el funcionamiento de los motores síncronos ante diferentes datos de entrada, y observar cómo varía la curva de par y la curva en V de Mordey frente a diferentes situaciones de funcionamiento.
- Profundizar en la utilización de GUIDE y ser capaz de programar correctamente las funciones necesarias para el correcto funcionamiento de la interfaz.
- Reducción de tiempos a la hora de estudiar los motores síncronos. La interfaz nos permite realizar numerosos cálculos y analizar diferentes situaciones de funcionamiento en un corto periodo de tiempo.
- Optimización de recursos económicos, humanos y materiales a la hora de analizar un motor síncrono mediante el uso de la interfaz gráfica.
- Evitar posibles fallos reales en el funcionamiento de un motor síncrono mediante la simulación previa de las situaciones de funcionamiento en la interfaz.
- Como objetivo personal, profundizar en la utilización de MATLAB durante el desarrollo de la interfaz, permitiéndome adquirir un mayor dominio del programa y aplicar dicho conocimiento a otros ámbitos externos a este proyecto.

4. BENEFICIOS DEL PROYECTO

Los beneficios que se van a obtener como consecuencia del desarrollo de este proyecto se pueden dividir en tres clases diferentes, beneficios técnicos, beneficios económicos y beneficios sociales. Los más importantes serán los beneficios técnicos debido al ámbito en el que se desarrolla el proyecto.

4.1. Beneficios técnicos

Los motores síncronos son máquinas presentes en numerosas aplicaciones industriales. Uno de sus principales ámbitos de utilización son aquellas aplicaciones en las que se necesita corregir el factor de potencia, acción en la que los motores síncronos ofrecen un elevado rendimiento. Además de para la corrección del factor de potencia, los motores síncronos también son utilizados para otras muchas aplicaciones industriales destacadas: ventiladores, bombas y compresores en la siderurgia; extrusoras en la industria del papel; compresores y ventiladores de alta capacidad en la industria química y petroquímica; bombas de inyección de agua en plataformas petrolíferas flotantes. Debido a la presencia de los motores síncronos en todos estos ámbitos de trabajo, es imprescindible poder asegurar su correcto funcionamiento a lo largo del tiempo.

Ser capaces de localizar posibles fallos del motor ante las diferentes situaciones de funcionamiento a las que se verá sometido, antes de que estos fallos se produzcan ahorraría numerosas averías y desperfectos en la máquina. La interfaz gráfica desarrollada a lo largo de este TFG permite detectar dichos errores de funcionamiento ante las diferentes situaciones que se nos presentan. Introduciendo en nuestra interfaz los datos nominales del motor síncrono, haciendo uso de los datos de los ensayos de vacío y cortocircuito de la máquina síncrona e introduciendo los datos de la situación de funcionamiento a la que va a estar sometido, la interfaz representará la curva de par y la curva en V de Mordey, indicando el punto de funcionamiento del motor. Mediante el análisis de los resultados se podrá predecir si la máquina tendrá dificultades para soportar dichas condiciones de funcionamiento. Como consecuencia, al predecir las posibles dificultades que tendrá el motor, se reducen los costes técnicos necesarios para poder asegurar el correcto funcionamiento de la máquina.

4.2. Beneficios económicos

La oportunidad de disponer de una interfaz que nos permita predecir posibles fallos en el funcionamiento del motor supone un gran ahorro económico si lo comparamos con otras alternativas como los ensayos de campo o la utilización de

modelos a escala. Además, el uso de la aplicación también supondrás un ahorro en lo que a labores de mantenimiento de la máquina se refiere, debido a que gracias a la aplicación se pueden predecir aquellas situaciones que no serán favorables para la máquina. Evitando que el motor funcione en dichas situaciones, el tiempo de vida de los componentes de la máquina se aumentará reduciendo el gasto destinado a recambios y labores de restauración de ciertas piezas.

Como consecuencia de lo ahora mencionado, tener al alcance una interfaz gráfica que nos de los resultados de las diferentes situaciones de funcionamiento a las que se someterá a la máquina, supone un ahorro en el mantenimiento de la misma y permite evitar posibles fallos futuros. Esto se traduce en ahorro en lo que a recursos económicos se refiere.

4.3. Beneficios sociales

Las numerosas aplicaciones de los motores síncronos exigen que estos aseguren un correcto funcionamiento bajo unas ciertas medidas de seguridad. El hecho de poder predecir posibles fallos y averías en los mismos mediante la utilización de la interfaz gráfica permite asegurar que se cumplirá con las necesidades requeridas.

Como consecuencia del cumplimiento de las necesidades de funcionamiento y seguridad de los motores síncronos, se evitarán posibles lesiones provocadas por fallos inesperados en la máquina a aquellos que trabajen con ellas o en un entorno cercano. Además, al garantizar su funcionamiento en largos periodos de tiempo, todos aquellos procesos en los que los motores síncronos están involucrados se llevarán a cabo de manera más continuada.

Este aumento en la seguridad y en el tiempo de funcionamiento de los motores, supone un gran beneficio para los trabajadores que habitualmente estén en contacto con ellos, y también para todas aquellas personas que se beneficien de las aplicaciones derivadas de los motores, que como se ha mencionado anteriormente son numerosas en el ámbito industrial.

5. ESTADO DEL ARTE

El objetivo principal de este Trabajo de Fin de Grado es el desarrollo de una interfaz gráfica que nos permita realizar cálculos en motores síncronos y nos permita graficar la curva de par y la curva en V de Mordey. Para poder llevar a cabo dicho objetivo, es necesario profundizar en el estudio de la máquina síncrona y más en concreto del motor síncrono.

En este apartado se van a estudiar las partes más significativas de una máquina síncrona, así como sus ensayos de vacío y cortocircuito, para posteriormente poder estudiar en profundidad el funcionamiento del motor síncrono. En este estudio del motor síncrono se explicarán sus principales características, sus curvas características, sus métodos de arranque, etc.

5.1. Elementos constitutivos de las máquinas síncronas

Como ya se ha mencionado anteriormente, las máquinas síncronas son máquinas de corriente alterna cuya velocidad en estado estacionario es proporcional a la frecuencia de la red a la que están conectadas. Además, el rotor, junto con el campo magnético creado por la corriente del campo de corriente continua en el rotor, gira a la misma velocidad que el campo magnético giratorio producido por las corrientes de la armadura, o en sincronismo con él, y da como resultado un par constante [4]

La máquina síncrona puede trabajar como generador o como motor, en función de la conversión de energía que realicen. Independientemente del modo de funcionamiento en el que se encuentre la máquina, generador o motor, los elementos más significativos de este tipo de máquinas son los siguientes:

- **Rotor:** es la parte móvil de la máquina y está recorrido por corriente continua. En función de la aplicación a la que se destine la máquina existen dos tipos, rotor cilíndrico y rotor de polos salientes. El rotor cilíndrico es de menor diámetro y mayor longitud, está destinado a velocidades superiores a 1000 rpm. El rotor de polos salientes es de mayor diámetro y está destinado a velocidades más bajas, menores de 1000 rpm.

Figura 3. Rotor de polos salientes *[1]*.

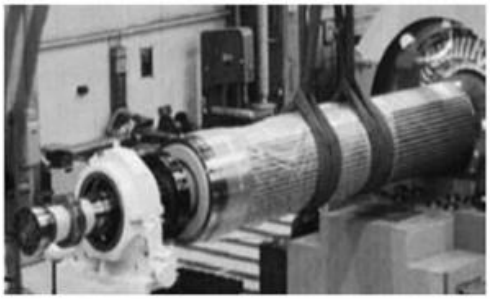

Figura 4. Rotor cilíndrico *[1]*.

- **Estator:** es la parte fija de la máquina y está recorrida por corriente alterna. Está formado por un conjunto de chapas apiladas y aisladas entre sí. Esto per-mite disminuir las pérdidas en el hierro del circuito magnético de la máquina y mejorar, por lo tanto, el rendimiento [1].
- **Bobinado del estator:** es el circuito eléctrico situado en el estator. Está formados por espiras en las que se inducen tensiones alternas y, cuando el generador trabaja en carga, son recorridas por corriente alterna [1].
- **Bobinado del rotor:** es el circuito eléctrico situado en el rotor, se alimenta mediante la inyección de corriente continua con la finalidad de producir un campo magnético de magnitud constante [5].
- **Bobinado amortiguador:** Los devanados de amortiguamiento son unas barras especiales dispuestas en ranuras labradas en la cara del rotor de un motor síncrono y en cortocircuito en cada extremo con un gran anillo en cortocircuito [6].
- **Sistema de excitación:** es el conjunto de equipos mediante los que alimentar el devanado del rotor con una tensión e intensidad de corriente continua. Hoy en día se utilizan principalmente dos tecnologías [1]:
  - **Sistema de excitación estático:** la corriente continua necesaria para alimentar el devanado del rotor se genera a base de equipos rectificadores basados en electrónica de potencia, para posteriormente introducirla en el devanado eléctrico a través de un sistema de escobillas que rozan sobre unos anillos que giran solidarios con el eje y con el devanado del rotor [1].

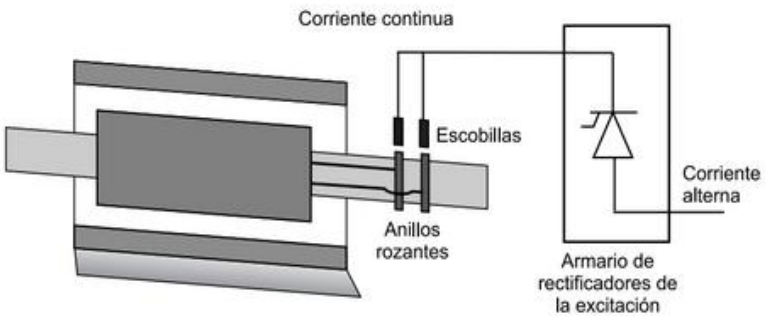

Figura 5. Sistema de excitación estático *[1]*.

- **Sistema de excitación brushless (sin escobillas):** la corriente continua se genera mediante un alternador inverso que proporciona corriente alterna que después se rectifica a través de unos diodos que giran solidariamente con el rotor [1].

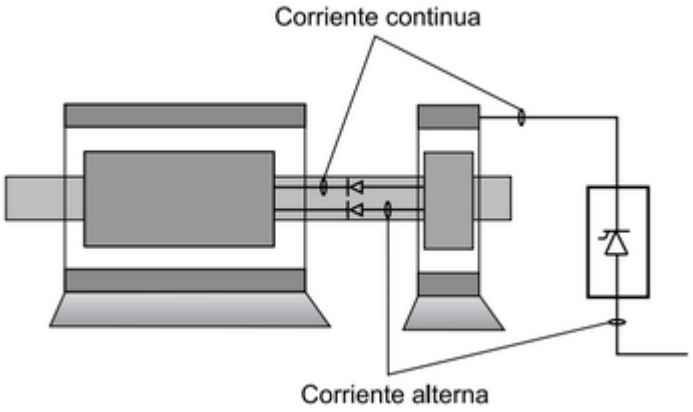

Figura 6. Sistema de excitación brushless *[1]*.

- **Sistema de refrigeración:** existen numerosas fuentes de refrigeración dentro de una máquina síncrona. Por este motivo es necesario refrigerar los elementos constitutivos de la máquina, así se alarga la vida útil de dichos elementos y se mejora su rendimiento. El medio más utilizado para conseguir una óptima refrigeración es el aire que fluye gracias a los ventiladores situados sobre el eje de la máquina.

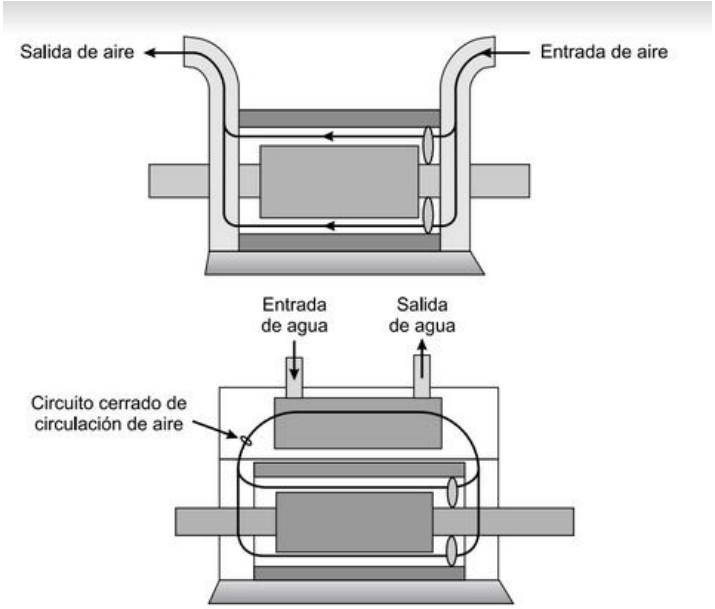

Figura 7. Sistema de refrigeración *[1]*.

- **Carcasa:** tanto rotor como estator van introducidos dentro de una carcasa. Mediante la carcasa se consigue apoyo, fijación y la protección de los elementos de la máquina.

5.2. Ensayos de la máquina síncrona

A la hora de trabajar con la máquina síncrona, es necesario definir una serie de parámetros indispensables para poder realizar los cálculos necesarios para poder representar la curva de par y la curva en V de Mordey, uno de los objetivos principales de nuestro trabajo. Esos parámetros indispensables son la relación entre la tensión de vacío (E_0) y la intensidad de excitación (I_E) por un lado, y la reactancia síncrona de la máquina (X_S) por otro.

La manera de poder definir estos dos parámetros es realizar los correspondientes ensayos de vacío y de cortocircuito. Estos ensayos se realizan con la máquina síncrona trabajando como generador, pero el valor de los parámetros se mantiene cuando la máquina pasa a trabajar como motor.

5.2.1. Ensayo de vacío

El objetivo principal de este ensayo es definir la relación entre la tensión de vacío y la corriente de excitación (E_0/I_E). Para realizar el ensayo, se hace girar el generador a velocidad nominal, se desconectan los terminales de cualquier carga y se establece la intensidad de excitación como cero. Una vez hecho esto, se va incrementando la

intensidad de excitación por etapas y se va midiendo el valor de la tensión [6]. De esta manera es posible dibujar la gráfica que nos relaciona ambos parámetros.

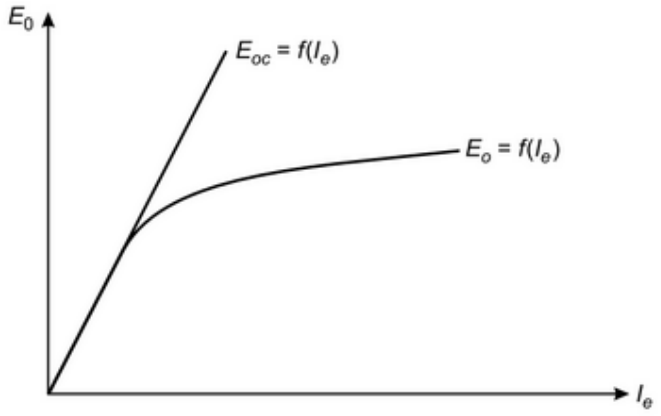

Figura 8. Curva del ensayo de vacío *[1]*.

Como se puede observar, al principio la curva es casi perfectamente lineal, hasta que se produce cierta saturación para intensidades de excitación elevadas. Esto se debe a que cuando la máquina trabaja en la zona de no saturación casi toda la fuerza magnetomotriz pasa a través del entrehierro y el incremento del flujo resultante es lineal. Sin embargo, cuando se satura el hierro, la reluctancia en éste se incrementa de manera notoria y el flujo se incrementa mucho más despacio con el incremento en la fuerza magnetomotriz [6].

5.2.2. Ensayo de cortocircuito

Mediante el ensayo de cortocircuito se consigue calcular el valor de la reactancia síncrona de la máquina. Para su realización, se establece la intensidad de excitación igual a cero y se cortocircuitan los terminales del generador mediante un conjunto de amperímetros [6]. Después, se mide la corriente en el estator según se va aumentando la corriente de excitación. En este caso se le ha llamado I_a a la corriente que circula por el estator e I_f a la corriente de excitación.

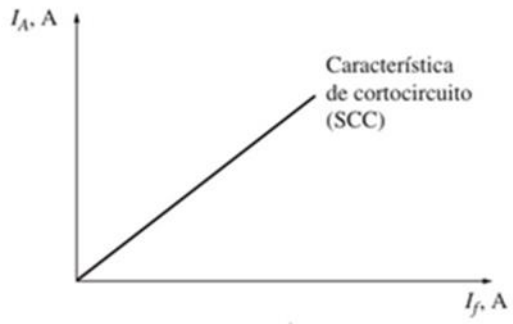

Figura 9. Curva del ensayo de cortocircuito *[6]***.**

Teniendo en cuenta el circuito equivalente del generador durante el ensayo:

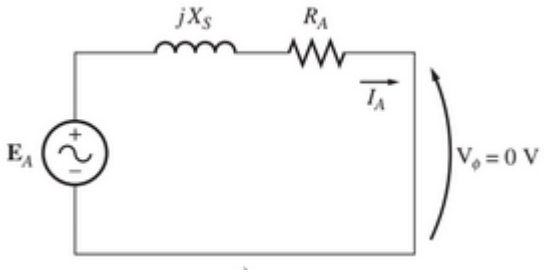

Figura 10. Circuito equivalente ensayo de cortocircuito *[6]***.**

Por lo tanto, se deduce que la expresión que nos proporcionará el valor de I_a será:

$$I_A = \frac{E_A}{\sqrt{{R_A}^2 + {X_S}^2}} \quad (2)$$

Teniendo en cuenta que $X_S >> R_A$, el valor de la reactancia síncrona de la máquina viene dado por:

$$X_S = \frac{E_A}{I_A} \quad (3)$$

5.3. El motor síncrono

En su aplicación como motor, la máquina síncrona convierte la energía eléctrica que absorbe de la red en energía mecánica. Esta energía mecánica es transmitida a través del eje de la máquina en forma de par. En el funcionamiento como motor, los bobinados del estator deben ser conectados a una red trifásica activa y debe hacerse circular corriente continua por su bobinado rotórico [1]. El uso de la máquina síncrona

como motor es menos habitual que su uso como generador, y está destinado principalmente a aplicaciones que requieran una velocidad de funcionamiento constante o aplicaciones en las que sea necesario que varios motores funcionen exactamente a la misma velocidad [7].

5.3.1. Principio de funcionamiento del motor síncrono

Como ya se ha mencionado anteriormente, para trabajar como motor se debe excitar el rotor de la máquina con corriente continua, y el bobinador estatórico debe conectarse a una red trifásica. Además, el motor síncrono precisa de un método de arranque, aspecto que se explicará a continuación.

Inicialmente, el rotor del motor se encuentra en reposo, debido a esto la intensidad de corriente continua que circula por sus bobinados crea un campo magnético de amplitud constante y fijo en el espacio [1].

Por otro lado, el bobinado estatórico, conectado a una red alterna trifásica, provoca la circulación de un sistema trifásico de intensidades que, por el teorema de Ferraris, genera un campo rotativo que gira a la velocidad de sincronismo [1].

La interacción entre los campos de rotor y estator provoca un par que tiende a lanzar el rotor en un sentido de giro durante medio periodo y en el sentido contrario durante el medio periodo siguiente. En la siguiente imagen se muestra la explicación de esta situación en la que intervienen las siguientes variables:

B=Inducción creada por los polos del rotor.

i(t)=Intensidad de alterna que circula por el bobinado del estator.

F=Fuerza que, aplicando el teorema de Laplace, se crea por la interacción entre la inducción y la intensidad. Por el principio de acción y reacción, esta fuerza tiende a lanzar al rotor en sentido contrario a su dirección de aplicación. Como puede apreciarse, la fuerza se aplica alternativamente en uno y otro sentido a intervalos de tiempo de medio periodo [1].

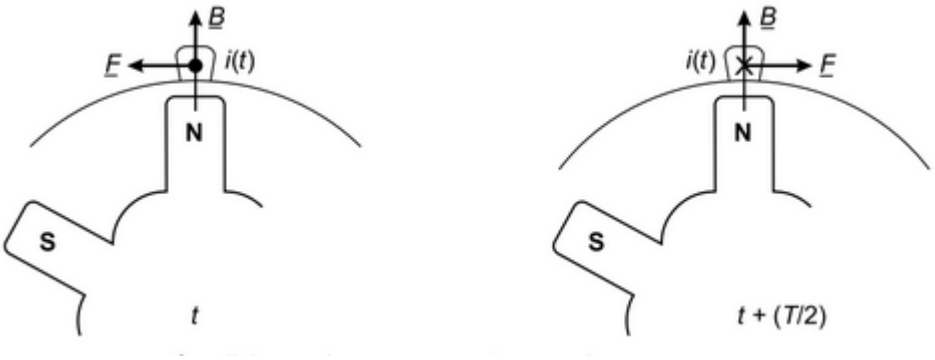

Figura 11. Par de arranque nulo del motor síncrono *[1]*.

Por este motivo, los motores síncronos no disponen de par de arranque y es necesario aplicarles un método de arranque para ponerlos en funcionamiento. Una vez se arranca el motor y se alcanza la velocidad de sincronismo, el par creado por la interacción de los campos de rotor y estator hace girar al eje con la velocidad síncrona en el sentido de rotación del campo rotativo creado por el estator [1].

5.3.2. Métodos de arranque

A la hora de realizar el arranque de un motor síncrono se dispone de diferentes métodos para lograrlo. Los métodos más utilizados son los 3 siguientes:

- Arranque mediante convertidor de frecuencia.
- Arranque mediante motor auxiliar.
- Arranque como motor asíncrono.

5.3.2.1. Arranque mediante convertidor de frecuencia

Cuanto más baja sea la frecuencia, mayor será el tiempo de duración de un semiperiodo [1]. Por este motivo, si los campos magnéticos del estator en un motor síncrono giran a una velocidad lo suficientemente baja, el rotor se acelerará hasta enlazar con el campo magnético del estator. Entonces se puede incrementar la velocidad de los campos magnéticos del estator aumentando gradualmente fe hasta su valor normal de 50 o 60 Hz [6].

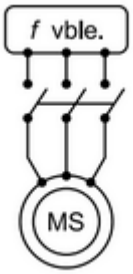

Figura 12. Arranque mediante convertidor de frecuencia *[1]*.

5.3.2.2. Arranque mediante motor auxiliar

Este método de arranque consiste en adjuntar un motor de arranque externo y llevar la máquina síncrona hasta su velocidad plena con un motor externo [6]. Una vez alcanzada la velocidad de sincronismo, se le acopla la carga mecánica a arrastrar. El motor auxiliar suele ser de reducida potencia, ya que el arranque se realiza en vacío [1].

Como motor auxiliar se suele utilizar un motor asíncrono del mismo número de polos que el motor síncrono o de un par de polos menos. En caso de utilizar un motor asíncrono del mismo número de polos, se lleva el motor hasta una velocidad ligeramente inferior a la de sincronismo, pero suficiente para que se produzca un fenómeno de autosincronización que hace que el eje del motor síncrono quede girando a la velocidad de sincronismo. En caso de utilizar un motor asíncrono de un par de polos menos, la conexión a red se realiza una vez desconectado el motor auxiliar y el grupo pasa suavemente por la velocidad de sincronismo [2].

5.3.2.3. Arranque como motor asíncrono

Es el método de arranque más utilizado. La presencia del denominado bobinado amortiguador en el rotor le proporciona al motor síncrono, durante el arranque, las características de un motor asíncrono del tipo jaula de ardilla [1]. El bobinado amortiguador, como ya se explicó en el apartado de las partes constitutivas de la máquina síncrona, son unas barras especiales dispuestas en ranuras labradas en la cara del rotor de un motor síncrono y en cortocircuito en cada extremo con un gran anillo en cortocircuito [6]. El proceso de arranque es el siguiente:

1. Se cierra el devanado del rotor sobre una resistencia de valor elevado [1].
2. Al cerrar el interruptor que conecta el devanado estatórico con la red trifásica, el motor síncrono se comporta como un motor asíncrono de tipo jaula de ardilla y llega a alcanzar una velocidad de giro próxima a la de sincronismo, pero ligeramente inferior [1].

3. Se pasa a alimentar al rotor mediante corriente continua, con lo que se origina un proceso de autosincronización que lleva al motor síncrono a alcanzar la velocidad de sincronismo [1].

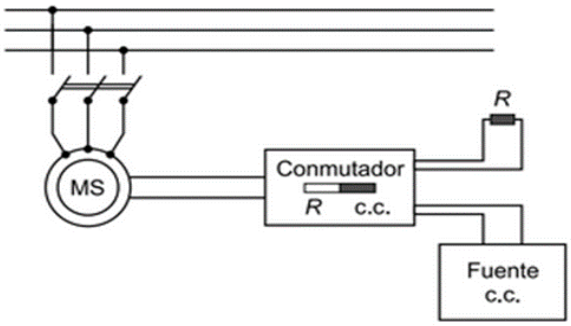

Figura 13. Arranque como motor asíncrono *[1]*.

Como se puede observar durante el proceso, es de gran importancia el conmutador que nos permite cerrar el devanado del rotor sobre una resistencia o alimentarlo mediante corriente continua en función de la posición en la que se sitúe. Para poder limitar la corriente inicial, se suele alimentar el motor con tensiones reducidas durante el proceso de arranque [1].

5.3.3. Diagrama vectorial

Cuando la máquina síncrona funciona como generador, la máquina entrega potencia activa a la red a la que se conecta su circuito estatórico [1]. La expresión vectorial que nos describe el funcionamiento de un generador síncrono de rotor cilíndrico es la siguiente:

$$\underline{E}_0 = \underline{V} + \underline{I} \cdot (R + jX_S) \quad (4)$$

Por el contrario, cuando la máquina síncrona se emplea como motor síncrono, la máquina absorbe potencia activa de la red, y por lo tanto se invierte el sentido de la intensidad que recorre los bobinados del estator. Como consecuencia, la expresión vectorial en su funcionamiento como motor es la misma que como generador, pero cambiando el signo de la intensidad:

$$\underline{E}_0 = \underline{V} + (-\underline{I}) \cdot (R + jX_S) \quad (5)$$

Reordenando la ecuación:

$$\underline{V} = \underline{E}_0 + \underline{I} \cdot (R + jX_S) \quad (6)$$

Partiendo de esta última expresión, se puede dibujar el diagrama vectorial del motor síncrono:

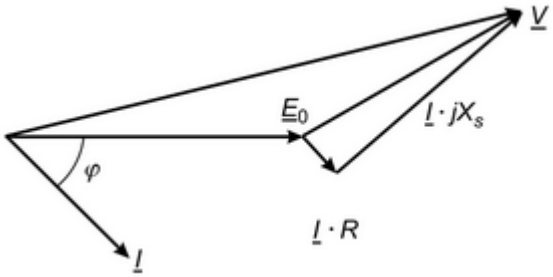

Figura 14. Diagrama vectorial del motor síncrono *[1]*.

Como se puede observar, a la hora de realizar el diagrama se ha supuesto un factor de potencia inductivo.

Teniendo en cuenta todo lo expuesto, también se puede dibujar el circuito equivalente correspondiente al motor. Al igual que en el diagrama vectorial, el circuito equivalente de un motor síncrono es exactamente igual al circuito equivalente de un generador síncrono, excepto en que la dirección de referencia de I está invertida [6].

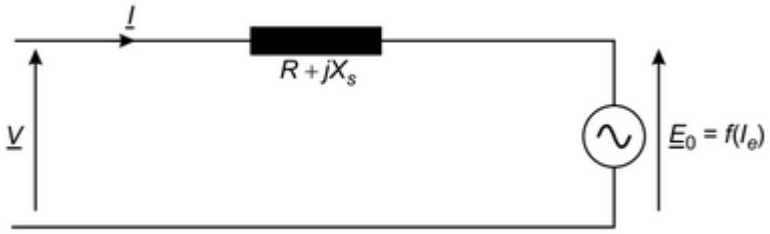

Figura 15. Circuito equivalente del motor síncrono *[1]*.

5.3.4. Funcionamiento del motor síncrono ante cambios en la carga

Si se fija una carga al eje de un motor síncrono, éste desarrollará suficiente par como para mantener el motor y su carga a velocidad síncrona [6].

Si se produce un cambio en dicha carga el motor síncrono también será capaz de mantener la velocidad. Si se incrementa la carga en el eje, en un principio el rotor perderá velocidad. Una vez se reduce la velocidad, el ángulo de par se incrementa aumentando así la potencia mecánica producida por el motor. De esta manera, el motor vuelve a funcionar a su velocidad inicial, pero con un ángulo de par mayor [6]. Si por el contrario se decrementa la carga en el eje, se produciría el efecto contrario. La velocidad se incrementaría en un principio, pero este incremento se vería contrarrestado por un decremento del ángulo de par [1]. Este proceso se puede observar representando

potencia activa del motor síncrono en función del ángulo de par, a tensión e intensidad de excitación constantes.

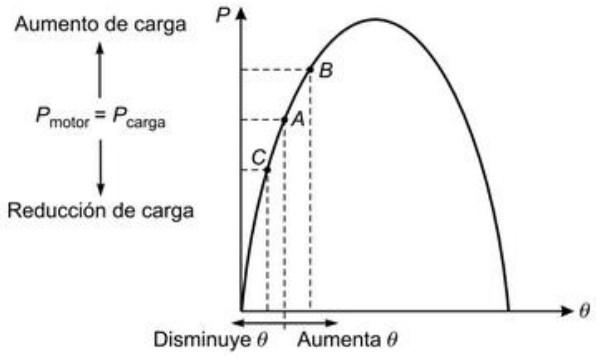

Figura 16. Curva de potencia en función del ángulo de par *[1]*.

La expresión de la potencia en función del ángulo de par es la siguiente:

$$P = 3 \cdot V \cdot \frac{E_0}{X_S} \cdot \sin(\theta) \quad (7)$$

5.3.5. Curvas características del motor síncrono

Además de por las características mencionadas en apartados anteriores, el funcionamiento del motor síncrono viene caracterizado por sus curvas a par constante y por sus curvas en V de Mordey.

5.3.5.1. Curvas a par constante

Como el motor síncrono se caracteriza por funcionar a velocidad constante, cuando hablamos de curvas a par constante, estamos hablando de curvas a potencia constante. Teniendo en cuenta la expresión (7):

$$C = \frac{P}{\omega} = cte \ (8) \ \rightarrow \ P = 3 \cdot V \cdot \frac{E_0}{X_S} \cdot \sin(\theta) = cte$$

Además, teniendo en cuenta que la tensión de la red de alimentación (V) es constante y que X_S también es constante, ante variaciones en la intensidad de excitación se tiene que cumplir lo siguiente:

$$E_0 \cdot \sin(\theta) = cte$$

Por lo tanto, se puede deducir que si se aumenta la excitación (sobreexcitación) aumentará el valor de la tensión de vacío E_0, y como consecuencia disminuirá el valor del ángulo de par. Si por el contrario se disminuye la excitación (subexcitación), disminuirá el valor de E_0, pero aumentará el valor del ángulo de par. Estas dos

situaciones se ven perfectamente reflejadas en la gráfica a par constante del motor síncrono.

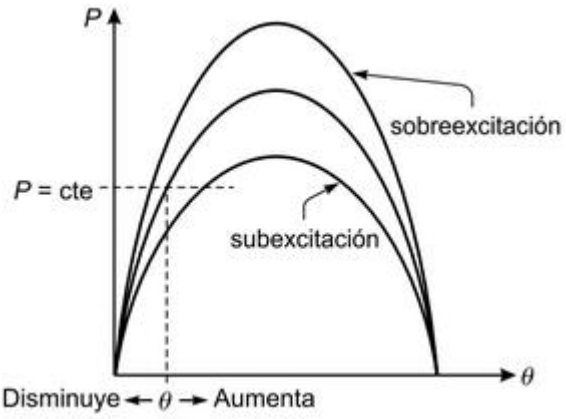

Figura 17. Curva de par para diferentes excitaciones *[1]*.

5.3.5.2. Curvas en V de Mordey

Las curvas en V de Mordey de un motor síncrono nos muestran la variación de la intensidad que recorre los bobinados del estator (I), frente a variaciones en la intensidad de excitación del rotor (I_E). Partiendo de la expresión que nos relaciona la potencia con la intensidad que recorre los bobinados del estator:

$$P = 3 \cdot V \cdot I \cdot \cos\varphi \quad (8)$$

Teniendo en cuenta que cada curva en V corresponde a una potencia fija y que la tensión de alimentación de la red V es constante, se deduce que:

$$I \cdot \cos\varphi = cte$$

Teniendo en cuenta esta condición, junto con la expresión (6) que nos muestra que la intensidad que recorre los bobinados del estator (I) varía cuando se producen cambios en la tensión de vacío (E_0), la cual está directamente relacionada con la intensidad de excitación (I_E), se puede deducir que el control de la intensidad de excitación (I_E) permite controlar la intensidad que circula por el estator (I) y el factor de potencia (cos φ) con el que trabaja el motor síncrono [1]. En la siguiente gráfica se muestran las curvas en V de Mordey de un motor síncrono para diferentes valores de la potencia:

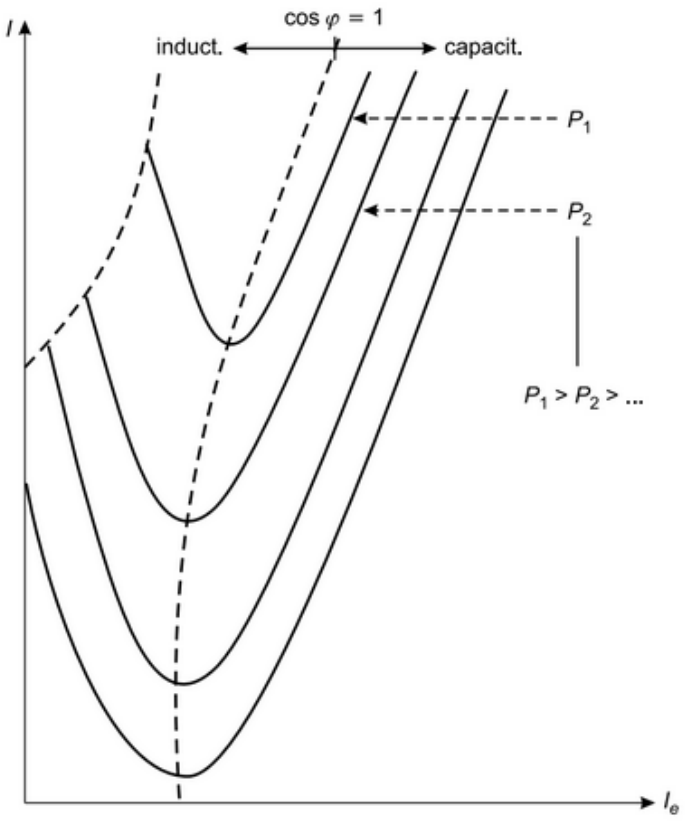

Figura 18. Curvas en V de Mordey *[1]*.

Observando la gráfica se pueden sacar las siguientes conclusiones:

- La corriente mínima del inducido (I) se presenta con un factor de potencia unitario cuando sólo se suministra al motor potencia real [6].
- En el caso de corrientes de excitación (I_E) menores que el valor que resulta en la mínima I, la corriente del inducido está en retraso (factor de potencia inductivo) y consume potencia reactiva [6].
- En el caso de corrientes de excitación (I_E) mayores que el valor que resulta en la mínima I, la corriente del inducido está en adelanto (factor de potencia capacitivo) y cede potencia reactiva [6].

5.3.6. Corrección del factor de potencia

Una de las principales aplicaciones de los motores síncrono es su utilización para corregir el factor de potencia. Un motor síncrono tiene la capacidad de absorber o suministrar potencia reactiva en un sistema trifásico para estabilizar el voltaje [3]. Como consecuencia, el motor síncrono es ampliamente utilizado para mejorar el factor de potencia en instalaciones industriales.

Cuando el motor síncrono se utiliza únicamente con este propósito, el motor se opera en vacío y recibe el nombre de capacitor síncrono [6]. A continuación, se muestra un diagrama vectorial del efecto del motor síncrono en la mejora del factor de potencia.

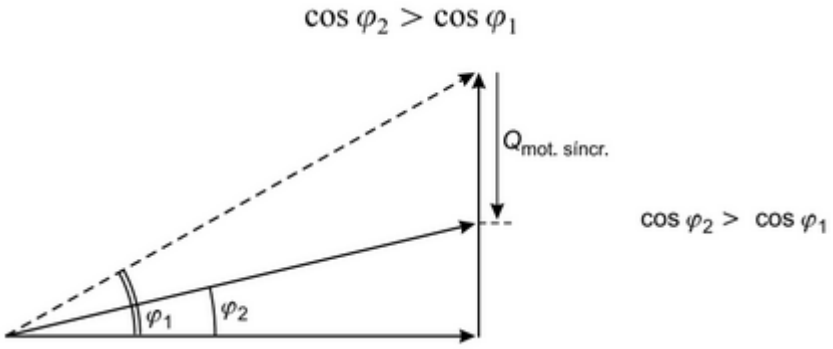

Figura 19. Motor síncrono como compensador de fase *[1]*.

6. ANÁLISIS DE ALTERNATIVAS

En este apartado se van a estudiar las diferentes alternativas de las que podemos disponer para la realización del proyecto. Por una parte, teniendo en cuenta que el objetivo de este trabajo es la realización de una interfaz gráfica, se analizarán diferentes opciones a la realización de la aplicación como pueden ser las pruebas de campo o la realización de modelos físicos a escalas y se justificará la decisión tomada. Por otra parte, se estudiarán los diferentes programas informáticos a los que tenemos acceso para la realización de la interfaz, teniendo en cuenta diferentes aspectos como el aspecto económico, la simplicidad del programa, el tiempo de simulación del mismo o la flexibilidad que nos ofrece. Teniendo en cuenta todos estos aspectos se realizará un estudio y se justificará la elección del programa más adecuado.

6.1. Análisis de alternativas para el estudio del motor síncrono.

Teniendo en cuenta que el principal objetivo de este trabajo es el desarrollo de una aplicación software que nos permita calcular las curvas de par y curva en V de Mordey frente a unas condiciones de carga determinadas, en este apartado se va a justificar la elección de la simulación software para el cálculo de dichas curvas frente a otras alternativas como son las pruebas de campo y la simulación de modelos físicos a escala.

A continuación, se presentan cada una de las 3 alternativas que se han tenido en cuenta.

6.1.1. Pruebas de campo

La realización de las pruebas de campo consiste en realizar las medidas necesarias en la propia máquina. En nuestro caso se tendrían que realizar los ensayos de vacío y de cortocircuito en primer lugar, y después poner a funcionar el motor en las condiciones de carga que se nos indican. Una vez realizadas las medidas necesarias, se analizan dichos datos y posteriormente se realizan los cálculos necesarios. De las alternativas disponibles es la que nos ofrece resultados que más se ajustan a la realidad, pero el coste económico es muy elevado y en caso de cometer un error en la realización de las medidas pueden provocar importantes averías en el motor.

6.1.2. Modelos físicos a escala

Mediante esta alternativa se realiza un modelo a escala de un motor síncrono real en el cual se pueden realizar los ensayos y medidas necesarias para la obtención de los resultados requeridos. Las medidas que obtenemos mediante este método no

son reales debido a que estamos trabajando con un modelo hecho a escala, pero se pueden llegar a obtener resultados lo suficientemente fiables si se han realizado de manera correcta las semejanzas eléctricas entre el modelo y el motor real. El claro inconveniente de esta alternativa es la gran inversión necesaria para poder crear un modelo de calidad que nos pueda ofrecer buenos resultados.

6.1.3. Simulación software

Esta alternativa consiste en el desarrollo de una aplicación software que nos permita simular las diferentes situaciones de funcionamiento que queremos analizar en el motor síncrono. Mediante la introducción de unos datos de entrada, como las características nominales del motor y los ensayos de vacío y cortocircuito, la aplicación nos proporcionará los resultados requeridos y nos dibujará la curva de par y la curva en V de Mordey. La simulación software es la alternativa que mayor cantidad de resultados nos ofrece y que menos inversión requiere. Dichos resultados no son reales, pero si se ha hecho un diseño adecuado pueden ser resultados bastante fiables.

6.1.4. Criterios de selección

En este apartado se explican los criterios que se han tenido en cuenta a la hora de tomar una decisión acerca de la alternativa más adecuada para nuestro proyecto.

- **Coste económico:** es un aspecto clave a la hora de estudiar las diferentes alternativas. Nos interesa elegir aquella alternativa que nos ofrezca buenas prestaciones, pero que su coste económico sea lo más reducido posible.
- **Recursos humanos y materiales:** se refiere al conjunto de personas y de materiales que serán necesarios para la correcta realización del trabajo. Nos interesa aquella alternativa que nos garantice una alta fiabilidad en los resultados obtenidos, pero cuya necesidad de recursos humanos y materiales necesarios sea lo más baja posible.
- **Cantidad de resultados:** nos interesa elegir la alternativa que nos permita obtener una mayor cantidad de resultados. Además, también habrá que tener en cuenta el tiempo necesario para la obtención de dichos resultados. La alternativa que mejor combine estos dos aspectos será la adecuada.
- **Fiabilidad de los resultados:** es uno de los aspectos más importantes a la hora de realizar nuestra elección. Es muy importante obtener unos resultados de fiabilidad y calidad suficientes. La alternativa que nos ofrezca resultados de una fiabilidad y calidad elevadas será la adecuada para estudiar el funcionamiento del motor síncrono.

6.1.5. Elección de la solución

En este apartado se va a elegir la alternativa que más se adecúa a nuestro proyecto, explicando las diferentes razones por las cuales se ha elegido y realizando una comparación con las otras alternativas.

Para llevar a cabo la elección se ha construido una matriz de priorización. En dicha matriz se presentan los diferentes criterios de selección mencionados en el anterior apartado, cada uno de los cuales se ha ponderado en función de su importancia a la hora de tomar la decisión. El peso que se le ha asignado a cada criterio es el siguiente:

- Coste económico: 40%.
- Recursos humanos y materiales: 10%.
- Cantidad de resultados: 20%.
- Fiabilidad de resultados: 30%.

Además, se ha evaluado de 0 a 10 puntos cada una de las alternativas en los diferentes criterios. El 0 significa que la alternativa evaluada no nos interesa en absoluto en lo que se refiere al criterio que se está analizando. El 10 significa que la alternativa evaluada cumple completamente nuestras necesidades en lo que se refiere al criterio que se está analizando.

	Peso	Valoración sobre 10		
		Pruebas de campo	Modelos físicos a ecala	Simulación software
Coste económico	40%	0	4	9
Recursos humanos / materiales	10%	0	6	10
Cantidad de resultados	20%	5	5	10
Fiabilidad de resultados	30%	9	7	6
TOTAL		3,70	5,30	8,40

Tabla 1. Matriz de priorización de alternativas.

Analizando los resultados obtenidos mediante la matriz de priorización, se concluye que la simulación software es la alternativa más adecuada para la realización de nuestro proyecto. Podemos observar como la simulación software es la alternativa

que mayor cantidad de resultados nos ofrece, necesitando menor cantidad de recursos humanos y materiales y requiriendo una inversión necesaria prácticamente nula. Aunque es la alternativa que nos ofrece unos resultados menos fiables, este déficit de fiabilidad se compensa con las otras ventajas que nos ofrece.

En lo que a las otras alternativas se refiere, podemos observar como las pruebas de campo nos ofrecen una gran fiabilidad de resultados, pero requiere una gran inversión económica y una gran cantidad de recursos humanos y materiales. La construcción de un modelo físico a escala por su parte, también nos ofrece una mayor fiabilidad en los resultados que la simulación software, pero su coste económico es mayo y la cantidad de resultados que nos ofrece es menor.

Las puntuaciones finales de las alternativas han sido las siguientes:

- Pruebas de campo: 3,7.
- Modelo físico a escala: 5,3.
- **Simulación software: 8,4.**

6.2. Análisis para la elección de la herramienta software.

Después de haber justificado la elección de la simulación software para el desarrollo de nuestro proyecto, se van a analizar las diferentes herramientas software a las que tenemos acceso para el desarrollo de la interfaz que nos permite analizar el funcionamiento de los motores síncronos.

A continuación, se presentan las diferentes herramientas que se han analizado para la elección de esta.

6.2.1. PTC Mathcad

PTC Mathcad dispone de la facilidad de uso y la familiaridad de un cuaderno de ingeniería, junto con notación matemática de actualización instantánea, inteligencia de unidades y potentes prestaciones de cálculo. Este software para matemáticas de ingeniería permite presentar los cálculos con gráficos, texto e imágenes en un solo documento [8].

6.2.2. ATP-EMTP

ATP-EMTP (Alternative Transients Program - ElectroMagnetic Transients Program) es una herramienta software que permite llevar a cabo la simulación digital de fenómenos transitorios de naturaleza electromagnética y electromecánica con fines de diseño, especificaciones de equipos o definición de parámetros eléctricos

fundamentales. Es una herramienta especialmente diseñada para analizar los diferentes elementos que componen un sistema eléctrico. [9].

6.2.3. PSCAD/EMTDC

PSCAD/EMTDC es una herramienta de simulación para el análisis de sistemas de energía eléctrica, con capacidad para modelizar redes no lineales, sistemas de control y equipos de electrónica de potencia. EMTDC es el software de simulación y nos permite simular respuestas instantáneas en el dominio del tiempo. Por otra parte, PSCAD es un interfaz gráfico que permite al usuario diseñar gráficamente el circuito eléctrico, para después analizarlo, obtener resultados y manejar los datos en un entorno gráfico totalmente integrado [9].

6.2.4. MATLAB-GUIDE

Matlab es un programa optimizado para resolver problemas científicos y de ingeniería. El lenguaje de Matlab está basado en matrices por lo que resulta especialmente útil para el cálculo matricial. Además, nos ofrece la posibilidad visualizar gráficas para facilitar la visión de datos o resultados y obtener información a partir de ellos [10]. Su herramienta GUIDE (Graphical User Inerface Development Environment), nos ofrece la posibilidad de crear interfaces gráficas de usuario sencilla e interactivas, en las cuales podemos añadir una serie de elementos como botones, cajas de texto o gráficas a los que asignar una función concreta.

6.2.5. Criterios de selección

En este apartado se exponen los criterios que se han tenido en cuenta a la hora de elegir una herramienta software para el desarrollo de nuestra interfaz gráfica:

- **Flexibilidad:** este aspecto hace referencia a la posibilidad de realizar cambios en el diseño de nuestra interfaz sin tener que modificar la interfaz completa. Nos interesa aquella herramienta que nos facilite la realización de modificaciones sin exigirnos realizar grandes cambios en el diseño de nuestra interfaz.
- **Tiempo de simulación:** este aspecto hace referencia al tiempo que nuestra interfaz empleará a la hora de realizar los diferentes cálculos y funciones para los que se ha programado. Nos interesa la herramienta que nos ofrezca los tiempos de simulación más bajos.
- **Simplicidad del software:** nos interesa aquella herramienta que nos permita desarrollar nuestra interfaz en un entorno de trabajo sencillo, que nos permita trabajar de manera intuitiva.

- **Posibilidad de interfaz:** es el criterio más importante de todos los que se han tenido en cuenta. Nos interesa aquella herramienta que nos ofrezca la posibilidad de desarrollar una interfaz gráfica que se adecúe a nuestras necesidades.
- **Coste económico:** hace referencia al coste que nos supondrá la herramienta software seleccionada. Nos interesa aquella herramienta que nos ofrezca un mayor número de prestaciones con el coste más bajo posible.

6.2.6. Elección de la herramienta

En este apartado se va a seleccionar la herramienta software que más nos conviene para el desarrollo de nuestra interfaz, argumentando la elección y comparando la herramienta seleccionada con las demás.

Para facilitar la comparación entre las diferentes opciones se ha hecho uso de una matriz de priorización. En ella se han incluido los diferentes criterios de selección, asignando a cada uno de ellos un peso específico en función de su importancia. El peso asignado a cada criterio es el siguiente:

- Flexibilidad: 20%.
- Tiempo de simulación: 10%.
- Simplicidad del software: 10%.
- Posibilidad de interfaz: 40%.
- Coste económico: 20%.

El criterio al que más peso se le ha asignado ha sido a la posibilidad de la herramienta para realizar una interfaz gráfica. Además, a la hora de realizar la matriz de priorización se ha evaluado de 0 a 10 puntos cada una de las herramientas en cada uno de los diferentes criterios. El 0 significa que la herramienta en cuestión no es de nuestro interés, mientras que el 10 indica que cumple perfectamente con nuestras necesidades.

	Peso	Valoración			
		Mathcad	ATP-EMTP	PSCAD/EMTDC	MATLAB - GUIDE
Flexibilidad	20%	10	10	10	10
Tiempo de simulación	10%	4	0	4	5
Simplicidad del software	10%	6	1	5	9
Posibilidad de interfaz	40%	0	0	6	10
Coste económico	20%	5	10	5	5
TOTAL		4,00	4,10	6,30	8,40

Tabla 2. Matriz de priorización de herramientas software.

A la vista de los resultados obtenidos mediante la matriz de priorización, se puede concluir que la herramienta software más adecuada para el desarrollo de nuestra interfaz gráfica es MATLAB, y más en concreto su herramienta GUIDE. Podemos observar cómo MATLAB es la herramienta que mayor posibilidad de realizar la interfaz nos ofrece, junto con una flexibilidad y una simplicidad muy altas. Por otro lado, aunque no es la que más nos interesa desde el punto de vista económico, es la herramienta que mejor combina unas buenas prestaciones con un coste económico asumible.

En lo que respecta a las otras herramientas, podemos destacar que PSCAD/EMTDC es la única que también nos ofrece la opción de desarrollar una interfaz gráfica, pero en los demás aspectos no se adecúa a nuestras necesidades de la manera que lo hace MATLAB.

Las puntuaciones de cada una de las herramientas han sido las siguientes:

- Mathcad: 4.
- ATP-EMTP: 4,10.
- PSCAD/EMTDC: 6,30.
- **MATLAB-GUIDE: 8,40.**

7. DESCRIPCIÓN DE LA SOLUCIÓN

Tras decidir que MATLAB es la herramienta software más adecuada para llevar a cabo nuestra interfaz gráfica, en este apartado se describe el desarrollo de dicha interfaz, explicando los diferentes módulos en los que se ha dividido la misma y especificando las diferentes funciones que tienen cada uno de los módulos. Además, la interfaz se ha creado mediante la herramienta GUIDE que nos proporciona MATLAB, por lo que también se realizará una breve introducción de esta herramienta.

7.1. Introducción a GUIDE

GUIDE es una herramienta incluida dentro de MATLAB que nos permite desarrollar interfaces gráficas de usuario de una forma sencilla e intuitiva.

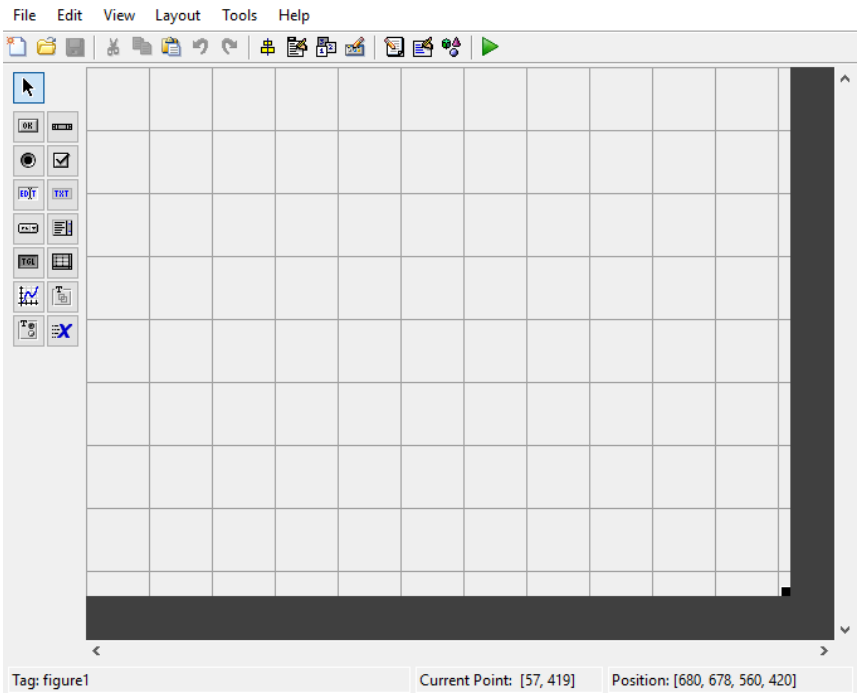

Figura 20. Entorno de trabajo GUIDE.

En la imagen se muestro el entorno de trabajo de GUIDE. En la parte izquierda podemos observar los diferentes elementos que podemos añadir a nuestra interfaz, simplemente arrastrándolos hacia el panel cuadriculado que vemos en la imagen. Además, GUIDE también nos permite ir ejecutando la interfaz a medida que la vamos desarrollando, permitiéndonos ver cómo va evolucionando su aspecto y su funcionalidad. En nuestra interfaz gráfica, los elementos utilizados han sido los siguientes:

- **Static text**: elemento destinado a mostrar un texto concreto, durante la ejecución de la interfaz este texto no se puede modificar.
- **Edit text**: elemento sobre el que se puede escribir texto, durante la ejecución de la interfaz se puede modificar.
- **Push button**: elemento que nos permite realizar la acción para la que ha sido programado cuando se ejecuta, durante la ejecución de la interfaz se pulsa el pushbutton y entonces se ejecuta la acción pertinente.
- **Radio button**: indica una opción que puede ser seleccionada.
- **Button group**: panel que agrupa diferentes radio buttons. Cuando se selecciona uno de ellos los demás se inhabilitan.
- **Panel**: elemento destinado a agrupar diferentes elementos dentro de la interfaz. Nos permite dividir la interfaz en diferentes módulos.
- **Pop-up menu** : nos muestra una lista de opciones que podemos seleccionar. Al seleccionar una de ellas se ejecuta la acción programada para dicha opción.
- **Axes**: elemento que nos permite introducir gráficas en la interfaz.

Cada uno de los elementos utilizados se pueden programar, asignándoles la función concreta que queremos que desarrollen dentro de la interfaz.

7.2. Interfaz gráfica de usuario

La interfaz desarrollada a lo largo de este proyecto nos permite calcular la curva de par y la curva en V de Mordey de un motor síncrono. Además, también nos permite calcular otro tipo de parámetros relacionados con el motor que pueden ser de gran interés. Ente estos parámetros podemos destacar el ángulo de par, la intensidad consumida por el motor, la velocidad de sincronismo o el factor de potencia (inductivo o capacitivo) con el que está trabajando el motor. Por otra parte, también nos permite visualizar, en las curvas antes mencionadas, el punto de funcionamiento en el que se encuentra el motor ante una situación de carga concreta. A continuación, se muestra una imagen que muestra el aspecto de nuestra interfaz gráfica:

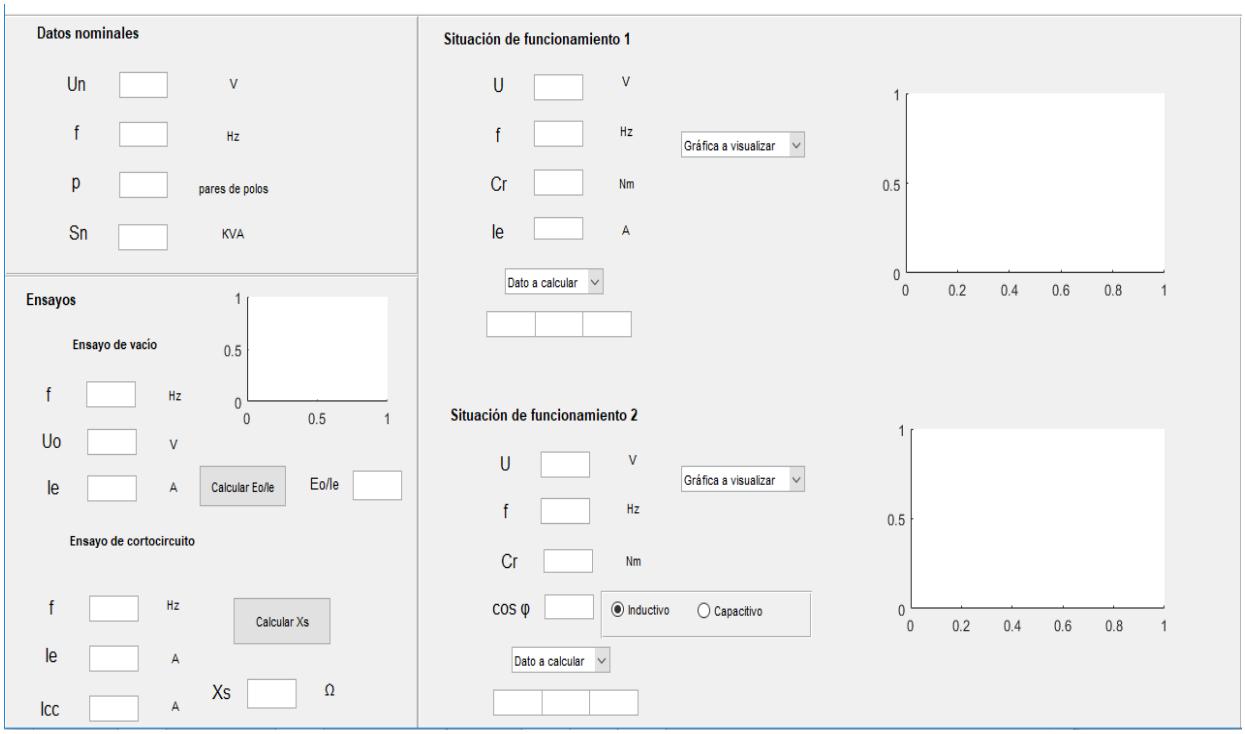

Figura 21. Interfaz gráfica de usuario.

La interfaz está compuesta por tres módulos claramente diferenciados. El primer módulo (parte superior izquierda) está formado por los datos nominales de la máquina. En este módulo se introducen los datos que encontramos en la placa característica del motor, entre los que destacan la tensión y frecuencia nominales, el número de pares de polos o la potencia nominal. El segundo módulo (parte inferior izquierda) recoge los datos de los ensayos de la máquina síncrona. En este módulo podemos diferenciar dos partes, una destinada al ensayo de vacío y otra destinada al ensayo de cortocircuito. En la parte destinada al ensayo de vacío, se recogen los datos de la frecuencia, la tensión y la intensidad de excitación a la que se realiza el ensayo, para poder calcular la relación existente entre la tensión de vacío y la intensidad de excitación. Además, también se realiza una gráfica que nos muestra dicha relación sin que se llegue a la saturación. En la parte destinada al ensayo de cortocircuito, se recogen los datos de la frecuencia, la intensidad de excitación y de la intensidad de cortocircuito, para poder calcular la reactancia síncrona de la máquina. El tercer módulo (parte derecha) está destinado al cálculo de las curvas y los diferentes parámetros del motor síncrono frente a dos situaciones de funcionamientos distintas. En ambas situaciones se tiene la opción de calcular las curvas características del motor, uno de los objetivos principales de este trabajo. Además, dependiendo de la situación en la que nos encontremos, podremos

calcular diferentes parámetros del motor como la intensidad, el ángulo de par, el factor de potencia, etc.

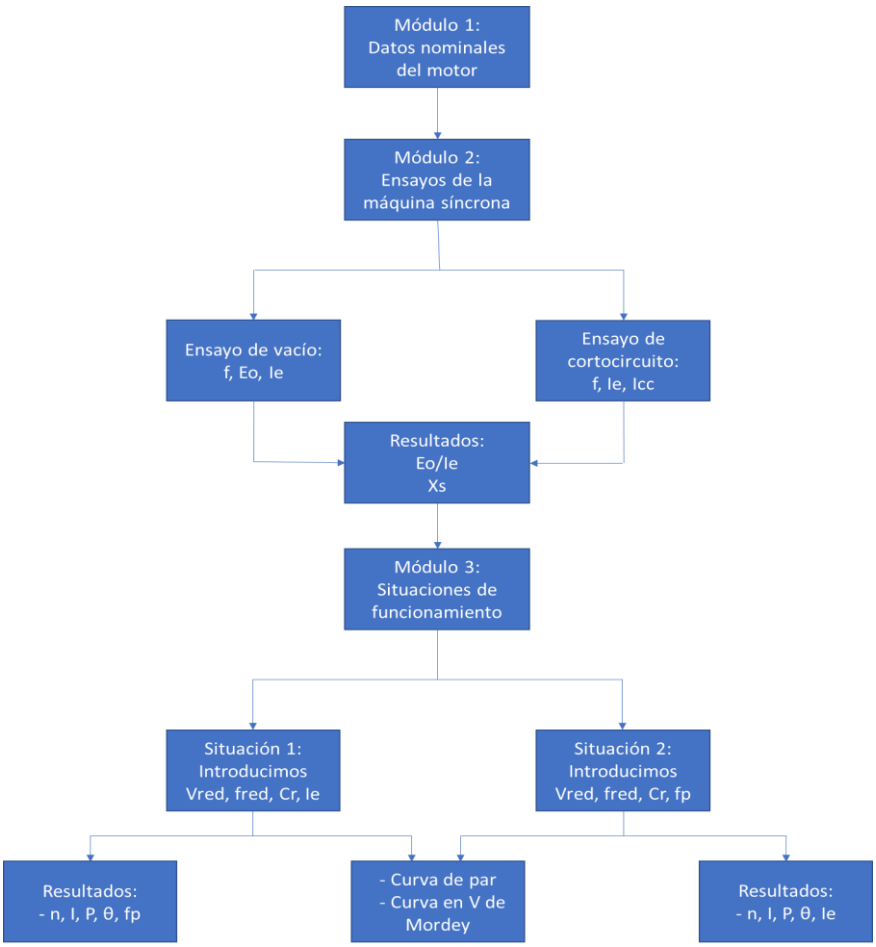

Figura 22. Diagrama de flujo de la interfaz.

7.2.1. Módulo 1: Datos nominales del motor síncrono

En este primer módulo se introducen los datos de tensión, frecuencia y potencia nominales del motor, además del número de pares de polos.

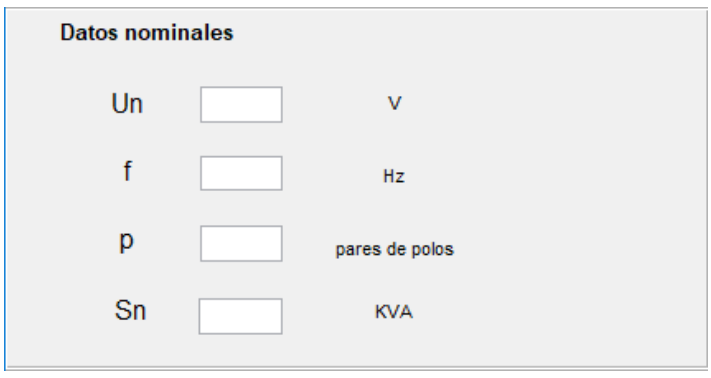

Figura 23. Módulo 1 de la interfaz.

En este panel, no se realiza ningún cálculo matemático, simplemente se introducen los datos nominales del motor en los edit text para su posterior uso a la hora de realizar cálculos en módulos posteriores. Para poder hacer uso de estos datos, el GUIDE nos proporciona el comando *get*, mediante el cual podemos guardar en una variable el dato introducido en el edit text. Para poder utilizar estos datos a la hora de realizar cálculos matemáticos, es necesario utilizar el comando *str2num*, que nos convierte el dato recogido en una variable numérica.

Se muestra un ejemplo de la recogida del dato del número de pares de polos y su conversión de string a número:

```
polos=str2num(get(handles.polos,'String'));
```

Se ha guardado en una variable llamada *polos* el dato introducido en la celda correspondiente al número de pares de polos del motor, para posteriormente convertir ese dato en un dato numérico mediante al comando *str2num*.

7.2.2. Módulo 2: Ensayos de la máquina síncrona

En este segundo módulo se recogen los datos de los ensayos de vacío y de cortocircuito de la máquina trabajando como generador. Mediante dichos ensayos obtendremos los datos de la reactancia síncrona y de la relación Eo/Ie.

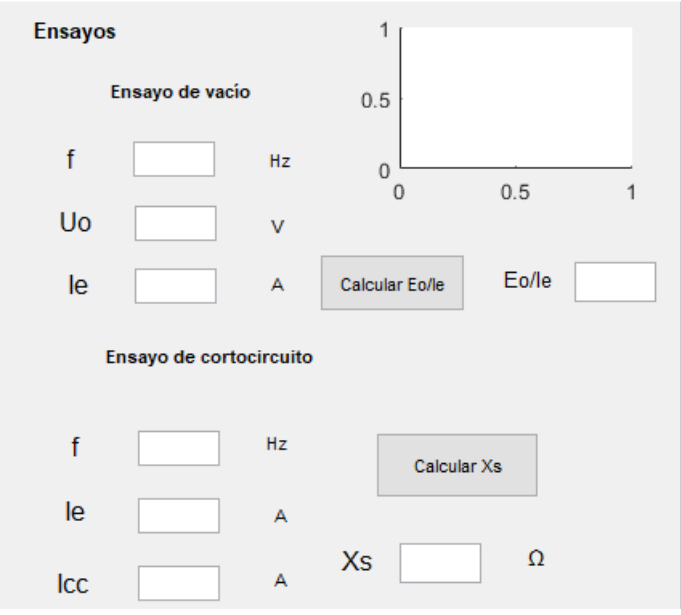

Figura 24. Módulo 2 de la interfaz.

Como se puede observar hay una primera parte destinada al ensayo de vacío, en la que se recogen los datos de la frecuencia, la tensión y la intensidad de excitación empleadas durante el ensayo. Una vez introducidos los datos, estos se guardan en diferentes variables y se transforman a datos numéricos para su posterior uso a la hora de realizar cálculos. Mediante el push button llamado 'Calcular Eo/Ie', se calcula la relación existente entre la tensión de vacío y la intensidad de excitación, además de realizar la gráfica que nos muestra dicha relación ante diferentes valores de la intensidad de excitación. Se adjunta el fragmento de código correspondiente a la programación del push button:

```
U0=str2num(get(handles.Uo,'String'));
Iee=str2num(get(handles.Iee,'String'));
E0=U0/sqrt(3);
rela=E0/Iee;
set(handles.relacion,'String',rela);
IE=linspace(0,20,50)
E00=rela*IE
plot(handles.axes1,IE,E00);
xlabel(handles.axes1,'Ie(A)');
ylabel(handles.axes1,'Eo(V)');
```

En lo que respecta a la segunda parte del módulo, ésta está destinada a recoger los datos del ensayo de cortocircuito. Los datos a introducir son la frecuencia, la intensidad de excitación y la intensidad de cortocircuito. De manera similar a el caso anterior, se convertirán los datos introducidos a variables numéricas, y mediante el push button llamado 'Calcular Xs' obtendremos la reactancia síncrona del motor. Se ha utilizado la expresión (3) para el cálculo de la reactancia síncrona. Se adjunta el código de programación del push button:

```
rel=str2num(get(handles.relacion,'String'));
ie=str2num(get(handles.Ie,'String'));
E0=rel*ie;
icc=str2num(get(handles.Icc,'String'));
Xs=E0/icc;
set(handles.edit10,'String',Xs);
```

Por último, se muestran dos imágenes del funcionamiento de cada una de las dos partes del módulo:

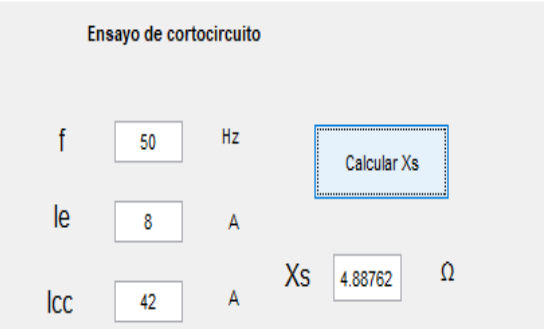

Figura 25. Ejemplo ensayo de cortocircuito.

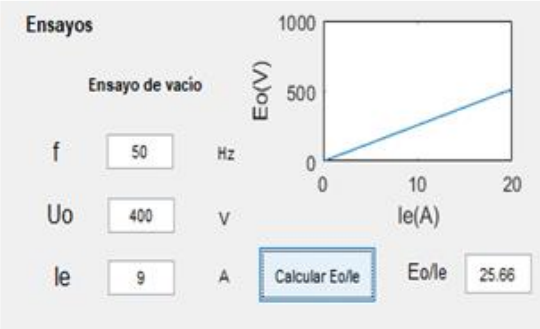

Figura 26. Ejemplo ensayo de vacío.

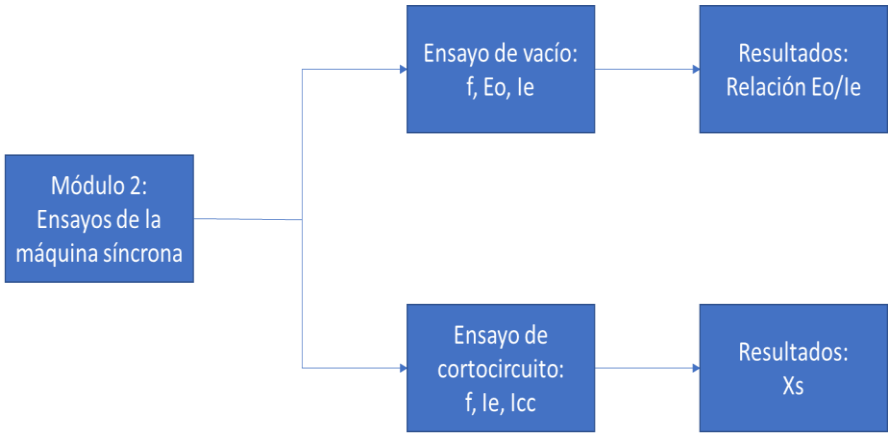

Figura 27. Diagrama de flujo módulo 2.

7.2.3. Módulo 3: Situaciones de funcionamiento del motor síncrono

En este tercer módulo se recogen dos situaciones de funcionamiento diferentes, en las que, ante unos datos de entrada, se calculan diferentes parámetros del motor y se grafican la curva de par y la curva en V de Mordey.

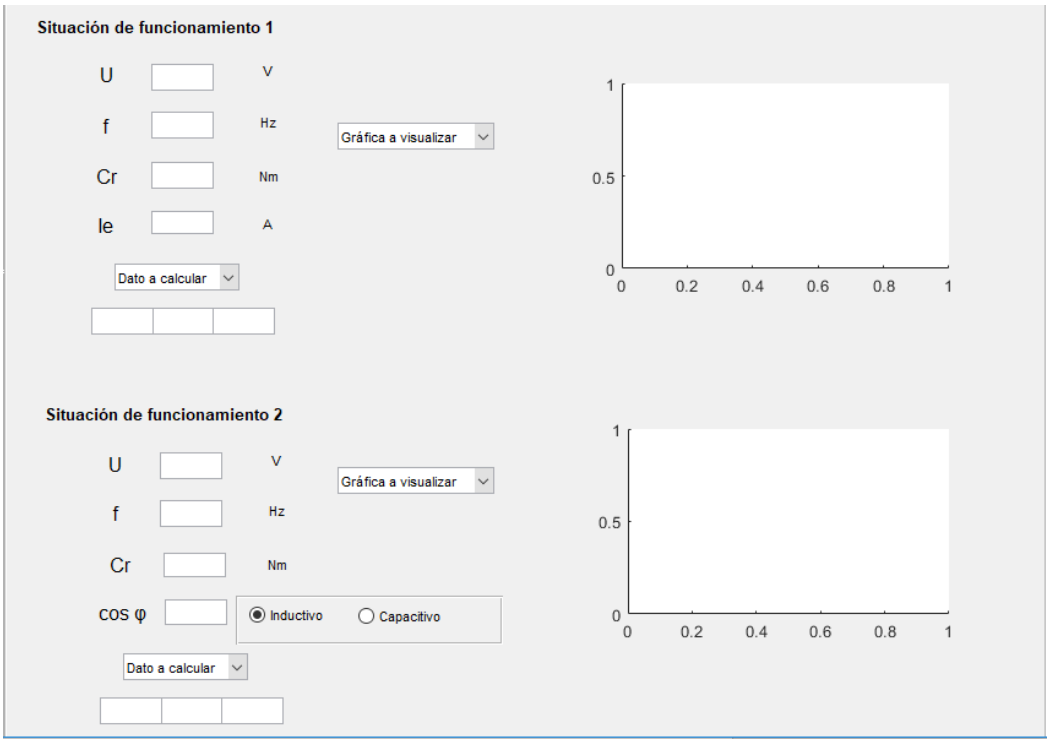

Figura 28. Módulo 3 de la interfaz.

En las dos situaciones de funcionamiento podemos observar tres zonas claramente diferenciadas. Una primera zona en la que se nos piden unos datos de entrada, una segunda zona en la que se nos muestra el resultado del parámetro que queremos calcular, y una tercera zona en la que podemos realizar la gráfica de la curva de par o de la curva en V de Mordey. Se han utilizado las expresiones (1), (6), (7) y (8) para el cálculo de los diferentes parámetros.

En lo que respecta a la primera situación de funcionamiento, los datos de entrada que se nos exigen son la tensión de la red, la frecuencia de la red, el par resistente al que tendrá que hacer frente el motor y la intensidad de excitación a la que se ha excitado el motor. Mediante el tratamiento de estos datos de entrada, junto con los datos de los ensayos y los datos nominales del motor, la interfaz nos permite calcular diferentes parámetros. Para poder elegir qué parámetro queremos calcular, se ha utilizado un pop-up menu que nos despliega una lista con todos los parámetros que nos permite calcular la interfaz. Por otra parte, en lo que respecta a las curvas características del motor, también se ha utilizado un pop-up menu que nos permita elegir la curva que queremos visualizar. A continuación, se adjunta un pequeño fragmento del código correspondiente a los dos pop-up menu, y también imágenes que nos muestran el despliegue de estos elementos:

- Fragmento de código del primer pop-up menu (se muestra la programación para el cálculo de la velocidad de sincronismo y la potencia):

```
function popupmenu1_Callback(hObject, eventdata, handles)
contenido=get(hObject,'String');
a=get(hObject,'Value');
texto=contenido(a);
switch cell2mat(texto)
    case 'Velocidad de sincronismo'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
      set(handles.resultado,'String',v);
      set(handles.dato,'String','n');
      set(handles.unidades,'String','rpm');
    case 'Potencia'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60;
        set(handles.resultado,'String',P);
        set(handles.dato,'String','Potencia');
        set(handles.unidades,'String','W');
```

- Fragmento de código del segundo pop-up menu (se muestra la programación de la curva de par):

```
contenido=get(hObject,'String');
b=get(hObject,'Value');
texto=contenido(b);
switch cell2mat(texto)
    case 'Curva de par'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
```

```
par=str2num(get(handles.Cm,'String'));
P=(2*pi*v*par)/60
Xs=str2num(get(handles.edit10,'String'))
rel=str2num(get(handles.relacion,'String'));
ie=str2num(get(handles.Ie2,'String'));
E0=rel*ie
V=str2num(get(handles.Vred,'String'))
angulo1=rad2deg(asin((P*Xs)/(V*sqrt(3)*E0)))
angulo=degtorad(angulo1);
angulo=linspace(0,pi,100);
y=sqrt(3)*V*E0*sin(angulo)/Xs
plot(handles.axes2,rad2deg(angulo),y,angulo1,P,'*');
xlabel(handles.axes2,'Ángulo de par');
ylabel(handles.axes2,'Potencia activa');
title(handles.axes2,'Curva de par');
grid(handles.axes2,'on');
```

- Imágenes de los pop-up menu:

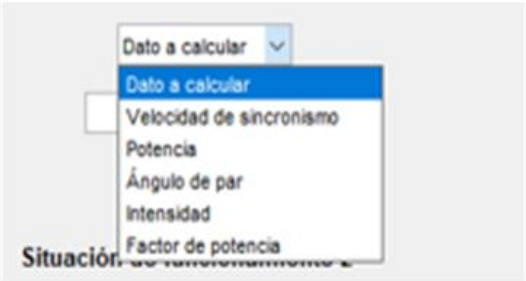

Figura 29. Pop-up menu desplegado (1).

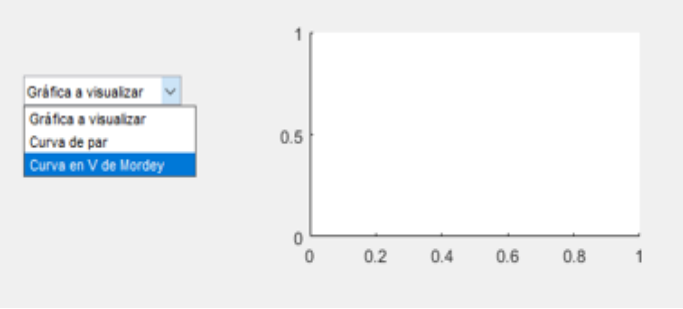

Figura 30. Pop-up menu desplegado (2).

En lo que respecta a la segunda situación de funcionamiento, los datos de entrada que se nos piden son la tensión de red, la frecuencia de red, el par resistente y el factor de potencia, permitiéndonos indicar si este último es inductivo o capacitivo. Al igual que en la primera situación, disponemos de un primer pop-up menu que nos permite calcular diferentes parámetros del motor, y de un segundo pop-up menu que nos permite seleccionar la curva que queremos graficar. A continuación, se adjunta un pequeño fragmento del código correspondiente a ambos pop-up menu, junto con imágenes de ambos elementos desplegados:

- Fragmento de código del primer pop-up menu (se muestra la programación para el cálculo de la intensidad):

```
case 'Intensidad'
        valor=get(handles.radiobutton1,'Value');
        if valor==1
            polos=str2num(get(handles.polos,'String'));
            f=str2num(get(handles.frecuencia5,'String'));
            v=(60*f)/polos;
            par=str2num(get(handles.Cm2,'String'));
            P=(2*pi*v*par)/60;
            Un=str2num(get(handles.Vred2,'String'));
            fp=str2num(get(handles.fp2,'String'));
            I=P/(sqrt(3)*Un*fp);
            angulo=acos(fp);
            I1=I*cos(-angulo)+i*I*sin(-angulo)
            set(handles.resultado2,'String',abs(I1));
            set(handles.dato2,'String','Intensidad');
            set(handles.unidades2,'String','A');
        else

            polos=str2num(get(handles.polos,'String'));
            f=str2num(get(handles.frecuencia5,'String'));
            v=(60*f)/polos;
            par=str2num(get(handles.Cm2,'String'));
            P=(2*pi*v*par)/60;
```

```
        Un=str2num(get(handles.Vred2,'String'));
        fp=str2num(get(handles.fp2,'String'));
        I=P/(sqrt(3)*Un*fp);
        angulo=acos(fp);
        I1=I*cos(angulo)+i*I*sin(angulo)
        set(handles.resultado2,'String',abs(I1));
        set(handles.dato2,'String','Intensidad');
        set(handles.unidades2,'String','A');
    end
```

- Fragmento de código del segundo pop-up menu (se muestra la programación de la curva en V de Mordey):

```
case 'Curva en V de Mordey'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60
        Xss=str2num(get(handles.edit10,'String'));
        Vv=str2num(get(handles.Vred2,'String'));
        E01=P*Xss/(sqrt(3)*Vv)
        rel=str2num(get(handles.relacion,'String'));
        Iemin=E01/rel
        Iemin1=round(Iemin,1)
        Iemin2=Iemin1*10
        i_f= (Iemin2:1:180)/10;
        b=(180-Iemin2)+1
        i_a=zeros(1,b);
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60;
        Un=str2num(get(handles.Vred2,'String'));
```

```
fp=str2num(get(handles.fp2,'String'));
I=P/(sqrt(3)*Un*fp)
angulo=acos(fp)
valor=get(handles.radiobutton1,'Value');
if valor==1
I1=I*cos(-angulo)+i*I*sin(-angulo)
Xs=str2num(get(handles.edit10,'String'));
Xs=i*Xs
reac=abs(Xs)
V=Un/sqrt(3)+i*0
E1=V-(I1*Xs)
rel=str2num(get(handles.relacion,'String'));
for ii=1:b
    E2=rel*i_f(ii);
    angulo2=asin((abs(E1)/abs(E2))*sin(angle(E1)));
    E2=E2*(cos(angulo2)+i*sin(angulo2));
    i_a(ii)=(V-E2)/(Xs);
end
iee=abs(E1)/rel
plot(handles.axes5,i_f,abs(i_a),iee,abs(I1),'*');
axis([0 20 0 70]);
xlabel(handles.axes5,'Ie');
ylabel(handles.axes5,'I');
title(handles.axes5,'Curva en V de mordey');
grid(handles.axes5,'on');
else
I1=I*cos(angulo)+i*I*sin(angulo)
Xs=str2num(get(handles.edit10,'String'));
Xs=i*Xs
reac=abs(Xs)
V=Un/sqrt(3)+i*0
E1=V-(I1*Xs)
rel=str2num(get(handles.relacion,'String'));
for ii=1:b
```

```
            E2=rel*i_f(ii);
            angulo2=asin((abs(E1)/abs(E2))*sin(angle(E1)));
            E2=E2*(cos(angulo2)+i*sin(angulo2));
            i_a(ii)=(V-E2)/(Xs);
        end
        iee=abs(E1)/rel;
        plot(handles.axes5,i_f,abs(i_a),iee,abs(I1),'*');
        axis([0 20 0 70]);
        xlabel(handles.axes5,'Ie');
        ylabel(handles.axes5,'I');
        title(handles.axes5,'Curva en V de mordey');
        grid(handles.axes5,'on');
        end
```

- Imágenes de los pop-up menu:

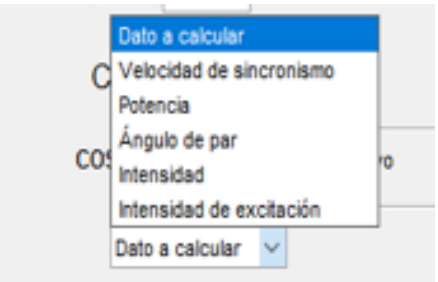

Figura 31. Pop-up menu desplegado (3).

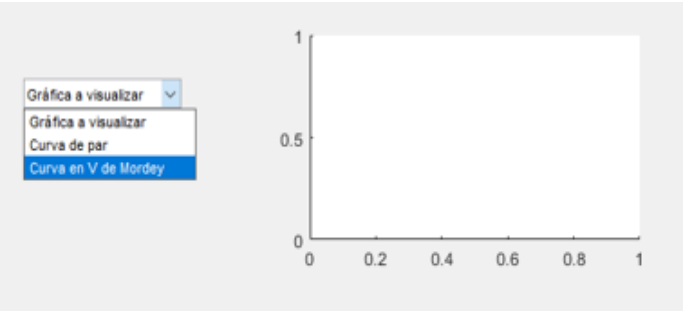

Figura 32. Pop-up menu desplegado (4).

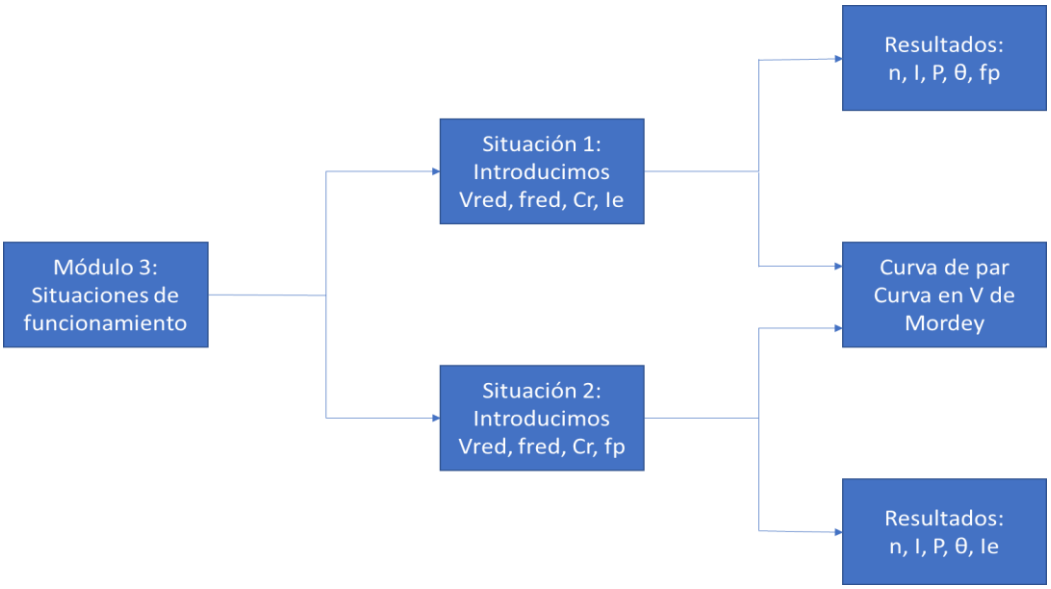

Figura 33. Diagrama de flujo del módulo 3.

El código completo de todos los pop-up menu, junto con el de todos los demás elementos de la interfaz se adjunta en el anexo.

7.2.4. Mensajes de advertencia y error

Además de todo lo mencionado con anterioridad, la interfaz está diseñada para mostrar ciertos mensajes de error o advertencia cuando la situación de funcionamiento de nuestro motor no es la adecuada.

Por una parte, a la hora de introducir en el módulo 3 los datos del par resistente, la aplicación está programada para avisarnos si con ese valor del par resistente introducido superamos el valor de la potencia nominal de la máquina. A continuación, se muestra el código de programación del mensaje de error:

```
sn=str2num(get(handles.Sn,'String'));
Sn=sn*1000;
par=str2num(get(hObject,'String'));
polos=str2num(get(handles.polos,'String'));
f=str2num(get(handles.frecuencia4,'String'));
v=(60*f)/polos;
P=(2*pi*v*par)/60;
if P>Sn
errordlg('La potencia supera la potencia nominal de la màquina','ERROR');
end
```

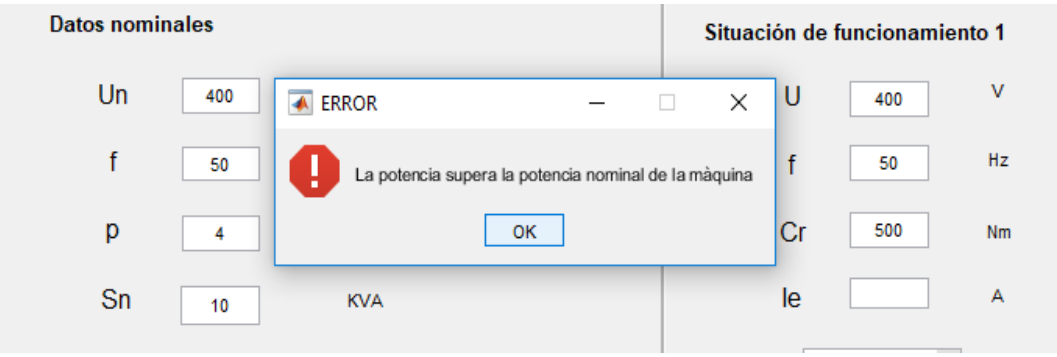

Figura 34. Ejemplo del mensaje de error.

Por otra parte, nuestra interfaz también está programada para mostrarnos un mensaje de advertencia cuando se dan dos situaciones concretas:

- En la primera situación de funcionamiento la intensidad de excitación no es suficiente para hacer frente al par resistente introducido.

```
polos=str2num(get(handles.polos,'String'));
f=str2num(get(handles.frecuencia4,'String'));
v=(60*f)/polos;
par=str2num(get(handles.Cm,'String'));
P=(2*pi*v*par)/60
Xss=str2num(get(handles.edit10,'String'));
V=str2num(get(handles.Vred,'String'));
E01=P*Xss/(sqrt(3)*V)
rel=str2num(get(handles.relacion,'String'));
Iemin=E01/rel
Ie22=str2num(get(hObject,'String'));
if Ie22<Iemin
warndlg('La Ie debe ser mayor','ADVERTENCIA');
end
```

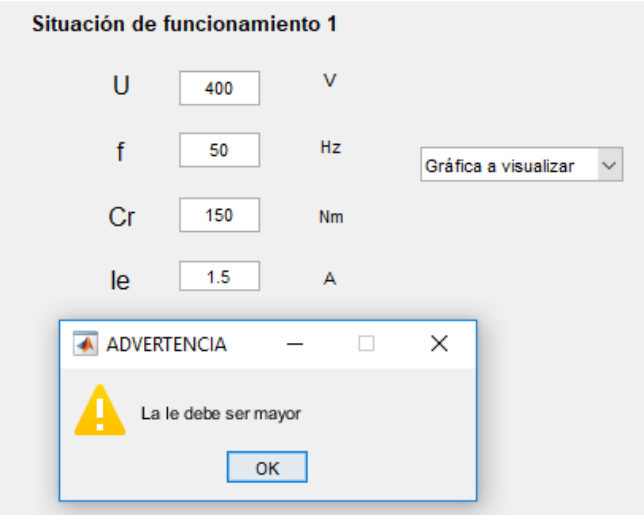

Figura 35. Ejemplo del mensaje de advertencia (1).

- En la segunda situación de funcionamiento el factor de potencia (caso inductivo) no es suficiente para hacer frente al par resistente:

```
valor=get(handles.radiobutton1,'Value');
if valor==1
polos=str2num(get(handles.polos,'String'));
f=str2num(get(handles.frecuencia5,'String'));
v=(60*f)/polos;
par=str2num(get(handles.Cm2,'String'));
P=(2*pi*v*par)/60
Xss=str2num(get(handles.edit10,'String'));
V=str2num(get(handles.Vred2,'String'));
E01=P*Xss/(sqrt(3)*V)
V1=V/sqrt(3)
E02=-i*E01
Xs=i*Xss
I1=(E02-V1)/Xs
I2=abs(I1)
fpmin=P/(3*V1*I2)
fp=str2num(get(hObject,'String'));
if fp<fpmin
warndlg('El factor de potencia debe ser mayor','ADVERTENCIA');
end
```

```
end
```

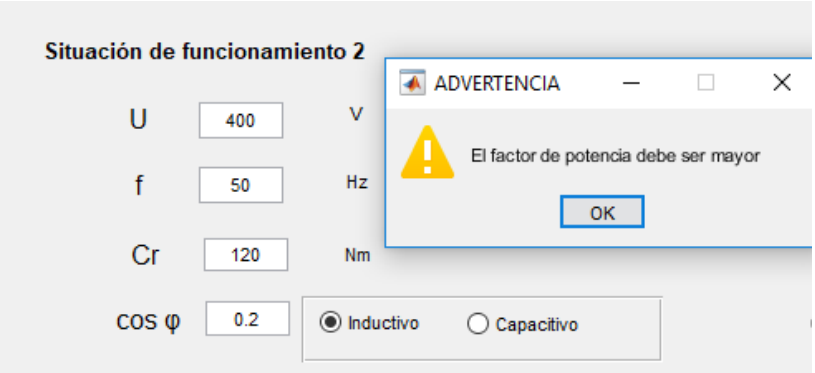

Figura 36. Ejemplo del mensaje de advertencia (2).

8. EJEMPLOS DE APLICACIÓN

En este apartado se va a mostrar el funcionamiento de la interfaz gráfica mediante dos ejemplos prácticos. Para ello, se resolverán 2 problemas que se adecúen a cada una de las dos situaciones de funcionamiento, y se irán resolviendo dichos problemas paso a paso.

8.1. Ejemplo 1

Se parte de los siguientes datos de los ensayos de vacío y cortocircuito de un generador síncrono:

- **Ensayo de vacío:**
  - **U_O: 400 V**
  - **I_E: 8,37 A**

- **Ensayo de cortocircuito:**
  - **I_{CC}: 59,61 A**
  - **I_E: 8 A**

Se hace funcionar la máquina ahora como motor conectado a una red de 400 V y 50 Hz, con una I_E de 6 A y un par resistente de 250 Nm. El motor tiene 6 pares de polos y una potencia nominal de 25 KVA. Se desprecian los efectos de la saturación, las pérdidas y la resistencia del inducido.

El primer paso para empezar la resolución del problema es calcular la relación E_0/I_E y la reactancia síncrona X_S mediante los datos de los ensayos de vacío y cortocircuito de la máquina trabajando como generador. Para ello, primero se realizarán los cálculos manualmente, y después se comprobarán los resultados con nuestra interfaz. En lo que a la interfaz se refiere, se introducirán los datos nominales del motor y los de los ensayos, y ya podremos calcular los parámetros mencionados.

- Ensayo de vacío:

$$\frac{E_0}{I_E} = \frac{\frac{400}{\sqrt{3}}}{8{,}37} = 27{,}5914$$

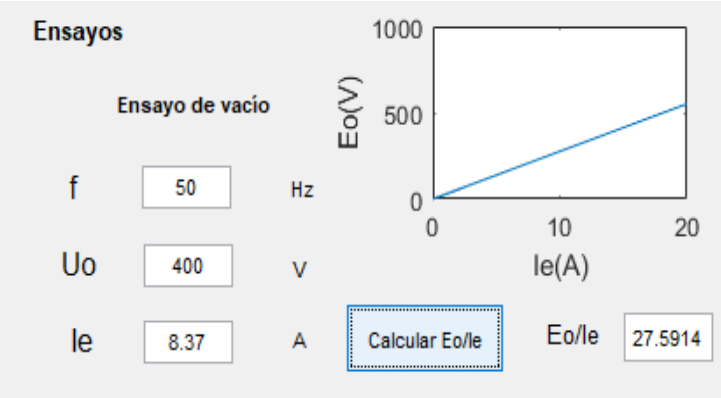

Figura 37. Cálculo relación E_0/I_E.

- Ensayo de cortocircuito:

$$\frac{E_{0cc}}{I_E} = \frac{E_{0cc}}{8} = 27{,}5914 \rightarrow E_{0cc} = 220{,}7312\ V$$

$$X_S = \frac{E_{0cc}}{I_{cc}} = \frac{220{,}7312}{59{,}61} = 3{,}7029\ \Omega$$

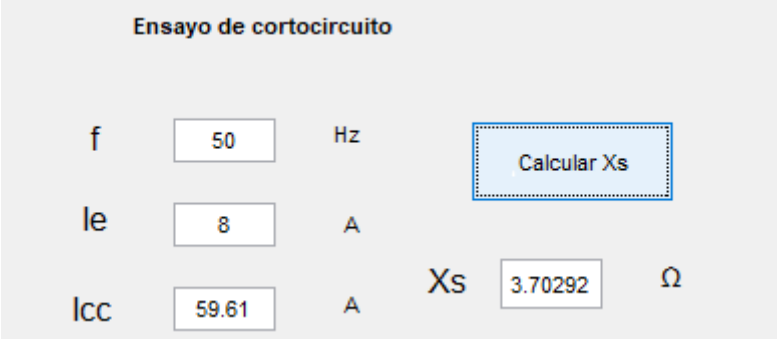

Figura 38. Cálculo X_S.

Una vez calculados los parámetros correspondientes a los ensayos de vacío y cortocircuito, se procederá a calcular el resto de los parámetros del motor (n, P, θ, I, fp). Para ello, junto con los datos nominales del motor, se introducirán los datos de entrada de la primera situación de funcionamiento. Al igual que en el paso anterior, se realizará el cálculo manual de dichos parámetros, y después se comprobará dicho resultado con el que nos proporciona la interfaz.

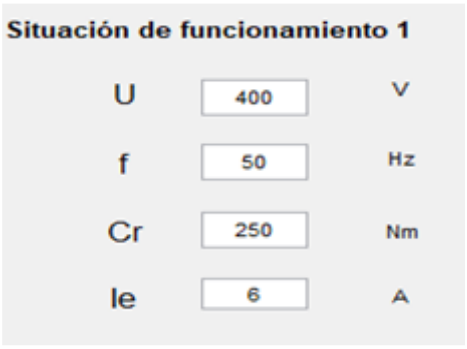

Figura 39. Datos de entrada situación 1.

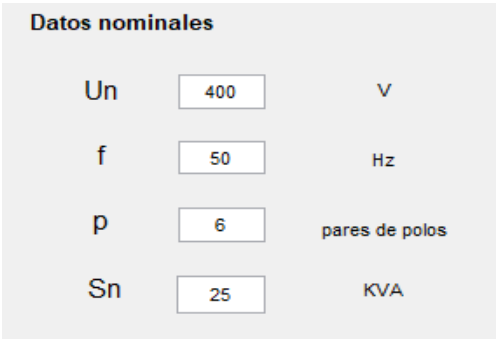

Figura 40. Datos nomines del motor.

- Velocidad de sincronismo:

$$n = \frac{60 \cdot f}{p} = \frac{60 \cdot 50}{6} = 500\ rpm.$$

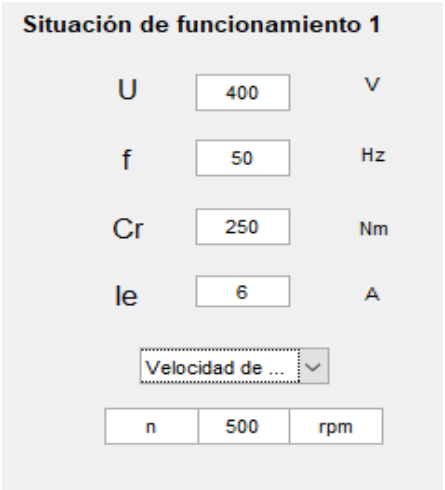

Figura 41. Cálculo de n.

- Potencia:

$$P = 2 \cdot \pi \cdot \frac{n}{60} \cdot Cr = 2 \cdot \pi \cdot \frac{500}{60} \cdot 250 = 13090\ W$$

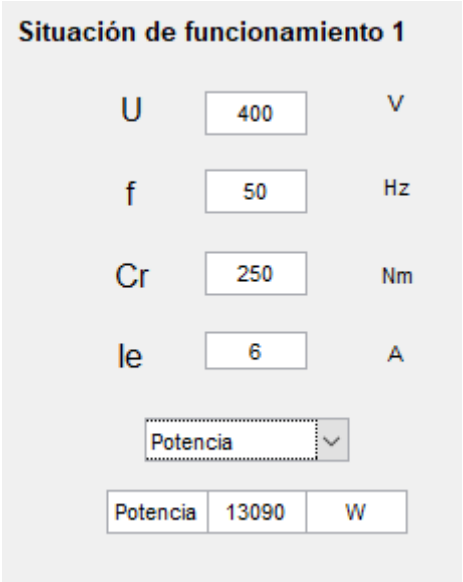

Figura 42. Cálculo de P.

- Ángulo de par:

$$P = 3 \cdot V \cdot \frac{E_0}{X_S} \cdot \sin(\theta) \rightarrow \theta = \sin^{-1}\left(\frac{P \cdot X_S}{E_0 \cdot 3 \cdot V}\right)$$

$$\frac{E_0}{I_E} = 27{,}5914 \rightarrow E_0 = 165{,}5484\ V$$

$$\theta = \sin^{-1}\left(\frac{13090 \cdot 3{,}7029}{165{,}5484 \cdot 3 \cdot \frac{400}{\sqrt{3}}}\right) = 24{,}999\ grados$$

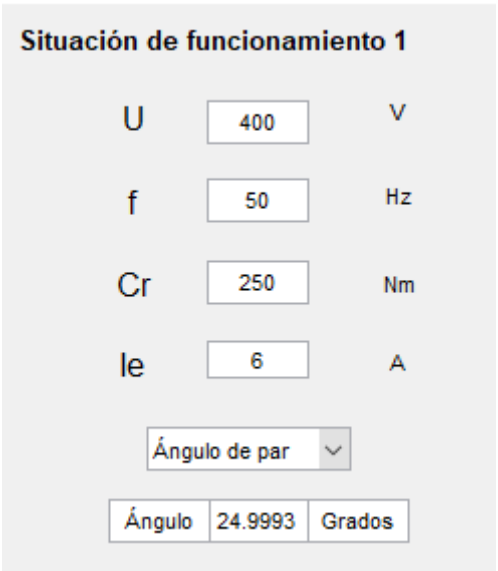

Figura 43. Cálculo de θ.

- Intensidad:

$$\underline{V} = \underline{E}_0 + \underline{I} \cdot (R + jX_S) \rightarrow \underline{I} = \frac{(\underline{V} - \underline{E}_0)}{(R + jX_S)}$$

$$\underline{I} = \frac{(\frac{400}{\sqrt{3}} - 165{,}5484 \angle -25^{\circ})}{(0 + j \cdot 3{,}7029)} = 28.885 \angle -49{,}147^{\circ} \; A$$

Figura 44. Cálculo de I.

- Factor de potencia:

$$\underline{I} = 28.885 \angle -49{,}147^{\circ} \; A \rightarrow fp = \cos(49.147) = 0{,}6541(inductivo)$$

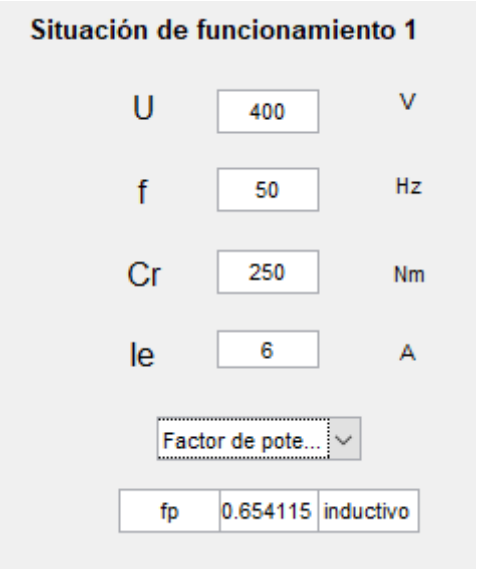

Figura 45. Cálculo del factor de potencia.

Por último, se representan la curva de par y la curva en V de Mordey del motor. Para ello, seleccionamos en nuestra interfaz la curva que queremos visualizar. En cada una de las curvas también se visualiza el punto de funcionamiento en el que nos encontramos.

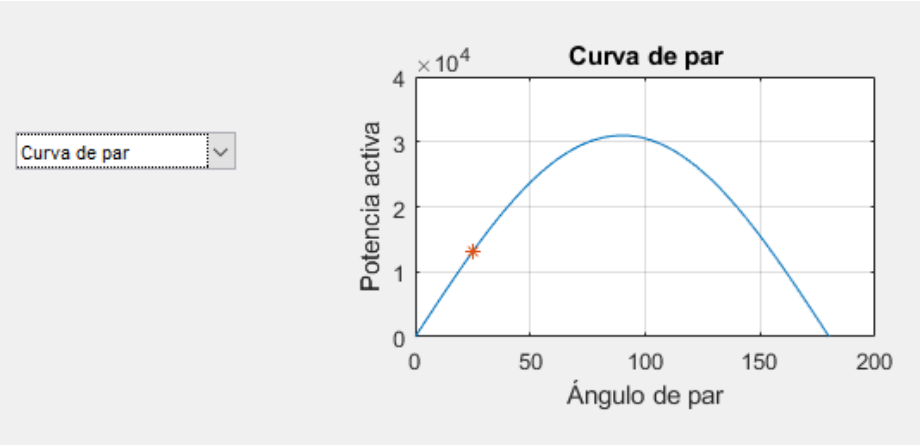

Figura 46. Curva de par.

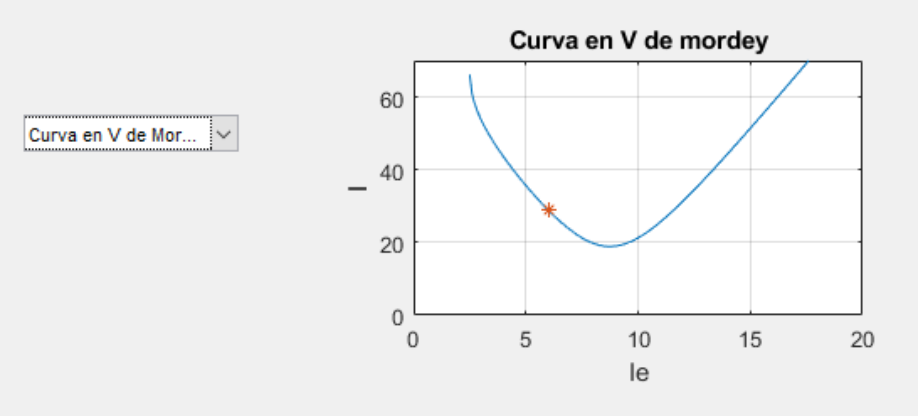

Figura 47. Curva en V de Mordey.

Para mostrar la sencillez y funcionalidad de la aplicación, se va a mostrar una situación en la que se supera la potencia nominal de la máquina y otra en la que la intensidad de excitación no es suficiente para el par que se quiere arrastrar.

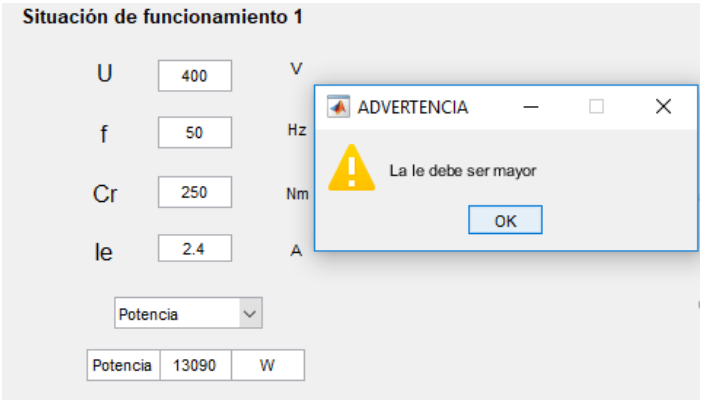

Figura 48. Mensaje de advertencia.

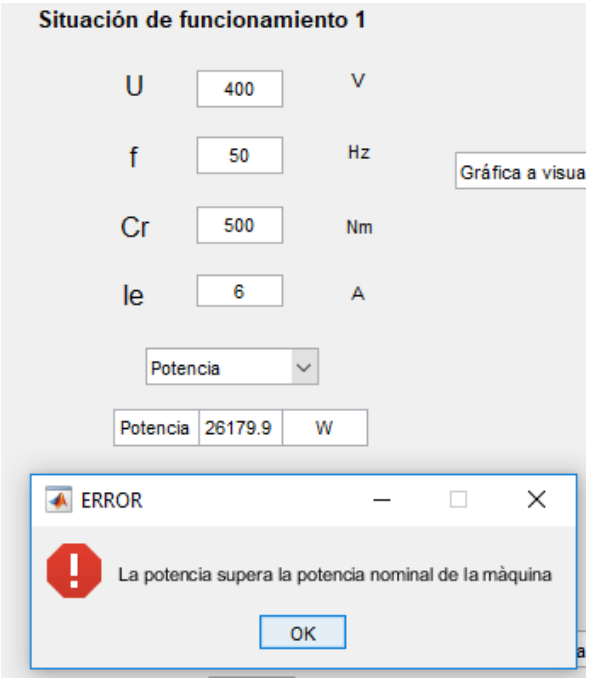

Figura 49. Mensaje de error.

8.2. Ejemplo 2

Se parte de los siguientes datos de los ensayos de vacío y cortocircuito de un generador síncrono:

- **Ensayo de vacío:**
  - **U_O: 3000 V**
  - **I_E: 12 A**

- **Ensayo de cortocircuito:**
  - **I_{CC}: 48,11 A**
  - **I_E: 5 A**

Se hace funcionar la máquina ahora como motor conectado a una red de 3000 V y 50 Hz, con un factor de potencia 0.9 inductivo y un par resistente de 1080 Nm. El motor tiene 2 pares de polos y una potencia nominal de 250 KVA.

El primer paso para empezar la resolución del problema es calcular la relación E_0/I_E y la reactancia síncrona X_S mediante los datos de los ensayos de vacío y cortocircuito de la máquina trabajando como generador. Para ello, primero se realizarán los cálculos manualmente, y después se comprobarán los resultados con nuestra interfaz. En lo que a la interfaz se refiere, se introducirán los datos nominales del motor y los de los ensayos, y ya podremos calcular los parámetros mencionados.

- Ensayo de vacío:

$$\frac{E_0}{I_E} = \frac{\frac{3000}{\sqrt{3}}}{12} = 144{,}338$$

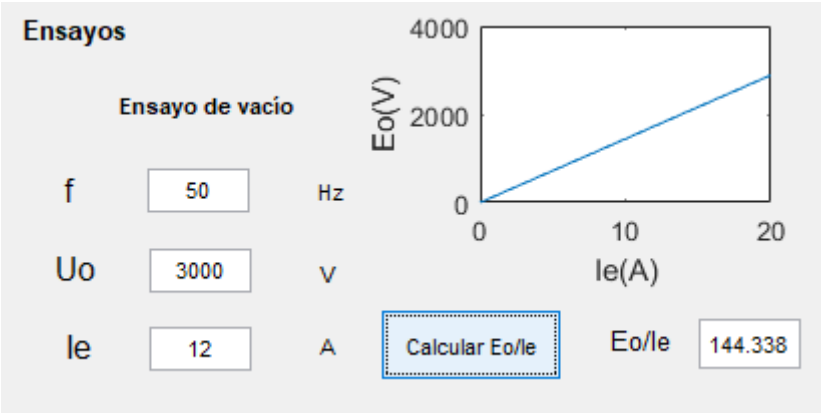

Figura 50. Cálculo relación E_0/I_E (2).

- Ensayo de cortocircuito:

$$\frac{E_{0cc}}{I_E} = \frac{E_{0cc}}{5} = 144{,}338 \rightarrow E_{0cc} = 721{,}69\ V$$

$$X_S = \frac{E_{0cc}}{I_{cc}} = \frac{721{,}69}{48{,}11} = 15{,}0008\ \Omega$$

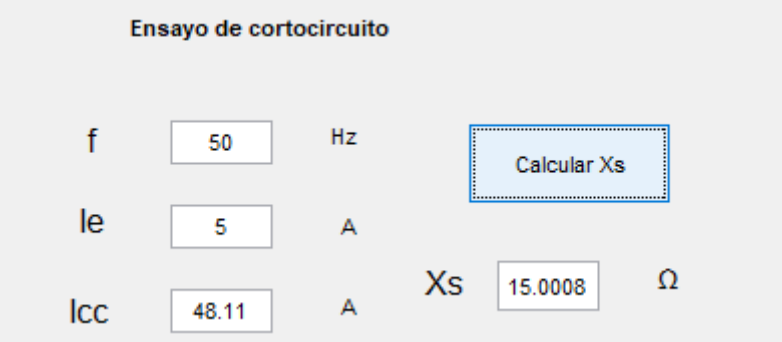

Figura 51. Cálculo X_S (2).

Una vez calculados los parámetros correspondientes a los ensayos de vacío y cortocircuito, se procederá a calcular el resto de los parámetros del motor (n, P, I, θ, I_E). Para ello, junto con los datos nominales del motor, se introducirán los datos de entrada de la segunda situación de funcionamiento. Al igual que en el paso anterior, se realizará el cálculo manual de dichos parámetros, y después se comprobará dicho resultado con el que nos proporciona la interfaz.

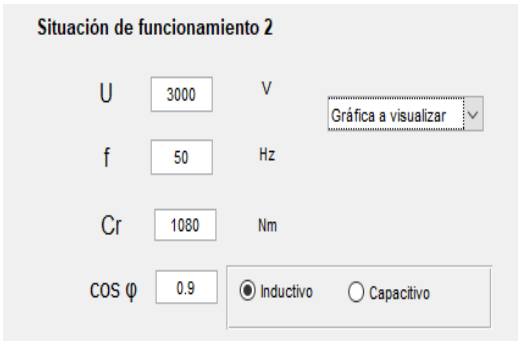

Figura 52. Datos de entrada situación 2.

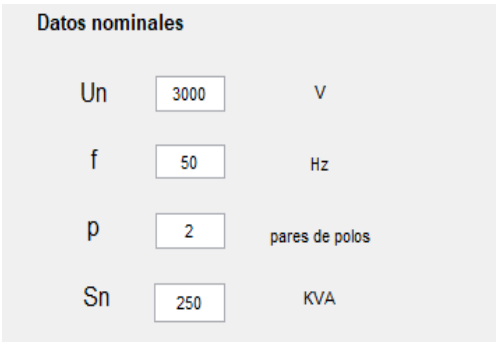

Figura 53. Datos nominales del motor (2).

- Velocidad de sincronismo:

$$n = \frac{60 \cdot f}{p} = \frac{60 \cdot 50}{2} = 1500\ rpm.$$

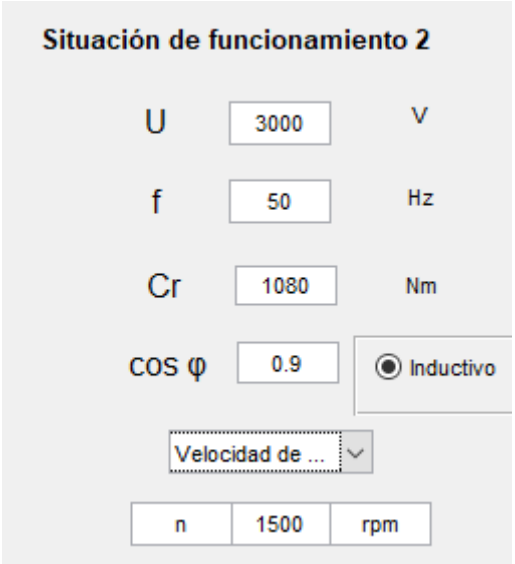

Figura 54. Cálculo n (2).

- Potencia:

$$P = 2 \cdot \pi \cdot \frac{n}{60} \cdot Cr = 2 \cdot \pi \cdot \frac{1500}{60} \cdot 1080 = 169646\, W$$

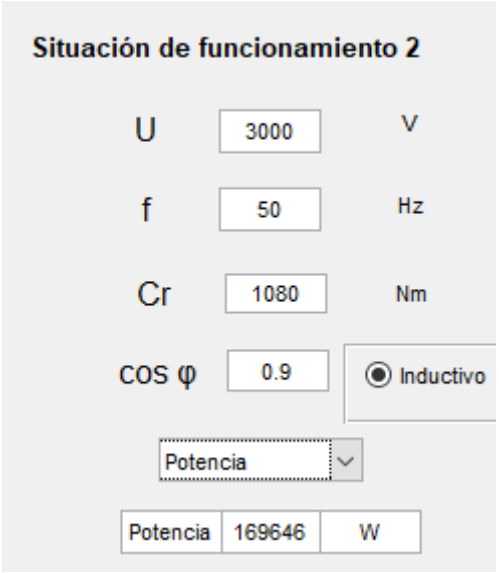

Figura 55. Cálculo P (2).

- Intensidad:

$$P = 3 \cdot V \cdot I \cdot \cos\varphi \rightarrow I = \frac{P}{3 \cdot V \cdot \cos\varphi}$$

$$I = \frac{169646}{3 \cdot \frac{3000}{\sqrt{3}} \cdot 0{,}9} = 36{,}276\, A$$

$$\varphi = \cos^{-1}(0{,}9) = 25{,}842\, grados \rightarrow \underline{I} = 36{,}276 \angle -25{,}842^{\circ}\, A$$

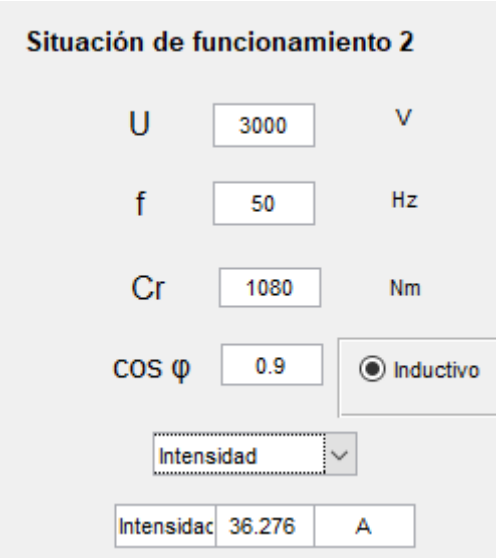

Figura 56. Cálculo I (2).

- Ángulo de par:

$$\underline{V} = \underline{E}_0 + \underline{I} \cdot (R + jX_S) \rightarrow \underline{E}_0 = \underline{V} - \underline{I} \cdot (R + jX_S)$$

$$\underline{E}_0 = \frac{3000}{\sqrt{3}} - \left(36{,}276\angle{-25{,}842°} \cdot (0 + j \cdot 15{,}0008)\right) = 1573{,}04\angle{-18{,}1401°}\ V$$

$$\theta = 18{,}1401\ grados$$

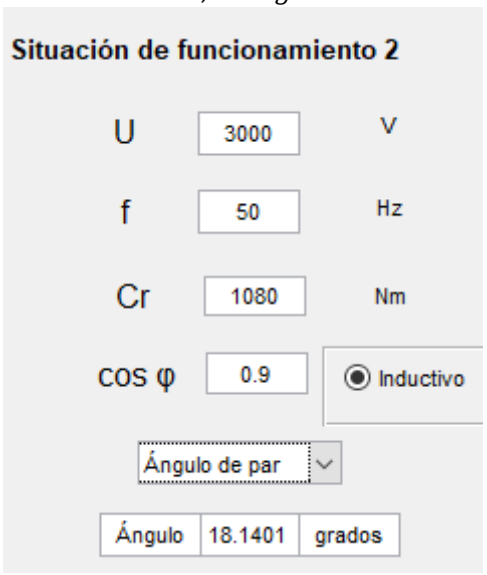

Figura 57. Cálculo θ (2).

- Intensidad de excitación:

$$E_0 = 1573{,}04\ V \rightarrow \frac{E_0}{I_E} = 144{,}338 \rightarrow\ I_E = 10{,}898\ A$$

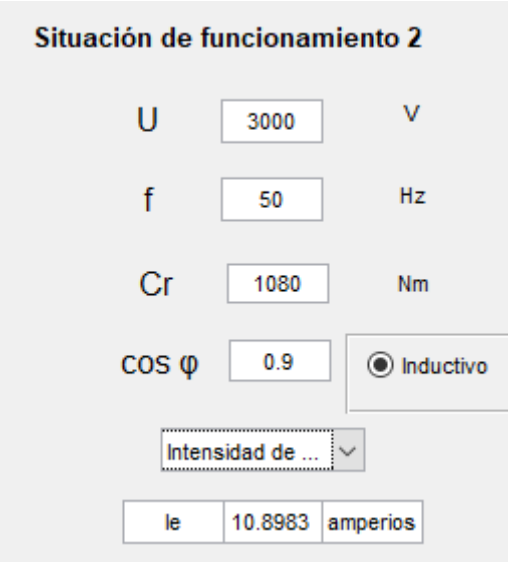

Figura 58. Cálculo IE.

Por último, se representan la curva de par y la curva en V de Mordey del motor. Para ello, seleccionamos en nuestra interfaz la curva que queremos visualizar. En cada una de las curvas también se visualiza el punto de funcionamiento en el que nos encontramos.

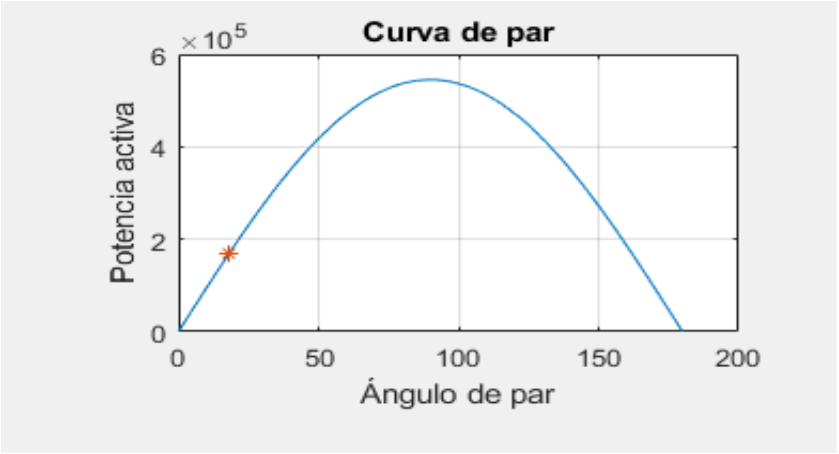

Figura 59. Curva de par (2).

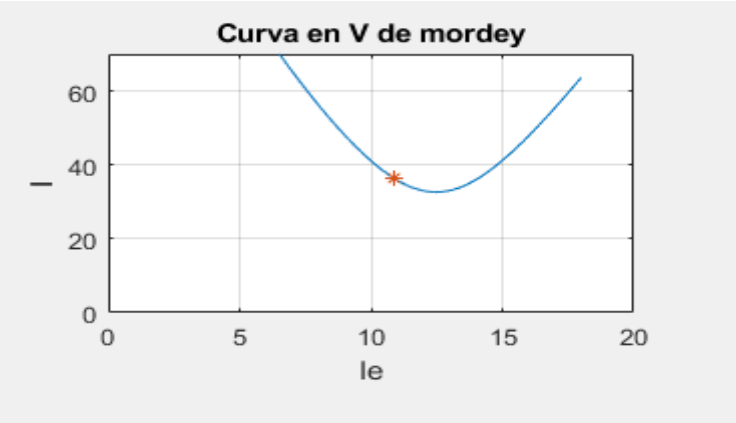

Figura 60. Curva en V de Mordey (2).

Para mostrar la sencillez y funcionalidad de la aplicación, se va a mostrar una situación en la que el ángulo de par es de 90 grados. Para ello, se calcula el factor de potencia mínimo (inductivo) que debe tener la máquina y se muestra como varía el punto de funcionamiento en la curva de par.

$$P = 3 \cdot V \cdot I \cdot \cos\varphi \rightarrow \cos\varphi = \frac{P}{3 \cdot V \cdot I}$$

$$E_0 = \frac{P \cdot X_S}{3 \cdot V} = 489.752\,V \rightarrow \underline{I} = \frac{\underline{V} - \underline{E}_0}{(R + jX_S)} = 119.99\ \angle -74^\circ\ A$$

$$\cos\varphi = \frac{169646}{3 \cdot \frac{3000}{\sqrt{3}} \cdot 119{,}99} = 0.27209$$

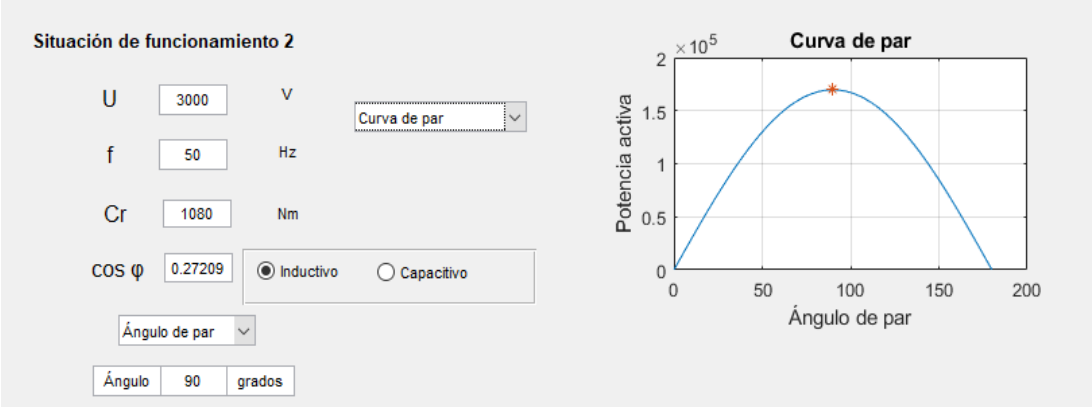

Figura 61. Funcionamiento θ=90 grados.

9. DESCRIPCIÓN DE TAREAS.DIAGRAMA DE GANTT

A continuación, se describen las diferentes tareas llevadas a cabo para la realización del TFG. A lo largo del desarrollo del mismo las personas que han tomado parte en el proyecto han sido un ingeniero junior (alumno encargado de realizar el TFG) y un ingeniero senior (el profesor encargado de dirigir el TFG). Las tareas desarrolladas han sido las siguientes:

- **Análisis del software:** El proyecto comienza el día 6 de noviembre de 2018, tras la primera reunión entre el profesor y el alumno. En esta primera reunión el profesor indica al alumno los primeros pasos a seguir para comenzar con el trabajo. En esta fase el alumno comienza a introducirse en el ámbito de trabajo del GUIDE, software utilizado para la realización de la interfaz. Este proceso mediante el cual el alumno debe ser capaz de manejar el programa con soltura comienza el 13 de noviembre de 2018 y concluye el 12 de diciembre de 2018.
- **Búsqueda de información:** El 17 de diciembre de 2018 tiene lugar la segunda reunión entre el alumno y profesor en la que éste le indica al alumno que debe comenzar con la búsqueda de información sobre el tema del TFG. En esta nueva etapa el alumno debe recopilar toda la información posible haciendo uso de las diferentes fuentes que tenga a su disposición. Esta fase de búsqueda de información concluye el 22 de enero de 2019.
- **Desarrollo de la interfaz gráfica:** Una vez ha tenido lugar la tercera reunión el 24 de enero de 2019, en la que el profesor y el alumno han clasificado la información más relevante de la recopilada por el alumno, comienza la fase de desarrollo de la interfaz gráfica. Durante esta etapa el alumno, guiado por las pautas indicadas por el profesor, deberá diseñar la aplicación software que nos permitirá cumplir con los objetivos de este proyecto. Esta fase comienza el 28 de enero de 2019 y concluye el 6 de marzo de 2019.
- **Corrección de errores:** Una vez finalizado el diseño inicial de la interfaz gráfica, tiene lugar la cuarta reunión el 7 de marzo de 2019. En esta reunión el profesor indica al alumno las diferentes modificaciones que se pueden llevar a cabo para mejor el funcionamiento de la aplicación. Durante esta etapa el alumno deberá modificar la interfaz siguiendo las indicaciones recibidas. Esta fase comienza el 11 de marzo de 2019 y concluye el 28 de abril de 2019.
- **Redacción del documento:** El día 29 de abril de 2019 tiene lugar la quinta reunión. En esta reunión el profesor da el visto bueno a la interfaz y comienza la etapa en la que el alumno debe redactar el informe del TFG. Esta etapa comienza el 2 de mayo de 2019 y concluye el 3 de julio de 2019.

- **Corrección del documento:** Una vez finalizada la redacción del documento tiene lugar una sexta reunión para corregir los posibles errores que el alumno haya podido cometer. Esta reunión tiene lugar el 4 de julio de 2019. Tras esta reunión el alumno comienza una última etapa en la que deberá llevar a cabo las modificaciones indicadas por el profesor. Esta fase comienza el 5 de julio de 2019 y concluye el 15 de julio de 2019. Una vez finalizada esta etapa tiene lugar una última reunión en la que se comprueba que el TFG está listo para ser entregado. Esta última reunión se celebra el día 16 de julio de 2019, fecha en la que se da por concluido el proyecto.

A continuación, se muestra el diagrama de Gantt realizado con las diferentes tareas que se han realizado en el desarrollo del proyecto.

Id	Nombre de tarea	Comienzo	Fin
1	H.0	mar 06/11/18	mar 06/11/18
2	R.1	mar 06/11/18	mar 06/11/18
3	T.1	mié 07/11/18	mié 12/12/18
4	H.1	jue 13/12/18	jue 13/12/18
5	R.2	lun 17/12/18	lun 17/12/18
6	T.2	mar 18/12/18	mar 22/01/19
7	H.2	mié 23/01/19	mié 23/01/19
8	R.3	jue 24/01/19	jue 24/01/19
9	T.3	lun 28/01/19	mié 06/03/19
10	R.4	jue 07/03/19	jue 07/03/19
11	T.4	lun 11/03/19	dom 28/04/19
12	H.3	lun 29/04/19	lun 29/04/19
13	R.5	lun 29/04/19	lun 29/04/19
14	T.5	jue 02/05/19	mié 03/07/19
15	R.6	jue 04/07/19	jue 04/07/19
16	T.6	vie 05/07/19	lun 15/07/19
17	R.7	mar 16/07/19	mar 16/07/19
18	H.4	mar 16/07/19	mar 16/07/19

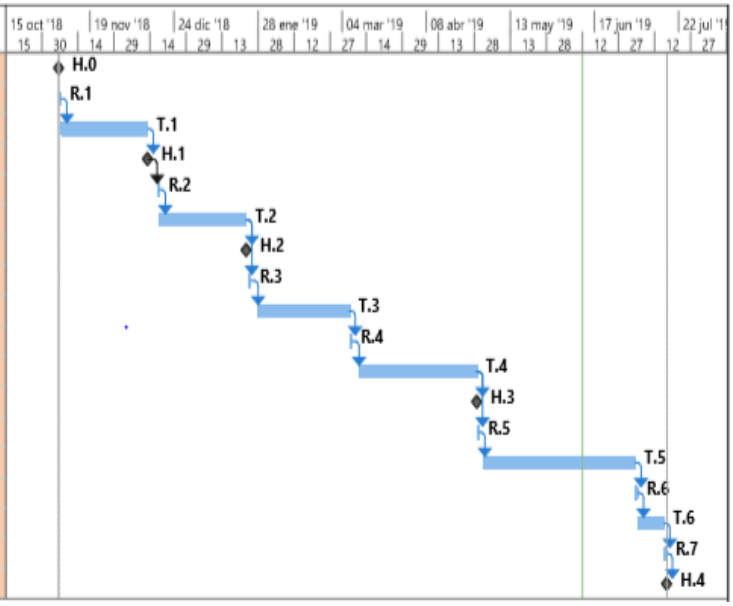

Figura 62. Diagrama de Gantt.

Nombre	Descripción	Fecha inicio	Fecha fin
H.0	Inicio del proyecto	06/11/2018	06/11/2018
R.1	Primera reunión	06/11/2018	06/11/2018
T.1	Análisis del software	13/11/2018	12/12/2018
H.1	Fin del análisis	13/12/2018	13/12/2018
R.2	Segunda reunión	17/12/2018	17/12/2018
T.2	Búsqueda de información	18/12/2018	22/01/2019
H.2	Fin de la búsqueda	23/01/2019	23/01/2019
R.3	Tercera reunión	24/01/2019	24/01/2019
T.3	Desarrollo de la interfaz gráfica	28/01/2019	06/03/2019
R.4	Cuarta reunión	07/03/2019	07/03/2019
T.4	Corrección de errores en la interfaz	11/03/2019	28/04/2019
H.3	Fin de la interfaz gráfica	29/04/2019	29/04/2019
R.5	Quinta reunión	29/04/2019	29/04/2019
T.5	Redacción del documento	02/05/2019	03/07/2019
R.6	Sexta reunión	04/07/2019	04/07/2019
T.6	Revisión del documento	05/07/2019	15/07/2019
R.7	Última reunión	16/07/2019	16/07/2019
H.4	Fin del proyecto	16/07/2019	16/07/2019

Tabla 3. Elementos del diagrama de Gantt.

10. PRESUPUESTO

El presupuesto se divide en 3 partidas: horas internas, amortizaciones y gastos. En la partida de horas internas se incluyen las horas invertidas en este TFG por un ingeniero senior (director del TFG) y un ingeniero junior (alumno que realiza el TFG). En la partida de amortizaciones se incluyen las licencias de Office y MATLAB y el ordenador. En la partida de gastos se incluyen los gastos de oficina que se han producido a lo largo del desarrollo del proyecto. Destacar que se ha considerado que los gastos indirectos suponen un 7% del presupuesto.

Horas internas			
Concepto	Nº de horas	€/hora	Total
Ingeniero senior	45h	50€/h	2.250 €
Ingeniero junior	190h	20€/h	3.800 €
SUBTOTAL			**7.050 €**

Tabla 4. Partida de horas internas.

Amortizaciones				
Concepto	Precio producto	Vida útil (horas)	Utilización (horas)	Total
Ordenador	1.300 €	25000h	130h	6,76 €
Licencia MATLAB	70 €	1500h	60h	2,80 €
Licancia Office	100 €	2750h	60h	2,18 €
SUBTOTAL				**11,74 €**

Tabla 5. Partida de amortizaciones.

Gastos	
Concepto	Total
Material	35 €
SUBTOTAL	**35 €**

Tabla 6. Partida de gastos.

Resumen	
Horas internas	7.050 €
Amortizaciones	11,74 €
Gastos	35 €
SUBTOTAL	**7.096,74 €**
Costes indirectos (7%)	496,77 €
TOTAL	**7.593,51 €**

Tabla 7. Resumen del presupuesto.

11. CONCLUSIONES

La máquina síncrona es de vital importancia dentro del sistema eléctrico de potencia y del ámbito industrial. Aunque su principal modo de funcionamiento es como generador asíncrono, también su aplicación como motor es de gran importancia, especialmente en aplicaciones que requieran trabajar a una velocidad contante o aplicaciones en las que se desee mejorar o corregir el factor de potencia.

Los motores síncronos están diseñados para unos valores nominales concretos. Si dichos valores se sobrepasan, pueden aparecer fallos inesperados o un envejecimiento prematuro de los componentes del motor. Además, estos fallos pueden causar, además de daños materiales, daños humanos a los operarios que trabajan con los motores o en un entorno cercano a ellos.

Tener a mano una interfaz gráfica que nos permite realizar diferentes cálculos sobre diferentes parámetros de los motores síncronos, puede ser de gran ayuda. Por una parte, nos permite conocer las condiciones bajo las que trabajará el motor para una situación concreta de funcionamiento. Por otra parte, nos permitirá observar si en algún momento se sobrepasa el valor nominal de la potencia del motor, pudiendo evitar futuros fallos en situaciones de funcionamiento real. De esta manera, mediante el uso de la interfaz se podrían evitar fallos inesperados que supongan un coste económico o aumenten el riesgo de los trabajadores.

A lo largo de este trabajo, se ha ido desarrollando una interfaz que nos permite calcular diferentes parámetros de los motores síncronos y representar la curva de par y la curva en V de Mordey de dichos motores. Además, también nos muestra mensajes de error y advertencia cuando la situación de funcionamiento que estamos introduciendo no es la adecuada.

12. REFERENCIAS

[1] **J. Mazón, J. F. Miñambres, M. Á. Zorrozua, G. Buigues y V. Valverde**, Guía de autoaprendizaje de máquinas eléctricas, Madrid: PEARSON EDUCACIÓN, S.A., 2008.

[2] **J. Fraile Mora**, Máquinas eléctricas, Madrid: McGRAW-HILL, S.A., 2003.

[3] **T. Wildi**, Máquinas eléctricas y sistemas de potencia, México: PEARSON EDUCACIÓN, 2006.

[4] **A. E. Fitzgerald, C. Kingsley y S. D. Umans**, Electric machinery, McGraw-Hill, 2002.

[5] **J. M. Aller**, Máquinas eléctricas rotativas: Introducción a la teoría general, Caracas: EQUINOCIO, 2008.

[6] **S. J. Chapman**, Máquinas eléctricas, México, D.F.: McGraw-Hill, 2012.

[7] **A. Hughes**, Electric Motors and Drives.Fundamentals, Types and Applications, Newnes, 2006.

[8] «**Página web oficial PTC,»** [En línea]. Available: https://www.ptc.com/es/products/mathcad.

[9] **M. I. Zamora, Á. J. Mazón, E. Fernández, K. J. Sagastabeitia, I. Albizu, P. Eguía, E. Torres y V. Valverde**, Simulación de Sistemas Eléctricos, Madrid: Pearson Educación S.A, 2005.

[10] **«Página web oficial mathworks,»** [En línea]. Available: https://es.mathworks.com/products/matlab.

13. ANEXO. CÓDIGO COMPLETO DE LA INTERFAZ

```
function varargout = motor2(varargin)
% MOTOR2 MATLAB code for motor2.fig
%      MOTOR2, by itself, creates a new MOTOR2 or raises the existing
%      singleton*.
%
%      H = MOTOR2 returns the handle to a new MOTOR2 or the handle to
%      the existing singleton*.
%
%      MOTOR2('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in MOTOR2.M with the given input
arguments.
%
%      MOTOR2('Property','Value',...) creates a new MOTOR2 or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before motor2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to motor2_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 motor2

% Last Modified by GUIDE v2.5 04-Jul-2019 11:06:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @motor2_OpeningFcn, ...
                   'gui_OutputFcn',  @motor2_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 motor2 is made visible.
function motor2_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 motor2 (see VARARGIN)

% Choose default command line output for motor2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes motor2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = motor2_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;



function Un_Callback(hObject, eventdata, handles)
% hObject    handle to Un (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 Un as text
%        str2double(get(hObject,'String')) returns contents of Un as a
double


% --- Executes during object creation, after setting all properties.
function Un_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Un (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 edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (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 edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2
as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (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 polos_Callback(hObject, eventdata, handles)
% hObject    handle to polos (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 polos as text
%        str2double(get(hObject,'String')) returns contents of polos
as a double


% --- Executes during object creation, after setting all properties.
function polos_CreateFcn(hObject, eventdata, ~)
% hObject    handle to polos (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 edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (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 edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5
as a double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (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 relacion_Callback(hObject, eventdata, handles)
% hObject    handle to relacion (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 relacion as text
%        str2double(get(hObject,'String')) returns contents of
relacion as a double


% --- Executes during object creation, after setting all properties.
function relacion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to relacion (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 pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and ,user data (see GUIDATA)
x=str2num(get(handles.relacion,'String'));
Ie=linspace(0,20,50);
E0=x*Ie;
plot(handles.axes1,Ie,E0);
xlabel('Ie');
ylabel('Eo');



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (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 edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7
as a double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (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 Ie_Callback(hObject, eventdata, handles)
% hObject    handle to Ie (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 Ie as text
%        str2double(get(hObject,'String')) returns contents of Ie as a
double


% --- Executes during object creation, after setting all properties.
function Ie_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ie (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 Icc_Callback(hObject, eventdata, handles)
% hObject    handle to Icc (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 Icc as text
%        str2double(get(hObject,'String')) returns contents of Icc as
a double


% --- Executes during object creation, after setting all properties.
function Icc_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Icc (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 pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

rel=str2num(get(handles.relacion,'String'));
ie=str2num(get(handles.Ie,'String'));
E0=rel*ie;
icc=str2num(get(handles.Icc,'String'));
Xs=E0/icc;
set(handles.edit10,'String',Xs);

function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (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 edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10
as a double



% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (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 checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
x=str2num(get(handles.relacion,'String'));
Ie=linspace(0,20,50);
E0=x*Ie;
plot(handles.axes1,Ie,E0);
xlabel(handles.axes1,'Ie');
ylabel(handles.axes1,'E0');

% Hint: get(hObject,'Value') returns toggle state of checkbox1
```

```
function Vred_Callback(hObject, eventdata, handles)
% hObject    handle to Vred (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 Vred as text
%        str2double(get(hObject,'String')) returns contents of Vred as
a double


% --- Executes during object creation, after setting all properties.
function Vred_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Vred (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 frecuencia4_Callback(hObject, eventdata, handles)
% hObject    handle to frecuencia4 (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 frecuencia4 as text
%        str2double(get(hObject,'String')) returns contents of
frecuencia4 as a double


% --- Executes during object creation, after setting all properties.
function frecuencia4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to frecuencia4 (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 Ie2_Callback(hObject, eventdata, handles)
% hObject    handle to Ie2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60
        Xss=str2num(get(handles.edit10,'String'));
        V=str2num(get(handles.Vred,'String'));
        E01=P*Xss/(sqrt(3)*V)
        rel=str2num(get(handles.relacion,'String'));
        Iemin=E01/rel
        Ie22=str2num(get(hObject,'String'));
        if Ie22<Iemin
            warndlg('La Ie debe ser mayor','ADVERTENCIA');
        end


% Hints: get(hObject,'String') returns contents of Ie2 as text
%        str2double(get(hObject,'String')) returns contents of Ie2 as
a double


% --- Executes during object creation, after setting all properties.
function Ie2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ie2 (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 Cm_Callback(hObject, eventdata, handles)
% hObject    handle to Ie2 (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 Ie2 as text
%        str2double(get(hObject,'String')) returns contents of Ie2 as
a double
```

```
sn=str2num(get(handles.Sn,'String'));
Sn=sn*1000;
par=str2num(get(hObject,'String'));
polos=str2num(get(handles.polos,'String'));
f=str2num(get(handles.frecuencia4,'String'));
v=(60*f)/polos;
P=(2*pi*v*par)/60;
if P>Sn
    errordlg('La potencia supera la potencia nominal de la
màquina','ERROR');
end



% --- Executes during object creation, after setting all properties.
function Cm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ie2 (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 selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
contenido=get(hObject,'String');
a=get(hObject,'Value');
texto=contenido(a);
switch cell2mat(texto)
    case 'Velocidad de sincronismo'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
      set(handles.resultado,'String',v);
      set(handles.dato,'String','n');
      set(handles.unidades,'String','rpm');
    case 'Potencia'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60;
        set(handles.resultado,'String',P);
        set(handles.dato,'String','Potencia');
        set(handles.unidades,'String','W');
    case 'Ángulo de par'
```

```
        polos=str2num(get(handles.polos,'String'))
        f=str2num(get(handles.frecuencia4,'String'))
        v=(60*f)/polos
        par=str2num(get(handles.Cm,'String'))
        P=(2*pi*v*par)/60
        Xs=str2num(get(handles.edit10,'String'))
        rel=str2num(get(handles.relacion,'String'))
        ie=str2num(get(handles.Ie2,'String'))
        E0=rel*ie
        V=str2num(get(handles.Vred,'String'))
        angulo=asind((P*Xs)/(V*sqrt(3)*E0))
        set(handles.resultado,'String',angulo);
        set(handles.dato,'String','Ángulo');
        set(handles.unidades,'String','Grados');
    case 'Intensidad'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60
        Xs=str2num(get(handles.edit10,'String'))
        rel=str2num(get(handles.relacion,'String'));
        ie=str2num(get(handles.Ie2,'String'));
        E0=rel*ie
        V=str2num(get(handles.Vred,'String'))
        angulo=asin((P*Xs)/(V*sqrt(3)*E0));
        V1=(V/sqrt(3))+i*0;
        E00=E0*cos(angulo)-i*E0*sin(angulo);
        Xs1=i*Xs;
        I=(V1-E00)/Xs1;
        set(handles.resultado,'String',abs(I));
        set(handles.dato,'String','Intensidad');
        set(handles.unidades,'String','Amperios');
    case 'Factor de potencia'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60
        Xs=str2num(get(handles.edit10,'String'))
        rel=str2num(get(handles.relacion,'String'));
        ie=str2num(get(handles.Ie2,'String'));
        E0=rel*ie
        V=str2num(get(handles.Vred,'String'))
        angulo=asin((P*Xs)/(V*sqrt(3)*E0));
        V1=(V/sqrt(3))+i*0;
        E00=E0*cos(-angulo)+i*E0*sin(-angulo)
        Xs1=i*Xs;
        I=(V1-E00)/Xs1
        fp=-angle(I)
        fp=cos(fp)
        Imed=P/(3*V1)
        E001=(Imed*Xs1)+V1
```

```
        iemed=abs(E001)/rel
        iemed1=round(iemed,2)
        set(handles.resultado,'String',fp);
        set(handles.dato,'String','fp');
        if ie>iemed1
        set(handles.unidades,'String','capacitivo');
        elseif ie<iemed1
        set(handles.unidades,'String','inductivo');
        else
        set(handles.unidades,'String','');
        end
end


% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all
properties.function popupmenu1_CreateFcn(hObject, eventdata,
handles)function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu 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 resultado_Callback(hObject, eventdata, handles)
% hObject    handle to resultado (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 resultado as text
%        str2double(get(hObject,'String')) returns contents of
resultado as a double
```

```
% --- Executes during object creation, after setting all properties.
function resultado_CreateFcn(hObject, eventdata, handles)
% hObject    handle to resultado (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 dato_Callback(hObject, eventdata, handles)
% hObject    handle to dato (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 dato as text
%        str2double(get(hObject,'String')) returns contents of dato as
a double


% --- Executes during object creation, after setting all properties.
function dato_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dato (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 unidades_Callback(hObject, eventdata, handles)
% hObject    handle to unidades (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 unidades as text
%        str2double(get(hObject,'String')) returns contents of
unidades as a double
```

```
% --- Executes during object creation, after setting all properties.
function unidades_CreateFcn(hObject, eventdata, handles)
% hObject    handle to unidades (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 checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox2
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60
        Xs=str2num(get(handles.edit10,'String'))
        rel=str2num(get(handles.relacion,'String'));
        ie=str2num(get(handles.Ie2,'String'));
        E0=rel*ie
        V=str2num(get(handles.Vred,'String'))
        angulo1=rad2deg(asin((P*Xs)/(V*sqrt(3)*E0)));
        angulo=degtorad(angulo1);
        angulo=linspace(0,pi,100);
        y=sqrt(3)*V*E0*sin(angulo)/Xs
        plot(handles.axes2,rad2deg(angulo),y);




% --- Executes on button press in checkbox3.
function checkbox3_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox3
```

```
% --- Executes on button press in checkbox4.
function checkbox4_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox4



function Vred2_Callback(hObject, eventdata, handles)
% hObject    handle to Vred2 (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 Vred2 as text
%        str2double(get(hObject,'String')) returns contents of Vred2
as a double



% --- Executes during object creation, after setting all properties.
function Vred2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Vred2 (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 frecuencia5_Callback(hObject, eventdata, handles)
% hObject    handle to frecuencia5 (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 frecuencia5 as text
%        str2double(get(hObject,'String')) returns contents of
frecuencia5 as a double



% --- Executes during object creation, after setting all properties.
function frecuencia5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to frecuencia5 (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 fp2_Callback(hObject, eventdata, handles)
% hObject    handle to fp2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
valor=get(handles.radiobutton1,'Value');
        if valor==1
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60
        Xss=str2num(get(handles.edit10,'String'));
        V=str2num(get(handles.Vred2,'String'));
        E01=P*Xss/(sqrt(3)*V)
        V1=V/sqrt(3)
        E02=-i*E01
        Xs=i*Xss
        I1=(E02-V1)/Xs
        I2=abs(I1)
        fpmin=P/(3*V1*I2)
        fp=str2num(get(hObject,'String'));
        if fp<fpmin
            warndlg('El factor de potencia debe ser
mayor','ADVERTENCIA');
        end
        end
% Hints: get(hObject,'String') returns contents of fp2 as text
%        str2double(get(hObject,'String')) returns contents of fp2 as
a double



% --- Executes during object creation, after setting all properties.
function fp2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to fp2 (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 Cm2_Callback(hObject, eventdata, handles)
% hObject    handle to Cm2 (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 Cm2 as text
%        str2double(get(hObject,'String')) returns contents of Cm2 as
a double
sn=str2num(get(handles.Sn,'String'));
Sn=sn*1000
polos=str2num(get(handles.polos,'String'));
f=str2num(get(handles.frecuencia5,'String'));
v=(60*f)/polos;
par=str2num(get(hObject,'String'));
P=(2*pi*v*par)/60;
if P>Sn
    errordlg('La potencia supera la potencia nominal de la
máquina','ERROR');
end

% --- Executes during object creation, after setting all properties.
function Cm2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Cm2 (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 selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
contenido=get(hObject,'String');
a=get(hObject,'Value');
texto=contenido(a);
switch cell2mat(texto)
    case 'Velocidad de sincronismo'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
```

```
        v=(60*f)/polos;
      set(handles.resultado2,'String',v);
      set(handles.dato2,'String','n');
      set(handles.unidades2,'String','rpm');
    case 'Potencia'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60;
        set(handles.resultado2,'String',P);
        set(handles.dato2,'String','Potencia');
        set(handles.unidades2,'String','W');

    case 'Intensidad'
        valor=get(handles.radiobutton1,'Value');
        if valor==1
            polos=str2num(get(handles.polos,'String'));
            f=str2num(get(handles.frecuencia5,'String'));
            v=(60*f)/polos;
            par=str2num(get(handles.Cm2,'String'));
            P=(2*pi*v*par)/60;
            Un=str2num(get(handles.Vred2,'String'));
            fp=str2num(get(handles.fp2,'String'));
            I=P/(sqrt(3)*Un*fp);
            angulo=acos(fp);
            I1=I*cos(-angulo)+i*I*sin(-angulo)
            set(handles.resultado2,'String',abs(I1));
            set(handles.dato2,'String','Intensidad');
            set(handles.unidades2,'String','A');
        else

            polos=str2num(get(handles.polos,'String'));
            f=str2num(get(handles.frecuencia5,'String'));
            v=(60*f)/polos;
            par=str2num(get(handles.Cm2,'String'));
            P=(2*pi*v*par)/60;
            Un=str2num(get(handles.Vred2,'String'));
            fp=str2num(get(handles.fp2,'String'));
            I=P/(sqrt(3)*Un*fp);
            angulo=acos(fp);
            I1=I*cos(angulo)+i*I*sin(angulo)
            set(handles.resultado2,'String',abs(I1));
            set(handles.dato2,'String','Intensidad');
            set(handles.unidades2,'String','A');
        end
    case 'Ángulo de par'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60;
        Un=str2num(get(handles.Vred2,'String'));
```

```
        fp=str2num(get(handles.fp2,'String'));
        I=P/(sqrt(3)*Un*fp)
        angulo=acos(fp)
        valor=get(handles.radiobutton1,'Value');
        if valor==1
        I1=I*cos(-angulo)+i*I*sin(-angulo)
        Xs=str2num(get(handles.edit10,'String'));
        Xs=i*Xs
        V=Un/sqrt(3)+i*0
        E=V-(I1*Xs)
        fgk=rad2deg(-angle(E))
        set(handles.resultado2,'String',fgk);
        set(handles.dato2,'String','Ángulo');
        set(handles.unidades2,'String','grados');

        else
        I1=I*cos(angulo)+i*I*sin(angulo)
        Xs=str2num(get(handles.edit10,'String'));
        Xs=i*Xs
        V=Un/sqrt(3)+i*0
        E=V-(I1*Xs)
        fgk=rad2deg(-angle(E))
        set(handles.resultado2,'String',fgk);
        set(handles.dato2,'String','Ángulo');
        set(handles.unidades2,'String','grados');

        end
    case 'Intensidad de excitación'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60;
        Un=str2num(get(handles.Vred2,'String'));
        fp=str2num(get(handles.fp2,'String'));
        I=P/(sqrt(3)*Un*fp)
        angulo=acos(fp)
        valor=get(handles.radiobutton1,'Value');
        if valor==1
        I1=I*cos(-angulo)+i*I*sin(-angulo)
        Xs=str2num(get(handles.edit10,'String'));
        Xs=i*Xs;
        V=Un/sqrt(3)+i*0;
        E=V-(I1*Xs)
        E0=abs(E)
        rel=str2num(get(handles.relacion,'String'));
        Ie=E0/rel;
        set(handles.resultado2,'String',Ie);
        set(handles.dato2,'String','Ie');
        set(handles.unidades2,'String','amperios');
        else
        I1=I*cos(angulo)+i*I*sin(angulo)
        Xs=str2num(get(handles.edit10,'String'));
```

```
        Xs=i*Xs;
        V=Un/sqrt(3)+i*0;
        E=V-(I1*Xs)
        E0=abs(E)
        rel=str2num(get(handles.relacion,'String'));
        Ie=E0/rel;
        set(handles.resultado2,'String',Ie);
        set(handles.dato2,'String','Ie');
        set(handles.unidades2,'String','amperios');
        end

end

% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu 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 resultado2_Callback(hObject, eventdata, handles)
% hObject    handle to resultado2 (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 resultado2 as text
%        str2double(get(hObject,'String')) returns contents of
resultado2 as a double


% --- Executes during object creation, after setting all properties.
function resultado2_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to resultado2 (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 dato2_Callback(hObject, eventdata, handles)
% hObject    handle to dato2 (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 dato2 as text
%        str2double(get(hObject,'String')) returns contents of dato2
as a double


% --- Executes during object creation, after setting all properties.
function dato2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dato2 (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 unidades2_Callback(hObject, eventdata, handles)
% hObject    handle to unidades2 (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 unidades2 as text
%        str2double(get(hObject,'String')) returns contents of
unidades2 as a double


% --- Executes during object creation, after setting all properties.
```

```
function unidades2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to unidades2 (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 checkbox5.
function checkbox5_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox5



% --- Executes on button press in checkbox6.
function checkbox6_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox6



% --- Executes on button press in checkbox7.
function checkbox7_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox7



% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu4
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu4
```

```
contenido=get(hObject,'String');
b=get(hObject,'Value');
texto=contenido(b);
switch cell2mat(texto)
    case 'Curva de par'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60
        Xs=str2num(get(handles.edit10,'String'))
        rel=str2num(get(handles.relacion,'String'));
        ie=str2num(get(handles.Ie2,'String'));
        E0=rel*ie
        V=str2num(get(handles.Vred,'String'))
        angulo1=rad2deg(asin((P*Xs)/(V*sqrt(3)*E0)))
        angulo=degtorad(angulo1);
        angulo=linspace(0,pi,100);
        y=sqrt(3)*V*E0*sin(angulo)/Xs
        plot(handles.axes2,rad2deg(angulo),y,angulo1,P,'*');
        xlabel(handles.axes2,'Ángulo de par');
        ylabel(handles.axes2,'Potencia activa');
        title(handles.axes2,'Curva de par');
        grid(handles.axes2,'on');

    case 'Curva en V de Mordey'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'));
        P=(2*pi*v*par)/60
        Xss=str2num(get(handles.edit10,'String'));
        V=str2num(get(handles.Vred,'String'));
        E01=P*Xss/(sqrt(3)*V)
        rel=str2num(get(handles.relacion,'String'));
        Iemin=E01/rel
        Iemin1=round(Iemin,1)
        Iemin2=Iemin1*10
        i_f=(Iemin2:1:185)/10;
        b=(185-Iemin2)+1
        i_a=zeros(1,b);
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia4,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm,'String'))
        P=(2*pi*v*par)/60
        Xs=str2num(get(handles.edit10,'String'))
        rel=str2num(get(handles.relacion,'String'))
        ie=str2num(get(handles.Ie2,'String'))
        E0=rel*ie
        V=str2num(get(handles.Vred,'String'))
        V1=V/sqrt(3)
        angulo1=asin((P*Xs)/(V1*3*E0))
```

```
        E1=E0*(cos(-angulo1)+i*sin(-angulo1))
        Xs1=i*Xs;
        I=(V1-E1)/Xs1;
        for ii=1:b
            E2=rel*i_f(ii);
            angulo2=asin((abs(E1)/abs(E2))*sin(angle(E1)));
            E2=E2*(cos(-angulo2)+i*sin(-angulo2));
            i_a(ii)=(V1-E2)/(Xs);
        end
        plot(handles.axes2,i_f,abs(i_a),ie,abs(I),'*');
        axis(handles.axes2,[0 20 0 70]);
        xlabel(handles.axes2,'Ie');
        ylabel(handles.axes2,'I');
        title(handles.axes2,'Curva en V de mordey');
        grid(handles.axes2,'on');




end



% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu 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 selection change in popupmenu5.
function popupmenu5_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu5
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu5
contenido=get(hObject,'String');
b=get(hObject,'Value');
texto=contenido(b);
```

```
switch cell2mat(texto)
    case 'Curva de par'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60;
        Un=str2num(get(handles.Vred2,'String'));
        fp=str2num(get(handles.fp2,'String'));
        I=P/(sqrt(3)*Un*fp)
        angulo=acos(fp)
        valor=get(handles.radiobutton1,'Value');
        if valor==1
        I1=I*cos(-angulo)+i*I*sin(-angulo)
        Xs=str2num(get(handles.edit10,'String'));
        Xs=i*Xs
        reac=abs(Xs)
        V=Un/sqrt(3)+i*0
        E=V-(I1*Xs)
        E0=abs(E)
        fgk=rad2deg(-angle(E))
        angulo=linspace(0,pi,180)
        y=3*V*E0*sin(angulo)/reac
        plot(handles.axes5,rad2deg(angulo),y,fgk,P,'*')
        xlabel(handles.axes5,'Ángulo de par');
        ylabel(handles.axes5,'Potencia activa');
        title(handles.axes5,'Curva de par');
        grid(handles.axes5,'on');
        else
        I1=I*cos(angulo)+i*I*sin(angulo)
        Xs=str2num(get(handles.edit10,'String'));
        Xs=i*Xs
        reac=abs(Xs)
        V=Un/sqrt(3)+i*0
        E=V-(I1*Xs)
        E0=abs(E)
        fgk=rad2deg(-angle(E))
        angulo=linspace(0,pi,180)
        y=3*V*E0*sin(angulo)/reac
        plot(handles.axes5,rad2deg(angulo),y,fgk,P,'*')
        xlabel(handles.axes5,'Ángulo de par');
        ylabel(handles.axes5,'Potencia activa');
        title(handles.axes5,'Curva de par');
        grid(handles.axes5,'on');
        end
    case 'Curva en V de Mordey'
        polos=str2num(get(handles.polos,'String'));
        f=str2num(get(handles.frecuencia5,'String'));
        v=(60*f)/polos;
        par=str2num(get(handles.Cm2,'String'));
        P=(2*pi*v*par)/60
        Xss=str2num(get(handles.edit10,'String'));
        Vv=str2num(get(handles.Vred2,'String'));
```

```
E01=P*Xss/(sqrt(3)*Vv)
rel=str2num(get(handles.relacion,'String'));
Iemin=E01/rel
Iemin1=round(Iemin,1)
Iemin2=Iemin1*10
i_f= (Iemin2:1:180)/10;
b=(180-Iemin2)+1
i_a=zeros(1,b);
polos=str2num(get(handles.polos,'String'));
f=str2num(get(handles.frecuencia5,'String'));
v=(60*f)/polos;
par=str2num(get(handles.Cm2,'String'));
P=(2*pi*v*par)/60;
Un=str2num(get(handles.Vred2,'String'));
fp=str2num(get(handles.fp2,'String'));
I=P/(sqrt(3)*Un*fp)
angulo=acos(fp)
valor=get(handles.radiobutton1,'Value');
if valor==1
I1=I*cos(-angulo)+i*I*sin(-angulo)
Xs=str2num(get(handles.edit10,'String'));
Xs=i*Xs
reac=abs(Xs)
V=Un/sqrt(3)+i*0
E1=V-(I1*Xs)
rel=str2num(get(handles.relacion,'String'));
for ii=1:b
    E2=rel*i_f(ii);
    angulo2=asin((abs(E1)/abs(E2))*sin(angle(E1)));
    E2=E2*(cos(angulo2)+i*sin(angulo2));
    i_a(ii)=(V-E2)/(Xs);
end
iee=abs(E1)/rel
plot(handles.axes5,i_f,abs(i_a),iee,abs(I1),'*');
axis([0 20 0 70]);
xlabel(handles.axes5,'Ie');
ylabel(handles.axes5,'I');
title(handles.axes5,'Curva en V de mordey');
grid(handles.axes5,'on');
else
I1=I*cos(angulo)+i*I*sin(angulo)
Xs=str2num(get(handles.edit10,'String'));
Xs=i*Xs
reac=abs(Xs)
V=Un/sqrt(3)+i*0
E1=V-(I1*Xs)
rel=str2num(get(handles.relacion,'String'));
for ii=1:b
    E2=rel*i_f(ii);
    angulo2=asin((abs(E1)/abs(E2))*sin(angle(E1)));
    E2=E2*(cos(angulo2)+i*sin(angulo2));
    i_a(ii)=(V-E2)/(Xs);
end
```

```
        iee=abs(E1)/rel;
        plot(handles.axes5,i_f,abs(i_a),iee,abs(I1),'*');
        axis([0 20 0 70]);
        xlabel(handles.axes5,'Ie');
        ylabel(handles.axes5,'I');
        title(handles.axes5,'Curva en V de mordey');
        grid(handles.axes5,'on');
        end



end

% --- Executes during object creation, after setting all properties.
function popupmenu5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu 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 pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

U0=str2num(get(handles.Uo,'String'));
Iee=str2num(get(handles.Iee,'String'));
E0=U0/sqrt(3);
rela=E0/Iee;
set(handles.relacion,'String',rela);
IE=linspace(0,20,50)
E00=rela*IE
plot(handles.axes1,IE,E00);
xlabel(handles.axes1,'Ie(A)');
ylabel(handles.axes1,'Eo(V)');

function Uo_Callback(hObject, eventdata, handles)
% hObject    handle to Uo (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 Uo as text
```

```
%        str2double(get(hObject,'String')) returns contents of Uo as a
double


% --- Executes during object creation, after setting all properties.
function Uo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Uo (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 Iee_Callback(hObject, eventdata, handles)
% hObject    handle to Iee (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 Iee as text
%        str2double(get(hObject,'String')) returns contents of Iee as
a double


% --- Executes during object creation, after setting all properties.
function Iee_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Iee (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 radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of radiobutton2



function Sn_Callback(hObject, eventdata, handles)
% hObject    handle to Sn (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 Sn as text
%        str2double(get(hObject,'String')) returns contents of Sn as a
double



% --- Executes during object creation, after setting all properties.
function Sn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sn (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
```