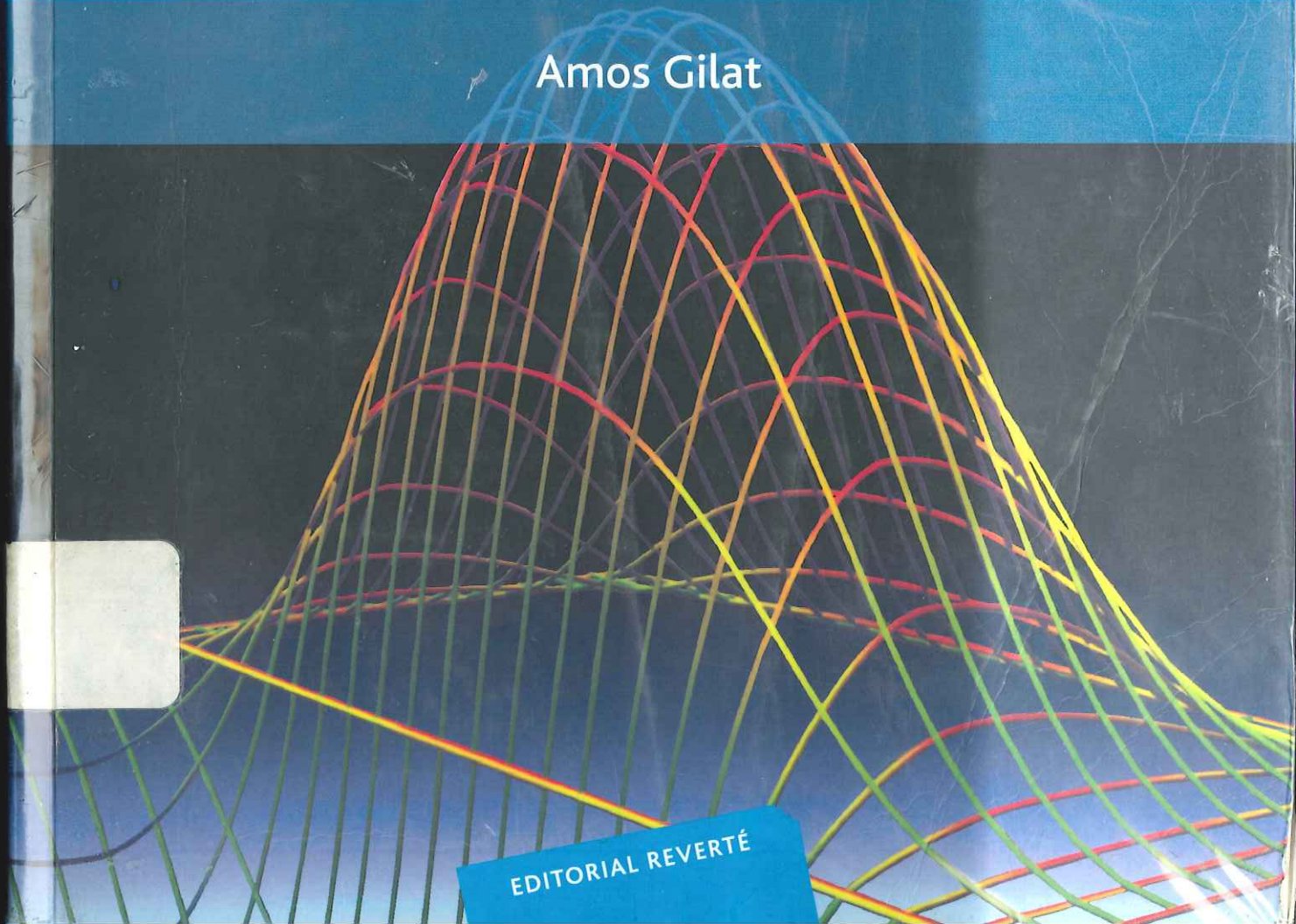
Matlab®

Una introducción con ejemplos prácticos

Amos Gilat



EDITORIAL REVERTÉ

Matlab®

Una introducción con ejemplos prácticos

Matlab®

Una introducción con ejemplos prácticos

Amos Gilat

Department of Mechanical Engineering
The Ohio State University

EDITORIAL REVERTÉ, S. A.

Barcelona • Bogotá • Buenos Aires • Caracas • México

Título de la obra original:
MATLAB: An Introduction With Applications, 2/Edition – W/471-69420-7

Edición original en lengua inglesa publicada por
John Wiley & Sons, Inc., Hoboken (NJ), USA

Versión española por
Dr. José Antonio Macías Iglesias
Profesor del Departamento de Ingeniería Informática de la
Escuela Politécnica Superior, Universidad Autónoma de Madrid (España)

Propiedad de:
EDITORIAL REVERTÉ, S. A.
Loreto, 13-15. Local B
08029 Barcelona. ESPAÑA
Tel: (34) 93 419 33 36
Fax: (34) 93 419 51 89
e-mail: reverte@reverte.com
www.reverte.com

MAQUETACIÓN: REVERTÉ-AGUILAR, S. L.

Edición en español:

© Editorial Reverté, S. A., 2006
ISBN-10: 84-291-5035-8
ISBN-13: 978-84-291-5035-3
Depósito Legal: B-22303-2006

Impreso en España - Printed in Spain

Impreso por Liberdúplex, S. L. U.

Prefacio

MATLAB® es un lenguaje muy popular en el ámbito de la computación científica que es utilizado por estudiantes, ingenieros y científicos en universidades, institutos de investigación e industrias a lo largo y ancho del mundo. Su popularidad se debe, fundamentalmente, a su potencia y su facilidad de uso. En los primeros cursos universitarios, éste software puede verse como la siguiente herramienta a utilizar por los estudiantes después de la calculadora científica, utilizada comúnmente en el bachillerato científico.

Este libro se ha escrito en base a la experiencia llevada a cabo durante varios años de enseñanza en primeros cursos universitarios de ingeniería. El objetivo era escribir un libro que permitiera enseñar este software de una manera amistosa, sin intimidar por su aparente complejidad. De esta forma el libro está escrito en un lenguaje directo. En algunas páginas pueden encontrarse listas descriptivas y relacionadas con ciertos conceptos, en vez de grandes cantidades explícitas de texto. El libro incluye además numerosos ejercicios de ejemplo utilizados comúnmente en matemáticas, ciencias e ingeniería, los cuales son similares a los problemas con los que pueden encontrarse los usuarios que se enfrente por primera vez a MATLAB.

La segunda edición de este libro está actualizada para ser utilizada con MATLAB 7. Se incluye además un nuevo Capítulo (11) que trata sobre cálculo simbólico con MATLAB, y una nueva sección en el Capítulo 4 que muestra la importancia de trabajar con la exportación e importación de datos.

Me gustaría dar las gracias a varios de mis colegas de la Universidad del Estado de Ohio: Profesor Richard Freuler, Mark Walter, Brian Harper, Walter Lampert y el Dr. Mike Parke por leer diferentes secciones de este libro y proponerme modificaciones. También quiero agradecer, por su implicación y apoyo, a los Profesores Robert Gustafson, John Demel y al Dr. John Merrill del programa de primer año de Ingeniería de la Universidad del Estado de Ohio. Quiero agradecer especialmente al Profesor Mike Lichtenstieger (OSU) y a mi hermana Tal Gilat (de la Universidad de Stanford) sus revisiones sobre el libro, así como las críticas y los comentarios aportados.

Me gustaría expresar mi gratitud a todos aquellos que han revisado este texto en sus distintas fases de desarrollo, incluyendo a Betty Barr, de la Universidad de Houston; Andrei G. Chakhovskoi, de la Universidad de California, Davis; Rober Kina, de la Universidad de Toledo; Richard Kwor, de la Universidad de Colorado en Colorado Springs; Larry Lagerstrom, de la Universidad de California, Davis; Yueh-Jaw Lin, de la Universidad de Akron; H. David Sheets, del Canisius College; Geb Thomas, de la Universidad de Iowa; Brian Vick, del Instituto Politécnico y Universidad de Virginia; Jay Weitzen, de la Universidad de Massachussets, Lowell; y a Jane Patterson Fife, de la Universidad del Estado de Ohio. También me gustaría constatar el apoyo de Hayton, Ken Santor, Carolina Sieg, Catherine Hepburn, Simon Durkin, John Stout y Jay Beck, todos ellos de John Wiley & Sons.

Espero que este libro sea útil, así como de ayuda y provecho para todos los usuarios del software MATLAB.

Amos Gilat
Columbus, Ohio

A mis padres, Schoschana y Haim Gelbwacks

Índice analítico

Introducción

MATLAB es un potente lenguaje diseñado para la computación técnica. El nombre MATLAB proviene de Matrix LABoratory, dado que el tipo de dato básico que gestiona es una matriz (array). MATLAB puede ser utilizado en computación matemática, modelado y simulación, análisis y procesamiento de datos, visualización y representación de gráficos, así como para el desarrollo de algoritmos.

MATLAB es ampliamente conocido y utilizado en universidades e institutos para el aprendizaje en cursos básicos y avanzados de matemáticas, ciencias y, especialmente, ingeniería. En la industria se utiliza habitualmente en investigación, desarrollo y diseño de prototipos. El programa estándar de MATLAB comprende una serie de herramientas (funciones) que pueden ser utilizadas para resolver problemas comunes. Pero MATLAB incorpora, además, otras librerías específicas llamadas *toolboxes*, que son colecciones de funciones especializadas y diseñadas para resolver problemas muy específicos. Como ejemplos de estas colecciones se podrían citar las ideadas para el procesamiento de señales, el cálculo simbólico y el diseño de sistemas de control.

Hasta hace poco, la mayoría de los usuarios de MATLAB eran personas que tenían conocimientos previos sobre lenguajes de programación como FORTRAN o C, y que decidieron cambiarse a MATLAB una vez que este software se hizo suficientemente popular. En consecuencia, la mayor parte de la literatura disponible sobre MATLAB supone de una u otra forma que el lector posee conocimientos sobre programación. Los libros sobre MATLAB a menudo tratan aspectos avanzados o ampliaciones específicas para un campo muy concreto. En los últimos años, sin embargo, MATLAB se enseña en los institutos como el primer lenguaje de programación (y quizás el único) que aprenden los estudiantes. Para ellos es una necesidad el disponer de libros que enseñen MATLAB sin exigir experiencia anterior en programación.

El propósito de este libro

MATLAB: Introducción y aplicaciones está dirigido a estudiantes que utilizan MATLAB por primera vez y que no tienen conocimientos previos sobre programación. La obra se puede utilizar como un libro de texto para estudiantes de los primeros cursos de ingeniería, o incluso para charlas y seminarios orientados al aprendizaje de MATLAB. El presente libro también pretende ser una referencia en cursos más avanzados de ciencias e ingeniería, donde MATLAB se utilice básicamente como una herramienta para la resolución de problemas. También se pretende que este texto sirva para el autoaprendizaje de estudiantes e ingenieros. Además, también puede ser un complemento ideal en cursos donde se utiliza MATLAB, pero donde el profesor no tiene tiempo suficiente para cubrir todo el temario existente.

Conceptos recogidos en el libro

MATLAB es un software muy extenso, por ello es imposible tratar todos los temas relacionados en un solo libro. Este texto está enfocado principalmente a aprender los aspectos más básicos de MATLAB.

Una vez haya entendido estos conceptos, el estudiante podrá enfrentarse a otros aspectos más avanzados utilizando el menú de ayuda (Help) de la aplicación.

El orden de presentación de los diferentes temas ha sido elaborado cuidadosamente, a partir de varios años de experiencia en la enseñanza de MATLAB en un curso de introducción a la ingeniería. Los diferentes conceptos se presentan en un orden que permite al alumno seguir el libro secuencialmente de un capítulo a otro. Todos los conceptos se exponen de forma completa en el lugar que corresponde, y luego se utilizan en capítulos posteriores.

El primer capítulo describe la estructura y características básicas de MATLAB, así como la utilización del programa para resolver operaciones aritméticas muy básicas con escalares, como si se tratara de una calculadora. Los siguientes dos capítulos se dedican a las matrices o arrays. El dato básico con el que opera MATLAB es el array, el cual no requiere de dimensionado previo. Este concepto que hace de MATLAB un potente programa de cálculo, puede ser un tanto difícil de entender para estudiantes que hasta el momento sólo han trabajado con álgebra lineal y vectores. El libro esta escrito de forma que el concepto de array se introduce gradualmente para luego acabar profundizando mucho más en él. El Capítulo 2 explica cómo se crean arrays, y el Capítulo 3 trata las distintas operaciones matemáticas que se pueden llevar a cabo con ellos.

A partir de estos conceptos básicos, en el Capítulo 4 se presentan los ficheros script. El Capítulo 5 continúa con los gráficos bidimensionales para seguir avanzando, en el Capítulo 6, con el diseño de funciones. Las funciones han sido separadas intencionadamente de los ficheros script para poder adiestrar gradualmente al estudiante sin tener que exigirle conocimientos previos de programación. Así, la programación en MATLAB se aborda en el Capítulo 7, donde se presentan las instrucciones de flujo de un programa mediante sentencias condicionales y bucles.

Los siguientes tres capítulos abordan aspectos más avanzados. El Capítulo 8 trata sobre cómo MATLAB puede ser usado para realizar cálculos con polinomios, ajuste de curvas e interpolación. En el Capítulo 9 se trata la representación de gráficos en tres dimensiones, mientras que en el Capítulo 10 se ven conceptos sobre análisis numérico, como la resolución de ecuaciones no lineales, cálculo de máximos y mínimos de una función, integración numérica, así como la resolución de ecuaciones diferenciales ordinarias de primer orden. Finalmente, en el Capítulo 11 se verá en gran detalle cómo se utiliza MATLAB para realizar operaciones de cálculo simbólico. Este último capítulo es nuevo y se ha añadido a partir de esta segunda versión del libro.

Estructura típica de un capítulo del libro

En todos los capítulos los conceptos se introducen gradualmente de forma que los conceptos puedan ser asimilados fácilmente. El aprendizaje y uso de MATLAB se pone de manifiesto mediante el texto y los ejemplos descritos. Incluso alguno de los ejemplos, como los de los Capítulos 1-3, se muestran como tutoriales. Los ejemplos se muestran con un fondo gris oscuro junto con una explicación adicional resaltada en una caja de fondo blanco. La idea es que el lector pueda ir probando los ejemplos para ir obteniendo experiencia en el uso de MATLAB. Además, cada capítulo incluye problemas de ejemplo que son aplicaciones de MATLAB a la resolución de problemas matemáticos, científicos y técnicos. Cada ejemplo incluye el enunciado del problema y la solución de forma detallada. Algunos ejemplos se presentan en medio del capítulo. Todos los capítulos (excepto el Capítulo 2) tienen al final una sección con varios problemas de ejemplo de aplicaciones de MATLAB. Estos problemas pueden ser resueltos de distintas formas. Las soluciones a estos problemas están escritas de forma que se puedan seguir fácilmente. La idea es motivar a los alumnos para que puedan escribir sus propias soluciones y compararlas con las propuestas en el libro. Se incluyen problemas generales de las ramas de matemáticas, ciencias y problemas de diferentes disciplinas relacionadas con ingeniería.

Cálculo simbólico

MATLAB es básicamente un software pensado para trabajar con cálculo numérico. Sin embargo, se pueden realizar operaciones matemáticas simbólicas si se instala la correspondiente librería, es decir, *Symbolic Math*. Esta librería está incluida en la versión académica de MATLAB y puede ser añadida al programa estándar.

Software y hardware

El programa MATLAB, como otros tipos de software, está siendo actualizado continuamente, por lo que frecuentemente aparecen nuevas versiones. Este libro está basado en la versión de MATLAB 7.0, Release 14. Debe notarse, no obstante, que el libro cubre la parte básica de MATLAB, y por lo tanto los cambios de versión no son demasiado importantes a este respecto. Además, aunque el libro está basado en una versión del software que funciona bajo el sistema operativo Windows, todo lo que se explica en el texto puede probarse sin problemas en otras plataformas o máquinas. No obstante, se supone que el usuario tiene ciertos conocimientos básicos en el uso del sistema operativo y que dispone del software MATLAB correctamente instalado en su ordenador. Para obtener una mayor información sobre estos aspectos debe consultarse la documentación técnica de MATLAB referente al uso en otros sistemas operativos y ordenadores.

El orden en la explicación de conceptos dentro del libro

Escribir un libro con una ordenación conceptual adecuada para todo tipo de lectores es, probablemente, una tarea imposible. El orden de aparición de los conceptos en este libro es tal que los conceptos básicos de MATLAB (arrays y operaciones sobre arrays) se abordan primero, y, como ya se dijo, cada concepto se trata más exhaustivamente en otra parte para hacer del libro una referencia fácil de seguir o de consultar en un momento dado. Algunas personas, sin embargo, probablemente preferirían seguir el libro de una forma distinta, especialmente cuando se utiliza como libro de texto en una clase sobre MATLAB. Por ejemplo, algunos profesores podrían abordar la parte básica de ficheros scripts (las primeras cuatro secciones del Capítulo 4), antes de explicar los Capítulos 2 y 3. De la misma forma las operaciones lógicas y relacionales (Sección 7.1), junto con los polinomios (Sección 8.1), podrían ser consideradas operaciones matemáticas y presentadas, por tanto, junto con el resto de las operaciones del Capítulo 3.

Capítulo 1
Primeros pasos con MATLAB

Este capítulo describe las características y el propósito de las diferentes ventanas en MATLAB. En concreto, se presenta la Ventana de Comandos (Command Window), la cual se detalla ampliamente durante casi todo el capítulo. El Capítulo 1 muestra cómo usar MATLAB para realizar operaciones aritméticas con escalares, de forma similar a como se utiliza una calculadora de bolsillo. También se incluye el uso de funciones básicas con escalares y la definición de variables escalares (el operador de asignación), así como la utilización de esas variables en cálculos aritméticos.

1.1 Comenzando con MATLAB. Las ventanas

Para comenzar, se supone que el software está ya instalado en el ordenador desde el que se vaya a trabajar y que el usuario ha iniciado el programa. Una vez que el programa se inicia, la primera ventana que aparece (como se muestra en la Figura 1.1) contiene tres pequeñas ventanas en su interior denominadas, por un lado, la Ventana de Comandos (Command Window), la Ventana del Directorio Actual (Current Directory Window) y la Ventana del Histórico de Comandos (Command History Window). Esta es la visión por defecto que ofrece MATLAB, y en ella sólo aparecen tres de las ocho ventanas que hay en la aplicación. En la Tabla 1.1 se ofrece un listado más detallado del resto de las ventanas y su propósito. El botón **Start** (Inicio) que aparece en la parte inferior izquierda se utiliza para acceder a las herramientas y características diversas que contiene MATLAB.

A continuación se describen brevemente cuatro ventanas: la Ventana de Comandos, la Ventana de Gráficos (Figure Window), la Ventana del Editor (Editor Window) y la Ventana de Ayuda (Help Window) que se utilizan ampliamente a lo largo del libro. En capítulos posteriores se describen con más detalles.

Command Window (Ventana de Comandos): La Ventana de Comandos es la ventana principal de MATLAB. Se abre cuando se ejecuta la aplicación y es conveniente tenerla siempre abierta como única ventana visible. Para hacer esto sólo es necesario cerrar el resto de las ventanas (pulsando sobre el icono **x** en la parte superior derecha de la ventana que se quiera cerrar), o seleccionado la opción **Desktop Layout** (Distribución del Escritorio) en el menú **Desktop** (Escritorio), pulsando seguidamente sobre **Command Window Only** (Sólo Ventana de Comandos) en el submenú que se abre. La forma de trabajar en la Ventana de Comandos se describe en detalle en la Sección 1.2.

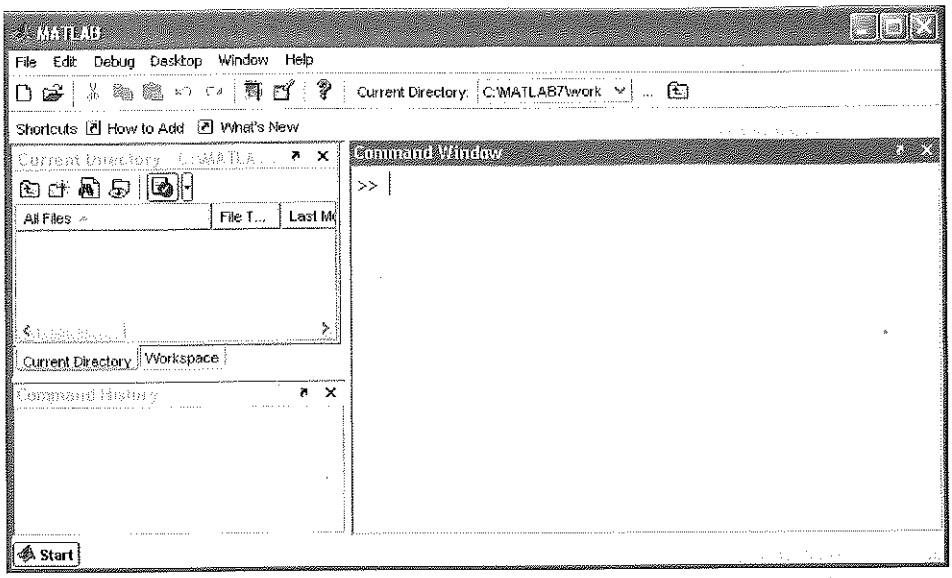


Figura 1.1: El escritorio de MATLAB.

Tabla 1.1: Las ventanas de MATLAB.

Nombre	Significado	Propósito
Command Window	Ventana de Comandos	Es la ventan principal, se utiliza para introducir variables y ejecutar programas.
Figure Window	Ventana de Gráficos	Se utiliza para visualizar gráficos MATLAB.
Editor Window	Ventana del Editor	Se usa para crear y depurar ficheros de *script* y funciones MATLAB.
Help Window	Ventana de Ayuda	Proporciona ayuda e información sobre MATLAB.
Launch Pad Window	Ventana de Plataforma	Da acceso a herramientas, demos y documentación.
Command History Window	Ventana del Histórico de Comandos	Almacena y visualiza los comandos que se introducen en la Ventana de Comandos
Workspace Window	Ventana del Espacio de Trabajo	Proporciona información sobre las variables utilizadas.
Current Directory Window	Ventana del Directorio de Trabajo Actual	Muestra los ficheros que hay en el directorio de trabajo actual.

Figure Window (Ventana de Gráficos): Esta ventana se abre automáticamente cuando un comando MATLAB ejecuta la visualización de un gráfico. La ventana muestra el gráfico creado por dicho comando. En la Figura 1.2 se muestra un ejemplo de esta ventana, aunque en el Capítulo 5 se da una descripción más detallada de su uso y funcionamiento.

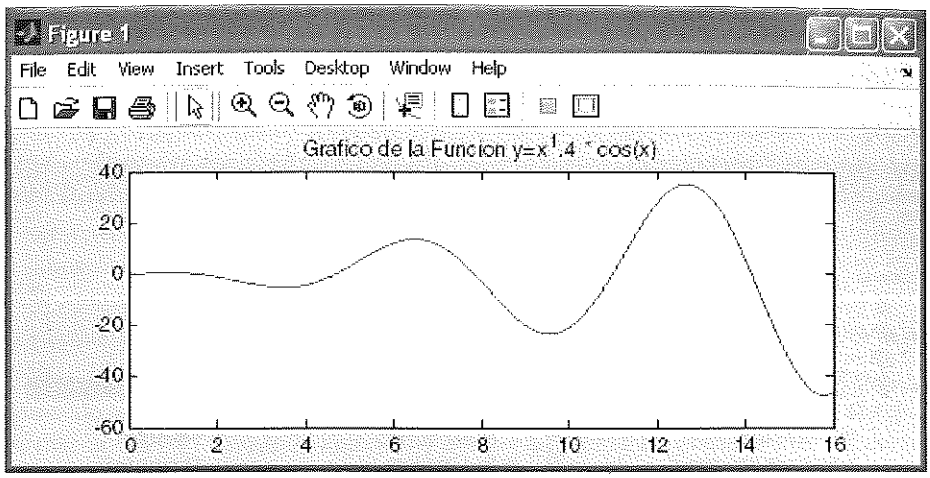


Figura 1.2: Ejemplo de una Ventana de Gráficos representando un gráfico MATLAB.

Editor Window (Ventana del Editor): Esta ventana se abre mediante las opciones del menú **File** y se utiliza para escribir y editar programas. En la Figura 1.3 aparece un ejemplo de este tipo de ventana. En el Capítulo 4, no obstante, se profundizará más en la creación de ficheros de *script*, y en el Capítulo 6 sobre la creación de funciones mediante la Ventana del Editor.

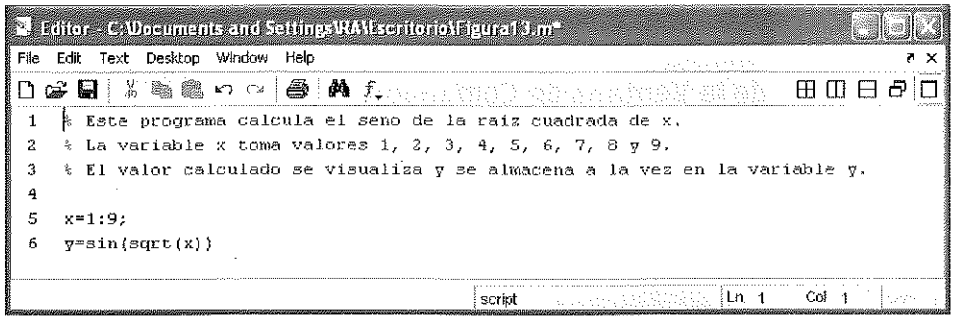


Figura 1.3: Ejemplo de una Ventana del Editor para la edición de código MATLAB.

Help Window (Ventana de Ayuda): La Ventana de Ayuda da acceso a la documentación para ayudar al usuario de MATLAB en el manejo de la aplicación. Esta ventana puede ser abierta directamente desde el menú Help que está en la barra de herramientas de la ventana principal de MATLAB. La Ventana de Ayuda es interactiva y puede ser utilizada para obtener ayuda sobre cualquier aspecto de MATLAB. La Figura 1.4 muestra una ventana de este tipo abierta.

Cuando MATLAB se inicia por primera vez, la pantalla tendrá un aspecto similar al que se muestra en la Figura 1.1 de la página 6. Para usuarios poco experimentados es mejor, quizás, cerrar todas las ventanas restantes excepto la Ventana de Comandos. Las ventanas que se han cerrados pueden ser reabiertas de nuevo mediante el menú **Desktop** (Escritorio). Las ventanas que se muestra en la

Figura 1.1 se pueden visualizar seleccionando primero **Desktop Layout** (Distribución del Escritorio), en el menú **Desktop** (Escritorio), y seleccionando finalmente **Default** (Distribución por Defecto) en el submenú correspondiente.

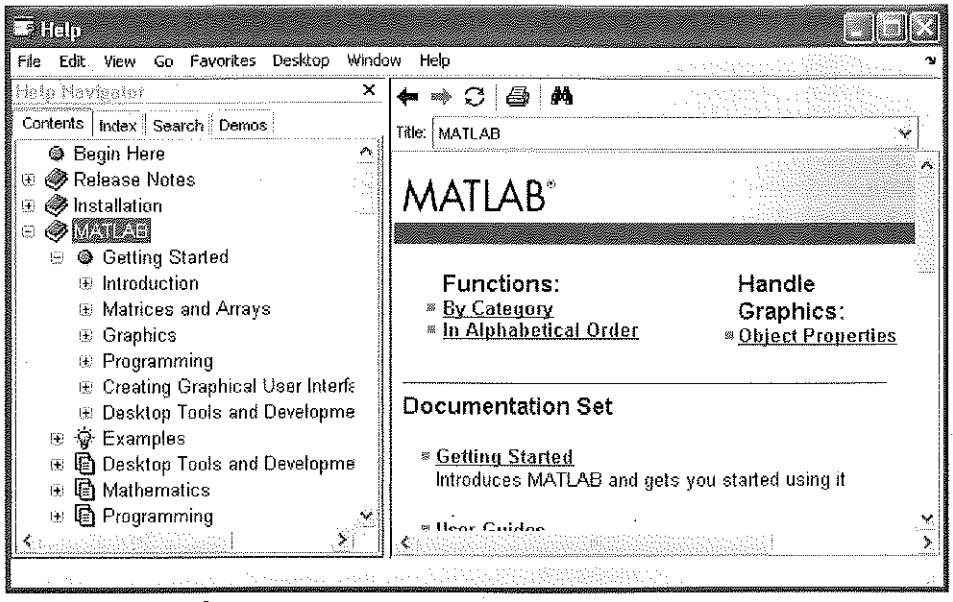


Figura 1.4: Ejemplo de una Ventana de Ayuda abierta.

1.2 Utilización de la Ventana de Comandos

La Ventana de Comandos es la ventana principal de MATLAB, y se utiliza para la ejecución de comandos, abrir otras ventanas, ejecutar programas escritos por el usuario y gestionar el software de MATLAB. En la Figura 1.5 se muestra un ejemplo de la Ventana de Comandos con un código simple MATLAB que será explicado con más detalle posteriormente en este capítulo.

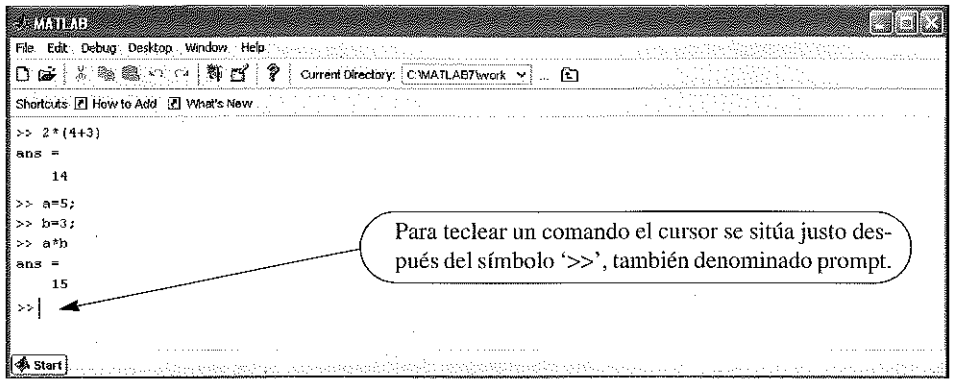


Figura 1.5: Ejemplo de una Ventana de Comandos.

Puntualizaciones para trabajar en la Ventana de Comandos:

- Para teclear un comando el cursor debe estar situado después del símbolo '>>', también denominado *prompt*.
- Una vez que el comando se ha tecleado y se pulsa la tecla **Intro**, el comando es ejecutado. Sin embargo, sólo se ejecuta el último comando. Todo lo ejecutado anteriormente permanece inalterado.
- Se puede teclear más de un comando en una sola línea. Para ello sólo hay que poner una coma entre comando y comando. Cuando se pulsa la tecla **Intro**, todos los comandos se ejecutan en orden de izquierda a derecha.
- No es posible ir hacia arriba, a una línea anterior, realizar una corrección y reejecutar de nuevo un comando.
- Un comando anteriormente tecleado puede ser invocado de nuevo. Para ello sólo hay que utilizar las flechas arriba (↑, Comando anterior) y abajo (↓, Comando posterior) de los cursores para localizar el comando deseado, visualizarlo en el *prompt* y hacer cuantas modificaciones sean necesarias antes de ejecutarlo de nuevo pulsando la tecla **Intro**.
- Si un comando es demasiado grande y ocupa más de una línea, éste se puede distribuir en una segunda línea tecleando al final de la primera puntos suspensivos (...) y pulsando la tecla **Intro**. Seguidamente se continúa la escritura del comando en la línea siguiente. Un comando de gran tamaño se podría distribuir en más de una línea hasta un límite máximo de 4096 caracteres.

El punto y coma (;):

Cuando se teclea un comando en la Ventana de Comandos y se pulsa la tecla **Intro**, el comando es ejecutado inmediatamente. Cualquier salida que genere el comando se visualizará en la Ventana de Comandos. Si se teclea un punto y coma (;) al final de un comando, la salida de dicho comando no será visualizada. Esto puede ser útil cuando el resultado de un comando es obvio o conocido, o cuando, por ejemplo, la salida es demasiado grande

Si se teclean varios comandos en la misma línea, la salida de cualquiera de esos comandos no se visualizará si entre los comandos se escribe un punto y coma en lugar de coma.

El símbolo %:

Cuando se teclea el símbolo % (tanto por cierto) al principio de una línea, MATLAB considerará dicha línea como un comentario. Esto significa que cuando se pulsa la tecla **Intro** la línea no será ejecutada. El carácter % seguido de texto (comentario) puede ser utilizado también después de un comando (en la misma línea). El comentario no tiene ningún efecto sobre la ejecución del comando.

Habitualmente no es necesario poner comentarios en la Ventana de Comandos. Éstos se utilizan más bien en programas para añadir descripciones o para explicar partes del código (ver Capítulos 4 y 6).

El comando `clc` *:*

El comando `clc` (teclear `clc` y pulsar **Intro**) borra la Ventana de Comandos. A medida que se van tecleando y ejecutando comandos en la Ventana de Comandos, ésta se va llenando. Una vez que se teclea `clc` la ventana se limpia. Este comando no cambia nada que haya sido creado antes. Por ejemplo, si previamente se han definido variables (ver Sección 1.6), éstas aún seguirán definidas, con sus valores asignados, y podrán seguir usándose normalmente. Esto tampoco afecta a los comandos que han sido tecleados anteriormente, los cuales pueden ser recuperados pulsando las flechas arriba y abajo.

1.3 Operaciones aritméticas con escalares

En este capítulo sólo veremos operaciones aritméticas con escalares, es decir, con números. Como explicaremos más adelante, los números pueden ser utilizados en cálculos aritméticos directamente (como si se tratara de una calculadora), o pueden ser asignados a variables que posteriormente podrán ser usadas en los cálculos. Los símbolos de operaciones aritméticas son los siguientes:

Operación	Símbolo	Ejemplo
Suma	+	5 + 3
Resta	-	5 – 3
Multiplicación	*	5 * 3
División derecha	/	5 / 3
División izquierda	\	5 \ 3 = 3 / 5
Exponenciación	^	5 ^ 3 (5^3 = 125)

Es importante decir que todos estos símbolos, excepto la división izquierda, son los mismos que pueden encontrarse en cualquier calculadora de mano. Para escalares, la división izquierda es la inversa de la división derecha. La división izquierda, sin embargo, se utiliza habitualmente para operaciones con matrices, que serán ampliamente explicadas en el Capítulo 3.

1.3.1 *Orden de precedencia*

MATLAB ejecuta los cálculos en función del orden de precedencia que se muestra en la relación siguiente. Este orden es el mismo que se utiliza en la mayoría de las calculadoras.

Precedencia	Operación matemática
Primero	Paréntesis. Para paréntesis anidados, el más interno es el que primero se ejecuta.
Segundo	Exponenciación.
Tercero	Multiplicación, división (igual precedencia).
Cuarto	Suma y resta.

En una expresión que contiene varias operaciones, las operaciones con mayor precedencia son ejecutadas antes que las operaciones que tienen menos. Si dos o más operaciones tienen la misma precedencia, la expresión será entonces ejecutada de izquierda a derecha. Como se detalla en la siguiente sección, se pueden utilizar paréntesis para cambiar el orden de los cálculos.

1.3.2 *Utilización de MATLAB a modo de calculadora*

La forma más simple de utilizar MATLAB es como si fuera una calculadora. Para hacer esto sólo es necesario teclear las expresiones matemáticas en la Ventana de Comandos y pulsar la tecla **Intro**. MATLAB calculará la expresión tecleada y responderá visualizando `ans =` y el resultado numérico de la expresión en la línea siguiente. En el Tutorial 1.1 se muestran distintos ejemplos de uso.

Tutorial 1.1: Utilización de MATLAB como una calculadora.

```
>> 7 + 8/2

ans =
   11

>> (7 + 8)/2

ans =
    7.5000

>> 4 + 5/3 + 1

ans =
    7.6667

>> 5^3/2

ans =
   62.5000

>> 27^(1/3) + 32^0.2

ans =
    5

>> 27^1/3 + 32^0.2

ans =
   11

>> 0.7854 - (0.7854)^3/(1*2*3) + 0.785^5/(1*2*3*4*5) ...
   - (0.785)^7/(1*2*3*4*5*6*7)

ans =
    0.7071
```

Teclear y pulsar **Intro**.

8/2 se ejecuta en primer lugar.

Teclear y pulsar **Intro**.

7+8 se ejecuta en primer lugar.

5/3 se ejecuta en primer lugar.

5^3 se ejecuta en primer lugar y /2 es ejecutado después.

1/3 se ejecuta primero, 27^(1/3) y 32^0.2 se ejecutan después, y + es ejecutado en último lugar.

27^1 y 32^0.2 se ejecutan primero, /3 se ejecuta después, y + se ejecuta lo último.

Si la expresión no cabe en una sola línea, teclee tres puntos seguidos '...' y pulse **Intro** para continuar escribiendo la expresión en la línea siguiente.

Esta última expresión se corresponde con los cuatro primeros términos de la serie de Taylor para sen(π/4).

1.4 Formatos de visualización de números

El usuario puede controlar el formato en que MATLAB visualiza la salida en la Ventana de Comandos. En el Tutorial 1.1, la salida estaba en formato de punto fijo con 4 dígitos decimales (formato `short`), que es el formato por defecto para valores numéricos. Esta visualización se puede cambiar mediante el comando `format`. Una vez que se introduce el comando `format`, todas las salidas que siguen serán visualizadas en el formato especificado. Los distintos formatos disponibles se presentan en la Tabla 1.2.

MATLAB dispone de muchos otros formatos para visualizar números. Para obtener más información sobre este comando puede teclearse `help format` en la Ventana de Comandos. El formato en el cual se visualizan los números no afecta a la forma en la que MATLAB realiza los cálculos ni a las variables almacenadas.

Tabla 1.2: Formatos de visualización de números.

Comando	Descripción	Ejemplo
`format short`	Punto fijo con 4 dígitos decimales. Resolución: $0.001 \leq$ *número* ≤ 1000 De los contrario el formato es `short e`.	>> 290/7 ans = 41.4286
`format long`	Punto fijo con 14 dígitos decimales. Resolución: $0.001 \leq$ *número* ≤ 100 De los contrario el formato es `long e`.	>> 290/7 ans = 41.42857142857143
`format short e`	Notación científica con 4 dígitos decimales.	>> 290/7 ans = 4.1429e+001
`format long e`	Notación científica con 15 dígitos decimales.	>> 290/7 ans = 4.142857142857143e+001
`format short g`	5 primeros dígitos fijos o en punto flotante.	>> 290/7 ans = 41.429
`format long g`	15 primeros dígitos fijos o en punto flotante.	>> 290/7 ans = 41.4285714285714
`format bank`	2 dígitos decimales.	>> 290/7 ans = 41.43
`format compact`	Elimina las líneas vacías para permitir más líneas con información en la ventana.	
`format loose`	Añade líneas vacías (lo contrario de `compact`).	

1.5 Funciones matemáticas básicas

Además de las operaciones aritméticas básicas, las expresiones que se construyen en MATLAB pueden contener funciones. MATLAB ofrece una vasta colección de librerías que contienen funciones predefinidas. Una función se compone básicamente de un nombre y de unos argumentos entre paréntesis. Por ejemplo, la función que calcula la raíz cuadrada de un número es `sqrt(x)`. Su nombre es `sqrt`, y el argumento es `x`. Cuando se utiliza esta función, se le puede pasar como argumento un número, una variable a la que se le ha asignado anteriormente un valor numérico (esto se explica en la Sección 1.6), o una expresión computable que puede estar compuesta por números y/o variables. Las funciones también pueden servir como parámetros y para componer expresiones más complejas. El Tutorial 1.2 muestra ejemplos de uso de la función `sqrt(x)` cuando MATLAB se utiliza a modo de calculadora con escalares.

Tutorial 1.2: **Utilización de la función** `sqrt`.

```
>> sqrt(64)                                   El argumento es un número.

ans =
   8

>> sqrt(50 + 14*3)                           El argumento es una expresión.

ans =
   9.5917

>> sqrt(54 + 9*sqrt(100))                    El argumento incluye una función.

ans =
   12

>> (15 + 600/4)/sqrt(121)                    La función está incluida en una expresión.

ans =
   15

>>
```

En las Tablas 1.3 a 1.5 se muestran algunas de las funciones predefinidas de MATLAB que se utilizan con mayor frecuencia. Para obtener un listado más amplio y detallado se puede utilizar la Ventana de Ayuda de MATLAB.

Tabla 1.3: Funciones matemáticas elementales.

Función	Descripción	Ejemplo
`sqrt(x)`	Raíz cuadrada.	>> sqrt(81) ans = 9
`exp(x)`	Exponencial (e^x).	>> exp(5) ans = 148.4132
`abs(x)`	Valor absoluto.	>> abs(-24) ans = 24
`log(x)`	Logaritmo natural Logaritmo de base e (ln).	>> log(1000) ans = 6.9078
`log10(x)`	Logaritmo en base 10.	>> log10(1000) ans = 3.0000
`factorial(x)`	Función factorial $x!$ (x debe ser un entero positivo).	>> factorial(5) ans = 120

Tabla 1.4: Funciones trigonométricas.

Función	Descripción	Ejemplo
`sin(x)`	Seno del ángulo x (x en radianes).	>> sin(pi/6) ans = 0.5000
`cos(x)`	Coseno del ángulo x (x en radianes).	>> cos(pi/6) ans = 0.8660
`tan(x)`	Tangente del ángulo x (x en radianes).	>> tan(pi/6) ans = 0.5774
`cot(x)`	Cotangente del ángulo x (x en radianes).	>> cot(pi/6) ans = 1.7321

Tabla 1.5: Funciones de redondeo.

Función	Descripción	Ejemplo
`round(x)`	Redondea al entero más próximo.	>> round(17/5) ans = 3
`fix(x)`	Redondea hacia cero.	>> fix(13/5) ans = 2
`ceil(x)`	Redondea hacia infinito.	>> ceil(11/5) ans = 3
`floor(x)`	Redondea hacia menos infinito.	>> floor(-9/4) ans = -3
`rem(x,y)`	Retorna el resto de la división de x entre y.	>> rem(13,5) ans = 3
`sign(x)`	Función de signo. Devuelve 1 si $x > 0$, -1 si $x < 0$, y 0 si $x = 0$.	>> sign(5) ans = 1

Las funciones trigonométricas inversas son: `asin(x)`, `acos(x)`, `atan(x)` y `acot(x)`. Las funciones trigonométricas hiperbólicas son: `sinh(x)`, `cosh(x)`, `tanh(x)` y `coth(x)`. Las tablas anteriores utilizan `pi` que se corresponde con el valor π (ver Sección 1.6.3).

1.6 Definición de variables escalares

Una variable es un nombre compuesto por una letra o una combinación de varias letras (y dígitos) al cual se le asigna un valor numérico. Una vez que se ha asignado un valor numérico a la variable, ésta puede ser utilizada en expresiones matemáticas, funciones y cualquier otro tipo de comando MATLAB. Una variable es, de hecho, el nombre de una posición de memoria. Cuando se define una nueva variable, MATLAB localiza un espacio apropiado de memoria donde se guardará el valor asignado a dicha variable. En realidad, al utilizar una variable se está utilizando el valor asignado a ella. Cuando a una variable existente se le asigna un nuevo valor, el contenido de esta posición de memoria es reemplazado con el nuevo valor, y el antiguo por tanto es eliminado. En este capítulo sólo consideraremos variables a las que se les han asignado valores numéricos escalares. La asignación y manipulación de variables multidimensionadas (arrays) se verá más delante en el Capítulo 2.

1.6.1 *El operador de asignación*

En MATLAB, el símbolo = se llama operador de asignación. Este operador asigna un valor a una variable.

```
Nombre_de_Variable = Valor numérico o expresión computable
```

- La parte izquierda de la operación de asignación sólo puede contener un nombre de variable. La parte derecha puede ser un número o una expresión computable que puede incluir números y/o variables a las que se les hayan asignado previamente valores numéricos. Cuando se pulsa la tecla **Intro**, MATLAB asigna a la variable el valor numérico de la parte derecha de la operación de asignación y muestra el valor asignado en las dos líneas siguientes.

  El siguiente ejemplo muestra cómo funciona el operador de asignación:

```
>> x = 15
x =
   15
>> x = 3*x - 12
x =
   33
>>
```

Se asigna el número 15 a la variable x.

MATLAB muestra el nombre de la variable y el valor asignado.

A x se le asigna un nuevo valor. El nuevo valor es 3 veces el antiguo valor de *x* menos 12.

La última sentencia ($x = 3x - 12$) muestra la diferencia entre el operador de asignación y el signo igual. Si en esta sentencia el signo = significara igual, el valor de x sería 6 (despejando el valor x en la ecuación).

La utilización de variables previamente definidas para crear nuevas variables se muestra en el siguiente ejemplo.

```
>> a = 12
a =
   12
>> B = 4
B =
   4
>> C = (a - B) + 40 - a/B*10
C =
   18
>>
```

Asigna el valor 12 a la variable a.

Asigna el valor 4 a la variable B.

Asigna el valor de la expresión de la parte derecha a la variable C.

- Cuando se teclea un punto y coma al final del comando y se pulsa la tecla **Intro**, MATLAB no visualiza el nombre de la variable y el valor asignado (aunque la variable existe y se ha almacenado correctamente en memoria).
- Si una variable ya existe, al teclear su nombre y pulsar **Intro** se visualizará la variable y el valor asignado en las dos líneas siguientes.

Así, si repetimos el último ejemplo utilizando punto y coma:

```
>> a = 12;
>> B = 4;
>> C = (a - B) 40 - a/B*10;
>> C
C =
   18
>>
```

Las variables a, B y C han sido definidas, pero su valor no se visualiza por haberse incluido un punto y coma al final de cada sentencia de asignación.

Se visualiza el nombre y el contenido de la variable C una vez que se ha tecleado el nombre de la variable y se ha pulsado la tecla **Intro**.

- Se puede realizar más de una asignación de variables en la misma línea. Para ello las asignaciones deben ir separadas por coma (se deben dejar también espacios después de cada coma). Cuando se pulsa la tecla **Intro**, las asignaciones se ejecutan de izquierda a derecha, y a continuación se visualizan las variables y sus asignaciones. Para evitar que se visualice una asignación sólo es necesario sustituir la coma por un punto y coma. Por ejemplo, las asignaciones de las variables a, B y C de arriba se pueden hacer en la misma línea, como se muestra en el ejemplo que sigue.

```
>>a = 12, B = 4; C = (a - B) + 40 - a/B*10
a =
   12
C =
   18
```

La variable B no se muestra, ya que se ha tecleado un punto y coma al final de la sentencia de asignación correspondiente.

- A una variable que ya existe se le puede reasignar un nuevo valor. Por ejemplo:

```
>> ABB = 72;
>> ABB = 9;
>> ABB
ABB =
   9
>>
```

Se asigna el valor 72 a la variable ABB.

Se asigna un nuevo valor de 9 a la variable ABB.

Cuando se teclea su nombre y se pulsa la tecla **Intro** se muestra el valor actual de la variable.

- Una vez que una variable ha sido definida, ésta puede ser utilizada como argumento en llamadas a funciones. Por ejemplo:

```
>> x = 0.75;
>> E = sin(x)^2 + cos(x)^2
E =
   1
>>
```

1.6.2 *Reglas sobre el nombre de las variables*

- Pueden tener una longitud de hasta 63 caracteres en MATLAB 7 (31 caracteres en MATLAB 6.0).
- Pueden contener letras, dígitos y el carácter de subrayado.
- Deben empezar por una letra.
- MATLAB es un lenguaje que distingue entre letras mayúsculas y minúsculas. Por ejemplo: `AA`, `Aa`, `aA` y `aa` son nombres de cuatro variables diferentes.
- Hay que evitar poner a las variables el nombre de funciones del sistema (por ejemplo: `cos`, `sin`, `exp`, `sqrt`, etc.). Si se utiliza el nombre de una función para definir una variable, esa función no podrá ser utilizada.

1.6.3 *Variables predefinidas*

Hay una serie de variables ya predefinidas en MATLAB que pueden ser utilizadas por el usuario y que contienen valores también constantes o predefinidos. Estas variables son las siguientes:

- `ans` Esta variable contiene el resultado de la última sentencia que no ha sido asignada a un valor específico (ver Tutorial 1.1). Si el usuario no asigna el valor de una expresión a una variable, MATLAB automáticamente guarda el resultado en `ans`.
- `pi` Es el número π.
- `eps` Representa la diferencia más pequeña entre dos números. Es igual a 2^(-52), que es aproximadamente 2.2204e-016.
- `inf` Representa el infinito.
- `i` Se define como la raíz cuadrada de -1, es decir: 0 + 1.0000i.
- `j` Es equivalente a i.
- `NaN` Es la abreviatura de Not a Number (no numérico). Se usa cuando MATLAB no puede determinar un valor numérico válido. Por ejemplo en la operación 0/0.

Estas variables pueden ser redefinidas para cambiar su valor por defecto. La variables `pi, eps` e `inf` no se suelen redefinir comúnmente ya que se utilizan con frecuencia en muchas aplicaciones. Otras variables predefinidas, como `i` y `j` se redefinen a veces (comúnmente en la creación de bucles), siempre y cuando no se utilicen números complejos en el código.

1.7 Comandos útiles en el uso de variables

Los siguientes comandos pueden ser utilizados para eliminar variables o para obtener información sobre variable que hayan sido creadas previamente. Cuando estos comandos se teclean y se pulsa la tecla **Intro**, proporcionan información o ejecutan las tareas que se muestran a continuación:

Comando	Resultado
`clear`	Borra todas las variables de la memoria.
`clear x y z`	Borra sólo las variables `x, y, z` de la memoria.
`who`	Muestra un listado de las variables almacenadas en memoria.
`whos`	Muestra un listado de las variables almacenadas en memoria y su tamaño, junto con la información sobre su clase y longitud.

1.8 Ejemplos de aplicación con MATLAB

Problema de ejemplo 1.1: Identidad trigonométrica

Sea la siguiente identidad trigonométrica:

$$\cos^2\frac{x}{2} = \frac{\tan x + \operatorname{sen} x}{2\tan x}$$

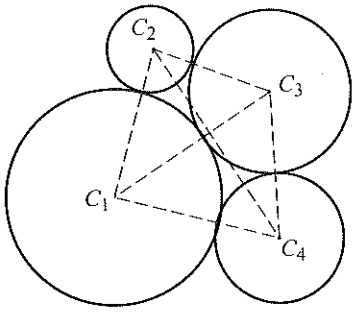
Verifique que dicha identidad es correcta calculando ambos miembros de la ecuación, sustituyendo el valor de x por $x = \frac{\pi}{5}$.

Solución

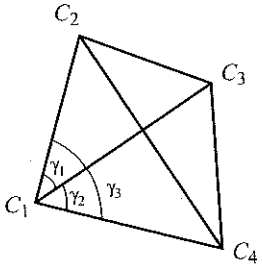
```
>> x = pi/5;                                   Define x.
>> LHS = cos(x/2)^2                            Calcula el primer miembro de la ecuación.
LHS =
   0.9045
>> RHS = (tan(x) + sin(x))/(2*tan(x))          Calcula el segundo miembro de la ecuación.
RHS =
   0.9045
>>
```

Problema de ejemplo 1.2: Geometría y trigonometría

Sean los cuatro círculos que se muestran en la figura adjunta. Estos círculos están en contacto tangencialmente uno con otro en un punto, tal y como se muestra en la figura. Determinar la distancia entre los centros C_2 y C_4.
Los radios R_i de los círculos son los siguientes:
$R_1 = 16$ mm, $R_2 = 6{,}5$ mm, $R_3 = 12$ mm y $R_4 = 9{,}5$ mm.



Solución



Las líneas que conectan los centros de los círculos forman cuatro triángulos. En dos de estos triángulos, $\Delta C_1C_2C_3$ y $\Delta C_1C_2C_4$, las longitudes de los lados son conocidas. Esta información se utilizará para calcular los ángulos γ_1 y γ_2 en dichos triángulos utilizando la ley de los cosenos. Por ejemplo, γ_1 puede calcularse a partir de la expresión:

$$(C_2C_3)^2 = (C_1C_2)^2 + (C_1C_3)^2 - 2(C_1C_2)(C_1C_3)\cos\gamma_1$$

Seguidamente, la longitud del lado C_2C_4 puede ser calculada teniendo en cuenta el triángulo $\Delta C_1C_2C_4$. Esto se hace, de nuevo, aplicando la ley de los cosenos (las longitudes C_1C_2 y C_1C_4 son conocidas y el ángulo γ_3 es la suma de los ángulos γ_1 y γ_2).

```
>> R1 = 16; R2 = 6.5; R3 = 12; R4 = 9.5;
>> C1C2 = R1 + R2; C1C3 = R1 + R3; C1C4 = R1 + R4;
>> C2C3 = R2 + R3; C3C4 = R3 + R4;
>> Gama1 = acos(C1C2^2 + C1C3^2 - C2C3^2)/(2*C1C2*C1C3));
>> Gama2 = acos(C1C3^2 + C1C4^2 - C3C4^2)/(2*C1C3*C1C4));
>> Gama3 = Gama1 + Gama2;
>> C2C4 = sqrt(C1C2^2 + C1C4^2-2*C1C2*C1C4*cos(Gama3))
C2C4 =
   33.5051
```

Define los radios.

Calcula las longitudes de los lados.

Calcula γ_1, γ_2 y γ_3.

Calcula la longitud del lado C_2C_4.

Problema de ejemplo 1.3: Transferencia de calor

Un objeto con una temperatura inicial T_0 se introduce en el instante $t = 0$ dentro de una cámara que tiene una temperatura constante T_3. Entonces, el objeto experimenta un cambio de temperatura que se corresponde con la ecuación:

$$T = T_s + (T_0 - T_s)e^{-kt}$$

donde T es la temperatura del objeto en el instante t, y k es una constante.

Una lata de soda, con una temperatura de 120° F (la dejaron olvidada dentro del coche), se introduce en un frigorífico que tiene en su interior una temperatura de 38° F. Calcular, redondeando el resultado al grado más próximo, la temperatura de la lata después de tres horas. Considerar $k = 0{,}45$. Deben definirse primero todas las variables y seguidamente se calculará la temperatura utilizando un solo comando MATLAB.

Solución

```
>> Ts = 38; T0 = 120; k = 0.45; t = 3;
>> T = round(Ts + (t0-Ts)*exp(-k*t))
T =
   59
>>
```

Redondeo al entero más próximo.

Problema de ejemplo 1.4: Interés compuesto

El saldo o monto B de una cuenta de ahorros después de t años cuando se deposita un capital P a una tasa de interés anual r, con n períodos de capitalización anuales, viene dado por la siguiente ecuación:

$$B = P\left(1 + \frac{r}{n}\right)^{nt} \qquad (1)$$

Si los intereses se capitalizan anualmente, el monto puede expresarse de la forma:

$$B = P(1+r)^t \qquad (2)$$

En una cuenta de ahorro se invierten 5000 € durante un período de 17 años, con un interés compuesto con capitalización anual. En una segunda cuenta se invierten otros 5000 €, pero esta vez a un interés compuesto con capitalización mensual. En ambas cuentas la tasa de interés es del 8,5%. Utilizar MATLAB para determinar cuanto tiempo (en años y meses) tarda el monto de la segunda cuenta en ser igual que el de la primera después del período de 17 años.

Solución

Se seguirán los siguientes pasos:
a) Calcular *B* para los 5000 € invertidos con interés compuesto anual después de 17 años, utilizando la ecuación (2).
b) Calcular *t* para el monto *B* calculado en el punto anterior (*a*) para el interés compuesto mensualmente, utilizando la ecuación (1).
c) Determinar el número de años y meses que corresponden a *t*.

```
>> P = 5000; r = 0.085; ta = 17; n = 12;
>> B = P*(1 + r)^ta
B =
   2.0011e + 004
>> t = log(B/P)/n*log(1 + r/n))
t =
   16.3737
>> years = fix(t)
years =
   16
>> months = ceil((t - years)*12)
months =
   5
```

Paso (*a*): Calcular `B` con la ecuación (2).

Paso (*b*): Despejar *t* de la ecuación (1) y calcular `t`.

Paso (*c*): Determinar el número de años.

Determinar el número de meses.

1.9 Problemas

Resuelva los siguientes problemas utilizando la Ventana de Comandos de MATLAB.

1. Calcule:

   a) $\dfrac{35{,}7 \cdot 64 - 7^3}{45 + 5^2}$

   b) $\dfrac{5}{4} \cdot 7 \cdot 6^2 + \dfrac{3^7}{(9^3 - 652)}$

2. Calcule:

   a) $(2+7)^3 + \dfrac{273^{2/3}}{2} + \dfrac{55^2}{3}$

   b) $2^3 + 7^3 + \dfrac{273^3}{2} + 55^{3/2}$

3. Calcule:

   a) $\dfrac{3^7 \log(76)}{7^3 + 546} + \sqrt[3]{910}$

   b) $43 \cdot \dfrac{\left(\sqrt[4]{250} + 23\right)^2}{e^{(45-3^3)}}$

4. Calcule:

   a) $\cos^2\left(\dfrac{5\pi}{6}\right) \operatorname{sen}\left(\dfrac{7\pi}{8}\right)^2 + \dfrac{\tan\left(\dfrac{\pi}{6}\ln 8\right)}{\sqrt{7}}$

   b) $\cos\left(\dfrac{5\pi}{6}\right)^2 \operatorname{sen}^2\left(\dfrac{7\pi}{8}\right) + \dfrac{\tan\left(\dfrac{\pi}{6}\ln 8\right)}{7 \cdot \dfrac{5}{2}}$

5. Defina la variable x como $x = 13{,}5$, y calcule:

   a) $x^3 + 5x^2 - 26{,}7x - 52$

   b) $\dfrac{\sqrt{14x^3}}{e^{3x}}$

   c) $\log\left|x^2 - x^3\right|$

6. Defina las variables x y z como $x = 9{,}6$ y $z = 8{,}1$, y calcule:

   a) $xz^2 - \left(\dfrac{2z}{3x}\right)^{3/5}$

   b) $\dfrac{443z}{2x^3} + \dfrac{e^{-xz}}{(x+z)}$

7. Defina las variables *a, b, c* y *d* como:
   $a = 15{,}62$, $b = -7{,}08$, $c = 62{,}5$ y $d = 0{,}5\,(ab - c)$, y calcule:

   a) $a + \dfrac{ab}{c}\dfrac{(a+d)^2}{\sqrt{|ab|}}$ *b*) $de^{\left(\frac{d}{2}\right)} + \dfrac{\dfrac{ad+cd}{\dfrac{20}{a}+\dfrac{30}{b}}}{(a+b+c+d)}$

8. Calcule (escribiendo un solo comando) el radio r de una esfera de 350 cm^3 de volumen. Una vez calculado r, utilice este valor para calcular el área de la superficie de la esfera.

9. Dadas las siguientes identidades trigonométricas:

   a) $\operatorname{sen} 2x = 2 \operatorname{sen} x \cos x$

   b) $\cos \frac{x}{2} = \sqrt{\frac{1 + \cos x}{2}}$

   Verifique que ambas son correctas calculando para ello cada lado de la identidad, sustituyendo el valor de x por $x = \frac{5}{24}\pi$.

10. Dadas las siguientes identidades trigonométricas:

    a) $\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}$

    b) $\tan \frac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}}$

    Verifique que ambas son correctas calculando para ello cada lado de la identidad, sustituyendo el valor de x por $x = \frac{3}{17}\pi$.

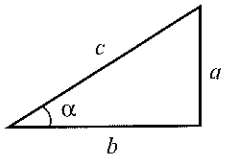
11. Defina dos variables: *alpha* = $5\pi/9$ y *beta* = $\pi/7$. Utilice estas variables para demostrar que la siguiente identidad trigonométrica es correcta. Calcule para ello ambos lados de la identidad a partir de su ecuación.

$$\cos \alpha - \cos \beta = 2 \operatorname{sen} \frac{1}{2}(\alpha + \beta) \operatorname{sen} \frac{1}{2}(\beta - \alpha)$$

12. En el triángulo adjunto $a = 11$ cm y $c = 21$ cm. Defina las variables a y c y calcule:

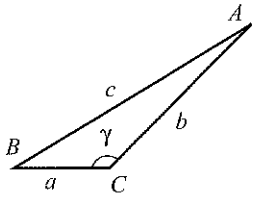
    a) El valor b a partir del teorema de Pitágoras, utilizando una sola línea en la Ventana de Comandos.

    b) El ángulo α en grados, utilizando para ello el valor b calculado anteriormente junto con la función `acos(x)`. Hágalo empleando una sola línea de la Ventana de Comandos.



13. En el triángulo adjunto $a = 18$ cm, $b = 35$ cm y $c = 50$ cm. Defina a, b y c como variables y posteriormente calcule el ángulo γ (en grados) sustituyendo las variable en la ecuación de la regla de los cosenos.

    (La regla o ley de los cosenos: $c^2 = a^2 + b^2 - 2ab \cos \gamma$)



14. La distancia d de un punto (x_0, y_0) a una recta $Ax + By + C = 0$ viene dada por:

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

Determine la distancia del punto (2, –3) a la recta $3x + 5y - 6 = 0$. Primero defina las variables A, B, C, x_0 e y_0. Después calcule d. Utilice las funciones `abs` y `sqrt`.

15. Se empaquetan ramos de flores en cajas, de forma que en cada caja se introduce una docena de flores. Determinar cuántas cajas son necesarias para empaquetar 751 flores. Utilice la función `ceil`.

16. Defina las siguientes variables:

    precio_mesa = 256,95 €

    precio_silla = 89,99 €

    Seguidamente cambie el formato de visualización a `bank` y:

    a) Calcule el coste de dos mesas y ocho sillas.

    b) Lo mismo que en *a*), pero añada un 5,5% de IVA.

    c) Lo mismo que en *b*), pero redondee el total del coste al euro más próximo.

17. Cuando se suman fracciones debe calcularse el mínimo común múltiplo para poder realizar la operación correctamente. Por ejemplo, el mínimo común múltiplo de 1/4 y 1/10 es 20. Utilice la Ventana de Ayuda de MATLAB para encontrar una función apropiada que calcule el mínimo común múltiplo de dos números. Utilice después esa función para demostrar que el mínimo común múltiplo de:

    a) 4 y 10 es 20.

    b) 6 y 38 es 114.

18. La magnitud M de un terremoto en la Escala Richter viene dada por: $M = \frac{2}{3} \log \frac{E}{E_0}$, donde E es la energía emitida por el terremoto y $E_0 = 10^{4,4}$ julios es una constante (energía de un terremoto más pequeño de referencia). Determine cuantas veces más energía emite un terremoto que registra 7,2 en la Escala Richter respecto a otro que registra 5,3.

Capítulo 2
Arrays

El array es una estructura fundamental que MATLAB utiliza para almacenar y manipular datos. Es una lista de números dispuestos en filas y/o columnas. En un array unidimensional los números están agrupados en filas o columnas, a diferencia de los arrays bidimensionales, donde los elementos se distribuyen en filas y columnas. En los arrays se puede almacenar información de manera similar a una tabla. En las disciplinas científicas, a los arrays unidimensionales se les denomina comúnmente vectores, mientras que a los bidimensionales se les denomina matrices. En este Capítulo 2 se narra cómo crear y manipular arrays, mientras que el Capítulo 3 explica cómo usar arrays en operaciones matemáticas. Además de los arrays numéricos, en MATLAB los arrays pueden estar compuestos por caracteres; en este caso se denominan cadenas o *strings*. Estas estructuras se verán con más en detalle en la Sección 2.10.

2.1 Creación de arrays unidimensionales (vectores)

Un array unidimensional es una sucesión de números distribuidos en una fila o en una columna. Un ejemplo de este tipo de representación es la posición de un punto en el espacio mediante un sistema tridimensional de coordenadas cartesianas. Como muestra la Figura 2.1, la posición del punto A queda definida a partir de una sucesión de tres números: 2, 4 y 5, que son las coordenadas del punto en cuestión.

La posición del punto A puede ser representada en términos de su vector posición:

$$\mathbf{r}_A = 2\mathbf{i} + 4\mathbf{j} + 5\mathbf{k}$$

donde **i**, **j** y **k** son los vectores unitarios en la dirección de los ejes x, y y z, respectivamente. Los números 2, 4 y 5 se pueden usar para definir un vector fila o columna.

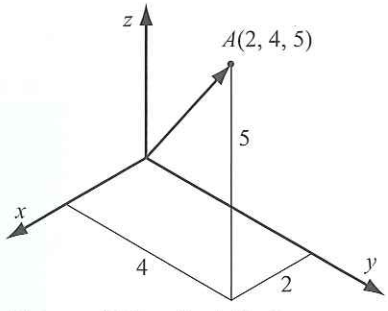


Figura 2.1: Posición de un punto.

Cualquier sucesión de números puede ser transformada en un vector. En la Tabla 2.1, por ejemplo, se muestran datos sobre crecimiento demográfico. Estos datos se pueden utilizar para crear dos listas de números: una para los años y otra para la población. Los números en cada lista se pueden poner en una fila o en una columna.

En MATLAB, un vector se crea asignando sus elementos a una variable. Esto se puede hacer de distintas formas, dependiendo del origen de la información utilizada para componer los elementos del vector. Cuando un vector contiene números que son

Tabla 2.1: Datos demográficos.

Año	1984	1986	1988	1990	1992	1994	1996
Población (Millones)	127	130	136	145	158	178	211

conocidos (por ejemplo, coordenadas de un punto *A*), el valor de cada elemento se introduce directamente. Cada uno de estos elementos puede ser también una expresión matemática con variables predefinidas, números y funciones. A menudo, los elementos de un vector fila son series de números con distancia o espaciado constante. En estos casos el vector puede ser creado a partir de comandos MATLAB. Un vector se puede crear también a partir del resultado de una operación matemática, como se verá en el Capítulo 3.

Creación de un vector a partir de una lista de números conocidos:

Para crear el vector sólo es necesario teclear sus elementos (números) dentro de un par de corchetes [ ].

```
nombre_variable = [ elementos del vector ]
```

Vector fila: para crear un vector fila deben teclearse los elementos con un espacio o una coma entre cada elemento, dentro de los corchetes.

Vector columna: para crear un vector columna deben teclearse los elementos con un punto y coma entre cada elemento, o pulsando la tecla **Intro** después de cada elemento dentro de los dos corchetes.

El Tutorial 2.1 muestra cómo se pueden crear vectores fila y columna a partir de los datos de la Tabla 2.1 y de las coordenadas del punto *A* vistas anteriormente.

Tutorial 2.1: Creación de vectores a partir de una serie de datos.

```
>> yr = [1984 1986 1988 1990 1992 1994 1996]
yr =
     1984   1986   1988   1990   1992   1994   1996
>> pop = [127; 130; 136; 145; 158; 178; 211]
pop =
   127
   130
   136
   145
   158
   178
   211
>> pmtAH = [2, 4, 5]
pmtAH =
   2   4   5
```

Se asigna la lista de años al vector fila `yr`.

Se asigna la lista de datos demográficos al vector columna `pop`.

Se asignan las coordenadas del punto *A* al vector fila `pntAH`.

Tutorial 2.1 (continuación): Creación de vectores a partir de una serie de datos.

```
>> pntAV = [2
   4
   5]

pmtAV =
   2
   4
   5

>>
```

Se asignan las coordenadas del punto *A* al vector columna `pntAV`.
(Después de teclear cada elemento hay que pulsar la tecla **Intro**.)

Creación de un vector con distancia constante a partir de la especificación del primer término, de la distancia y del último término:

En un vector con distancia o espaciado constante, la diferencia entre los elementos es siempre la misma. Por ejemplo, en el vector: $v = 2\ 4\ 6\ 8\ 10$, la distancia entre los elementos es 2. Un vector donde el primer término es m, el espacio o distancia es q, y el último término es n, se puede crear a partir de la expresión:

`nombre_variable = [m:q:n]` o bien `nombre_variable = m:q:n`

(los corchetes son opcionales)

He aquí algunos ejemplos:

```
>> x = [1: 2:13]

x =
   1   3   5   7   9   11   13

>> y = [1.5:0.1:2.1]

y =
   1.5000   1.6000   1.7000   1.8000   1.9000   2.0000   2.1000

>> z = [-3:7]

z =
   -3   -2   -1   0   1   2   3   4   5   6   7

>> xa = [21:-3:6]

xa =
   21   18   15   12   9   6

>>
```

El primer elemento es 1, la distancia es 2, el último elemento es 13.

El primer elemento es 1,5, la distancia es 0,1, el último elemento es 2,1.

El primer elemento es –3, el último es 7.
Si se omite, la distancia por defecto es 1.

El primer elemento es 21, la distancia es –3, el último elemento es 6.

- Si los números `m`, `q` y `n` son tales que el valor de `n` no se puede obtener añadiendo las diferencias `q` a `m`, entonces (para un `n` positivo) el último elemento del vector será el último número que no exceda `n`.

Creación de un vector con distancia constante a partir de la especificación del primer y último término, así como del número de términos:

Un vector en el cual el primer elemento es *xi*, el último elemento es *xf*, y el número de elementos es *n*, puede ser creado utilizando el comando `linspace` (MATLAB determina, en este caso, la distancia correcta entre los elementos):

```
nombre_variable = linspace(xi, xf, n)
```

He aquí algunos ejemplos:

```
>> va = linspace(0, 8, 6)                    6 elementos, el primero es 0 y el último 8.
va =
   0   1.6000   3.2000   4.8000   6.4000   8.0000
>> vb = linspace(30, 10, 11)                 11 elementos, el primero es 30 y el último 10.
vb =
   30   28   26   24   22   20   18   16   14   12   10
>> u = linspace(49, 5, 0.5)                  El primer elemento es 49,5, el último es 0,5.

                                             Si se omite el número de elementos,
u =                                          éste es, por defecto, 100.
   Columns 1 through 10
   49.5000   49.0051   48.5101   48.0152   47.5202   47.0253   46.5303   46.0354   45.5404   45.0455
   .........                                 Se visualizan 100 elementos.
   Columns 91 through 100
   4.9554   4.4596   3.9646   3.4697   2.9747   2.4798   1.9848   1.4899   0.9949   0.5000
>>
```

2.2 Creación de arrays bidimensionales (matrices)

Un array bidimensional, también llamado matriz, distribuye los números en columnas y filas. Las matrices se usan para almacenar información como si fuera una tabla. Las matrices juegan un papel muy importante en el álgebra lineal, y son comúnmente usadas en ciencias para modelizar y resolver problemas.

En una matriz cuadrada, el número de filas y columnas coincide. Por ejemplo, la matriz:

$$\begin{matrix} 7 & 4 & 9 \\ 3 & 8 & 1 \\ 6 & 5 & 3 \end{matrix}$$ Matriz de 3 × 3

es cuadrada, ya que está formada por tres filas y tres columnas. En general, el número de filas y de columnas puede ser diferente. Por ejemplo, la matriz:

$$\begin{matrix} 31 & 26 & 14 & 18 & 5 & 30 \\ 3 & 51 & 20 & 11 & 43 & 65 \\ 28 & 6 & 15 & 61 & 34 & 22 \\ 14 & 58 & 6 & 36 & 93 & 7 \end{matrix}$$ Matriz de 4 × 6

tiene cuatro filas y seis columnas. Una matriz $m \times n$ tiene m filas y n columnas, además el producto de m por n nos da el tamaño de la matriz (número de elementos).

Para crear una matriz sólo hay que asignar los valores correspondientes a una variable. Para hacer esto es necesario teclear los elementos, fila por fila, entre corchetes [ ]. Primero se escribe el corchete izquierdo [, seguidamente se teclea la primera columna separando los elementos por espacio o comas. Antes de teclear la siguiente fila hay que poner un punto y coma o pulsar **Intro**. Finalmente se pone el corchete derecho] al final de la última fila.

nombre_variable = [elementos de la 1ª fila; elementos de la 2ª fila; elementos de la 3ª fila; ..... ; elementos de la última fila]

Los elementos de una matriz pueden ser números o expresiones matemáticas que incluyan números, variables predefinidas y funciones. Todas las filas deben tener el mismo número de elementos. Si un elemento es cero, también debe ser tecleado. MATLAB visualiza un mensaje de error si se intenta definir una matriz incompleta. El Tutorial 2.2 muestra ejemplos de matrices definidas de distinta forma.

Tutorial 2.2: Creación de matrices.

```
>> a = [5 35 43; 4 76 81; 21 32 40]

a =
    5   35   43
    4   76   81
   21   32   40

>> b = [7 2 76 33 8
   1 98 6 25 6
   5 54 68 9 0]

b =
    7    2   76   33    8
    1   98    6   25    6
    5   54   68    9    0

>> cd = 6; e = 3; h = 4;

>> Mat = [e, cd*h, cos(pi/3); h^2, sqrt(h*h/cd), 14]

Mat =
    3.0000   24.0000    0.5000
   16.0000    1.6330   14.0000

>>
```

Se pone un punto y coma después de cada línea (antes de la siguiente fila).

Se pulsa la tecla **Intro** después de cada línea (antes de la siguiente fila).

Se visualizan 100 elementos.

Se definen los elementos de la matriz a partir de distintas expresiones matemáticas.

Las filas de una matriz se pueden introducir también como vectores utilizando la notación para crear vectores con distancia constante, o utilizando también el comando `linspace`. Por ejemplo:

```
>> A = [1:2:11; 0:5:25; linspace(10, 60, 6); 67 2 43 68 4 13]
A =
    1    3    5    7    9   11
    0    5   10   15   20   25
   10   20   30   40   50   60
   67    2   43   68    4   13
>>
```

En este ejemplo, las primeras dos filas se introducen como vectores utilizando la notación de distancia o espaciado constante. La tercera fila se introduce utilizando el comando `linspace`, mientas que en la última fila se introducen los elementos individualmente.

2.2.1 *Los comandos* `zeros`, `ones` *y* `eye`

Los comandos `zeros(m, n)`, `ones(m, n)` y `eye(n)` se utilizan para crear matrices que contendrán elementos con valores especiales. Los comandos `zeros(m, n)` y `ones(m, n)` crean matrices de *m* filas y *n* columnas en las que todos los elementos son unos y ceros respectivamente. El comando `eye(n)` crea una matriz cuadrada de *n* filas y *n* columnas en la cual los elementos de la diagonal son unos, siendo ceros el resto de los elementos. Esta matriz también se denomina matriz identidad. Estos son algunos ejemplos de uso:

```
>> zr = zeros(3,4)
zr =
     0     0     0     0
     0     0     0     0
     0     0     0     0
>> ne = ones(4,3)
ne =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
>> idn = eye(5)
idn =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
>>
```

Las matrices también se pueden crear a partir del resultado de operaciones matemáticas con vectores y matrices. Este tipo de operaciones se explican con detalle en el Capítulo 3.

2.3 Puntualizaciones sobre las variables en MATLAB

- Todas las variables en MATLAB son arrays. Un escalar es simplemente un array de un solo elemento. Un vector es un array con una sola fila o columna de elementos. Una matriz es un array con elementos distribuidos en filas y columnas.
- Las variables (escalares, vectores o matrices) se definen a partir de la entrada, cuando a la variable se le asigna un valor. No hay necesidad de definir explícitamente el tamaño del array antes de que se produzca la asignación.
- Una vez que la variable existe como escalar, vector o matriz, su contenido y tamaño puede cambiar. Por ejemplo, un escalar puede transformarse en una matriz o un vector, así como una matriz puede convertirse en un escalar. Estos cambios se llevan a cabo mediante la eliminación o inserción de nuevos elementos. Esto se explicará con detalle en la Secciones 2.7 y 2.8.

2.4 El operador de transposición

El operador de transposición, cuando se aplica a un vector, transforma la fila en columna y viceversa. Cuando se aplica a una matriz, éste transforma las filas en columnas y viceversa. El operador de transposición se expresa tecleando una comilla simple ' a continuación de la variable que se va a transponer. Por ejemplo:

```
>> aa = [3 8 1]
aa =
 3 8 1
>> bb = aa'
bb =
   3
   8
   1
>> C = [2 55 14 8; 21 5 32 11; 41 64 9 1]
C =
    2   55   14    8
   21    5   32   11
   41   64    9    1
>> D = C'
D =
    2   21   41
   55    5   64
   14   32    9
    8   11    1
>>
```

Define un vector fila aa.

Define un vector fila bb a partir de la transpuesta de aa.

Define una matriz C con 3 filas y 4 columnas.

Define una matriz D a partir de la transpuesta de la matriz D. (D tiene 4 filas y 3 columnas.)

2.5 Manipulación de arrays

Los elementos dentro de un array (ya sea un vector o una matriz) pueden ser manipulados individualmente o en grupo. Esto es especialmente útil cuando se necesita redefinir sólo algunos de los elemen-

tos para ser utilizados en cálculos específicos, o cuando un subgrupo de elementos se utiliza para definir una nueva variable.

2.5.1 *Vectores*

La dirección de un elemento en un vector es su posición en la fila (o columna). Si tenemos un vector llamado `ve`, `ve(k)` referencia al elemento en la posición *k*. La primera posición es 1. Por ejemplo, si el vector *ve* tiene nueve elementos:

$$ve = 35 \quad 46 \quad 78 \quad 23 \quad 5 \quad 14 \quad 81 \quad 3 \quad 55$$

entonces

$$ve(4) = 23,\ ve(7) = 81,\ \text{y}\ ve(1) = 35.$$

Un elemento de un vector *v(k)* se puede utilizar como una variable. Por ejemplo, es posible cambiar el valor de un solo elemento de un vector reasignando un nuevo valor a la dirección específica del elemento en cuestión. Esto se hace tecleando: *v(k) = valor*. Un elemento de un vector puede ser utilizado también como variable en expresiones matemáticas. Por ejemplo:

```
>> VCT = [35 46 78 23 5 14 81 3 55]
VCT =
   35   46   78   23    5   14   81    3   55
>> VCT(4)
ans =
 23
>> VCT(6) = 273
VCT =
   35   46   78   23    5  273   81    3   55
>> VCT(2) + VCT(8)
ans =
 49
>> VCT(5)^VCT(8) + sqrt(VCT(7))
ans =
 134
>>
```

Define un vector.

Visualiza el cuarto elemento.

Asigna un nuevo valor al sexto elemento.

Se visualiza el vector completo.

Utilización de los elementos del vector en operaciones matemáticas.

2.5.2 *Matrices*

La dirección de un elemento en una matriz es su posición definida a partir del número de fila y de columna dentro de la propia matriz. Si tenemos una matriz *ma*, el elemento *ma(k, p)* se refiere al que ocupa la fila *k* y la columna *p*.

Por ejemplo, si la matriz es: $ma = \begin{bmatrix} 3 & 11 & 6 & 5 \\ 4 & 7 & 10 & 2 \\ 13 & 9 & 0 & 8 \end{bmatrix}$.

Entonces, $ma(1,1) = 3$ y $ma(2,3) = 10$.

Al igual que sucede con los vectores, es posible cambiar el valor de un solo elemento de la matriz asignándole un nuevo valor. Así mismo, los elementos se pueden utilizar individualmente como variables en expresiones matemáticas y funciones. He aquí algunos ejemplos:

```
>> MAT = [3 11 6 5; 4 7 10 2; 13 9 0 8]

MAT =
    3   11    6    5
    4    7   10    2
   13    9    0    8

>> MAT(3,1) = 20

MAT =
    3   11    6    5
    4    7   10    2
   20    9    0    8

>> MAT(2,4) - MAT(1,2)

ans =
   -9

>>
```

Crea una matriz 3 × 4.

Asigna un nuevo valor al elemento (3,1).

Utilización de los elementos de la matriz en expresiones matemáticas.

2.6 Utilización de los dos puntos (:) en la manipulación de arrays

Los dos puntos se utilizan para acceder a un rango de elementos dentro de un vector o una matriz.

Para vectores:

va(:) se refiere a todos los elementos del vector *va* (ya sea un vector fila o columna).

va(*m*:*n*) se refiere a todos los elementos comprendidos entre las posiciones *m* y *n* del vector *va*.
Ejemplo:

```
>> va = [4 15 8 12 34 2 50 23 11]

va =
    4   15    8   12   34    2   50   23   11

>> u = va(3:7)

u =
    8   12   34    2   50

>>
```

Se crea el vector `va`.

Se crea un vector u a partir de los elementos de las posiciones 3 a 7 del vector `va`.

Para matrices:

$A(:,n)$ se refiere a los elementos de la columna n de la matriz A.

$A(n,:)$ se refiere a los elementos de la fila n de la matriz A.

$A(:,m:n)$ se refiere a los elementos entre las columnas m y n de la matriz A.

$A(m:n,:)$ se refiere a los elementos entre las filas m y n de la matriz A.

$A(m:n,p:q)$ se refiere a los elementos de la fila m a la n, y a los de la columna p a la q de la matriz A.

En el Tutorial 2.3 se muestra la utilización del símbolo dos puntos en la manipulación de los elementos de un array.

Tutorial 2.3: Utilización de los dos puntos en la manipulación de arrays.

```
>> A = [1 3 5 7 9 11; 2 4 6 8 10 12; 3 6 9 12 15 18; 4 8 12 16 20 24; 5 10 15 20 25 30]
A =
   1   3   5   7   9  11
   2   4   6   8  10  12
   3   6   9  12  15  18
   4   8  12  16  20  24
   5  10  15  20  25  30
>> B = A(:,3)
B =
   5
   6
   9
  12
  15
>> C = A(2,:)
C =
   2   4   6   8  10  12
>> E = A(2:4,:)
E =
   2   4   6   8  10  12
   3   6   9  12  15  18
   4   8  12  16  20  24
>> F = A(1:3,2:4)
F =
   3   5   7
   4   6   8
   6   9  12
>>
```

Se define una matriz A con 5 filas y 6 columnas.

Se define un vector columna B a partir de los elementos de la columna 3 de la matriz A.

Se define un vector fila C a partir de los elementos de la fila 2 de la matriz A.

Se define una matriz B a partir de los elementos en las filas 2 a 4 de la matriz A.

Se crea una matriz F a partir de los elementos de las filas 1 a 3, columnas 2 a 4 de la matriz A.

En el Tutorial 2.3 se ve cómo pueden crearse nuevas matrices y vectores a partir de otras existentes, utilizando rangos de elementos o rangos de filas y columnas (usando :). Sin embargo, también es posible crear nuevas variables seleccionando solamente elementos específicos, o filas y columnas específicas, de variables creadas previamente. La forma de hacer esto es teclear los elementos seleccionados (así como las filas o columnas) dentro de corchetes, de la forma siguiente:

```
>> v = 4:3:34

v =
   4   7   10   13   16   19   22   25   28   31   34

>> u = v([3, 5, 7:10])

u =
   10   16   22   25   28   31

>> A = [10:-1:4; ones(1,7); 2:2:14; zeros(1,7)]

A =
   10   9   8   7   6   5   4
    1   1   1   1   1   1   1
    2   4   6   8   10  12  14
    0   0   0   0   0   0   0

>> B = A([1,3],[1,3,5:7])

B =
   10   8   6   5   4
    2   6   10  12  14
```

Crea un vector con 11 elementos.

Crea un vector u a partir del tercero, quinto y del séptimo al décimo elemento de v.

Crea la matriz A de 4 × 7.

Crea una matriz B a partir de las filas primera y tercera, así como de la primera, tercera y quita a séptima columnas de A.

2.7 Adición de nuevos elementos a variables ya creadas

Una variable que ya ha sido creada previamente puede alterar su estructura inicial mediante la inserción de nuevos elementos (como ya comentamos, un escalar es un vector de un solo elemento). Un vector (una matriz con una sola fila o columna) puede cambiar de tamaño para contener más elementos de los que ya tiene; e incluso puede convertirse en una matriz bidimensional. A las matrices también se les puede añadir filas y columnas para obtener un array de tamaño distinto del inicial. Para añadir elementos a un array simplemente hay que asignar nuevos elementos a los ya existentes. También se pueden añadir variables previamente creadas.

Adición de elementos a un vector:

Se pueden añadir elementos nuevos a un vector existente mediante una asignación de estos elementos. Por ejemplo, si un vector tiene 4 elementos, éste puede crecer de tamaño asignando nuevos números a las posiciones 5, 6, etc. Si un vector tiene, en general, n elementos y se le asigna un nuevo valor en la posición $n+2$ o mayor, MATLAB asigna ceros a los elementos que hay entre el último elemento del vector original y el nuevo elemento añadido. Por ejemplo:

```
>> DF = 1:4

DF =
   1   2   3   4

>> DF(5:10) = 10:5:35
```

Define un vector DF con 4 elementos.

Añade 6 elementos, empezando en la quinta posición.

```
DF =
   1   2   3   4   10   15   20   25   30   35

>> AD = [5 7 2]                          Define un vector AD con 3 elementos.

AD =
   5   7   2

>> AD(8) = 4                             Asigna un nuevo valor al octavo elemento.

AD =
   5   7   2   0   0   0   0   4         MATLAB asigna ceros como elementos
                                         en las posiciones cuarta a séptima.
>> AR(5) = 24                            Asigna un nuevo valor al quinto elemento de un nuevo vector.

AR =
   0   0   0   0   24                    MATLAB asigna ceros como elementos en las posiciones primera a cuarta.

>>
```

También es posible insertar nuevos elementos a un vector añadiéndole un vector existente. He aquí dos ejemplos:

```
>> RE = [3 8 1 24];                      Define un vector RE con 4 elementos.

>> GT = 4:3:16;                          Define un vector GT con 5 elementos.

>> KNH = [RE GT]                         Define un nuevo vector KNH uniendo
                                         los elementos de RE y GT.
KNH =
   3   8   1   24   4   7   10   13   16

>> KNV = [RE'; GT']

KNV =                                    Crea un nuevo vector columna KNV
   3                                     uniendo los elementos de RE' y GT'.
   8
   1
   24
   4
   7
   10
   13
   16
```

Adición de elementos a una matriz:

A una matriz existente se le pueden añadir nuevas filas y columnas asignándole valores a las nuevas filas o columnas dentro de la matriz. Nuevamente, esto se puede realizar asignando nuevos valores o añadiendo una variable predefinida. En cualquier caso debe tenerse especial cuidado, ya que el tamaño de las columnas y filas añadidas debe coincidir con el de la matriz original. Ejemplos:

```
>> E = [1 2 3 4; 5 6 7 8]
```

Define una matriz E de 2 × 2.

```
E =
   1   2   3   4
   5   6   7   8

>> E(3,:) = [10:4:22]
```

Añade el vector 10 14 18 22 en la tercera fila de E.

```
E =
    1    2    3    4
    5    6    7    8
   10   14   18   22

>> K = eye(3)
```

Define una matriz K de 3 × 3.

```
K =
   1   0   0
   0   1   0
   0   0   1

>> G = [E K]
```

Crea una nueva matriz G resultado de añadir la matriz K y la matriz E. El número de filas de E y K debe coincidir.

```
G =
    1    2    3    4    1    0    0
    5    6    7    8    0    1    0
   10   14   18   22    0    0    1
```

Si una matriz tiene un tamaño de $m \times n$, y se le asigna un nuevo elemento a una dirección más allá del tamaño de la matriz, MATLAB incrementará el tamaño de la matriz para incluir el nuevo elemento. Así mismo se añadirían ceros en los huecos. Ejemplos:

```
>> AW = [3 6 9; 8 5 11]
```

Define una matriz AW de 2 × 3.

```
AW =
   3   6   9
   8   5   11

>> AW(4,5) = 17
```

Asigna un nuevo valor al elemento (4,5).

```
AW =
   3   6   9   0   0
   8   5   11  0   0
   0   0   0   0   0
   0   0   0   0   17

>> BG(3,4) = 15
```

MATLAB cambia el tamaño original de la matriz a 4 × 5, asignando ceros a los elementos intermedios creados.

Asigna un nuevo valor al elemento (3,4) de una nueva matriz BG.

```
BG =
   0   0   0   0
   0   0   0   0
   0   0   0   15

>>
```

MATLAB crea una matriz de 3 × 4 y asigna ceros a todos los elementos excepto a BG(3,4).

2.8 Eliminación de elementos

Un elemento, o un rango de elementos, de una variable existente puede ser eliminado simplemente reasignando el conjunto vacío [] (corchetes sin elementos en su interior) a dichos elementos. Esta operación puede llevarse a cabo tanto en matrices como en vectores. Ejemplos:

```
>> kt = [2 8 40 65 3 55 23 15 75 80]
kt =
   2   8   40   65   3   55   23   15   75   80
>> kt(6) = []
kt =
   2   8   40   65   3   23   15   75   80
>> kt(3:6) = []
kt =
   2   8   15   75   80
>> mtr = [5 78 4 24 9; 4 0 36 60 12; 56 13 5 89 3]
mtr =
    5   78    4   24    9
    4    0   36   60   12
   56   13    5   89    3
>> mtr(:,2:4) = []
mtr =
    5    9
    4   12
   56    3
>>
```

Define un vector `kt` de 10 elementos.

Elimina el sexto elemento.

El vector tiene ahora 9 elementos.

Elimina los elementos de las posiciones 3 a la 6.

El vector tiene ahora 5 elementos.

Define una matriz `mtr` de 3 × 5.

Elimina todas las columnas de la 2 a la 4.

2.9 Funciones para la manipulación de arrays

MATLAB posee una amplia variedad de funciones para la manipulación de arrays. Algunas de estas funciones se presentan en la Tabla 2.2.

Existen muchas más funciones, de las cuales se puede obtener información detallada utilizando la Ventana de Ayuda de MATLAB. Para ello sólo hay que seleccionar, en la Ventana de Ayuda, "Functions by Category" (Funciones por Categoría), seguidamente "Mathematics" (Matemáticas) y finalmente "Arrays and Matrices" (Arrays y Matrices).

Como vimos en el Capítulo 1, el comando `who` visualizaba el listado de variables actuales que residen en memoria. Ampliando la funcionalidad del anterior, el comando `whos` además

Tabla 2.2: Funciones MATLAB para la manipulación de arrays.

Función	Descripción	Ejemplo
`length(A)`	Devuelve el número de elementos de `A`.	>> A = [5 9 2 4]; >> length(A) ans = 4
`size(A)`	Devuelve un vector fila [m,n], donde `m` y `n` representan el tamaño $m \times n$ del array `A`.	>> A = [6 1 4 0 12; 5 19 6 8 2] A = 6 1 4 0 12 5 19 6 8 2 >> size(A) ans = 2 5
`reshape(A, m, n)`	Reordena una matriz `A`, que tiene `r` filas y `s` columnas, a una matriz de `m` filas y `n` columnas. El valor de `r` por `s` debe ser igual al de `m` por `n`.	>> A = [5 1 6; 8 0 2] A = 5 1 6 8 0 2 >> B = reshape(A,3,2) B = 5 0 8 6 1 2
`diag(v)`	Cuando `v` es un vector, este comando crea una matriz cuadrada con los elementos de `v` en la diagonal.	>> v = [7 4 2]; >> A = diag(v) A = 7 0 0 0 4 0 0 0 2
`diag(A)`	Cuando `A` es una matriz, este comando crea un vector a partir de los elementos de la diagonal de `A`.	>> A = [1 2 3; 4 5 6; 7 8 9] A = 1 2 3 4 5 6 7 8 9 >> vec = diag(A) vec = 1 5 9

visualizaba el tamaño, bytes y clase de las variables actuales en la memoria. Veamos un ejemplo con arrays:

```
>> a = 7;
>> E = 3;
>> d = [5, a+E, 4, E^2]
d =
   5   10   4   9
>> g = [a, a^2, 13; a*E, 1, a^E]
g =
    7   49   13
   21    1  343
>> who
Your variables are:
  E a d g
>> whos
Name Size Bytes     Class
  E   1x1     8    double array
  a   1x1     8    double array
  d   1x4    32    double array
  g   2x3    48    double array
Grand total is 12 elements using 96 bytes
>>
```

Creación de las variables `a`, `E`, `d` y `g`.

El comando `who` visualiza las variables que se encuentran actualmente en memoria.

El comando `whos` visualiza las variables que se encuentran actualmente en memoria, junto con la información sobre sus tamaños.

Problema de ejemplo 2.1: Creación de una matriz

Utilizando los comandos `ones` y `zeros`, crear una matriz de 4 × 5 en la cual las primeras dos filas sean ceros y las dos siguientes sean unos.

Solución

```
>> A(1:2,:) = zeros(2,5)
A =
   0   0   0   0   0
   0   0   0   0   0
>> A(3:4,:) = ones(2,5)
A =
   0   0   0   0   0
   0   0   0   0   0
   1   1   1   1   1
   1   1   1   1   1
>>
```

Primero se crea una matriz de 2 × 5 con ceros.

Se añaden las filas 3 y 4 con unos.

Problema de ejemplo 2.2: Creación de una matriz

Crear una matriz de 6 × 6 en la cual las dos filas centrales, junto con las dos columnas centrales, sean unos, siendo el resto de elementos cero.

Solución

```
>> AR = zeros(6,6)
AR =
   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
```

Primero se crea una matriz de 6 × 6 con ceros.

```
>> AR(3:4,:) = ones(2,6)
AR =
   0   0   0   0   0   0
   0   0   0   0   0   0
   1   1   1   1   1   1
   1   1   1   1   1   1
   0   0   0   0   0   0
   0   0   0   0   0   0
```

Se reasigna el valor uno al conjunto de filas que va de la tercera a la cuarta.

```
>> AR(:,3:4) = ones(6,2)
AR =
   0   0   1   1   0   0
   0   0   1   1   0   0
   1   1   1   1   1   1
   1   1   1   1   1   1
   0   0   1   1   0   0
   0   0   1   1   0   0
```

Se reasigna el valor uno al conjunto de columnas que va de la tercera a la cuarta.

Problema de ejemplo 2.3: Manipulación de matrices

Sean dos matrices, una A de tamaño 5 × 6, y otra B de tamaño 3 × 5. Sea también un vector v de longitud 9:

$$A = \begin{bmatrix} 2 & 5 & 8 & 11 & 14 & 17 \\ 3 & 6 & 9 & 12 & 15 & 18 \\ 4 & 7 & 10 & 13 & 16 & 19 \\ 5 & 8 & 11 & 14 & 17 & 20 \\ 6 & 9 & 12 & 15 & 18 & 21 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 10 & 15 & 20 & 25 & 30 \\ 30 & 35 & 40 & 45 & 50 & 55 \\ 55 & 60 & 65 & 70 & 75 & 80 \end{bmatrix}$$

$$v = \begin{bmatrix} 99 & 98 & 97 & 96 & 95 & 94 & 93 & 92 & 91 \end{bmatrix}$$

Crear los tres arrays en la Ventana de Comandos y, seguidamente, escribir un solo comando que reemplace: 1) las cuatro últimas columnas de las filas 1 y la 3 de *A* por las cuatro primeras columnas de las primeras dos filas de la matriz *B*; 2) las cuatro últimas columnas de la cuarta fila de *A* por los elementos 5 a 8 de *v*; y 3) las cuatro últimas columnas de la quinta fila de *A* por las columnas 2 a 5 de la tercera fila de *B*.

Solución

```
>> A = [2:3:17; 3:3:18; 4:3:19; 5:3:20; 6:3:21]
A =
   2   5   8  11  14  17
   3   6   9  12  15  18
   4   7  10  13  16  19
   5   8  11  14  17  20
   6   9  12  15  18  21
>> B = [5:5:30; 30:5:55; 55:5:80]
B =
    5   10   15   20   25   30
   30   35   40   45   50   55
   55   60   65   70   75   80
>> v = [99:-1:91]
v =
   99   98   97   96   95   94   93   92   91
>> A([1 3 4 5],3:6) = [B([1 2],1:4); v(5:8); B(3,2:5)]
A =
   2   5   5  10  15  20
   3   6   9  12  15  18
   4   7  30  35  40  45
   5   8  95  94  93  92
   6   9  60  65  70  75
```

Matriz de 4 × 4 compuesta por las columnas 3 a 6 de las filas 1, 3, 4 y 5.

Matriz de 4 × 4. Las primeras dos filas son las columnas 1 a 4 de las filas 1 y 2 de la matriz B. La tercera fila son los elementos 5 a 8 del vector v. La fila cuarta son las columnas 2 a 5 de la fila 3 de la matriz B.

2.10 Cadenas de caracteres y variables de tipo *string*

- Una cadena o *string* es simplemente un array de caracteres. Para crear una cadena solo es necesario teclear los caracteres que la forman entre comillas simples.
- Las cadenas pueden incluir letras, dígitos, algunos símbolos y espacios. Sin embargo, y debido a la versión en inglés de MATLAB, las cadenas no admiten símbolos correspondientes al juego extendido de caracteres ASCII, como por ejemplo la letra eñe o los acentos. Deben por tanto evitarse estos símbolos a la hora de construir cadenas de caracteres. (A pesar de lo dicho, pueden existir algunas versiones de MATLAB preparadas para aceptar el juego de caracteres extendidos ASCII. En este libro no lo utilizaremos.)

- Son ejemplos de cadenas válidas: 'ad ef', '3%fr2', '{edcba:21!', 'MATLAB', 'annio', 'Jose' y 'camion'. Sin embargo, son ejemplos de cadenas inválidas: 'año', 'José' y 'camión'.
- Para introducir la comilla simple como carácter dentro de la cadena es necesario teclearla dos veces seguidas.
- Cuando comienza a teclearse una cadena, el color del texto en la pantalla cambiar a tonalidad púrpura, una vez que se teclea la primera comilla. Cuando se teclea la última comilla, al final de la cadena, el color de la cadena cambia a marrón.

Las cadenas de caracteres tienen diferentes usos en MATLAB. Normalmente se utilizan como salida para visualizar ciertos mensajes de texto (Capítulo 4), en comandos de formato para gráficos (Capítulo 5) y como argumentos de algunas funciones (Capítulo 6).

- Cuando las cadenas se utilizan en gráficos (etiquetas de ejes, título y notas de texto), los caracteres que forman la cadena pueden ser representados con una fuente, tamaño, posición, color, etc. específicos. El Capítulo 5 trata estos aspectos en detalle.

Las cadenas pueden ser igualmente asignadas a variables de forma similar a como hemos visto hasta ahora. He aquí algunos ejemplos:

```
>> a = 'FRty 8'
a =
   FRty 8
>> B = 'Mi nombre es Jose Antonio'
B =
   Mi nombre es Jose Antonio
>>
```

Cuando a una variable se le asigna una cadena, se almacena en memoria de la misma forma que sucede con los números. Cada carácter, incluido el espacio, se corresponde con un elemento en el array. Esto implica que una línea de caracteres es un vector fila en el cual el número de elementos se corresponde con el número de caracteres. A los elementos del vector se accede mediante sus posiciones. Por ejemplo, en el vector B que se definió antes, el cuarto elemento se corresponde con la letra n, el doceavo elemento con la J, y así sucesivamente.

```
>> B(4)
ans =
   n
>> B(14)
ans =
   J
>>
```

Al igual que con los vectores que contienen números, es posible también cambiar los elementos de una cadena mediante asignación directa por la posición de los elementos. Por ejemplo, en el vector B que se definió antes, el nombre Jose puede ser reemplazado por Juan de la forma:

```
>> B(14:17) = 'Juan'
B =
   Mi nombre es Juan Antonio
```

Utilización de los dos puntos para asignar nuevos caracteres a los elementos 14 a 17 del vector `B`.

Las cadenas también pueden almacenarse en una matriz. Al igual que sucede con los números, la forma de hacerlo es tecleando un punto y coma ; (o pulsando la tecla **Intro**) al final de cada fila. En este caso cada fila debe teclearse como una cadena, es decir, debe estar encerrada entre comillas. Además, como sucede con las matrices de números, el número de elementos en todas las filas debe ser el mismo. Este requisito puede causar algunos problemas cuando la intención del usuario es crear filas con palabras distintas. En este caso podrían utilizarse espacios para asegurarse que todas las filas tienen el mismo número de elementos.

MATLAB proporciona una función llamada `char` que crea un array con filas que tienen el mismo número de caracteres a partir de datos de entrada (en forma de filas) que no tienen por qué ser de la misma longitud. MATLAB hace que la longitud de todas las filas sea igual a la de mayor tamaño, a base de añadir espacios al final de las líneas más cortas. La función `char` admite como parámetros de entrada las cadenas separadas por coma, según el formato:

```
nombre_variable = char ('cadena 1', 'cadena 2', 'cadena 3', …, 'cadena N')
```

Por ejemplo:

```
>> Info = char('Nombre del Estudiante:','Luis Garcia','Curso:','Primero')
Info =
   Nombre del Estudiante:
   Luis Garcia
   Curso:
   Primero
>>
```

A la variable `Info` se le asignan cuatro filas de cadenas, cada una de diferente longitud.

La función `char` crea un array con cuatro filas de la misma longitud que la fila más grande, añadiendo espacios en blanco a las líneas más cortas.

Una variable se puede definir como un número o una cadena que tenga los mismos dígitos. Por ejemplo, como se muestra más abajo, `x` se define como el número 536, e `y` se define como una cadena que contiene los dígitos 536.

```
>> x = 536
x =
   536
>> y = '536'
y =
   536
>>
```

Estas dos variables son completamente distintas, aunque parecen idénticas en la pantalla. La variable `x` puede ser usada en operaciones matemáticas, mientras que la variable `y` no.

2.11 Problemas

1. Cree un vector fila que contenga los elementos: 32, 4, 81, $e^{2,5}$, $\cos(\pi/3)$ y 14,12.
2. Cree un vector columna que contenga los elementos: 55, 14, ln(51), 987, 0 y 5sen(2,5 π).
3. Cree un vector fila en el cual el primer elemento sea 1 y el último elemento sea 33, con una distancia de 2 entre los elementos (1, 3, 5, ..., 33).
4. Cree un vector columna en el cual el primer elemento sea 15, la distancia de los elementos sea –5, y donde el último elemento sea –25. (Un vector columna se puede crear a partir de la transposición de un vector fila.)
5. Cree un vector fila con 15 elementos igualmente distanciados, en el cual el primer elemento sea 7 y el último 40.
6. Cree un vector columna con 12 elementos igualmente distanciados, en el cual el primer elemento sea –1 y el último –15.
7. Cree un vector, llamado `Aprimero`, que tenga 16 elementos, siendo el primero el 4, con un incremento de 3 y siendo el último elemento el 49. Posteriormente utilice el símbolo dos puntos para crea un nuevo vector, llamado `Asegundo`, que tenga ocho elementos. Los primeros cuatro elementos serán los primeros cuatro elementos del vector `Aprimero`, y los cuatro últimos serán los cuatro últimos elementos del vector `Aprimero`.
8. Cree una matriz como la que se muestra más abajo utilizando la notación de vectores para crear vectores con distanciado constante, y/o el comando `linspace` para crear las filas.

$$B = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 & 16 & 19 & 22 & 25 \\ 72 & 66 & 60 & 54 & 48 & 42 & 36 & 30 & 24 \\ 0 & 0{,}125 & 0{,}250 & 0{,}375 & 0{,}500 & 0{,}625 & 0{,}750 & 0{,}875 & 1{,}000 \end{bmatrix}$$

9. Cree la siguiente matriz A: $A = \begin{bmatrix} 6 & 43 & 2 & 11 & 87 \\ 12 & 6 & 34 & 0 & 5 \\ 34 & 18 & 7 & 41 & 9 \end{bmatrix}$

   Utilice la matriz A para:

   a) Crear un vector fila de cinco elementos llamado `va`, que contenga los elementos de la segunda fila de A.

   b) Crear un vector fila de seis elementos llamado `vb`, que contenga los elementos de la cuarta columna de A.

   c) Crear un vector fila de diez elementos llamado `vc`, que contenga los elementos de la primera y segunda fila de A.

   d) Crear un vector fila de seis elementos llamado `vd` que contenga los elementos de la segunda a la quinta columna de A.

10. Cree la siguiente matriz C: $C = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 \\ 3 & 6 & 9 & 12 & 15 \\ 7 & 14 & 21 & 28 & 35 \end{bmatrix}$

Utilice la matriz C para:

a) Crear un vector columna de tres elementos llamado `ua`, que contenga los elementos de la tercera columna de C.

b) Crear un vector columna de cinco elementos llamado `ub`, que contenga los elementos de la segunda fila de C.

c) Crear un vector columna de nueve elementos llamado `uc`, que contenga los elementos de la primera, tercera y quinta columna de C.

d) Crear un vector columna de diez elementos llamado `ud`, que contenga los elementos de la primera y segunda fila de C.

11. Cree la siguiente matriz A: $A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 \\ 21 & 18 & 15 & 12 & 9 & 6 & 3 \\ 5 & 10 & 15 & 20 & 25 & 30 & 35 \end{bmatrix}$

a) Cree una matriz B de 3×4 a partir de la primera, tercera y cuarta fila, y de la primera, tercera, quinta y séptima columna de la matriz A.

b) Cree un vector fila de 15 elementos llamado u, a partir de los elementos de la tercera fila y de la quinta a la séptima columna de la matriz A.

12. Utilizando las funciones `zeros`, `ones` y `eye`, cree los siguientes arrays:

a) $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ *b*) $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ *c*) $\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$

13. Utilizando el comando `eye`, cree un array A como el que se muestra más abajo en la parte izquierda. A continuación, utilice los dos puntos para acceder a los elementos en el array, cambiando el array para que sea como el de la parte derecha.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad A = \begin{bmatrix} 2 & 2 & 2 & 0 & 5 & 5 & 5 \\ 2 & 2 & 2 & 0 & 5 & 5 & 5 \\ 3 & 3 & 3 & 0 & 5 & 5 & 5 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \\ 4 & 4 & 7 & 0 & 9 & 9 & 9 \end{bmatrix}$

14. Utilizando las funciones `zeros` y `ones`, cree una matriz 3 × 5 en la cual la primera, segunda y quinta columnas sean ceros, y la tercera y cuarta columnas sean unos.

15. Cree una matriz de 5 × 7 en la cual la primera fila contenga los números: 1 2 3 4 5 6 7, la segunda fila contenga: 8 9 10 11 12 13 14, la tercera fila contenga los números del 15 al 21, y así sucesivamente. A partir de esta matriz, cree otra nueva de 3 × 4 compuesta por las filas 2 a la 4 y las columnas de la 3 a la 6 de la primera matriz.

16. Cree una matriz A de 3 × 3 donde todos los elementos sean 1. Cree también una matriz B de 2 × 2 donde todos los elementos sean 5. A continuación, añada nuevos elementos a la matriz A a base de añadir la matriz B, de manera que A quede finalmente de la siguiente forma:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 0 & 0 & 5 & 5 \end{bmatrix}$$

Capítulo 3
Operaciones matemáticas con arrays

Las variables en MATLAB, una vez creadas, se pueden utilizar para realizar operaciones matemáticas de distinta índole. En el Capítulo 1 vimos cómo utilizar variables escalares en operaciones matemáticas. El hecho de ser escalar implica que estas variables se pueden considerar como arrays de dimensión 1×1 (arrays de una fila y una columna, es decir, con un solo elemento). Los arrays, no obstante, pueden ser de una dimensión (una fila o una columna), dos dimensiones (filas y columnas) o incluso tener dimensiones superiores. En estos casos las operaciones matemáticas son un tanto más complejas. MATLAB está especialmente diseñado para llevar acabo operaciones avanzadas con arrays, con vistas a aplicaciones prácticas en el campo de las ciencias. En este capítulo se presentan las operaciones matemáticas básicas con arrays que se pueden realizar utilizando MATLAB.

Para empezar se estudiarán las operaciones básicas de suma y resta de arrays en la Sección 3.1. En MATLAB, las operaciones de multiplicación, división y exponenciación se pueden llevar a cabo de dos formas diferentes. La primera utilizando los símbolos estándar (*, / y ^) y a partir de las reglas del álgebra lineal, se presenta en las Secciones 3.2 y 3.3. La segunda forma de operar se llama computación elemento a elemento y se explica en la Sección 3.4. En esta última se utilizan los símbolos .*, ./ y .^ (se teclea un punto delante del símbolo estándar). Además, en ambas formas de operar existe el operador división izquierda (.\ o \) que también se explica en las Secciones 3.3 y 3.4.

Nota para los usuarios que utilizan por primera vez MATLAB:

A pesar de que primero se presentarán las operaciones generales sobre matrices y luego las operaciones elemento a elemento, este orden puede invertirse sin problemas, ya que estos conceptos son independientes el uno del otro. Por otro lado, se espera que la mayoría de los lectores posean conocimientos sobre operaciones con matrices y sobre álgebra lineal, de forma que puedan seguir sin problemas los conceptos que se narran en las Secciones 3.2 y 3.3. Otros lectores, sin embargo, pueden preferir leer la Sección 3.4 primero.

3.1 Suma y resta

Las operaciones + (suma) y – (resta) se pueden utilizar con arrays de tamaños idénticos, es decir, aquellos que tienen el mismo número de filas y de columnas. La suma, así como la resta, de dos arrays se lleva a cabo sumando o restando sus elementos.

En general, si A y B son dos arrays (por ejemplo, matrices de 2 × 3),

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \quad \text{y} \quad B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \end{bmatrix}$$

La matriz que se obtiene de sumar A y B es:

$$\begin{bmatrix} (A_{11}+B_{11}) & (A_{12}+B_{12}) & (A_{13}+B_{13}) \\ (A_{21}+B_{21}) & (A_{22}+B_{22}) & (A_{23}+B_{23}) \end{bmatrix}$$

Cuando a un array se le añade o se le sustrae un escalar (un número), éste se añade o se sustrae de todos los elementos del array. He aquí algunos ejemplos:

```
>> VectA = [8 5 4]; VectB = [10 2 7];

>> VectC = VectA + VectB

VectC =
   18    7   11

>> A = [5 -3 8; 9 2 10]

A =
    5   -3    8
    9    2   10

>> B = [10 7 4; -11 15 1]

B =
   10    7    4
  -11   15    1

>> A - B

ans =
   -5  -10    4
   20  -13    9

>> C = A + B

C =
   15    4   12
   -2   17   11

>> C - 8
```

Se definen dos vectores.

Se define un vector `VectC` que es igual a la suma `VectA + VectB`.

Se definen dos matrices `A` y `B` de 2 × 3.

Se realiza la resta de matrices `A - B`.

Se define una matriz `C` que es igual al resultado de la suma `A + B`.

Se resta el número 8 de la matriz `C`.

```
ans =
     7    -4     4
   -10     9     3

>>
```

3.2 Multiplicación de arrays

La operación de multiplicación * es ejecutada por MATLAB según las reglas propias del álgebra lineal. Esto significa que si A y B son dos matrices, la operación $A*B$ se ejecuta solamente si el número de columnas de la matriz A es igual al número de filas de la matriz B. El resultado es una matriz que tiene el mismo número de filas que A y el mismo número de columnas que B. Por ejemplo, si A es una matriz de 4×3 y B es una matriz de 3×2:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \quad \text{y} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{41} \end{bmatrix}$$

entonces, la matriz que se obtiene a partir de la operación $A*B$ tiene dimensión 4×2, y sus elementos son:

$$\begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}) & (A_{11}B_{12} + A_{22}B_{22} + A_{13}B_{32}) \\ (A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}) & (A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32}) \\ (A_{31}B_{11} + A_{32}B_{21} + A_{33}B_{31}) & (A_{31}B_{12} + A_{32}B_{22} + A_{33}B_{32}) \\ (A_{41}B_{11} + A_{42}B_{21} + A_{43}B_{31}) & (A_{41}B_{12} + A_{42}B_{22} + A_{43}B_{32}) \end{bmatrix}$$

Veamos a continuación un ejemplo numérico de lo explicado anteriormente:

$$\begin{bmatrix} 1 & 4 & 3 \\ 2 & 6 & 1 \\ 5 & 2 & 8 \end{bmatrix} \begin{bmatrix} 5 & 4 \\ 1 & 3 \\ 2 & 5 \end{bmatrix} = \begin{bmatrix} (1 \cdot 5 + 4 \cdot 1 + 3 \cdot 2) & (1 \cdot 4 + 4 \cdot 3 + 3 \cdot 6) \\ (2 \cdot 5 + 6 \cdot 1 + 1 \cdot 2) & (2 \cdot 4 + 6 \cdot 3 + 1 \cdot 6) \\ (5 \cdot 5 + 2 \cdot 1 + 8 \cdot 2) & (5 \cdot 4 + 2 \cdot 3 + 8 \cdot 6) \end{bmatrix} = \begin{bmatrix} 15 & 34 \\ 18 & 32 \\ 43 & 74 \end{bmatrix}$$

El producto de la multiplicación de dos matrices cuadradas (deben ser del mismo tamaño) es también una matriz cuadrada del mismo tamaño. Sin embargo, la multiplicación de matrices no es conmutativa. Esto significa que si A y B son de dimensión $n \times n$, entonces $A*B \neq B*A$. Además, operaciones como la potencia sólo se pueden ejecutar con matrices cuadradas (ya que $A*A$ sólo puede resolverse si el número de columnas de la primera matriz es igual al número de filas de la segunda matriz).

Dos vectores se pueden multiplicar sólo si ambos tienen el mismo número de elementos. Es necesario además que uno sea un vector fila y el otro un vector columna. La multiplicación de un vector fila por tantos elementos como contenga un vector columna da como resultado una matriz de 1×1, es decir, un escalar. Esto es lo que se denomina producto escalar de dos vectores. MATLAB posee la fun-

ción `dot(a,b)`, que calcula el producto escalar de dos vectores. Cuando se utiliza la función `dot`, los vectores `a` y `b` pueden ser vectores fila o columna (ver Tabla 3.1). Sin embargo, la multiplicación de un vector columna tantas veces como elementos tenga otro vector fila, si ambos tienen n elementos, da como resultado una matriz de dimensión $n \times n$.

Tutorial 3.1: Multiplicación de arrays.

```
>> A = [1 4 2; 5 7 3; 9 1 6; 4 2 8]
A =
   1   4   2
   5   7   3
   9   1   6
   4   2   8
>> B = [6 1; 2 5; 7 3]
B =
   6   1
   2   5
   7   3
>> C = A*B
C =
  28   27
  65   49
  98   32
  84   38
>> D = B*A
??? Error using ==> *
Inner matrix dimensions must agree.
>> F = [1 3; 5 7]
F =
   1   3
   5   7
>> G = [4 2; 1 6]
G =
   4   2
   1   6
>> F*G
ans =
   7   20
  27   52
>> G*F
```

Se define una matriz `A` de 4 × 4.

Se define una matriz `B` de 3 × 2.

Se multiplica la matriz `A` por la matriz `B`, y se asigna el resultado a la variable `C`.

Se intenta multiplicar la matriz `B` por la matriz `A`: `B*A`. La operación retorna un error, ya que el número de columnas de `B` es 2, y el número de filas de `A` es 4.

Se definen dos matrices, `F` y `G`, de 2 × 2.

Se multiplica `F*G`.

Se multiplica `G*F`.

Tutorial 3.1 (continuación): Multiplicación de arrays.

```
ans =
   14   26
   31   45

>> AV = [2 5 1]

AV =
    2    5    1

>> BV = [3; 1; 4]

BV =
    3
    1
    4

>> AV*BV

ans =
   15

>> BV*AV

ans =
    6   15    3
    2    5    1
    8   20    4

>>
```

Como puede comprobarse, el resultado de multiplicar G*F no es el mismo que el de multiplicar F*G.

Se define un vector fila AV de tres elementos.

Se define un vector columna BV de tres elementos.

Se multiplica AV por BV. El resultado es un escalar (producto escalar de dos vectores).

Se multiplica BV por AV. El resultado es una matriz de 3 × 3.

Cuando se multiplica un array por un número (de hecho, un número es un array de 1 × 1), cada elemento del array es multiplicado por dicho número.

Por ejemplo:

```
>> A = [2 5 7 0; 10 1 3 4; 6 2 11 5]

A =
    2    5    7    0
   10    1    3    4
    6    2   11    5

>> b = 3

b =
    3

>> b*A

ans =
    6   15   21    0
   30    3    9   12
   18    6   33   15
```

Se define una matriz A de 3 × 3.

Se asigna el número 3 a la variable b.

Se multiplica la matriz A por b. Esto se puede hacer tecleando b*A o bien A*b.

```
>> C = A*5
C =
   10   25   35    0
   50    5   15   20
   30   10   55   25
>>
```

Se multiplica la matriz A por 5 y se asigna el resultado a una nueva variable C.
(Tecleando C = 5*A se produce un resultado equivalente.)

Las reglas de multiplicación de matrices basadas en el álgebra lineal permiten crear y resolver sistemas de ecuaciones lineales. Por ejemplo, el siguiente sistema de tres ecuaciones con tres incógnitas:

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + A_{13}x_3 &= B_1 \\ A_{21}x_1 + A_{22}x_2 + A_{23}x_3 &= B_2 \\ A_{31}x_1 + A_{32}x_2 + A_{33}x_3 &= B_3 \end{aligned}$$

se puede escribir con matrices de la siguiente forma:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

y en notación matricial:

$$AX = B \quad \text{donde } A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \text{y} \quad B = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}.$$

3.3 División de arrays

La operación de división también está relacionada con las reglas del álgebra lineal. Esta operación es más compleja que las anteriores y por tanto tan solo se dará una breve explicación sobre ella. En cualquier libro sobre álgebra lineal pueden encontrarse más detalles.

La operación de división se puede explicar con la ayuda de otros dos conceptos relacionados con las operaciones de matrices: la matriz identidad y la operación de inversión o matriz inversa.

Matriz identidad:

La matriz identidad es una matriz cuadrada en donde los elementos de la diagonal son unos y el resto de los elementos son ceros. Como se vio en la Sección 2.2.1, la matriz identidad en MATLAB se puede crear con el comando `eye`. La matriz identidad multiplicada por otra matriz (o vector) da como resultado la misma matriz original (la multiplicación debe ser efectuada, no obstante, a partir de las ya

conocidas reglas del álgebra lineal). En realidad, el resultado es equivalente a multiplicar por el escalar 1. Por ejemplo:

$$\begin{bmatrix}7 & 3 & 8\\ 4 & 11 & 5\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}7 & 3 & 8\\ 4 & 11 & 5\end{bmatrix} \quad \text{o} \quad \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}8\\ 2\\ 15\end{bmatrix} = \begin{bmatrix}8\\ 2\\ 15\end{bmatrix} \quad \text{o} \quad \begin{bmatrix}6 & 2 & 9\\ 1 & 8 & 3\\ 7 & 4 & 5\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}6 & 2 & 9\\ 1 & 8 & 3\\ 7 & 4 & 5\end{bmatrix}$$

Si una matriz A es cuadrada, ésta puede ser multiplicada por la matriz identidad I de derecha a izquierda o de izquierda a derecha:

$$AI = IA = A$$

Inversa de una matriz:

Se dice que una matriz B es la inversa de una matriz A si al multiplicar ambas matrices el producto es la matriz identidad. Ambas matrices deben ser cuadradas, y el orden de la multiplicación puede ser AB o BA.

$$BA = AB = I$$

Obviamente B es la inversa de A, y A es la inversa de B. Ejemplo:

$$\begin{bmatrix}2 & 1 & 4\\ 4 & 1 & 8\\ 2 & -1 & 3\end{bmatrix}\begin{bmatrix}5{,}5 & -3{,}5 & 2\\ 2 & -1 & 0\\ -3 & 2 & 1\end{bmatrix} = \begin{bmatrix}5{,}5 & -3{,}5 & 2\\ 2 & -1 & 0\\ -3 & 2 & 1\end{bmatrix}\begin{bmatrix}2 & 1 & 4\\ 4 & 1 & 8\\ 2 & -1 & 3\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

La inversa de una matriz A se denota comúnmente por A^{-1}. En MATLAB, el inverso de una matriz se puede obtener o bien elevando A a -1, A^{-1}, o bien con la función `inv(A)`. Veamos un ejemplo de todo esto a continuación:

```
>> A = [2 1 4; 4 1 8; 2 -1 3]

A =
   2   1   4
   4   1   8
   2  -1   3

>> B = inv(A)

B =
    5.5000   -3.5000    2.0000
    2.0000   -1.0000         0
   -3.0000    2.0000   -1.0000

>> A*B
```

Se crea la matriz A.

Se utiliza la función inv para calcular la inversa de A. El resultado se asigna a B.

El resultado de multiplicar A por B nos da la matriz identidad.

```
ans =
   1   0   0
   0   1   0
   0   0   1

>> A*A^-1

ans =
   1   0   0
   0   1   0
   0   0   1
```

Se calcula el inverso de A elevando esta matriz a –1. Además se multiplica de nuevo por A, lo que nos da finalmente la matriz identidad.

- No todas las matrices tienen inversa. Una matriz tiene inversa sólo si es cuadrada y su determinante no es cero.

Determinantes:

El determinante puede verse como una función asociada a las matrices cuadradas. A continuación se verá una breve explicación sobre cálculo de determinantes. Para obtener más detalles se recomienda al lector acudir a cualquier bibliografía sobre álgebra lineal.

Un determinante es una función que asocia un número, llamado el determinante de la matriz, a cada matriz cuadrada A. El determinante se denota comúnmente por det(A) o $|A|$. El determinante se calcula a partir de una regla específica. Por ejemplo, para una matriz de segundo orden con dimensión 2×2, la regla para el cálculo de su determinante es como sigue:

$$A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}, \text{ por ejemplo, } \begin{vmatrix} 6 & 5 \\ 3 & 9 \end{vmatrix} = 6 \cdot 9 - 5 \cdot 3 = 39$$

El determinante de una matriz cuadrada se puede calcular con el comando `det` (ver Tabla 3.1).

División de arrays:

MATLAB posee dos tipos de divisiones de arrays: la división derecha y la división izquierda.

División izquierda \:

La división izquierda se utiliza para resolver ecuaciones matriciales $AX = B$. En esta ecuación X y B son vectores columna. La ecuación en sí puede ser resuelta multiplicando en la parte izquierda de ambos miembros de la igualdad por el inverso de A:

$$A^{-1}AX = A^{-1}B$$

El primer miembro de la ecuación es X, ya que:

$$A^{-1}AX = IX = X$$

Por lo tanto, la solución a $AX = B$ es:

$$X = A^{-1}B$$

En MATLAB, esta última ecuación se puede escribir utilizando el carácter de la división izquierda:

$$X = A\backslash B$$

Aunque las dos últimas operaciones aparentan proporcionar resultados similares, hay que destacar que el método utilizado por MATLAB para calcular X es diferente. En el primero, MATLAB calcula A^{-1} y después utiliza este valor para multiplicarlo por B. En el segundo (división izquierda), la solución X se obtiene numéricamente a partir de un método basado en la eliminación gaussiana. El método de la división izquierda de MATLAB está especialmente indicado para resolver conjuntos de ecuaciones lineales, ya que el cálculo manual de la inversa puede ser menos preciso que el método de la eliminación gaussiana cuando se emplean matrices de gran tamaño.

División derecha /:

La división derecha se utiliza para resolver ecuaciones matriciales $XC = D$. En esta ecuación X y D son vectores fila. La ecuación anterior se puede resolver multiplicando la parte derecha de ambos miembros de la igualdad por la inversa de C:

$$XC\,C^{-1} = D\,C^{-1}$$

que resulta en:

$$X = D\,C^{-1}$$

En MATLAB, esta última ecuación se puede escribir utilizando el carácter de división derecha:

$$X = D/C$$

El siguiente ejemplo muestra el uso de las divisiones izquierda y derecha, así como de la función `inv`, para resolver ecuaciones lineales.

Problema de ejemplo 3.1: Resolución de tres ecuaciones lineales (división de arrays)

Utilizar los operadores de array que se estimen oportunos para resolver el siguiente sistema de ecuaciones lineales.

$$\begin{aligned} 4x - 2y + 6z &= 8 \\ 2x + 8y + 2z &= 4 \\ 6x + 10y + 3z &= 0 \end{aligned}$$

Solución

Utilizando las ya conocidas reglas del álgebra lineal, el sistema de ecuaciones anterior se puede representar de forma matricial: $AX = B$, o también de la forma $XC = D$:

$$\begin{bmatrix} 4 & -2 & 6 \\ 2 & 8 & 2 \\ 6 & 10 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix} \quad \text{o} \quad \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 4 & -2 & 6 \\ 2 & 8 & 2 \\ 6 & 10 & 3 \end{bmatrix}^{T} = \begin{bmatrix} 8 & 4 & 0 \end{bmatrix}$$

La solución a ambas formas es como se muestra a continuación:

```
>> A = [4 -2 6; 2 8 2; 6 10 3];
```
Composición de la forma AX = B.

```
>> B = [8; 4; 0];
>> X = A\B
```
Resolución de X = A\B, mediante la división izquierda.

```
X =
   -1.8049
    0.2927
    2.6341
>> Xb = inv(A)*B
```
Resolución de $X = A^{-1}B$, utilizando la inversa de A.

```
Xb =
   -1.8049
    0.2927
    2.6341
>> C = [4 2 6; -2 8 10; 6 2 3];
```
Composición de la forma XC = D.

```
>> D = [8 4 0];
>> Xc = D/C
```
Resolución de X = D/C, mediante la división derecha.

```
Xc =
   -1.8049   0.2927   2.6341
>> Xd = d*inv(C)
```
Resolución de $X = DC^{-1}$, utilizando la inversa de C.

```
Xd =
   -1.8049   0.2927   2.6341
```

3.4 Operaciones elemento a elemento

En las Secciones 3.2 y 3.3 vimos que cuando se utilizan los caracteres ordinarios para multiplicación y división con arrays (* y /) el resultado obedece a las reglas del álgebra lineal ya conocidas y explicadas. Sin embargo, hay veces en las que se requiere llevar a cabo operaciones elemento a elemento. Algunas operaciones ya vistas, como la suma y la resta de arrays, son por definición operaciones elemento a elemento, ya que cuando dos arrays se suman o o se restan, la operación se ejecuta con los elementos que ocupan la misma posición en los arrays. Las operaciones elemento a elemento se pueden realizar únicamente si los arrays tienen el mismo tamaño.

Para que las operaciones de multiplicación, exponenciación y división de arrays se realicen elemento a elemento, en MATLAB hay que teclear un punto delante del operador aritmético correspondiente.

Símbolo	Descripción	Símbolo	Descripción
.*	Multiplicación	./	División derecha
.^	Exponenciación	.\	División izquierda

Si dos vectores a y b contienen los elementos: $a = [a_1\ a_2\ a_3\ a_4]$ y $a = [b_1\ b_2\ b_3\ b_4]$, la multiplicación, división y exponenciación elemento a elemento de los dos vectores resultaría:

$$a\ .*\ b = [a_1b_1\ a_2b_2\ a_3b_3\ a_4b_4]$$
$$a\ ./\ b = [a_1/b_1\ a_2/b_2\ a_3/b_3\ a_4/b_4]$$
$$a\ .\hat{}\ b = [(a_1)^{b_1}\ (a_2)^{b_2}\ (a_3)^{b_3}\ (a_4)^{b_4}]$$

Si dos matrices A y B contienen los elementos:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \quad \text{y} \quad B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}.$$

la multiplicación y división elemento a elemento de las dos matrices resultaría:

$$A\ .*\ B = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & A_{13}B_{13} \\ A_{21}B_{21} & A_{22}B_{22} & A_{23}B_{23} \\ A_{31}B_{31} & A_{32}B_{32} & A_{33}B_{33} \end{bmatrix} \qquad A\ ./\ B = \begin{bmatrix} A_{11}/B_{11} & A_{12}/B_{12} & A_{13}/B_{13} \\ A_{21}/B_{21} & A_{22}/B_{22} & A_{23}/B_{23} \\ A_{31}/B_{31} & A_{32}/B_{32} & A_{33}/B_{33} \end{bmatrix}$$

la operación de exponenciación de la matriz A resultaría:

$$A\ .\hat{}\ n = \begin{bmatrix} (A_{11})^n & (A_{12})^n & (A_{13})^n \\ (A_{21})^n & (A_{22})^n & (A_{23})^n \\ (A_{31})^n & (A_{32})^n & (A_{33})^n \end{bmatrix}$$

En el Tutorial 3.2 se muestran distintos ejemplos de estas operaciones.

Tutorial 3.2: Operaciones elemento a elemento.

```
>> A = [2 6 3; 5 8 4]                    Define un array A de 2 × 3.
A =
   2   6   3
   5   8   4
>> B = [1 4 10; 3 2 7]                   Define un array B de 2 × 3.
B =
   1   4   10
   3   2    7
```

Tutorial 3.2 (continuación): Operaciones elemento a elemento.

```
>> A .* B
ans =
    2   24   30
   15   16   28
>> C = A ./ B
C =
   2.0000   1.5000   0.3000
   1.6667   4.0000   0.5714
>> B .^3
ans =
    1   64   1000
   27    8    343
>> A * B
??? Error using ==> *
  Inner matrix dimensions must agree.
>>
```

Multiplicación elemento a elemento de A por B.

División elemento a elemento de A entre B. El resultado se asigna a la variable C.

Exponenciación elemento a elemento de B. El resultado es un array en el que cada elemento es el correspondiente de B elevado a 3.

Si se intenta multiplicar A por B, el sistema da un error. A y B no pueden ser multiplicadas de acuerdo con las reglas del álgebra lineal, ya que el número de columnas de A no se corresponde con el número de filas de B.

Las operaciones elemento a elemento son muy útiles para calcular el valor de una función pasándole distintos valores de sus argumentos. Para hacer esto sólo es necesario definir un vector que contenga las variables independientes, y después usar el vector en operaciones elemento a elemento para crear un vector en el cual cada elemento se corresponda con el valor de la función en cuestión. Por ejemplo:

```
>> x = [1:8]
x =
   1   2   3   4   5   6   7   8
>> y = x.^2 - 4*x
y =
   -3   -4   -3   0   5   12   21   32
>>
```

Se crea un vector con ocho elementos.

El vector x se utiliza en operaciones elemento a elemento para obtener el vector y (resultado del cálculo sobre los elementos de x).

En el ejemplo anterior lo que se está calculando realmente es la función $y = x^2 - 4x$. Como x está elevado al cuadrado, se necesita operar elemento a elemento. Cada elemento del vector y es el resultado de operar sobre el correspondiente elemento de x sustituido en la ecuación anterior. Otro ejemplo es el siguiente:

```
>> z = [1:2:15]
z =
   1   3   5   7   9   11   13   15
>> y = (z .^ 3 + 5*z)./(4*z .^2 - 10)
y =
  -1.0000   1.64154   1.6667   2.0323   2.4650   2.9241   3.3964   3.8764
>>
```

Se crea un vector con ocho elementos.

El vector z se utiliza en operaciones elemento a elemento para obtener el vector y (resultado del cálculo sobre los elementos de z).

En este caso, la función con la que se trabaja es $y = \dfrac{z^3 + 5z}{4z^2 - 10}$. Las operaciones elemento a elemento se han utilizado en este último ejemplo tres veces: para calcular z^3 y z^2, y también para dividir el numerador por el denominador de la función.

3.5 Utilización de arrays en funciones predefinidas de MATLAB

Las funciones predefinidas de MATLAB están construidas de forma que se pueda introducir un array como argumento (entrada) de la función. De esta forma, la operación que ejecuta la función en sí se aplicará a cada uno de los elementos del array, como si se tratara de operaciones elemento a elemento. El resultado (salida) de estas funciones es también un array, en el cual cada elemento es calculado a partir de ejecutar la función sobre los correspondientes elementos del array de entrada. Por ejemplo, si se introduce un vector `x` de siete elementos como argumento de la función `cos(x)`, el resultado es también un vector con siete elementos que se corresponden con el valor del coseno de cada uno de los elementos del vector de entrada `x`. Ejemplo:

```
>> x = [0:pi/6:pi]
x =
   0   0.5236   1.0472   1.5708   2.0944   2.6180   3.1416
>> y = cos(x)
y =
   1.0000   0.86660   0.5000   0.0000   -0.5000   -0.8660   -1.0000
```

Veamos también un ejemplo en donde el argumento de la función es una matriz:

```
>> D = [1 4 9; 16 25 36; 49 64 81]
D =
    1    4    9
   16   25   36
   49   64   81
>> H = sqrt(D)
```

Se crea un array D de 3 × 3.

```
H =
   1   2   3
   4   5   6
   7   8   9
>>
```

Se crea un array H, de dimensión 3 × 3, en el cual cada elemento contiene la raíz cuadrada de cada uno de los elementos de D.

La capacidad que MATLAB ofrece para poder utilizar arrays como argumentos en funciones se denomina vectorización.

3.6 Funciones predefinidas para trabajar con arrays

MATLAB posee un gran número de funciones predefinidas para trabajar con arrays. En la Tabla 3.1 se listan algunas de estas funciones.

Tabla 3.1: Funciones predefinidas para arrays.

Función	Descripción	Ejemplo
`mean(A)`	Si A es un vector, retorna el valor medio de los elementos.	>> A = [5 9 2 4]; >> mean(A) ans = 5
`C = max(A)`	Si A es un vector, C contendrá el elemento mayor de A. Si A es una matriz, C contendrá un vector fila que representa el elemento mayor de cada columna de A.	>> A = [5 9 2 4 11 6 7 11 0]; >> C = max(A) C = 11
`[d,n] = max(A)`	Si A es un vector, d contendrá el elemento mayor de A, y n la posición del elemento (la posición de la primera aparición, si el valor mayor se repite varias veces en el vector).	>> A = [5 9 2 4 11 6 7 11 0]; >> [d,n] = max(A) d = 11 n = 5
`min(A)`	Lo mismo que max(A), pero para el elemento menor.	>> A = [5 9 2 4]; >> min(A) asn = 2
`[d,n] = min(A)`	Lo mismo que [d, n] = max(A), pero para el elemento menor.	>> A = [5 9 2 4]; >> [d,n] = min(A) d = 2 n = 3

Tabla 3.1: Funciones predefinidas para arrays.

`sum(A)`	Si `A` es un vector, calcula la suma de sus elementos.	>> A = [5 9 2 4]; >> sum(A) ans = 20
`sort(A)`	Si `A` es un vector, devuelve el mismo vector ordenado en orden ascendente..	>> A = [5 9 2 4]; >> sort(A) ans = 2 4 5 9
`median(A)`	Si `A` es un vector, devuelve el valor de la mediana de los elementos del vector.	>> A = [5 9 2 4]; >> median(A) ans = 4.5000
`std(A)`	Si `A` es un vector, devuelve las desviación estándar de los elementos del vector.	>> A = [5 9 2 4]; >> std(A) ans = 2.9439
`det(A)`	Devuelve el valor del determinante de la matriz cuadrada `A`.	>> A = [2 4; 3 5]; >> det(A) ans = -2
`dot(a,b)`	Calcula el producto escalar de dos vectores `a` y `b`. Los vectores pueden ser de tipo fila o columna.	>> a = [1 2 3]; >> b = [3 4 5]; >> dot(a,b) ans = 26
`cross(a,b)`	Calcula el producto cruzado de dos vectores `a` y `b`, (`axb`). Ambos vectores deben tener tres elementos.	>> a = [1 2 3]; >> b = [2 4 1]; >> cross(a,b) ans = -5 3 -2
`inv(A)`	Devuelve la inversa de una matriz cuadrada `A`.	>> A = [2 -2 1; 3 2 -1; 2 -3 2]; >> inv(A) ans = 0.2000 0.2000 0 -1.6000 0.4000 1.0000 -2.6000 0.4000 2.0000

3.7 Generación de números aleatorios

Frecuentemente, la simulación de ciertos procesos físicos e ingenieriles requiere de la utilización de números aleatorios para determinados cálculos. MATLAB posee dos comandos, denominados `rand` y `randn`, que pueden ser utilizados para asignar números aleatorios a variables.

El comando `rand`*:*

Este comando genera números aleatorios distribuidos uniformemente entre 0 y 1. Se puede utilizar para asignar estos números a escalares, vectores o matrices, tal y como se muestra en la Tabla 3.2.

Tabla 3.2: El comando `rand`.

Comando	Descripción	Ejemplo
`rand`	Genera un número aleatorio entre 0 y 1.	>> rand ans = 0.2311
`rand(1,n)`	Genera un vector fila de n números aleatorios entre 0 y 1.	>> a = rand(1,4) a = 0.6068 0.4860 0.8913 0.7621
`rand(n)`	Genera una matriz n × n de números aleatorios entre 0 y 1.	>> b = rand(3) b = 0.4565 0.4447 0.9218 0.0185 0.6154 0.7382 0.8214 0.7919 0.1763
`rand(m,n)`	Genera una matriz de m × n de números aleatorios entre 0 y 1.	>> c = rand(2,4) c = 0.4057 0.9169 0.8936 0.3529 0.9355 0.4103 0.0579 0.8132
`randperm(n)`	Genera un vector fila con n elementos que son permutaciones aleatorias de enteros entre 1 y n.	>> randperm(8) ans = 8 2 7 4 3 6 5 1

Otras veces se necesitan números aleatorios que estén distribuidos en un intervalo distinto a (0, 1), o incluso que solamente sean enteros. Para obtener resultados distintos sólo es necesario aplicar las operaciones matemáticas oportunas a la función `rand`. Para obtener números aleatorios que se distribuyan en un intervalo (a, b) sólo es necesario multiplicar el resultado de `rand` por $(b - a)$, y añadir el producto a a:

$$(b - a)*\text{rand} + a$$

Por ejemplo, se podría crear un vector de 10 elementos con números aleatorios entre –5 y 10 como se muestra a continuación ($a = -5$, $b = 10$):

```
>> v = 15*rand(1,10) - 5

v =
   -1.8640   0.6973   6.7499   5.2127   1.9164   3.5174   6.9132   -4.1123   4.0430   -42460
```

Los números enteros aleatorios se pueden generar utilizando alguna función de redondeo. Por ejemplo, se podría crear una matriz de 2 × 15 con números aleatorios enteros en un intervalo de 1 a 100, de la siguiente forma:

```
>> A = round(99*rand(2,15) + 1)

A =
   24   6  64  85  18  45  32  40  13  46  93  17  25  97  87
   25   9  20  18  99  35  37  60   5  87  27  87  65  67   2
```

El comando `randn`*:*

Este comando genera números aleatorios con distribución normal, es decir, con media 0 y desviación típica o estándar 1. Este comando se puede utilizar para generar un solo número, así como un vector o una matriz de números de este tipo. Por ejemplo, para crear una matriz de números aleatorios de 3 × 4 se haría:

```
>> d = randn(3,4)

d =
   -0.4326    0.2877    1.1892    0.1746
   -1.6656   -1.1465   -0.0376   -0.1867
    0.1253    1.1909    0.3273    0.7258
```

En este caso, la media y la desviación estándar de varios números también se pueden cambiar realizando operaciones matemáticas. Esto se hace multiplicando el número aleatorio generado por la desviación estándar deseada y añadiendo al final la media que se desee. Por ejemplo, se podría generar un vector de 10 números enteros con media 50 y desviación estándar 5 tecleando:

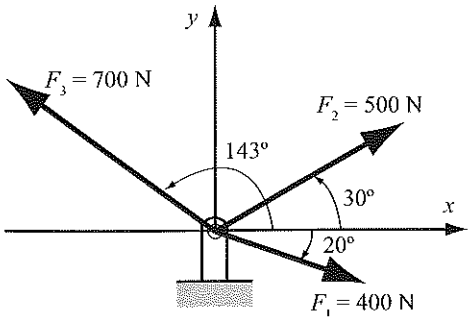
```
>> v = round(5*randn(1,15) + 50)

v =
   53   51   45   39   50   45   53   53   58   53   47   52   45   50   50
```

En los ejemplos anteriores, para obtener la parte entera se ha utilizado la función `round`.

3.8 Ejemplos de aplicaciones con MATLAB

Problema de ejemplo 3.2: Sistema equivalente de fuerzas (suma de vectores)

Se aplican tres fuerzas a un soporte, tal y como se muestra en la figura adjunta. Determinar la fuerza total (equivalente) aplicada al soporte.



Solución

Una fuerza puede considerarse como un vector que, desde el punto de vista físico, posee un módulo y una dirección. En un sistema de coordenadas cartesianas un vector bidimensional **F** puede ser expresado como:

$$\mathbf{F} = F_x\mathbf{i} + F_y\mathbf{j} = F\cos\theta\mathbf{i} + F\,\text{sen}\theta\mathbf{j} = F(\cos\theta\mathbf{i} + \text{sen}\theta\mathbf{j})$$

donde F es el módulo de la fuerza, y θ es el ángulo relativo al eje x. F_x y F_y son las componentes de F en las direcciones de los ejes x e y respectivamente, e **i** y **j** son los vectores unitarios en esas direcciones. Si F_x y F_y son conocidas, entonces F y θ pueden ser calculados de la siguiente forma:

$$F = \sqrt{F_x^2 + F_y^2} \quad \text{y} \quad \tan\theta = \frac{F_y}{F_x}$$

La fuerza total (equivalente) aplicada al soporte se obtiene sumando todas las fuerzas que actúan sobre él. La solución con MATLAB que se propone a continuación sigue los siguientes pasos:

- Se representa cada fuerza como un vector con dos elementos. El primer elemento es la componente x del vector, y el segundo elemento es la componente y.
- Se determina el vector fuerza equivalente sumando los vectores independientes.
- Se determina módulo y dirección de la fuerza equivalente.

```
>> F1M = 400; F2M = 500; F3M = 700;
>> Th1 = -20*pi/180; Th2 = 30*pi/180; Th3 = 143*pi/180;
>> F1 = F1M*[cos(Th1) sin(Th1)]
F1 =
   375.8770   -136.8081
>> F2 = F2M*[cos(Th2) sin(Th2)]
F2 =
   433.127   250.0000
>> F3 = F3M*[cos(Th3) sin(Th3)]
F3 =
   -559.0449   421.2705
>> Ftot = F1 + F2 + F3
```

Define variables con la magnitud de cada vector.

Define variables con el ángulo (en radianes) de cada vector.

Define los tres vectores.

Calcula el vector fuerza total.

```
Ftot =
   249.8449   534.4625

>> FtotM = sqrt(Ftot(1)^2 + Ftot(2)^2)

FtotM =
   589.9768

>> Th = (180/pi)*atan(Ftot(2)/Ftot(1))

Th =
   64.9453
```

Calcula el módulo del vector fuerza total.

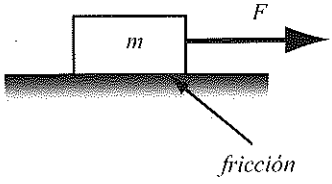
Calcula el ángulo (en grados) del vector fuerza total.

Finalmente, la fuerza equivalente tiene un módulo de 504.62 N, y su dirección es 37,03° relativa al eje x. En notación vectorial, esta fuerza puede representarse por: $\mathbf{F} = 402{,}83\mathbf{i} + 303{,}92\mathbf{j}$ N.

Problema de ejemplo 3.3: Experimento de fricción (procesamiento de arrays elemento a elemento)

El coeficiente de fricción μ se puede calcular experimentalmente midiendo la fuerza F requerida para mover una masa m. A partir de estos parámetros, el coeficiente de fricción se puede calcular de la forma:

$$\mu = F/(mg) \quad (g = 9{,}81 \text{ m/s}^2)$$



En la tabla siguiente se presentan los resultados de seis experimentos en los cuales se midió F. Determinar el coeficiente de fricción en cada experimento, así como el valor medio de todos los experimentos realizados.

Experimento	1	2	3	4	5	6
Masa m (kg)	2	4	5	10	20	50
Fuerza F(N)	12,5	23,5	30	61	118	294

Solución

A continuación se presenta la solución a este problema experimental utilizando MATLAB.

```
>> m = [2 4 5 10 20 50];

>> F = [12.5 23.5 30 61 117 294];

>> mu = F./(m*9.81)

mu =
   0.6371   0.5989   0.6116   0.6218   0.5963   0.5994

>> mu_media = mean(mu)

mu_media =
   0.6109
```

Se crea un vector m con los valores de las masas.

Se crea un vector F con los valores de las fuerzas.

Se calcula el valor μ para cada experimento, utilizando operaciones elemento a elemento sobre los vectores anteriores.

Se calcula la media de μ para cada uno de los experimentos, almacenados dentro del propio vector mu. Se utiliza para ello la función mean.

Problema de ejemplo 3.4: Análisis de circuitos resistivos (resolución de un sistema de ecuaciones lineales)

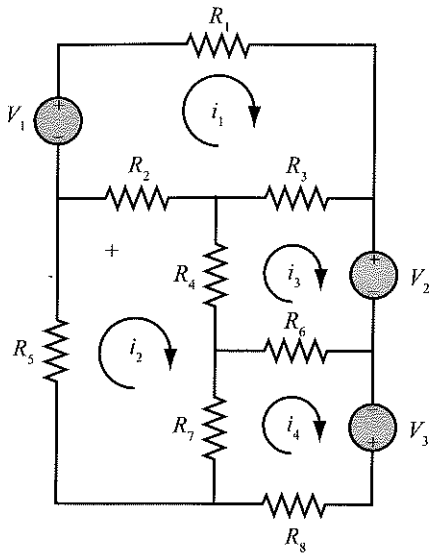
El circuito eléctrico anexo está formado por distintas resistencia y fuentes de alimentación. Determinar la intensidad de corriente que pasa por cada resistencia utilizando para ello las leyes de Kirchhoff para la resolución de circuitos resistivos. Los datos conocidos del circuito son los siguientes:

$V_1 = 20$ V, $V_2 = 12$ V, $V_3 = 40$ V

$R_1 = 18\ \Omega$, $R_2 = 10\ \Omega$, $R_3 = 16\ \Omega$

$R_4 = 6\ \Omega$, $R_5 = 15\ \Omega$, $R_6 = 8\ \Omega$

$R_7 = 12\ \Omega$, $R_8 = 14\ \Omega$.



Solución

Las leyes de Kirchhoff se basan en que la suma de los voltajes alrededor de un circuito cerrado es cero. Para operar, habitualmente se divide el circuito en distintas subredes o mallas, cada una con una intensidad de corriente concreta (i_1, i_2, i_3, i_4). Aplicando la segunda ley de Kirchhoff sobre voltajes a cada red, lo que obtenemos es un sistema de ecuaciones lineales para las corrientes (en nuestro caso cuatro). La solución a este sistema nos da el valor numérico de las corrientes correspondientes a cada malla. La intensidad de corriente que pasa por una resistencia que pertenece a dos mallas es la suma de las corrientes de las mallas correspondientes. Es conveniente asumir que todas las corrientes van en la misma dirección (en nuestro caso, en el sentido de las agujas del reloj). En cada una de las ecuaciones resultantes, el voltaje es positivo si la intensidad de corriente pasa por el polo negativo (–), y el voltaje de la resistencia será negativo en el caso de que la corriente vaya en el mismo sentido que el de la intensidad de corriente de la malla.

Las ecuaciones para las cuatro mallas que dan la solución a este problema son las siguientes:

$$
\begin{aligned}
V_1 - R_1 i_1 - R_3(i_1 - i_3) - R_2(i_1 - i_2) &= 0 \\
-R_5 i_2 - R_2(i_1 - i_2) - R_4(i_2 - i_3) - R_7(i_2 - i_4) &= 0 \\
-V_2 - R_6(i_3 - i_4) - R_4(i_3 - i_2) - R_3(i_3 - i_1) &= 0 \\
V_3 - R_8 i_4 - R_7(i_4 - i_2) - R_6(i_4 - i_3) &= 0
\end{aligned}
$$

Estas cuatro ecuaciones pueden ser representadas en la forma matricial $[A][x] = [B]$:

$$
\begin{bmatrix}
-(R_1 + R_2 + R_3) & R_2 & R_3 & 0 \\
R_2 & -(R_2 + R_4 + R_5 + R_7) & R_4 & R_7 \\
R_3 & R_4 & -(R_3 + R_4 + R_6) & R_6 \\
0 & R_7 & R_6 & -(R_6 + R_7 + R_8)
\end{bmatrix}
\begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix}
=
\begin{bmatrix} -V_1 \\ 0 \\ V_2 \\ -V_3 \end{bmatrix}
$$

A continuación se muestra la solución al sistema de ecuaciones utilizando MATLAB:

```
>> V1 = 20; V2 = 12; V3 = 40;
>> R1 = 18; R2 = 10; R3 = 16; R4 = 6;
>> R5 = 15; R6 = 8; R7 = 12; R8 = 14;
>> A = [-(R1 + R2 + R3) R2 R3 0 R2
   -(R2 + R4 + R5 + R7) R4 R7 R3 R4
   -(R3 + R4 + R6) R6 0 R7 R6
   -(R6 + R7 + R8)]
A =
   -44    10    16     0
    10   -43     6    12
    16     6   -30     8
     0    12     8   -34
>> B = [-V1; 0; V2; -V3]
B =
   -20
     0
    12
   -40
>> I = A\B
I =
   0.8411
   0.7206
   0.6127
   1.5750
>>
```

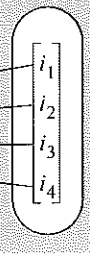
Se definen los voltajes y resistencias.

Se crea la matriz A.

Valores numéricos de la matriz A.

Se crea el vector columna.

Se resuelve el sistema de ecuaciones utilizando división izquierda.



El último vector columna calculado representa los valores de las intensidades de corriente en cada malla. Las corrientes que pasan por las resistencias R_1, R_5 y R_8 son $i_1 = 0{,}8411$ A, $i_2 = 0{,}7206$ A, e $i_4 = 1{,}5750$ A, respectivamente. Respecto a las otras resistencias, estas pertenecen a dos mallas a la vez, y por tanto sus corrientes son la suma de las corrientes en las mallas.

La corriente que pasa por la resistencia R_2 es $i_1 - i_2 = 0{,}1205$ A.

La corriente que pasa por la resistencia R_3 es $i_1 - i_3 = 0{,}2284$ A.

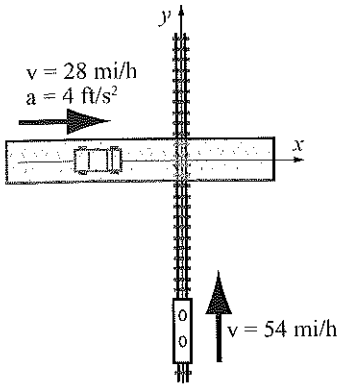
La corriente que pasa por la resistencia R_4 es $i_2 - i_3 = 0{,}1079$ A.

La corriente que pasa por la resistencia R_6 es $i_4 - i_3 = 0{,}9623$ A.

La corriente que pasa por la resistencia R_7 es $i_4 - i_2 = 0{,}8544$ A.

Problema de ejemplo 3.5: Movimiento de dos objetos

Un tren y un coche se aproximan a un cruce. En el instante $t = 0$, el tren está a 400 pies al sur del cruce, viajando hacia el norte a una velocidad de 54 millas por hora. En el mismo instante, el coche se encuentra a 200 pies al oeste del cruce, viajando hacia el este a una velocidad de 28 millas por hora, y con una aceleración de 4 pies por segundo al cuadrado. Determinar las posiciones del tren y del coche, la distancia entre ellos, así como la velocidad del tren relativa al coche en cada segundo, durante 10 segundos.



Para mostrar los resultados, crear una matriz de 11 × 6 en la cual cada fila sea el tiempo en la primera columna y en las siguientes cinco columnas la posición del tren, la posición del coche, la distancia entre el tren y el coche, la velocidad del coche y la velocidad del tren relativa al coche.

Solución

La posición de un objeto que se mueve a lo largo de una línea recta con una aceleración constante viene dada por $s = s_0 + v_0 t + \frac{1}{2}at^2$, donde s_0 y v_0 son la posición y la velocidad en $t = 0$, respectivamente, y a es la aceleración. Aplicando esta ecuación al tren y al coche, tenemos:

$$y = -400 + v_{0\ \text{tren}}t \qquad \text{(tren)}$$

$$x = -200 + v_{0\ \text{coche}}t + \frac{1}{2}a_{\text{coche}}t^2 \qquad \text{(coche)}$$

La distancia entre el coche y el tren es: $d = \sqrt{x^2 + y^2}$.
La velocidad del tren es constante y, en notación vectorial, es: $\mathbf{v}_{\text{tren}} = v_{0\ \text{tren}}\mathbf{j}$. El coche tiene aceleración y su velocidad en el instante t viene dada por: $\mathbf{v}_{\text{coche}} = (v_{0\ \text{coche}} + a_{\text{coche}}t)\mathbf{i}$. La velocidad del tren relativa al coche viene dada por: $\mathbf{v}_{t/c} = \mathbf{v}_{\text{tren}} - \mathbf{v}_{\text{coche}} = -(v_{0\ \text{coche}} + a_{\text{coche}}t)\mathbf{i} + v_{0\ \text{tren}}\mathbf{j}$. El módulo de esta velocidad es la longitud del vector.

El problema se resuelve creando un vector `t` con 11 elementos para los instantes 0 a 10 s, y posteriormente calculando las posiciones del tren y del coche, la distancia entre ellos, y la velocidad del tren relativa al coche en cada instante de tiempo. Los siguientes comandos MATLAB nos proporcionan estos datos y dan la solución al problema.

```
>> v0tren = 54*5280/3600; v0coche = 28*5280/3600; acoche = 4;
>> t = 0:10;
>> y = -400 + v0tren*t;
>> x = -200 + v0coche*t + 0.5*acoche*t.^2;
>> d = sqrt(x.^2 + y.^2);
>> vcoche = v0coche + acoche*t;
```

Se crean las variables para las velocidades iniciales (en pies por segundo) y la aceleración.

Se crea el vector de tiempos `t`.

Se calculan las posiciones del tren y del coche.

Se calcula la distancia entre el tren y el coche.

Se calcula la velocidad del coche.

```
>> velocidad_trenRcoche = sqrt(vcoche.^2 + v0tren^2);
>> tabla = [t' y' x' d' velocidad_trenRcoche']
```

Se calcula la velocidad del tren relativa a la del coche.

Se crea la tabla de 11 × 6 con los datos calculados que pide el problema.

```
tabla =
```

0	-400.0000	-200.0000	447.2136	41.0667	89.2139
1.0000	-320.8000	-156.9333	357.1284	45.0667	91.1243
2.0000	-241.6000	-109.8667	265.4077	49.0667	93.1675
3.0000	-162.4000	-58.8000	172.7171	53.0667	95.3347
4.0000	-83.2000	-3.7333	83.2837	57.0667	97.6178
5.0000	-4.0000	55.3333	55.4777	61.0667	100.0089
6.0000	75.2000	118.4000	140.2626	65.0667	102.5003
7.0000	154.4000	185.4667	241.3239	69.0667	105.0849
8.0000	233.6000	256.5333	346.9558	73.0667	107.7561
9.0000	312.8000	331.6000	455.8535	77.0667	110.5075
10.0000	392.0000	410.6667	567.7245	81.0667	113.3333

```
>>
```

Tiempo (s) | Posición del Tren (pies) | Posición del Coche (pies) | Distancia coche–tren (pies) | Velocidad del Coche (pies/s) | Velocidad del Tren Relativa a la del coche (pies/s)

Nota: en los comandos MATLAB anteriores, `tabla` es el nombre de la matriz que contiene los datos calculados.

En este problema MATLAB muestra los resultados (números) sin texto adjunto. La forma de visualizar texto y números a la vez como resultado de un cálculo se explica detenidamente en el Capítulo 4.

3.9 Problemas

Nota: *Los problemas que siguen se pueden resolver introduciendo comandos en la Ventana de Comandos de MATLAB. No obstante, sería más conveniente usar ficheros script, que serán explicados en el próximo capítulo. Podría considerarse la posibilidad de estudiar primero el Capítulo 4 (al menos las tres primeras secciones), y posteriormente resolver los problemas que aquí se plantean utilizando scripts. En cualquier caso, al final del Capítulo 4 se sugieren también una serie de problemas adicionales sobre operaciones con arrays.*

1. Sean los siguientes vectores:

   $\mathbf{u} = 4\mathbf{i} + 9\mathbf{j} - 5\mathbf{k}$

   $\mathbf{v} = -3\mathbf{i} + 6\mathbf{j} - 7\mathbf{k}$

   Utilice MATLAB para calcular el producto escalar $\mathbf{u} \cdot \mathbf{v}$ de estos vectores de dos formas distintas:

   a) Definiendo **u** como un vector fila y **v** como un vector columna. Utilizar posteriormente la multiplicación matricial.

   b) Utilizando la función `dot`.

2. Sea la función $y = (x^2 + 1)^3 x^3$, calcular el valor de y para los siguientes valores de x: –2,5 –2 –1,5 –1 –0,5 0 0,5 1 1,5 2 2,5 3. Resuelva el problema creando primero un vector x y después creando un vector y, utilizando las operaciones elemento a elemento para el cálculo.

3. La profundidad de un pozo, d, en metros se puede determinar a partir del tiempo que tarda en caer una piedra a su interior (velocidad inicial cero). Este cálculo viene determinado por: $d = \frac{1}{2}gt^2$, donde t es el tiempo en segundos y $g = 9{,}81$ m/s^2.

   Calcular d para $t = 1, 2, 3, 4, 5, 6, 7, 8, 9$ y 10 s. (Cree un vector t y calcule d utilizando operaciones elemento a elemento.)

4. Defina x e y como vectores, tales que $x = 2, 4, 6, 8$ y 10, e $y = 3, 6, 9, 12, 15$. Posteriormente utilice estos vectores en la siguiente expresión para calcular z, a partir de operaciones elemento a elemento.

$$z = \frac{xy + \frac{y}{x}}{(x+y)^{(y-x)}} + 12^{x/y}$$

5. Defina los escalares $h = 0{,}9$ y $k = 12{,}5$, y los vectores $x = 1, 2, 3, 4$, $y = 0{,}9, 0{,}8, 0{,}7, 0{,}6$ y $z = 2{,}5, 3, 3{,}5, 4$. Posteriormente utilice estas variables para calcular T utilizando operaciones elemento a elemento.

$$T = \frac{xyz}{(h+k)^{k/5}} + \frac{ke^{\left(\frac{z}{x}+y\right)}}{z^h}$$

6. Demuestre que $\lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n = e$

   Para hacer esto cree un vector n que tenga los elementos: 1 10 100 500 1000 2000 4000 y 8000. Seguidamente cree un vector y en el cual cada elemento será calculado a partir de los elementos de n mediante $(1 + \frac{1}{n})^n$.

   Compare los elementos de y con el valor de e (teclee `exp(1)` para obtener el valor de e).

7. Utilice MATLAB para demostrar que la serie numérica $\sum_{n=1}^{\infty} \frac{1}{n^2}$ converge a $\pi^2/6$. Para hacer esto, calcule la suma para:

   a) $n = 100$

   b) $n = 1000$

   c) $n = 10\,000$

   Para cada apartado, cree un vector `v` en el cual el primer elemento sea 1, con incremento 1, y como último término 100, 1000 ó 10 000. Utilice posteriormente operaciones elemento a elemento para crear un vector en el cual los elementos sean $1/n^2$. Finalmente, utilice la función `sum` para sumar los términos de la serie. Compare los valores obtenidos en los apartados *a*, *b* y *c* con el valor $\pi^2/6$ (no olvide teclear punto y coma al final de cada comando, ya que si no se visualizarán los vectores completos.)

8. Utilice MATLAB para demostrar que la serie $\sum_{n=0}^{\infty} \frac{1}{(2n+1)(2n+2)}$ converge a ln 2. Para hacer esto, calcule la suma para:

   a) $n = 50$

   b) $n = 5000$

   c) $n = 5000$

Para cada apartado, cree un vector n en el cual el primer elemento sea 0, el incremento 1 y el último término 50, 500 ó 5000. Posteriormente calcule, mediante operaciones elemento a elemento, un vector en el cual los elementos sean $\frac{1}{(2n+1)(2n+2)}$. Finalmente utilice la función sum para sumar los términos de la serie. Compare el valor obtenido en los apartados *a*, *b* y *c* con ln 2.

9. Cree las siguientes matrices:

$$A = \begin{bmatrix} 5 & 2 & 4 \\ 1 & 7 & -3 \\ 6 & -10 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 11 & 5 & -3 \\ 0 & -12 & 4 \\ 2 & 6 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 7 & 14 & 1 \\ 10 & 3 & -2 \\ 8 & -5 & 9 \end{bmatrix}$$

a) Calcule $A + B$ y $B + A$ para demostrar que la suma de matrices cumple la propiedad conmutativa.

b) Calcule $A + (B + C)$ y $(A + B) + C$ para demostrar que la suma de matrices cumple la propiedad asociativa.

c) Calcule $5(A + C)$ y $5A + 5C$ para demostrar que, cuando se multiplica una matriz por un escalar, la multiplicación cumple la propiedad distributiva.

d) Calcule $A*(B + C)$ y $A*B + A*C$ para demostrar que la multiplicación de matrices cumple la propiedad distributiva.

10. Utilice las matrices *A*, *B* y *C* del problema anterior para contestar a las siguientes preguntas:

a) ¿Es $A*B = B*A$?

b) ¿Es $A*(B*C) = (A*B)*C$?

c) ¿Es $(A*B)^t = B^t*A^t$? (^{t} significa transpuesta)

d) ¿Es $(A + B)^t = A^t + B^t$?

11. Resuelva el siguiente sistema de ecuaciones lineales:

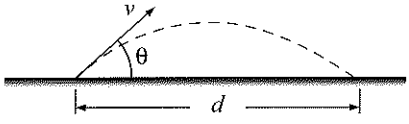
$5x + 4y - 2z + 6w = 4$

$3x + 6y + 6z + 4{,}5w = 13{,}5$

$6x + 12y - 2z + 16w = 20$

$4x - 2y + 2z - 4w = 6$

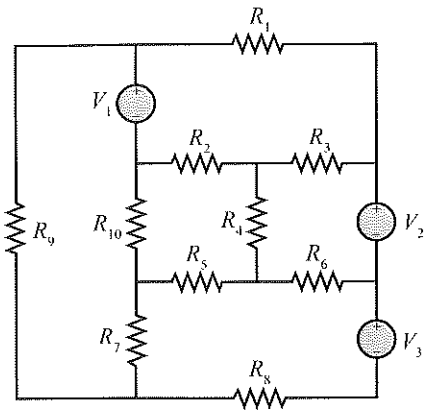
12. Un proyectil se dispara con una velocidad de 750 m/s. Calcule la distancia *d* a la que el proyectil alcanza el suelo si el ángulo de lanzamiento θ cambia de 5° a 85° en incrementos de 5°. Utilice operaciones elemento a elemento. Para visualizar los resultados cree una matriz de 17 × 2 en la cual los elementos de la primera columna sean los ángulos de lanzamiento, y los de la segunda las correspondientes distancias redondeadas al entero más próximo.



13. Dos proyectiles, *A* y *B*, se disparan en el mismo instante desde el mismo punto. El proyectil *A* se dispara a una velocidad de 680 m/s con un ángulo de 65°, mientras que el proyectil *B* se dispara a una velocidad de 780 m/s con un ángulo de 42°. Calcule qué proyectil llega antes a tierra. Seguidamente, tome el tiempo de vuelo t_f de ese proyectil y divídalo en diez incrementos, creando para

ello un vector t con 11 elementos igualmente espaciados (el primer elemento será 0 y el último t_f). Calcule la distancia entre los dos proyectiles para cada una de estas 11 tabulaciones de t.

14. El circuito eléctrico que se muestra a continuación contiene resistencias y fuentes de alimentación. Calcule la intensidad de corriente que pasa por cada resistencia. Utilice para ello una división por mallas en base a las leyes de Kirchhoff (vea el Problema de ejemplo 3.4). Los datos conocidos sobre este circuito son los siguientes:



$V_1 = 38$ V, $V_2 = 20$ V, $V_3 = 24$ V

$R_1 = 15\ \Omega$, $V_2 = 18\ \Omega$, $R_3 = 10\ \Omega$

$R_4 = 9\ \Omega$, $R_5 = 5\ \Omega$, $R_6 = 14\ \Omega$

$R_7 = 8\ \Omega$, $R_8 = 13\ \Omega$, $R_9 = 5\ \Omega$

$R_{10} = 2\ \Omega$.

Capítulo 4
Ficheros script

Hasta ahora los comandos MATLAB que hemos visto se ejecutaban en la Ventana de Comandos. Aunque todos los comandos MATLAB se pueden ejecutar de esta forma, la utilización de la Ventana de Comandos se restringe normalmente a la ejecución de un número pequeño de comandos con salidas bien controladas. En caso contrario, cuando el número de sentencias en demasiado elevada, es necesario plantearse la escritura y ejecución de código de otra forma. El problema proviene, básicamente, de que la Ventana de Comandos no es suficientemente interactiva, los comandos no pueden ser guardados y ejecutados de nuevo a petición del usuario. Eso implica que cada vez que se pulsa la tecla **Intro** sólo se ejecute el último comando, y todo lo anterior permanece inalterable. Si se necesita realizar alguna corrección o cambio sobre algunos de esos comandos previamente ejecutados, será necesario volver a escribirlos y ejecutarlos de nuevo de uno en uno.

Otra forma diferente de ejecutar comandos en MATLAB es crear un fichero con los comandos para ejecutarlo posteriormente. Cuando se ejecuta el fichero en cuestión, los comandos que contiene son ejecutados en el orden en que aparecen en el fichero. Además, si fueran necesarias correcciones o cambios posteriores, sólo habría que editar el fichero y ejecutarlo de nuevo. Los ficheros que se utilizan para este propósito en MATLAB se denominan ficheros de procesamiento de comandos o ficheros script.

4.1 Notas sobre los ficheros script

- Un fichero script es una secuencia de comandos MATLAB, también denominada programa.
- Cuando se ejecuta un fichero script, MATLAB ejecuta los comandos en el orden en que éstos han sido escritos, igual que si se ejecutaran uno a uno en la Ventana de Comandos.
- Cuando un fichero script contiene un comando que produce una salida o resultado (p. ej. una asignación de variable sin un punto y coma al final), la salida se visualiza en la Ventana de Comandos.
- La utilización de ficheros script es conveniente, ya que éstos pueden ser editados (es decir, se pueden corregir o modificar), y se pueden ejecutar tantas veces como se quiera.
- Los ficheros script se pueden crear y editar en cualquier editor de texto. Asimismo el texto se puede copiar y pegar desde estos editores a MATLAB para ser ejecutado.
- Los ficheros script también se denominan ficheros M, ya que la extensión .m es la que utiliza y reconoce MATLAB cuando se guardan estos ficheros.

4.2 Manipulación de ficheros script

En MATLAB, los ficheros script se crean y editan con la Ventana del Editor/Depurador. Para abrir esta ventana hay que ir al menú **File** (Fichero), se selecciona **New** (Nuevo) y después **M-file** (Fichero M). Después de esto se abrirá una ventana, como la que se muestra en la Figura 4.1, correspondiente el editor y depurador de ficheros script de MATLAB.

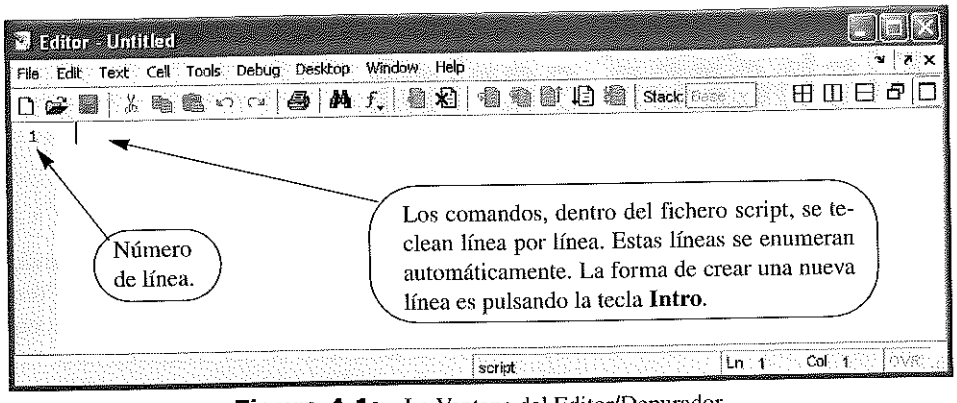


Figura 4.1: La Ventana del Editor/Depurador.

Una vez que se ha abierto la Ventana del Editor/Depurador de scripts, los comandos se van introduciendo línea por línea. MATLAB enumera automáticamente las líneas cada vez que se pulsa la tecla **Intro**. Los comandos también se pueden escribir en cualquier editor o procesador de textos y después trasladarse (copiar/pegar) al editor. En la Figura 4.2 se muestra un ejemplo de un pequeño programa. Como puede apreciarse, las primeras líneas de un fichero script suelen ser comentarios (los cuales no se ejecutan) que comienzan por el carácter %, y que describen de forma resumida la funcionalidad del código que sigue a continuación.

Antes de ejecutar un fichero script se tiene que guardar en disco. Para hacer esto sólo hay que ir a la opción **Save As** (Guardar como) del menú **File** (Fichero), y a continuación seleccionar la localización del archivo y su nombre. Las reglas de asignación de nombres a ficheros script son las mismas que para variables, ya que éstos deben comenzar por una letra y pueden incluir dígitos y el carácter de subrayado, hasta un máximo de 63 caracteres. No se pueden utilizar como nombres para ficheros script las variables ya definidas por el usuario, así como los nombres de variables predefinidas y comandos y funciones existentes en MATLAB.

4.3 Ejecución de un fichero script

Un fichero script se puede ejecutar, bien tecleando su nombre en la Ventana de Comandos (y pulsando la tecla **Intro**) o bien directamente desde la Ventana del Editor a través del icono **Run** (Ejecutar), tal y como se muestra en la Figura 4.2. Sin embargo, antes de hacer esto el usuario debe asegurarse de que MATLAB puede encontrar o tener acceso al fichero. Para ello, el fichero debe estar almacenado bien en el directorio actual por defecto o bien en la ruta de búsqueda de archivos.

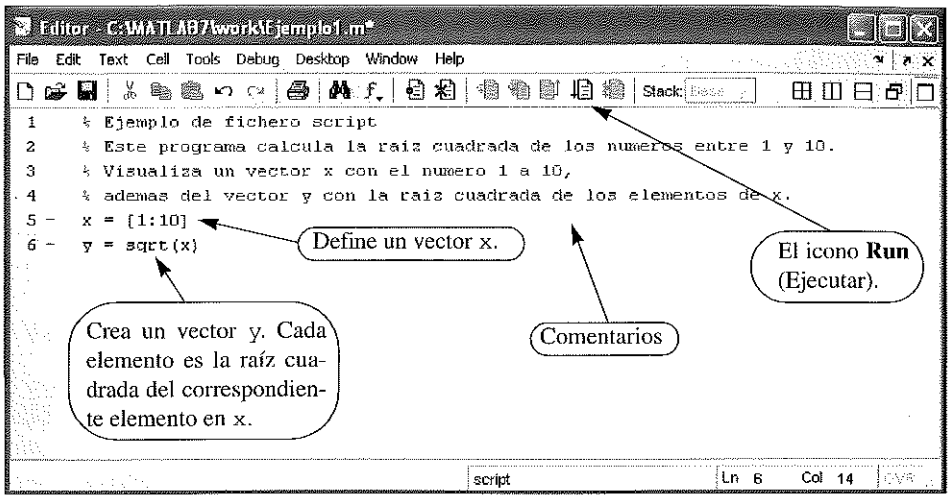


Figura 4.2: Ejemplo de programa en la Ventana del Editor/Depurador.

4.3.1 *El directorio de trabajo actual*

El directorio de trabajo actual se muestra en el campo "Current Directory" (Directorio Actual), "Current Directory", en la barra de herramientas superior de la Ventana de Comandos (ver Figura 4.3). El directorio actual se puede cambiar bien desde la Ventana del Directorio Actual (Current Directory Window) o bien tecleando el comando `cd` en la Ventana de Comandos. Una vez que se utilizan dos o más directorios de trabajo en una sesión, es posible cambiar de uno a otro en el campo de **Directorio Actual** de la Ventana de Comandos. La **Ventana del Directorio Actual**, que se muestra en la Figura 4.4, se abre desde el menú **Desktop** (Escritorio). El Directorio Actual también se puede cambiar eligiendo la unidad y carpeta (ubicación) donde se guardará el fichero en disco.

Una forma alternativa de cambiar el directorio de trabajo es usar el comando `cd` en la Ventana de Comandos. Para cambiar el directorio actual a otra unidad de disco diferente, hay que teclear `cd`, espacio, después el nombre de la unidad seguida por dos puntos :, y a continuación pulsar la tecla **Intro**. Por ejemplo, para cambiar el directorio actual a la unidad A (que se corresponde habitualmente con la unidad de disco de 3 ½), se teclea `cd A:`. Si el fichero script se va a guardar en una carpeta dentro de una

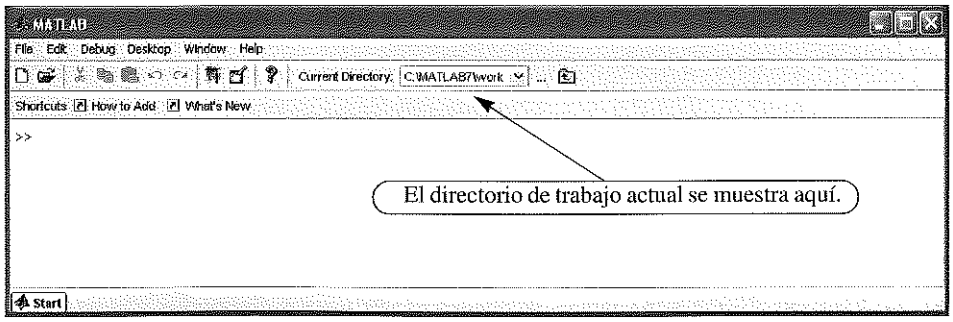


Figura 4.3: El campo Directorio Actual en la Ventana de Comandos.

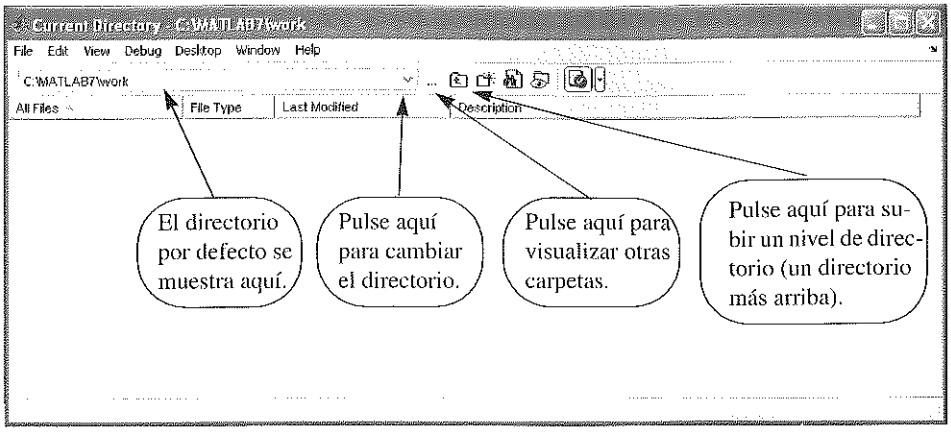


Figura 4.4: La Ventana del Directorio Actual.

unidad de disco, es necesario especificar dicha ruta. Para hacer esto sólo es necesario teclear la ruta como cadena en el comando `cd`. Por ejemplo, `cd('D:\Capitulo 4')` establece la ruta a la carpeta Capitulo 4 que se encuentra en la unidad de disco D. El siguiente ejemplo muestra cómo se cambia el directorio por defecto a la unidad A. Seguidamente se ejecuta el fichero script de la Figura 4.2, que ha sido almacenado en la unidad A como: Capitulo4Ejemplo1.m, y es ejecutado finalmente tecleando su nombre en la Ventana de Comandos y pulsando la tecla **Intro**.

```
>> cd a:
>> Capitulo4Ejemplo1
x =
    1    2    3    4    5    6    7    8    9   10
y =
  Columns 1 through 8
    1.0000    1.4142    1.7321    2.0000    2.2361    2.4495    2.6458    2.8284
  Columns 9 through 10
    3.0000    3.1623
```

El directorio de trabajo actual es ahora la unidad A.

Se ejecuta el fichero script correspondiente, tecleando su nombre y pulsando seguidamente la tecla **Intro**.

La salida generada por el fichero script (vectores x e y) se muestran en la Ventana de Comandos.

4.3.2 *Ruta de búsqueda*

Cuando el usuario ejecuta un fichero script o una función, MATLAB busca el fichero en los directorios que se enumeran en la ruta de búsqueda. Estos directorios se muestran en particular en la Ventana de la Ruta de Búsqueda que se puede abrir seleccionando la opción **Set Path** (Establecer Ruta) en el menú **File** (Fichero). Una vez abierta esta ventana, se pueden añadir nuevos directorios o borrar los existentes, modificando la ruta de búsqueda de archivos en MATLAB.

4.4 Variables globales

Las variables globales son variables que, una vez creadas en MATLAB, son reconocidas en cualquier parte, incluidos todos los ficheros script. En general, cualquier variable definida en la Ventana de Comandos es reconocida en cualquier fichero script. Cuando una variable se define en un fichero script ésta puede ser usada en la Ventana de Comandos. En otras palabras, cualquier variable creada existe, y por lo tanto puede ser manipulada tanto en ficheros script como en la Ventana de Comandos.

Existen, no obstante, tipos especiales de ficheros en MATLAB que no permiten compartir sus variables. Estos ficheros serán explicados en el Capítulo 6 (Sección 6.3).

4.5 Valores de entrada en un fichero script

Cuando se ejecuta un fichero script, las variables utilizadas en los cálculos dentro del fichero deben tener valores asignados previamente. La asignación de valores a estas variables se puede realizar de tres formas, dependiendo de dónde y cómo se haya definido la variable.

1. Variable definida y asignada en el fichero script

En este caso, la asignación del valor a la variable forma parte del fichero. Si el usuario quiere ejecutar el fichero con un valor diferente para dicha variable, el fichero debe ser editado para asignar un nuevo valor a la variable. Una vez que el fichero se ha guardado en disco, se puede ejecutar de nuevo.

El siguiente es un ejemplo de un caso de este tipo. El fichero script (guardado en disco como Capitulo4Ejemplo2) calcula la puntuación media de tres jugadas distintas.

```
% Este fichero script calcula la puntuación media de tres jugadas distintas.
% La asignación de las variables que contienen los puntos es parte del fichero script.
jugada1 = 75;
jugada2 = 93;
jugada3 = 68;
puntuacion_media = (jugada1 + ugada2 + jugada3)/3
```

Las variables son asignadas dentro del propio fichero script.

Una vez ejecutado el fichero, la Ventana de Comandos tendrá un aspecto similar a este:

```
>> Capitulo4Ejemplo2
puntuacion_media =
   78.6667
>>
```

El fichero script se ejecuta a partir de teclear el nombre del fichero que lo contiene.

Se muestra la variable `puntuacion_media`, junto con su valor, en la Ventana de Comandos.

2. Variable definida y asignada en la Ventana de Comandos

En este caso, la asignación de un valor a la variable se realiza en la Ventana de Comandos (recuerde que la variable es reconocida sin problemas dentro del fichero script). Si el usuario quiere ejecutar el

fichero script con un valor diferente para esta variable, se debe asignar el nuevo valor en la Ventana de Comandos, y después ejecutar el fichero de nuevo.

En el ejemplo anterior, en el cual el script consistía en un programa que calculaba la puntuación media de tres jugadas, el fichero script almacenado para este caso (guardado como Capitulo4Ejemplo3) sería:

```
% Este fichero script calcula la puntuación media de tres jugadas distintas.
% La asignación de las variables que contienen los puntos (jugada1, jugada2, jugada3)
% se lleva a cabo en la Ventana de Comandos.

puntuacion_media = (jugada1 + jugada2 + jugada3)/3
```

Una vez ejecutado el fichero, la Ventana de Comandos tendrá un aspecto similar a este:

```
>> jugada1 = 67;
>> jugada2 = 90;
>> jugada3 = 81;
>> Capitulo4Ejemplo3
puntuacion_media =
   79.3333
>> jugada1 = 87;
>> jugada2 = 70;
>> jugada3 = 50;
>> Capitulo4Ejemplo3
puntuacion_media =
   69
>>
```

Asignación de variables en la Ventana de Comandos.

Se ejecuta el fichero script.

La salida del fichero script se muestra en la Ventana de Comandos.

Se asignan nuevos valores a las variables.

El fichero script se ejecuta de nuevo.

La salida del fichero script se muestra en la Ventana de Comandos.

3. Variable definida y asignada en el fichero script, pero además se introduce un valor concreto para la variable cuando se ejecuta el fichero en la Ventana de Comandos

En este caso, la variable se define en el fichero script y cuando se ejecuta dicho fichero al usuario se le pide un valor concreto, a través de la Ventana de Comandos, para asignárselo a la variable del fichero script. Para hacer esto se utiliza el comando `input`. La forma de utilizar este comando es la siguiente:

```
nombre_variable = input ('Mensaje que se muestra
                         en la Ventana de Comandos')
```

Cuando se ejecuta el comando `input` como parte del script, la cadena que va entre paréntesis se visualiza en la Ventana de Comandos. La cadena simboliza un mensaje a partir del cual se le pedirá al usuario un valor para la variable `nombre_variable`. Introducido el valor y pulsada la tecla **Intro**, la variable será asignada con el valor que el usuario haya tecleado. Al igual que sucede con otras variables, el comando `input` deberá finalizarse con punto y coma para que el valor de la variable no se muestre en la Ventana de Comandos. A continuación se muestra un fichero script que utiliza el comando `input` para introducir los puntos de las distintas jugadas, a partir del ya conocido programa que calcula la media de la puntuación del juego.

```
% Este fichero script calcula la puntuacion media de tres jugadas distintas.
% La asignación de las variables que contienen los puntos (jugada1, jugada2, jugada3)
% se lleva a cabo mediante el comando input.
jugada1 = input('Introduzca la puntuacion de la primera jugada ');
jugada2 = input('Introduzca la puntuacion de la segunda jugada ');
jugada3 = input('Introduzca la puntuacion de la tercera jugada ');
puntuacion_media = (jugada1 + jugada2 + jugada3)/3
```

Una vez ejecutado el fichero, la Ventana de Comandos tendrá un aspecto similar a este (el fichero script ha sido guardado con el nombre de Capitulo4Ejemplo4):

```
>> Capitulo4Ejemplo4
Introduzca la puntuacion de la primera jugada 67
Introduzca la puntuacion de la segunda jugada 91
Introduzca la puntuacion de la tercera jugada 70
puntuacion_media =
   76
>>
```

MATLAB muestra los mensajes de los comandos `input`. El usuario introduce cada uno de estos valores pulsando, al final de ellos, la tecla **Intro**.

En este ejemplo se asignan valores escalares a las variables. También se podrían asignar valores a vectores y matrices, utilizando para ello la notación ya presentada, consistente en asignar los valores entre corchetes.

El comando `input` también se puede utilizar para asignar una cadena a una variable. Esto se puede realizar de dos formas. Una es utilizar el comando como ya se ha venido mostrando, introduciendo en este caso la cadena entre comillas simples una vez que el mensaje del comando `input` se visualiza en la Ventana de Comandos. Otra forma es utilizar una opción del comando `input` que

permite concretar que los caracteres serán introducidos como cadena. Este comando tiene la siguiente forma:

```
nombre_variable = input('Mensaje','s')
```

donde 's', dentro del comando, indica que se introducirán caracteres en la entrada. En este caso, cuando el mensaje aparece, la cadena se puede introducir sin comillas simples, pero es asignada a la variable como una cadena. En el Problema de ejemplo 7.4 se incluye un ejemplo de utilización de `input` con esta opción.

4.6 Comandos de salida

Como ya se dijo anteriormente, MATLAB genera automáticamente la salida de la ejecución de algunos comandos. Por ejemplo, cuando a una variable se le asigna un valor, o cuando se teclea el nombre de una variable anteriormente creada y se pulsa **Intro**, MATLAB visualiza el nombre de la variable y su valor asignado. Esto sucede siempre, excepto cuando se teclea al final del comando un punto y coma. Además de esta forma de mostrar datos como salida, MATLAB posee varios comandos que pueden ser utilizados para generar mensajes y salidas más elaboradas. Esto es necesario cuando, aparte del valor de una variable y su nombre, es necesario mostrar mensajes informativos sobre la naturaleza de ciertos datos de salida y/o gráficos. Dos comandos de uso muy frecuente para el fin comentado son `fprintf` y `disp`. El comando `disp` muestra la salida en la pantalla, mientras que el comando `fprintf` se puede usar para visualizar un resultado en la pantalla o para guardarlo en un fichero. Estos comandos se pueden utilizar en la Ventana de Comandos de MATLAB o en un fichero script. Cuando se utilizan en un fichero script, la salida que generan se muestra directamente en la Ventana de Comandos.

4.6.1 *El comando* `disp`

Este comando se utiliza para visualizar un texto o el contenido de una variable sin mostrar su nombre. El formato de este comando es:

```
disp(nombre de variable) o disp('Mensaje')
```

- Cada vez que se ejecuta el comando `disp`, la salida aparece en una nueva línea. Por ejemplo:

```
>> abc = [5 9 1; 7 2 4];
>> disp(abc)
   5   9   1
   7   2   4
>> disp('El problema no tiene solucion')
El problema no tiene solucion
>>
```

Se asigna un array de 2 × 3 a la variable `abc`.

Se utiliza el comando `disp` para visualizar el contenido de la variable `abc`.

Se muestra el contenido del array sin visualizar el nombre de dicho array.

Utilización del comando `disp` para visualizar un mensaje en pantalla.

El siguiente ejemplo muestra el uso de `disp` en un fichero script que calcula la puntuación media de tres jugadas distintas:

```
% Este fichero script calcula la puntuación media de tres jugadas distintas.
% La asignación de las variables que contienen los puntos (jugada1, jugada2, jugada3)
% se lleva a cabo mediante el comando input.
% Se utiliza el comando disp para visualizar la salida.

jugada1 = input('Introduzca la puntuacion de la primera jugada ');
jugada2 = input('Introduzca la puntuacion de la segunda jugada ');
jugada3 = input('Introduzca la puntuacion de la tercera jugada ');
puntuacion_media = (jugada1 + jugada2 + jugada3)/3
disp(' ')                                   % Muestra una línea en blanco.
disp('La puntuacion media del juego es:')   % Muestra el texto.
disp(' ')                                   % Muestra una línea en blanco.
disp(puntuacion_media)                      % Muestra el valor de la variable puntuacion_media.
```

Cuando se ejecuta este fichero (guardado como Capitulo4Ejemplo5), la salida que puede verse en la Ventana de Comandos es la siguiente:

```
>> Capitulo4Ejemplo5
Introduzca la puntuacion de la primera jugada 89
Introduzca la puntuacion de la segunda jugada 60
Introduzca la puntuacion de la tercera jugada 82

                                            % Se muestra una línea en blanco.
La puntuacion media del juego es:           % Se muestra el mensaje deseado.
                                            % Se muestra una línea en blanco.
 77                                         % Se muestra el valor de la variable puntuacion_media.
```

- Sólo se puede visualizar una sola variable en un comando `disp`. Si se necesitan visualizar juntos los elementos de dos variables, se debe primero crear una nueva variable (que contenga los elementos que se van a visualizar) y después mostrarlos en pantalla.

En algunos casos es necesario visualizar datos en forma de tabla. Esto se puede hacer definiendo una variable de tipo array con los elementos a visualizar y luego utilizar el comando `disp` para mos-

trar el array. Los encabezados para la visualización de arrays también se pueden crear con el comando `disp`, aunque no se puede controlar el formato y por lo tanto no se podría variar el ancho de cada columna y la distancia entre ellas. La única forma de hacer esto es utilizando espacios adicionales. A continuación se muestra un ejemplo de fichero script que muestra datos, en forma de tabla, a partir de los datos sobre población que ya aparecieron en el Capítulo 2:

```
yr = [1984 1986 1988 1990 1992 1994 1996];
pop = [127 130 136 145 158 178 211];
tablaYP(:,1) = yr';
tablaYP(:,2) = pop';
disp('    ANNO    POBLACION')
disp('          (MILLONES)')
disp(' ')
disp(tablaYP)
```

Se introducen los datos de la población en dos vectores fila.

Se introduce `yr` como primera columna de la matriz `tablaYP`.

Se introduce `pop` como segunda columna de la matriz `tablaYP`.

Se visualiza el encabezado (primera línea).

Se visualiza el encabezado (segunda línea).

Se visualiza una línea en blanco.

Se visualiza la matriz `tablaYP`.

Cuando se ejecuta este script (guardado en disco como TablaPob), la salida en la Ventana de Comandos es como sigue:

```
>> TablaPob
   ANNO        POBLACION
               (MILLONES)

    1984          127
    1986          130
    1988          136
    1990          145
    1992          158
    1994          178
    1996          211
```

Se muestran los encabezados.

Se visualiza una línea en blanco.

Se muestra la matriz `tablaYP`.

Otro ejemplo, en el cual se visualiza una tabla, aparece en el Problema de ejemplo 4.3. Las tablas también pueden ser creadas y visualizadas con el comando `fprintf`, que será explicado en la próxima sección.

4.6.2 *El comando* `fprintf`

El comando `fprintf` se utiliza para visualizar salidas de programas (texto y datos) en la pantalla, o bien para almacenarlas en un fichero. Con este comando, y a diferencia de `disp`, la salida puede tener un formato preestablecido. En este caso se pueden combinar texto y resultados numéricos provenientes de cálculos o variables predefinidas en la misma línea. Además, el formato de los números se puede controlar directamente con este comando.

Gracias a su expresividad, el comando `fprintf` es útil en la visualización de salidas, pero esta misma razón hace que este comando sea un tanto complejo y con una sintaxis larga en algunos casos. Para evitar cualquier confusión, vamos a presentar este comando gradualmente. Primero veremos cómo utilizar `fprintf` para visualizar mensajes de texto para después ver cómo podemos combinar números y cadenas en la salida. Seguidamente veremos cómo dar formato a la visualización de números, y finalmente cómo utilizar este comando para almacenar datos en disco.

Uso del comando `fprintf` *para visualizar mensajes de texto:*

Para la visualización de texto, el comando `fprintf` se utiliza de la forma:

```
fprintf('Mensaje en forma de cadena')
```

Por ejemplo:

```
fprintf('El problema no tienen solucion. Por favor, compruebe los datos de entrada.')
```

Si esta línea es parte de un fichero script, una vez ejecutada, se visualizará el siguiente mensaje en la Ventana de Comandos:

```
El problema no tiene solucion. Por favor, compruebe los datos de entrada.
```

Con el comando `fprintf` es posible empezar una nueva línea e indicarlo en mitad de la cadena que se introduce como parámetro. Esto se hace insertando `\n` antes del carácter que va a empezar en la línea siguiente. Por ejemplo, si se inserta `\n` después de la primera oración del ejemplo anterior, el comando quedaría:

```
fprintf('El problema no tienen solucion. \nPor favor, compruebe los datos de entrada.')
```

Una vez ejecutado, la salida de este comando sería:

```
El problema no tiene solucion.
Por favor, compruebe los datos de entrada.
```

Al carácter `\n` se le denomina carácter de escape. Es uno de los caracteres utilizados para controlar la salida. Además, existen estos otros caracteres de escape que pueden ser insertados dentro de una cadena:

`\b`	Carácter de borrado
`\t`	Tabulación horizontal.

Cuando un programa contiene más de un comando `fprintf`, la salida que se genera es continua o seguida (el comando `fprintf` no salta automáticamente de línea). Esto sucede incluso cuando existen otras sentencias entre comandos `fprintf`. El siguiente ejemplo de fichero script es una muestra de ello:

```
fprintf('El problema no tiene solucion. Por favor, compruebe los datos de entrada.')
x = 6; d = 19 + 5*x;
fprintf('Intente ejecutar el programa mas tarde.')
y = d + x;
fprintf('Utilice diferentes valores en la entrada.')
```

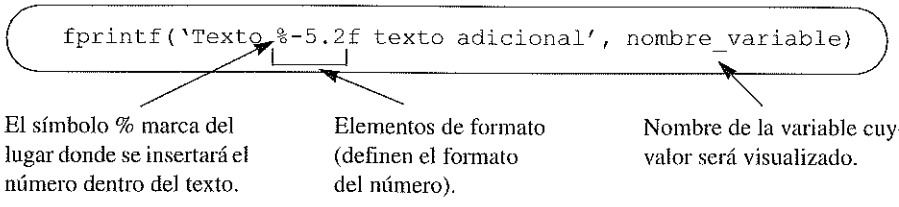
Al ejecutar este fichero, la salida que puede verse en la Ventana de Comandos es:

```
El problema no tiene solucion. Por favor, compruebe los datos de entrada. Intente ejecutar el programa mas
tarde. Utilice diferentes valores en la entrada.
```

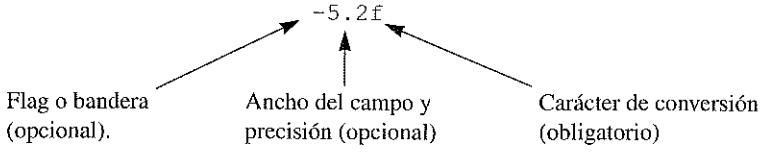
Para saltar una línea antes de escribir un mensaje con el comando `fprintf`, el carácter `\n` debe aparecer al principio de la cadena.

Utilización del comando `fprintf` para visualizar datos y texto juntos:

Para visualizar texto y datos (valores de variables) juntos, el comando `fprintf` debe utilizarse siguiendo la sintaxis:



Los elementos de formato son;



El flag o bandera, cuyo carácter es opcional, puede ser uno de los siguientes:

Carácter utilizado para la bandera	Descripción
- (signo menos)	Justificación izquierda del número dentro del campo
+ (signo más)	Visualiza el carácter de signo (+ o –) delante del número
0 (cero)	Añade ceros si el número es más pequeño que el campo.

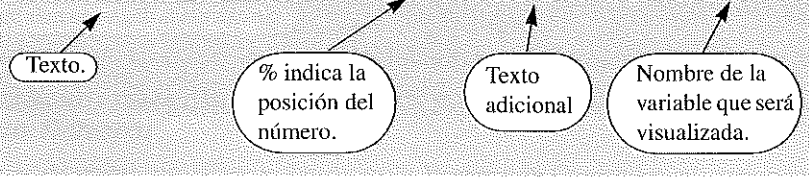
La especificación del ancho y precisión del campo (5.2 en el ejemplo anterior) es opcional. El primer número (5 en nuestro ejemplo) es el ancho del campo, el cual nos indica el menor número de dígitos en la visualización. Si el número que se visualiza es menor que el ancho del campo, se añadirán ceros o espacios delante del número en cuestión. La precisión se corresponde con el segundo número (2 en el ejemplo anterior), y especifica el número de dígitos que se mostrarán a la derecha del punto decimal.

El último elemento es el correspondiente al formato de conversión (f en el ejemplo anterior). Éste es obligatorio. A continuación se muestran los caracteres de conversión más utilizados:

e	Notación exponencial en minúsculas (ej. 1.709098e+001).
E	Notación exponencial en mayúsculas (ej. 1.709098E+001).
f	Notación de punto fijo (ej. 17.090980).
g	Representación en formato corto de las notaciones e o f.
G	Representación en formato corto de las notaciones E o f.
i	Entero.

Se puede obtener información adicional sobre estos y otros formatos utilizando la ayuda de MATLAB, en el menú **Help** (Ayuda). A continuación se muestra un ejemplo de utilización del comando `fprintf` en el programa de cálculo de la puntuación media de tres jugadas.

```
% Este fichero script calcula la puntuación media de tres jugadas distintas.
% La asignación de la variable (esta vez un vector) que contiene los puntos (jugada)
% se lleva a cabo mediante el comando input.
% Se utiliza el comando fprintf para visualizar la salida.
jugada(1) = input('Introduzca la puntuacion de la primera jugada ');
jugada(2) = input('Introduzca la puntuacion de la segunda jugada ');
jugada(3) = input('Introduzca la puntuacion de la tercera jugada ');
puntuacion_media = mean (jugada);
fprintf('Se consiguio una puntuacion media de %f en las tres jugadas.', puntuacion_media)
```



Como se puede comprobar, además de utilizar el comando `fprintf`, este programa se diferencia de los anteriores en que la puntuación se almacena en un vector llamado `jugada`, y el valor medio se calcula esta vez utilizando la función para el cálculo de la media: `mean`. La Ventana de Comandos resultante después de esta ejecución del fichero script (almacenado con nombre Capitulo4Ejemplo6) es la siguiente:

```
>> Capitulo4Ejemplo6
Introduzca la puntuacion de la primera jugada 75
Introduzca la puntuacion de la primera jugada 60
Introduzca la puntuacion de la primera jugada 81
Se consiguio una puntuacion media de 72.000000 en las tres jugadas.
>>
```

La salida generada por el comando `fprintf` combina texto con números (valor de la variable).

Con el comando `fprintf` es posible insertar tantos valores numéricos (variables) como se quiera dentro del texto. Para hacer esto sólo es necesario teclear % (seguido del formato del elemento) en tantos sitios como se desee dentro de la cadena de texto, poniendo además después de la cadena, y separadas por comas, las variables a las que se hacen referencia. Debe haber una por cada %, ordenadas de izquierda a derecha por orden de aparición en el texto según se van reemplazando por las marcas %. Según esta aclaración, el formato general del comando `fprintf` es el siguiente:

```
fprintf('...texto...%g...%g...%f...', variable1, variable2, variable3)
```

He aquí un ejemplo materializado en el siguiente fichero script:

```
% Este programa calcula la distancia de un proyectil en vuelo,
% a partir de su velocidad inicial y del ángulo de disparo.
% Se utiliza el comando fprintf para visualizar la salida, mezclando texto y datos numericos.

v = 1584; %Velocidad inicial (km/h)
theta = 30; %Angulo (grados)
vms = v*1000/3600;                                   Se transforma la velocidad a m/s.
t = vms*sin(30*pi/180)/9.81;                         Se calcula el tiempo al punto más alto.
d = vms*cos(30*pi/180)*2*t/100;                      Se calcula la distancia máxima.
fprintf('Un proyectil disparado a %3.2f grados con una velocidad de %4.2f km/h, recorre una distancia de %g
  km.', theta,v,d)
```

Cuando se ejecuta este fichero (guardado como Capitulo4Ejemplo7), la salida en la Ventana de Comandos es la siguiente:

```
>> Capitulo4Ejemplo7
Un proyectil disparado a 30.0 grados con una velocidad de 1584.00 km/h, recorre una distancia de 17.09 km.
>>
```

Consideraciones adicionales sobre el comando `fprintf`*:*

- Si se desea introducir una comilla simple dentro del texto, debe teclearse doblemente dentro de la cadena.
- El comando `fprintf` permite vectorización. Esto significa que cuando se introduce una variable de tipo array para ser visualizada, el comando muestra tantos elementos como haya en el array, utilizando incluso una visualización apropiada, en forma de columnas y filas, para el caso de las matrices.

Por ejemplo, el siguiente fichero script crea una matriz `T` de 2 × 5, en la cual la primera fila está compuesta por los números del 1 al 5, y la segunda por sus raíces cuadradas.

```
x = 1:5;                                          (Se crea el vector x.)
y = sqrt(x);                                      (Se crea el vector y.)
T = [x; y]          (Se crea la matriz T de 2 × 5, la primera fila es x y la segunda y.)
fprintf('Si el numero es: %i, su raiz cuadrada es: %f\n',T)
          (El comando fprintf visualiza dos números en cada línea, a partir de la variable T.)
```

Cuando se ejecuta este fichero, la salida en la Ventana de Comandos es como sigue:

```
T =                                               (Se muestra la matriz T de 2 × 5.)
   1.0000   2.0000   3.0000   4.0000   5.0000
   1.0000   1.4142   1.7321   2.0000   2.2361

Si el numero es: 1, su raiz cuadrada es: 1.000000
Si el numero es: 2, su raiz cuadrada es: 1.414214
Si el numero es: 3, su raiz cuadrada es: 1.732051
Si el numero es: 4, su raiz cuadrada es: 2.000000
Si el numero es: 5, su raiz cuadrada es: 2.236068
```

(El comando `fprintf` repite su salida 5 veces, utilizando los números de la matriz `T`, columna tras columna.)

Utilización del comando `fprintf` *para guardar la salida generada en un fichero:*

Además de poder visualizar la salida en la Ventana de Comandos, el comando `fprintf` nos da la posibilidad volcar esta salida a un fichero. La ventaja es que, guardando el resultado, es posible reutilizarlo en MATLAB o en otras aplicaciones.

Para guardar la salida en un fichero se requieren los siguientes pasos:

a) Abrir el fichero utilizando el comando `fopen`.

b) Escribir la salida al fichero utilizando el comando `fprintf`.

c) Cerrar el fichero utilizando el comando `fclose`.

Paso a:

Antes de que los datos sean escritos en un fichero, éste debe ser abierto. Para hacer esto se necesita el comando `fopen`, que crea un fichero nuevo vacío o abre un fichero ya existente en disco para trabajar con él. La sintaxis del comando es la siguiente:

```
fid = fopen('nombre_fichero','permisos')
```

`fid` es una variable denominada identificador de archivo. Esta variable guarda un escalar correspondiente al fichero que se ha abierto. El nombre de fichero se escribe junto con su extensión. En cuanto a los permisos, se corresponden con una serie de códigos que le dicen al sistema cómo abrir el archivo. Alguno de los permisos más comúnmente usados son los siguientes:

- `'r'` Indica que el fichero se abre para leer de él (incluida por defecto).
- `'w'` Indica que el fichero se abre para escritura. Si el fichero ya existe, su contenido será eliminado. Si el fichero no existe se creará vacío.
- `'a'` Indica lo mismo que 'w', excepto que si el fichero existe entonces los datos serán añadidos al final del fichero (los elementos existentes no se eliminan).

Si no se incluyen permisos en el comando `fopen`, se toma el permiso por defecto, que es 'r'. Para obtener una descripción más detallada de estos y otros permisos, puede acudirse a la ayuda de MATLAB.

Paso b:

Una vez que el fichero está abierto, se utiliza el comando `fprintf` para escribir la salida en el fichero. El comando `fprintf` se utiliza de forma idéntica a como se ha explicado anteriormente. La única diferencia es que ahora hay que poner el identificador de archivo (`fid`) delante del texto en el comando, de la forma:

```
fprintf(fid,'texto %-5.2f texto adicional', nombre_variable)
```

Se ha añadido el descriptor de archivo `fid` al comando `fprintf`

Paso c:

Cuando se ha terminado de escribir los datos en el archivo, éste debe ser cerrado utilizando el comando `fclose`. La sintaxis de este comando es la siguiente:

```
fclose(fid)
```

Notas adicionales sobre uso del comando `fprintf` para guardar la salida en un fichero:

- El fichero se crea en el directorio de trabajo por defecto.
- Es posible utilizar el comando `fprintf` para escribir en más de un fichero a la vez. Para hacer esto simplemente se abren tantos ficheros como sean necesarios con el comando `fopen`, obteniendo distintos identificadores para cada archivo (p. ej. `fid1`, `fid2`, `fid3`, etc.), después se

utiliza el identificador deseado dentro del comando `fprintf` para volcar la salida a uno u otro fichero. Todos los ficheros abiertos deben ser cerrados al final con el comando `fclose`.

A continuación se muestra un ejemplo de fichero script que ilustra la utilización del comando `fprintf` para volcar datos en dos archivos distintos. El programa genera dos tablas de conversión de unidades. Una de las tablas convierte magnitudes de velocidad, de millas por hora a kilómetros por hora. La otra tabla convierte magnitudes de fuerza, de libras-fuerza a newtons. Cada conversión se guarda en un fichero de texto distinto (con extensión.txt).

```
% En este fichero script se utiliza fprintf para escribir la salida en dos ficheros.
% Se crean dos tablas de conversión que se guardan en dos ficheros distintos.
% Una convierte mi/h a km/h, y la otra lb-fuerz a N.
clear all
Vmph = 10:10:100;                                   Se crea un vector de velocidades en mi/h.
Vkmh = Vmph.*1.609;                                 Se convierten mi/h a km/h.
TBL1 = [Vmph; Vkmh];                                Se crea una tabla (matriz) con dos filas.
Flb = 200:200:2000;                                 Se crea un vector de fuerzas en lb-fuerza.
FN = Flb.*4.448;                                    Se convierten lb-fuerza a N.
TBL2 = [Flb; FN]                                    Se crea una tabla (matriz) con dos filas.
fid1 = fopen('VmphtoVkm.txt','w');                  Se abre (se crea la primera vez) un
                                                    fichero.txt llamado VmphtoVkm.
fid2 = fopen('FlbtoFN.txt','w');                    Se abre (se crea la primera vez) un fichero.txt llamado FlbtoFN.
fprintf(fid1,'Tabla de Conversión de Velocidades\n\n');   Se escriben un título y una línea
                                                    en blanco en el fichero fid1.
fprintf(fid1,' mi/h km/h \n');                      Se escriben unos encabezados en el fichero fid1.
fprintf(fid1,' %8.2f %8.2f\n',TBL1);                Se escriben los datos de la variable TBL1 al fichero fid1.
fprintf(fid2,'Tabla de Conversión de Fuerzas\n\n');
fprintf(fid2,' Lb-fuerza Newtons \n');              Se escribe la tabla de conversión de fuerzas
fprintf(fid2,' %8.2f %8.2f\n',TBL2);                (datos de la tabla TBL2) al fichero fid2.
fclose(fid1);
fclose(fid2);                                       Se cierran los ficheros fid1 y fid2.
```

Cuando el fichero script anterior es ejecutado, se crean dos ficheros de texto (.txt) en el directorio de trabajo actual, llamados VmphtoVkm y FlbtoFN. Estos ficheros se pueden abrir con cualquier aplicación que sea capaz de procesar ficheros de texto (.txt). En la Figuras 4.5 y 4.6 se muestran los contenidos de estos dos ficheros utilizando Microsoft Word.

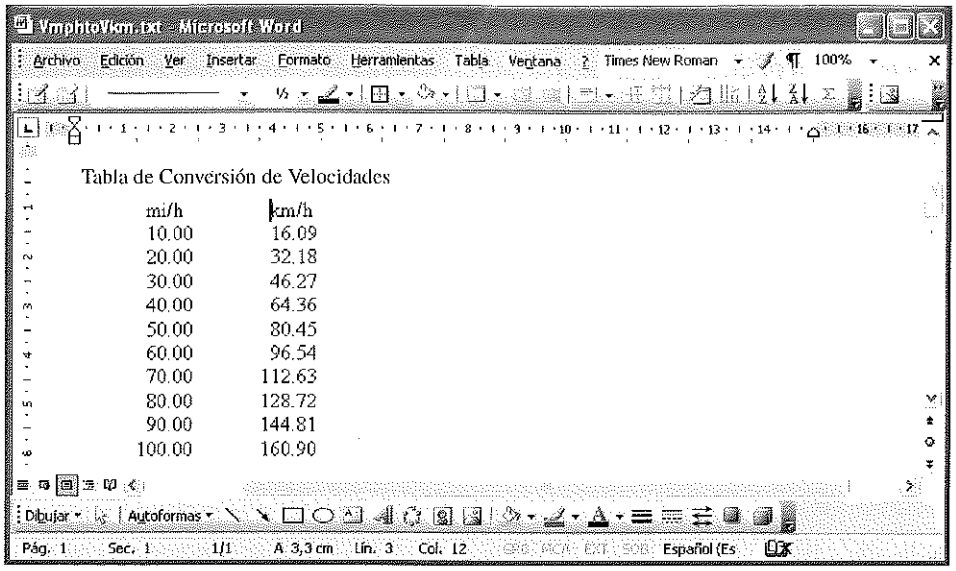


Figura 4.5: El contenido del fichero VmphtoVkm utilizando Word.

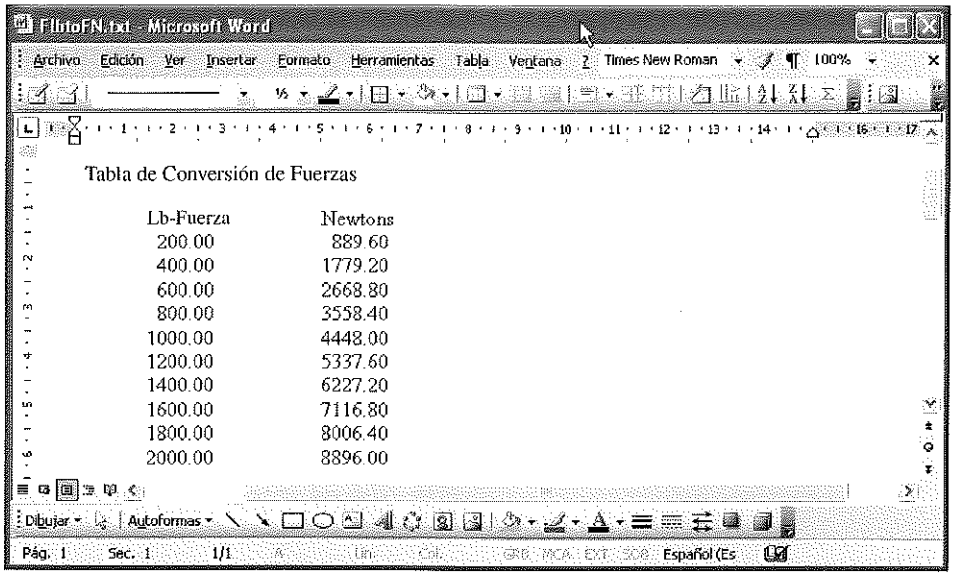


Figura 4.6: El contenido del fichero FlbtoFN utilizando Word.

4.7 Importación y exportación de datos

MATLAB es un software comúnmente utilizado para el análisis de datos experimentales que pueden provenir de distintas fuentes, incluidos otros programas. La forma de procesar estos datos externos es importarlos a MATLAB. De forma análoga, los datos calculados por MATLAB pueden ser transferidos o exportados a otras aplicaciones. Aunque existen distintos tipos de datos (números, texto, audio,

gráficos e imágenes), en esta sección se describirá exclusivamente cómo importar y exportar datos numéricos, que es probablemente el tipo de datos más comúnmente usado en este tipo de transferencias de información. Para otros tipos de datos puede consultarse la ayuda de MATLAB.

La importación de datos se puede llevar a cabo mediante comandos o utilizando un Asistente de Importación de Datos (Import Wizard). Los comandos son útiles cuando el formato de los datos importados es conocido. MATLAB posee varios comandos que pueden ser utilizados para importar distintos tipos de datos. Los comandos de importación se pueden incluir en un fichero script, de forma que los datos se importen dentro del programa y antes de ser procesados por éste. El Asistente de Importación de Datos es útil cuando el formato de los datos (o el comando utilizado para la importación de datos) no es conocido. Este asistente determina el formato de los datos y posteriormente los importa automáticamente.

4.7.1 *Comandos utilizados en la importación y exportación de datos*

Esta sección muestra en detalle cómo transferir datos desde/hacia hojas de cálculo Excel. Microsoft Excel es una herramienta muy utilizada para la gestión de datos, además los datos que maneja esta aplicación son altamente compatibles con bastantes tipos de dispositivos y aplicaciones convencionales. MATLAB permite la transferencia de datos directamente a formatos como .csv y ASCII, así como hojas de cálculo de tipo Lotus 123. Para mayor detalle sobre otros comandos, lo mejor es mirar, como siempre, la ayuda de MATLAB en la Ventana de Ayuda.

Importación y exportación de datos hacia/desde Excel:

Para llevar a cabo la importación de datos desde Excel se utiliza el comando `xlsread`. Este comando importa los datos de una hoja de cálculo Excel a una variable de tipo array. La forma más simple de utilizar este comando es:

```
nombre_variable = xlsread('nombre_fichero')
```

- `'nombre_fichero'`, introducido como cadena, es el nombre del fichero Excel. La ubicación de este fichero debe ser el directorio de trabajo actual o bien estar en la ruta de búsqueda.
- Si el fichero Excel importado tiene más de una hoja de cálculo sólo se importarán los datos de la primera de las hojas.

Si un fichero Excel contiene más de una hoja, se puede utilizar otra versión del comando `xlsread` para decidir cuál de ellas importar:

```
nombre_variable = xlsread('nombre_fichero','nombre_hoja')
```

- El nombre de la hoja debe introducirse como cadena.

Otra opción del mismo comando permite importar sólo una región (grupo de filas y columnas) de una hoja de cálculo determinada a partir de un fichero Excel:

```
nombre_variable = xlsread('nombre_fichero','nombre_hoja', 'rango')
```

- El `'rango'`, introducido como cadena, es una región rectangular de la hoja definida por la dirección (en notación Excel) de las celdas con respecto a las esquinas opuestas (superior izquierda, inferior derecha). Por ejemplo, 'C2:E5' representa una región de dimensión 4 × 3 a partir de las filas 2, 3, 4 y 5, y de las columnas C, D y E.

El paso inverso, es decir, la exportación de datos MATLAB a una hoja Excel, se lleva a cabo mediante el comando `xlswrite`, cuya sintaxis en su versión reducida es:

```
xlswrite('nombre_fichero',nombre_variable)
```

- `'nombre_fichero'`, introducido como cadena, es el nombre del fichero Excel al cual se quieren exportar los datos. El fichero debe estar en el directorio actual. Si el fichero no existe se creará con el nombre especificado por parámetro.
- `nombre_variable` es el nombre de la variable MATLAB que contiene los datos que serán exportados.
- Los argumentos `'nombre_hoja'` y `'rango'` también pueden ser añadidos al comando `xlswrite` para exportar los datos a una hoja concreta del fichero, dentro de un rango específico.

A continuación se muestra un ejemplo en el cual se importan los datos de la hoja de cálculo Excel de la Figura 4.7 a MATLAB, utilizando el comando `xlsread`.

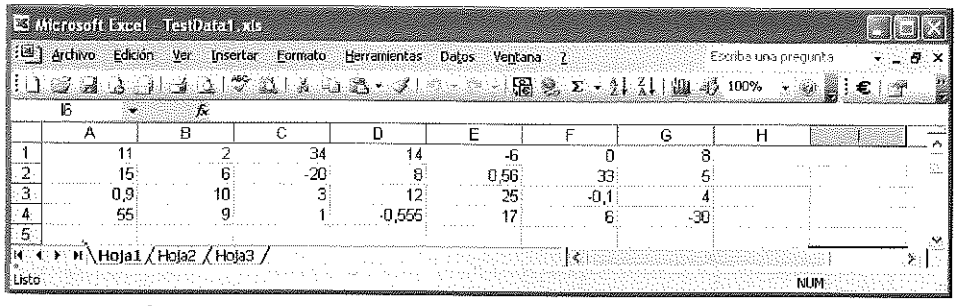


Figura 4.7: Hoja de cálculo Excel con los datos que se importarán.

La hoja de cálculo creada se llama DatosTest1, y se encuentra en la unidad de disco A. Una vez que el directorio por defecto se ha cambiado a la unidad A, se pueden importar los datos en MATLAB y asignárselos a la variable DATOS, como sigue:

```
>> DATOS = xlsread('DatosTest1')

DATOS =
   11.0000    2.0000   34.0000   14.0000   -6.0000         0    8.0000
   15.0000    6.0000  -20.0000    8.0000    0.5600   33.0000    5.0000
    0.9000   10.0000    3.0000   12.0000  -25.0000   -0.1000    4.0000
   55.0000    9.0000    1.0000   -0.5550   17.0000    6.0000  -30.0000
```

4.7.2 *Utilización del Asistente de Importación de Datos*

La utilización del Asistente de Importación de Datos permite importar datos a MATLAB de una forma sencilla, sin tener que hacer suposiciones previas sobre el formato de los datos. Este asistente se activa utilizando la opción **Import Data** (Importar Datos) del menú **File** (Fichero) de la Ventana de Comandos. También se puede activar tecleando el comando `uiimport` desde la propia Ventana de Comandos. El asistente comienza visualizando una ventana de selección de archivos que muestra todos los ficheros y formatos reconocidos por el asistente. El usuario debe seleccionar el fichero en cuestión que contiene los datos a importar, y después pulsar **Open** (Abrir). Seguidamente el asistente abre el fichero

y visualiza una parte de los datos en una ventana de previsualización, de forma que el usuario pueda verificar que los datos son los correctos. El asistente tratará de procesar los datos, y si lo consigue entonces visualizará las variables que ha creado con una parte de los datos. Cuando el usuario pulsa el botón **Next** (Siguiente), el asistente muestra los separadores de columnas utilizados para la estructuración de los datos. Si los datos son apropiados, el usuario puede proceder, pulsando de nuevo **Next** (Siguiente), o escoger un separador diferente de columnas para los datos. En la siguiente ventana, el asistente muestra el nombre y el tamaño de las variables creadas en MATLAB. Cuando los datos son numéricos, las variables tendrán el mismo nombre que el fichero importado. Una vez que el asistente ha finalizado, pulsando el botón **Finish** (Finalizar), los datos se importan en MATLAB.

Como ejemplo ilustrativo, se muestra la utilización del Asistente de Importación de Datos para importar datos numéricos en formato ASCII, guardados en un fichero de texto (.txt). Los datos están guardados con el nombre TestData2, tal y como se muestra en la Figura 4.8.

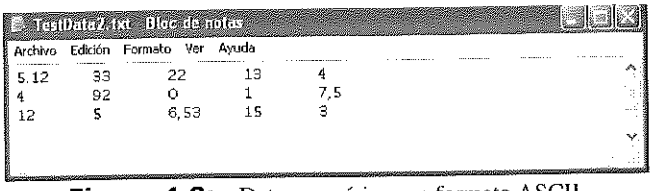


Figura 4.8: Datos numéricos en formato ASCII.

En las Figuras 4.9 a 4.11 se muestran los pasos descritos anteriormente en el proceso de importación del fichero TestData2. La tercera figura muestra el nombre de la variable TestData2 en MATLAB, y

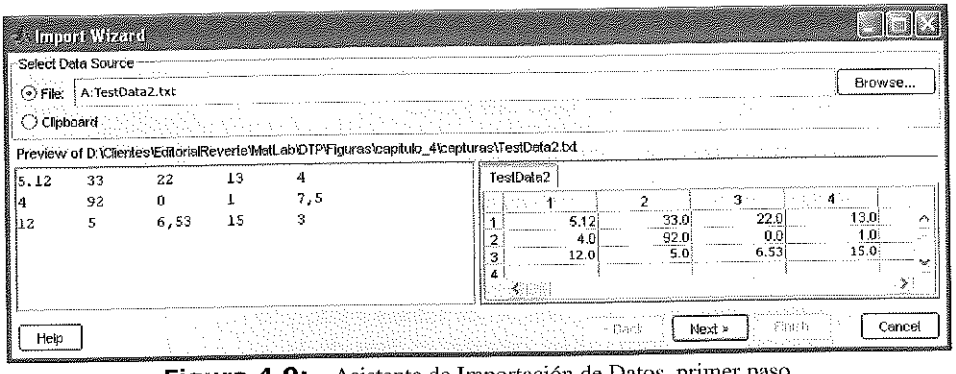


Figura 4.9: Asistente de Importación de Datos, primer paso.

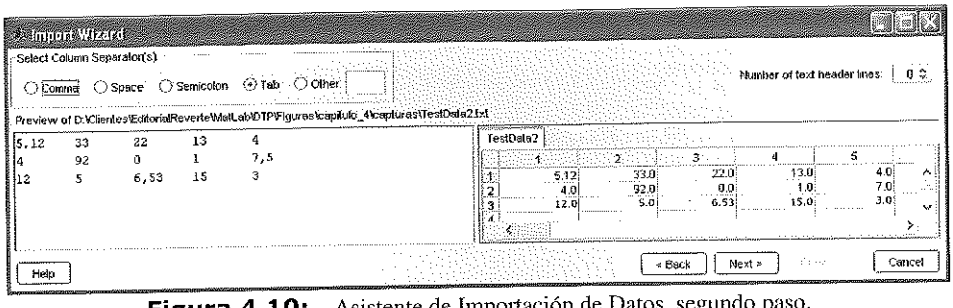


Figura 4.10: Asistente de Importación de Datos, segundo paso.

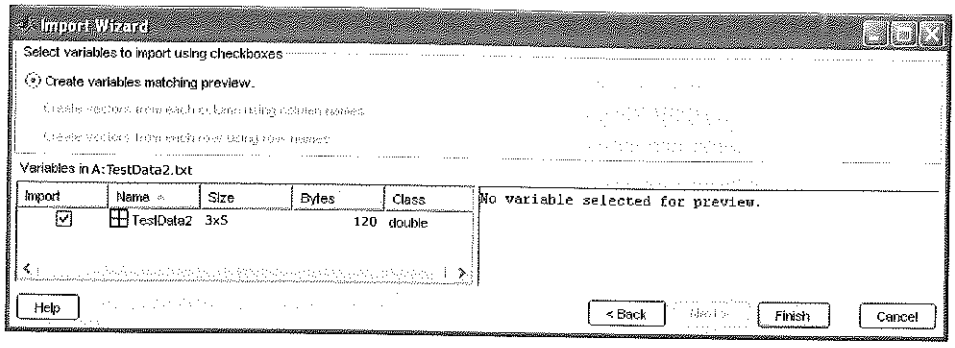


Figura 4.11: Asistente de Importación de Datos, tercer paso.

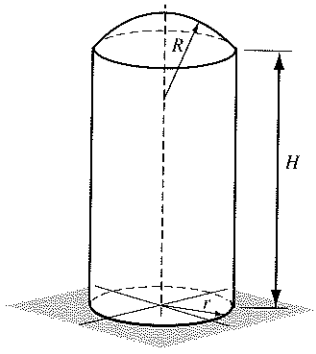
su tamaño de 3 × 5. Una vez finalizado el proceso de importación, los datos pueden verse en la Ventana de Comandos simplemente poniendo el nombre de la variable creada, tal y como se muestra a continuación:

```
>> TestData2

TestData2 =
    5.1200   33.0000   22.0000   13.0000    4.0000
    4.0000   92.0000         0    1.0000    7.5000
   12.0000    5.0000    6.5300   15.0000    3.0000
```

4.8 Ejemplo de aplicaciones con MATLAB

Problema de ejemplo 4.1: Altura y área de la superficie de un silo

Un silo con estructura cilíndrica de radio r posee una parte superior esférica de radio R. La altura de la porción cilíndrica es H. Escribir un fichero script que determine la altura H a partir de los valores r y R, y del volumen V. Además, el programa debe calcular el área de la superficie del silo. Los datos conocidos son $r = 30$ pies, $R = 45$ pies y $V = 120.000$ pies^3. Asignar estos valores directamente en la Ventana de Comandos.



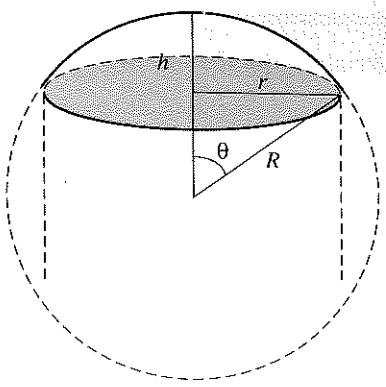
Solución

El volumen total del silo se obtiene sumando el volumen de la parte cilíndrica y el de la parte correspondiente al techo esférico. El volumen de un cilindro se calcula a partir de la expresión:

$$V_{\text{cilindro}} = \pi r^2 H$$

y el volumen del techo esférico se obtiene a partir de:

$$V_{\text{techo}} = \frac{1}{3}\pi h^2(3R - h)$$



donde $h = R - R\cos\theta = R(1 - \cos\theta)$, y θ se obtiene a partir de la expresión $\text{sen}\ \theta = \frac{r}{R}$.

Utilizando la ecuación de arriba, la altura H de la parte cilíndrica se puede expresar de la forma:

$$H = \frac{V - V_{\text{techo}}}{\pi r^2}$$

El área de la superficie del silo se calcula sumando el área de la parte esférica y el de la parte cilíndrica:

$$S = S_{\text{cilindro}} + S_{\text{techo}} = 2\pi H + 2\pi R h$$

A continuación se muestra el programa, en forma de fichero script llamado silo, que resuelve el problema propuesto:

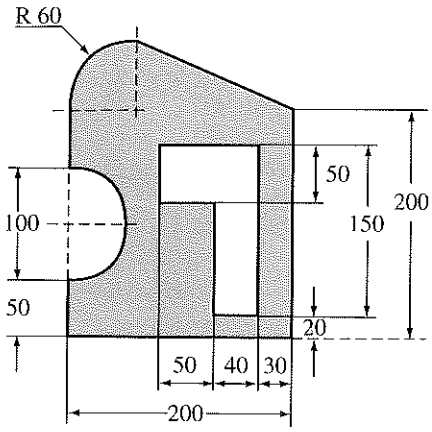
```
theta = asin(r/R);                                  % Se calcula θ.
H = R*(1-cos(theta));                               % Se calcula h.
Vtecho = pi*h^2*(3*R-h)/3;                          % Se calcula el volumen del techo.
H = (V-Vtecho)/(pi*r^2);                            % Se calcula H.
S = 2*pi*(r*H + R*h);                               % Se calcula el área de la superficie S y se visualizan los resultados con fprintf.
fprintf('La altura H es: %f pies.',H);
fprintf('\n El area de la superficie del silo es %f pies cuadrados.',S)
```

La Ventana de Comandos, después de ejecutar el fichero script, se muestra a continuación.

```
>> r = 30; R = 45; V = 200000;                      % Se asignan los valores r, R y V.
>> silo                                             % Se ejecuta el fichero script llamado silo.
La altura H es: 64.727400 pies.
El area de la superficie del silo es 15440.777753 pies cuadrados.
```

Problema de ejemplo 4.2: Centroide de un área compuesta

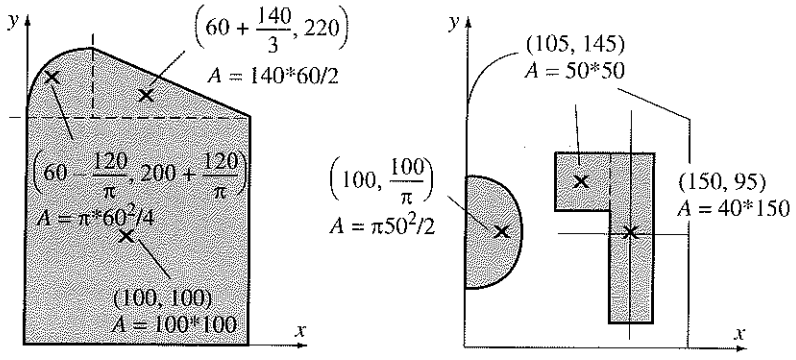
Escribir un programa, utilizando un fichero script, que calcule las coordenadas del centroide de un área compuesta como el de la figura adjunta. Un área compuesta puede ser dividida en secciones independientes cuyo centroide es conocido. El usuario necesita dividir el área en secciones y conocer las coordenadas del centroide (dos números), así como el área de cada sección (un número). Cuando se ejecute el script, éste debe pedir al usuario que introduzca los tres números como fila de una matriz. El usuario debe introducir tantas filas como secciones haya. Las secciones que representan agujeros tendrán área negativa. Como salida, el programa debe mostrar las coordenadas del centroide del área compuesta. Los datos conocidos se indican en la figura.



Dimensiones en mm.

Solución

El área se divide en seis secciones, como se muestra en la siguiente figura. El área total se calcula sumando las tres secciones de la parte izquierda y restando las tres secciones de la derecha. La localización y coordenadas del centroide de cada sección vienen marcadas en la figura, así como el área de cada sección.



Unidades: coordenadas mm, area mm^2.

Las coordenadas $\overline{X}$ e $\overline{Y}$ del centroide del área total vienen dadas por:

$\overline{X} = \frac{\Sigma A\bar{x}}{\Sigma A}$ y $\overline{Y} = \frac{\Sigma A\bar{y}}{\Sigma A}$, donde, $\bar{x}$, $\bar{y}$ y A son respectivamente las coordenadas del centroide y el área de cada sección.

A continuación se muestra el fichero script que calcula las coordenadas del centroide de un área compuesta. Este fichero se ha guardado en disco con el nombre de Centroide.

```
% Este programa calcula las coordenadas del centroide
% de un area compuesta.
clear C xs ys As
C = input('Introduzca una matriz en la que cada fila tenga tres elementos. \n En cada fila debe introducir las
   coordenadas x e y del centroide, asi como el area de la seccion.\n');
xs = C(:1)';
ys = C(:2)';
As = C(:,3)';
A = sum(As);
x = sum(As.*xs)/A;
y = sum(As.*ys)/A;
fprintf('Las coordenadas del centroide son: (%f, %F)',x,y)
```

Se crea un vector fila para la coordenada x de cada sección (primera columna de C).

Se crea un vector fila para la coordenada y de cada sección (segunda columna de C).

Se crea un vector fila para el área de cada sección (tercera columna de C).

Se calcula el área total.

Se calculan las coordenadas del centroide del área compuesta.

A continuación se muestra la Ventana de Comandos una vez ejecutado el script.

```
>> Centroide
Introduzca una matriz en la que cada fila tenga tres elementos.
En cada fila debe introducir las coordenadas x e y del centroide, asi como el area de la seccion.
[100   100   200*200
60-120/pi   200+120/pi   pi*60^2/4
60+140/3   220   140*60/2
100   100/pi   -pi*50^2/2
150   95   -40*150
105   145   -50*50]
Las coordenadas del centroide son: (85.387547, 131.211809)
>>
```

Entrada de datos en forma de matriz, donde en cada fila de C se introducen los elementos x, y y A de cada sección.

Problema de ejemplo 4.3: Divisor de voltaje

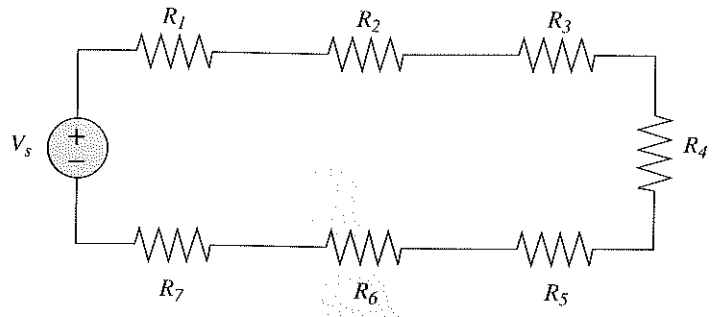
Cuando un conjunto de resistencias se conectan serie en un circuito eléctrico, el voltaje a través de cada una de ellas viene dado por la siguiente regla de división:

$$v_n = \frac{R_n}{R_{eq}} v_s$$

donde v_n es el voltaje a través de la resistencia R_n, siendo n es el número de la resistencia. $R_{eq} = \Sigma R_n$ es la resistencia equivalente, y v_s es el voltaje fuente. La potencia disipada en cada resistencia viene dada por:

$$P_n = \frac{R_n}{R_{eq}^2} v_s^2$$

La siguiente figura muestra un circuito con siete resistencias conectadas en serie.



Escribir un programa en un fichero script que calcule los voltajes de cada resistencia, así como la potencia disipada por cada una, de un circuito con resistencias conectadas en serie. Cuando el fichero se ejecute, éste debe pedir al usuario que introduzca el voltaje de la fuente y después el valor de cada una de las resistencias, en un vector. El programa debe mostrar una tabla con las resistencias en la primera columna, el voltaje a través de la resistencia en la segunda columna, y la potencia disipada por cada resistencia en la tercera columna. Seguidamente, el programa debe visualizar la intensidad de corriente a través del circuito, así como la potencia total del mismo.

Los valores conocidos de v_s y R_n son los siguientes:

$v_s = 24$ V, $R_1 = 20\ \Omega$, $R_2 = 14\ \Omega$, $R_3 = 12\ \Omega$, $R_4 = 18\ \Omega$, $R_5 = 8\ \Omega$, $R_6 = 15\ \Omega$, $R_7 = 10\ \Omega$.

Solución

El fichero script que resuelve este problema se muestra a continuación.

```
% Este programa calcula el voltaje a través de cada resistencia,
% en un circuito con resistencias conectadas en serie.
vs = input('Por favor, introduzca el voltaje de la fuente ');
Rn = input('Introduzca los valores de las resistencias en forma de vector fila \n');
Req = sum(Rn);                    Calcula la resistencia equivalente.
vn = Rn*vs/Req;                   Aplica la regla de división de voltajes.
Pn = Rn*vs^2/Req^2;               Calcula la potencia en cada resistencia.
```

```
i = vs/Req;
Ptotal = vs*i;
Tabla = [Rn', vn', Pn'];
disp('')
disp(' Resistencia Voltaje Potencia ')
disp(' (Ohmnios) (Voltios) (Vatios) ')
disp('')
disp(Tabla)
disp('')
fprintf('La intensidad de corriente del circuito es de %f Amperios.', i)
fprintf('\n La potencia total disipada en el circuito es de %f Vatios.', Ptotal)
```

Calcula la intensidad de corriente del circuito.

Calcula la potencia total del circuito.

Crea la variable `Tabla` con los vectores `Rn`, `vn` y `Pn` como columnas.

Visualiza los encabezados de las columnas y líneas vacías antes y después de ellos.

Muestra la variable `Tabla`.

La Ventana de Comandos, tras la ejecución del fichero, queda como se muestra a continuación.

```
>> DivisorVoltaje
Por favor, introduzca el voltaje de la fuente 24
Introduzca los valores de las resistencias en forma de vector fila
[20 14 12 18 8 15 10]

 Resistencia      Voltaje      Potencia
   (Ohmios)     (Voltios)     (Vatios)
     20.0000       4.9485       1.2244
     14.0000       3.4639       0.8571
     12.0000       2.9691       0.7346
     18.0000       4.4536       1.1019
      8.0000       1.9794       0.4897
     15.0000       3.7113       0.9183
     10.0000       2.4742       0.6122


La intensidad de corriente del circuito es de 0.247423 Amperios.
La potencia total disipada en el circuito es de 5.938144 Vatios.
```

Nombre del fichero script.

Voltaje de la fuente de alimentación. Valor introducido por el usuario.

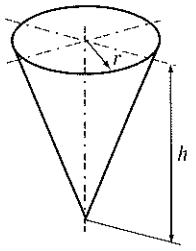
Valores de las resistencias introducidos en forma de vector.

4.9 Problemas

Nota: *Además de los problemas que se proponen a continuación, todos los problemas del Capítulo 3 se pueden resolver también utilizando ficheros script.*

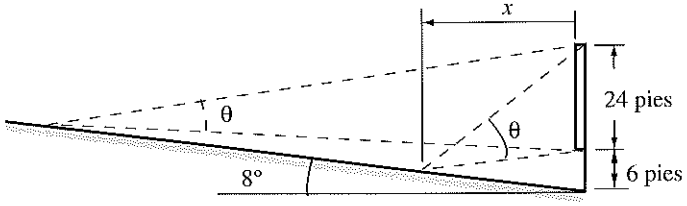
Los siguientes problemas deben ser resueltos creando un programa en un fichero script y posteriormente ejecutándolo en la Ventana de Comandos.

1. Se ha diseñado sobre papel una copa cónica que tiene un volumen de 250 cm^3. Determine el radio r de la base y el área de la superficie S de este diseño para una serie de distintos bocetos de copas que tienen de altura h de 5, 6, 7, 8 y 9 cm. El cálculo del volumen V y del área superficial vienen dados por las fórmulas:



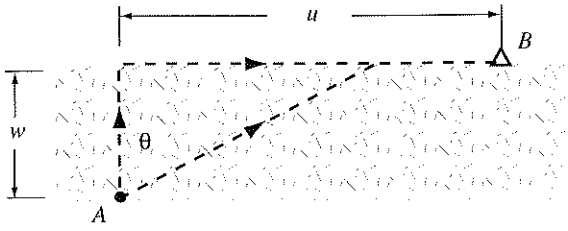
$$V = \frac{1}{3}\pi r^2 h \qquad S = \pi r\sqrt{r^2 + h^2}$$

2. En un cine, el ángulo θ a partir del cual un espectador ve la película depende de la distancia x del espectador a la pantalla. Para un cine de dimensiones como las que se muestran en la figura adjunta, calcule el ángulo θ (en grados) para los espectadores que están sentados a distancias de 30, 45, 60, 75 y 90 pies de la pantalla.



3. La población de un determinado país es de 50 millones, cantidad que se duplicará previsiblemente en 20 años. Calcule la población dentro de 5, 10 y 15 años definiendo un vector t con 3 elementos y utilizando operaciones elemento a elemento para el cálculo. El crecimiento de la población se puede modelar mediante la ecuación $P = P_0 2^{t/d}$, donde P es la población en el instante t, P_0 es la población en el instante $t = 0$ y d es el doble del tiempo.

4. Un excursionista necesita cruzar un área arenosa para poder ir del punto A a un campamento que se encuentra en el punto B. Para hacer esto puede cruzar la zona arenosa perpendicularmente al camino y a continuación andar a lo largo de él, o también puede cruzar la zona arenosa con un



ángulo θ hasta el camino, y luego caminar a lo largo del camino. El excursionista camina a una velocidad de 3,5 km/h por la arena, y a 5 km/h por el camino. Calcule el tiempo que le lleva alcanzar el campamento contemplando distintos ángulos θ de 0, 10, 20, 30, 40, 50 y 60 grados.
La distancias w y u son, respectivamente, w = 4,5 km, y u = 14 km. Escriba un programa en un fichero script que resuelva este problema. Calcule todas las variables dentro del fichero script. Visualice los resultados en una tabla de dos columnas en la cual la primera columna sea θ y la segunda columna el tiempo t correspondiente.

5. Escriba un fichero script que calcule el balance de una cuenta de ahorros a final de año, durante 10 años. La cuenta tiene un capital inicial de $1000 y un interés de 6,5% que produce beneficios anualmente. Visualice la información en una tabla.

   Para un capital inicial A, y una tasa de interés r, el balance B, después de n años, viene dado por la expresión:

$$B = A\left(1 + \frac{r}{100}\right)^n$$

6. La velocidad v y la distancia d, en función del tiempo, de un coche que tiene una velocidad constante a, vienen dadas por:

$$v(t) = at \quad \text{y} \quad d(t) = \frac{1}{2}at^2$$

   Determine v y d para cada segundo, durante 10 segundos, para un coche con una aceleración a = 1,55 m/s^2. Muestre los resultados en una tabla de tres columnas en la cual la primera sea el tiempo (s). Muestre en la segunda la distancia (m) y en la tercera la velocidad (m/s).

7. Cuando se conectan diferentes resistencias en paralelo en un circuito eléctrico, la corriente a través de cada una de estas resistencias viene dada por:

$$i_n = \frac{v_s}{R_n}$$

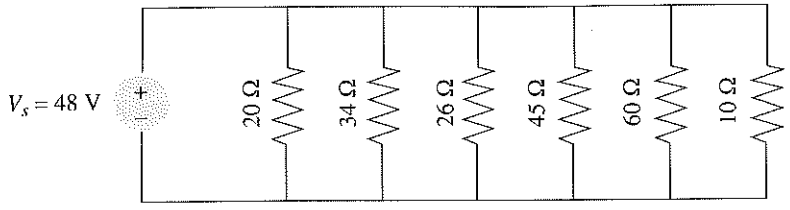
   donde i_n y R_n representan la intensidad de corriente a través de la resistencia n y su valor de resistencia propiamente dicho, siendo v_s el potencial de la fuente. La resistencia equivalente, R_{eq}, en este caso, se puede calcular a partir de la expresión:

$$\frac{1}{R_{\text{eq}}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_n}$$

   La intensidad de corriente de la fuente viene dada por: $i_s = v_s/R_{eq}$, y la potencia, P_n, disipada por cada resistencia vienen dada por: $P_n = v_s i_n$.

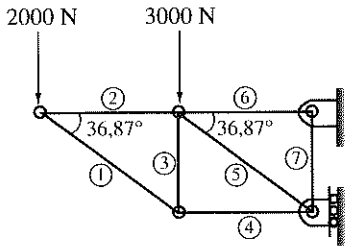
   Escriba un programa, utilizando un fichero script, que calcule la corriente que pasa por cada resistencia, así como la potencia disipada por cada una, en un circuito como el que se muestra en la figura con resistencias colocadas en paralelo. Cuando el fichero script se ejecute, éste debe pedir al usuario que introduzca el voltaje de la fuente, y después, en un vector, los valores correspondientes a las resistencias. El programa debe mostrar en una tabla las resistencias en la primera columna, la

corriente que pasa por cada una de ellas en la segunda, y la potencia que disipan en la tercera columna. Después, el programa debe mostrar también la intensidad de corriente de la fuente y la potencia total del circuito.



8. La gráfica de la función $f(x) = ax^3 + bx^2 + cx + d$ pasa por los puntos (–2, –3,4), (–0,5, 5,525), (1, 16,7) y (2,5, 70,625). Calcule las constantes a, b, c y d escribiendo para ello un sistema de ecuaciones con cuatro incógnitas, utilizando posteriormente MATLAB para resolver el sistema.

9. Cuando se llevan a cabo cálculos de estructuras es habitual trabajar con sistemas como el que se muestra en la figura adjunta, consistente en una estructura compuesta de miembros o elementos encadenados unos con otros por sus extremos, y donde lo que se trata es determinar las fuerzas que inciden sobre cada elemento. Para la estructura que se muestra en la figura adjunta, las fuerzas de los siete miembros vienen determinadas por las siguientes siete ecuaciones:



$F_1\text{sen}(36{,}87°) = -2000$

$F_1\cos(36{,}87°) + F_2 = 0$

$F_3 + F_1\text{sen}(36{,}87°) = 0$

$F_4 - F_1\cos(36{,}87°) = 0$

$-F_3 - F_5\text{sen}(36{,}87°) = 3000$

$F_6 + F_5\cos(36{,}87°) - F_2 = 0$

$F_5\text{sen}(36{,}87°) + F_7 = 0$

Escriba las ecuaciones en forma matricial y utilice MATLAB para calcular las fuerzas de los elementos de esta estructura. Una fuerza positiva implica una fuerza de tensión, mientras que una fuerza negativa implica una fuerza de compresión. Visualice los resultados en una tabla.

Capítulo 5
Gráficos bidimensionales

Los gráficos son herramientas muy utilizadas para presentar todo tipo de información; información que puede proceder de cualquier campo del conocimiento, pero especialmente de las disciplinas relacionadas con las ciencias y la ingeniería, donde MATLAB es ampliamente utilizado. Con los comandos de MATLAB se pueden crear distintos tipos de gráficos: estándares con ejes lineales, logarítmicos o semilogarítmicos, de barras y escaleras, polares, de malla y de superficies de contorno tridimensional, etc. Estos gráficos se pueden personalizar para que tengan la apariencia deseada. Así, se puede establecer el tipo, el color y el grosor de línea; se pueden añadir líneas de referencia y cuadrículas; y también títulos y comentarios. Además se pueden superponer varios gráficos sobre un mismo sistema de ejes de coordenadas, o poner varios gráficos en una misma página. Cuando un gráfico tiene varios tipos de datos, también se pueden añadir leyendas.

En este capítulo se describe cómo se utiliza MATLAB para crear y dar forma a gráficos de dos dimensiones. Los gráficos tridimensionales se abordarán más adelante, en el Capítulo 9. En la Figura 5.1 se muestra un ejemplo de gráfico bidimensional creado con MATLAB. En él se ven dos curvas que muestran la variación de la intensidad de la luz en función de la distancia. Una de las curvas se dibujó a partir de datos medidos en un experimento, mientras que la otra muestra la variación de la luz según un modelo teórico. Los ejes de la figura son líneas rectas, y para dibujar las curvas se utilizaron diferentes tipos de línea (una sólida y la otra a trazos). La curva teórica se representa mediante una línea sólida, mientras que la curva que muestra los datos experimentales es a trazos. Además, en esta última curva los puntos que representan los datos experimentales se indican con un marcador circular. Aunque aquí no se aprecie, la línea a trazos que conecta los datos experimentales es en realidad roja, y así se ve cuando se representa en la Ventana de Gráficos (Figure Window). Como se puede ver, el aspecto de la Figura 5.1 ha sido modificado; es decir, se le ha dado *formato* o se *ha formateado*. Así, contiene elementos tales como el título del gráfico, los títulos en los ejes x e y, la leyenda, marcadores y texto encerrado en una caja como nota aclaratoria.

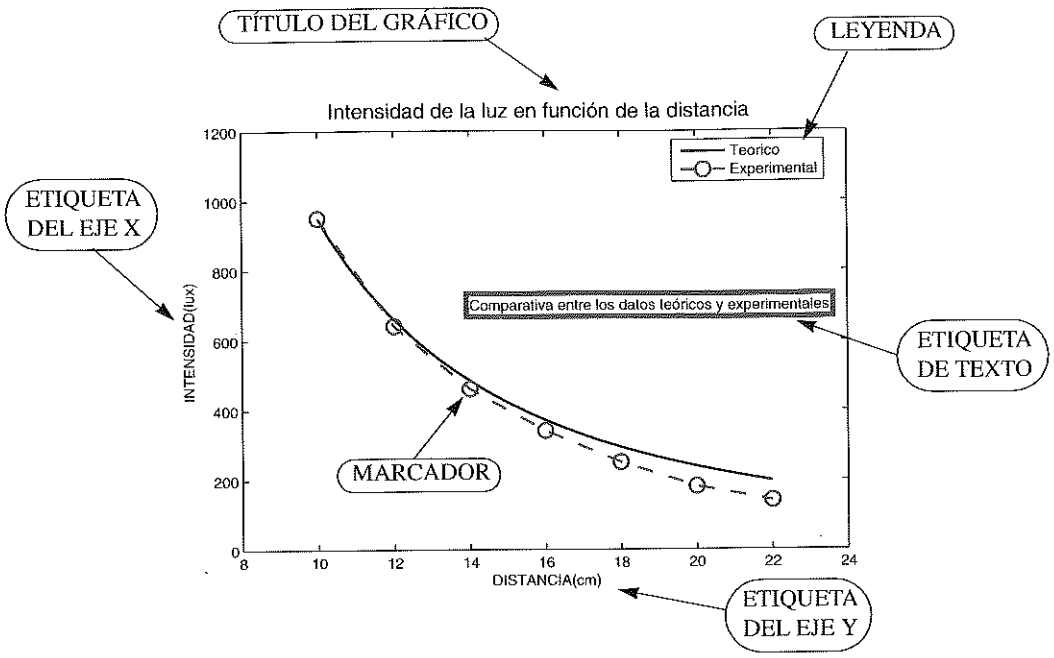
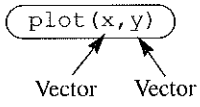


Figura 5.1: Ejemplo de un gráfico bidimensional con distintos elementos de formato.

5.1 El comando `plot`

El comando `plot` se utiliza para crear gráficos bidimensionales. La forma más sencilla de utilizar este comando es la siguiente



Los argumentos `x` e `y` son vectores (arrays unidimensionales). Ambos vectores **deben** tener el mismo número de elementos. Cuando se ejecuta el comando `plot`, el gráfico se crea en la Ventana de Gráficos. Si no se ha abierto previamente, esta ventana se abrirá automáticamente al ejecutar el comando. La represtación del gráfico que se mostrará se corresponde con una curva donde los valores de `x` serán los de la abscisa (eje horizontal) y los de `y` los de la ordenada (eje vertical). La curva se construye mediante segmentos de recta que unen los puntos cuyas coordenadas están definidas por los elementos de los vectores `x` e `y`. Estos vectores pueden tener cualquier nombre, ya que esto no afecta a la representación gráfica. Sin embargo, el vector que se introduce como primer argumento de `plot` será el que defina el eje horizontal, mientras que el segundo definirá el eje vertical.

El gráfico generado contendrá ejes con una escala lineal y un rango por defecto. Por ejemplo, si el vector `x` contiene los elementos 1, 2, 3, 5, 7, 7,5, 8, 10 y el vector `y` contiene los elementos 2, 6,5, 7, 7, 5,5, 4, 6, 8, una forma sencilla de representar `y` en función de `x` es tecleando los siguientes comandos en la Ventana de Comandos:

```
>> x = [1 2 3 5 7 7.5 8 10];
>> y = [2 6.5 7 7 5.5 4 6 8];
>> plot(x,y)
```

Una vez que se ejecuta el comando `plot`, la Ventana de Gráficos se abrirá automáticamente y mostrará una curva como la que se representa en la Figura 5.2.

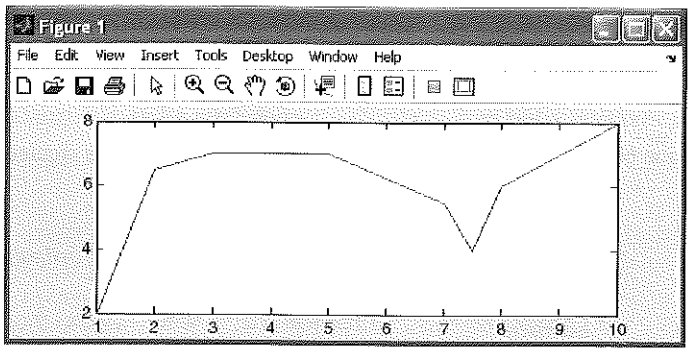
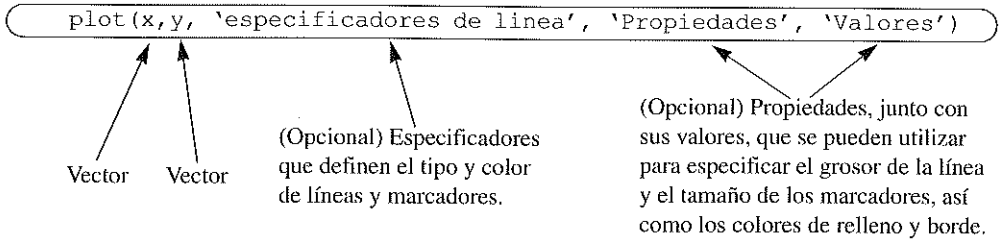


Figura 5.2: La Ventana de Gráficos mostrando un gráfico sencillo.

En la pantalla o monitor la curva aparece en color azul, que es el color de línea por defecto.

Para personalizar los gráficos, si se desea, el comando `plot` admite en su sintaxis otros argumentos que se pueden utilizar para definir el color y estilo de líneas y marcadores. Al utilizar estas opciones, el comando `plot` amplía su sintaxis:



Especificadores de línea:

Los especificadores de línea son opcionales y se pueden utilizar para definir el color y estilo de línea, así como el tipo de marcadores (en caso de que se desee incluirlos). Los especificadores de estilo de línea son:

Estilo de línea	Especificador
sólida (por defecto)	-
discontinua	--
punteada	:
rayas y puntos	-.

Para concretar el color, tenemos los siguientes especificadores:

Color de línea	Especificador
rojo	r
verde	g
azul	b
cian	c
magenta	m
amarillo	y
negro	k
blanco	w

Las posibles marcas que podemos incluir en un gráfico vienen definidas por los siguientes especificadores:

Tipo de marcador	Especificador
signo más	+
círculo	o
asterisco	*
punto	.
cuadrado	s
diamante	d
estrella de cinco puntas	p
estrella de seis puntas	h

Nota sobre el uso de especificadores:

- Los especificadores se introducen como cadenas dentro del comando `plot`.
- Dentro de la cadena, los especificadores se pueden teclear en cualquier orden.
- Los especificadores son opcionales. Esto significa que un comando puede contener uno, dos o tres especificadores, o bien ninguno.

He aquí algunos ejemplos de su uso:

`plot(x,y)`	Línea azul sólida que conecta los puntos, sin marcadores (por defecto).
`plot(x,y,'r')`	Línea roja sólida que conecta los puntos.
`plot(x,y,'--y')`	Línea amarilla discontinua que conecta los puntos.
`plot(x,y,'*')`	Puntos con marcadores de tipo asterisco (sin líneas que unan los puntos).
`plot(x,y,'g:d')`	Línea verde punteada que une puntos con marcadores, en forma de diamante.

`Propiedades` *y* `Valores`*:*

Las propiedades y los valores también forman parte de la sintaxis de `plot` vista anteriormente. Éstos son opcionales, y se utilizan para concretar el grueso de la línea, el tamaño de los marcadores, así como los colores de relleno y del borde del marcador. La propiedad se teclea como cadena, seguida por una coma y su valor correspondiente Se puede indicar más de una propiedad seguida de su valor, también separado por comas. De esta forma deben constituirse pares de propiedad y valor dentro de un mismo comando `plot`.

En la siguiente tabla se muestran propiedades y sus posibles valores:

Propiedad	Descripción	Posible valor de la propiedad
`LineWidth` o `linewidth`)	Especifica el grosor de la línea.	Un número representado en unidades de puntos (el valor por defecto es 0,5)
`MarkerSize` (o `markersize`)	Especifica el tamaño de las marcas.	Un número representado en unidades de puntos.
`MarkerEdgeColor` o `markeredgecolor`)	Especifica el color del marcador, o el color del borde de la línea para marcadores con relleno.	Especificadores de color, como los vistos en tablas anteriores, introducidos en forma de cadena.
`MarkerFaceColor` o `markerfacecolor`)	Especifica el color de relleno de los marcadores.	Especificadores de color, como los vistos en tablas anteriores, introducidos en forma de cadena.

Por ejemplo, el comando:

```
plot(x,y,'-mo','LineWidth',2,'markersize',12,
'MarkerEdgeColor','g','markerfacecolor','y')
```

crea un gráfico en el cual una línea sólida de color magenta une puntos que se representan mediante marcadores en forma de círculo. El grosor de línea es de dos puntos, y el tamaño de los círculos utilizados como marcadores es de 12 puntos. Además los marcadores (círculos) tienen bordes de color verde, y amarillo como color de relleno.

Notas sobre los especificadores de línea y sus propiedades:

Los especificadores de línea correspondientes al estilo, color de línea y el tipo de marcador pueden ser también asignados utilizando propiedades y valores. Para ello se utilizan las siguientes propiedades:

Especificador	Propiedad	Posible valor para la propiedad
Estilo de línea	`LineStyle` (o `linestyle`)	Especificador de estilo de línea, como los vistos en tablas anteriores, introducidos en forma de cadena.
Color de línea	`Color` (o `color`)	Especificadores de color, como los vistos en tablas anteriores, introducidos en forma de cadena.
Marcador	`Marker` (o `marker`)	Especificador de marcador, como los vistos en tablas anteriores, introducido en forma de cadena.

Como sucede con todos los comandos MATLAB vistos hasta ahora, el comando `plot` puede ser ejecutado directamente desde la Ventana de Comandos o desde un programa escrito en un fichero script. También se puede utilizar en un fichero de funciones (esto se verá en el Capítulo 6). Se recuerda que antes de ejecutar el comando `plot`, los vectores `x` e `y` deben existir y tener elementos asignados. Esto se lleva a cabo mediante los mecanismos ya explicados en capítulos anteriores, introduciendo los

valores directamente mediante los comandos pertinentes, o bien asignándolos a través de operadores y/o como resultados de operaciones aritméticas. Los siguientes dos subapartados explican algunos ejemplos de creación de gráficos sencillos.

5.1.1 *Generación de gráficos a partir de datos [dados]*

En este caso, primero se crean vectores con los datos [dados], y luego se utilizan estos vectores en el comando `plot` para generar el gráfico. He aquí un ejemplo de representación gráfica de los datos correspondientes a ventas de una compañía desde el año 1988 a 1994.

Año	1988	1989	1990	1991	1992	1993	1994
Ventas (millones)	8	12	20	22	18	24	27

Para representar estos datos, el listado de años se asigna a un vector (llamado `an`) y las ventas a un segundo vector (llamado `ven`). A continuación se muestra la Ventana de Comandos donde se crean los vectores y se utiliza el comando `plot`:

```
>> an = [1988:1:1994];
>> ven = [8 12 20 22 18 24 27];
>> plot(an,ven,'--r*','linewidth',2,'markersize',12)
>>
```

Especificadores de línea: línea discontinua con marcadores rojos en forma de asterisco.

Propiedad y valor: el ancho de línea es de 2 puntos y el tamaño del marcador (asteriscos) es de 12 puntos.

La Figura 5.3 muestra la Ventana de Gráficos que se abre tras la ejecución del comando `plot`. El gráfico aparece en la pantalla en color rojo.

5.1.2 *Generación de gráficos a partir de funciones*

En muchos casos es necesario representar un gráfico a partir de una determinada función matemática. Esto se puede hacer utilizando los comandos `plot` y `fplot`. La utilización de `plot` para este cometido se explicará a continuación, mientras que el comando `fplot` se verá en la próxima sección.

Para representar una función del tipo $y = f(x)$ con el comando `plot`, el usuario necesita crear primero un vector con los valores de x del dominio de la función que se va a representar. Seguidamente deberá crear un vector y con los correspondientes valores de $f(x)$, utilizando para ello operaciones elemento a elemento (como las vistas en el Capítulo 3). Una vez creados ambos vectores, se puede utilizar el comando `plot` para representar la función.

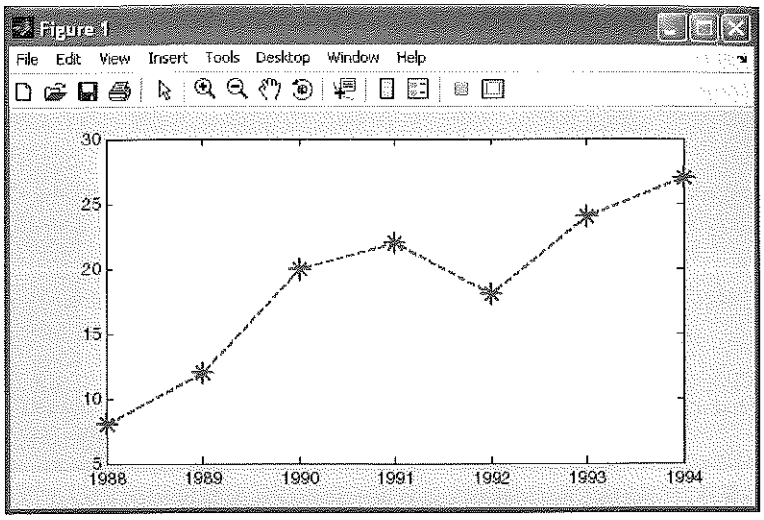


Figura 5.3: Ventana de Gráficos con la representación de los datos de ventas.

Como ejemplo, vamos a utilizar el comando `plot` para representar gráficamente la función $y = 3{,}5^{-0{,}5x}\cos(6x)$ en el intervalo $-2 \leq x \leq 4$. Un programa que genera el gráfico de esta función se muestra en el siguiente fichero script.

```
% Fichero script que genera el grafico de
% la funcion: 3.5.^(-0.5*x).*cos(6*x)
x = [-2:0.01:4];          % Se crea el vector x con el dominio de la función.
y = 3.5.^(-0.5*x).*cos(6*x); % Se crea el vector con los valores de y para cada uno de los elementos x.
plot(x,y)                 % Se dibuja y como función de x.
```

Una vez que se ejecuta el fichero, en la Ventana de Gráficos aparece el gráfico de la función deseada, tal y como se muestra en la Figura 5.4.

Tal y como se explicó, MATLAB representa las funciones mediante segmentos de líneas rectas que conectan cada uno de los puntos que resultan de la función para cada uno de los valores del dominio. Para obtener una resolución adecuada en la representación gráfica, los elementos del vector `x` deben ser apropiados. Cuánto más rápidamente varíe una función, menor deberá ser el espaciado entre los valores del dominio. En el ejemplo anterior, el valor del espaciado o distancia entre los elementos es 0,01, y produce el gráfico que se muestra en la Figura 5.4. Sin embargo, si representamos esa función con el mismo dominio pero con un espaciado mucho mayor (por ejemplo de 0,3), la gráfica que se obtiene, mostrada en la Figura 5.5, aparece distorsionada. Se trata de la misma función, pero con menor resolución. Observe también que esta figura no se muestra en la Ventana de Gráficos, sino que está copiada desde esta ventana. Para copiarla y pegarla en otras aplicaciones se utiliza el menú **Edit** (Edición) y se selecciona **Copy Figure** (Copiar Figura).

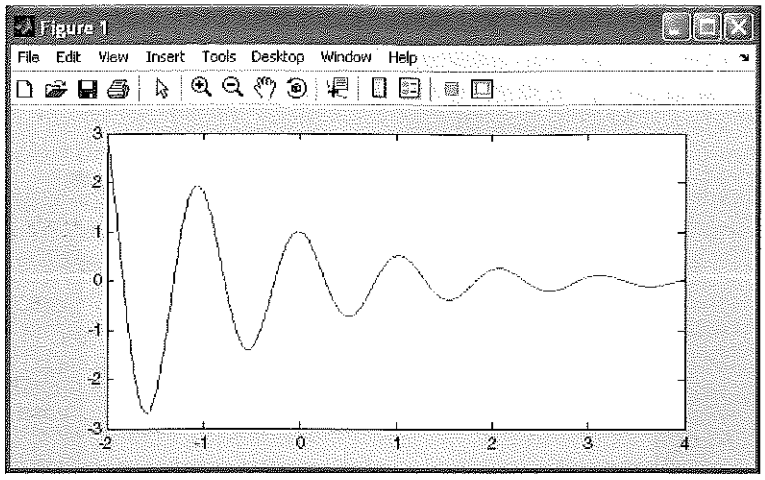


Figura 5.4: Ventana de Gráficos con la representación de la función $y = 3{,}5^{-0{,}5x}\cos(6x)$.

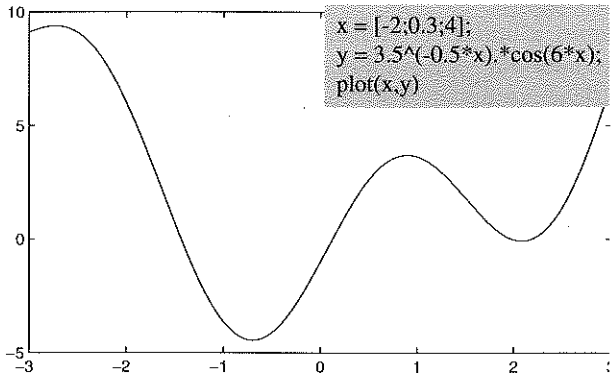


Figura 5.5: Representación gráfica de la función $y = 3{,}5^{-0{,}5x}\cos(6x)$ con un espaciado mayor.

5.2 El comando `fplot`

El comando `fplot` representa gráficamente una función de la forma $y = f(x)$ entre unos límites especificados por el usuario. El comando tiene la siguiente sintaxis:

```
fplot ('funcion', limites, especificadores de linea)
```

- `'funcion'`: Función que se desea representar.
- `limites`: El dominio de x y, opcionalmente, los límites del eje y.
- `especificadores de linea`: Especificadores que definen el tipo y color de línea, así como los marcadores utilizados en la representación.

`funcion`: La función se puede teclear directamente como cadena dentro del comando. Por ejemplo, si la función que se quiere representar es $f(x) = 8x^2 + 5\cos(x)$, esta función se puede introducir en forma de

cadena como: '8*x^2+5*cos(x)'. La función también puede incluir funciones predefinidas de MATLAB, así como otras funciones creadas por el usuario (esto se verá en el Capítulo 6).

- La función que se va a representar se puede construir dentro de la cadena utilizando cualquier letra como variable. Por ejemplo, la función anterior se puede teclear como '8*z^2+5*cos(z)' o como '8*t^2+5*cos(t)'.
- La función no puede incluir variables previamente definidas. Por ejemplo, en la función de arriba no es posible asignar 8 a una variable y después utilizarla en la función dentro del comando `fplot`.

`limites`: Los límites se especifican mediante un vector de dos elementos [`xmin`, `xmax`] que define el dominio de la variable `x`, o también mediante un vector de cuatro elementos [`xmin`, `xmax`, `ymin`, `ymax`] que define los dominios para los ejes `x` e `y`, de dos en dos respectivamente.

`especificaciones de linea`: Funcionan igual que en el comando `plot`. Como ejemplo, para representar una función del tipo $y = x^2 + 4\text{sen}(2x) - 1$ para $-3 \leq x \leq 3$, se puede introducir el siguiente comando en la Ventana de Comandos:

```
>> fplot ('x^2+4*sin(2*x)-1',[-3 3])
```

La Figura 5.6 muestra el resultado final que se genera al teclear el comando anterior.

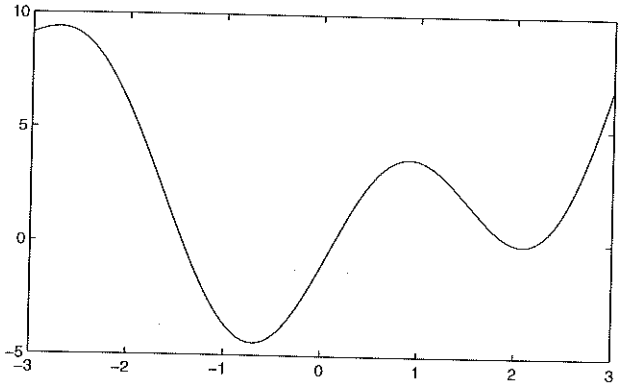


Figura 5.6: Representación gráfica de la función $y = x^2 + 4\text{sen}(2x) - 1$ para $-3 \leq x \leq 3$.

5.3 Representación gráfica de varias funciones a la vez

A veces resulta interesante representar varias funciones a la vez, tal y como se vio en la Figura 5.1. La representación de varias funciones se puede realizar de tres formas diferentes. Una de ellas consiste en utilizar el comando `plot`, otra en utilizar los comandos `hold on` y `hold off`, y la tercera consiste en utilizar el comando `line`.

5.3.1 *Utilización del comando* `plot`

Para representar más de un gráfico con este comando es necesario ampliar su sintaxis, tecleando las funciones que se van a representar como pares de vectores, de la forma:

```
plot(x, y, v, u, t, h)
```

Este comando crearía tres funciones: y frente a x, v frente a u y h frente a t, todas ellas en la misma región gráfica de la Ventana de Gráficos. Para ello, los vectores deben ser de la misma longitud. MATLAB dibuja automáticamente las gráficas en distintos colores para que éstas se puedan identificar más fácilmente. En cualquier caso, es posible añadir además especificadores de línea para cada par de vectores o para cada función. Por ejemplo, el comando:

```
plot(x, y, '-b', u, v, '--r', t, h, 'g:')
```

representaría y frente a x con una línea sólida de color azul, v frente a u con una línea a trazos discontinuos de color rojo, y h frente a t con una línea punteada verde.

Problema de ejemplo 5.1: Representación gráfica de una función y sus derivadas

Dibujar la función $y = 3x^3 - 26x + 10$, así como su primera y segunda derivadas, en el intervalo $-2 \leq x \leq 4$. Todas las funciones deben representarse juntas en el mismo gráfico.

Solución

La primera derivada de la función es $y' = 9x^2 - 26$.
La segunda derivada es $y'' = 18x$
A continuación se escribe un fichero script que crea un vector x y calcula los valores y, y' e y'':

```
x = [-2:0.01:4];
y = 3*x.^3-26*x+6;
yd = 9*x.^2-26;
ydd = 18*x;
plot(x,y, '-b',x,yd,'--r',x,ydd,':k')
```

Se crea el vector x con el dominio de la función.
Se crea un vector y con los valores de la función en cada punto x.
Se crea un vector yd con los valores de la primera derivada en cada punto x.
Se crea un vector ydd con los valores de la segunda derivada en cada punto x.
Se crean los gráficos y frente a x, yd frente a x, e ydd frente a x, en la Ventana de Gráficos.

La ejecución de este programa produce la salida gráfica que se muestra en la Figura 5.7.

5.3.2 *Utilización de los comandos* `hold on` *y* `hold off`

La forma de representar varias funciones en un mismo gráfico con estos comandos es utilizar primero el comando `plot` para representar la primera función, y luego introducir el comando `hold on`. Este comando mantiene la Ventana de Gráficos con el primer gráfico abierto, conservando los mismos ejes y el formato establecido (ver Sección 5.4). Una vez introducido este comando se proceden a ejecutar tantos comandos `plot` como se quieran. Finalmente se introduce o ejecuta el comando `hold off` para decirle al sistema que no se desean más representaciones sobre la misma región gráfica, eliminando las posibles propiedades de ejes y formato que se hubieran introducido. El comando `plot` vuelve al estado por defecto.

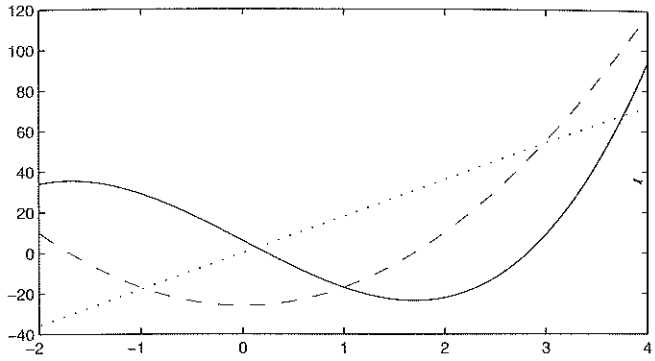


Figura 5.7: Representación gráfica de la función $y = 3x^3 - 26x + 10$, junto con su primera y segunda derivadas.

Como ejemplo se verá la solución al Problema de ejemplo 5.1 utilizando los comandos `hold on` y `hold off`, tal y como se muestra en el siguiente fichero script:

```
x = [-2:0.01:4];
y = 3*x.^3-26*x+6;
yd = 9*x.^2-26;
ydd = 18*x;
plot (x,y,'-b')
hold on
plot (x,yd,'--r')
plot (x,ydd,':k')
hold off
```

Se crea la primera gráfica.

Se añaden dos gráficas más.

5.3.3 *Utilización del comando* `line`

Con el comando `line` se pueden añadir curvas (líneas) adicionales a un gráfico que ya existe. La sintaxis del comando es la siguiente:

```
line(x,y,Propiedades,Valores)
```

(Opcional) Las propiedades, junto con sus valores, se pueden utilizar para especificar el estilo, el color y el grueso de la línea, así como el tipo, el tamaño y los colores de borde y relleno de los marcadores.

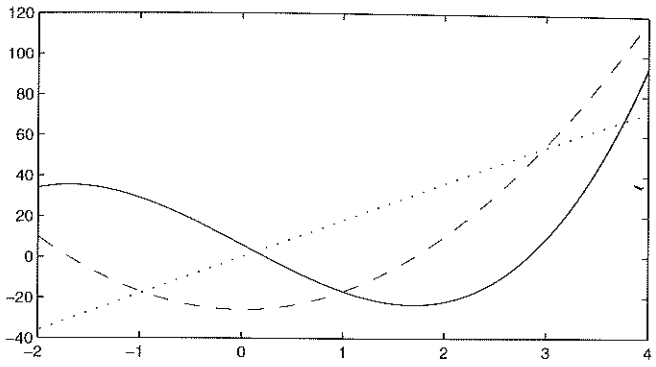


Figura 5.7: Representación gráfica de la función $y = 3x^3 - 26x + 10$, junto con su primera y segunda derivadas.

Como ejemplo se verá la solución al Problema de ejemplo 5.1 utilizando los comandos `hold on` y `hold off`, tal y como se muestra en el siguiente fichero script:

```
x = [-2:0.01:4];
y = 3*x.^3-26*x+6;
yd = 9*x.^2-26;
ydd = 18*x;
plot (x,y,'-b')
hold on
plot (x,yd,'--r')
plot (x,ydd,':k')
hold off
```

Se crea la primera gráfica.

Se añaden dos gráficas más.

5.3.3 *Utilización del comando* `line`

Con el comando `line` se pueden añadir curvas (líneas) adicionales a un gráfico que ya existe. La sintaxis del comando es la siguiente:

```
line(x,y,Propiedades,Valores)
```

(Opcional) Las propiedades, junto con sus valores, se pueden utilizar para especificar el estilo, el color y el grueso de la línea, así como el tipo, el tamaño y los colores de borde y relleno de los marcadores.

La sintaxis del comando `line` es muy parecida a la del comando `plot` (ver Sección 5.1), aunque este comando no tiene especificadores de línea. El estilo, el color y los marcadores utilizados se deben especificar mediante las ya conocidas propiedades y sus correspondientes valores. Estas propiedades son opcionales; si no se introduce ninguna, MATLAT asigna propiedades por defecto. Por ejemplo, el comando:

```
line(x,y,'linestyle','--','color','r','marker','o')
```

crearía, en un gráfico existente, una nueva curva discontinua con marcas circulares.

La diferencia fundamental entre `plot` y `line` es que el comando `plot` crea un nuevo gráfico cuando se ejecuta, mientras que el comando `line` simplemente añade una nueva curva a uno que ya existe previamente. Para crear una representación de varias funciones, la primera se crea con un comando `plot`, y seguidamente se introducen tantos comandos `line` como funciones adicionales se desee (si se introduce un comando `line` antes que uno `plot` el sistema dará un error).

La solución al Problema de ejemplo 5.1, representado en la Figura 5.7, puede también escribirse utilizando `plot` y sucesivos comandos `line`, como se muestra en el siguiente fichero script:

```
x = [-2:0.01:4]
y = 3*x.^3-26*x+6;
yd = 9*x.^2-26;
ydd = 18*x;
plot(x,y,'LineStyle','-','color','b');
line(x,yd,'LineStyle','--','color','r');
line(x,ydd,'LineStyle',':','color','k');
```

5.4 Formateado de una representación gráfica

Los comandos `plot` y `fplot` crean representaciones gráficas de apariencia muy sencilla. Habitualmente es necesario formatearlas para que tengan un determinado aspecto y para añadirles información propia del gráfico como, por ejemplo, el título del gráfico, etiquetas y escalas personalizadas en los ejes de coordenadas, leyendas, cuadrículas, etiquetas de texto, etc.

Los gráficos se pueden formatear o bien utilizando comandos de MATLAB a continuación de los comandos `plot` o `fplot`, o bien utilizando el editor de gráficos de la Ventana de Gráficos. El primer método es útil cuando los comandos `plot` o `fplot` son parte de un programa (fichero script). Cuando un programa incluye comandos de formato, cada vez que se ejecuta crea un gráfico con formato. Por otra parte, el formato que se aplica mediante el editor de gráficos de la Ventana de Gráficos sólo se mantiene para un gráfico en concreto, y se tendrá que volver a aplicar la próxima vez que se cree el gráfico.

5.4.1 *Formateado de una representación gráfica mediante comandos*

Los comandos de formato se introducen después de aquellos que crean o visualizan gráficos: `plot` y `fplot`. Existen varios comandos de este tipo, entre ellos se encuentran:

Los comandos `xlabel` *e* `ylabel`*:*

Esto comandos sirven para poner un título, en forma de texto, a los ejes. En realidad definen etiquetas que se situarán cerca de cada eje, `xlabel` para el eje *x* e `ylabel` para eje *y*. Su sintaxis es:

```
xlabel('texto');
ylabel('texto');
```

El comando `title`*:*

Este comando añade un título (principal) al gráfico, en la parte superior del mismo. Su sintaxis es:

```
title('texto')
```

El comando `text`*:*

Este comando permite situar una etiqueta de texto dentro del gráfico. El comando admite dos variantes:

```
text(x,y,'texto')
gtext('texto')
```

El comando `text` coloca el texto en el gráfico de manera que el primer carácter se sitúe en el punto con coordenadas `x` e `y` (según los ejes del gráfico). En cambio, el comando `gtext` coloca el texto en la posición especificada por el usuario. Cuando se ejecuta este comando, se abre la Ventana de Gráficos y el usuario especifica la posición pulsando con el ratón en el punto deseado.

El comando `legend`*:*

Este comando coloca una leyenda en la representación gráfica. Las leyendas incluyen una muestra del tipo de línea de cada función que se representa y una etiqueta especificada por el usuario, que permite indicar a qué corresponde cada muestra. La sintaxis de este comando es la siguiente:

```
legend('cadena1','cadena2',...,posicion)
```

Las cadenas son las etiquetas que se colocan junto a las muestras de línea, y su orden debe corresponderse con el orden en el cual se han introducido las funciones. La variable `posicion` es un número opcional que especifica el sitio en el que se situará la leyenda dentro del gráfico. Los valores posibles son:

`posicion = -1`	Sitúa la leyenda fuera de los límites establecidos por los ejes del gráfico, en el lado derecho.
`posicion = 0`	Sitúa la leyenda dentro de los límites establecidos por los ejes del gráfico en una posición que interfiera lo menos posible con el gráfico.
`posicion = 1`	Sitúa la leyenda en la esquina superior derecha del gráfico (opción por defecto).
`posicion = 2`	Sitúa la leyenda en la esquina superior izquierda del gráfico.
`posicion = 3`	Sitúa la leyenda en la esquina inferior izquierda del gráfico.
`posicion = 4`	Sitúa la leyenda en la esquina inferior derecha del gráfico.

Formateado del texto introducido con los comandos `xlabel`, `ylabel`, `title`, `text` *y* `legend`:

Los textos que se insertan mediante comandos, y que luego serán visualizados cuando se ejecuten, pueden ser formateados. El formateado permite definir la fuente y el tamaño, la posición (subíndice, superíndice), el estilo (itálica, negrita, etc.) y el color de los caracteres, así como el color de fondo y muchos otros detalles de la presentación. Más abajo se describen algunos de los formatos más habituales. En la Ventana de Ayuda, en los menús "Text" y "Text Properties", se puede encontrar una explicación detallada de todas las posibilidades que ofrece MATLAB para formatear texto. Un texto se puede formatear bien introduciendo modificadores dentro de la cadena, o bien añadiendo al comando `Propiedades` y `Valores` opcionales a continuación de las cadenas.

Los modificadores son caracteres que se insertan dentro de la cadena. Algunos de ellos son los siguientes::

Modificador	Efecto conseguido sobre el texto
`\bf`	Negrita.
`\it`	Itálica.
`\rm`	Normal.
`\fontname{fuente}`	Cambiar a la fuente de letra especificada.
`\fontsize{tamaño}`	Cambiar el tamaño de la fuente de letra.

El formato se aplica al texto que se encuentra entre el modificador y el final de la cadena. Sin embargo, también es posible aplicar formato a una parte de la cadena; para ello hay que teclear el modificador y a continuación el texto que será modificado entre llaves { }.

Subíndice y superíndice:

Un carácter puede ser visualizado como subíndice o superíndice tecleando _ (carácter de subrayado) o ^ (carácter circunflejo o elevado a), respectivamente. Para visualizar varios caracteres consecutivos de esta forma, sólo hay que introducir el texto entre llaves { } a continuación del carácter _ o ^.

Caracteres griegos:

Los caracteres griegos se pueden introducir en el texto tecleando `\nombre_letra` dentro de la cadena. Si se desea que estos caracteres aparezcan en minúsculas el nombre de la letra griega deberá teclearse también en minúsculas (todos los caracteres que la componen). Para visualizar una letra griega en mayúsculas, la letra griega deberá teclearse también con el primer carácter en mayúsculas. He aquí algunos ejemplos de estas combinaciones:

Caracteres a introducir en la cadena	Letra griega correspondiente	Caracteres a introducir en la cadena	Letra griega correspondiente
`\alpha`	α	`\Phi`	Φ
`\beta`	β	`\Delta`	Δ
`\gamma`	γ	`\Gamma`	Γ
`\theta`	θ	`\Lambda`	Λ
`\pi`	π	`\Omega`	Ω
`\sigma`	σ	`\Sigma`	Σ

Como se explicó anteriormente, también es posible dar formato al texto dentro de la cadena para comandos gráficos a partir de parámetros opcionales del tipo propiedades y valores que se introducen después de la cadena a la que afecta. La sintaxis es en general la misma que para los comandos `xlabel, ylabel, title` y `text` vistos anteriormente. Veamos no obstante la sintaxis a partir del comando `text`:

```
text(x,y,'Texto', Propiedades, Valores)
```

Propiedades serán cadenas predefinidas que irán seguidas por sus correspondientes valores, tal y como vimos en secciones anteriores para casos similares. Algunas de las propiedades posibles, junto con sus valores permitidos, son las siguientes:

Propiedad	Descripción	Valores posibles
`Rotation`	Especifica la orientación del texto.	Escalar (grados). Valor por defecto: 0
`FontAngle`	Permite cambiar entre caracteres en itálica o normales.	`normal, italic`. Valor por defecto: `normal`.
`FontName`	Especifica la fuente de letra del texto.	Nombres de fuente disponible en el sistema.
`FontSize`	Especifica el tamaño de la fuente.	Escalar (puntos). Valor por defecto: 10
`FontWeight`	Especifica el ancho de los caracteres.	`light, normal, bold`. Valor por defecto: `normal`.
`Color`	Especifica el color del texto.	Especificadores de color (véase Sección 5.1). Valor por defecto: ninguno.
`BackgroundColor`	Especifica el color de fondo (área rectangular).	Especificadores de color (véase Sección 5.1). Valor por defecto: ninguno.
`EdgeColor`	Especifica el color del borde de una caja rectangular alrededor del texto.	Especificadores de color (véase Sección 5.1). Valor por defecto: ninguno.
`LineWidth`	Especifica el grosor del borde de una caja rectangular alrededor del texto.	Escalares (puntos). Valor por defecto: 0.5

El comando `axis`:

Cuando el comando `plot(x,y)` se ejecuta, MATLAB crea los ejes correspondientes para la representación gráfica, basándose en los valores máximo y mínimo de los valores posibles que toman `x` e `y`. El comando `axis` permite cambiar el rango de los ejes, así como su apariencia. A veces ciertas gráficas de funciones son más susceptibles de ser representadas en rangos mucho mayores a los establecidos por defecto, mejorando así su presentación final. A continuación se muestran algunos de los posibles formatos que acepta el comando `axis` para el cometido enunciado anteriormente:

`axis([xmin,xmax,ymin,ymax])` Establece los límites de ambos ejes, `x` e `y`, entre los valores máximos y mínimos a partir de los valores `xmin, xmax, ymin, ymax`.

`axis equal` Establece la misma escala en ambos ejes.

`axis square` Establece la región de los ejes en un cuadrado.

`axis tight` Establece los límites de los ejes en función del rango de los datos.

El comando `grid`**:**

`grid on` Añade una cuadrícula a la representación gráfica.
`grid off` Elimina la cuadrícula de la representación gráfica.

Un ejemplo de gráfico formateado con los comandos vistos anteriormente es el que se muestra en la Figura 5.1. Este gráfico se generó mediante el siguiente fichero script:

```
x = [10:0.1:22];
y = 95000./x.^2;
xd = [10:2:22];
yd = [950 640 460 340 250 180 140];
plot(x,y,'-','LineWidth',1.0)
xlabel('DISTANCIA (cm)');
ylabel('INTENSIDAD (lux)');
title('\fontname{Arial} Intensidad de la Luz en Funcion de la Distancia','FontSize',14)
axis([8 24 0 1200])
text(14,700,'Comparativa entre los datos teoricos y experimentales.','EdgeColor','r','LineWidth',2)
hold on
plot(xd,yd,'ro--','linewidth',1.0,'markersize',10)
legend('Teorico','Experimento',0)
hold off
```

Texto con formato dentro del comando `title`.

Texto con formato dentro del comando `text`.

5.4.2 *Formateado de una representación gráfica mediante el editor gráfico*

La apariencia de un gráfico generado en la Ventana de Gráficos se puede modificar de forma interactiva pulsando con el ratón sobre el gráfico y/o utilizando los menús. La Figura 5.8 muestra la Ventana

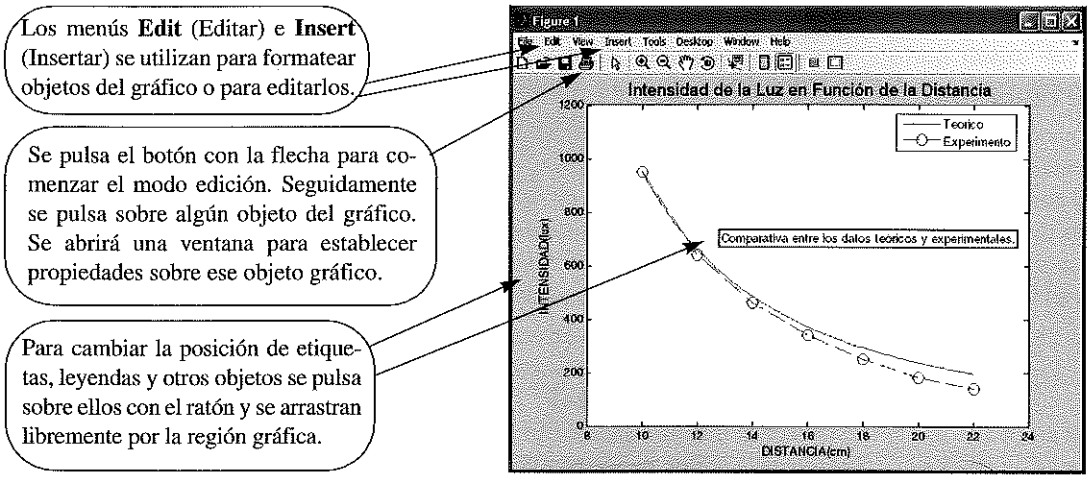


Figura 5.8: Formateado de un gráfico mediante el editor gráfico.

de Gráficos con el gráfico generado en la Figura 5.1. El editor gráfico se puede utilizar para añadir elementos nuevos o componentes en una representación gráfica, o para modificar el formato de los elementos previamente introducidos mediante comandos.

5.5 Gráficos con ejes logarítmicos

Algunas veces es necesario realizar representaciones gráficas utilizando uno o ambos ejes con escala logarítmica (log). Las escalas logarítmicas proporcionan mecanismos para representar datos con un amplio rango de valores. Además, proporcionan una forma de identificar ciertas características de los datos y posibles relaciones matemáticas adecuadas par construir modelos a partir de esos datos (véase Sección 8.2.2).

Los comandos utilizados en MATLAB para representar gráficos con ejes logarítmicos son:

`semilogy(x,y)`	Representa `y` frente a `x` con escala logarítmica (en base 10) para el eje `y` y escala lineal para el eje `x`.
`semilogx(x,y)`	Representa `y` frente a `x` con escala logarítmica (en base 10) para el eje `x` y escala lineal para el eje `y`.
`loglog(x,y)`	Representa `y` frente a `x` con escala logarítmica (en base 10) para ambos ejes.

Al igual que antes, se pueden añadir especificadores, `Propiedades` y `Valores` a los comandos (de forma opcional), de manera similar a como se estudió para el caso de `plot`. La Figura 5.9 muestra distintos ejemplos de representación gráfica de la función $y = 2^{(-0,2x + 10)}$ para $0,1 \leq x \leq 60$, utilizando distintas combinaciones de escalas logarítmicas y lineales.

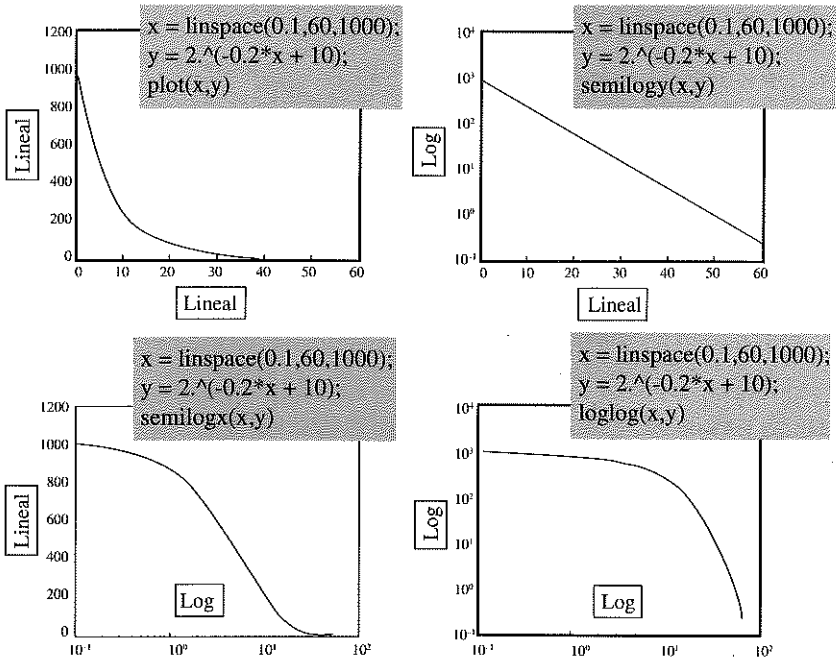


Figura 5.9: Representación gráfica de la función $y = 2^{(-0,2x + 10)}$ con combinaciones de escalas logarítmicas y lineales.

Notas sobre los gráficos con escalas logarítmicas:

- El número cero no se puede representar gráficamente en una escala logarítmica, ya que el logaritmo de cero no está definido.
- Los números negativos no se pueden representar gráficamente en una escala logarítmica, ya que el logaritmo de un número negativo no está definido.

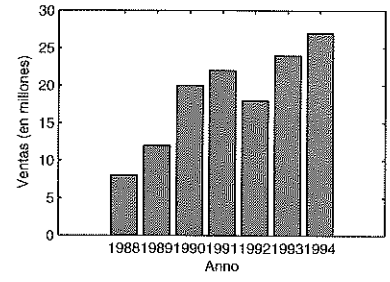
5.6 Representación de gráficos especiales

Todos los gráficos que han sido tratados hasta ahora en este capítulo son gráficos de línea en los cuales los puntos estaban conectados por líneas. En otras situaciones es necesario realizar representaciones gráficas con otro tipo de geometría, de forma que éstas sean más efectivas para el tipo de dato que se representa. MATLAB ofrece distintos comandos destinados a la representación de otros tipos de gráficos, como son los gráficos de barras, de escaleras, de tarta (o circulares), de tallo (o de líneas verticales), etc. A continuación se mostrarán algunos de estos comandos, aunque, como siempre, se puede obtener un listado más exhaustivo consultando la ayuda de MATLAB, eligiendo "Functions by Category", seguidamente "Graphics" y después "Basic Plots and Graphs" o "Specialized Plotting".

Los gráficos que se presentan a continuación se han generado a partir de los datos de ventas de la Sección 5.1.1.

Gráfico de barras verticales

Formato de la función:
`bar(x,y)`

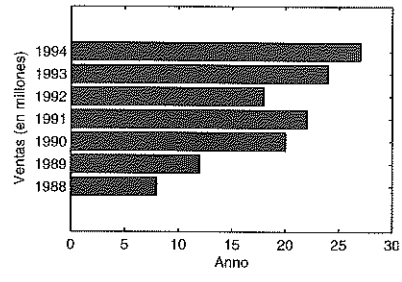



```
yr = [1988:1994];
sle = [8 12 20 22 18 24 27];
bar(yr,sle,'r')
xlabel('Anno')
ylabel('Ventas
  (en millones)')
```

Barras en color rojo

Gráfico de barras horizontales

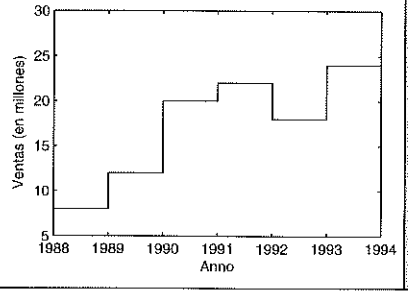
Formato de la función:
`barh(x,y)`




```
yr = [1988:1994];
sle = [8 12 20 22 18 24 27];
barh(yr,sle,'r')
xlabel('Ventas (en millones')
ylabel('Anno)')
```

Gráfico de escaleras

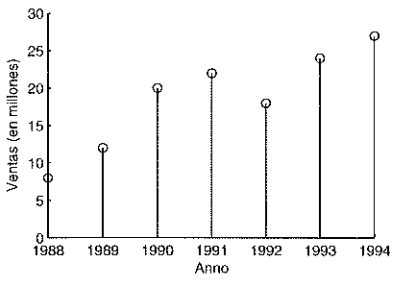
Formato de la función:
`stairs(x,y)`




```
yr = [1988:1994];
sle = [8 12 20 22 18 24 27];
stairs(yr,sle)
xlabel('Anno')
ylabel('Ventas (en millones)')
```

Gráfico de tallo o líneas verticales

Formato de la función:
`stem(x,y)`



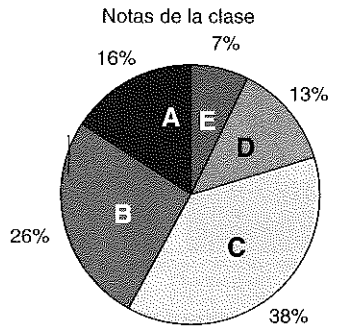

```
yr = [1988:1994];
sle = [8 12 20 22 18 24 27];
stem(yr,sle)
xlabel('Anno')
ylabel('Ventas (en millones)')
```

Los gráficos de tarta o circulares son útiles para la visualización de tamaños relativos de cantidades diferentes pero relacionadas. Por ejemplo, la siguiente tabla muestra las notas (en letras: A, B, C, D y E) obtenidas por una clase. Estos datos se utilizan para crear un gráfico de tarta.

Nota	A	B	C	D	E
Número de alumnos	11	18	26	9	5

Gráfico de tarta o circular

Formato de la función:
`pie(x)`




```
grd = [11 18 26 9 5];
pie(grd)
title('Notas de la clase')
```

Matlab dibuja secciones de diferente color. Las letras (clases) se añadieron utilizando el editor grafico.

5.7 Histogramas

Los histogramas son gráficos que muestran la distribución de una serie de datos. Para ello, el rango completo de los datos se divide en subrangos menores denominados intervalos de forma que el histograma muestra cuántos puntos hay en cada intervalo. El histograma es un gráfico de barras verticales en el cual el ancho de cada barra se corresponde con el rango del intervalo, y la altura de la barra se corresponde con el número de puntos dentro del intervalo. Los histogramas en MATLAB se crean con el comando `hist`, cuya sintaxis en su forma más simple es:

```
hist(y)
```

donde `y` es un vector que contiene los datos de los puntos que se van a estudiar. MATLAB divide el rango de los datos en 10 intervalos igualmente espaciados, seguidamente representa el número de datos que están dentro de cada intervalo.

Por ejemplo, los siguientes datos representan la temperatura máxima diaria (en grados Fahrenheit) en Washington DC, durante el mes de abril de 2002: 58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 91 93 89 91 80 59 69 56 64 63 66 64 74 63 69 (datos del Ministerio de Oceanografía y Meteorología de Estados Unidos). Se puede obtener un histograma para la representación de estos datos con los siguientes comandos:

```
>> y = [58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 91 93 89 91 80 59 69 56 64 63 66 64 74 63 69];
>> hist(y)
```

En la Figura 5.10 se muestra el gráfico generado a partir de estos comandos (las etiquetas de los ejes fueron añadidas posteriormente con el editor gráfico). El menor valor en este conjunto de datos es 48, y el mayor 93. Eso significa que el rango es 45 y que, por tanto, el ancho de cada intervalo es 4,5. El rango del primer intervalo es de 48 a 52,5, que contiene dos puntos. El rango del segundo intervalo es de 52,5 a 57, que contiene tres puntos, y así sucesivamente. Observe que los rangos 75 a 79,5 y 84 a 88,5 no contienen ningún punto.

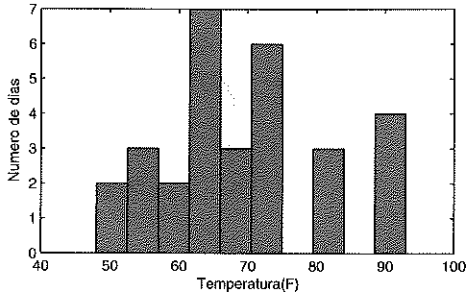


Figura 5.10: Histograma de temperaturas.

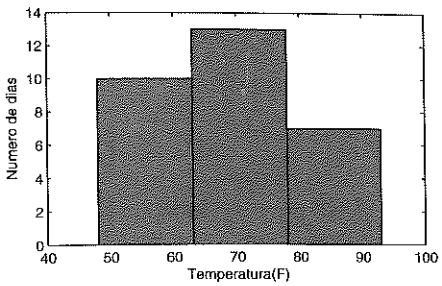
Como se dijo anteriormente, la división del histograma es, por defecto, de 10 intervalos. Sin embargo es posible que ésta no sea la que el usuario prefiere o desea para su representación gráfica. Para cambiar los intervalos se puede utilizar otra versión ampliada del comando `hist` que permite especificar el número de intervalos o el punto central de cada intervalo, como se muestra en las siguientes dos sintaxis de dicho comando:

`hist(y,numero_intervalos))` o bien `hist(y,x)`

donde `numero_intervalos` es un escalar que define el número de intervalos del histograma. De esta forma MATLAB divide el rango en intervalos de idéntico tamaño. Con respecto a la otra sintaxis, `x` es un vector que especifica la localización del centro de cada intervalo (la distancia entre los centros no tienen por qué ser igual para todos los intervalos). Los bordes de cada intervalo están a medio camino entre los centros.

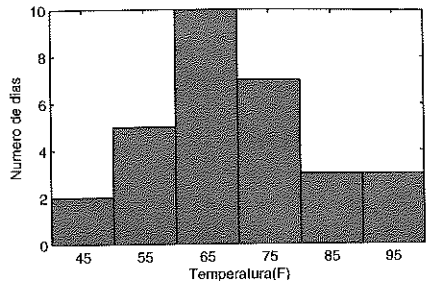
En el siguiente ejemplo el rango de temperaturas se divide en 3 intervalos. Para ello sólo hay que teclear el comando:

```
>> hist(y,3)
```



Como se muestra, la salida corresponde justamente a un histograma con tres intervalos igualmente espaciados.

El número y ancho de cada intervalo se puede especificar también utilizando un vector x que contiene los elementos que definen los centros de los intervalos. Por ejemplo, como se muestra a continuación se define un histograma que visualiza los datos de temperaturas en seis intervalos iguales con un ancho correspondiente a 10 grados. Los elementos que contiene el vector x para este caso son: 45, 55, 65, 75, 85 y 95. El dibujo se obtiene tecleando los comandos:




```
>> x = [45:10:95]
x =
   45   55   65   75   85   95
>> hist(y,x)
```

El comando `hist` se puede utilizar en operaciones que proporcionan salidas numéricas en lugar de representar gráficamente el histograma. Para obtener el número de puntos de cada intervalo se puede teclear cualquiera de los siguientes comandos:

`n = hist(y)` `n = hist(y,numero_intervalos)` `n = hist(y,x)`

La salida `n` es de tipo vector. El número de elementos en `n` es igual al número de intervalos, siendo cada elemento de `n` el número de elementos (frecuencia) dentro del intervalo en cuestión. Por ejemplo, el histograma de la Figura 5.10 se puede crear de la siguiente forma:

```
>> n = hist(y)
n =
   2   3   2   7   3   6   0   3   0   4
```

El vector muestra los elementos de cada intervalo.

El vector `n` nos muestra que el primer intervalo contiene 2 puntos, el segundo 3, y así sucesivamente.

Otro salida numérica opcional que se pude obtener con el comando `hist` es la localización del centro de cada intervalo. Este tipo de salida se puede obtener con cualquiera de las siguientes variaciones de `hist`:

```
[n xout] = hist(y)
```

```
[n xout] = hist(y,numero_intervalos)
```

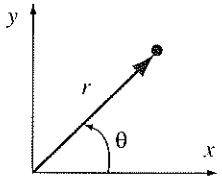
donde `xout` es un vector en el cual el valor de cada elemento representa la localización del centro del intervalo correspondiente. Por ejemplo, en el histograma de la Figura 5.10:

```
>> [n xout] = hist(y)
n =
   2   3   2   7   3   6   0   3   0   4
xout =
   50.2500   54.7500   59.2500   63.7500   68.2500   72.7500   77.2500   81.7500   86.2500   90.7500
```

El vector `xout` nos muestra que el centro del primer intervalo es 50,25, el centro del segundo intervalo es 54,75, y así sucesivamente.

5.8 Gráficos en coordenadas polares

Las coordenadas polares, en las cuales la posición de un punto queda definida a partir del ángulo θ y del radio *r* (distancia), se utilizan frecuentemente en operaciones científicas de distinto tipo. En MATLAB se utiliza el comando `polar` para dibujar un gráfico en coordenadas polares. Este comando tiene la forma:

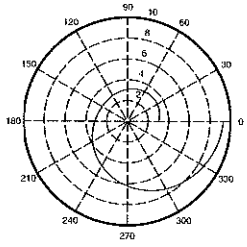



```
polar(theta, radio, 'especificadores de linea')
```

Vector — Vector — (Opcional) Especificadores que definen el tipo y color de la línea, así como del marcador utilizado en la representación.

donde `theta` y `radio` son vectores cuyos elementos definen las coordenadas de los puntos que se van a representar. El comando `polar` dibuja los puntos sobre una rejilla especial ideada para representar coordenadas polares. Los especificadores de línea son los mismos que los utilizados en el comando `plot`. Para representar una función $r = f(\theta)$ en un dominio dado, primero se debe crear el vector θ con los valores deseados, y a continuación el vector *r* correspondiente a los valores $f(\theta)$, utilizando operaciones elemento a elemento. A continuación sólo resta utilizar estos dos vectores en el comando `polar`.

Por ejemplo, para representar la función $r = 3\cos^2(0{,}5\theta) + \theta$ para $0 \le \theta \le 2\pi$, se utilizarían los siguientes comandos:

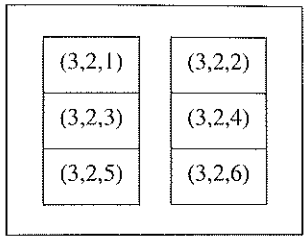


```
t = linspace(0,2*pi,200);
r = 3*cos(0.5*t).^2+t;
polar(t,r)
```

5.9 Representación de más de un gráfico en la misma página

Para representar más de un gráfico en la misma ventana gráfica (esta vez de forma separada) se utiliza el comando `subplot`, cuya sintaxis es:

```
subplot(m,n,p)
```

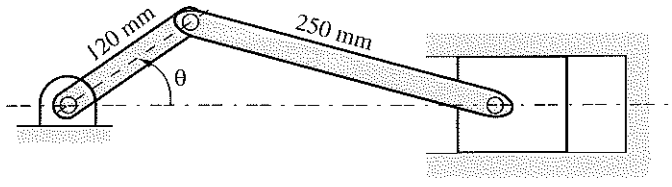


Este comando divide la Ventana de Gráficos en subventanas rectangulares de dimensión $m \times n$, donde es posible representar más de un gráfico de forma independiente. Los gráficos, en ese caso, son gestionados en forma de matriz $m \times n$, donde cada elemento es un subgráfico. Los subgráficos son numerados de 1 a $m \cdot n$, siendo el número 1 el subgráfico de la esquina superior izquierda, y el último será el que hace el número $m \cdot n$. Los números se incrementan de izquierda a derecha dentro de una fila, desde la primera hasta la última. El comando `subplot(m,n,p)` hace que el subgráfico con numeración `p` sea el utilizado, en un momento dado, como subgráfico por defecto. Esto significa que el próximo comando gráfico que se ejecute (cualquiera de los ya vistos) creará un gráfico (con su formato correspondiente) en esa posición o subgráfico indicado. Por ejemplo, el comando `subplot(3,2,1)` crea 6 áreas en forma de 3 filas y 2 columnas, y hace que la primera de ellas (la de la esquina superior izquierda) sea el subgráfico por defecto (la número 1). En el Problema de ejemplo 5.2 se muestra un ejemplo de utilización de este comando.

5.10 Ejemplos de aplicaciones con MATLAB

Problema de ejemplo 5.2: Funcionamiento del mecanismo de un pistón

El mecanismo de un pistón conectado mediante una varilla a una manivela es un problema clásico utilizado frecuentemente en aplicaciones de ingeniería. En el mecanismo que se muestra en la figura adjunta, la manivela tiene una velocidad constante de rotación de 500 rpm.



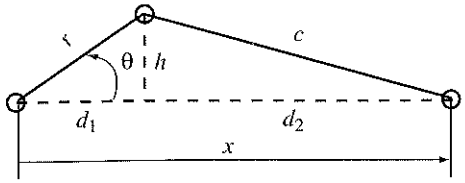
Calcular y representar gráficamente la posición, velocidad y aceleración del pistón para una de las revoluciones de la manivela. Representar tres gráficos distintos en la misma ventana gráfica. Considerar que $\theta = 0°$ en el instante $t = 0$.

Solución

La manivela va rotando con velocidad angular constante $\dot{\theta}$. Esto significa que si tomamos $\theta = 0°$ cuando $t = 0$, entonces, el ángulo θ para un determinado instante de tiempo t viene dado por $\theta = \dot{\theta}t$, y que $\ddot{\theta} = 0$ en cualquier instante de tiempo.

Las distancias d_1 y h vienen dadas por

$$d_1 = r\cos\theta \quad \text{y} \quad h = r\,\text{sen}\,\theta$$



Conociendo h, la distancia d_2 se puede calcular utilizando el teorema de Pitágoras:

$$d_2 = (c^2 - h^2)^{1/2} = (x^2 - r^2\text{sen}^2\theta)^{1/2}$$

La posición x del pistón viene dada por:

$$x = d_1 + d_2 = r\cos\theta + (c^2 - r^2\,\text{sen}^2\theta)^{1/2}$$

La derivada de x con respecto al tiempo nos da la velocidad del pistón:

$$\dot{x} = -r\dot{\theta}\,\text{sen}\,\theta - \frac{r^2\dot{\theta}\,\text{sen}\,2\theta}{2(c^2 - r^2\,\text{sen}^2\theta)^{1/2}}$$

La segunda derivada de x con respecto al tiempo nos da la aceleración del pistón:

$$\ddot{x} = -r\dot{\theta}^2\cos\theta - \frac{4r^2\dot{\theta}^2\cos 2\theta(c^2 - r^2\,\text{sen}^2\theta) + (r^2\dot{\theta}\,\text{sen}\,2\theta)^2}{4(c^2 - r^2\,\text{sen}^2\theta)^{3/2}}$$

En esta ecuación se ha tomado $\dot{\theta}$ con valor cero.

El programa MATLAB (fichero script) que calcula y representa la posición, velocidad y aceleración del pistón para una revolución de la manivela, se muestra a continuación:

```
THDrpm = 500; r = 0.12; c = 0.25;                    % Se define θ, r y c.
THD = THDrpm*2*pi/60;                                % Se lleva a cabo un cambio de unidad para θ, de rpm a rad/s.
tf = 2*pi/THD;                                       % Se calcula el tiempo para una revolución de la manivela.
t = linspace(0,tf,200);                              % Se crea un vector para el tiempo, de 200 elementos.
TH = THD*t;                                          % Se calcula θ para cada t.
d2s = c^2-r^2*sin(TH).^2;                            % Se calcula d2 para cada θ.
x = r*cos(TH) + sqrt(d2s);                           % Se calcula x para cada θ.
xd = -r*THD*sin(TH) - (r^2*THD*sin(2*TH))./(2*sqrt(d2s));
xdd = -r*THD^2*cos(TH)-(4*r^2*THD^2*cos(2*TH).*d2s +
  (r^2*sin(2*TH)*THD).^2)./(4*d2s.^(3/2));           % Se calcula ẋ y ẍ para cada θ.
subplot(3,1,1)
plot(t,x)                                            % Se representa x frente a t.
grid                                                 % Se da formato al primer gráfico.
```

```
xlabel('Tiempo (s)')
ylabel('Posicion (m)')
subplot(3,1,2)
plot(t,xd)                    % Se representa ẋ frente a t.
grid                          % Se da formato al segundo gráfico.
xlabel('Tiempo (s)')
ylabel('Velocidad (m/s)')
subplot(3,1,3)
plot(t,xdd)                   % Se representa ẍ frente a t.
grid                          % Se da formato al tercer gráfico.
xlabel('Tiempo (s)')
ylabel('Aceleracion (m/s^2)')
```

Un vez que se ejecuta este fichero script se generan los tres gráficos en la misma ventana, como se muestra en la Figura 5.11. Esta figura muestra claramente que la velocidad del pistón es cero en los puntos finales del rango de su viaje, cuando cambia de dirección en su movimiento. También se puede comprobar que la aceleración es máxima en puntos muy cercanos a los cambios de movimiento que éste experimenta.

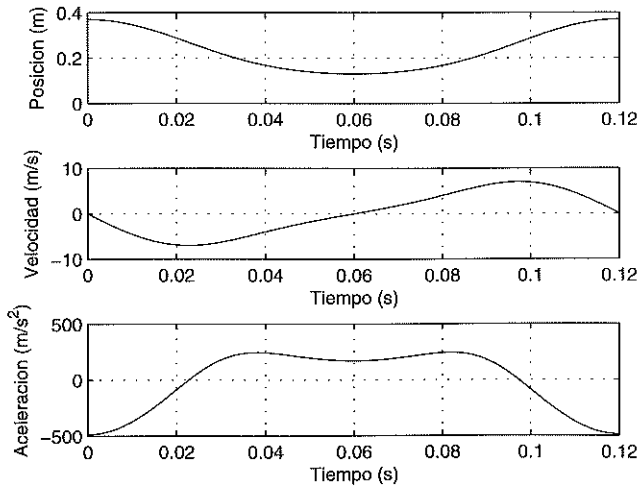
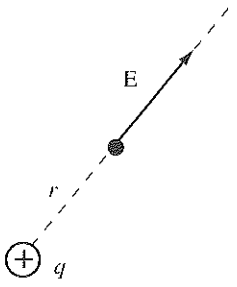


Figura 5.11: Posición, velocidad y aceleración del pistón en función del tiempo.

Problema de ejemplo 5.3: Dipolo eléctrico

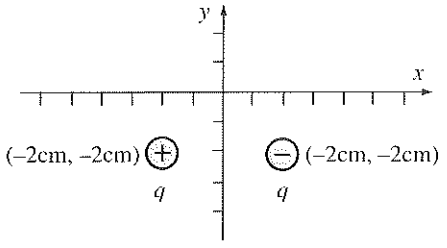
El campo eléctrico en un punto generado por una carga eléctrica se puede representar como un vector **E**, cuya magnitud E viene dada por la ley de Coulomb:

$$E = \frac{1}{4\pi\varepsilon_0}\frac{q}{r^2}$$



donde $\varepsilon_0 = 8{,}8541878 \times 10^{-12}\dfrac{C^2}{N \cdot m^2}$ es la permitividad (coeficiente dieléctrico) del vacío, q es la magnitud de la carga y r es la distancia entre la carga y el punto. La dirección de **E** forma una línea recta que conecta la carga con el punto. El vector **E** apunta en dirección contraria a q si q es positiva; por el contrario apuntará en dirección hacia q si q es negativa. Cuando se colocan una carga positiva y otra negativa, de igual magnitud, separadas entre sí una cierta distancia, se crea un dipolo eléctrico. El campo eléctrico **E**, en cualquier punto, se obtiene por superposición de los campos eléctricos de cada carga.

Sea un dipolo eléctrico con $q = 12 \times 10^{-9}$ C, tal como se muestra en la figura adjunta. Calcular y representar la magnitud del campo eléctrico a lo largo del eje x, desde $x = -5$ cm hasta $x = 5$ cm.



Solución

El campo eléctrico en cualquier punto $(x, 0)$ a lo largo del eje x se obtiene sumando el campo eléctrico generado por cada una de las cargas.

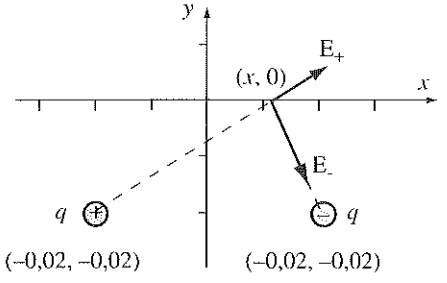
$$\mathbf{E} = \mathbf{E}_- + \mathbf{E}_+$$

La magnitud del campo eléctrico viene dada por lo longitud del vector **E**.

El problema se resuelve a partir de los siguientes pasos:

Paso 1: Crear un vector x con los puntos a lo largo del eje x.

Paso 2: Calcular la distancia (y distancia al cuadrado) desde cada carga hasta los puntos del eje x.



$$r_{menos} = \sqrt{(0{,}02 - x)^2 + 0{,}02^2} \qquad r_{más} = \sqrt{(x + 0{,}02x)^2 + 0{,}02^2}$$

Paso 3: Construir vectores unitarios con dirección desde cada carga a los distintos puntos sobre el eje x.

$$\mathbf{E}_{\text{menosUV}} = \frac{1}{r_{\text{menos}}}((0{,}02 - x)\mathbf{i} - 0{,}02\mathbf{j}) \qquad \mathbf{E}_{\text{másUV}} = \frac{1}{r_{\text{más}}}((x + 0{,}02)\mathbf{i} + 0{,}02\mathbf{j})$$

Paso 4: Calcular la magnitud de los vectores $\mathbf{E}_-$ y $\mathbf{E}_+$ en cada punto utilizando la ley de Coulomb.

$$\mathbf{E}_{\text{menosMAG}} = \frac{1}{4\pi\varepsilon_0}\frac{q}{r^2_{\text{menos}}} \qquad \mathbf{E}_{\text{másMAG}} = \frac{1}{4\pi\varepsilon_0}\frac{q}{r^2_{\text{más}}}$$

Paso 5: Crear los vectores $\mathbf{E}_-$ y $\mathbf{E}_+$ multiplicando los vectores unitarios por las magnitudes.

Paso 6: Crear el vector $\mathbf{E}$ sumando los vectores $\mathbf{E}_-$ y $\mathbf{E}_+$.

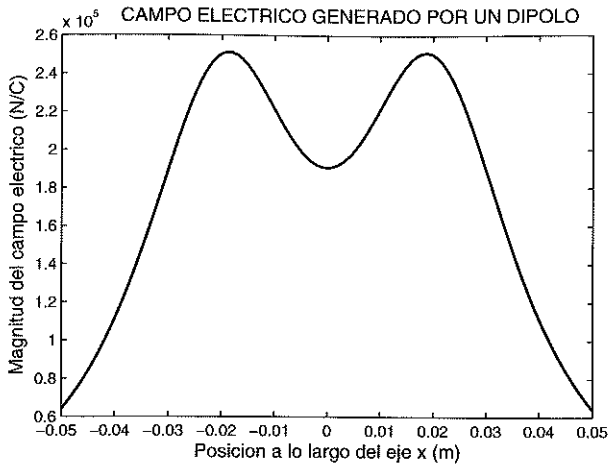
Paso 7: Calcular E, la magnitud (longitud) de $\mathbf{E}$.

Paso 8: Representar E en función de x.

A continuación se muestra el fichero script que resuelve este problema:

```
q = 12e-9;                                                    % Se crea el vector columna x.
epsilon0 = 8.854187e-12;
x = [-0.05:0.001:0.05]';
rmenosS = (0.02-x).^2 + 0.02^2; rmenos = sqrt(rmenosS);       % Paso 2, cada variable es un vector columna.
rmasS = (x + 0.02).^2 + 0.02^2; rmas = sqrt(rmasS);
EmenosUV = [((0.02-x)./rmenos),(-0.02./rmenos)];              % Pasos 3 y 4, cada variable es una matriz de dos columnas donde cada fila contiene un vector para cada x correspondiente.
EmasUV = [((x+.02)./rmas),(0.02./rmas)];
EmenosMAG = (q/(4*pi*epsilon0))./rmenosS;
EmasMAG = (q/(4*pi*epsilon0))./rmasS;
Emenos = [EmenosMAG.*EmenosUV(:,1),EmenosMAG.*EmenosUV(:,2)]; % Paso 5.
Emas = [EmasMAG.*EmasUV(:,1), EmasMAG.*EmasUV(:,2)];
E = Emenos + Emas;                                            % Paso 6.
EMAG = sqrt(E(:,1).^2+E(:,2).^2);                             % Paso 7.
plot(x,EMAG,'k','linewidth',1)
xlabel('Posicion a lo largo del eje x (m)','FontSize',12)     % Paso 8. Representación gráfica del problema.
ylabel('Magnitud del campo electrico (N/C)','FontSize',12)
title('CAMPO ELECTRICO GENERADO POR UN DIPOLO','FontSize',12)
```

La siguiente figura muestra el resultado obtenido una vez ejecutado el script anterior.



5.11 Problemas

1. Represente dos gráficos, de forma separada, de la función $f(x) = 0{,}6x^5 - 5x^3 + 9x + 2$; uno de los gráficos debe estar en el dominio $-4 \leq x \leq 4$, y el otro en el dominio $-2{,}7 \leq x \leq 2{,}7$.

2. Represente la función $f(x) = \dfrac{x^2 - x + 1}{x^2 + x + 1}$ para $-10 \leq x \leq 10$.

3. Utilice el comando `fplot` para representar la función:

$$f(x) = 0{,}01x^5 - 0{,}03x^4 + 0{,}4x^3 - 2x^2 - 6x + 5$$

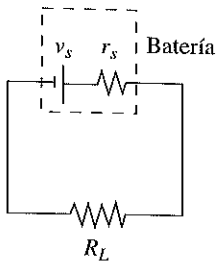
en el dominio $-4 \leq x \leq 6$.

4. Represente la función $f(x) = \dfrac{1{,}5x}{x - 4}$ para $-10 \leq x \leq 10$. Observe que esta función posee una asíntota vertical en el punto $x = 4$. Represente la función mediante la creación de dos vectores para el dominio de x. El primer vector (llámelo x1) contendrá los elementos -10 a 3,7, y el segundo vector (llámelo x2) los elementos 4,3 hasta 10. Adicionalmente habrá que crear dos vectores y1 e y2 para las correspondencias con los valores de la función sobre los dos vectores anteriormente creados para el dominio de x. Seguidamente represente la función mediante dos curvas en el misma región gráfica (y1 frente a x1, y2 frente a x2).

5. Represente la función $f(x) = \dfrac{x^2 - 5x + 10}{x^2 - 2x - 3}$ para $-10 \leq x \leq 10$. Observe que esta función tiene dos asíntotas verticales. Represente la función dividiendo el dominio de x en tres partes; una que vaya desde -10 hasta aproximadamente la asíntota izquierda, otra entre las dos asíntotas, y una tercera desde aproximadamente la asíntota derecha hasta 10. Establezca el rango del eje y entre -20 y 20.

6. Represente la función $f(x) = 3x\text{sen}(x) - 2x$ y su derivada, ambas en la misma región gráfica, en el intervalo $-2\pi \leq x \leq 2\pi$. Represente la función con una línea sólida, y su derivada con una línea discontinua. Añada una leyenda y etiquetas para los ejes.

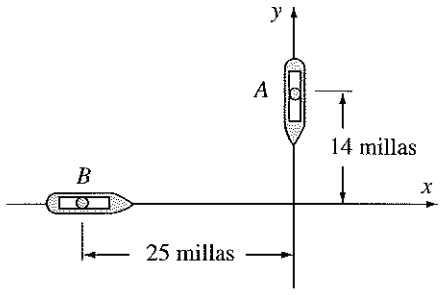
7. Un circuito eléctrico tiene una fuente de voltaje v_S con una resistencia interna r_S, y una resistencia de carga R_L, tal y como se muestra en la figura adjunta. La potencia P disipada en la carga de la resistencia viene dada por:



$$P = \frac{v_S^2 R_L}{(R_L + r_s)^2}$$

Represente la potencia P en función de R_L, en el intervalo $1 \leq R_L \leq 10\ \Omega$, suponiendo los siguientes valores para el circuito: $v_S = 12$ V, $r_S = 2{,}5\ \Omega$.

8. Un barco A viaja hacia el sur a una velocidad de 8 millas/hora, mientras que otro barco B viaja hacia el este a una velocidad de 16 millas/hora. A las 7 AM los barcos están a las distancias que se indican en la figura adjunta. Represente la distancia entre los barcos en función del tiempo para las siguientes 4 horas. El eje horizontal debe mostrar el tiempo actual del día, comenzando por las 7AM, mientras que el eje vertical debe mostrar la distancia. Ponga etiquetas a los ejes. Si la visibilidad es de 8 millas, estime la hora a partir de la cual las personas de un barco pueden ver a las del otro.



9. El valor V de un capital inicial P en una cuenta de ahorro que paga un interés anual r, viene dado por: $V = P\left(1 + \frac{r}{m}\right)^{mt}$, donde m es el número de capitalizaciones anuales, y t es el número de años. (Si el interés se capitaliza una vez al año, entonces $m = 1$, si se hace trimestralmente $m = 4$, y así sucesivamente.) Si el interés se capitaliza continuamente, el valor de V viene dado por $V = Pe^{rt}$.

   Considere un capital inicial de 5000 € a 15 años, con un interés anual del 7,5%. Muestre la diferencia entre tener un capital anual, trimestral y continuo, representando para ello el valor del capital en función del tiempo (años) para cada método de capitalización. Represente los tres casos en una misma ventana gráfica, utilice una línea diferente para cada curva, ponga etiquetas a los ejes, añada una leyenda y también un título general del gráfico.

10. La forma del "Gateway Arch" de San Luis viene determinado por la ecuación:

$$y = 693{,}8 - 68{,}8 \cosh\left(\frac{x}{99{,}7}\right) \text{ ft.}$$

Represente un gráfico del mismo.

11. La magnitud M, en la escala Richter, de un terremoto viene dada por:

$$M = \frac{2}{3} \log \frac{E}{10^{4{,}4}}$$

donde E es la energía en julios liberada por el terremoto.

Haga un gráfico de E (en ordenadas) frente a M (en abscisas) para $3 \leq M \leq 8$. Utilice una escala logarítmica para E y una lineal para M. Etiquete los ejes y añada un título al gráfico.

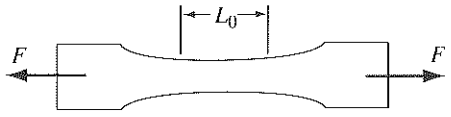
12. La posición x en función del tiempo t de una partícula que se mueve a lo largo de una línea recta viene dada por:

$$x(t) = 0{,}4t^3 - 2t^2 - 5t + 13 \text{ metros.}$$

La velocidad $v(t)$ de la partícula se calcula mediante la derivada de $x(t)$ con respecto al tiempo t, y la aceleración $a(t)$ se calcula derivando $v(t)$ con respecto al tiempo t.

Deduzca las expresiones de la velocidad y la aceleración de la partícula y represente su posición, velocidad y aceleración en función del tiempo para $0 \leq t \leq 7$ s. Utilice el comando `subplot` para crear tres gráficos en la misma Ventana de Gráficos, representando la posición en la parte superior, la velocidad en el medio y la aceleración al final. Etiquete los ejes apropiadamente con las unidades correctas.

13. En un ensayo típico de tracción, una probeta (o muestra) en forma de "hueso de perro" se somete a tracción en una máquina. Durante el ensayo se mide la fuerza F necesaria para estirar la probeta y la longitud L de la sección de referencia. Estos datos se utilizan para hacer un diagrama tensión-deformación del material. Hay dos definiciones de tensión y deformación: una técnica y otra real. Técnicamente, la tensión σ_t y la deformación ε_t se definen como



$\sigma_t = \dfrac{F}{A_0}$ y $\varepsilon_t = \dfrac{L - L_0}{L_0}$, donde L_0 y A_0 son la longitud de referencia inicial y el área transversal inicial de la probeta, respectivamente. Por el contrario, las definiciones verdaderas de la tensión σ_v y de la deformación ε_v son:

$$\sigma_v = \frac{F}{A_0}\frac{L}{L_0} \text{ y } \varepsilon_v = \ln\frac{L}{L_0}.$$

A continuación se dan las medidas de fuerza (F) y de longitud de referencia (L) de un ensayo de tracción realizado sobre una probeta de aluminio. Antes del ensayo, la probeta tiene una sección transversal circular de radio 6,4 mm. La longitud de referencia inicial es $L_0 = 25$ mm. Utilice los datos para calcular y representar gráficamente en un mismo gráfico las curvas tensión-deformación según las dos definiciones dadas, la técnica y la verdadera. Etiquete los ejes y las curvas convenientemente. Cuando la fuerza se mide en newtons (N) y el área en m^2, la tensión viene dada en Pascales (Pa).

F(N)	0	13345	26689	40479	42703	43592	44482	44927
L(mm)	25	25,037	25,073	25,113	25,122	25,125	25,132	25,144
F(N)	45372	46276	47908	49035	50265	53213	56161	
L(mm)	25,164	25,208	25,409	25,646	26,084	27,398	29,150	

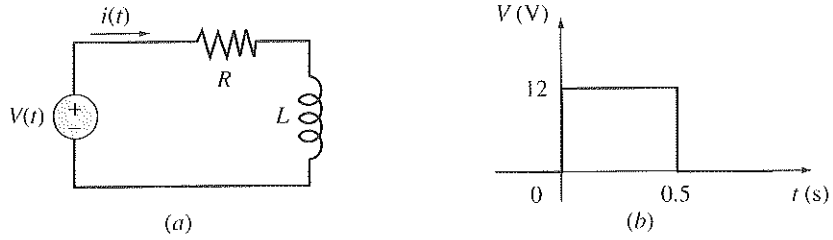
14. Una resistencia de $R = 4\ \Omega$ y un inductor de $L = 1{,}3$ H se conectan en un circuito a una fuente de voltaje, como se muestra en la Figura (a) (circuito RL). Cuando la fuente de voltaje aplica un

pulso de voltaje rectangular con una amplitud de $V = 12$ V durante 0,5 s, como se muestra en la Figura (b), la intensidad de corriente $i(t)$ en el circuito en función del tiempo viene dada por:

$$i(t) = \frac{V}{R}(1 - e^{(-Rt)/L}) \text{ para } 0 \le t \le 0{,}5 \text{ s}$$

$$i(t) = e^{-(Rt)/L}\frac{V}{R}(e^{(0{,}5R)/L} - 1) \text{ para } 0{,}5 \le t \text{ s}$$

Represente gráficamente la corriente en función del tiempo para $0 \le t \le 2$ s.



15. La órbita de los planetas alrededor del Sol se puede modelar, de forma aproximada, mediante la ecuación polar:

$$r = \frac{eP}{1 - e\cos\theta}$$

A continuación se muestran los valores de las constantes P y e para cuatro planetas. Dibuje las órbitas de estos cuatro planetas en un solo gráfico (utilizando el comando `hold on`)

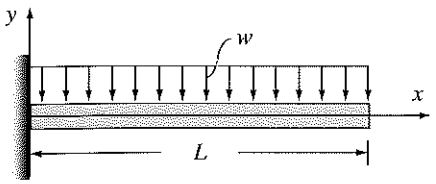
Planeta	P ($\times 10^6$ m)	e
Mercurio	269,2	0,206
Venus	15913	0,00677
Tierra	8964	0,0167
Marte	2421	0,0934

16. La densidad de probabilidad radial $P_r(r)$ de un átomo de hidrógeno en excitación, en su primer estado de excitación (números cuánticos $n = 2$ y $l = 0$), viene dada por:

$$P_r(r) = \frac{1}{8a_0}\left(\frac{r}{a_0}\right)^2\left(2 - \frac{r}{a_0}\right)e^{-r/a_0}$$

donde $a_0 = 52{,}92 \times 10^{-11}$ m es el radio de Bohr.
Construya un gráfico de P_r en función de r/a_0 para $0 \le r/a_0 \le 15$.

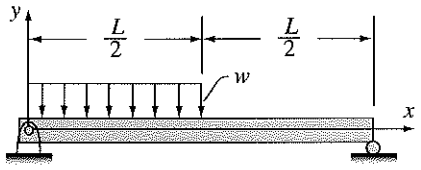
17. Una viga en voladizo es aquella que se encuentra sujeta por un extremo mientras que su otro extremo está libre. La deflexión y en el punto x de una viga de este tipo cargada con una carga distribuida uniformemente w viene dada por la ecuación:



$$y = \frac{-w}{24EI}(x^4 - 4Lx^3 + 6L^2x^2)$$

donde E es el módulo de elasticidad, I el momento de inercia y L la longitud de la viga. La viga que se muestra en la figura adjunta posee las siguientes características: $L = 6$ m, $E = 70 \times 10^9$ Pa (aluminio), $I = 9{,}19 \times 10^{-6}$ m^4 y $w = 80 \times 10^3$ N/m.
Represente gráficamente la deflexión de la viga y en función de x.

18. En la figura adjunta se muestra una viga apoyada sobre soportes, sometida a una carga w con distribución constante sobre la mitad de su longitud. En este caso, la deflexión y en función de x viene dada por las ecuaciones:



$$y = \frac{-wx}{384EI}(16x^3 - 24Lx^2 + 9L^3) \text{ para } 0 \le x \le \frac{L}{2}$$

$$y = \frac{-wx}{384EI}(8x^3 - 24Lx^2 + 17L^2x - L^3) \text{ para } \frac{L}{2} \le x \le L$$

donde E es el módulo de elasticidad, I el momento de inercia y L la longitud de la viga. La viga que se muestra en la figura adjunta posee las siguientes características: $L = 20$ m, $E = 200 \times 10^9$ Pa (acero), $I = 348 \times 10^{-6}$ m^4 y $w = 5 \times 10^3$ N/m.
Represente gráficamente la deflexión de la viga y en función de x.

19. El límite de fluencia (el límite del rango elástico) de la mayoría de los metales es sensible a la rapidez con que se cargan estos materiales (velocidad de carga). Los datos que se muestran a continuación muestran el límite de fluencia de cierto acero en función de varias velocidades de carga.

Velocidad de carga (s^{-1})	0,00007	0,0002	0,05	0,8	4,2	215	3500
Límite de fluencia (MPa)	345	362	419	454	485	633	831

El límite de fluencia en función de la velocidad de carga se puede modelar mediante la siguiente ecuación:

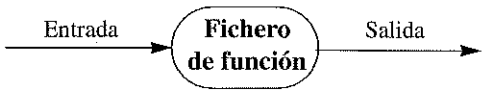
$$\sigma_n = 350\left[\left(\frac{\varepsilon_t}{210}\right)^{0,16} + 1\right]$$

donde σ_{fl} es el límite de fluencia en MPa, y ε_t es la velocidad de carga en s^{-1}. Construya un gráfico del límite de fluencia (eje vertical) en función de la velocidad de carga (eje horizontal). Utilice una escala lineal para el límite de fluencia y una logarítmica para la velocidad de carga. Muestre los puntos procedentes de los datos con marcadores y los correspondientes al modelo con una línea sólida. Etiquete los ejes y añada una leyenda al gráfico.

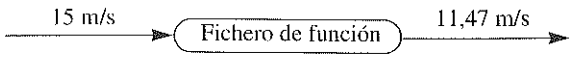
Capítulo 6
Funciones y ficheros de función

Una función matemática, $f(x)$, asocia un único número (valor de la función) a cada uno de los valores de x. Las funciones se pueden expresar en la forma $y = f(x)$, donde $f(x)$ es habitualmente una expresión matemática en función de x. Cuando se introduce un valor x (entrada) en la expresión de la función, se obtiene un valor y (salida). Existen muchas funciones que están ya programadas en MATLAB (funciones predefinidas), y que pueden ser utilizadas en expresiones simplemente tecleando su nombre junto con el argumento de entrada (ver Sección 1.5); ejemplos de estas funciones son `sin(x)`, `cos(x)`, `sqrt(x)` y `exp(x)`. Frecuentemente, a la hora de programar, existe la necesidad de operar con funciones distintas que no están predefinidas. Cuando la expresión de la función es sencilla y sólo necesita ser ejecutada una vez, ésta se puede incluir como código del propio programa. Sin embargo, cuando la expresión se tiene que evaluar muchas veces para diferentes tipos de argumentos, es conveniente crear una función definida por el usuario. Una vez que se ha creado la función (y ha sido almacenada en disco), ésta puede ser utilizada de la misma forma que una función predefinida de MATLAB.

Una función definida por el usuario es un programa MATLAB que el usuario crea y almacena en disco en forma de un fichero que contiene la propia función (ficheros de función), de forma que esta función pueda ser utilizada al igual que el resto de las funciones del sistema. La función puede contener expresiones matemáticas simples o códigos que impliquen cálculos más complejos. En muchos casos este tipo de rutinas se comportan como subprogramas dentro de otros programas de mayor tamaño. La principal características de los ficheros de función es que tienen una entrada y una salida. Esto implica que los cálculos que se realizan dentro de los ficheros de función se llevan a cabo utilizando datos de entrada, y que los resultados de estos cálculos son transferidos fuera del fichero de función a través de la salida. La entrada y la salida pueden estar compuestas por una o más variables, y cada una de ellas pueden ser escalares, vectores o arrays de cualquier tamaño. A continuación se muestra de forma esquemática cómo se comportan los ficheros de función:



Un ejemplo muy sencillo de función definida por el usuario es la que calcula la altura máxima que alcanza una pelota cuando se tira hacia arriba con una velocidad dada. Para una velocidad v_0, la altura máxima h_{max} viene dada por: $h_{max} = \frac{v_0^2}{2g}$, donde g es la aceleración de la gravedad. Esto se puede escribir en forma de función de la siguiente manera: $h_{max}(v_0) = \frac{v_0^2}{2g}$. En este caso la entrada de la función es la velocidad (un número), y la salida es la altura máxima (un número). Por ejemplo, en unidades SI ($g = 9{,}81$ m/s^2), si la entrada es 15 m/s, entonces la salida será 11,47 m.



Además del uso que se le pueda dar a las funciones desde la perspectiva matemática, los ficheros de función se pueden utilizar como subprogramas dentro de otros programas de mayor tamaño. De esta forma se pueden construir programas grandes uniendo bloques más pequeños que se pueden evaluar y depurar independientemente. Los ficheros de función son similares a lo que se conoce comúnmente en otros lenguajes de programación como subrutinas (en el caso de Basic y Fortran), procedimientos (en Pascal) o funciones (en C).

Los ficheros de función se explicarán en las Secciones 6.1 a la 6.7. Además de estos ficheros de función que se pueden almacenar en disco de forma independiente y pueden ser invocados desde programas, MATLAB proporciona una opción que permite definir funciones matemáticas definidas por el usuario dentro de un programa (no en un fichero separado). Esta acción se puede realizar con el comando `inline` que se explicará en la Sección 6.8.

6.1 Creación de un fichero de función

Los ficheros de función se crean y editan como si se trataran de ficheros scripts, es decir, utilizando la Ventana de Edición/Depuración. Esta ventana, como ya se vio, se puede abrir desde la Ventana de Comandos. En el menú **File**, se selecciona **New** y luego **M-file**. Una vez que se ha abierto, la Ventana de Edición/Depuración mostrará un aspecto similar al que se ilustra en la Figura 6.1. La primera línea dentro de un fichero de función debe ser la propia definición de la función, como ser verá en la próxima sección.

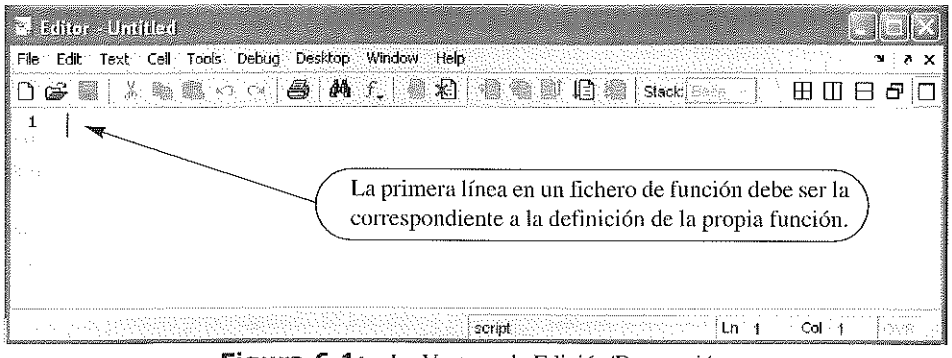


Figura 6.1: La Ventana de Edición/Depuración.

6.2 Estructura de un fichero de función

La estructura típica de un fichero de función se muestra en la Figura 6.2. En concreto, la función que se muestra calcula la amortización mensual y el pago total de un préstamo. La entrada a la función son la cantidad del préstamo, la tasa de interés anual y la duración del préstamo (en años). La salida de la función es la amortización mensual del préstamo y también el pago total por el préstamo.

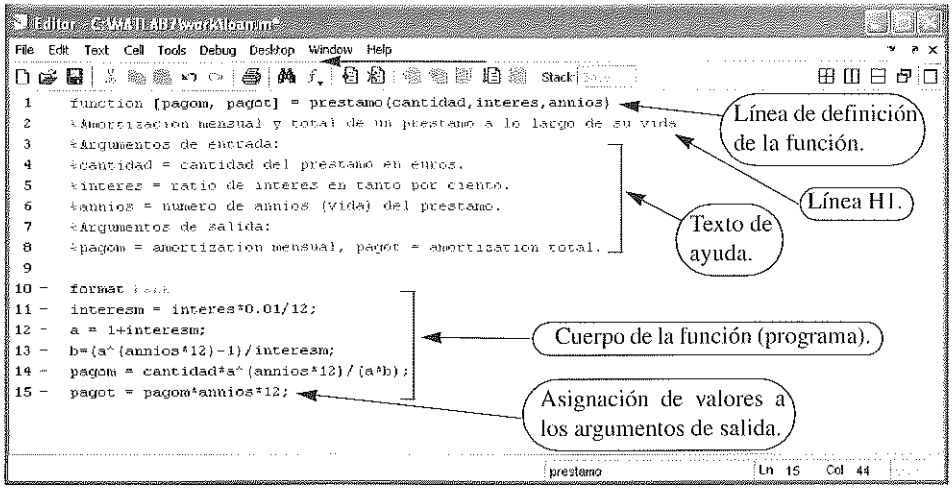


Figura 6.2: Estructura de un fichero de función.

Las distintas partes de los ficheros de función se describirán con más detalle en las próximas secciones.

6.2.1 *Línea de definición de la función*

La primera línea ejecutable en un fichero de función debe ser la definición de la propia función. En otro caso, el fichero será considerado como un fichero script normal. La línea de definición de la función:

- Define que el fichero será tratado como un fichero de función.
- Define el nombre de la función.
- Define el número y orden de los argumentos de entrada y salida.

La forma que tiene la línea de definición es:

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

La palabra function debe ser la primera palabra que aparezca, tecleada en minúsculas.

Lista de argumentos de salida encerrados entre corchetes.

Nombre de la función.

Lista de argumentos de entrada encerrados entre paréntesis.

La palabra `function`, tecleada en minúsculas, debe ser la primera palabra de la primera línea del fichero. En la pantalla esta palabra aparecerá en azul. El nombre de la función se introduce después del signo igual. Este nombre puede estar formado por letras, dígitos y el carácter de subrayado. Las reglas para el nombre de la función son las mismas que para la construcción de nombres de variables, descritas en la Sección 1.6.2. Es una buena costumbre evitar usar para este propósito nombres de funciones ya predefinidas por MATLAB, así como nombres de variables ya definidas por el usuario o internamente por MATLAB.

6.2.2 *Argumentos de entrada y salida*

Los argumentos de entrada y salida se utilizan para transferir datos hacia dentro y hacia fuera de la propia función. Los argumentos de entrada se introducen entre paréntesis a continuación del nombre de la función. Por lo general, las funciones se declaran con al menos un argumento de entrada, aunque también es posible definir funciones que no tengan ningún argumento de entrada. En cualquier caso, si la función se piensa para más de un parámetro, los argumentos de entrada deben ir separados por comas. Normalmente el código interno que contiene la función está pensado para operar sobre los argumentos de entrada, y por tanto se presupone que estos deben ser valores apropiados. Esto implica que las expresiones matemáticas que forman el cuerpo de la función deben ser escritas según las dimensiones de los argumentos, ya sean estos escalares, vectores o arrays. En el ejemplo de la Figura 6.2 se tienen tres argumentos de entrada `(cantidad, interes, annios)`, y en las expresiones matemáticas se cosideran como escalares. Los valores reales de los parámetros de entrada se asignan cuando la función se utiliza (se llama). De manera similar, si los argumentos son vectores o arrays, las expresiones matemáticas dentro del cuerpo de la función se deben escribir para que efectúen operaciones algebraicas lineales u operaciones elemento a elemento.

Los argumentos de salida que se encuentran entre corchetes en la parte izquierda del operador de asignación transfieren la salida desde el fichero de función. Estos ficheros pueden tener uno o varios argumentos de salida, e incluso pueden no tener ninguno. Si hay más de uno, entonces los argumentos se deben separar por comas. Si sólo hay un argumento de salida, éste se puede teclear sin corchetes. **Para que funcione correctamente un fichero de función, a los argumentos de salida se les deben asignar valores durante la ejecución del código correspondiente al cuerpo de la función**. En el ejemplo de la Figura 6.2 podemos observar dos parámetros de salida `[pagom, pagot]`. Cuando una función no posee argumentos de salida, se puede omitir el operador de asignación en la definición de la función. Este tipo de funciones también son comunes, y aunque no retornan ningún valor pueden encargarse de generar un gráfico, por ejemplo, o volcar datos a un fichero.

También es posible transferir cadenas en un fichero de función. Para hacer esto sólo hay que tratar a la cadena como si fuera una variable más que forma parte de las variables de entrada (texto encerrado entre comillas simples). Las cadenas se pueden utilizar para transferir nombres de otras funciones dentro de un fichero de función.

Habitualmente todas las entradas y salidas de un fichero de función se transfieren a través de argumentos o parámetros de entrada y salida. Por lo demás, todo lo relativo a la entrada y salida de valores en ficheros script sigue siendo válido para este caso, y por tanto se puede aplicar a los ficheros de función. Esto significa que cualquier variable a la que se le asigne cualquier valor en un fichero de función será visualizada en la pantalla, a menos que se ponga un punto y coma al final del comando de asignación. Además, el comando `input` se puede utilizar también para introducir datos interactivamente, y los comandos `disp`, `fprintf` y `plot` para visualizar información textual en pantalla, volcarla a un fichero o representar gráficos, como hemos visto para ficheros script. Veamos algunos ejemplos de primeras líneas de definición de funciones con diferentes combinaciones de argumentos de entrada y salida.

Línea de definición de función	Comentarios
function [pagom, pagot] = prestamo(cantidad,interes,annios)	Tres argumentos de entrada, dos argumentos de salida.
function [A] = RectArea(a,b)	Dos argumentos de entrada, un argumento de salida.
function [V,S] = EsferaVolArea(r)	Una variable de entrada, dos variables de salida.
function trayectoria(v,h,g)	Tres argumentos de entrada, ningún argumento de salida.

6.2.3 *La línea H1 y las líneas de texto de ayuda*

La línea H1 y las líneas de texto de ayuda son líneas de comentarios (líneas que empiezan por el carácter tanto por cierto %) a continuación de la línea de definición de la función. Aunque estas líneas son opcionales, son muy útiles para proporcionar información sobre la función en sí. La línea H1 es la primera línea de comentario y normalmente contiene el nombre y una descripción corta de la función. Cuando un usuario teclea (en la Ventana de Comandos) `lookfor palabra`, MATLAB busca la `palabra` especificada en las líneas H1 de todas las funciones, de forma que si la palabra se encuentra en alguna de las líneas H1, ésta se muestra al usuario (la línea H1 completa).

Las líneas de texto de ayuda son líneas de comentarios que siguen a la línea H1. Estas líneas contienen una explicación de la función y cualquier descripción relacionada con sus argumentos de entrada y salida. Las líneas de comentarios que se teclean entre la línea de definición de la función y la primera línea de código de la función (la línea H1 y el texto de ayuda) se muestran automáticamente cuando el usuario teclea el comando `help nombre_funcion` en la Ventana de Comandos. Este comando se puede utilizar tanto con funciones definidas por el usuario como con funciones MATLAB. Por ejemplo, para la función `prestamo` de la Figura 6.2, cuando se teclea el comando `help prestamo` en la Ventana de Comandos (hay que comprobar que la función se ha guardado en disco, en el directorio por defecto o en la ruta de búsqueda), la salida de dicho comando será:

```
>> help prestamo
Amortizacion mensual y total de un prestamo
Argumentos de entrada:
cantidad = cantidad del prestamo en euros.
interes = tasa de interes en tanto por ciento.
annios = numero de annios del prestamo.
Argumentos de salida:
pagom = amortizacion mensual, pagot = amortizacion total.
```

Un fichero de función puede contener además otros comentarios en el cuerpo de la función, aunque estos son ignorados por el comando `help`.

6.2.4 *Cuerpo de la función*

El cuerpo de la función contiene el programa (código) que realiza las operaciones especificadas. El código puede contener cualquier comando MATLAB de los ya vistos, es decir, cálculos, asignaciones, llamadas a funciones MATLAB, instrucciones de control de flujo (sentencias condicionales y bucles) como las que se verán en el Capítulo 7, comentarios, líneas en blanco, así como entradas y salidas de datos de forma interactiva.

6.3 Variables locales y globales

Todas las variables de un fichero de función son locales; cualquier variable, incluidos los argumentos de entrada y salida, se pueden utilizar dentro del fichero. Esto significa que las variables definidas en el fichero sólo se reconocerán dentro de él. Cuando se ejecuta una función, MATLAB utiliza un área especial de memoria separada del espacio de trabajo habitual utilizado para los ficheros scripts y la Ventana de Comandos. En un fichero de función se asignan nuevos valores a las variables de entrada cada vez que se invoca la función. Estas variables se utilizan para realizar los cálculos dentro del fichero de función. Cuando la función termina su ejecución, el valor del argumento de salida es transferido a las variables utilizadas en el momento en el que la función se invocó. Esto significa que el fichero de función puede tener variables con el mismo nombre que las variables definidas en un fichero script o en la Ventana de Comandos. El fichero de función no reconoce variables con el mismo nombre que hayan sido creadas fuera de la función. La asignación de valores a esas variables en el fichero de función no cambia su asignación inicial en ningún momento.

Cada fichero de función tiene sus propias variables locales, las cuales no se comparten con otras funciones o con el espacio de trabajo de la Ventana de Comandos y los ficheros de scripts. Sin embargo, también es posible crear variables que se puedan reconocer comúnmente en diferentes ficheros de función y en el espacio de trabajo. Esto se lleva a cabo declarando las variables como globales, utilizando la siguiente sintaxis:

```
global = nombre_variable
```

En un comando global se pueden declarar más de una variable a la vez, separándolas por espacios.

```
global CONSTANTE_GRAVITATORIA CoeficienteDeFriccion
```

- La variable que se desee reconocer en varios ficheros de función debe ser declarada como `global` en cada uno de ellos. De esta forma, la variable será reconocida en cada uno de estos ficheros y se podrá operar normalmente con ella.
- El comando `global` debe aparecer antes de que la variable sea utilizada. Es recomendable teclear el comando `global` al principio del fichero.
- El comando `global` debe ser tecleado en la Ventana de Comandos y/o en un fichero script para que una variable sea reconocida en el espacio de trabajo.
- El valor de la variable puede ser asignado, o reasignado, en cualquier lugar donde ésta sea de uso común.
- Es recomendable utilizar nombres suficientemente descriptivos (o utilizar letras mayúsculas) para las variables globales, de forma que se puedan distinguir de otro tipo de variables.

6.4 Almacenamiento de un fichero de función

Un fichero de función debe ser almacenado en disco antes de utilizarse. Esto se hace, al igual que con los ficheros script, utilizando la opción **Save as** del menú **File**, seleccionando la localización donde se guardará y proporcionando el nombre del fichero en cuestión. Es muy recomendable que el fichero tenga el mismo nombre que la función que contiene, en concreto el nombre que aparece en la línea de definición de la función. De esta forma la función se podrá invocar (llamar) utilizando su nombre. (Si un fichero de función se almacena en disco con otro nombre distinto del de la función que contiene, habrá que utilizar ese nombre para invocar la función.) Los ficheros de función se almacenan con la extensión .m. Por ejemplo:

Línea de definición de la función	Nombre del fichero
function [pagom, pagot] = prestamo(cantidad,interes,annios)	prestamo.m
function [A] = RectArea(a,b)	RectArea.m
function [V,S] = EsferaVolArea(r)	EsferaVolArea.m
function trayectoria(v,h,g)	trayectoria.m

6.5 Utilización de ficheros de función

Una función definida por el usuario se utiliza de la misma forma que una función predefinida en MATLAB. La función se puede ejecutar desde la Ventana de Comandos, desde un fichero script o también desde otra función. Para utilizar el fichero de función, éste debe estar almacenado o bien en el directorio de trabajo actual o bien encontrarse en la ruta de búsqueda (ver Secciones 4.3.1 y 4.3.2).

Una función se utiliza asignando su salida a una variable (o variables), como si se tratara de cualquier otra operación matemática, o simplemente tecleando su nombre en la Ventana de Comandos o en otro fichero. En cualquier caso el usuario debe conocer exactamente los argumentos de entrada y salida. Un argumento de entrada puede ser un número, una expresión computable, o también puede ser una variable que tenga un valor asignado. Los argumentos son asignados según su posición en la lista de parámetros de entrada y salida en la línea de definición de la propia función.

A continuación se muestran dos formas de utilizar una función a partir de la función `prestamo`, función definida por el usuario y descrita en la Figura 6.2. Recordemos que esta función calcula la amortización mensual y pago total (tiene dos argumentos de salida) de un préstamo. Los argumentos de entrada son la cantidad del préstamo, la tasa de interés anual y la duración del préstamo (número de años). En la primera ilustración la función `prestamo` se utiliza con números como argumentos de entrada:

```
>> [mensual, total] = prestamo(25000,7.5,4)

mensual =
   600.72

total =
   28834.47
```

El primer argumento es la cantidad de dinero del préstamo, el segundo la tasa de interés, y el tercero el número de años.

En la segunda ilustración la función `prestamo` se utiliza con dos variables previamente asignadas, además de un número, como argumentos de entrada.

```
>> a = 70000; b = 6.5;
>> [x y ] = prestamo(a,b,30)
x =
   440.06
y =
   158423.02
```

Se definen las variables `a` y `b`.

Se utilizan `a`, `b` y el número 30 como parámetros de entrada. Además se usan las variables `x` (cantidad mensual) e `y` (cantidad total) como argumentos de salida.

6.6 Ejemplos de ficheros de función

Problema de ejemplo 6.1: Función matemática

Escribir un fichero de función (llamado `Cap6Uno`) para la función $f(x) = \dfrac{x^4\sqrt{3x+5}}{(x^2+1)}$. La entrada de la función es x y la salida es $f(x)$. Escribir la función tal que x sea un vector. Utilizar la función para calcular:

a) $f(x)$ en $x = 6$.

b) $f(x)$ en $x = 1, 3, 5, 7, 9$ y 11.

Solución

El fichero de función para la función matemática $f(x)$ es:

```
function y = Cap6Uno(x)
y = (x.^4.*sqrt(3*x+5))./(x.^2+1).^2
```

Línea de definición de la función.

Asignación del argumento de salida.

Observe que la expresión matemática en el fichero de función se codifica mediante operaciones elemento a elemento. De esta forma, si la entrada x es un vector, entonces la salida y también será un vector. La función se almacena en el directorio de trabajo o en la ruta de búsqueda, y seguidamente se ejecuta en la Ventana de Comandos, tal y como se muestra en las siguientes ilustraciones.

a) Para calcular el valor de la función en $x = 6$ se procede a ejecutar la función tecleando `Cap6Uno(6)` en la Ventana de Comandos, o asignando el valor de la función a una nueva variable, como se muestra a continuación:

```
>> Cap6Uno(6)
ans =
   4.5401
>> F = Cap6Uno(6)
F =
   4.5401
```

b) Para calcular el valor de la función para varios valores de x, se creará en primer lugar un vector con los sucesivos valores de x, y después se utilizará el vector como argumento de entrada en la llamada a la función:

```
>> x = 1:2:11
x =
   1   3   5   7   9   11
>> Cap6Uno(x)
ans =
   0.7071   3.0307   4.1347   4.8971   5.5197   6.0638
```

Otra forma de hacer esto mismo es teclear el vector x directamente como argumento de entrada de la función:

```
>> H = Cap6Uno([1:2:11])
H =
   0.7071   3.0307   4.1347   4.8971   5.5197   6.0638
```

Problema de ejemplo 6.2: Conversión de unidades de temperatura

Escribir una función (llamada `FaC`) que convierta grados F a grados C. Utilizar la función para resolver el siguiente problema. El cambio en la longitud de un objeto, ΔL, se debe al cambio de temperatura, ΔT, que viene determinado por la siguiente expresión: $\Delta L = \alpha L\, \Delta T$, donde α es el coeficiente de dilatación. Determinar la variación del área de una chapa de aluminio ($\alpha = 23 \times 10^{-6}$ 1/ºC) de forma rectangular (4,5 m × 2,25 m) cuando la temperatura cambia de 40ºF a 92ºF.

Solución

Veamos primero la función que convierte grados F a grados C:

```
function C = FaC(F)                          % Línea de definición de la función.
%FaC convierte grados F a grados C
C = 5*(F-32)./9;                             % Asignación del parámetro de salida.
```

A continuación se muestra el fichero script (llamado Capitulo6Ejemplo2) que calcula la variación del área de la chapa debido al cambio de temperatura:

```
a1 = 4.5; b1 = 2.25; T1 = 40; T2 = 92; alpha = 23e-6;
deltaT = FaC(T2)-FaC(T1);                    % Utilizando la función FaC se calcula la
                                             % diferencia de temperatura en grados C.
a2 = a1 + alpha*a1*deltaT;                   % Se calcula la nueva longitud.
```

```
b2 = b1 + alpha*b1*deltaT;                                   Se calcula el nuevo ancho.
CambioArea = a2*b2 - a1*b1;                                  Se calcula la variación del área.
fprintf('La variación del area es de %6.5f metros cuadrados.',CambioArea)   Salida mostrada al usuario.
```

Tras ejecutar el fichero en la Ventana de Comandos se nos muestra la solución al problema:

```
>> Capitulo6Ejemplo2
El cambio en el area es de 0.01346 metros cuadrados.
```

6.7 Comparativa entre los ficheros script y los ficheros de función

Los estudiantes que se enfrentan por primera vez a MATLAB suelen tener problemas en entender exactamente las diferencias entre un fichero script y un fichero de función, ya que muchos de los problemas que se les proponen pueden ser resueltos utilizando ambos tipos de ficheros. Vamos a resumir las diferencias y similitudes más importantes entre ficheros script y ficheros de función:

- Ambos se pueden guardar con extensión .m (por esta razón muchas veces se denominan, de forma arbitraria, ficheros M).
- La primera línea en un fichero de función es la línea de definición de la función.
- Las variables de un fichero de función son locales. Las variables de un fichero script son reconocidas en la Ventana de Comandos.
- Los ficheros script pueden utilizar variables definidas en el espacio de trabajo.
- Los ficheros script contienen secuencias de comandos MATLAB (sentencias).
- Los ficheros de función pueden aceptar datos a través de argumentos de entrada, y pueden devolver datos como argumentos de salida.
- Cuando un fichero de función se almacena en disco, el nombre del fichero debe ser el mismo que el nombre de la función que contiene.

6.8 Funciones en línea

Las funciones en línea se pueden utilizar para construir funciones matemáticas sencillas, funciones matemáticas extensas y complicadas que requieren mucha programación, y subprogramas dentro de programas extensos. En general se usan en casos en los que un valor relativamente simple debe obtenerse varias veces durante la ejecución de un programa. Una función en línea, también denominada de línea o *inline* (en inglés), se define directamente dentro del código del programa (no en un fichero aparte, como en el caso de los ficheros de función), y se utiliza también dentro del mismo código. Las funciones en línea se pueden definir en cualquier parte de un código MATLAB. Estas funciones se crean con la siguiente sintaxis:

```
nombre = inline('expresión matematica en forma de cadena')
```

- La expresión matemática puede tener una o más variables independientes.
- Se puede usar cualquier letra, excepto i y j, como variables independientes en la expresión matemática.

- La expresión matemática puede incluir cualquier función predefinida MATLAB e incluso también las definidas por el usuario.
- La expresión se debe escribir según la dimensión de los argumentos de entrada y salida (operaciones elemento a elemento, o cálculos algebraicos).
- La expresión no puede incluir variables predefinidas.
- Una vez que se define la función, ésta se puede utilizar tecleando su nombre e introduciendo un valor (o varios) por cada argumento de entrada entre paréntesis (véase el ejemplo de más abajo).
- La función `inline` también se puede utilizar como argumento en otras funciones.

Función `inline` *con una variable independiente:*

Por ejemplo, la función: $f(x) = \dfrac{e^{x^2}}{\sqrt{x^2+5}}$ se puede definir en MATLAB (en la Ventana de Comandos) como una función de línea, con un argumento escalar de entrada x:

```
>> FA = inline('exp(x^2)/sqrt(x^2+5)')

FA =
     Inline function:
     FA(x) = exp(x^2)/sqrt(x^2+5)

>>
```

Como puede comprobarse, si no se teclea un punto y coma al final de la definición de la función, MATLAB responde visualizando la función. Una vez definida, ésta se puede utilizar para calcular resultados a partir de distintos valores de x, tal y como se muestra a continuación:

```
>> FA(2)

ans =
    18.1994

>> z = FA(3)

z =
    2.1656e+003

>>
```

Si por alguna razón se prevé que x pudiera contener más de un valor, es decir, ser un array, entonces los cálculos internos de la función se deberían cambiar para realizar operaciones elemento a elemento, como sigue:

```
>> FA = inline('exp(x.^2)./sqrt(x.^2+5)')

FA =
     Inline function:
     FA(x) = exp(x.^2)./sqrt(x.^2+5)

>> FA([1 0.5 2])                    Se utiliza un vector como argumento.

ans =
    1.1097    0.5604   18.19994
```

Función `inline` *con más de una variable independiente:*

Se pueden definir funciones que tienen dos o más variables independientes utilizando este otro formato más completo de `inline`:

```
nombre = inline('expresion matematica', 'arg1', 'arg2',
                                        'arg3', ..., 'argN')
```

Con este formato se definen el orden y el número de los argumentos de la función, aunque si no se indican los argumentos, MATLAB los extrae automáticamente en orden alfabético. Por ejemplo, la función $f(x, y) = 2x^2 - 4xy + y^2$ se puede definir en línea de la forma:

```
>> HA = inline('2*x^2-4*x*y+y^2')

HA =
     Inline function:
     HA(x,y) = 2*x^2-4*x*y+y^2
```

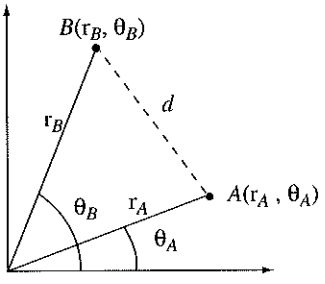
Después de su definición, la función se puede utilizar para distintos valores de x e y. Por ejemplo, HA(2,3) nos devuelve:

```
>> HA(2,3)

ans =
    -7
```

En el Problema de ejemplo 6.3 se muestra otro ejemplo de creación y manipulación de funciones en línea de más de un argumento o parámetro de entrada.

Problema de ejemplo 6.3: Distancia entre puntos en coordenadas polares

Escribir una función en línea que calcule la distancia entre dos puntos situados en un plano. La posición de los puntos viene dada en coordenadas polares. Utilizar la función definida para calcular la distancia entre el punto $A(2, \pi/6)$ y el punto $B(5, 3\pi/4)$.



Solución

La distancia entre dos puntos en coordenadas polares se puede calcular utilizando la ley de cosenos:

$$d = \sqrt{r_A^2 + r_B^2 - 2r_A r_B \cos(\theta_A - \theta_B)}$$

La fórmula para el cálculo de la distancia se introduce primero como función en línea de cuatro argumentos de entrada. Seguidamente se utiliza dicha función para calcular la distancia entre los puntos A y B.

```
d = inline('sqrt(rA^2+rB^2-2*rA*rB*cos(thetaB-thetaA))',...
   'rA','thetaA','rB','thetaB')
```

Se define el siguiente orden en los argumentos de entrada: `rA`, `thetaA`, `rB`, `thetaB`.

```
d =
    Inline function:
    d(rA,thetaA,rB,thetaB) = sqrt(rA^2+rB^2-2*rA*rB*cos(thetaB-thetaA))

>> DistAaB = d(2,pi/6,5,3*pi/4)

DistAaB =
    5.8461
```

Los argumentos se introducen en el orden establecido anteriormente, en la creación de la función en línea.

6.9 *El comando* `feval`

El comando `feval` (del inglés *function evaluate*, evaluar una función) evalúa el valor de una función para un valor (o valores) dado del argumento (o argumentos) de la función. Su sintaxis es:

```
variable = feval('nombre_funcion', valor_argumento)
```

El valor que retorna `feval` se puede asignar directamente a una variable. En caso de que el comando se ejecute sin argumentos, MATLAB visualiza `ans =` y el valor de la función.

- El nombre de la función debe introducirse en forma de cadena.
- La función puede ser una función MATLAB u otra cualquiera definida por el usuario.
- Se puede introducir más de un argumento de entrada, para ello los argumentos se separan por comas y se introducen en orden, justo después del nombre de la función.
- Se pueden establecer más de un argumento de salida. Para ello sólo es necesario introducir las variables de la parte izquierda de la igualdad entre corchetes, como ya se vio anteriormente para otro tipo de comandos.

Veamos un par de ejemplos de utilización de `feval` con funciones predefinidas MATLAB:

```
>> feval('sqrt',64)

ans =
    8

>> x = feval('sin',pi/6)

x =
    0.5000
```

El siguiente ejemplo muestra el uso del comando `feval` con la función definida por el usuario `prestamo`, ya presentada en secciones anteriores (Figura 6.2):

```
>> [M,T] = feval('prestamo',50000,3.9,10)

M =
    502.22

T =
    60266.47
```

Se calcula un préstamo de 50000 euros, a un interés del 3,9% durante 10 años.

Amortización mensual del préstamo.

Cantidad total pagada al final del préstamo.

Cuándo utilizar `feval`*:*

En los ejemplos vistos anteriormente, la salida producida por `feval` es la misma que si la función fuera utilizada directamente del modo tradicional ya estudiado. De hecho, el último ejemplo lo podríamos volver a escribir como:

```
>> [M,T] = prestamo(50000,3.9,10)

M =
   502.22

T =
   60266.47
```

que devuelve un resultado idéntico al visto anteriormente. Parece obvio que no es necesario utilizar `feval` para calcular el valor de una función cuando se puede utilizar la propia función.

El comando `feval` es útil en situaciones en las que se necesita calcular el valor de una función dentro de otra función, y es posible que esa función interna sea una diferente cada vez que se llama a la función externa. Para hacer esto se importa el nombre (cadena) de la función interna a través de los argumentos de entrada de la función externa, y después se utiliza dicha cadena como nombre de la función en el comando `feval`.

Por ejemplo, MATLAB posee una función predefinida llamada `fzero`. Esta función calcula los ceros (puntos de corte con el eje x) de una función de una variable. `fzero` se puede utilizar para resolver diferentes funciones. Para hacerlo se transfiere el nombre de la función que se va a resolver dentro de la función `fzero` mediante los argumentos de entrada. De esta forma `feval` se utiliza dentro del código de la función `fzero` para calcular la función que se desee en cada caso. La función `fzero` se explicará con más detalle en el Capítulo 10.

6.10 Ejemplos de aplicaciones con MATLAB

Problema de ejemplo 6.4: Crecimiento y decrecimiento exponencial

El crecimiento y decrecimiento exponencial de una cantidad se puede modelizar a partir de la expresión:

$$A(t) = A_0 e^{kt}$$

donde $A(t)$ y A_0 representan una magnitud en el instante t y en el instante 0, respectivamente, y k es una constante única específica para cada aplicación.

Escribir una función que utilice este modelo para predecir $A(t)$ (una magnitud en el instante t), conociendo los valores de $A(0)$ y de $A(t_1)$ en el instante t_1. Utilizar la siguiente línea de definición de la función: `At = expGD(A0, At1, t1, t)`, donde el argumento de salida `At` se corresponde con $A(t)$, y los argumentos de entrada `A0, At1, t1, t` se corresponden con A_0, $A(t_1)$, t_1 y t, respectivamente.

Utilizar la función en la Ventana de Comandos para operar sobre los dos casos de predicciones siguientes:

a) La población de México fue de 67 millones en el año 1980, y de 79 millones en 1986. Estimar la población en el año 2000.

b) La vida media de un material radiactivo es de 5,8 años. ¿Cuánto quedará de una muestra de 7 g de ese mismo material después de 30 años?

Solución

Para utilizar el modelo de crecimiento exponencial, el valor de la constante k debe ser calculado resolviendo k en términos de A_0, $A(t_1)$ y t_1:

$$k = \frac{1}{t_1} \ln \frac{A(t_1)}{A_0}$$

Una vez que k es conocido, el modelo se puede utilizar para estimar la población en cualquier año. Veamos la función MATLAB definida por el usuario para resolver ambos problemas:

```
function At = expGD(A0,At1,t1,t)          Línea de definición de la función.
%expGD calcula el crecimiento y decrecimiento exponencial
%Los argumentos de entrada son:
%A0: cantidad en el instante 0.
%At1: cantidad en el instante 1.
%t1: tiempo t1.
%t: tiempo t.
%Los argumentos de salida son :
%At: cantidad en el instante t.
k = log(At1/A0)/t1;                       Se calcula k.
At = A0*exp(k*t);          Se calcula A(t). Asignación del valor al argumento de salida de la función.
```

Una vez que se ha almacenado la función, ésta puede usarse en la Ventana de Comandos para resolver los dos casos propuestos. Para el caso a) $A_0 = 67$, $A(t_1) = 79$, $t_1 = 6$ y $t = 20$:

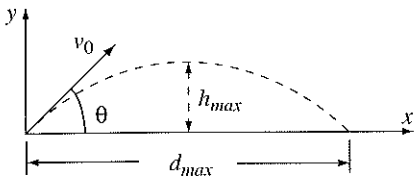
```
>> expGD(67,79,6,20)
ans =
   116.03          Estimación de la población en el año 2000.
```

Para el caso b) $A_0 = 7$, $A(t_1) = 3{,}5$ (ya que t_1 corresponde a la vida media, que es el tiempo requerido para el que el material se desintegre hasta la mitad de su cantidad inicial), $t_1 = 5{,}8$ y $t = 30$:

```
>> expGD(7,3.5,5.8,30)
ans =
    0.19          Cantidad de material después de 30 años.
```

Problema de ejemplo 6.5: Movimiento de un proyectil

Crear un fichero de función que calcule la trayectoria de un proyectil. Los parámetros de entrada de la función serán la velocidad inicial y el ángulo en que el proyectil fue disparado. Los argumentos de salida de la función serán la altura y la distancia máxima alcanzadas. Además, la función generará un gráfico de la trayectoria. Posteriormente, utilizar esta función para calcular la trayectoria de un proyectil que se dispara a una velocidad de 230 m/s con un ángulo de 39°.



Solución

El movimiento de un proyectil se puede analizar considerando las componentes horizontal y vertical. La velocidad inicial v_0 se puede calcular a partir de estas componentes:

$$v_{0x} = v_0\cos(\theta) \qquad \text{y} \qquad v_{0y} = v_0\text{seno}(\theta)$$

En la dirección vertical, la velocidad y la posición del proyectil vienen dadas por:

$$v_y = v_{0y} - gt \qquad \text{y} \qquad y = v_{0y}t - \frac{1}{2}gt^2$$

El tiempo que tarda el proyectil en alcanzar el punto más alto ($v_y = 0$) y la altura correspondiente vienen dados por:

$$t_{hmax} = \frac{v_{0y}}{g} \qquad \text{y} \qquad h_{max} = \frac{v_{0y}^2}{2g}$$

El tiempo total de vuelo es el doble del tiempo que utiliza el proyectil en alcanzar el punto más alto, $t_{tot} = 2t_{hmax}$. En la dirección horizontal, la velocidad es constante, y la posición del proyectil viene dada por:

$$x = v_{0x}t$$

En la notación de MATLAB el nombre de la función y sus argumentos se definen como: `[hmax,dmax] = trayectoria(v0,theta)`, de forma que el fichero de función correspondiente sería:

```
function [hmax,dmax] = trayectoria(v0,theta)
%La funcion trayectoria calcula la altura y la distancia maximas que alcanza un proyectil
%y representa graficamente la trayectoria
%Los argumentos de entrada son:
%v0: velocidad inicial en m/s
%theta: angulo en grados.
%Los argumentos de salida son:
%hmax: altura maxima en metros.
%dmax: distancia maxima en metros.
```

Línea de definición de la función.

```
g=9.81;
v0x=v0*cos(theta*pi/180);
v0y=v0*sin(theta*pi/180);
thmax=v0y/g;
hmax=v0y^2/(2*g);
ttot=2*thmax;
dmax=v0x*ttot;
%Creacion de un grafico de la trayectoria
tplot=linspace(0,ttot,200);
x=v0x*tplot;
y=v0y*tplot-0.5*g*tplot.^2;
plot(x,y)
xlabel('DISTANCIA (m)')
ylabel('ALTURA (m)')
title('TRAYECTORIA DE UN PROYECTIL')
```

Se crea un vector tiempo de 200 elementos.

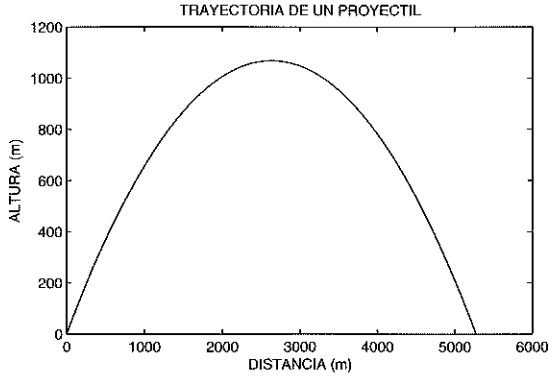
Se calculan las coordenadas x e y del proyectil para cada tiempo.

Se utilizan cálculos elemento a elemento.

Una vez guardada la función en disco, se utiliza la Ventana de Comandos para calcular los datos sobre un proyectil disparado a una velocidad de 230 m/s, con un ángulo de 39°.

```
>> [h d] = trayectoria(230,39)
h =
   1.0678e+003
d =
   5.2746e+003
```

A continuación se muestra la figura creada a partir del gráfico generado por la función:



6.11 Problemas

1. Escriba una función en MATLAB con dos argumentos de entrada y dos de salida. La función debe calcular la altura en pulgadas y la masa en libras de una persona a partir de su altura en centímetros y de su peso en kilogramos. Utilice la siguiente definición de función para el problema: `[in,lb] = SIaSTi(cm,kg)`. Los argumentos de entrada son la altura en centímetros y el peso en kilogramos, y los argumentos de salida son la altura en pulgadas y la masa en libras. Posteriormente utilice esta función en la Ventana de Comandos para:

   a) Determinar la altura en pulgadas y la masa en libras de una persona que mide 170 cm y pesa 70 kg.

   b) Determinar su propia altura y peso en pulgadas y libras, respectivamente.

2. Escriba una función MATLAB para la siguiente función matemática:

$$y(x) = 0{,}9x^4 - 12x^2 - 5x$$

   La entrada de la función será x, y la salida será y. Escriba la función de forma que x pueda ser un vector, y utilícela para:

   a) Calcular $y(-3)$ e $y(5)$.

   b) Representar gráficamente la función $y(x)$ para $-4 \leq x \leq 4$.

3. Escriba una función MATLAB para la siguiente función matemática:

$$r(\theta) = 2(1{,}1 - \text{sen}^2\theta)$$

   La entrada de la función será θ (en radianes) y la salida será r. Escriba la función de forma que θ pueda ser un vector, y utilícela para:

   a) Calcular $r(\pi/3)$ y $r(3\pi/2)$.

   b) Representar gráficamente (en coordenadas polares) $r(\theta)$ para $0 \leq \theta \leq 2\pi$.

4. Escriba una función MATLAB que calcule el máximo o mínimo local de una función cuadrática de la forma: $f(x) = ax^2 + bx + c$. Utilice la siguiente línea de definición de la función: `[x,y] = maxmin(a,b,c)`. Los argumentos de entrada son las constantes a, b y c, y los argumentos de salida son las coordenadas x e y del máximo o el mínimo de la función.

   Utilice la función para calcular el máximo o el mínimo de las siguientes funciones:

   a) $f(x) = 3x^2 - 18x + 48$

   b) $f(x) = -5x^2 + 10x - 3$

5. El valor P de una cuenta de ahorros, con un capital inicial P_0 y una tasa de interés anual r (en %) después de t años, viene dado por:

$$P = P_0\left(1 + \frac{r}{100}\right)^t$$

   Escriba una función que calcule el valor futuro de una cuenta de ahorros. Utilice para ello la siguiente línea de definición de función: `P = saval(P0,r,t)`. Las entradas de la función serán el capital inicial, la tasa de interés y el número de años. La salida será el valor de la cuenta a partir de los datos especificados en la entrada. Utilice posteriormente esta función para calcular el valor de un capital inicial de 10 000 €, a un interés anual del 6%, después de 13 años.

6. Escriba una función que convierta las unidades de un par de torsión (fuerza que causa la rotación de un objeto) de libras-pulgadas a newtons-metro. Utilice la siguiente línea de definición para la función: `Nm = lbintoNm(lbin)`. El argumento de entrada será el par en libras-pulgada, y el argumento de salida el par en newtons-metro. Utilice posteriormente esta función para convertir 500 libras-pulgada a newtons-metro.

7. Escriba una función que calcule los ángulos de un triángulo a partir de las longitudes de sus lados. Utilice para ello la siguiente línea de definición de función: `[alp,bet,gam] = triangulo(a,b,c)`. Utilice posteriormente esta función para calcular los siguientes triángulos:

   a) $a = 10$, $b = 15$, $c = 7$.

   b) $a = 6$, $b = 8$, $c = 10$.

   c) $a = 200$, $b = 75$, $c = 250$.

8. Escriba una función que calcule el vector unitario en la dirección de la recta que une dos puntos (A y B) en el espacio. Utilice la siguiente línea de definición de función: `n = unitvec(A,B)`. La entrada de la función serán dos vectores A y B, cada uno con tres elementos correspondientes a las coordenadas cartesianas de dichos puntos. La salida será un vector con tres componentes que representan las coordenadas del vector unitario en la dirección de A a B. Utilice posteriormente esta función para determinar los siguientes vectores unitarios:

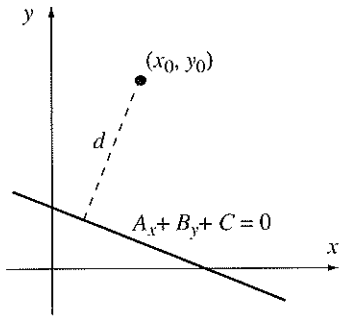
   a) En la dirección del punto (2, 6, 5) al punto (–10, 15, 9).

   b) En la dirección del punto (–10, 15, 9) al punto (2, 6, 5).

   c) En la dirección del punto (1, 1, 2) al punto (2, 1, 1).

9. La forma tradicional de la ecuación de la recta en el plano x-y es: $Ax + By + C = 0$. Además, cualquier punto queda determinado por sus coordenadas en dicho plano (x_0, y_0).

   Escriba una función MATLAB que calcule la distancia entre un punto y una recta en el plano x-y. Utilice para ello la siguiente definición de función: `d = DistPaL(x0,y0,A,B,C)`, donde los argumentos de entrada son las coordenadas del punto y las tres constantes de la ecuación de la recta. El argumento de salida será la distancia. Utilice posteriormente esta función para calcular la distancia en los siguientes casos:



   a) Punto: (2, –4), recta: $-2x + 3{,}5y - 6 = 0$.

   b) Punto: (11, 2): recta: $y = -2x + 6$, (observe que en este caso la ecuación de la recta no está representada de la forma tradicional expuesta anteriormente).

10. Escriba una función que calcule la nota final de un estudiante a partir de la nota de su examen final, sus dos exámenes parciales y de los cinco trabajos realizados durante el curso. Los exámenes parciales se puntúan de 0 a 100, y cada uno es un 20% de la nota final. El examen final tiene la misma escala de puntuación, y es un 40% de la nota final. Los trabajos, sin embargo, puntúan de 0 a 10, y todos ellos en conjunto representan un 20% de la nota final.

    La función debe tener la siguiente definición: `g = notasfinales(R)`, donde la entrada será una matriz `R` que contenga en cada fila las notas de cada estudiante. Además, por cada fila, se tendrán 8 columnas que representarán las notas de los trabajos (las cinco primeras), la nota de los dos exámenes parciales (las dos siguientes) y la nota del examen final (la última columna) de cada estudiante. La

salida de la función será un vector columna `g` con la nota final del curso. Cada fila de este vector será la nota final del estudiante cuyas notas se relacionan con la correspondiente la fila de la matriz `R`.

La función debe usarse para calcular las notas finales de cualquier número de estudiantes. Para el caso de un solo estudiante, la matriz `R` tendrá una sola fila. Aplique esta función en los siguientes casos:

a) Utilice la Ventana de Comandos para calcular la nota de un estudiante con las siguientes calificaciones: 10, 5, 8, 7, 9, 75, 87, 69.

b) Escriba un fichero script que pida al usuario las notas de los estudiantes y las almacene en un array (cada estudiante en una fila). El programa debe calcular seguidamente las notas finales utilizando la función `notasfinales`. Ejecute el fichero script en la Ventana de Comandos para calcular las notas finales de los siguientes cuatro estudiantes:
Estudiante A: 7, 9, 5, 8, 10, 90, 70, 85
Estudiante B: 6, 4, 7, 0, 7, 60, 71, 50
Estudiante C: 5, 9, 10, 3, 5, 45, 75, 80
Estudiante D: 8, 8, 7, 7, 9, 82, 81, 88

11. Cuando se conectan n resistencias en paralelo, su resistencia equivalente R_{Eq} viene determinada por:

$$\frac{1}{R_{Eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_n}$$

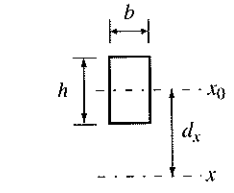
Escriba una función que calcule R_{Eq}. Utilice la siguiente definición: `REQ = req(R)`, donde la entrada será un vector en el cual cada elemento representa un valor de la resistencia, y la salida será el valor de la resistencia equivalente R_{Eq}. Utilice esta función para calcular la resistencia equivalente de las siguientes resistencias conectadas en paralelo: 50 Ω, 75 Ω, 300 Ω, 60 Ω, 500 Ω, 180 Ω y 200 Ω.

12. Escriba una función que proporcione un número entero aleatorio en un rango concreto especificado a partir de dos números. Utilice para ello la siguiente definición de función: `n = randint(a,b)`, donde los dos argumentos de entrada `a` y `b` son los números que determinan el rango, y la salida será el número aleatorio calculado `n`. Utilice posteriormente esta función en la Ventana de Comandos para:

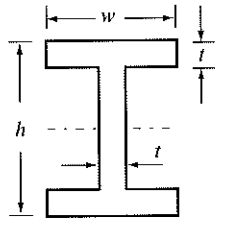
a) Generar un número aleatorio entre 1 y 49.

b) Generar un número aleatorio entre –35 y –2.

13. El momento de inercia superficial I_{x_0} de un rectángulo alrededor del eje x_0 que pasa por el centroide viene determinado por $I_{x_0} = \frac{1}{12}bh^3$. El momento de inercia alrededor del eje x paralelo a x_0 viene dado por $I_x = I_{x_0} + Ad_x^2$, donde A es el área del rectángulo, y d_x la distancia entre los dos ejes.

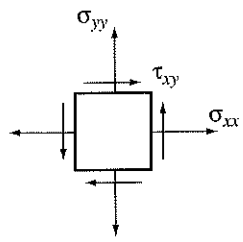


Escriba una función que calcule el momento de inercia superficial de una viga en forma de "I" alrededor del eje que pasa por su centroide (ver dibujo). Utilice para ello la siguiente definición de función: `I = Iviga(w,h,t)`. Las entradas de la función serán el ancho `w`, la altura `h` y el grosor `t` del nervio y las pestañas de la viga. (El momento de inercia del área compuesta se obtiene dividiendo el área en distintas partes y sumando el momento de inercia de cada una de ellas.)



Utiliza la función para calcular el momento de inercia de una viga en forma de "I" cuyas dimensiones son $w = 200$ mm, $h = 300$ mm y $t = 22$mm.

14. La representación bidimensional del estado de tensión en un punto de un material cargado queda definido por las tres componentes de la tensión σ_{xx}, σ_{yy}, τ_{xy}. Las tensiones normales máxima y mínima (tensiones principales) en el punto, σ_{max} y σ_{min}, se calculan a partir de las componentes de la tensión, de la forma:



$$\sigma_{\substack{max\\min}} = \frac{\sigma_{xx}+\sigma_{yy}}{2} \pm \sqrt{\left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)^2 + \tau_{xy}^2}$$

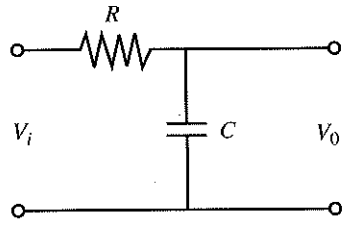
Escriba una función que calcule las tensiones principales a partir de las componentes de la tensión. Utilice para ello la siguiente línea de definición de función: `[Smax,Smin] = TensionPrincipal(Sxx,Syy,Sxy)`. Los argumentos de entrada serán las tres componentes de la tensión, y la salida las tensiones máxima y mínima.

Utilice posteriormente esta función para calcular las tensiones principales para los siguientes estados de tensión:

a) $\sigma_{xx} = 150$ MPa, $\sigma_{yy} = -40$ MPa y $\sigma_{xy} = 80$ MPa.

b) $\sigma_{xx} = -12$ ksi, $\sigma_{yy} = -16$ ksi y $\sigma_{xy} = -7$ ksi.

15. En un filtro paso-bajo (filtro que pasa señales de bajas frecuencias), la relación de voltajes viene determinada por:



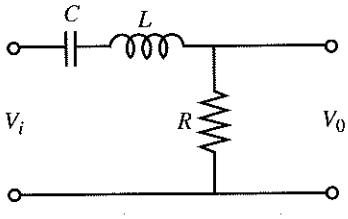
$$RV = \left|\frac{V_o}{V_i}\right| = \frac{1}{\sqrt{1+(\omega RC)^2}}$$

donde ω es la frecuencia de la señal de entrada.

Escriba una función que calcule la relación de voltajes. Utilice para ello la siguiente línea de definición: `RV = pasobajo(R,C,w)`. Los argumentos de entrada son el valor de la resistencia `R` en Ω (ohmios), la capacidad del condensador `C` en F (faradios) y la frecuencia `w` de la señal de entrada en rad/s. Diseñe la función de forma que `w` pueda ser un vector.

Escriba un fichero script que utilice la función `pasobajo` para generar un gráfico *RV* en función de ω, $10^{-2} \le \omega \le 10^6$ rad/s. El gráfico debe tener escala logarítmica en el eje horizontal (ω). Cuando se ejecute el fichero script, éste debe pedir al usuario que introduzca los valores de *R* y *C*. Etiquete los ejes convenientemente y ejecute el script para los valores $R = 1200\ \Omega$ y $C = 8\ \mu$F.

16. En un filtro paso-banda (filtro que pasa señales con frecuencias dentro de un cierto rango), la relación de voltajes viene determinada por:



$$RV = \left|\frac{V_o}{V_i}\right| = \frac{\omega RC}{\sqrt{1+(\omega^2 LC)^2+(\omega RC)^2}}$$

donde ω es la frecuencia de la señal de entrada.

Escriba una función que calcule la relación de voltajes. Utilice para ello la siguiente línea de definición: `RV = pasobanda(R,C,L,w)`. Los argumentos de entrada son el valor de la resis-

tencia `R` en Ω (ohmios), la capacidad del condensador `C` en F (faradios), el valor de la bobina (inductancia) `L` en H (henrios) y la frecuencia `w` de la señal de entrada en rad/s. Diseñe la función de forma que `w` pueda ser un vector.

Escriba un fichero script que utilice la función `pasobanda` para generar un gráfico *RV* en función de ω, $10^{-2} \leq \omega \leq 10^{7}$ rad/s. El gráfico debe tener escala logarítmica en el eje horizontal (ω). Cuando se ejecute el fichero script, éste debe pedir al usuario que introduzca los valores de *R*, *C* y *L*. Etiquete los ejes convenientemente y ejecute el script para los siguientes casos:

a) $R = 1100\ \Omega$, $C = 9\ \mu$F y $L = 7$ mH.

a) $R = 500\ \Omega$, $C = 300\ \mu$F y $L = 400$ mH.

Capítulo 7
Programación en MATLAB

Un programa de ordenador es una secuencia de comandos, también denominados instrucciones. En un programa sencillo los comandos se ejecutan uno detrás de otro en el orden en que son tecleados. Hasta ahora, todos los programas que se han presentado en este libro, tanto en ficheros script como en ficheros de función, son por lo general programas muy sencillos. Sin embargo, en muchas otras situaciones es necesario escribir programas más complejos cuyas instrucciones no tienen que ejecutarse en el mismo orden en el que se escribieron, o que se ejecuten distintos tipos de instrucciones o grupos de instrucciones en función del valor de una serie de variables del programa. Por ejemplo, un programa que calcule el coste de los envíos de paquetes en una empresa puede utilizar diferente expresiones matemáticas que calculan el coste dependiendo del tamaño y peso del paquete, del contenido (los libros, por ejemplo, son más baratos de enviar) y del tipo de servicio (correo aéreo, por carretera, etc.) En otras situaciones puede ser necesario repetir una secuencia de comandos varias veces dentro de un programa. Por ejemplo, en programas que resuelven ecuaciones numéricamente los cálculos se repiten hasta que el error del resultado sea menor que un cierto valor previamente establecido.

MATLAB proporciona diferentes instrucciones que se pueden utilizar para controlar el flujo de un programa. Las sentencias condicionales (Sección 7.2) y la estructura de programación `switch` (Sección 7.3) permiten saltarse comandos o ejecutar grupos específicos de comandos en diferentes situaciones. Por otro lado, las sentencias de iteración `for` y `while` (Sección 7.4) permiten repetir secuencias de comandos varias veces seguidas.

Parece obvio que cambiar el flujo de un programa requiere algún tipo de proceso de toma de decisiones dentro del programa. El ordenador debe decidir si ejecuta el siguiente comando o pasa por alto uno o varios comandos y continuar en otra línea de código distinta. El programa toma estas decisiones comparando los valores de las variables. Esto se lleva a cabo mediante operadores lógicos y relacionales, que serán explicados en la Sección 7.1.

Cabe decir también que los ficheros de función (explicados en el Capítulo 6) se pueden utilizar para programar. Un fichero de función es un subprograma. Cuando un programa llega a una línea con comandos que llaman a una función, entonces ésta es invocada, le proporciona una entrada, y "espera" a que la función termine de ejecutarse para devolver el resultado de salida. La función realiza los cálculos y transfiere el o los resultados al programa que la llamó, el cual continúa ejecutando la siguiente instrucción.

7.1 Operadores relacionales y lógicos

Un operador relacional compara dos números determinando si el enunciado de la comparación (por ejemplo, 5 < 8) es verdadero o falso. Si es verdadero, el valor asignado será 1, mientras que si es falso, el resultado será 0. Un operador lógico examina la veracidad o falsedad de los enunciados y devuelve un resultado verdadero (1) o falso (0) según un operador concreto. Por ejemplo, el operador lógico Y (AND) devuelve 1 si dos sentencias son verdaderas. Los operadores lógicos y relacionales se pueden utilizar en expresiones matemáticas y, como veremos en este capítulo, en combinación con otros comandos para llevar a cabo decisiones sobre el control de flujo de un programa MATLAB.

Operadores relacionales:

Los operadores relacionales en MATLAB son:

Operador relacional	Descripción
<	Menor que.
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
==	Igual a.
~=	Distinto de.

Observe que el operador "igual a" se representa por dos signos = (sin espacio de separación entre ellos), ya que el signo = sencillo es el operador de asignación visto en capítulos anteriores. De la misma forma, los otros operadores relacionales que están compuestos de dos símbolos (<=, >=, ~=) no deben llevar un espacio de separación entre ellos (se analizan en conjunto los dos).

- Los operadores relacionales se utilizan como operadores aritméticos dentro de expresiones matemáticas. El resultado de estos operadores se puede usar en otras operaciones matemáticas, para acceder a los elementos de un array y, junto con otros comandos MATLAB (por ejemplo, el comando `if`), para controlar el flujo de un programa.
- Cuando se comparan dos números, el resultado es 1 (valor lógico verdadero) si la comparación, en función del operador relacional, es verdadera, y 0 (valor lógico falso) si la comparación es falsa.
- Si se comparan dos escalares, el resultado será también un escalar que tomará el valor 1 ó 0. Si se comparan dos arrays (sólo si tienen el mismo tamaño), la comparación se llevará a cabo *elemento a elemento*, y el resultado será también un array del mismo tamaño que los anteriores. Éste contendrá ceros y unos como salida, resultado de la comparación de cada uno de los elementos, dos a dos, de los dos arrays de entrada.
- Si se compara un escalar con un array, el escalar se comparará con todos los elementos del array, de forma que el resultado será un array lógico, con unos y ceros, según el resultado de la comparación en cada posición del array.

Veamos algunos ejemplos:

```
>> 5 > 8

ans =
   0

>> a = 5 < 10

a =
   1

>> y = (6<10)+(7>8)+(5*3==60/4)

y =
   2

>> b = [15 6 9 4 11 7 14]; c = [8 20 9 2 19 7 10];

>> d = c >= b

d =
   0   1   1   0   1   1   0

>> b == c

ans =
   0   0   1   0   0   1   0

>> b ~= c

ans =
   1   1   0   1   1   0   1

>> f = b-c > 0

f =
   1   0   0   1   0   0   1

>> A = [2 9 4; -3 5 2; 6 7 -1]

A =
    2   9   4
   -3   5   2
    6   7  -1

>> B = A <= 2

B =
   1   0   0
   1   0   1
   0   0   1
```

Comprueba si 5 es mayor que 8.

Como la comparación es falsa (5 no es mayor que 8), la respuesta es 0 (falso).

Comprueba si 5 es menor que 10, y almacena el resultado de la comparación en a.

Como la comparación es verdadera (5 es menor que 10), el valor asignado a la variable a será 1 (verdadero).

Utilización de operadores relacionales en expresiones matemáticas.

Es igual a 1, ya que 6 es menor que 10.

Es igual a 0, ya que 7 no es mayor que 8.

Es igual a 1, ya que 5*3 es igual a 60/4.

Se suman los valores verdaderos (unos), por tanto el valor final que contendrá y será 2.

Se definen dos vectores, b y c.

Se comprueba si los elementos de c son mayores o iguales que los elementos de b. El resultado se guarda en d.

Se asigna un 1 en cada posición donde un elemento de c es mayor o igual que un elemento de b.

Comprueba si los elementos de b son iguales a los elementos de c.

Comprueba si los elementos de b son diferentes de los elementos de c.

Resta c a b y después comprueba qué elementos son mayores que 0.

Define una matriz A de 3 × 3

Comprueba si los elementos de A son menores o iguales que 2. Asigna el resultado a la matriz B.

- El resultado de una operación relacional con vectores será un vector con unos y ceros, también llamado vector lógico. Estos vectores se pueden utilizar para acceder a los elementos de un array. Cuando se utiliza un vector lógico para este propósito, se extraen del vector donde se aplica los elementos cuyas posiciones se corresponden con unos en el vector lógico. Por ejemplo:

```
>> r = [8 12 9 4 23 19 10]
r =
   8   12   9   4   23   19   10
>> s = r <= 10
s =
   1   0   1   1   0   0   1
>> t = r(s)
t =
   8   9   4   10
>> w = r(r<=10)
w =
   8   9   4   10
```

Se define un vector `r`.

Comprueba qué elementos de `r` son menores o iguales que 10.

Se genera un vector lógico `s` con unos en las posiciones donde los elementos de `r` son menores o iguales que 10, y con ceros donde esta condición no se cumple.

Se utiliza `s` para acceder a los elementos de `r`. El resultado se almacena en un nuevo vector `t`.

El vector `t` contiene los elementos de `r` en cuya posición `s` contenía unos.

Aquí se detalla cómo todo lo anterior se puede realizar en un solo paso, obteniendo el mismo resultado.

- Los vectores numéricos y los arrays con ceros y unos no son comparables a los arrays lógicos. Los vectores numéricos no se pueden utilizar para acceder a los elementos de otros vectores. Sin embargo, los arrays lógicos sí pueden ser utilizados en operaciones aritméticas. Una vez que se utiliza un array lógico en una operación aritmética, éste cambia su estatus al de array numérico.
- En una expresión matemática que incluya operadores relacionales y aritméticos, las operaciones aritméticas (+, –, *, /, \) tienen mayor precedencia que los operadores relacionales. Los operadores relacionales tienen entre sí igual precedencia y se evalúan de izquierda a derecha. Se pueden utilizar paréntesis para cambiar este orden de precedencia. Veamos algunos ejemplos:

```
>> 3 + 4 < 16/2
ans =
   1
>> 3 + (4 < 16)/2
ans =
   3.5000
```

+ y / se ejecutan primero.

La respuesta es 1, ya que 7 < 8 es verdadero.

4 < 16 se ejecuta primero. El resultado es 1, ya que es verdadero.

Se obtiene 3,5 de operar 3 + 1/2.

Operadores lógicos:

Los operadores lógicos en MATLAB son:

Tabla 7.1: Operadores lógicos predefinidos en MATLAB.

Operador lógico	Nombre	Descripción
& Ejemplo: A&B	Y (AND)	A y B. Funciona con dos operandos. El resultado es verdadero si ambos son verdaderos, en otro caso el resultado es falso (0).
| Ejemplo: A|B	O (OR)	A o B. Funciona con dos operandos. El resultado es verdadero si alguno de los dos es verdadero, en otro caso (los dos son falsos) el resultado es falso (0).
~ Ejemplo: ~A	NO (NOT)	No A. Funciona con un operando. Da la negación del operando, es decir, verdadero (1) si A es falso, y falso (cero) si A es verdadero.

- Los operadores lógicos funcionan con números. Un número distinto de cero es verdadero, y un cero es siempre falso.
- Los operadores lógicos (al igual que los operadores relacionales) se pueden utilizar como operadores aritméticos dentro de expresiones matemáticas. El resultado de estos operadores se puede usar en otras operaciones matemáticas y en el acceso a elementos de un array, y junto con otros comandos MATLAB (por ejemplo el comando `if`) para controlar el flujo de un programa.
- Los operadores lógicos (al igual que los relacionales) se pueden utilizar con escalares y arrays.
- Los operadores lógicos Y y O pueden trabajar con operandos escalares, arrays y con un array y un escalar. Si ambos operandos son escalares, el resultado será 1 ó 0. Si ambos son arrays (deben ser del mismo tamaño) el resultado se computará *elemento a elemento*, y será otro array del mismo tamaño. Cuando un operando es escalar y el otro es un array, la operación lógica se realiza entre el escalar y cada uno de los elementos del array; el resultado es un array del mismo tamaño con ceros y unos como elementos.
- La operación lógica NO está pensada para un solo operando. Cuando se utiliza con un escalar el resultado será 1 ó 0, mientras que si se utiliza con un array el resultado será otro array del mismo tamaño, con un uno en las posiciones que tengan valores iguales a cero, y cero en las posiciones donde el array tenga valores distintos de cero.

  Veamos algunos ejemplos:

```
>> 3&7                                              (3 Y 7.)
ans =
   1
                  3 y 7 son números distintos de 0, luego se consideran verdaderos, por lo tanto el resultado de la operación lógica es 1.
>> a = 5|0                                          (5 O 0 (se asigna a la variable a).)
a =
   1
                  El resultado es 1 porque al menos uno de los dos números es verdadero (distinto de cero).
>> ~25                                              (NO 25.)
ans =
   0
                  El resultado es 0 (falso), ya que 25, al ser un valor distinto de 0, se considera verdadero, y el opuesto es, por tanto, falso.
```

```
>> t = 25*((12&0)+(~0)+(0|5)
t =
   50
>> x = [9 3 0 11 0 15]; y = [2 0 13 -11 0 4];
>> x&y
ans =
   1   0   0   1   0   1
>> a = x|y
ans =
   1   1   1   1   0   1
>> ~(x+y)
ans =
   0   0   0   1   1   0;
```

Utilización de operadores lógicos en expresiones matemáticas.

Se definen dos vectores x e y.

El resultado de esta operación es un vector con unos en cada posición donde los elementos de x e y sean verdaderos (valores distintos de cero), y ceros en el resto de las posiciones donde esto no se cumple.

La salida es un vector con unos en cada posición donde los elementos de x o y sean verdaderos (valores distintos de cero), y ceros en el resto de posiciones donde esto no se cumple.

La salida es un vector con cero en cada posición donde el resultado de x + y sea verdadero (valores distintos de cero), y unos en cada posición donde el resultado de x + y sea falso (elementos iguales a 0).

Orden de precedencia:

Los operadores aritméticos, relacionales y lógicos se pueden combinar en expresiones matemáticas a gusto del usuario, siempre que éstas estén bien formadas. Cuando una expresión tiene combinaciones de este tipo, se debe tener en cuenta el siguiente orden de precedencia establecido por MATLAB:

Orden de precedencia	Operación
1 (la mayor)	Paréntesis (si hay paréntesis anidados, el más interno tiene mayor precedencia).
2	Exponenciación.
3	Operación lógica NO (~).
4	Multiplicación, división.
5	Suma, resta.
6	Operadores relacionales (>, <, >=, <=, ==, ~=).
7	Operación lógica Y (&).
8 (la menor)	Operación lógica O (|).

Si dos o más operaciones tienen la misma precedencia, la expresión se ejecutará en orden de izquierda a derecha.

Es importante destacar que el orden enumerado anteriormente se utiliza a partir de la versión 6 de MATLAB. Las versiones anteriores de MATLAB tienen un orden de precedencia ligeramente diferente (el operador & no tiene precedencia sobre el operador |), por tanto el usuario debe estar atento a estas variaciones en función de la versión de MATLAB que utilice. No obstante, estos problemas de compatibilidad se pueden solucionar utilizando paréntesis, incluso cuando estos no sean necesarios.

A continuación se muestran algunos ejemplos de expresiones que incluyen operadores aritméticos, relacionales y lógicos:

```
>> x = -2; y = 5;
>> -5 < x < -1
ans =
   0
>> -5 < x & x < -1
ans =
   1
>> ~(y<7)
ans =
   0
>> ~y<7
ans =
   1
>> ~((y>=8) | (x<-1))
ans =
   0
>> ~(y>=8) | (x<-1)
ans =
   1
```

Se definen dos variables x e y.

Esta desigualdad es correcta matemáticamente. La respuesta de MATLAB, sin embargo, es 0 (falso). MATLAB ejecuta la expresión de izquierda a derecha. –5 < x es verdadero (=1) y por lo tanto 1 < –1 es falso (0).

La sentencia matemática correcta para escribir la desigualdad anterior se construye utilizando el operador &. La desigualdad se ejecuta primero. Como ambos miembros son ahora 1 (verdadero), la respuesta es 1.

y < 7 se ejecuta primero, dando verdadero (1), por tanto ~1 es falso (0).

~y se ejecuta primero. y es verdadero (1), ya que es un valor distinto de cero, ~1 es 0, y 0 < 7 es verdadero (1).

y >= 8 (falso) y x < –1 (verdadero) se ejecutan primero. El operador O se ejecuta después (verdadero). ~ se ejecuta al final, dando como resultado falso (0).

y >= 8 (falso), y x < –1 (verdadero) se ejecutan primero. La negación de (y >= 8) se ejecuta después (verdadero). El operador O se ejecuta al final, dando como resultado verdadero (1).

Funciones lógicas predefinidas:

MATLAB posee funciones que producen un resultado equivalente a los operadores lógicos vistos anteriormente. Estas funciones son:

`and(A,B)`	Equivale a `A&B`
`or(A,B)`	Equivale a `A|B`
`not(A)`	Equivale a `~A`

Además, MATLAB posee otras funciones lógicas predefinidas, algunas de las cuales se describen en la tabla siguiente (Tabla 7.2):

Tabla 7.2: Funciones lógicas predefinidas en MATLAB.

Función	Descripción	Ejemplo

`xor(a,b)`	O exclusiva. Devuelve 1 (verdadero) si uno de los operandos es verdadero y el otro es falso.	`>> xor(7,0)`
		`ans =`
		`1`
		`>> xor(7,-5)`
		`ans =`
		`0`

Tabla 7.2: Funciones lógicas predefinidas en MATLAB (*Continuación*).

Función	Descripción	Ejemplo
`all(A)`	Devuelve 1 (verdadero) si todos los elementos del vector `A` son verdaderos (valores distintos de cero). Por el contrario, devuelve 0 si uno o más elementos son falsos (cero). Si `A` es una matriz, el operador trata las columnas como vectores, devolviendo un vector con unos y ceros.	>> A = [6 2 15 9 7 11]; >> all(A) ans = 1 >> B = [6 2 15 9 0 11]; >> all(B) ans = 0
`any(A)`	Devuelve 1 (verdadero) si algún elemento de `A` es verdadero (valor distinto de cero). Por el contrario, devuelve cero (falso) si todos los elementos son cero (falso). Si `A` es una matriz, el operador trata las columnas como vectores, devolviendo un vector con unos y ceros.	>> A = [6 0 15 0 0 11]; >> any(A) ans = 1 >> B = [0 0 0 0 0 0]; >> any(B) ans = 0
`find(A)` `find(A>d)`	Si `A` es un vector, devuelve los índices de los elementos distintos de cero. Si `A` es un vector, devuelve la dirección de los elementos que son mayores que `d` (se puede utilizar cualquier otro operador relacional).	>> A = [0 9 4 3 7 0 0 1 8]; >> find(A) ans = 2 3 4 5 8 9 >> find(A>4) ans = 2 5 9

A continuación se muestra la tabla de verdad de los cuatro operadores lógicos vistos hasta ahora (`y`, `o`, `no`, `xor`) (Tabla 7.3):

Tabla 7.3: Operadores lógicos.

Entrada		Salida				
A	B	Y A&B	O A|B	XOR (A,B)	NO ~A	NO ~B
falso	falso	falso	falso	falso	verdadero	verdadero
falso	verdadero	falso	verdadero	verdadero	verdadero	falso
verdadero	falso	falso	verdadero	verdadero	falso	verdadero
verdadero	verdadero	verdadero	verdadero	falso	falso	falso

Problema de ejemplo 7.1: Análisis de datos de temperaturas

Durante el mes de abril de 2002 se recogieron las siguientes temperaturas máximas (en °F) en Washington DC: 58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 91 93 89 91 80 59 69 56 64 63 66 64 74 63 69 (datos del Ministerio Nacional de Meteorología y Oceanografía de Estados Unidos). Utilizar operadores relacionales y lógicos para determinar:

a) El número de días en los que la temperatura fue superior a 75 °F.

b) El número de días en los que la temperatura estuvo situada entre 65 °F y 80 °F.

c) Los días del mes (la fecha) en los que la temperatura estuvo situada entre 50 °F y 60 °F.

Solución

En el fichero script que se muestra a continuación se introducen las temperaturas en un vector. Posteriormente se utilizan operadores relacionales y lógicos para analizar los datos.

```
T = [58 73 73 53 50 48 56 73 73 66 69 63 74 82 84
  91 93 89 91 80 59 69 56 64 63 66 64 74 63 69];
Tencima75 = T >= 75;
NdiasTencima75 = sum(Tencima75)
Tentre65y80 = (T>=65)&(T<=80);
NdiasTentre65y80=sum(Tentre65y80)
fechasTentre50y60 = find((T>=50)&(T<=60))
```

Se crea un vector con unos donde la temperatura es superior a 75 °F.

Se suman todos los unos del vector anterior.

Se crea un vector con unos donde `T` >= 65 y `T` <= 80.

Se suman todos los unos del vector anterior.

La función `find` devuelve la dirección de los elementos en `T` que tienen valores entre 50 y 60 °F.

El fichero script, que se guarda con el nombre Exp7_1, produce la siguiente salida cuando se ejecuta:

```
>> Exp7_1
NdiasTencima75 =
   7
NdiasTentre65y80 =
   12
fechasTentre50y60 =
   1   4   5   7   2   23
```

Durante 7 días las temperaturas se mantuvieron por encima de los 75 °F.

Durante 12 días las temperaturas se mantuvieron entre 65 y 80 °F.

Días del mes en los que la temperatura se mantuvo entre 50 y 60 °F.

7.2 Sentencias condicionales

Una sentencia condicional es una instrucción MATLAB que permite tomar decisiones sobre si se ejecuta un grupo de comandos que cumplen una condición o, por el contrario, omitirlos. En una sentencia condicional se evalúa una expresión condicional. Si la expresión es verdadera, el grupo de comandos se ejecutará. Si la expresión es falsa, MATLAB no ejecutará (saltará) el grupo de comandos en cuestión. La forma básica de una sentencia condicional es la siguiente:

```
if expresion_condicional_consistente_en_operaciones_relacionales_y_logicas.
```

Ejemplos:

```
if a < b
if c >= 5
if a == b
if a ~= 0
if (d<h) & (x>7)
if (x~=13) | (y<0)
```

Todas estas variables deben tener valores asignados.

- Las sentencias condicionales pueden ser parte de un programa escrito en forma de fichero script o fichero de función.
- Como se verá a continuación, cada sentencia `if` (si) debe ir acompañada de la sentencia de finalización `end` (fin).

La sentencia condicional `if` se utiliza habitualmente en estructuras de tipo `if-end`, `if-else-end`, e `if-elseif-else-end`, que se describirán a continuación.

7.2.1 *La estructura* `if-end`

La estructura condicional `if-end` se ve de forma esquemática en la Figura 7.1, que muestra cómo se introducen estos comandos en un programa, y, además, un diagrama de flujo simbólico que explica el funcionamiento (flujo del programa) de este tipo de sentencias. Cuando el programa se ejecuta y

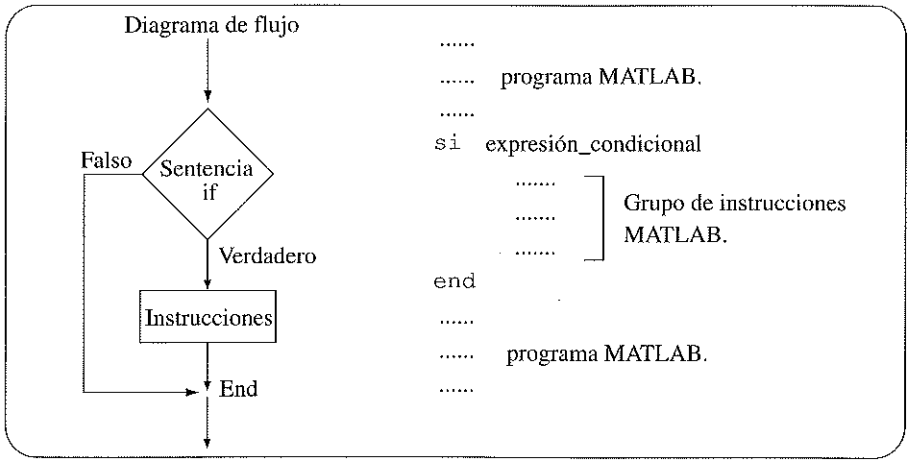


Figura 7.1: Estructura de la sentencia condicional `if-end`.

alcanza la sentencia `if`, éste evalúa la expresión condicional. Si la expresión resulta verdadera (1), el programa continúa hacia abajo ejecutando las siguientes instrucciones que siguen a la sentencia `if`, hasta alcanzar la sentencia `end`. Por el contrario, si la expresión condicional resulta falsa (0) al ser evaluada, el programa salta el bloque de instrucciones que hay entre la sentencias `if` y `end`, y continúa con la ejecución de los comandos que están después de `end`.

Las palabras reservadas `if` y `end` aparecen de color azul en pantalla y las instrucciones que se encuentran entre estas dos palabras se indentan (se tabulan hacia dentro) automáticamente para que el programa sea más fácil de leer y seguir. En el Problema de ejemplo 7.2 se muestra el uso de esta estructura `if-end`.

Problema de ejemplo 7.2: Cálculo del salario de un trabajador

Un trabajador cobra un determinado salario por hora hasta 40 horas semanales. Además, si hace horas extras (más de 40 horas semanales), esas horas se las pagan un 50% más. Escribir un programa (script) que calcule la paga semanal del trabajador. El programa pedirá al usuario que introduzca el total de horas y el salario por hora. El programa visualizará la paga semanal del trabajador.

Solución

El programa (script) primero calculará la paga semanal multiplicando el salario por el número de horas. Seguidamente, mediante una sentencia `if`, comprobará si el número de horas es mayor que 40. En caso afirmativo, ejecutará la siguiente línea para calcular la paga correspondiente a las horas extras. En caso de que sea falso el programa salta estas líneas y ejecuta el siguiente comando después del `end`.

```
t = input('Introduzca el numero de horas trabajadas');
h = input('Introduzca el salario por hora en euros');
PagaSemanal = t*h;
if (t>40)
   PagaSemanal = PagaSemanal+(t-40)*0.5*h;
end
fprintf('La paga semanal del trabajador es de %5.2f Euros', PagaSemanal);
```

A continuación se muestra la ejecución de este programa script (almacenado como PagaSemanal) para dos casos distintos:

```
>> PagaSemanal
Introduzca el numero de horas trabajadas 35
Introduzca el salario por hora en euros 8
La paga semanal del trabajador es de 280.00 Euros
>> PagaSemanal
Introduzca el numero de horas trabajadas 50
Introduzca el salario por hora en euros 10
La paga semanal del trabajador es de 550.00 Euros
```

7.2.2 *La estructura* `if-else-end`

Esta estructura proporciona el mecanismo necesario para ejecutar uno entre dos grupos de comandos posibles en función de la evaluación de una condición lógica. Se trata, pues, de discriminar entre dos opciones posibles, al contrario que la estructura estudiada en la sección anterior donde sólo era posible ejecutar o no un grupo de comandos. En la Figura 7.2 se muestra esquemáticamente cómo funciona esta estructura condicional. La primera línea es una sentencia `if` con una expresión condicional. Si esta condición es cierta, el programa ejecuta el Grupo 1 de instrucciones que se encuentra entre el comando `if` y el comando `else`, y luego salta hasta el comando `end`. Si la expresión condicional es falsa, el programa salta al comando `else` y ejecuta el Grupo 2 de instrucciones, es decir, las instrucciones que se encuentran entre el comando `else` y el comando `end`.

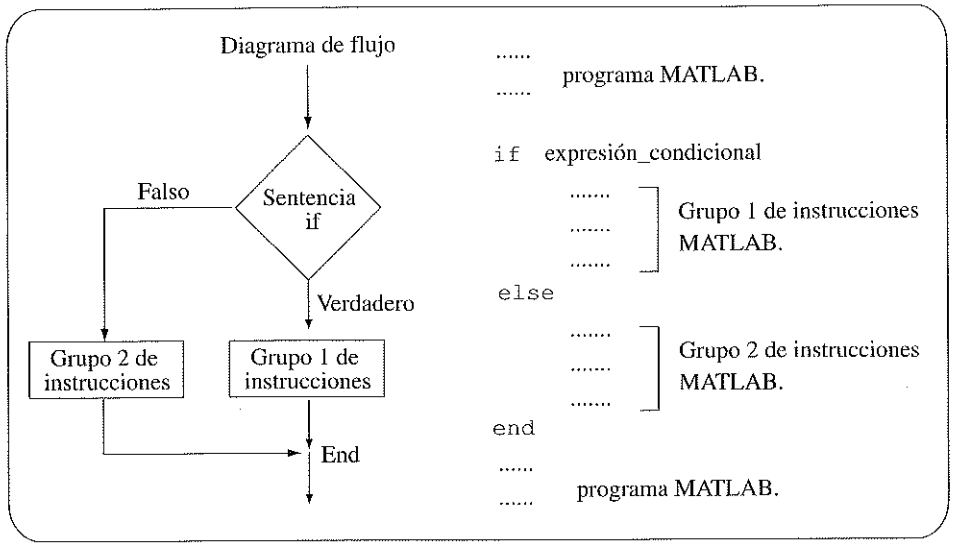
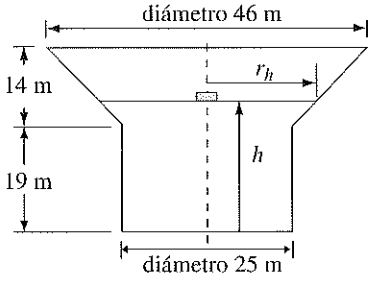


Figura 7.2: Estructura de la sentencia condicional `if-else-end`.

A continuación se muestra un ejemplo de utilización de este tipo de estructura en forma de fichero de función.

Problema de ejemplo 7.3: Nivel de un depósito de agua

Una torre para el almacenamiento de agua tiene la geometría que se muestra en la figura adjunta (la parte inferior es un cilindro, y la superior un cono truncado invertido). Dentro del depósito hay una boya que indica el nivel del agua. Escribir una función MATLAB que calcule el volumen de agua dentro del depósito a partir de la posición (altura h) de la boya. La entrada de la función será el valor de la altura h en metros, y la salida será el volumen que ocupa el agua en metros cúbicos.



Solución

Para $0 \leq h \leq 19$ m el volumen de agua se puede calcular a partir del volumen de un cilindro de altura h:

$$V = \pi 12{,}5^2\, h.$$

Para $19 < h \leq 33$ m el volumen de agua se calcula sumando el volumen del cilindro, con altura $h = 19$ m, y el volumen de agua en el cono:

$$V = \pi 12{,}5^2 \cdot 19 + \frac{1}{3}\pi(h-19)(12{,}5^2 + 12{,}5 \cdot r_h + r_h^2)$$

donde r_h es: $12{,}5 + \frac{10{,}5}{14}(h-19)$.

A continuación se muestra el código de la función `v = voluagua(h)` que resuelve el problema:

```
function v = voluagua(h)
%voluagua calcula el volumen de agua de un deposito.
%La entrada es el nivel del agua en metros.
%La salida es el volumen de agua en metros cubicos.
if h<=19
   v = pi*12.5^2*h;
else
   rh = 12.5 + 10.5*(h-19)/14;
   v = pi*12.5^2*19 + pi*(h-19)*(12.5^2 + 12.5*rh + rh^2)/3;
end
```

A continuación se muestran dos ejemplos de ejecución de la función en la Ventana de Comandos:

```
>> voluagua(8)
ans =
   3.9270e+003
>> VOL = voluagua(25.7)
VOL =
   1.4115e+004
```

7.2.3 *La estructura* `if-elseif-else-end`

La estructura `if-elseif-else-end` se muestra en la Figura 7.3. Esta estructura incluye dos sentencias condicionales (`if` y `elseif`), lo que hace posible ejecutar uno de entre tres grupos de instrucciones diferentes. La primera línea es una sentencia `if` con una expresión condicional. Si la expresión es verdadera, el programa ejecutará el Grupo 1 de instrucciones que se encuentra entre la sentencia `if`

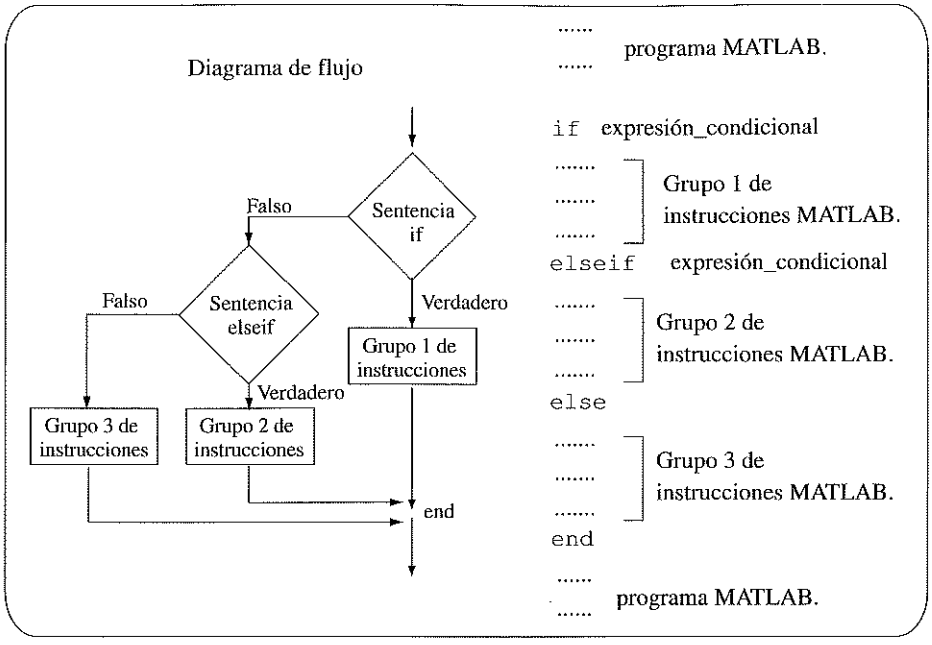


Figura 7.3: Estructura de la sentencia condicional `if-elseif-else-end`.

y `elseif`, y posteriormente saltará hasta el `end` (sin ejecutar el resto de los grupos). Si la condición evaluada es falsa, el programa saltará hasta la sentencia `elseif`. Si la condición que se evalúa en `elseif` es verdadera, entonces se ejecutará el Grupo 2 de instrucciones que se encuentra entre el comando `elseif` y el comando `else`, y posteriormente saltará hasta el `end`. Por el contrario, si la condición de `elseif` resulta falsa, el programa saltará al `else` y ejecutará el Grupo 3 de instrucciones que se encuentra entre el comando `else` y el comando `end`.

Es necesario destacar que se puede introducir más de una sentencia `elseif` y por tanto asociar nuevos grupos de instrucciones que se ejecutarán o no dependiendo de las condiciones establecidas. Además, la sentencia `else` es opcional. Esto significa que en el caso de que haya varios `elseif` y ningún `else`, si alguna condición de los `elseif` es verdadera, los comandos será ejecutados, pero en otro caso (todas las condiciones de los `elseif` son falsas) no se ejecutará ni se realizará ninguna operación.

7.3 *La sentencia* `switch-case`

La sentencia `switch-case` es otra estructura para el control de flujo de un programa. Esta sentencia proporciona un mecanismo para escoger y ejecutar un grupo de instrucciones de entre varios grupos posibles. La estructura de esta sentencia se muestra en la Figura 7.4.

- La primera línea en la sentencia `switch-case` tiene la forma:

`switch` expresión

donde la expresión puede ser un escalar o una cadena. Normalmente consiste en una variable a la que se le ha asignado un valor escalar o una cadena. No obstante, también puede consistir en una expresión matemática evaluable.

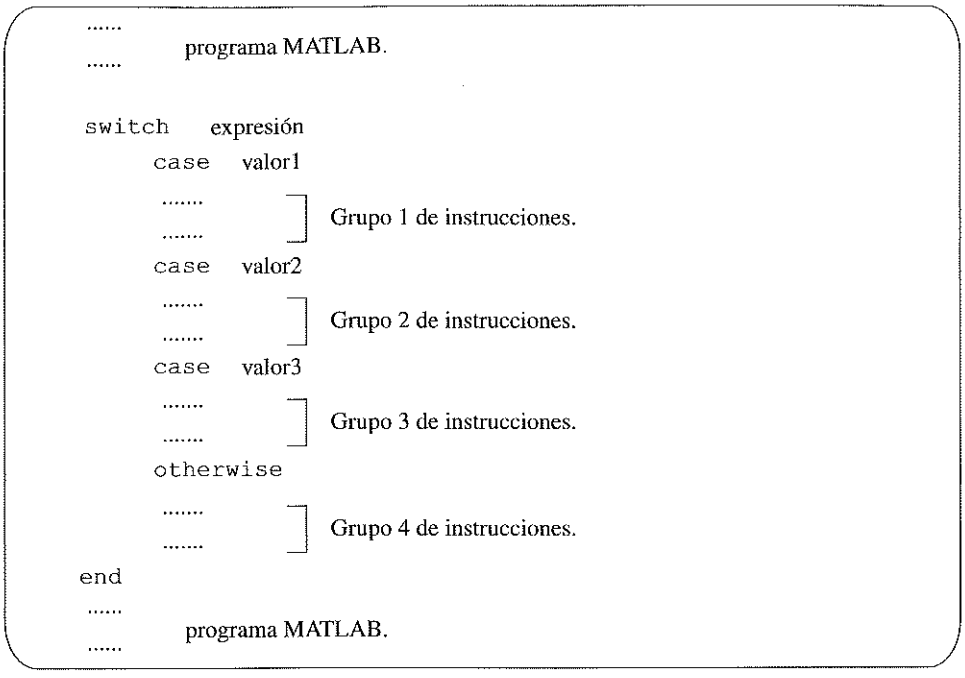


Figura 7.4: Estructura de la sentencia `switch-case`.

- A continuación del comando `switch` hay uno o varios comandos de tipo `case`, seguidos por valores que pueden ser escalares o cadenas (valor1, valor2, etc.). Cada comando `case` está asociado a la serie de instrucciones que se escriben a continuación de él.
- Después del último comando `case` puede haber, opcionalmente, un comando `otherwise`, seguido también de un grupo de instrucciones.
- La última línea debe de ser una sentencia `end`.

¿Cómo funciona la sentencia `switch-case`?

El valor de la expresión del `switch` se compara con los valores de cada comando `case`. Si se encuentra una coincidencia, entonces se ejecutan las instrucciones que siguen a dicho comando `case` (sólo un grupo de instrucciones, justo el que se encuentra después del propio comando `case`).

- Si hay más de una coincidencia, sólo se atenderá a la primera que se encuentra.
- Si no se encuentran coincidencias se ejecutan las instrucciones que siguen al comando `otherwise`, si éste ha sido declarado (es opcional).
- Si no se encuentran coincidencias y no está declarado el comando `otherwise`, entonces no se ejecutará ninguna instrucción o grupo de instrucciones.
- Una sentencia `case` puede tener más de un valor con el que establecer coincidencias. Para declararlos sólo es necesario ponerlos a continuación, uno detrás del otro, entre llaves y separados por comas, de la forma: `{valor1, valor2, valor3, ...}`. Este tipo de notación, que no se tratará en este libro, es lo que se conoce por array de celdas. En caso de tener más de un valor, el

grupo de instrucciones correspondientes al `case` se ejecutará siempre que al menos haya coincidencia con alguno de los valores indicados.

Nota para los programadores de C: *En MATLAB, la sentencia* `switch-case` *busca sólo hasta la primera coincidencia, saltando al final* (`end`) *una vez ejecutadas todas las instrucciones del* `case` *correspondiente. Esto difiere del uso que se hace de esta sentencia en el lenguaje C, donde se pueden encontrar y ejecutar instrucciones de más de un* `case`*, que se corresponden con más de una coincidencia, siendo por tanto necesaria alguna instrucción de ruptura de secuencia, como la sentencia* `break`*.*

Problema de ejemplo 7.4: Conversión de unidades de energía

Escribir un programa (en un fichero script) que convierta una cantidad de energía dada (trabajo) en J (joules), ft-lb (pies-libras), cal (calorías) o eV (electronvoltios) a la cantidad equivalente en otra unidad diferente especificada por el usuario. El programa pedirá al usuario que introduzca la cantidad de energía y su unidad, así como la unidad a la que se quiere realizar la conversión. La salida será la cantidad introducida convertida a la nueva unidad de energía elegida.

Los factores de conversión son: 1 J = 0,738 ft-lb = 0,239 cal = $6{,}24 \times 10^{18}$ eV. Utilizar posteriormente este programa para:

a) Convertir 150 J a ft-lb.

b) Convertir 2800 cal a J.

c) Convertir 2,7 eV a cal.

Solución

El programa incluye dos bloques de tipo `switch-case` y uno de tipo `if-else-end`. El primer bloque `switch-case` se utiliza para convertir la cantidad de entrada, con su correspondiente unidad de medida, a joules. El segundo bloque `switch-case` se utiliza para convertir la cantidad en joules a la nueva unidad, especificada por el usuario, para la conversión. La sentencia `if-else-end` se utiliza para genera un mensaje de error si las unidades se introducen incorrectamente.

```
Ein = input('Introduzca el valor de la energia (trabajo) que hay que convertir: ');
EinUnits = input('Introduzca la unidad actual de la energia (J, ft-lb, cal, eV): ','s');
EoutUnits = input('Introduzca la nueva unidad para la energia (J, ft-lb, cal, eV): ','s');
error = 0;
switch EinUnits
case 'J'
   EJ = Ein;
case 'ft-lb'
   EJ = Ein/0.738;
case 'cal'
   EJ = Ein/0.239;
case 'eV'
   EJ = Ein/6.24e18;
```

Se asigna 0 a la variable `error`.

Primer bloque `switch`. La expresión es una cadena que representa la unidad de la cantidad inicial introducida.

Cada uno de los valores de las cuatros sentencias `case` tiene una cadena que se corresponde con cada una de las posibles unidades introducidas, así como un comando que convierte `Ein` a `joules` (asignando el valor a `EJ`).

```
otherwise
   error = 1;
end
switch EoutUnits
case 'J'
   Eout = EJ;
case 'ft-lb'
   Eout = EJ*0.738;
case 'cal'
   Eout = EJ*0.239;
case 'eV'
   Eout = EJ*6.24e18;
otherwise
   error = 1
end
if error
   disp('ERROR, la unidad de entrada o la de conversion se ha tecleado incorrectamente.')
else
   fprintf('E = %g %s',Eout,EoutUnits)
end
```

Asigna 1 a la variable `error` si la cadena introducida no coincide con ninguno de los valores de las sentencias `case`. Si sucede esto, lo más probable es que el usuario haya tecleado mal la unidad en cuestión.

Segundo bloque `switch`. La expresión es una cadena que representa la nueva unidad para la cantidad de entrada.

Cada uno de los valores de las cuatro sentencias `case` tiene una cadena que se corresponde con cada una de las posibles unidades introducidas, así como un comando que convierte `EJ` a la nueva unidad (asignando el valor a `Eout`).

Asigna 1 a la variable error si la cadena introducida no coincide con ninguno de los valores de las sentencias case. Si sucede esto, lo más probable es que el usuario haya tecleado mal la unidad en cuestión.

Sentencia `if-else-end` donde se analiza el valor de la variable `error` para visualizar un mensaje de error al usuario.

Si `error` es falso (cero) se visualiza la conversión de energía realizada (salida del programa).

Como ejemplo, a continuación se muestra la salida de este fichero (guardado como ConversionEnergia), cuando se ejecuta desde la Ventana de Comandos con los datos dados en el apartado *b*) del problema.

```
>> ConversionEnergia
Introduzca el valor de la energia (trabajo) que hay que convertir: 2800
Introduzca la unidad actual de la energia (J, ft-lb, cal, eV): cal
Introduzca la nueva unidad para la energia (J, ft-lb, cal, eV): J
E = 11715.5 J
```

7.4 Bucles

Los bucles o iteraciones son otro de los métodos utilizados para controlar el flujo de un programa. En un bucle, la ejecución de uno o varios comandos se repite varias veces consecutivamente. Cada una de estas repeticiones se denomina paso o iteración. En cada paso se ejecutan grupos de instrucciones MATLAB como las que se han visto hasta ahora. MATLAB permite definir dos tipos distintos de bucle. Por un lado,

la estructura `for-end` (Sección 7.4.1) permite definir bucles donde el número de iteraciones queda definido al comienzo del bucle. Por otro lado, en la estructura `while-end` (Sección 7.4.2) el número de iteraciones no se conoce a priori, ya que depende de que se cumpla o no una condición determinada. Independientemente del número de iteraciones de uno u otro tipo de bucle, es posible salir de cualquiera de ellos (antes de que acaben) utilizando la sentencia `break` (ver Sección 7.6).

7.4.1 *Bucles del tipo* `for-end`

- En este tipo de bucles la ejecución de una o varias instrucciones se repite un número fijo de veces. La estructura de este tipo de bucles se muestra en la Figura 7.5.
- La variable índice del bucle puede tener cualquier nombre (normalmente se utiliza i, j, k, m o n como nombre de variable índice, aunque i y j no deberían usarse si se está trabajando con números complejos).
- En el primer paso `k` toma el valor inicial `f`, y el sistema ejecuta los comandos que se encuentran entre las instrucciones `for` y `end`. Seguidamente el sistema vuelve a la instrucción `for` para realizar el segundo paso. Ahora `k` toma el valor `k = f + s`, y los comandos que se encuentran entre las sentencias `for` y `end` se ejecutan otra vez con el nuevo valor de `k`. El proceso se repite hasta el último paso, es decir, cuando `k` toma el valor `t`. Entonces el programa no retorna a la instrucción `for`, sino que el bucle acaba y el sistema continúa ejecutando los comandos que se encuentran a continuación del comando `end`. Por ejemplo, si `k = 1:2:9`, se llevarían a cabo cinco iteraciones o pasos, y `k` tomaría los valores 1, 3, 5, 7 y 9 en cada iteración del bucle.
- El incremento `s` puede ser negativo (por ejemplo. `k = 25:-5:10` produce cuatro iteraciones con los valores `k` = 25, 20, 15, 10).
- Si el valor de incremento se omite, el incremento por defecto será de 1 (por ejemplo, `k = 3:7` produce cinco iteraciones con los valores `k` = 3, 4, 5, 6, 7).
- Si `f = t`, el bucle se ejecuta sólo una vez.
- Si `f > t` y `s > 0`, o si `f < t` y `s < 0`, el bucle no se ejecutará.
- Si los valores de `k`, `s` y `t` son tales que `k` no puede ser igual a `t`, entonces, si `s` es positivo, la última iteración es aquella en la que `k` tendrá el valor más alto posible y a la vez sea menor que `t` (por ejemplo, `k = 8:10:50` produce cinco iteraciones con valores `k` = 8, 18, 28, 38, 48). Si `s` es negativo, la última iteración será aquella en la que `k` es el menor valor posible y a su vez mayor que `t`.
- La variable `k` también puede tomar valores específicos (que no son incrementos del valor anterior). Para ello hay que introducir mediante un vector los valores que se deseen. Por ejemplo: `for k = [7 9 -1 3 3 5].`

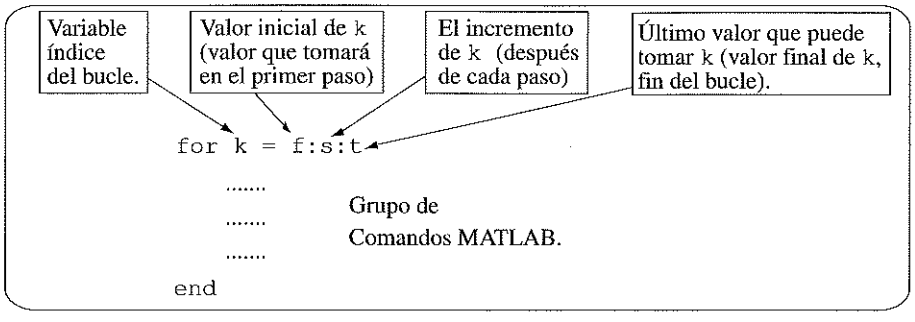


Figura 7.5: Estructura del bucle `for-end`.

- El valor de k no debe ser redefinido dentro del propio bucle. Si se hace, el bucle podría no funcionar como se esperaba.
- Todos los comandos `for` de un programa **deben** tener su correspondiente comando de finalización `end`.
- El valor de la variable índice del bucle (k) no se visualiza automáticamente. Sin embargo, es posible visualizarla tecleando k como un comando más dentro del bucle. A veces esto es útil en determinadas circunstancias, especialmente para depurar el programa.
- Cuando un bucle finaliza, la variable índice del bucle (k) toma el valor de la asignación en el último paso o iteración, es decir, el valor final indicado en la propia declaración del bucle.

Veamos un ejemplo sencillo (en forma de script) de utilización de la estructura `for-end`:

```
for k = 1:3:10
  x = k^2
end
```

Cuando se ejecuta este programa, el bucle itera cuatro veces. Por tanto k toma los siguientes valores durante los cuatro pasos: k = 1, 4, 7 y 10. Esto implica que los valores asignados a x (esto se hace en función de k) durante esos cuatro pasos son: x = 1, 16, 49 y 100, respectivamente. Como no se ha tecleado un punto y coma detrás de la segunda línea, el valor de x se visualizará en la Ventana de Comandos por cada iteración del bucle, produciendo la siguiente salida:

```
>> x =
   1

x =
   16

x =
   49

x =
   100
```

Problema de ejemplo 7.5: Series numéricas

a) Utilizar un bucle de tipo `for-end`, en un fichero script, para calcular la suma de los primeros n términos de la siguiente serie numérica: $\sum_{k=1}^{n} \frac{(-1)^k k}{(2)^k}$. Ejecutar el fichero script para $n = 4$ y $n = 20$.

b) La función sen(x) se puede aproximar utilizando la siguiente serie de Taylor:

$$\text{sen } x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

Escribir un fichero de función que calcule sen(x) utilizando la serie de Taylor. Utilizar para ello la siguiente definición de función: `y = Tsen(x,n)`. Los argumentos de entrada serán el ángulo `x` en grados y el número `n` de términos de la serie. Utilizar la función para aproximar el valor de sen(150°), sumando los primeros 3 términos y después sumando los 7 primeros términos. Comparar el resultado obtenido en una y otra aproximación.

Solución

a) El fichero script que calcula la suma de los primeros n términos de la serie se muestra a continuación:

```
n = input('Introduzca el numero de terminos: ');
S = 0;
for k = 1:n
   S = S + (-1)^k*k/2^k;
end
fprintf('El valor (suma) de la serie es: %f', S)
```

Inicializa la suma a cero.

Bucle `for-end`.

En cada paso se calcula un elemento de la serie y se va sumando al del paso anterior.

El sumatorio se implementa mediante un bucle, de forma que en cada paso se calcula un término de la serie (en la primera iteración el primer término, en la segunda, el segundo término, y así sucesivamente), y se suma al anterior (al calculado en la iteración anterior). Este programa se guarda en disco con el nombre Exp74a, y una vez que se ejecuta en la Ventana de Comandos produce la siguiente salida:

```
>> Exp74a
Introduzca el numero de terminos: 4
El valor (suma) de la serie es: -0.125000
>> Exp74a
Introduzca el numero de terminos: 20
El valor (suma) de la serie es: -0.222216
```

b) A continuación se muestra el fichero de función que aproxima el valor de sen(x) mediante la suma de los términos de la serie de Taylor propuesta en el enunciado:

```
function y = Tsen(x,n)
%Tsen aproxima el valor del seno de x utilizando una serie de Taylor.
%Los argumentos de entrada son:
%El angulo x en grados y el numero x de terminos de la serie.
```

```
xr = x*pi/180                                  Conversión del ángulo, de grados a radianes.
y = 0;
for k = 0:n-1
   y = y + (-1)^k*xr^(2*k+1)/factorial(2*k+1);   <-- Bucle for-end.
end
```

El primer elemento se corresponde con k igual a 0, lo que significa que para sumar los n términos de la serie, en la última iteración k debe ser igual a n-1. La función se utiliza en la Ventana de Comandos para calcular el seno de 150°, utilizando 3 y 7 términos de la serie:

```
>> Tsen(150,3)
                                   Se calcula el valor del seno de 150° mediante la suma
ans =                              de los tres primeros términos de la serie de Taylor.
   0.6523
>> Tsen(150,7)
                                   Se calcula el valor del seno de 150° mediante la suma
ans =                              de los siete primeros términos de la serie de Taylor.
   0.5000
                                   El valor exacto es 0,5.
```

Nota sobre el bucle `for-end` *y las operaciones elemento a elemento:*

En ciertas ocasiones se puede obtener el mismo resultado aplicando un bucle `for-end` o realizando operaciones elemento a elemento sobre arrays. El Problema de ejemplo 7.5 ilustra cómo funciona un bucle `for-end`, pero ese problema también se podría haber resuelto utilizando operaciones elemento a elemento (ver los Problemas 7 y 8 en la Sección 3.9). Las operaciones elemento a elemento con arrays son una de las características más notables de MATLAB, ya que ofrece un método de cálculo que en ciertas circunstancias evita la utilización de bucles. En general, las operaciones elemento a elemento son más rápidas que los bucles, y se recomiendan en aquellas situaciones en las que se pueden utilizar ambos métodos.

Problema de ejemplo 7.6: Modificación de los elementos de un vector

Sea el vector V = [5, 17, –3, 8, 0, –1, 12, 15, 20, –6, 6, 4, –7, 16]. Escribir un programa, en forma de fichero script, que multiplique por dos los elementos de V positivos y divisibles por 3 y/o por 5, y eleve al cubo los elementos negativos pero mayores que –5.

Solución

El problema se puede resolver utilizando un bucle `for-end` que contenga un bloque `if-elseif-end` en su interior. El número de pasos o iteraciones del bucle será igual al número de elementos del vector V. En cada paso se comprueba si un elemento de V verifica las condiciones establecidas para cambiar el elemento en cuestión. El elemento se cambia si satisface las condiciones del enunciado del problema. Veamos el fichero script que resuelve el problema:

```
V = [5, 17, -3, 8, 0, -7, 12, 15, 20, -6, 6, 4, -2, 16];
n = length(V);
for k = 1:n
  if V(k) > 0 & (rem(V(k),3) == 0 | rem(V(k),5) == 0
    jV(k) = 2*V(k);
  elseif V(k) < 0 & V(k) > -5
    V(k) = V(k)^3;
  end
end
V
```

Se inicializa n con el número de elementos de V.

Bucle `for-end`.

Condición `if-elseif-end`.

El programa se guarda con el nombre Exp75. La salida, tras ejecutar el programa en la Ventana de Comandos, es:

```
>> Exp75
V =
  10  17  -27  8  0  -7  24  30  40  -6  12  4  -8  16
```

7.4.2 *Bucles del tipo* `while-end`

Los bucles `while-end` se utilizan cuando se necesita un proceso iterativo, pero se desconoce previamente el número de iteraciones que hay que realizar. En este tipo de bucles el número de pasos no se especifica al principio, por tanto el proceso iterativo continúa mientras se satisface una condición determinada. La estructura de un bloque `while-end` se muestra en la Figura 7.6

```
while   expresión_condicional
    .....
    .....        Grupo de
    .....        instrucciones MATLAB.
end
```

Figura 7.6: Estructura del bucle `while-end`.

La primera línea de un bucle de este tipo incluye una expresión condicional. Cuando el programa llega a esa línea, comprueba dicha expresión. Si ésta resulta falsa (0), MATLAB salta directamente al final del bucle, es decir, a la sentencia `end`. Por el contrario, si la expresión condicional es verdadera (1), MATLAB ejecuta el grupo de comandos que se encuentran entre la sentencia `while` y la sentencia `end`. Seguidamente MATLAB retrocede de nuevo a la instrucción `while` y comprueba de nuevo la expresión condicional (para ésta y las sucesivas iteraciones). En definitiva, el proceso iterativo continúa hasta que la expresión condicional del `while` se hace falsa.

Para una ejecución correcta del bucle `while-end`*:*

- La expresión condicional del comando `while` debe incluir al menos una variable.
- Las variables de la expresión condicional deben tener valores asignados cuando MATLAB ejecute el comando `while` por primera vez.
- Al menos una de las variables de la expresión condicional debe cambiar de valor por la ejecución de los comandos que están entre la sentencia `while` y la sentencia `end`. En caso contrario, si ninguna de las variables cambia, la condición del `while` se mantendría siempre igual y el bucle nunca acabaría de iterar (bucle infinito).

En el siguiente programa se muestra un ejemplo sencillo de bucle `while-end`. En este programa, una variable x con valor inicial 1 se multiplica por dos en cada iteración. El bucle se repite mientras el valor de la variable sea menor o igual a 15.

```
x = 1
while x <= 15
   x = 2*x
end
```

Valor inicial de x.

El siguiente comando después de `while` se ejecutará siempre y cuando el valor de x sea menor o igual a 15.

En cada paso, el valor de x se duplica.

Cuando este programa se ejecuta, la salida que puede verse en la Ventana de Comandos es la siguiente:

```
x =
   1

x =
   2

x =
   4

x =
   8

x =
   16
```

Valor inicial de x.

En cada pasada, el valor de x se duplica.

Cuando x = 16, la expresión condicional sujeta al bucle `while` es falsa (16 no es menor o igual a 15). Esto provoca que el bucle termine.

Nota importante:

Cuando se escribe un bucle del tipo `while-end`, el programador debe asegurarse de que la variable o las variables que intervienen en la expresión condicional sean modificadas en el interior del bucle, mediante alguna sentencia apropiada, de forma que la condición del `while` se haga falsa y el proceso iterativo pueda terminar alguna vez. En caso contrario, el bucle continuará indefinidamente (bucle infinito) porque la condición de iteración es siempre cierta. Por ejemplo, en el programa anterior, si la expresión condicional fuera x >= 0,5, el bucle continuaría indefinidamente. Esta situación se puede evitar contando el número de pasos y deteniendo de alguna forma el bucle si este número sobrepasa un valor considerado como *excesivo*. Esto se puede llevar a cabo añadiendo el número máximo de pasos en la expresión condicional, o utilizando el comando de parada explícita `break` (Sección 7.6).

Como nadie está libre de cometer errores, los bucles infinitos pueden aparecer aunque se intente evitarlos. Cuando esto ocurra, el usuario puede parar la ejecución de un bucle infinito pulsando la combinación de teclas Ctrl + C o Ctrl + Break.

Problema de ejemplo 7.7: Representación de una función mediante series de Taylor

La función $f(x) = e^x$ se puede representar mediante la siguiente serie de Taylor: $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$

Escribir un programa script en MATLAB que calcule e^x utilizando la serie de Taylor indicada anteriormente. El programa debe calcular e^x sumando los términos de la serie en un proceso iterativo que se detenga cuando el valor absoluto del último término sumado sea inferior a 0,0001. Utilizar para ello un bucle del tipo `while-end`, limitando el número de iteraciones a 30. Si en la iteración número 30 el valor del término sumado no es inferior a 0,0001, el programa deberá parar y visualizar un mensaje que diga que se necesitan más de 30 términos.

Utilizar el programa para calcular e^2, e^{-4} y e^{21}.

Solución

Los primeros términos de la serie de Taylor tienen la forma:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Más abajo se muestra el código del programa que resuelve este problema. En primer lugar se pide al usuario que introduzca el valor de `x`. A continuación el programa asigna el valor 1 (el valor del primer término de la serie) a la variable `an`. Seguidamente, a la variable `S`, que guarda la suma acumulada, se le suma `an`. Después, a partir del segundo término en adelante, el programa utiliza un bucle `while` para calcular el término enésimo de la serie y sumarlo a la variable `S`. Mediante la variable `n`, el programa cuenta el número de términos calculados. La expresión condicional de la instrucción `while` será cierta en tanto que el valor absoluto del término enésimo `an` sea mayor que 0,0001, y el número de pasos `n` sea inferior a 30. Esto significa que si el término 30 no es menor que 0,0001 el bucle terminará.

```
x = input('Introduzca el valor x: ');
n = 1; an = 1; S = an;
while abs(an) >= 0.0001 & n <= 30          % Comienzo del bucle while.
   an = x^n/factorial(n);                  % Cáculo del término enésimo.
   S = S + an;                             % Se suma el término enésimo a la suma acumulada.
   n = n + 1;                              % Se cuenta el número de iteraciones.
end                                        % Final del bucle while.
```

```
if n >= 30
   disp('Se necesitan mas de 30 terminos')
else
   fprintf('exp(%f) = %f',x,S)
   fprintf('\nEl numero de terminos utilizados es: %i',n)
end
```

Estructura `if-else-end`.

El programa utiliza un bloque condicional del tipo `if-else-end` para visualizar los resultados. Si el bucle termina porque el término enésimo no es menor que 0,0001, se visualiza un mensaje que indica esta situación. Por el contrario, si el proceso resulta exitoso, se visualiza el valor de la función y el número de términos de la serie sumados. Cuando el programa se ejecuta, el número de pasadas depende realmente del valor de `x`. Veamos a continuación la ejecución del programa anterior (guardado como expox) para el cálculo de los valores e^2, e^{-4} y e^{21}:

```
>> expox
Introduzca el valor x: 2
exp(2.000000) = 7.389046
El numero de terminos utilizados es: 12
>> expox
Introduzca el valor x: -4
exp(-4.000000) = 0.018307
El numero de terminos utilizados es: 14
>> expox
Introduzca el valor x: 21
Se necesitan mas de 30 terminos
```

Se calcula exp(2).

Se suman en total 12 términos de la serie.

Se calcula exp(–4).

Se suman en total 14 términos de la serie.

Se calcula exp(21).

El programa avisa de que el cálculo ha sido incompleto, ya que se necesitan más de 30 términos.

7.5 Bucles anidados y sentencias condicionales anidadas

Los bucles y las sentencias condicionales se pueden anidar unos dentro de los otros. Esto significa que un bucle y/o una sentencia condicional puede empezar (y acabar) dentro de otro bucle y/o sentencia condicional. No existe límite sobre el número de bucles y sentencias condicionales que se pueden anidar. Sin embargo, es necesario recordar que cada sentencia `if`, `case`, `for` y `while` debe tener su correspondiente sentencia de finalización `end`. La Figura 7.7 muestra la estructura de un bucle `for-end` anidado dentro de otro bucle del mismo tipo. Por ejemplo, si en los bucles que muestra esa figura, `n` = 3 y `m` = 4, entonces para la primera iteración, `k` = 1, el bucle interno se ejecuta cuatro veces, tomando `h` los valores 1, 2, 3 y 4. En la siguiente iteración, cuando `k` = 2, el bucle interno se vuelve a ejecutar otras cuatro veces, tomando `h` nuevamente los valores 1, 2, 3 y 4. Finalmente, cuando `k` = 3, el bucle interno se vuelve a ejecutar otras cuatro veces. Cada vez que se teclea un bucle anidado MATLAB indenta automáticamente el nuevo bucle (interior) en función del bucle exterior. En el siguiente ejemplo se muestra la utilización de varias estructuras anidadas.

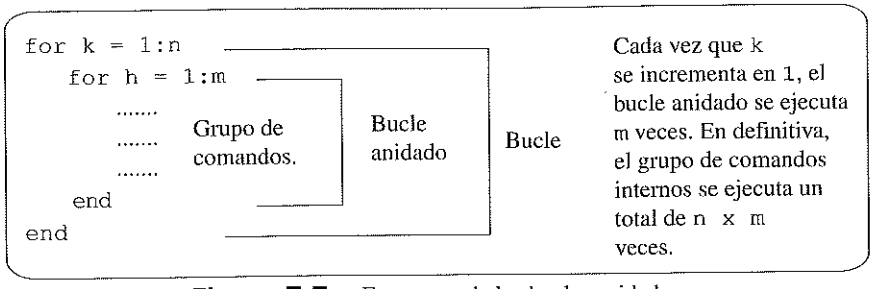


Figura 7.7: Estructura de los bucles anidados.

Problema de ejemplo 7.8: Creación e inicialización de una matriz mediante un bucle

Escribir un programa (fichero script) que cree una matriz de dimensión $n \times m$ cuyos elementos sean los que a continuación se detallan. El valor de los elementos de la primera fila deberá ser el número de la columna correspondiente. El valor de los elementos de la primera columna será el número de la fila correspondiente. El valor del resto de los elementos es igual a la suma del valor del elemento que se encuentra por encima de él y el valor del elemento que se encuentra a su izquierda. Cuando se ejecuta, el programa debe pedir al usuario que introduzca la dimensión (`n` y `m`) de la matriz en cuestión.

Solución

El programa que resuelve este problema, que se muestra más abajo, tiene dos bucles, uno de ellos anidado, y una estructura del tipo `if-elseif-else-end` también anidada. A los elementos de la matriz se les asignan valores fila por fila. La variable índice del primer bucle, `k`, indica la fila, y la variable índice del segundo bucle, `h`, indica la columna.

```
n = input('Introduzca el numero de filas: ');
m = input('Introduzca el numero de columnas: ');
A = [];                                          % Se define una matriz A vacía.
for k = 1:n                                      % Comienzo del primer bucle.
  for h = 1:m                                    % Comienzo del segundo bucle.
    if k == 1                                    % Comienzo de la sentencia condicional.
      A(k,h) = h;                                % Asigna valores a los elementos de la primera fila.
    elseif h == 1
      A(k,h) = k;                                % Asigna valores a los elementos de la primera columna.
    else
      A(k,h) = A(k,h-1) + A(k-1,h);              % Asigna valores a otros elementos.
    end                                          % Fin de la sentencia if.
  end                                            % Fin del bucle anidado for-end.
end                                              % Fin del bucle externo for-end.
A
```

Cuando se ejecuta este programa (guardado como Cap7Ej7) produce la siguiente salida en la Ventana de Comandos:

```
>> Cap7Ej7
Introduzca el numero de filas: 4
Introduzca el numero de columnas: 5
A =
   1   2   3   4   5
   2   4   7  11  16
   3   7  14  25  41
   4  11  25  50  91
```

7.6 *Los comandos* `break` *y* `continue`

El comando `break`*:*

- Cuando se introduce dentro de un bucle (de tipo `for` o `while`), este comando termina de forma completa la ejecución del bucle (no solamente del último paso, sino de todo el bucle). Cuando este comando aparece dentro de un bucle, MATLAB salta al final del bucle (sentencia `end`) y continúa con la sentencia que sigue al bucle (no vuelve hacia arriba en la ejecución).
- Si el comando `break` se encuentra dentro de un bucle anidado, sólo terminará la ejecución del bucle anidado.
- Cuando el comando `break` aparece fuera de un bucle, en un fichero script o de función, provoca la terminación del fichero en cuestión.
- El comando `break` se utiliza habitualmente dentro de sentencias condicionales. Dentro de los bucles proporciona una forma de acabar con el proceso iterativo en función de alguna condición establecida. Por ejemplo, cuando el número de iteraciones excede un determinado valor, o cuando se produce algún tipo de error en un proceso numérico. Cuando se teclea fuera de un bucle, el comando `break` proporciona una forma de terminar con la ejecución del fichero de forma explícita cuando, por ejemplo, los datos suministrados a una función no coinciden con los datos esperados.

El comando `continue`*:*

- El comando `continue` se puede usar dentro de un bucle (`for` o `while`) para detener la iteración actual y forzar la siguiente iteración del bucle.
- El comando `continue` normalmente forma parte de alguna sentencia condicional. Cuando MATLAB alcanza este comando dentro de un bucle, éste no sigue ejecutando el resto de los comandos del bucle, sino que salta a la sentencia `end` (fin del bucle) y a continuación comienza la siguiente interacción.

7.7 Ejemplos de aplicaciones MATLAB

Problema de ejemplo 7.9: Plan de pensiones

Una persona jubilada tiene 300 000 € en una cuenta de plan de pensiones que le paga un 5% de interés anual. Esta persona planea sacar dinero de su cuenta una vez al año. Empieza sacando 25 000 € después del primer año. Después, en el futuro, incrementará la cantidad retirada en función de la tasa de inflación anual. Por ejemplo, si la inflación es del 3%, retirará 25 750 € al final del segundo año. Calcular el número de años que esta persona tardará en sacar todo el dinero de su cuenta, suponiendo para ello una inflación anual constante del 2%. Representar en un gráfico las cantidades anuales de dinero retiradas y el balance de la cuenta a lo largo de los años.

Solución

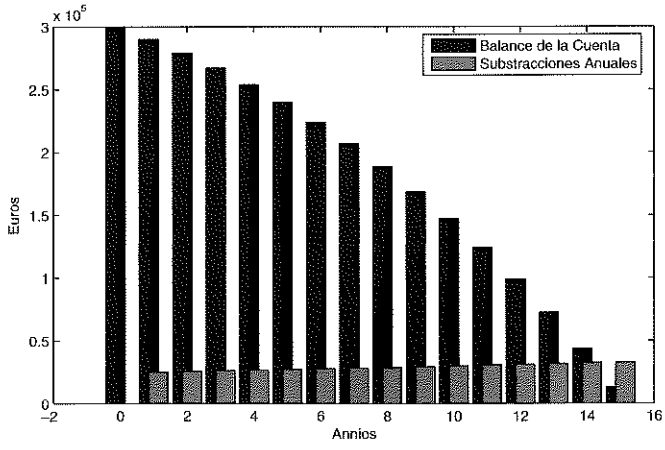
Este problema puede resolverse utilizando un bucle de tipo `while`, ya que el número de pasos o iteraciones no se conoce previamente. En cada iteración se calculará la cantidad de dinero que esta persona retira, así como el balance de la cuenta. El bucle continuará en tanto que el balance de la cuenta sea mayor o igual que la cantidad de dinero a sacar. A continuación se muestra un programa (en forma de script) que da la solución al problema. En este programa, `annio` es un vector cuyos elementos indican, mediante un número, el año en que se retira dinero de la cuenta. También, `W` es un vector con la cantidad de dinero que se retira cada año. Finalmente, `AB` es un vector que representa el balance de la cuenta de cada año.

```
interes = 0.05; inf = 0.02;
clear W AB annio
annio(1) = 0;                          % El primer elemento es el año 0.
W(1) = 0;                              % Cantidad inicial de dinero que se retira.
AB(1) = 300000;                        % Balance inicial del plan de pensiones.
Wsiguiente = 25000;                    % Cantidad que se retirará después de un año.
ABsiguiente = 300000*(1+interes);      % Balance de la cuenta después de un año.
n = 2;
   while ABsiguiente >= Wsiguiente     % El bucle itera mientras el balance siguiente sea mayor o
                                       % igual que la próxima cantidad de dinero que hay que retirar.
      annio(n) = n-1;
      W(n) = Wsiguiente;               % Cantidad retirada en el año n-1.
      AB(n) = ABsiguiente - W(n);      % Balance de la cuenta en el año n-1, después de retirar dinero.
      ABsiguiente = AB(n) * (1 + interes);   % Balance de la cuenta después del año adicional.
      Wsiguiente = W(n)*(1 + inf);     % Cantidad que se retirará después de otro año.
      n = n +1;
   end
fprintf('Se tardaran %f annios en sacar todo el dinero de la cuenta del plan de pensiones',annio(n-1))
bar(annio,[AB' W'], 2.0)
```

Cuando se guarda en disco (Cap7Ej9) y se ejecuta en la Ventana de Comandos, el programa produce la siguiente salida:

```
>> Cap7Ej9
Se tardaran 15 annios en sacar todo el dinero de la cuenta del plan de pensiones
```

Además, el programa genera el siguiente gráfico de barras (las etiquetas de los ejes y la leyenda se añadieron posteriormente con el editor gráfico).



Problema de ejemplo 7.10: Generación de números para el sorteo de la lotería

En un sorteo de lotería el jugador debe elegir varios números de entre una lista dada. Escribir una función MATLAB que genere una lista de n números enteros, distribuidos uniformemente entre un rango a y b. Todos los números seleccionados de la lista deben ser distintos.

a) Utilizar la función para generar una lista de 6 números que varíen entre 1 y 49.

b) Utilizar la función para generar una lista de 8 números que varíen entre 60 y 75.

c) Utilizar la función para generar una lista de 9 números que varíen entre −15 y 15.

Solución

A continuación se muestra una función MATLAB que resuelve el problema expuesto. La función utiliza la instrucción `rand` (ver Sección 3.7) y para asegurar que todos los números seleccionados sean distintos (que no haya números repetidos), se escogerán de uno en uno. Cada número se generará con la función `rand`, y se comparará con todos los números ya seleccionados anteriormente para ver si coincide con alguno de ellos. Si alguno se repite, no se seleccionará y se generará un nuevo número para reemplazarlo. El bucle `while` chequea precisamente esta condición, es decir, que el número no coincida con ningún otro de los almacenados en el vector `x`. Si se encuentra alguna coincidencia, se realizará un proceso iterativo de selección de un nuevo número hasta que éste sea diferente de los ya existentes en `x`.

```
function x = loteria(a,b,n)
% la función loteria selecciona n numeros (todos diferentes) del dominio a,b.
% x representa un vector con los n numeros seleccionados
for p = 1:n
  numero = round((b-a)*rand+1);
  if p == 1
    x(p) = numero;
  else
    r = 0;
    while r == 0
      r = 1;
      for k = 1:p-1
        if x(k) == numero
          numero = round((b-a)*rand+a);
          r = 0;
          break;
        end
      end
    end
  end
  x(p) = numero;
end
```

Selecciona un entero entre a y b.

Si el número seleccionado es el primero, se asigna directamente a x(1).

Si el número no es el primero, se comparará con los números seleccionados previamente.

Inicializa r a 0.

Vea la explicación más abajo.

Inicializa r a 1.

Bucle for para comparar el número nuevo con los ya existentes en x(p).

Si se encuentra una coincidencia, se selecciona un número nuevo y r se pone a 0.

Se para la última iteración. El programa regresa al bucle while. Inicializando r = 0 provoca que el bucle while comience de nuevo con un nuevo número.

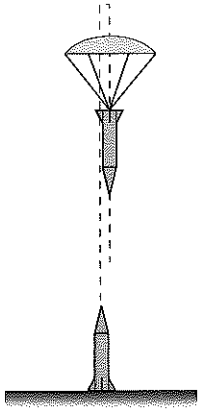
El número que no coincide con ningún otro número seleccionado previamente se asigna a x(p).

Esta función será luego utilizada en la Ventana de Comandos para resolver los tres casos planteados en el enunciado. Obsérvese que cada vez que se ejecute esta función producirá resultados distintos, aunque se utilicen los mismos parámetros de entrada. Esto se debe a que se trata de números aleatorios. Por este motivo los resultados obtenidos a continuación no coincidirán con los que obtenga el lector cuando ejecute esta función.

```
>> loteria(1,49,6)
ans =
  23   2   40   22   3   39
>> loteria(60,75,8)
ans =
  67   66   73   68   63   70   60   72
```

```
>> loteria(-15,15,9)

ans =
   -13   -4   0   -2   2   3   -12   12   8
```

Problema de ejemplo 7.11: Vuelo de un cohete



El vuelo de un cohete se modela de la siguiente manera. Durante los primeros 0,15 segundos el cohete es impulsado hacia arriba mediante un mecanismo de propulsión con una fuerza de 16 N. Luego continúa volando hacia arriba pero su velocidad irá disminuyendo debido a la fuerza de la gravedad. Después de alcanzar su altura máxima, el cohete comienza a descender. Cuando su velocidad alcanza los 20 m/s se abre un paracaídas (se supone que se abre instantáneamente), de forma que el cohete continúa descendiendo a una velocidad constante de 20 m/s hasta que llega al suelo.

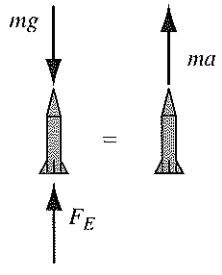
Escribir un programa que calcule y represente gráficamente la velocidad y la altitud del cohete en función del tiempo de vuelo.

Solución

El vuelo del cohete se puede modelar como si se tratara de una partícula que se mueve a lo largo de una línea recta en un plano vertical. La velocidad y la posición, en función del tiempo, de una partícula que se mueve con aceleración constante a lo largo de una línea recta vienen dados por la expresión:

$$v(t) = v_0 + at \qquad \text{y} \qquad s(t) = s_0 + v_0 t + \frac{1}{2}at$$

donde v_0 y s_0 son la velocidad y la posición iniciales, respectivamente. En el programa, el vuelo del cohete se divide en tres segmentos. Cada segmento se calcula mediante un bucle `while`, donde el tiempo se incrementará con cada iteración.



Segmento 1: Comprende los primeros 0,15 segundos, una vez que el mecanismo de propulsión del cohete se ha activado. Durante este periodo, el cohete viaja hacia arriba con una aceleración constante. La aceleración se puede calcular (figura de la derecha) mediante un diagrama donde se representa un cuerpo y la aceleración de su masa. A partir de la segunda ley de Newton, se obtiene que la suma de las fuerzas, en la dirección vertical, es igual a la masa por la aceleración (ecuación de equilibrio):

$$+\uparrow \Sigma F = F_E - mg = ma$$

Despejando la aceleración de la fórmula nos da:

$$a = \frac{F_E - mg}{m}$$

La velocidad y la altura, en función del tiempo, se pueden representar por:

$$v(t) = 0 + at \qquad \text{y} \qquad h(t) = 0 + 0 + \frac{1}{2}at^2$$

donde la velocidad y posición inicial son cero. En el programa, este segmento comienza cuando $t = 0$, de forma que bucle continuará iterando mientras $t < 0{,}15$ s. El tiempo, velocidad y altura al final de este segmento serán t_1, v_1 y h_1, respectivamente.

Segmento 2: Comprende el movimiento del cohete desde que el mecanismo de propulsión se detiene hasta que el paracaídas se abre. Durante este segmento, el cohete se mueve con una aceleración negativa constante g. La velocidad y altura del cohete, en función del tiempo, se pueden representar mediante las expresiones:

$$v(t) = v_1 - g(t - t_1) \qquad \text{y} \qquad h(t) = h_1 + v_1(t - t_1) - \frac{1}{2}g(t - t_1)^2$$

Durante este segmento, el bucle continúa iterando hasta que la velocidad del cohete es –20 m/s (es negativa, ya que el cohete se mueve hacia abajo). El tiempo y altura al final de este segmento serán t_2 y h_2, respectivamente.

Segmento 3: Comprende el movimiento del cohete desde que el paracaídas se abre hasta que el cohete llega al suelo. Durante este segmento el cohete se mueve con velocidad constante (aceleración cero). La altura, en función del tiempo, vendrá dada por: $h(t) = h_2 - v_{paracaídas}(t - t_2)$, donde $v_{paracaídas}$ es la velocidad constante que adquiere el cohete una vez abierto el paracaídas. Durante este segmento, el bucle continúa iterando siempre que la altura sea superior a cero.

El programa (fichero script) que lleva a cabo los cálculos se muestra a continuación:

```
  m = 0.05; g = 9.81; tProp = 0.15; Fuerza = 16; vParac = -20; Dt = 0.01;
clear t v h
n = 1;
t(n) = 0; v(n) = 0; h(n) = 0;
%Segmento 1
a1 = (Fuerza - m*g)/m;
while t(n) < tProp & n < 500000                    (Primer bucle while.)
  n = n + 1;
  t(n) = t(n-1) + Dt;
  v(n) = a1*t(n);
  h(n) = 0.5*a1*t(n)^2;
end
```

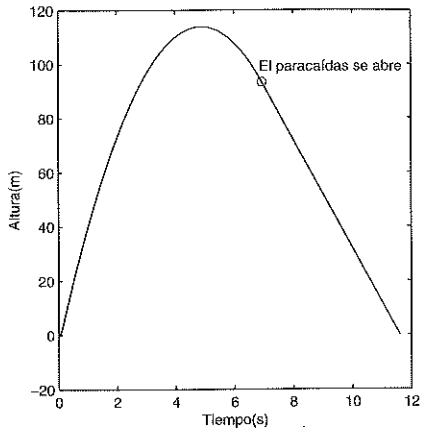
```
v1 = v(n); h1= h(n); t1 = t(n);
%Segmento 2
while v(n) >= vParac & n < 50000
  n = n + 1;
  t(n) = t(n-1) + Dt;
  v(n) = v1 - g*(t(n) - t1);
  h(n) = h1 + v1*(t(n) - t1) - 0.5*g*(t(n) - t1)^2;
end
v2 = v(n) ; h2 = h(n); t2 = t(n);
%Segmento 3
while h(n) > 0 & n < 50000
  n = n +1;
  t(n) = t(n-1) + Dt;
  v(n) = vParac;
  h(n) = h2 + vParac*(t(n)-t2);
end
subplot(1,2,1)
plot(t,h,t2,h2,'o')
subplot(1,2,2)
plot(t,v,t2,v2,'o')
```

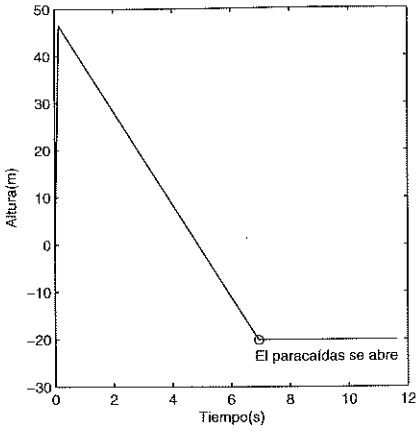
Segundo bucle `while`.

Tercer bucle `while`.

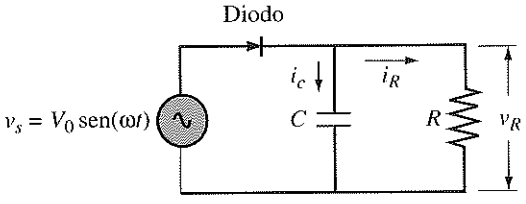
La precisión de los resultados depende del incremento de la magnitud de tiempo `Dt`. Un incremento de 0,01s proporciona un buen resultado. La expresión condicional del comando `while` incluye una condición sobre `n` (si `n` es superior a 50000, el bucle se detendrá). Esto se hace como medida de precaución, para evitar bucles infinitos en caso de que se produzca cualquier error en las sentencias dentro del bucle. Los gráficos generados por el programa se muestran a continuación (las etiquetas de los ejes y el texto adicional se han añadido utilizando el editor gráfico).

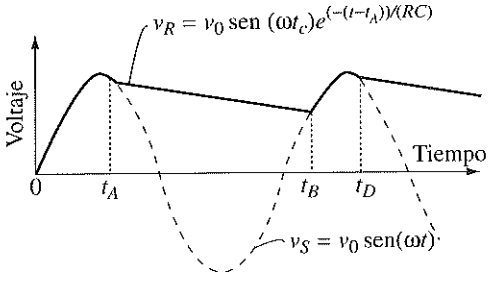
Nota: *Este problema se puede resolver y programar de distintas maneras. La solución que se ha explicado anteriormente es una de las soluciones posibles. Por ejemplo, en lugar de utilizar un bucle* `while`, *se podría haber calculado primero el tiempo en el que el paracaídas se abre y en el que el cohete llega a tierra, usando posteriormente un bucle* `for-end` *en vez del bucle* `while`. *Si los tiempos se calculan a priori, es posible realizar también operaciones elemento a elemento en vez de utilizar bucles.*





Problema de ejemplo 7.12: Conversor AC/DC





Un diodo rectificador de media onda es un circuito eléctrico que convierte voltaje AC (corriente alterna) a voltaje DC (corriente continua). En la figura adjunta se muestra un circuito compuesto por una fuente de voltaje, un diodo, un condensador y una resistencia de carga. El voltaje de la fuente es $v_s = v_0 \operatorname{sen}(\omega t)$, donde $\omega = 2\pi f$, siendo f la frecuencia. La forma en la que opera el circuito se muestra en la gráfica adjunta, donde la línea discontinua refleja el voltaje de la fuente y la línea sólida muestra el voltaje a través de la resistencia. En el primer ciclo, el diodo está activado (conduce corriente), desde $t = 0$ hasta $t = t_A$. En ese momento el diodo se apaga y la potencia que recibe la resistencia provendrá del condensador al descargarse. En el instante $t = t_B$ el diodo se activa de nuevo y continúa conduciendo la corriente hasta que $t = t_D$. Este ciclo continúa mientras que la fuente de voltaje esté activada. Para realizar un análisis simple del circuito se supone que el diodo es ideal y que el condensador no tiene carga inicial (en el instante $t = 0$). Cuando el diodo se activa, el voltaje de la resistencia y la corriente vendrán dados por:

$$v_R = v_0 \operatorname{sen}(\omega t) \qquad \text{e} \qquad i_g = v_0 \operatorname{sen}(\omega t)/R$$

La corriente en el condensador es:

$$i_C = \omega C v_0 \cos(\omega t)$$

Cuando el diodo está apagado, el voltaje a través de la resistencia vendrá dado por:

$$v_R = v_0 \operatorname{sen}(\omega t_A) e^{(-(t-t_A))/(RC)}$$

Los instantes en que el diodo se apaga (t_A, t_D, etc.) se calculan a partir de la condición $i_R = -i_C$. El diodo se enciende de nuevo cuando el voltaje de la fuente alcanza el voltaje a través de la resistencia (en la figura, t_B).

Escribir un programa en MATLAB que represente el voltaje a través de la resistencia v_R y el voltaje de la fuente v_s en función del tiempo, para $0 \leq t \leq 70$ ms. La resistencia de carga tiene una magnitud de 1800 Ω, el voltaje de la fuente $v_0 = 12$ V, y la frecuencia $f = 60$ Hz. Para analizar el efecto del tamaño del condensador sobre el voltaje a través de la resistencia de carga, ejecutar el programa dos veces, una para $C = 45$ μF y otra para $C = 10$ μF.

Solución

El programa que resuelve este problema, mostrado a continuación, tiene dos partes. La primera calcula el voltaje v_R cuando el diodo está activado, y la segunda cuando el diodo está apagado. Se utiliza la instrucción `switch` para alternar entre la primera y la segunda parte del programa. Los cálculos comienzan cuando el diodo está encendido (variable `estado = 'encendido'`). Cuando $i_R - i_C \leq 0$ el valor de la variable `estado` cambiará a `'apagado'`. De esta forma, el programa (segunda parte) calcula v_R para ese estado. Estos cálculos continúan hasta que $v_s \geq v_R$, cuando el programa cambia de nuevo volviendo a las ecuaciones válidas para el estado en que el diodo está encendido.

```
V0 = 12; C = 45e-6; R = 1800; f = 60;
Tf = 70e-3; w = 2*pi*f;
clear t VR Vs
t = 0:0.05e-3:Tf;
n = length(t);
estado = 'encendido'                          % Se asigna 'activado' a la variable estado.
for i = 1:n
  Vs(i) = V0*sin(w*t(i));                     % Se calcula el voltaje de la fuente en el instante t.
  switch estado
    case 'encendido'                          % El diodo está activado.
    VR(i) = Vs(i);
    iR = Vs(i)/R;
    iC = w*C*V0*cos(w*t(i));
    sumI = iR + iC;
    if sumI <= 0                              % Comprueba si iR - iC ≤ 0.
      estado = 'apagado'                      % Si es verdadero, estado será 'apagado'.
      tA = t(i);                              % Se asigna el valor a tA.
    end
```

```
    case 'apagado'
    VR(i) = V0*sin(w*tA)*exp(-(t(i)-tA)/(R*C));
    if Vs(i) >= VR(i)
       estado = 'encendido';
    end
  end
end
plot(t,Vs,':',t,VR,'k','linewidth',1)
xlabel('Tiempo (s)')
```

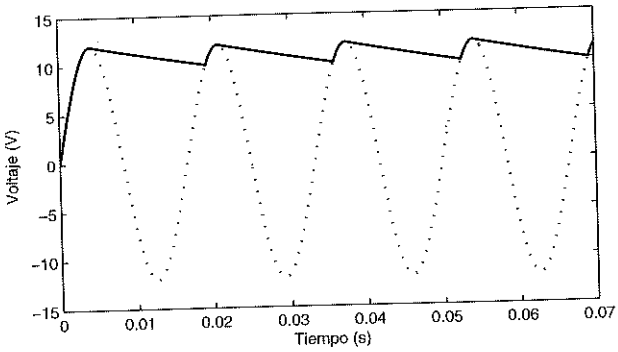
El diodo está desactivado.

Comprueba si $v_s \geq v_R$.

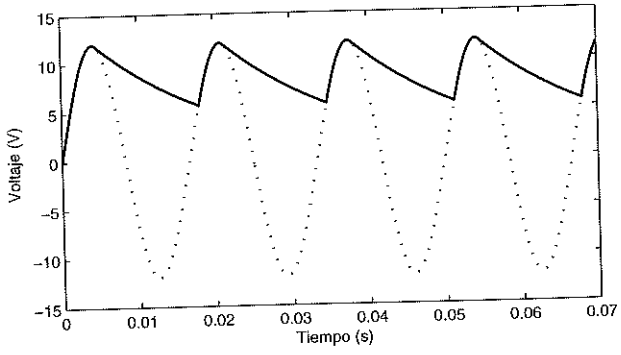
Si es verdadero, estado será 'encendido'.

Representación gráfica de v_s y v_R en función del tiempo.

Seguidamente se muestran los dos gráficos generados a partir del programa anterior. Uno de ellos muestra el resultado para $C = 45$ μF, y el otro para $C = 10$ μF. Como se puede observar, un condensador de mayor capacidad hará que el voltaje DC sea más suave (rizos más pequeños en la onda).



$C = 45$ μF



$C = 45$ μF

7.8 Problemas

1. Calcule las siguientes expresiones a mano, sin utilizar MATLAB. Utilice luego MATLAB para comprobar que el resultado es correcto.

   a) 5 <= 8 – 3

   b) $y = 7 < 3 - 1 + 6 > 2$

   c) $y = (7 < 3) - 1 + (6 > 2)$

   d) $y = 2 \times 4 + 5 == 7 + \frac{20}{4}$

2. Sean $a = 10$ y $b = 6$. Calcule las siguientes expresiones a mano, sin utilizar MATLAB. Utilice luego MATLAB para comprobar que el resultado es correcto.

   a) $y = a >= b$

   b) $y = a - b <= \frac{b}{2}$

   c) $y = a - \left(b <= \frac{b}{2}\right)$

3. Sean v = [4 –2 –1 5 0 1 –3 8 2] y w = [0 2 1 –1 0 –2 4 3 2]. Calcule las siguientes expresiones a mano, sin utilizar MATLAB. Utilice luego MATLAB para comprobar que el resultado es correcto.

   a) $v >= w$

   b) $w \sim= v$

4. A partir de los vectores v y w del ejercicio anterior, utilice operadores relacionales para crear un vector compuesto por los elementos de w que sean mayores que los elementos de v.

5. Calcule las siguientes expresiones a mano, sin utilizar MATLAB. Utilice luego MATLAB para comprobar que el resultado es correcto.

   a) 5&-2

   b) 8-2|6+5&~2

   c) ~(4&0)+8*~(4|0)

6. La temperatura máxima diaria (en °F) en Nueva York y Anchorage, Alaska, durante el mes de enero de 2001 vienen dadas en los siguientes vectores (datos tomados del Ministerio Nacional de Meteorología y Oceanografía de Estados Unidos).
   TNY = [31 26 30 33 33 39 41 41 34 33 45 42 36 39 37 45 43 36 41 37 32 32 35 42 38 33 40 37 36 51 50],
   TANC = [37 24 28 25 21 28 46 37 36 20 24 31 34 40 43 36 34 41 42 35 38 36 35 33 42 42 37 26 20 25 31].

   Escriba un programa script que calcule:

   a) La temperatura media en ese mes para cada ciudad.

   b) El número de días que estuvo la temperatura de cada ciudad por debajo de la media.

   c) El número de días, y a qué días del mes corresponden, en los cuales la temperatura de Anchorage fue mayor que la temperatura de Nueva York.

   d) El número de días, y a qué días del mes corresponden, en los cuales la temperatura fue igual en ambas ciudades.

e) El número de días, y a qué días del mes corresponden, en los cuales la temperatura de ambas ciudades se mantuvo por encima de 32 °F (sin helar).

7. Represente la siguiente función de dos formas distintas:

$$f(x) = \begin{cases} 4e^{x+2} & -6 \le x \le -2 \\ x^2 & -2 \le x \le 2{,}5 \\ (x+6{,}5)^{1/3} & 2{,}5 \le x \le 6 \end{cases}$$

a) Escribiendo un fichero script que utilice bucles y sentencias condicionales.

b) Cuando una función *f(x)* para utilizarla posteriormente en un fichero script.

8. Escriba un programa script que calcule las raíces reales de una función cuadrática $ax^2 + bx + c = 0$. Llame al fichero `raicescuad`. Cuando el fichero se ejecute, éste debe pedir al usuario que introduzca los valores de las constantes `a`, `b` y `c`. Para calcular las raíces de la ecuación, el programa calculará el discriminante *D*:

$$D = b^2 - 4ac$$

Si $D > 0$, el programa visualizará un mensaje del tipo: 'La ecuación tiene dos raíces', y los valores de las raíces se visualizarán en la línea siguiente.
Si $D = 0$, el programa visualizará un mensaje del tipo: 'La ecuación tiene una raíz', y el valor de la raíz se visualizará en la línea siguiente.
Si $D < 0$, el programa visualizará un mensaje del tipo: 'La ecuación no tiene raíces reales'.
Ejecute el fichero script en la Ventana de Comandos tres veces para calcular las soluciones de las siguientes ecuaciones:

a) $2x^2 + 8x - 3 = 0$

b) $15x^2 + 10x + 5 = 0$

c) $18x^2 + 12x + 2 = 0$

9. Utilice bucles para crear una matriz A de dimensión 4×7, en la cual el valor de cada elemento sea la suma de sus índices (el número de la fila y el número de la columna de cada elemento). Por ejemplo, el valor del elemento A(2,5) será 7.

10. Utilice bucles y sentencias condicionales para crear una matriz de dimensión 5×8, en la cual el valor de cada elemento sea igual a la raíz cuadrada de la suma de los índices de cada elemento, siempre que el elemento no se encuentre en una columna o fila par. El valor de un elemento que esté en una fila o columna par será igual a la suma del cuadrado de los índices. (Los índices de un elemento de una matriz son el número de fila y el número de columna que le corresponden.)

11. Escriba un programa (utilizando un bucle) que calcule la suma de los *m* primeros términos de la serie:

$$\sum_{n=0}^{m} (-1)^n \frac{1}{2n+1} \quad (n = 0, 1, 2, \ldots, m)$$

Esta serie se denomina serie de Leibniz, y converge a $\pi/4$. Ejecute el programa para $m = 10$ y $m = 500$. Compare posteriormente estos resultados con el valor exacto $\pi/4$.

12. Sea el vector x = [15 –6 0 8 –2 5 4 –10 0,5 3]. Escriba un programa que utilice sentencias condicionales y bucles para calcular la suma de los elementos positivos del vector x.

13. Escriba un programa script que encuentre el menor número entero impar que sea divisible por 3, y cuyo cubo sea mayor que 4000. Utilice un bucle que comience en 1 y se detenga cuando encuentre el número que cumpla las condiciones anteriores. Finalmente el programa visualizará el mensaje: 'El número pedido es: ', y visualizará seguidamente el número calculado.

14. Escriba una función que ordene los elementos de un vector de cualquier longitud, de mayor a menor. Utilice la siguiente línea de definición de función: `y = ordenar(x)`. La entrada de la función será un vector `x` de cualquier longitud, y la salida `y` será un vector que contendrá los elementos de `x` en orden descendente. No se puede utilizar la función predefinida de MATLAB `sort` para este ejercicio. Cree su propia función y pruébela con un vector de 14 elementos (enteros) generados aleatoriamente y distribuidos entre –30 y 30. Utilice la función `rand` de MATLAB para generar el vector inicial.

15. Escriba una función que ordene los elementos de una matriz. Utilice la siguiente línea de definición de función: `B = ordenarmatriz(A)`, donde `A` será una matriz de cualquier tamaño, y `B` será otra matriz del mismo tamaño con los elementos de `A` ordenados de forma ascendente, fila por fila. De esta forma, los elementos B(1,1) y B(m,n) serán, respectivamente, los elementos menor y mayor de la matriz ordenada. Pruebe posteriormente esta función en una matriz de dimensión 4×7, con números enteros generados aleatoriamente y distribuidos entre –30 y 30. Utilice la función `rand` de MATLAB para generar la matriz inicial.

16. Escriba un programa (fichero script) que calcule el coste de enviar un paquete en función de la siguiente tabla de precios:

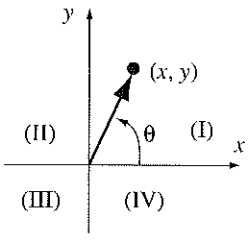
Tipo de servicio	Peso (0-2 libras)	Peso (2-10 libras)	Peso (10-50 libras)
Tierra	1,50 €	1,50 € + 0,50 € adicionales por cada libra o fracción de libra, a partir de las 2 libras de peso.	5,50 € + 0,30 € adicionales por cada libra o fracción de libra, a partir de las 10 libras de peso.
Aire	3,00 €	3,00 € + 0,50 € adicionales por cada libra o fracción de libra, a partir de las 2 libras de peso.	10,20 € + 0,60 € adicionales por cada libra o fracción de libra, a partir de las 10 libras de peso.
Nocturno	18 €	18 € + 6 € adicionales por cada libra o fracción de libra, a partir de las 10 libras de peso.	No se realizarán entregas para paquetes que pesen más de 10 libras.

El programa debe pedir al usuario que introduzca el peso y el tipo de servicio. Seguidamente, el programa visualizará el coste del servicio. Si se introduce un paquete que pese más de 50 libras para un servicio de aire o tierra, el programa visualizará un mensaje del tipo: 'No se realiza reparto por aire o tierra para paquetes con peso superior a las 50 libras'. Si se introduce el peso de

un paquete que supera las 10 libras de peso para un servicio nocturno, el programa visualizará un mensaje del tipo: 'No se realizan entregas nocturnas para paquetes que pesen mas de 10 libras'. Ejecute el programa e introduzca los valores 0,5, 6,3, 20 y 50,4 libras para servicios de tierra y aire, así como 2, 8,1 y 13 libras para el servicio de reparto nocturno.

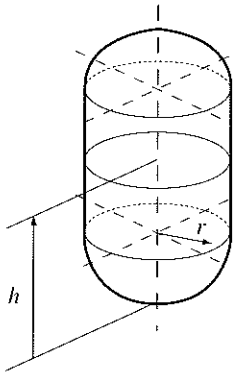
17. Sea el vector x = [1:50]. Escriba un programa en un fichero script que borre del vector x aquellos elementos que son divisibles por 3, 4 ó 5. Al final el programa debe mostrar el vector resultante.

18. Escriba una función MATLAB que calcule las coordenadas polares de un punto correspondiente a un sistema de coordenadas cartesianas, en un plano de dos dimensiones. Utilice la siguiente línea de definición de función para ello: `[theta radio] = CartesianoAPolar(x,y)`. Los argumentos de entrada serán las coordenadas cartesianas `x` e `y` del punto, y los argumentos de salida serán el ángulo θ y la distancia radial (`radio`) al punto en cuestión. El ángulo θ vendrá dado en grados, y su medida será relativa al eje `x` positivo, de tal forma que sea un número positivo en los cuadrantes I, II y III, y un número negativo en el cuadrante IV. Utilice posteriormente esta función para calcular las coordenadas polares de los puntos (15, 3), (–7, 12), (–17, –9) y (10, –6,5).

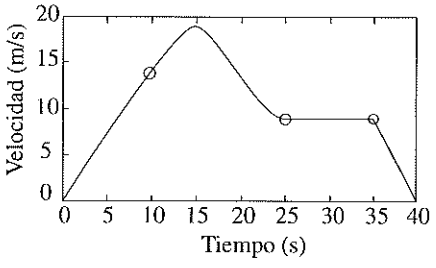


19. Un depósito de gasóleo tiene la forma de un cilindro vertical rematado por sus dos extremos hemisféricos, tal y como se muestra en la figura adjunta. El radio del cilindro y de los hemisferios es r = 40 cm, y la longitud de la parte cilíndrica es 1,2 metros.

    Escriba una función (definida de la forma: `V = Vtanque(h)`) que calcule el volumen del depósito en función de la altura h. Utilice posteriormente la función para representar un gráfico del volumen en función de la altura, para $0 \le h \le 2$ metros.



20. La velocidad, en función del tiempo, de una partícula que se mueve a lo largo de una línea recta, se representa en el gráfico adjunto y viene dada por las siguientes ecuaciones:

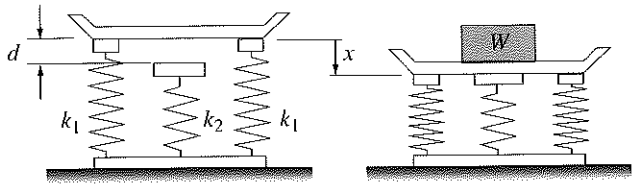


$$v(x) = \begin{cases} 1{,}4t & 0 \le t \le 10 \text{ s} \\ 14 + 5\text{sen}\left(\frac{\pi}{10}(t-10)\right) & 10 \le t \le 25 \text{ s} \\ 9 & 25 \le t \le 35 \text{ s} \\ 9 - \frac{9}{5}(t-35) & 35 \le t \le 40 \text{ s} \end{cases}$$

Escriba dos funciones MATLAB: una de ellas debe calcular la velocidad de la partícula en un instante t (utilice la siguiente definición de función: `v = velocidad(t)`), y la otra función

deberá calcular la aceleración de la partícula también en el instante t (utilice para ello la siguiente definición de función: `a = aceleracion(t)`). Escriba posteriormente un programa, en un fichero script, que represente las gráficas de la velocidad y la aceleración, en función del tiempo, de una partícula en movimiento (las dos gráficas deben aparecer en la misma ventana gráfica). Para ello, dentro del fichero script, cree primero un vector t, para $0 \leq t \leq 40$ segundos, y después utilice las funciones `velocidad` y `aceleracion` para crear los vectores `v` y `a`, que se utilizarán para generar la representación gráfica.

21. Una báscula se compone de una bandeja sujeta a una serie de muelles, tal y como se muestra en la figura adjunta. Cuando se sitúa un objeto en la bandeja, ésta se mueve hacia abajo de forma que el peso del objeto se puede calcular a partir del desplazamiento de la bandeja. Inicialmente, sólo los dos muelles exteriores soportan el peso. Sin embargo, si el objeto es lo suficientemente pesado, la bandeja hará contacto con el tercer muelle situado justo entre los otros dos exteriores.



$k_1 = 800$ N/m, $k_2 = 1700$ N/m, $d = 20$ mm.

Escriba una función que calcule el peso W de un objeto en función del desplazamiento x de la bandeja en la báscula. Utilice la siguiente definición para dicha función: `W = bascula(x)`.

a) Utilice posteriormente esta función en la Ventana de Comandos para calcular el peso de dos objetos que producen un desplazamiento de la bandeja de 1,5 y 3,1 cm.

b) Escriba un programa script que represente gráficamente el peso en función del desplazamiento x, para $0 \leq x \leq 4$ cm.

Capítulo 8
Polinomios, curvas de ajuste e interpolación

Los polinomios son expresiones matemáticas utilizadas muy frecuentemente en el modelado y resolución de problemas científicos. En muchos casos, el polinomio representa una forma práctica de crear ecuaciones para resolver determinados problemas, correspondiéndose la solución a estos problemas con la solución del polinomio. MATLAB posee una gran variedad de funciones pensadas para trabajar con polinomios. El uso de estas funciones se explicará en detalle en la Sección 8.1.

La creación de una curva de ajuste es un proceso consistente en encontrar una función que pueda ser utilizada para modelar datos. Esta función no tiene que pasar necesariamente por cada uno de los puntos que representan los datos estudiados, sino que sirve más bien para hacer un modelo con el menor error de aproximación posible. En general no existen limitaciones sobre el tipo de ecuación que se puede utilizar como curva de ajuste de datos. Sin embargo, a menudo se utilizan ajuste polinómicos, exponenciales y de potencia. En MATLAB, las curvas de ajuste se pueden realizar mediante un programa script, o analizando interactivamente los datos que se muestran en la ventana gráfica. En la Sección 8.2 se describe cómo se puede utilizar la programación MATLAB para realizar curvas de ajuste a partir de polinomios u otro tipo de funciones. La Sección 8.4 describe la interfaz básica para realizar ajustes e interpolación de forma interactiva.

La interpolación es el proceso consistente en estimar valores a partir de datos que representan puntos concretos del espacio. La interpolación más sencilla se realiza dibujando líneas rectas entre los puntos. La versión más sofisticada consiste, sin embargo, en utilizar datos de puntos adicionales a los que originalmente se tienen. Las Secciones 8.3 y 8.4 tratan en detalle cómo llevar a cabo varios tipos de interpolación utilizando MATLAB.

8.1 Polinomios

Los polinomios son funciones que tienen la forma:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

donde los coeficientes a_n, a_{n-1}, ... , a_1, a_0 son números reales, y n un número entero positivo que es el grado u orden del polinomio. Algunos ejemplos de polinomios son los que se muestran a continuación:

$f(x) = 5x^5 + 6x^2 + 7x + 3$ Polinomio de grado 5.

$f(x) = 2x^2 - 4x + 10$ Polinomio de grado 2.

$f(x) = 11x - 5$ Polinomio de grado 1.

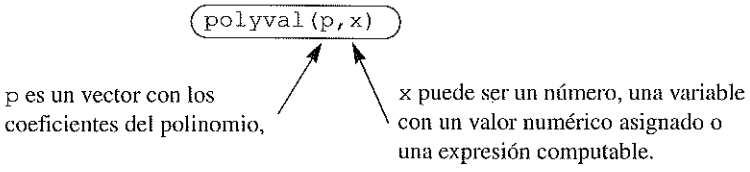
$f(x) = 6$ Constante. Polinomio de grado 0.

En MATLAB los polinomios se representan mediante un vector fila en el cual los elementos simbolizan los coeficientes del polinomio: a_n, a_{n-1}, ... , a_1, a_0. El primer elemento es el coeficiente de la variable x de mayor grado. El vector debe contener todos los coeficientes, incluidos los que son 0. Por ejemplo:

Polinomio	Representación en MATLAB mediante un vector	
$8x + 5$	$(8x + 5)$	p = [8 5]
$2x^2 - 4x + 10$	$(2x^2 - 4x + 10)$	d = [2 –4 10]
$6x^2 - 150$	$(6x^2 + 0x - 150)$	h = [6 0 –150]
$5x^5 + 6x^2 - 7x$	$(5x^5 + 0x^4 + 0x^3 + 6x^2 - 7x + 0)$	c = [5 0 0 6 –7 0]

8.1.1 *Valor de un polinomio*

El valor de un polinomio en un punto x se puede calcular mediante la función `polyval`, que tiene la siguiente sintaxis:



`x` también puede ser una matriz o un vector. En este caso el valor se calculará para cada uno de los elementos que contiene el array, utilizando operaciones elemento a elemento; el resultado será un vector, o una matriz, con los valores correspondientes al polinomio.

Problema de ejemplo 8.1: Cálculo de polinomios con MATLAB

Sea el polinomio $f(x) = x^5 - 12{,}1x^4 + 40{,}59x^3 - 17{,}015x^2 - 71{,}95x + 35{,}88$

a) Calcular $f(9)$.

b) Representar gráficamente el polinomio $f(x)$ en el dominio $-1{,}5 \leq x \leq 6{,}7$.

Solución

El problema se resolverá directamente en la Ventana de Comandos.

a) Se crea un vector `p` con los coeficientes del polinomio. Seguidamente se utiliza la función `polyval` para calcular el valor del polinomio en el punto $x = 9$.

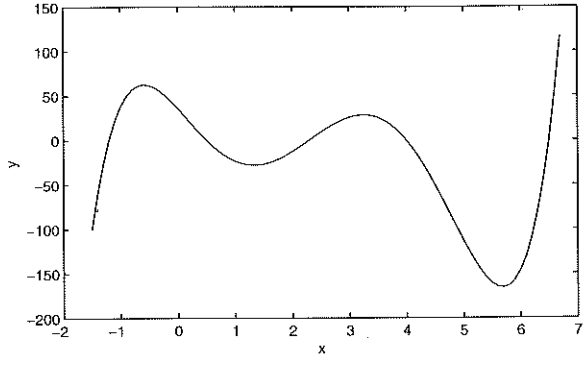
```
>> p = [1 -12.1 40.59 -17.015 -71.95 35.88];
>> polyval(p,9)
ans =
   7.2611e+003
```

b) Para representar gráficamente el polinomio en función del vector x, primero se define un rango de valores con el dominio de representación del polinomio (desde –1,5 hasta 6,7). Seguidamente se crea un vector y con los valores del polinomio para cada valor o elemento de x. Finalmente se lleva a cabo la representación gráfica de y en función de x.

```
>> x = -1.5:0.1:6.7;
>> y = polyval(p,x);
>> plot(x,y)
```

Se calcula el valor del polinomio para cada elemento del vector x.

La representación gráfica generada por MATLAB es la siguiente (las etiquetas de los ejes se han añadido utilizando el editor gráfico):



8.1.2 *Raíces de un polinomio*

Las raíces de un polinomio son los valores de x que hacen que el valor del polinomio sea igual a cero. Por ejemplo, las raíces del polinomio $f(x) = x^2 - 2x - 3$ son los valores de x para los cuales $x^2 - 2x - 3 = 0$. Estos valores son $x = -1$ y $x = 3$.

MATLAB posee una función, denominada `roots`, que permite calcular las raíces de un polinomio. La sintaxis de esta función es:

```
r = roots(p)
```

`r` es un vector columna con las raíces del polinomio,

`p` es un vector fila con los coeficientes del polinomio.

Por ejemplo, las raíces del polinomio del Problema de ejemplo 8.1 se pueden calcular de la siguiente manera:

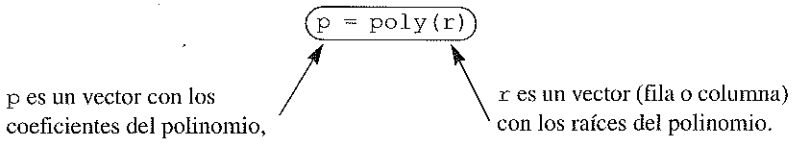
```
>> p = [1 -12.1 40.59 -17.015 -71.95 35.88];
>> r = roots(p)
r =
   6.5000
   4.0000
   2.3000
  -1.2000
   0.5000
```

Cuando las raíces son conocidas, el polinomio se puede escribir de la forma:
$f(x) = (x + 1{,}2)(x - 0{,}5)(x - 2{,}3)(x - 4)(x - 6{,}5)$

El comando `roots` es muy útil para calcular las raíces de una ecuación cuadrática. Por ejemplo, para encontrar las raíces de $f(x) = 4x^2 + 10x - 8$, sólo hay que teclear:

```
>> roots([4 10 -8])
ans =
  -3.1375
   0.6375
```

Cuando las raíces de un polinomio son conocidas, se puede utilizar el comando `poly` para calcular los coeficientes del polinomio. La sintaxis de este comando es:



Por ejemplo, los coeficientes del polinomio del Problema de ejemplo 8.1 se pueden obtener a partir de las raíces del propio polinomio (ver más arriba), de la forma:

```
>> r = [6.5 4 2.3 -1.2 0.5];
>> p = poly(r)
p =
   1.0000  -12.1000   40.5900  -17.0150  -71.9500   35.8800
```

8.1.3 *Suma, multiplicación y división de polinomios*

Suma:

Dos polinomios pueden ser sumados o restados sumando o restando sus vectores de coeficientes. Si los polinomios no tienen el mismo grado (los vectores de coeficiente tienen distinto tamaño), el vector

más corto debe ser modificado, añadiendo ceros por la izquierda, para que tenga la misma longitud que el vector más largo. Por ejemplo, los polinomios:

$$f_1(x) = 3x^6 + 15x^5 - 10x^3 - 3x^2 + 15x - 40 \quad \text{y} \quad f_2(x) = 3x^3 - 2x - 6$$

se pueden sumar de la siguiente forma:

```
>> p1 = [3 15 0 -10 -3 15 -40];
>> p2 = [3 0 -2 -6];
>> p = p1 + [0 0 0 p2]
p =
   3   15   0   -7   -3   13   -46
```

Se añaden ceros a la izquierda de `p2`, ya que `p1` es de grado 6 y `p2` es de grado 3.

Multiplicación:

Para multiplicar dos polinomios se utiliza la función `conv` de MATLAB, cuya sintaxis es la siguiente:

```
c = conv(a,b)
```

`c` es un vector que contiene los coeficientes del polinomio producto, resultado de la multiplicación.

`a` y `b` son vectores que contienen los coeficientes de los polinomios a multiplicar.

- En el caso de la multiplicación, los polinomios no tienen por qué ser del mismo grado.
- La multiplicación de tres o más polinomios se lleva a cabo mediante el uso repetitivo de la función `conv`.

Por ejemplo, el producto de los polinomios $f_1(x)$ y $f_2(x)$ anteriores da como resultado:

```
>> pm = conv(p1,p2)
pm =
   9   45   -6   -78   -99   65   -54   -12   -10   240
```

que se corresponde con el polinomio:

$$9x^9 + 45x^8 - 6x^7 - 78x^6 - 99x^5 + 65x^4 - 54x^3 - 12x^2 - 10x + 240$$

División:

Para dividir un polinomio entre otro se utiliza la función `deconv`, cuya sintaxis es la siguiente:

```
[q,r] = deconv(u,v)
```

`q` es un vector que contiene los coeficientes del polinomio cociente de la división, y `r` es otro vector que contiene los coeficientes del polinomio resto de la división.

`u` es un vector que contiene los coeficientes de polinomio numerador, y `v` otro vector con los coeficientes del polinomio denominador.

Por ejemplo, la división de $2x^3 + 9x^2 + 7x - 6$ entre $x + 3$ se puede llevar a cabo de la siguiente forma:

```
>> u = [2 9 7 -6];
>> v = [1 3];
>> [a b] = deconv(u,v)
a =
   2   3   -2
b =
   0   0   0   0
```

El resultado (cociente) de la operación es el polinomio: $2x^2 + 3x - 2$.

El resto de la división es cero.

Veamos a continuación otro ejemplo de división con resto distinto de cero. Se trata de dividir el polinomio $2x^6 - 13x^5 + 75x^3 + 2x^2 - 60$ entre $x^2 - 5$:

```
>> w = [2 -13 0 75 2 0 -60];
>> z = [1 0 -5];
>> [g h ] = deconv(w,z)
g =
   2   -13   10   10   52
h =
   0   0   0   0   0   50   200
```

El cociente es: $2x^4 - 13x^3 + 10x^2 + 10x + 52$.

El resto es: $50x - 200$.

Es decir, el resultado de la división es: $2x^4 - 13x^3 + 10x^2 + 10x + 52 + \dfrac{50x + 200}{x^2 - 5}$.

8.1.4 *Derivada de un polinomio*

Para calcular la derivada de un polinomio se utiliza la función predefinida por MATLAB `polyder`. Esta función también se puede utilizar para calcular la derivada del producto y del cociente de dos polinomios. Por tanto, este comando tiene tres sintaxis distintas:

`k = polyder(p)`	Derivada de un polinomio individual, donde `p` es un vector con los coeficientes del polinomio y `k` es un vector con los coeficientes de la derivada del polinomio `p`.
`k = polyder(a,b)`	Derivada del producto de dos polinomios cuyos vectores de coeficientes son `a` y `b`. El resultado es un vector `k` con los coeficientes de la derivada del producto de polinomios.
`[n d] = polyder(u,v)`	Derivada del cociente de dos polinomios cuyos vectores de coeficientes `u` y `v` representan los polinomios numerador y denominador. Por otro lado, `n` y `d` son los vectores con los coeficientes de los polinomios numerador y denominador después de realizar la derivada de la división.

La única diferencia entre los dos últimos comandos es el número de argumentos de salida. Si MATLAB detecta dos argumentos de salida calcula la derivada del cociente de dos polinomios, mientras que si detecta uno calculará la derivada del producto.

Por ejemplo, si $f_1(x) = 3x^2 - 2x + 4$ y $f_2(x) = x^2 + 5$, la derivada de $3x^2 - 2x + 4$, la derivada de $(3x^2 - 2x + 4)(x^2 + 5)$ y la derivada de $\dfrac{3x^2 - 2x + 4}{x^2 + 5}$ se pueden calcular de la forma:

```
>> f1 = [3 -2 4]; f2 = [1 0 5];
>> k = polyder(f1)
k =
   6   -2
>> d = polyder(f1,f2)
d =
   12   -6   38   -10
>> [n d] = polyder(f1,f2)
n =
   2   22   -10
d =
   1   0   10   0   25
```

Se crean los vectores de coeficientes f_1 y f_2.

La derivada de f_1 es: $6x - 2$.

La derivada de $f_1 * f_2$ es: $12x^3 - 6x^2 + 38x - 10$.

La derivada de $\dfrac{3x^2 - 2x + 5}{x^2 + 5}$ es $\dfrac{2x^2 + 22x - 10}{x^4 + 10x^2 + 25}$.

8.2 Curvas de ajuste

El cálculo de curvas de ajuste, también llamado análisis de regresión, es un proceso que consiste en ajustar mediante una función un conjunto de datos representados por puntos. Esta función puede ser utilizada posteriormente como modelo matemático para los datos. Como hay diferentes tipos de funciones (lineales, logarítmicas, exponenciales, etc.) el cálculo de curvas de ajuste puede llegar a ser un proceso complejo. En determinadas ocasiones existe a priori una idea predeterminada sobre el tipo de función que se puede utilizar para ajustar los datos. En este caso el proceso se simplifica, ya que sólo habrá que calcular los coeficientes de dicha función. En otros problemas, cuando no se sabe nada sobre el tipo de datos con los que se trabaja, lo normal es realizar primero distintas representaciones gráficas de los datos para tener una idea sobre el tipo de función que se puede utilizar para obtener el mejor ajuste posible de los datos. Esta sección describe algunas técnicas básicas para llevar a cabo curvas de ajuste, así como las herramientas que MATLAB ofrece para este propósito.

8.2.1 *Curvas de ajuste mediante polinomios. La función* `polyfit`

Se pueden emplear polinomios para realizar ajustes sobre datos (puntos) de dos formas distintas. En una de ellas el polinomio pasa por todos los puntos dados, y en la otra el polinomio no pasa necesariamente por todos los puntos, pero en general ofrece una buena aproximación de los datos que se tienen. A continuación se describen estas dos formas de trabajar.

Polinomios que pasan por todos los puntos:

Cuando se tienen n puntos de tipo (x_i, y_i), es posible encontrar un polinomio de grado $n - 1$ que pase por todos los puntos. Por ejemplo, si tenemos dos puntos es posible escribir una ecuación lineal de la forma $y = mx + b$ que pase por estos dos puntos. Si lo que tenemos son tres puntos, la ecuación tendrá la forma

$y = ax^2 + bx + c$. Generalizando, para n puntos tendremos un polinomio del tipo: $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$. El coeficiente de los polinomios se obtendrá sustituyendo cada punto en la expresión general anterior, resolviendo posteriormente las n ecuaciones resultantes. Como se verá más adelante en esta sección, los polinomios de grado alto pueden producir errores considerables si se utilizan para estimar los valores que se encuentran entre datos (puntos).

Polinomios que no pasan necesariamente por todos los puntos:

Cuando se tienen n puntos, es posible definir un polinomio de grado menor a $n - 1$ que no tiene por qué pasar necesariamente por todos los puntos dados, pero que puede darnos una buena aproximación de los datos. El método más común para encontrar el mejor ajuste es el método de los mínimos cuadrados. En este método los coeficientes del polinomio se calculan minimizando la suma de los cuadrados de los valores residuales de todos los puntos (datos) estudiados. El valor residual de un punto se define como la diferencia entre el valor del polinomio (en ese punto) y el del punto dado. Por ejemplo, considérese el caso en que se pretende encontrar la ecuación de la línea recta que mejor se ajuste a cuatro datos (puntos), tal y como se muestra en la Figura 8.1.

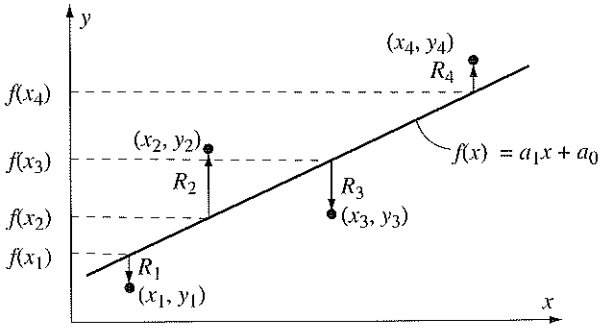


Figura 8.1: Ajuste por mínimos cuadrados mediante un polinomio de primer grado para cuatro puntos.

Los puntos son (x_1, y_1), (x_2, y_2), (x_3, y_3) y (x_4, y_4). El polinomio de primer grado necesario para el ajuste es: $f(x) = a_1x + a_0$. El valor residual R_i en cada punto es la diferencia entre el valor de la función en x_i e y_i, es decir, $R_i = f(x_i) - y_i$. Como se pretende obtener el cuadrado de cada residuo, extendiendo esta expresión para todos los puntos se tendrá:

$$R = [f(x_1) - y_1]^2 + [f(x_2) - y_2]^2 + [f(x_3) - y_3]^2 + [f(x_4) - y_4]^2$$

o, después de sustituir la ecuación del polinomio en cada punto:

$$R = [a_1x_1 + a_0 - y_1]^2 + [a_1x_2 + a_0 - y_2]^2 + [a_1x_3 + a_0 - y_3]^2 + [a_1x_4 + a_0 - y_4]^2$$

Finalmente R_i queda en función de a_1 y a_0. Para obtener el mínimo de R se calcula la derivada parcial de R con respecto a a_1 y a_0 (dos ecuaciones), igualando las expresiones resultantes a cero.

$$\frac{\partial R}{\partial a_1} = 0 \quad \text{y} \quad \frac{\partial R}{\partial a_0} = 0$$

El resultado es un sistema de ecuaciones con dos incógnitas, a_1 y a_0. La solución de este sistema de ecuaciones proporciona los valores de los coeficientes del polinomio que mejor se ajusta a los datos. Se puede seguir el mismo procedimiento en el caso de tener más puntos, así como polinomios de grado superior. El lector que desee indagar más sobre el método de mínimos cuadrados puede consultar cualquier libro sobre análisis numérico.

La forma de realizar ajustes con polinomios en MATLAB es mediante la función `polyfit`, cuya sintaxis más básica se muestra a continuación:

```
p = polyfit(x,y,n)
```

p es el vector con los coeficientes del polinomio de ajuste.

x es un vector con las coordenadas horizontales de los datos (variable independiente), y es el vector con las coordenadas verticales (variable dependiente) de los datos, n es el grado del polinomio de ajuste deseado.

La función `polyfit` se utiliza para calcular polinomios de ajuste sobre m puntos, para cualquier grado hasta, como máximo, $m - 1$. Si $n = 1$, el polinomio resultante será una línea recta, si $n = 2$ será una parábola, y así sucesivamente. El polinomio calculado pasará por todos los puntos si $n = m - 1$ (el grado del polinomio es menor que el número de puntos). Es necesario señalar que los polinomios con grados elevados, o que pasen por todos los puntos, no dan siempre el mejor ajuste posible. Los polinomios de grado alto pueden a veces desviarse significativamente entre algunos puntos dados.

En la Figura 8.2 se muestra cómo polinomios de distinto grado pueden ajustar el mismo conjunto de datos iniciales. Los puntos estudiados son: (0,9, 0,9), (1,5, 1,5), (3, 2,5), (4, 5,1), (6, 4,5), (8, 4,9) y (9,5, 6,3). Estos puntos se ajustan utilizando la función `polyfit` con polinomios de grado uno a seis. Cada gráfico de la Figura 8.2 muestra el mismo conjunto de datos, marcados mediante círculos, y una

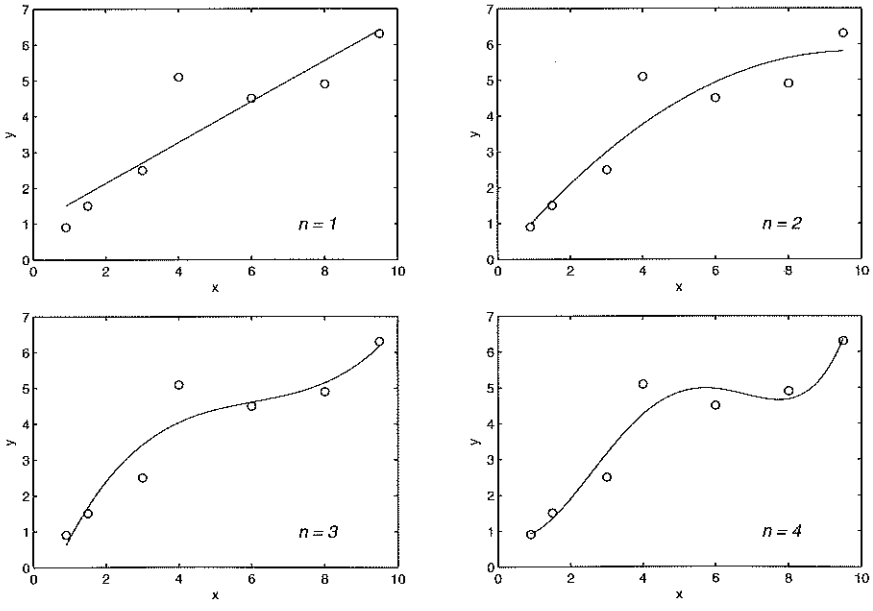


Figura 8.2: Ajuste de datos mediante polinomios de distinto grado.

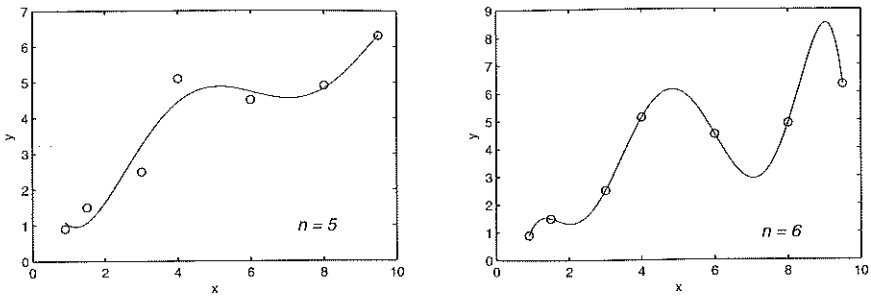


Figura 8.2: Ajuste de datos mediante polinomios de distinto grado. (*Continuación*)

curva de ajuste que se corresponde con un polinomio de un determinado grado. Como puede verse, el polinomio de grado $n = 1$ es una línea recta, mientras que el polinomio de grado $n = 2$ es una línea ligeramente curva. Al incrementar el grado del polinomio la curva se ajusta más a los puntos. Cuando $n = 6$, la línea pasa por todos los puntos. Sin embargo, entre algunos de los puntos, la línea se desvía significativamente de la tendencia original marcada por los datos estudiados.

A continuación se muestra el fichero script utilizado para generar uno de los gráficos de la Figura 8.2, en concreto el gráfico para $n = 3$. Para realizar esta representación se creó un vector `xp` con un espaciado de una magnitud muy pequeña. Este vector se utiliza posteriormente en la función `polyval` para crear el vector `yp` con los valores del polinomio para cada elemento de `xp`.

```
x = [0.9 1.5 3 4 6 8 9.5];
y = [0.9 1.5 2.5 5.1 4.5 4.9 6.3];
p = polyfit(x,y,3)
xp = 0.9:0.1:9.5;
yp = polyval(p,xp);
plot(x,y,'o',xp,yp)
xlabel('x');
ylabel('y');
```

Se crean los vectores `x` e `y` con los valores de las coordenadas de los puntos a estudiar.

Se crea un vector `p` utilizando `polyfit`.

Se crea un vector `xp` para representar gráficamente el polinomio.

Se crea un vector `yp` con los valores del polinomio en cada punto `xp`.

Se representan gráficamente el polinomio y los siete puntos estudiados.

Una vez que se ejecuta el fichero script anterior, en la Ventana de Comandos se muestra la siguiente salida:

```
p =
   0.0220   -0.4005    2.6138   -1.4158
```

Esto significa que el polinomio de tercer grado de la Figura 8.2 tiene la forma:

$$0{,}0220x^3 - 0{,}4005x^2 + 2{,}6138x - 1{,}4158.$$

8.2.2 *Otras curvas de ajustes*

En ciencia e ingeniería, así como en muchas otras situaciones, es necesario utilizar funciones distintas de las polinómicas para ajustar datos. Desde el punto de vista teórico se puede utilizar cualquier función para modelar datos dentro de un rango establecido. Sin embargo, para un conjunto específico de datos algunas funciones proporcionan mejores ajustes que otras. Además, determinar los mejores coeficientes de ajuste de algunas funciones puede resultar más complejo que en otras. En esta sección se abordará el ajuste a curvas mediante funciones que se utilizan habitualmente: potencia, exponencial, logarítmica y recíproca. La forma de estas funciones es la siguiente:

$y = bx^m$ (función potencia)

$y = be^{mx}$ o $y = b10^{mx}$ (función exponencial)

$y = m\ln(x) + b$ o $y = m\log(x) + b$ (función logarítmica)

$y = \dfrac{1}{mx + b}$ (función recíproca)

Es fácil ajustar datos mediante estas funciones utilizando el comando `polyfit`. Para ello sólo hay que escribir de nuevo las funciones en forma de polinomio lineal ($n = 1$), respetando la forma:

$$y = mx + b$$

La función logarítmica ya tiene esta forma, para el resto de ecuaciones se rescribiría:

$\ln(y) = m\ln(x) + \ln(b)$ (función potencia)

$\ln(y) = mx + \ln(b)$ o $\log(y) = mx + \log(b)$ (función exponencial)

$\dfrac{1}{y} = mx + b$ (función recíproca)

Estas ecuaciones reflejan una relación lineal entre $\ln(y)$ y $\ln(x)$ para la función potencia, entre $\ln(y)$ y x para la función exponencial, entre y y $\ln(x)$ o $\log(x)$ para la función logarítmica y entre $1/y$ y x para la función recíproca. Esto significa que se puede utilizar el comando `polyfit(x,y,1)` para calcular las constantes m y b que permitan el mejor ajuste posible si se utilizan los siguientes argumentos en lugar de sólo `x` e `y`:

Función		**Forma de uso del comando** `polyfit`
potencia	$y = bx^m$	`p = polyfit(log(x),log(y),1)`
exponencial	$y = be^{mx}$ o bien	`p = polyfit(x,log(y),1)` o bien
	$y = b10^{mx}$	`p = polyfit(x,log10(y), 1)`
logarítmica	$y = m\ln(x) + b$ o bien	`p = polyfit(log(x),y,1)` o bien
	$y = m\log(x) + b$	`p = polyfit(log10(x),y,1)`
recíproca	$y = \dfrac{1}{mx + b}$	`p = polyfit(x,1./y,1)`

El resultado de la función `polyfit` se asigna a `p`, que es un vector de dos elementos. El primer elemento, `p(1)`, es la constante `m`, y el segundo elemento, `p(2)`, es `b` para las funciones logarítmica y recíproca, ln (b) o log (b) para la función exponencial y ln (b) para la función potencia ($b = e^{p(2)}$ o $b = 10^{p(2)}$ para la función exponencial y $b = e^{p(2)}$ para la función potencia).

Para un conjunto dado de datos, es posible predecir, hasta cierto punto, cuál de las funciones proporcionará un buen ajuste. Para ello hay que representar los datos utilizando diferentes combinaciones de ejes lineales y logarítmicos. Si en uno de los gráficos los datos parecen ajustarse a una línea recta, entonces la función correspondiente puede proporcionar un buen ajuste según la siguiente lista;

Eje x	Eje y	Función
lineal	lineal	lineal $y = mx + b$
logarítmica	logarítmica	potencia $y = bx^m$
lineal	logarítmica	exponencial $y = be^{mx}$ o bien $y = b10^m$
logarítmica	lineal	logarítmica $y = m\ln(x) + b$ o bien $y = m\log(x) + b$
lineal	lineal plot $(1/y)$	recíproca $y = \dfrac{1}{mx + b}$

Otras consideraciones a la hora de elegir una función de ajuste:

- Las funciones exponenciales no pasan por el origen.
- Las funciones exponenciales sólo sirven para realizar ajustes de datos cuyos valores y sean todos positivos o todos negativos.
- Las funciones logarítmicas no pueden modelar valores nulos (cero) o negativos de x.
- En la función potencia, $y = 0$ cuando $x = 0$.
- La función recíproca no puede modelar $y = 0$.

El ejemplo siguiente muestra el proceso habitual de ajuste a una función a partir de una serie de datos conocidos.

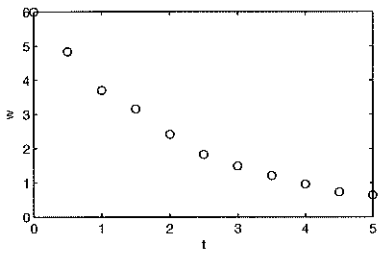
Problema de ejemplo 8.2: Curva de ajuste para una serie de puntos

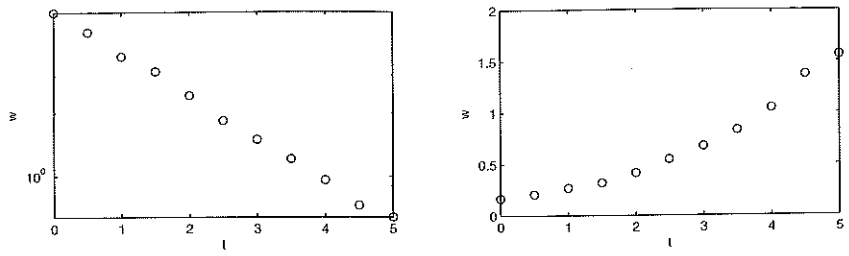
Encontrar una función $w = f(t)$ (t es la variable independiente y w la dependiente) que mejor ajuste los datos que se detallan a continuación.

t	0,0	0,5	1,0	1,5	2,0	2,5	3,0	3,5	4,0	4,5	5,0
w	6,00	4,83	3,70	3,15	2,41	1,83	1,49	1,21	0,96	0,73	0,64

Solución

En primer lugar se representan los datos con escalas lineales en ambos ejes. La figura que se muestra a la derecha indica que una función lineal no proporciona el mejor ajuste porque los puntos parecen no seguir una línea recta. De las restantes funciones, la logarítmica también se excluye, ya que el primer punto es $t = 0$. Lo mismo sucede con la función potencia, ya que para $t = 0$, $w \neq 0$. Para averiguar cuál de las otras dos funciones posibles (exponencial y recíproca) proporcionará el mejor ajuste, se representan nuevamente los datos, como se muestra en los dos gráficos de abajo. El de la izquierda tiene escala logarítmica en el eje vertical y lineal en el horizontal. El de la derecha tiene escala lineal en ambos ejes, y $1/w$ se representa en el eje vertical.





En el gráfico de la izquierda los puntos parecen seguir una línea recta. Esto indica que una función exponencial, de la forma $y = be^{mx}$, puede dar un mejor ajuste para estos datos. A continuación se muestra el fichero script que calcula las constantes b y m y que dibuja los puntos y el ajuste calculado.

```
t = 0:0.5:5;
w = [6 4.83 3.7 3.15 2.41 1.83 1.49 1.21 0.96 0.73 0.64];
p = polyfit(t,log(w),1);
m = p(1)
b = exp(p(2))
tm = 0:0.1:5;
wm = b*exp(m*tm);
plot(t,w,'o',tm,wm)
```

Se crean los vectores `t` y `w` con las coordenadas de los datos.

Se utiliza la función `polyfit` con `t` y `log(w)`.

Se calcula el coeficiente `b`.

Se crea un vector `tm` para dibujar el polinomio.

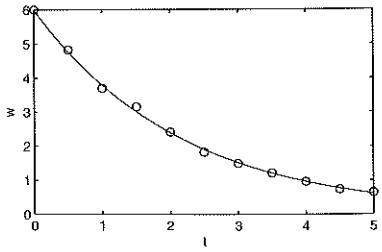
Se calcula el valor de la función para cada elemento `tm`.

Se representan tanto los datos como la función de ajuste.

Cuando se ejecuta, el programa calcula los valores de las constantes m y b y los muestra en la Ventana de Comandos.

```
m =
   -0.4580

b =
    5.9889
```

Además, el gráfico generado por el programa muestra los puntos y la función de ajuste (las etiquetas de los ejes se añadieron posteriormente con el editor gráfico):



Es importante señalar que, además de las funciones de ajustes ya vistas, se pueden escribir muchas otras funciones en una forma apropiada para utilizarlas en el comando `polyfit`. En el problema de ejemplo 8.7 se utiliza la función $y = e^{(a_2x^2 + a_1x + a_0)}$ para ajustar datos utilizando `polyfit` con un polinomio de tercer grado.

8.3 Interpolación

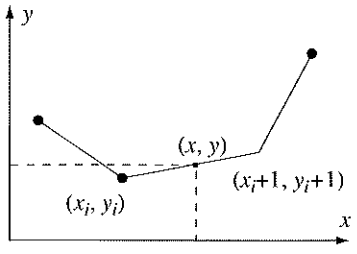
La interpolación es un proceso para estimar valores que se encuentran entre los puntos (datos) conocidos. MATLAB posee funciones para la interpolación basadas en polinomios y en transformadas de Fourier. En esta sección veremos las primeras, pero las segundas no se tratarán en este libro. En el proceso de interpolación unidimensional, a cada punto se le asigna una variable independiente (x) y una variable dependiente (y). La interpolación en dos dimensiones necesita de dos variables independientes (x e y) y de una variable dependiente (z).

Interpolación unidimensional:

Cuando sólo se tienen dos puntos (datos) conocidos, éstos se pueden unir mediante una línea recta. Es decir, se puede utilizar la ecuación de la recta (polinomio de primer grado) para estimar diferentes puntos entre dos conocidos. Como se vio en la sección anterior, si se conocen tres o cuatro puntos, se puede determinar un polinomio de segundo o tercer grado que pase por esos puntos y utilizarlo para estimar valores que se encuentran entre ellos. Cuanto mayor es el número de puntos conocidos, mayor será el grado del polinomio necesario para que pase por todos esos puntos. Sin embargo, el polinomio resultante no tiene por qué proporcionar necesariamente una buena aproximación de los valores que se encuentran entre los puntos conocidos. Este hecho se ilustra en la Figura 8.2 para $n = 6$.

Si en lugar de considerar todos los puntos conocidos (utilizando el polinomio que pasa por todos esos puntos) se consideran sólo unos pocos puntos pertenecientes al entorno donde se va a realizar la interpolación, se puede obtener una interpolación mucho más precisa. En este método, llamado interpolación segmentaria o *spline*, se utilizan varios polinomios de grado relativamente bajo que se aplican cada uno de ellos sólo a una pequeña región del conjunto de datos conocidos.

La forma más sencilla de interpolación segmentaria es la llamada interpolación segmentaria lineal. Este método, que se muestra en el gráfico de la derecha, consiste en unir dos puntos adyacentes (conocidos) mediante una recta (polinomio de primer grado). Esta ecuación de la recta que pasa por dos puntos adyacentes, (x_i, y_j) y (x_{i+1}, y_{j+1}), se puede utilizar para calcular el valor de y para cualquier x que se encuentre entre esos puntos. Esta ecuación es:

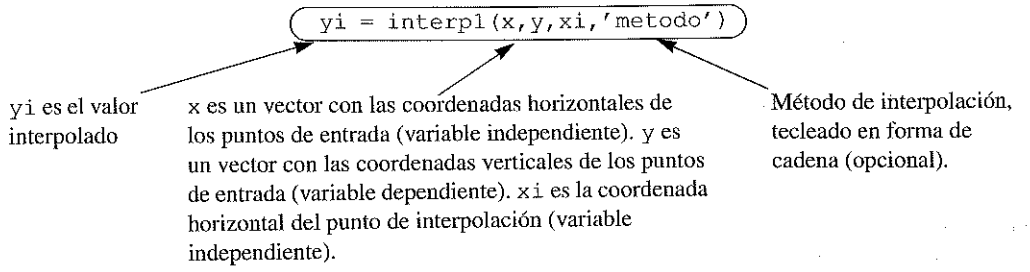


$$y = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}x + \frac{y_i x_{i+1} - y_i x_i}{x_{i+1} - x_i}$$

En la interpolación lineal, la recta que pasa por dos puntos tiene pendiente constante, pero la pendiente cambia en todos los puntos. Se puede obtener una curva de interpolación más suave si se utilizan polinomios cuadráticos o cúbicos. En estos métodos, denominados interpolación segmentaria cuadrática o cúbica, se utilizan polinomios de segundo o tercer grado para interpolar entre todos los pares de puntos. Los coeficientes de los polinomios se calculan utilizando datos adicionales provenientes de puntos adyacentes a los dos seleccionados. La explicación teórica sobre el cálculo de las constantes de los

polinomios está fuera del alcance de este libro, aunque esta información se puede consultar en cualquier bibliografía relacionada con el cálculo o análisis numérico.

En MATLAB las interpolaciones unidimensionales se realizan mediante la función `interp1` (el último carácter es el número uno), que tiene la siguiente sintaxis:



- El vector `x` debe ser monótono (los elementos deben estar en orden ascendente o descendente).
- `xi` puede ser un escalar (interpolación con un punto) o un vector (interpolación con varios puntos). Dependiendo de uno u otro caso, `yi` será un escalar o un vector con los correspondientes valores interpolados.
- MATLAB puede realizar interpolaciones mediante distintos métodos programados, que pueden especificarse en el parámetro `'metodo'` mediante las cadenas:

`'nearest'`	devuelve el valor del punto más cercano al interpolado.
`'linear'`	utiliza interpolación segmentaria lineal.
`'spline'`	utiliza interpolación segmentaria cúbica.
`'pchip'`	utiliza interpolación de Hermite cúbica (también `'cubic'`).

- Cuando se utilizan los métodos `'nearest'` y `'linear'`, los valores `xi` deben estar dentro del dominio de `x`. Por el contrario, si se utilizan los métodos `'spline'` o `'pchip'`, `xi` puede contener valores fuera del dominio de `x`.
- El método `'spline'` puede producir errores considerablemente grandes si los datos de entrada no son uniformes, es decir, si algunos puntos están más próximos entre sí que otros.
- La especificación del método de interpolación (`'metodo'`) es opcional. Si no se especifica ningún método, se utilizará el método por defecto: `'linear'`.

Problema de ejemplo 8.3: Interpolación

Los puntos que se especifican en la tabla de más abajo son todos puntos pertenecientes a la función $f(x) = 1{,}5^x\cos(2x)$. Utilizar los métodos de interpolación `'linear'`, `'spline'` y `'pchip'` para calcular el valor de y entre esos puntos. Dibujar un gráfico para cada método de interpolación. Representar en esos gráficos los puntos, la función y la curva correspondiente al método de interpolación.

x	0	1	2	3	4	5
y	1,0	–0,6242	–1,4707	3,2406	–0,7366	–6,3717

Solución

El siguiente programa script resuelve el problema enunciado:

```
x = 0:1.0:5;
y = [1.0 -0.6242 -1.4707 3.2406 -0.7366 -6.3717];
xi = 0:0.1:5;
yilin = interp1(x,y,xi,'linear');
yispl = interp1(x,y,xi,'spline');
yipch = interp1(x,y,xi,'pchip');
yfun = 1.5.^xi.*cos(2*xi)
subplot(1,3,1)
plot(x,y,'o',xi,yfun,xi,yilin,'--');
subplot(1,3,2)
plot(x,y,'o',xi,yfun,xi,yispl,'--');
subplot(1,3,3)
plot(x,y,'o',xi,yfun,xi,yipch,'--');
```

Crea los vectores `x` e `y` con las coordenadas de los puntos.

Crea el vector `xi` con los puntos para la interpolación.

Calcula el vector *y* de interpolación lineal.

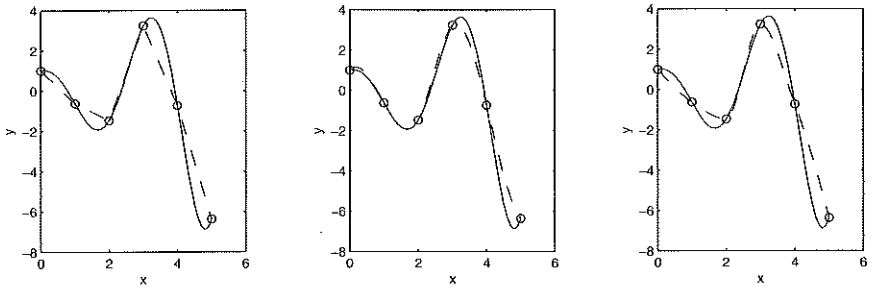
Calcula el vector *y* de interpolación segmentaria.

Calcula el vector *y* de interpolación de Hermite cúbica.

Calcula los puntos *y* con la función.

Representa gráficamente los puntos, la función y la curva de interpolación en cada caso.

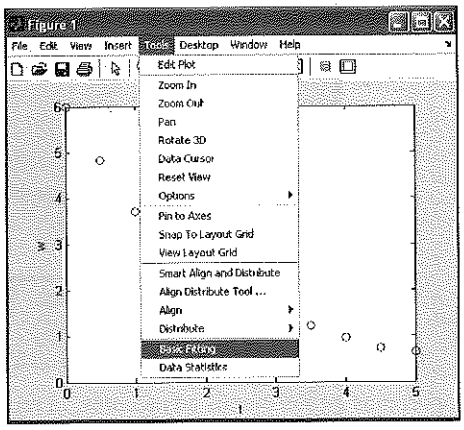
Abajo se muestran las tres gráficas generadas por el programa (las etiquetas de los ejes se añadieron posteriormente mediante el editor gráfico). Los puntos conocidos se indican mediante marcadores circulares, la curva de interpolación mediante una línea discontinua, y la función original con una línea sólida. La gráfica de la izquierda representa la interpolación tipo '`linear`', la del centro la '`spline`', y la de la derecha la interpolación de tipo '`pchip`'.



8.4 Interfaz básica para el ajuste

La interfaz básica para el ajuste es una herramienta utilizada para llevar a cabo la interpolación y la creación de curvas de ajuste de manera interactiva. Por medio de esta interfaz un usuario puede:

- Construir curvas de ajuste para datos con polinomios de distintos grados, hasta un máximo de 10, así como la utilización de métodos de interpolación segmentaria y de Hermite.
- Representar simultáneamente varios ajustes en un solo gráfico de manera que se puedan comparar.
- Representar los valores residuales de varios polinomios de ajuste y compararlos con el valor de la norma de los residuos.
- Calcular el valor de puntos concretos mediante los distintos ajustes realizados.
- Añadir al gráfico las ecuaciones de los polinomios.



Para utilizar la interfaz básica de interpolación el usuario tiene primero que hacer una representación gráfica de los datos con los que va a trabajar. Seguidamente la interfaz se activa seleccionando la opción **Basic Fitting** del menú **Tools**, tal y como se muestra en la figura de la derecha. En la Figura 8.3 se muestra la ventana correspondiente a la interfaz básica de interpolación. Cuando esta ventana se abre, sólo aparece un panel (panel **Plot fits** o de gráfico de ajuste). Esta ventana se puede extender para que muestre un segundo panel (el panel **Numerical results** o de resultados numéricos), pulsando para ello sobre el botón →. Un solo click del ratón abrirá la primera sección del panel, y un segundo click permitirá que la ventana adquiera el aspecto que se muestra en la Figura 8.3. La ventan puede ser de nuevo reducida pulsando sobre el botón ←. Los primeros dos bloques de opciones de la Ventana de la Interfaz Básica de Ajuste están relacionados con la selección de los datos o puntos a estudiar:

Select data (seleccionar datos): En un gráfico que contiene más de un conjunto de datos, se utiliza para seleccionar un conjunto específico de datos o puntos para la creación de curvas de ajuste. Sólo se puede seleccionar un conjunto de entre todos los existentes, aunque se pueden llevar a cabo varios ajustes de forma simultánea sobre el mismo conjunto.

Center and scale X data (centrado y escalado de los datos X): Cuando se activa esta casilla, los datos serán centrados con valor medio cero y escalados con desviación típica uno. Esta opción puede resultar útil para mejorar la precisión de los cálculos numéricos.

Check to display fits on figure (seleccionar el ajuste que se visualizará en el gráfico): Se utiliza para seleccionar el ajuste que se visualizará en la Ventana Gráfica. Esta selección incluye la interpolación con el método segmentario utilizado por la función `spline`, la interpolación de Hermite utilizada por la función `pchip` y los polinomios de varios grados utilizados por la función `polyfit`. Se pueden seleccionar y visualizar varios ajustes de manera simultánea.

Show equations (mostrar ecuaciones): Cuando esta opción se encuentra seleccionada, se mostrarán las ecuaciones de los polinomios utilizados en el ajuste. La ecuación se visualiza utilizando el número de cifras significativas configuradas en el menú adyacente **Significant digits**.

Plot residuals (representación de residuos): Esta opción permite representar los valores residuales de cada punto (los valores residuales se explican en la Sección 8.2.1). Seleccionando las opciones

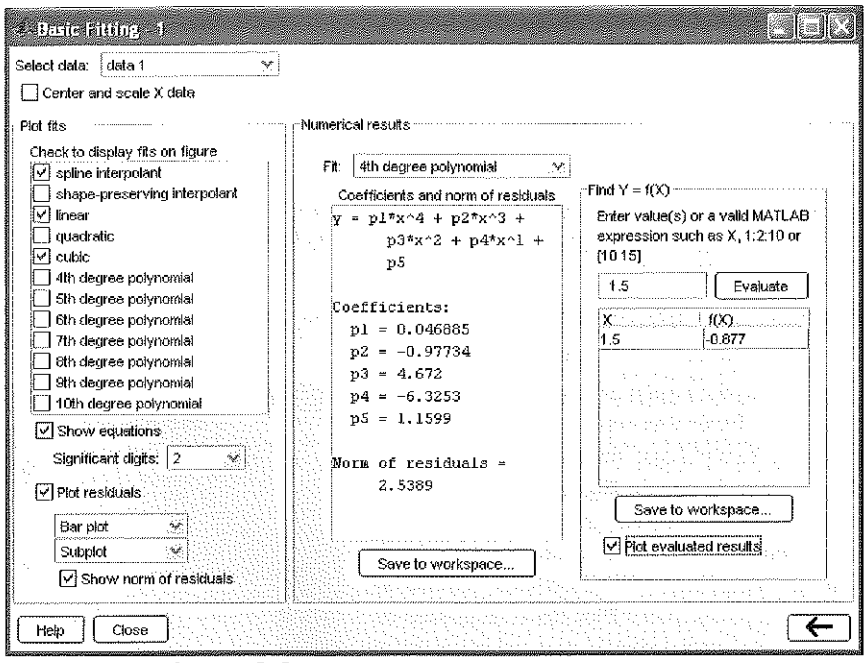


Figura 8.3: Ventana de la Interfaz Básica de Ajuste.

oportunas en los menús adyacentes, la representación de los residuos podrá tomar forma de gráfico de barras, gráfico de dispersión o un gráfico lineal (primer menú desplegable). Además, la representación se puede visualizar como un subgráfico de la misma ventana gráfica o como un gráfico aparte en una ventana gráfica distinta (segundo menú desplegable).

Show norm of residuals (mostrar la norma de los residuos): Esta opción sirve para mostrar la norma de los residuos en el mismo gráfico donde éstos se visualizan. La norma de los residuos es una medida de la calidad del ajuste. Cuanto menor sea la norma, mejor será el ajuste realizado.

Las tres opciones siguientes, que se encuentran en el panel **Numerical results** (resultados numéricos), permiten configurar la información numérica de un ajuste, independientemente del número de ajustes visualizados:

Fit (ajuste): Cuando se selecciona esta opción se obtienen los valores numéricos del ajuste. El ajuste se representará gráficamente sólo si se ha indicado en el panel **Plot fit**.

Coeficients and norm of residuals (coeficientes y norma de los residuos): Muestra los resultados numéricos del polinomio de ajuste seleccionado en el menú **Fit**. Esto incluye los coeficientes del polinomio y la norma de los residuos. Estos resultados se pueden almacenar en disco pulsando sobre el botón **Save to workspace** (guardar en el espacio de trabajo).

Find Y = f(x) (encontrar Y = f(x)): Esta opción proporciona una manera de interpolar (o extrapolar) valores numéricos para valores específicos de la variable independiente. Para ello sólo es necesario introducir el valor de la variable independiente en el cuadro de diálogo y pulsar sobre el botón **Evaluate** (calcular). Cuando la opción **Plot evaluated result** (representar el valor calculado) está activada, el punto calculado se visualizará también sobre el gráfico.

Como ejemplo, a continuación se utiliza la interfaz básica de ajuste para ajustar los datos del Problema de ejemplo 8.3. La ventana mostrada en la Figura 8.3 se corresponde con la interfaz básica de interpolación para este problema. La Figura 8.4 muestra un dibujo de los puntos, a curva de interpolación (tipo 'spline'), dos polinomios de ajuste (lineal y cúbico) con su correspondiente representación gráfica, así como el valor del punto $x = 1{,}5$ introducido en el cuadro de diálogo **Find Y = f(x)**. La ventana gráfica también incluye una representación gráfica de los residuos de los polinomios de ajuste y la norma para cada caso. l

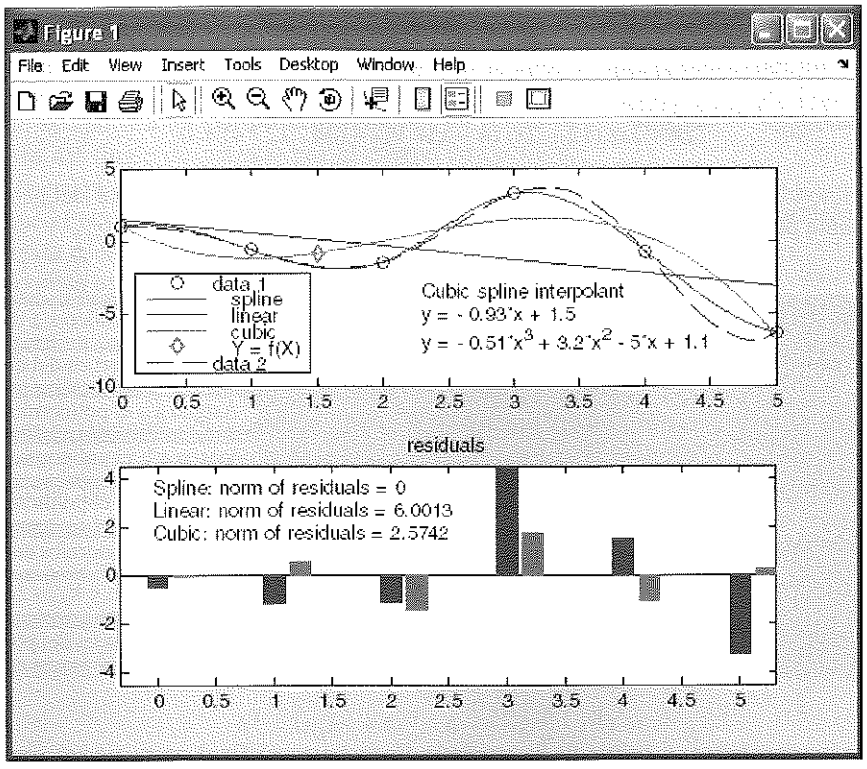
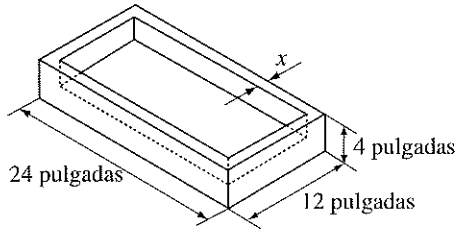


Figura 8.4: La ventana gráfica modificada mediante la interfaz básica de ajuste de MATLAB.

8.5 Ejemplo de aplicaciones MATLAB

Problema de ejemplo 8.4: Cálculo del grosor de una caja

Las dimensiones exteriores de una caja rectangular (el fondo y las cuatro caras, excluyendo la parte superior) fabricada de aluminio son 24 × 12 × 4 pulgadas. El grosor del fondo y de las caras de la caja es x. Deducir una expresión que relacione el peso de la caja con su grosor x. Calcular el grosor x para una caja que pesa 15 libras. El peso específico del aluminio es 0,101 libras/pulgadas^3.



Solución

El volumen del aluminio V_{Al} se puede calcular a partir del peso W de la caja, de la forma:

$$V_{Al} = \frac{W}{\gamma}$$

donde γ es el peso específico. Por tanto, el volumen del aluminio a partir de las dimensiones de la caja vendrá dado por:

$$V_{Al} = 24 \cdot 12 \cdot 4 - (24 - 2x)\,(12 - 2x)\,(4 - x)$$

donde el volumen interior de la caja se resta al volumen exterior. Esta ecuación puede escribirse de la siguiente forma:

$$(24 - 2x)\,(12 - 2x)\,(4 - x) + V_{Al} - (24 \cdot 12 \cdot 4) = 0$$

que es un polinomio de tercer grado. Una de las raíces del polinomio será el valor del grosor x de la caja. El siguiente programa script calcula este valor:

```
W = 15; gama = 0.101;                    % Asigna valores a W y gama.
VAlum = W/gama;                          % Calcula el volumen del aluminio.
a = [-2 24];                             % Asigna a a el polinomio 24 – 2x.
b = [-2 12];                             % Asigna a b el polinomio 12 – 2x.
c = [-1 4];                              % Asigna a c el polinomio 4 – x.
Vin = conv(c,conv(a,b));                 % Multiplica los tres polinomios anteriores.
polyeq = [0 0 0 (VAlum – 24*12*4)] + Vin % Suma Vin y V_Al – 24*12*4.
x = roots(polyeq)                        % Calcula las raíces del polinomio.
```

Como puede comprobarse, a partir de la segunda línea hasta el final, para poder sumar `Vin` y $V_{Al} - 24*12*4$, ésta última expresión se debe representar como un polinomio de grado idéntico al de `Vin`, ya que `Vin` es un polinomio de tercer grado. Cuando se guarda (Cap8ProgEj4) y se ejecuta el script, se visualiza el valor calculado x:

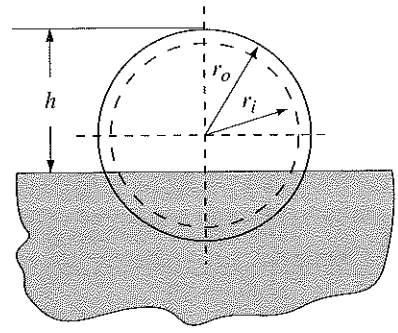
```
>> Cap8ProgEj4

polyeq =
   -4.0000   88.0000  -576.0000  148.5149

x =
   10.8656 + 4.4831i
   10.8656 - 4.4831i
    0.2687
```

El polinomio es: $-4x^3 + 88x^2 - 576x + 148{,}515$.

El polinomio tiene una raíz real, $x = 0{,}2687$ pulgadas, que se corresponde con el valor del grosor de las paredes de aluminio de la caja.

Problema de ejemplo 8.5: Cálculo de la altura de una boya

Una esfera de aluminio de pared delgada se utiliza como boya de señalización. La esfera tiene un radio de 60 cm, y el grosor de la pared de aluminio, cuya densidad es $\rho_{Al} = 2690$ kg/m^3, es de 12 mm. La boya está situada en el océano, donde la densidad del agua es de 1030 kg/m^3. Calcular la altura h entre la parte superior de la boya y la superficie del agua.



Solución

Según el principio de Arquímedes, la fuerza ascencional de empuje que se aplica a un cuerpo sumergido en un fluido es igual al peso del fluido desplazado por el cuerpo en cuestión. Entonces, la esfera de aluminio estará sumergida a una profundidad tal que el peso de la esfera sea igual al peso del fluido desplazado por la parte de la esfera que está sumergida en el agua.

El peso de la esfera vendrá dado por:

$$W_{\text{esfera}} = \rho_{Al} V_{Al} g = \rho_{Al} \frac{4}{3}\pi(r_o^3 - r_i^3)g$$

donde V_{Al} es el volumen del aluminio, r_0 y r_i representan el radio exterior e interior de la esfera, respectivamente, y g es la aceleración de la gravedad.

El peso del agua desplazado por la parte de la esfera que está sumergida vendrá dado por:

$$W_{\text{agua}} = \rho_{\text{agua}} V_{\text{agua}} g = \rho_{\text{agua}} \frac{1}{3}\pi(2r_o - h)g(r_o + h)g$$

Igualando ambos pesos resulta la siguiente ecuación:

$$h^3 - 3r_o h^2 + 4r_o^3 - 4\frac{\rho_{Al}}{\rho_{\text{agua}}}(r_o^3 - r_i^3) = 0$$

Esta ecuación es un polinomio de tercer grado con h como incógnita. La raíz de dicho polinomio será la solución al problema.

Con MATLAB se puede obtener la solución escribiendo el polinomio y utilizando la función `roots` para calcular el valor de h. A continuación se muestra el código correspondiente a esta solución:

```
rexterior = 0.60; rinterior = 0.588;          % Asigna valores a los radios.
rhoalum = 2690; rhoagua = 1030;               % Asigna las densidades a las variables.
a0=4*rexterior^3-4*rhoalum*(rexterior^3-rinterior^3)/rhoagua;   % Calcula el coeficiente a0.
p = [1 -3*rexterior 0 a0];                    % Asigna los coeficientes del polinomio.
h = roots(p)                                  % Calcula las raíces del polinomio.
```

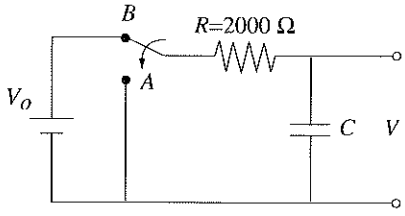
Una vez que se ejecuta el fichero en la Ventana de Comandos (guardado como Cap8ProEj5.m), MATLAB muestra tres posibles valores para la solución h, ya que el polinomio calculado es de tercer grado. Sin embargo, la única solución posible para este problema es $h = 0{,}9029$ m.

```
>> Cap8ProEj5

h =
   1.4542
   0.9029
  -0.5570
```

El polinomio posee tres raíces. La única solución posible, en este caso, es 0,9029 m.

Problema de ejemplo 8.6: Cálculo del tamaño de un condensador

Un condensador eléctrico posee una capacidad desconocida. Para calcular su capacidad se conecta a un circuito como el que se muestra en la figura adjunta. En este circuito, el conmutador se conecta primero a B, de forma que el condensador se carga. Seguidamente, el conmutador se conecta a A, de forma que el condensador se descarga a través de la resistencia. Cuando el condensador se está descargando, el valor del voltaje que circula a través de él se mide durante 10 segundos, en intervalos de un segundo. Los valores medidos se muestran en la siguiente tabla.



t(s)	1	2	3	4	5	6	7	8	9	10
V(V)	9,4	7,31	5,15	3,55	2,81	2,04	1,26	0,97	0,74	0,58

Representar gráficamente el voltaje en función del tiempo y calcular la capacidad del condensador ajustando una curva exponencial a los puntos anteriores.

Solución

Cuando el condensador se descarga a través de la resistencia, el voltaje del condensador en función del tiempo viene dado por:

$$V = V_0 e^{(-t)/(RC)}$$

donde V_0 es el voltaje inicial, R el valor de la resistencia y C la capacidad del condensador. Tal y como se vio en la Sección 8.2.2, una función exponencial se puede escribir en forma lineal. Si representamos la ecuación exponencial anterior de forma lineal para $\ln(V)$ y t, ésta quedará:

$$\ln(V) = \frac{-1}{RC}t + \ln(V_0)$$

Esta ecuación tiene la forma $y = mx + b$ y se puede ajustar a los puntos utilizando la función `polyfit(x,y,1)` con t como variable independiente x y $\ln(V)$ con variable independiente y. Los

coeficientes m y b, calculados mediante la función `polyfit`, se pueden utilizar para obtener C y V_0, de la forma:

$$C = \frac{-1}{Rm}t \quad \text{y} \quad V_0 = e^b$$

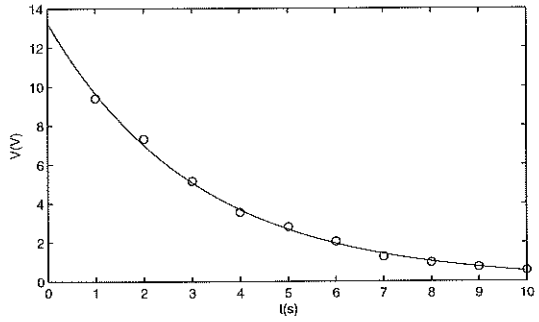
El siguiente programa, en forma de fichero script, calcula la mejor función de ajuste exponencial para los puntos dados, calculando C y V_0 y dibujando los puntos y la función de ajuste.

```
R = 2000;                                          % Define R.
t = 1:10;                                          % Asigna los puntos a los vectores t y V.
v = [9.4 7.31 5.15 3.55 2.81 2.04 1.26 0.97 0.74 0.58];
p = polyfit(t,log(v),1);                           % Utiliza la función polyfit con t y log(v).
C = -1/(R*p(1))                                    % Calcula C a partir de p(1), que contiene el valor de m en la ecuación lineal.
V0 = exp(p(2))                                     % Calcula V0 a partir de p(2), que contiene el valor de b en la ecuación lineal;
tplot=0:0.1:10;                                    % Crea un vector tplot de tiempo para dibujar la función.
vplot = V0*exp(-tplot./(R*C));                     % Crea el vector vplot para dibujar la función.
plot(t,v,'o',tplot,vplot)                          % Dibuja los puntos y la función de ajuste.
```

Cuando se ejecuta este fichero (guardado en disco como Cap8ProEj6.m), los valores de C y V_0 aparecen en la Ventana de Comandos, de la forma:

```
>> Cap8ProEj6
C =                                                % La capacidad del condensador es de 1600 μF.
   0.0016

V0 =
   13.2796
```

El programa crea además el siguiente gráfico (las etiquetas de los ejes se añadieron posteriormente con el editor gráfico):



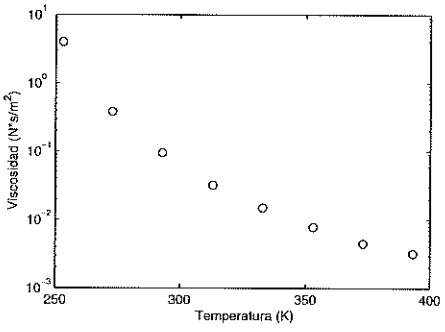
Problema de ejemplo 8.7: Dependencia de la viscosidad con la temperatura

La viscosidad, μ, es una propiedad de los fluidos que caracteriza su resistencia a fluir. La viscosidad de la mayoría de los fluidos es muy sensible a la temperatura. A continuación se muestra una tabla donde se representa la viscosidad de un aceite tipo SAE 10W a diferentes temperaturas (datos tomados del libro B.R. Munson, D.F. Young y T.H. Okiishi, "Fundamental of Fluid Mechanics", 4ª Edición, John Wiley and Sons, 2002). Calcular una ecuación para ajustar los datos de la tabla.

T (°C)	–20	0	20	40	60	80	100	120
μ (N s/m^2) ($\times 10^{-5}$)	4	0,38	0,095	0,032	0,015	0,0078	0,0045	0,0032

Solución

Para ver qué tipo de ecuación proporciona el mejor ajuste de los datos anteriores, se representará gráficamente μ en función de T (temperatura absoluta), utilizando una escala lineal para T y una escala logarítmica para μ. El gráfico que se muestra a la derecha nos indica que los puntos no parecen estar dispuestos a lo largo de una línea recta. Esto significa que una función exponencial simple de la forma $y = b^{emx}$, que modela una línea recta en estos ejes, no proporcionará un buen ajuste en este caso. Como los puntos de la figura parecen ajustarse mejor a una línea curva, una función que posiblemente dé mejores resultados para el ajuste será:



$$\ln(\mu) = a_2T^2 + a_1T + a_0$$

Esta función se puede ajustar a los datos utilizando la función `polyfit(x,y,2)` (polinomio de segundo grado), donde la variable independiente es T y la variable dependiente es ln(μ). La ecuación anterior se puede resolver en μ, para dar la viscosidad en función de la temperatura:

$$\mu = e^{(a_2T^2 + a_1T + a_0)} = e^{a_0}e^{a_1T}e^{a_2T^2}$$

El programa siguiente calcula la mejor función de ajuste posible, y crea un gráfico para visualizar los puntos y la función calculada.

```
T = [-20:20:120];
mu = [4 0.38 0.095 0.032 0.015 0.0078 0.0045 0.0032];
TK = T + 273;
p = polyfit(TK, log(mu),2)
Tplot = 273 + [-20:120];
```

```
muplot = exp(p(1)*Tplot.^2+p(2)*Tplot+p(3));
semilogy(TK,mu,'o',Tplot,muplot)
```

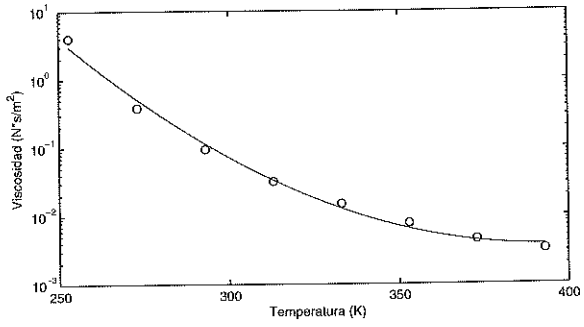
Cuando este programa se ejecuta (después de haber sido almacenado en disco con el nombre Cap8ProEj7.m), se calculan los coeficientes mediante la función `polyfit`, y éstos se visualizan en la Ventana de Comandos (ver más abajo) como elementos del vector `p`.

```
>> Cap8ProEj7
p =
   0.0003   -0.2685   47.1673
```

Con estos coeficientes se puede calcular la viscosidad del aceite en función de la temperatura, de la forma:

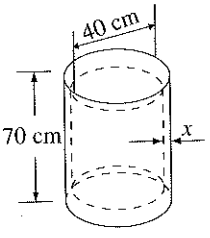
$$\mu = e^{(0{,}0003T^2 - 0{,}2685T + 47{,}1673)} = e^{47{,}1673} e^{(-0{,}2685)T} e^{0{,}0003T^2}$$

El gráfico generado muestra que la ecuación calculada se ajusta bien a los puntos dados en el enunciado (las etiquetas de los ejes se añadieron posteriormente con el editor gráfico).

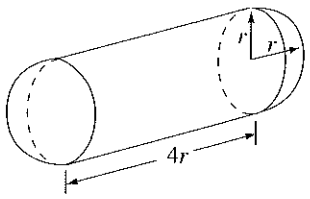


8.6 Problemas

1. Dibuje el polinomio $y = 1{,}5x^4 - 5x^2 + x + 2$ para $-2 \le x \le 2$. Primero cree un vector para x, luego utilice la función `polyval` para calcular y. Finalmente utilice la función `plot` para generar el gráfico.
2. Divida el polinomio $15x^5 + 35x^4 - 37x^3 - 19x^2 + 41x - 15$ entre el polinomio $5x^3 - 4x + 3$.
3. Divida el polinomio $4x^4 + 6x^3 - 2x^2 - 5x + 3$ entre el polinomio $x^2 + 4x + 2$.
4. Un depósito cilíndrico de aceite, fabricado con acero inoxidable, tiene un diámetro exterior de 40 cm y una longitud de 70 cm. Calcule el grosor x del depósito cuya masa es de 18 kg. La densidad del acero inoxidable es de 7920 kg/m^3.



5. Sea un depósito de gas en forma de cilindro circular con un hemisferio en cada uno de sus extremos, como el que se muestra en la figura adjunta. El radio del cilindro es r y su longitud es $4r$. Determine r si el volumen del depósito es de 30 m^3.



6. Escriba una función MATLAB que sume o reste dos polinomios de cualquier grado. Utilice la siguiente línea de definición para la función: `p = PoliOpera(p1,p2,operacion)`. Los primeros dos argumentos de entrada, `p1` y `p2`, serán los vectores con los coeficientes de los dos polinomios (si los polinomios no son del mismo grado, la función deberá añadir necesariamente los ceros que le faltan al vector más corto). El tercer argumento será una cadena que puede tomar los valores `'sumar'` o `'restar'`, para sumar o restar los polinomios introducidos como parámetro.

   Utilice la función anterior para sumar y restar los siguientes polinomios:

$$f_1(x) = x^5 - 7x^4 + 11x^3 - 4x^2 - 5x - 2 \quad \text{y} \quad f_2(x) = 9x^2 - 10x + 6$$

7. Escriba una función MATLAB que calcule el máximo (o el mínimo) de una ecuación cuadrática del tipo:

$$f(x) = ax^2 + bx + c$$

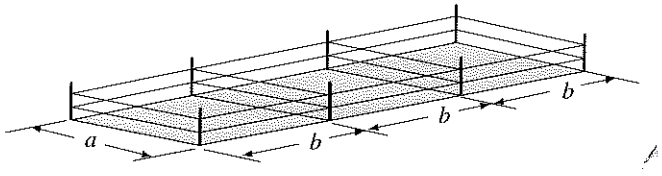
   Defina la función de la forma `[x,y,w] = maxomin(a,b,c)`. Los argumentos de entrada serán los coeficientes `a`, `b` y `c`. Los argumentos de salida serán el valor `x` del máximo (o del mínimo), el valor y del máximo (o del mínimo) y w, que valdrá 1 si y es máximo o 2 si y es mínimo.

   Utilice la función anterior para calcular el máximo o el mínimo de las siguientes funciones:

   a) $f(x) = 6x^2 - 18x + 6$

   b) $f(x) = -4x^2 - 20x + 5$

8. Un granjero desea construir tres corrales rectangulares idénticos, tal y como se muestra en la figura adjunta. Dispone de 60 metros de valla para construirlos. Calcule las dimensiones a y b que maximizan el área total que se encierra entre las vallas. Para resolver el problema se debe encontrar el máximo de una función cuadrática. Utilice la función del problema 7 para encontrar el máximo.



9. Dados los siguientes puntos:

x	–5	–4	–2,2	–1	0	1	2,2	4	5	6	7
y	0,1	0,2	0,8	2,6	3,9	5,4	3,6	2,2	3,3	6,7	8,9

a) Ajuste los datos mediante un polinomio de primer grado. Represente gráficamente tanto los puntos como el polinomio.

b) Ajuste los datos mediante un polinomio de tercer grado. Represente gráficamente tanto los puntos como el polinomio.

c) Ajuste los datos mediante un polinomio de cuarto grado. Represente gráficamente tanto los puntos como el polinomio.

d) Ajuste los datos mediante un polinomio de décimo grado. Represente gráficamente tanto los puntos como el polinomio.

10. La tabla siguiente representa la población de China entre los años 1940 y 2000:

Año	1940	1950	1960	1970	1980	1990	2000
Población (millones)	537	557	682	826	981	1135	1262

a) Calcule la función exponencial que mejor se ajusta a los datos anteriores. Utilice esta función para estimar la población de China en el año 1955.

b) Utilice una curva, mediante una función cuadrática (polinomio de segundo grado), para ajustar los datos anteriores. Utilice esta función para estimar la población de China en el año 1955.

c) Utilice los métodos 'spline' y 'linear' para interpolar los datos anteriores. Estime la población de China en el año 1955 utilizando los métodos de interpolación anteriores.

Para cada uno de estos apartados, represente gráficamente los puntos (con círculos) y las correspondientes curvas de ajuste o interpolación. Observe que en el apartado *c* se piden dos curvas de interpolación distintas. El valor real de la población de China en el año 1955 fue de 614,4 millones.

11. La densidad estándar del aire, D (resultado de calcular la media de distintas medidas), a diferentes alturas, h, desde el nivel del mar hasta los 33 km, viene dada en la tabla que se muestra a continuación:

h (km)	0	3	6	9	12	15
D (kg/m^3)	1,2	0,91	0,66	0,47	0,31	0,19
h (km)	18	21	24	27	30	33
D (kg/m^3)	0,12	0,075	0,046	0,029	0,018	0,011

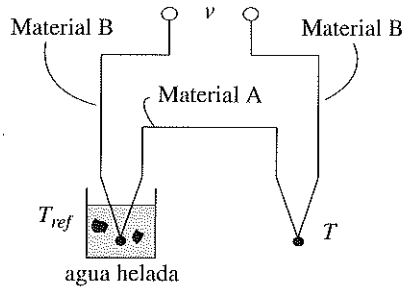
a) Haga las siguientes cuatro representaciones gráficas de los puntos, representando siempre la densidad en función de la altura: 1) ambos ejes en escala lineal, 2) h en escala logarítmica y D en escala lineal, 3) h con escala lineal y D con escala logarítmica, 4) ambos ejes con escala logarítmica. Basándose en estos gráficos, elija la función (lineal, potencia, exponencial o logarítmica) que mejor se ajuste a los puntos y calcule, además, los coeficientes de dicha función.

b) Represente la función y los puntos utilizando escalas lineales.

12. Escriba una función MATLAB que ajuste datos mediante una función potencia, de la forma: $y = bx^m$. Utilice la siguiente línea para la definición de la función: `[b,m] = powerfit(x,y)`, donde los argumentos de entrada x e y serán los vectores con los coeficientes de los puntos, y los argumentos de salida b y m serán las constantes del ajuste para la ecuación potencia calculada. Utilice esta función para ajustar los datos que se muestran a continuación y represene gráficamente tanto la función como los puntos:

x	0,5	1,9	3,3	4,7	6,1	7,5
y	0,8	10,1	25,7	59,2	105	122

13. Un termopar es un sensor utilizado para medir la temperatura. Se construye uniendo dos alambres de materiales distintos. Para medir la temperatura, los termopares se conectan a un circuito como el que se muestra en la figura adjunta. Uno de los termopares se introduce en un medio con una temperatura constante conocida T_{ref} (por ejemplo, agua helada), y el otro se sitúa donde se desee medir la temperatura T. Cuando las temperaturas medidas difieren se genera un voltaje v. Este voltaje v se puede expresar en función de la temperatura mediante una expresión del tipo:



$$v = K_s(T - T_{ref})$$

donde K_s es una constante que depende de los materiales utilizados para construir el termopar.

Los siguientes datos son los resultados de un experimento realizado para calcular la constante K_s de un termopar. Utilice estos datos para calcular K_s mediante una curva de ajuste.

T (°C)	25	100	200	300	400	500	600	700
v (mV)	1,11	4,03	8,16	12,62	16,54	20,90	23,7	29,15

14. El límite elástico de los metales depende en gran parte del tamaño del grano. Para estos metales, la relación entre el límite elástico y el diámetro medio del grano d viene dada por la ecuación de Hall-Petch:

$$\sigma_y = \sigma_0 + kd^{\left(\frac{-1}{2}\right)}$$

Los siguientes datos son los resultados de medidas del diámetro medio del grano y el límite elástico:

d (mm)	0,005	0,009	0,016	0,025	0,040	0,062	0,085	0,110
σ_y (MPa)	205	150	135	97	89	80	70	67

a) Utilice una curva de ajuste para calcular la constante σ_y y k en la ecuación de Hall-Petch para este material. Utilice las constantes para calcular, mediante la ecuación, el límite elástico de un material cuyo tamaño de grano es 0,05 mm. Represente en un gráfico los puntos (utilizando marcadores en forma de círculo) y la ecuación de Hall-Petch (utilizando una línea sólida).

b) Utilice interpolación lineal para calcular el límite elástico de un material con un tamaño de grano de 0,05 mm. Haga un gráfico donde se muestren los datos con círculos y la interpolación lineal mediante una línea sólida.

c) Utilice interpolación cúbica para calcular el límite elástico de un material con un tamaño de grano de 0,05 mm. Haga un gráfico donde se muestren los datos con círculos y la interpolación lineal mediante una línea sólida.

15. La ecuación de un gas ideal relaciona volumen, presión, temperatura y la cantidad de gas mediante la expresión:

$$V = \frac{nRT}{P}$$

donde V es el volumen en litros, P es la presión en atm, T es la temperatura en grados K, n es el número de moles y R es la constante de los gases.

Se realiza un experimento para calcular el valor de la constante R en el cual se comprimen 0,05 moles de gas a diferentes volúmenes, aplicando una presión dada al gas. Se registra, para cada volumen, la presión y la temperatura del gas. Utilizando los datos que se muestran a continuación, calcule R y represente gráficamente V frente a T/P, y ajuste los datos a los puntos mediante una ecuación lineal:

V (L)	0,75	0,65	0,55	0,45	0,35
T (°C)	25	37	45	56	65
P (atm)	1,63	1,96	2,37	3,00	3,96

16. La viscosidad es una propiedad de los fluidos que caracteriza su resistencia a fluir. La viscosidad de la mayoría de los fluidos es muy sensible a la temperatura. Para los gases, la variación de la viscosidad con la temperatura se representa comúnmente por la ecuación de Suzerainty:

$$\mu = \frac{CT^{3/2}}{T+S}$$

donde μ es la viscosidad, T la temperatura absoluta, y C y S constantes empíricas. En la tabla siguiente se muestran valores de la viscosidad del aire a distintas temperaturas (datos tomados del libro B.R. Munson, D.F. Young y T.H. Okiishi, "Fundamental of Fluid Mechanics", 4ª Edición, John Wiley and Sons, 2002):

T (°C)	–20	0	40	100	200	300	400	500	1000
μ (N s/m^2) ($\times 10^{-5}$)	1,63	1,71	1,87	2,17	2,53	2,98	3,32	3,64	5,04

Calcule el valor de las constantes C y S mediante una curva de ajuste. Represente gráficamente la viscosidad frente a la temperatura en grados centígrados. Utilice círculos para representar los datos y una línea sólida para la curva de ajuste a partir de la ecuación de Suzerainty.

La curva de ajuste se puede obtener escribiendo de nuevo la ecuación de Suzerainty, de la forma:

$$\frac{T^{3/2}}{\mu} = \frac{1}{C}T + \frac{S}{C}$$

y utilizando un polinomio de primer grado.

Capítulo 9
Gráficos tridimensionales

Los gráficos tridimensionales (3-D) proporcionan una manera muy práctica de representar datos de más de dos variables. MATLAB proporciona distintas opciones y funciones destinadas a la representación de gráficos tridimensionales. Estos gráficos, al igual que las representaciones bidimensionales vistas hasta hora, permiten ser representados con suficiente expresividad, permitiendo elegir la apariencia así como distintos efectos especiales en el momento de su representación. En este capítulo se tratan muchas de las características de los gráficos tridimensionales. No obstante, se puede obtener información adicional sobre este tema en la ayuda de MATLAB, en "**Plotting and Data Visualization**" (representación y visualización de datos).

Este capítulo es, de alguna manera, una continuación del Capítulo 5, donde se tratan con detalle los gráficos bidimensionales. A diferencia de éstos, los gráficos tridimensionales no se utilizan tan frecuentemente, por eso se presentan en este libro en un capítulo a parte. Por otro lado, es más que recomendable empezar por el capítulo de gráficos bidimensionales antes de abordar los gráficos 3-D, para así tener una base mucho más sólida sobre la representación gráfica de datos en MATLAB. Se supone, por tanto, que el lector que aborda este capítulo está ya familiarizado con los gráficos 2-D.

9.1 Gráficos de línea

Un gráfico 3-D de línea está constituido por una línea que se obtiene uniendo una serie de puntos en un espacio tridimensional. La forma más sencilla y básica de crear un gráfico 3-D es mediante la función `plot3` de MATLAB, cuya sintaxis es bastante similar a la de la función `plot` ya vista anteriormente:

```
plot3(x,y,z,'Especificadores de linea', 'Propiedades', 'Valores')
```

`x`, `y` y `z` representan los vectores con las coordenadas de los puntos.

(Opcional) Especificadores que definen el tipo y el color de la línea y los marcadores.

(Opcional) Propiedades con valores que pueden ser utilizadas para especificar el grosor de la línea, el tamaño y los colores del borde y el relleno de los marcadores.

- Los tres vectores con las coordenadas de los puntos deben tener el mismo número de elementos.
- Los especificadores de línea, las propiedades y los valores son los mismos que los vistos en el capítulo sobre gráficos bidimensionales (ver Sección 5.1).

Por ejemplo, si las coordenadas x, y y z vienen dadas en función de la variable t mediante las ecuaciones:

$$x = \sqrt{t}\ \text{sen}\ (2t)$$
$$y = \sqrt{t}\ \cos\ (2t)$$
$$z = 0{,}5t$$

se puede realizar una representación gráfica de los puntos, dentro del intervalo $0 \leq t \leq 6\pi$, mediante el siguiente fichero script:

```
t = 0:0.1:6*pi;
x = sqrt(t).*sin(2*t);
y = sqrt(t).*cos(2*t) ;
z = 0.5*t ;
plot3(x,y,z,'k','linewidth',1)
grid on
xlabel('x'); ylabel('y'); zlabel('z')
```

Una vez ejecutado este fichero, el gráfico que se visualiza es el que se muestra en la Figura 9.1.

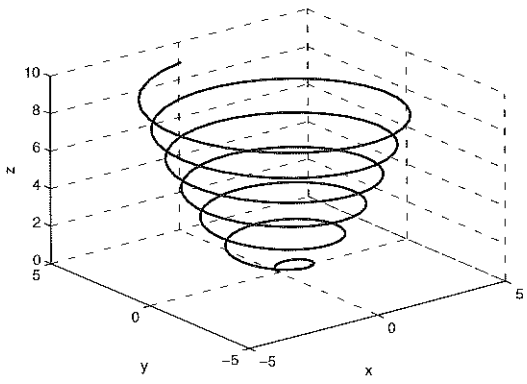


Figura 9.1: Representación gráfica de la función $x = \sqrt{t}\ \text{sen}\ (2t)$, $y = \sqrt{t}\ \cos\ (2t)$, $z = 0{,}5t$ en el intervalo $0 \leq t \leq 6\pi$.

9.2 Gráficos de malla y de superficie

Los gráficos de malla y superficie son gráficos tridimensionales utilizados para representar funciones que tienen la forma $z = f(x, y)$, donde x e y son variables independientes, y z es la variable dependiente. Esto implica que, en un dominio dado, el valor de z se puede calcular en función de cualquier combinación de x e y. Los gráficos de malla y de superficie se generan en tres pasos. El primer paso es crear una malla o rejilla en el plano x-y que cubra el dominio de la función. El segundo paso es calcular el valor de z en cada punto de la rejilla. El tercer paso es representar el gráfico. A continuación se explican estos tres pasos más detenidamente.

Creación de una rejilla en el plano x-y:

Una rejilla, en este contexto concreto, se puede definir como un conjunto de puntos correspondientes al plano x-y del dominio de la función. La densidad de la rejilla (número de puntos utilizados para definir el dominio) debe ser definida por el usuario. La Figura 9.2 muestra un ejemplo de rejilla correspondiente al dominio $-1 \le x \le 3$ y $1 \le y \le 4$.

En esta rejilla la distancia entre los puntos es de 1. Los puntos de la rejilla se pueden definir mediante dos matrices X e Y que contienen las coordenadas de todos los puntos x e y, respectivamente:

$$X = \begin{bmatrix} -1 & 0 & 1 & 2 & 3 \\ -1 & 0 & 1 & 2 & 3 \\ -1 & 0 & 1 & 2 & 3 \\ -1 & 0 & 1 & 2 & 3 \end{bmatrix} \quad \text{y} \quad Y = \begin{bmatrix} 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

La matriz X se construye mediante filas idénticas, ya que en cada fila los puntos tienen la misma coordenada x. De manera similar, la matriz Y se construye con columnas idénticas, ya que en cada columna los puntos tienen la misma coordenada y.

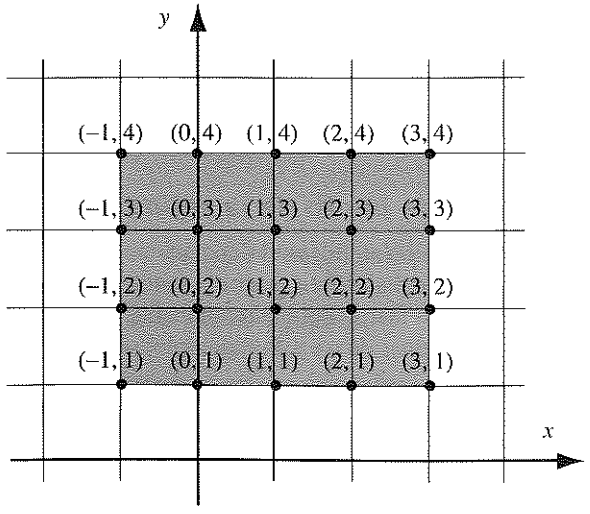
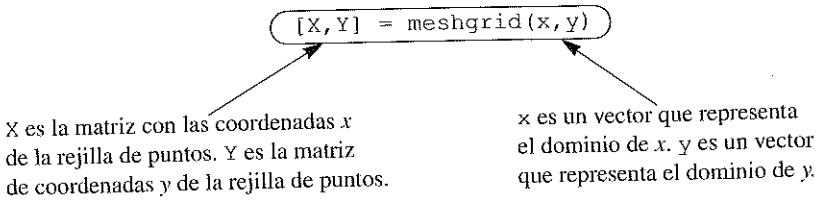


Figura 9.2: Rejilla en el plano x-y para el dominio $-1 \le x \le 3$ y $1 \le y \le 4$, con un espaciado de 1.

MATLAB posee una función denominada `meshgrid` que se puede utilizar para crear automáticamente las matrices *X* e *Y*. La sintaxis de esta función es la siguiente:



El primero y el último de los elementos de los vectores `x` e `y` representan los límites del dominio. La densidad de la rejilla viene determinada por el número de elementos en los vectores. Por ejemplo, las matrices de malla *X* e *Y* que se corresponden con la rejilla de la Figura 9.2 son las que se muestran a continuación, utilizando el comando `meshgrid`:

```
>> x = -1:3;
>> y = 1:4;
>> [X,Y] = meshgrid(x,y)
X =
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3

Y =
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
     4     4     4     4     4
```

Una vez que las matrices de la rejilla han sido creadas, éstas se pueden utilizar para calcular el valor de *z* en cada punto de la rejilla.

Cálculo del valor de z en cada punto de la rejilla:

El valor de *z* en cada punto se puede calcular utilizando operaciones elemento a elemento. Cuando las variables independientes *x* e *y* son matrices (deben tener el mismo número de elementos), la variable dependiente calculada será también una matriz del mismo tamaño. El valor de *z* en cada punto se calcula a partir de los valores correspondientes de *x* e *y*. Por ejemplo, si *z* viene dada a partir de la función:

$$z = \frac{xy^2}{x^2 + y^2}$$

el valor de cada punto *z* en la rejilla se puede calcular de la siguiente forma:

```
>> Z = X.*Y.^2./(X.^2+Y.^2)

Z =
   -0.5000   0   0.5000   0.4000   0.3000
   -0.8000   0   0.8000   1.0000   0.9231
   -0.9000   0   0.9000   1.3846   1.5000
   -0.9412   0   0.9412   1.6000   1.9200
```

Una vez creadas las tres matrices, éstas se pueden utilizar para generar gráficos de malla o de superficie.

Creación de gráficos de malla y de superficie:

La creación de gráficos de malla (o de mallado) se lleva a cabo mediante el comando `mesh`, mientras que la creación de gráficos de superficie se realiza mediante el comando `surface`. La sintaxis de estos comandos es la siguiente:

`mesh(X,Y,Z)` `surf(X,Y,Z)`

donde `X` e `Y` son las matrices con las coordenadas de la rejilla, y `Z` la matriz con los valores de z sobre la rejilla de puntos. Un gráfico de malla se compone de líneas que unen los puntos. Lo mismo ocurre con los gráficos de superficie, aunque en este caso las áreas resultantes entre los huecos de la malla se rellenan con colores.

El siguiente fichero script contiene un ejemplo completo del programa que crea un rejilla y genera un gráfico de malla (o de superficie) de la función $z = \dfrac{xy^2}{x^2+y^2}$ en el dominio $-1 \le x \le 3$ y $1 \le y \le 4$:

```
x = -1:0.1:3;
y = 1:0.1:4;
[X,Y] = meshgrid(x,y)
Z = X.*Y.^2./(X.^2+Y.^2)
mesh(X,Y,Z)
xlabel('x'); ylabel('y'); zlabel('z')
```

Se puede sustituir por `surf(X,Y,Z)` para hacer un gráfico de superficie.

Observe que en este programa los vectores `x` e `y` tienen un espaciado mucho menor que el del ejemplo mostrado anteriormente. Un espaciado menor crea una rejilla más densa. Los gráficos generados por este último programa se muestran en la página siguiente.

Comentarios adicionales sobre el comando `mesh`:

- Los gráficos creados tienen colores que pueden variar en función de la magnitud z. La variación del color se añade como característica a la visualización de gráficos 3-D. Se pueden cambiar las propiedades del color para que éste sea constante, utilizando el editor gráfico de la Ventana de Gráficos (seleccionar la flecha de edición y pulsar en la figura para abrir Property Editor Window

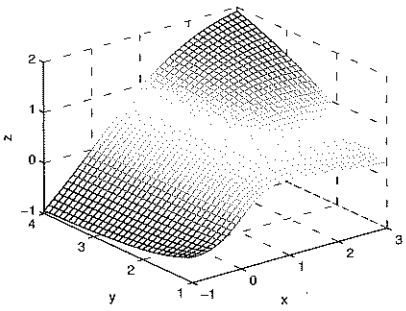


Gráfico de malla

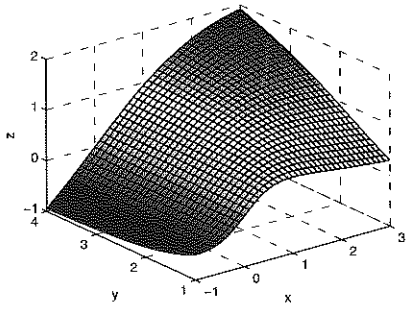


Gráfico de superficie

(la Ventana de Propiedades del Editor). El color se puede cambiar en Mesh Properties (la lista de propiedades de los gráficos de malla). También se puede utilizar, para este mismo cometido, el comando `colormap(C)`, donde `C` es un vector de tres elementos. Cada elemento del vector representa la intensidad de los colores rojo, verde y azul (RGB), respectivamente. Estos elementos pueden tomar valores entre 0 (mínima intensidad) y 1(máxima intensidad). Algunas tonalidades típicas más comúnmente utilizadas son las siguientes:

C = [0 0 0] Negro C = [1 0 0] Rojo C = [0 1 0] Verde

C = [0 0 1] Azul C = [1 1 0] Amarillo C = [1 0 1] Magenta

C = [0,5 0,5 0,5] Gris

- Cuando se ejecuta el comando `mesh`, la rejilla se activa por defecto. Para ocultar la rejilla se puede teclear el comando `grid off`. Para volver a activar la rejilla basta con teclear `grid on`.
- Para dibujar una caja que rodee al gráfico se puede utilizar el comando `box on`.
- Los comandos `mesh` y `surf` se pueden utilizar en la forma `mesh(Z)` y `surf(Z)`. En estos casos, los valores de Z se representan en función de los índices de la propia matriz Z. El número de fila está sobre el eje *x*, y el número de columna sobre el eje *y*.

Existen además otros comandos utilizados para representar gráficos tridimensionales, similares a los vistos hasta ahora. La Tabla 9.1 muestra un resumen de estos comandos. Todos los ejemplos de gráficos contenidos en esta tabla han sido generados a partir de la función $z = 1{,}8^{-1{,}5\sqrt{x^2+y^2}} \operatorname{sen}(x) \cos(0{,}5y)$, para el dominio $-3 \leq x \leq 3$ y $-3 \leq y \leq 3$.

Tabla 9.1: Gráficos de malla y de superficie.

Tipo de gráfico	Ejemplo de representación	Programa

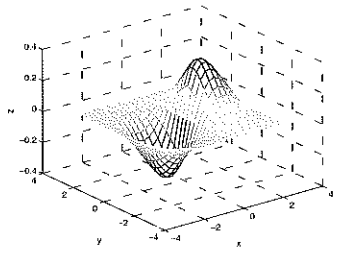
Gráfico de malla
 	
Formato de la función:
`mesh(X, Y, Z)`
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		mesh(X,Y,Z)
		xlabel('x'); ylabel('y'); zlabel('z')

Tabla 9.1: Gráficos de malla y de superficie. (*Continuación*)

Tipo de gráfico	Ejemplo de representación	Programa

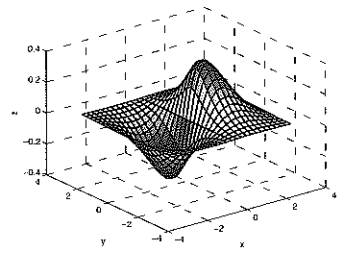
Gráfico de superficie
 	
Formato de la función:
`surf(X, Y, Z)`
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		surf(X,Y,Z)
		xlabel('x'); ylabel('y'); zlabel('z')

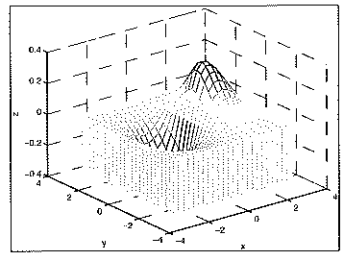
Gráfico de malla con cortina (dibuja una cortina alrededor de la malla)
 	
Formato de la función:
`meshz(X, Y, Z)`
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		meshz(X,Y,Z)
		xlabel('x'); ylabel('y'); zlabel('z')

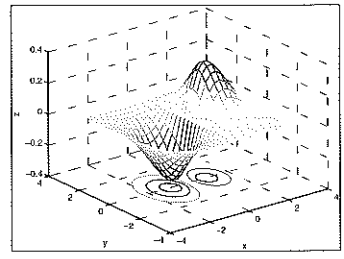
Gráfico de malla con contorno (dibuja un contorno debajo de la malla)
 	
Formato de la función:
`meshc(X, Y, Z)`
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		meshc(X,Y,Z)
		xlabel('x'); ylabel('y'); zlabel('z')

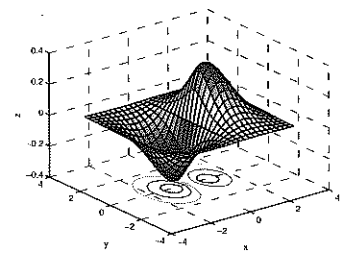
Gráfico de superficie con contorno (dibuja un contorno debajo de la superficie)
 	
Formato de la función:
`surfc(X, Y, Z)`
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		surfc(X,Y,Z)
		xlabel('x'); ylabel('y'); zlabel('z')

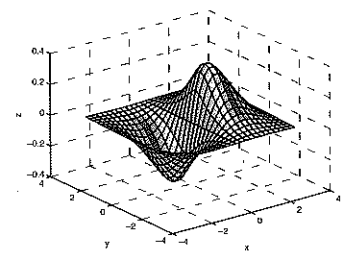
Gráfico de superficie con alumbrado
 	
Formato de la función:
`surfl(X, Y, Z)`
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		surfl(X,Y,Z)
		xlabel('x'); ylabel('y'); zlabel('z')

Tabla 9.1: Gráficos de malla y de superficie. (*Continuación*)

Tipo de gráfico	Ejemplo de representación	Programa

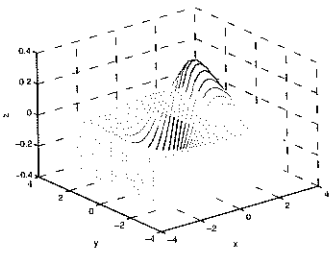
Gráfico de cascada (dibuja una malla unidireccional)
 	
Formato de la función:
`waterfall(X,Y,Z)`
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		waterfall(X,Y,Z)
		xlabel('x'); ylabel('y'); zlabel('z')

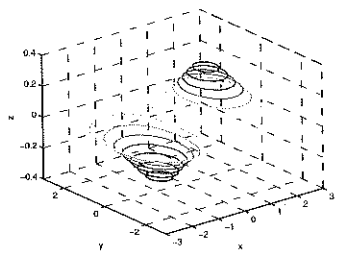
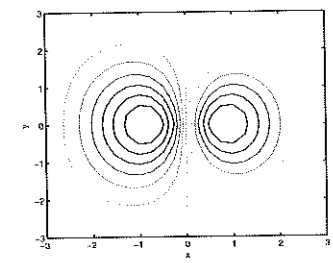
Gráfico de contorno 3-D
 	
Formato de la función:
`contour3(X, Y, Z, n)`
 	
n es el número de niveles de contorno (opcional)
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		contour3(X,Y,Z,15)
		xlabel('x'); ylabel('y'); zlabel('z')

Gráfico de contorno 2-D (dibuja proyecciones de niveles de contorno sobre el plano *x*-*y*)
 	
Formato de la función:
`contour(X, Y, Z,n)`
 	
n es el número de niveles de contorno (opcional)
		x = -3:0.25:3;
		y = -3:0.25:3;
		[X, Y] = meshgrid(x,y);
		Z = 1.8.^(-1.5*sqrt(X.^2 + Y.^2)).*cos(0.5*Y).*sin(X);
		contour(X,Y,Z,15)
		xlabel('x'); ylabel('y'); zlabel('z')

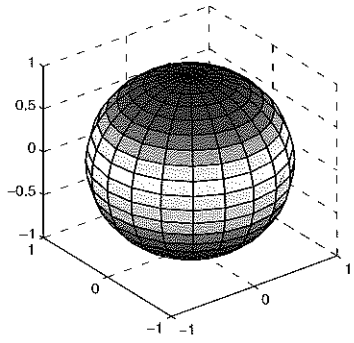
9.3 Gráficos Especiales

MATLAB posee otras funciones para crear varios tipos especiales de gráficos tridimensionales. La lista completa de estas funciones se puede encontrar en la ayuda de MATLAB, buscando en "Plotting and Data Visualization" (representación y visualización de datos). La Tabla 9.2 describe algunas de estas funciones.

Los ejemplos mostrados en la tabla siguiente no muestran en su totalidad los distintos parámetros y opciones que estos comandos especiales permiten. Para más detalles sobre cada uno de estos comandos sólo es necesario teclear `help nombre_comando` en la Ventana de Comandos de MATLAB.

Tabla 9.2: Gráficos 3-D especiales.

Tipo de gráfico	Ejemplo de representación	Programa

Esfera
 	
Formato de la función:
`sphere`
 	
Devuelve las matrices X, Y y Z de una esfera unitaria de 20 secciones.
`sphere(n)`
 	
En este caso se especifica el número `n` de secciones que se desean.
		sphere
		o:
		[X,Y,Z] = sphere(20);
		surf(X,Y,Z)

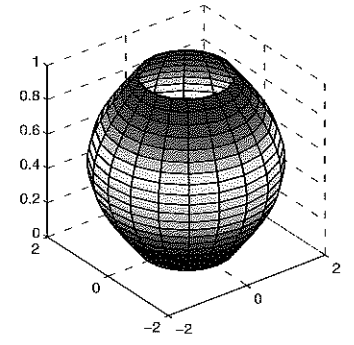
Cilindro
 	
Formato de la función:
`[X,Y,Z] = cylinder(r)`
 	
Devuelve las matrices X, Y y Z de un cilindro de perfil *r*.
		t = linspace(0,pi,20);
		r = 1 + sin(t);
		[X,Y,Z] = cylinder(r);
		surf(X,Y,Z)
		axis square

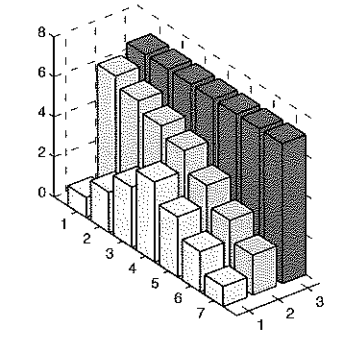
Gráfico de barras 3-D Formato de la función: `bar3(Y)` Cada elemento de `Y` es una barra. Las columnas se agrupan para la representación gráfica.		Y = [1 6.5 7; 2 6 7; 3 5.5 7; 4 5 7; 3 4 7; 2 3 7; 1 2 7]; bar3(Y)

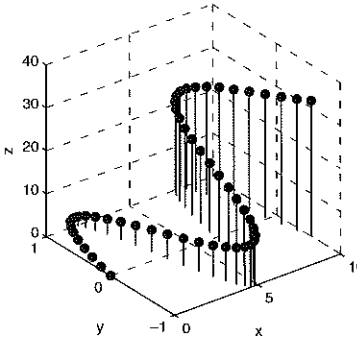
Gráfico de tallo o líneas verticales 3-D (dibuja una secuencia de puntos con marcadores y líneas verticales a partir del plano *x*-*y*)
 	
Formato de la función:
`stem3(X,Y,Z)`
		t = 0:0.2:10;
		x = t;
		y = sin(t);
		z = t.^1.5;
		stem3(x,y,z,'fill')
		grid on
		xlabel('x'); ylabel('y'); zlabel('z')

Tabla 9.2: Gráficos 3-D especiales. (*Continuación*)

Tipo de gráfico	Ejemplo de representación	Programa

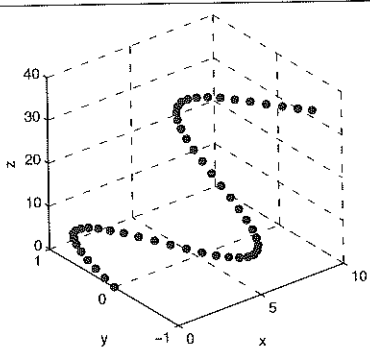
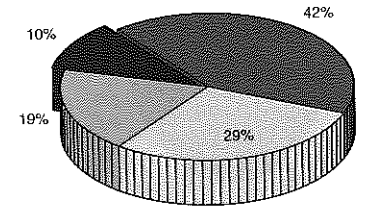
Gráfico de dispersión 3-D
 	
Formato de la función:
`scatter3(X,Y,Z)`
		t = 0:0.2:10;
		x = t;
		y = sin(t);
		z = t.^1.5;
		scatter3(x,y,z,'filled')
		grid on
		colormap([0.1 0.1 0.1])
		xlabel('x'); ylabel('y'); zlabel('z')

Gráfico de tarta 3-D Formato de la función: `pie3(X,explode)`		X = [5 9 14 20]; explode = [0 0 1 0]; pie3(X,explode) `explode` es un vector, de la misma longitud que X, compuesto por unos y ceros. Un uno indica la porción de la tarta que estará separada del resto las secciones.

9.4 *El comando* `view`

El comando `view` controla la dirección desde la que se verá el gráfico generado. Para ello se especifica la dirección en términos de los ángulos de azimut y elevación, tal y como se muestra en la Figura 9.3, o definiendo un punto en el espacio desde el cual se verá el gráfico. La sintaxis del comando `view` para establecer el ángulo de visión del gráfico es la siguiente:

```
view(az,el)  o  view([az,el])
```

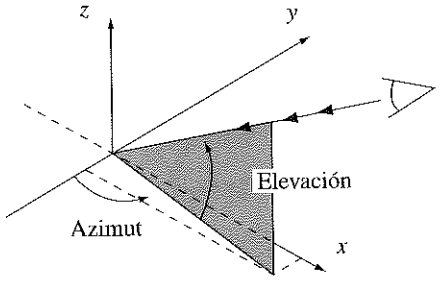
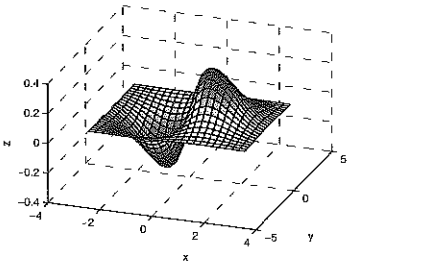


Figura 9.3: Ángulos azimut y de elevación.

- `az` es el azimut, es decir, el ángulo (en grados) medido en el plano *x*-*y* a partir del eje *y* negativo, y definido positivo en la dirección contraria a las agujas del reloj.
- `el` es el ángulo de elevación (en grados) desde el plano *x*-*y*. Un valor positivo indicará un ángulo que se abre en la dirección del eje *z*.
- Los ángulos de visión por defecto son $az = -37{,}5°$ y $el = 30°$.

Por ejemplo, el gráfico de superficie de la Tabla 9.1 se representa de nuevo en la Figura 9.4 con ángulos de visión $az = 20°$ y $el = 35°$.



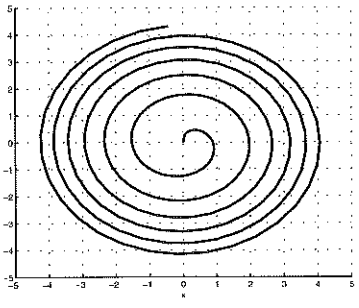
```
x = -3:0.25:3;
y = -3:0.25:3;
[X, Y] = meshgrid(x,y);
Z = 1.8.^(-1.5*sqrt(X.^2 +
   Y.^2)).*cos(0.5*Y).*sin(X);
surf(X,Y,Z);
view(20,35);
```

Figura 9.4: Gráfico de superficie de la función $z = 1{,}8^{-1{,}5\sqrt{x^2+y^2}} \operatorname{sen}(x) \cos(0{,}5y)$, con ángulos de visión $az = 20°$ y $el = 35°$.

- Escogiendo el azimut y la elevación apropiados, el comando `view` permite dibujar proyecciones 3-D en varios planos, según la siguiente tabla:

Plano de proyección	**Valor *az***	**Valor *el***
x-*y* (vista aérea)	0	90
x-*z* (vista lateral)	0	0
y-*z* (vista lateral)	90	0

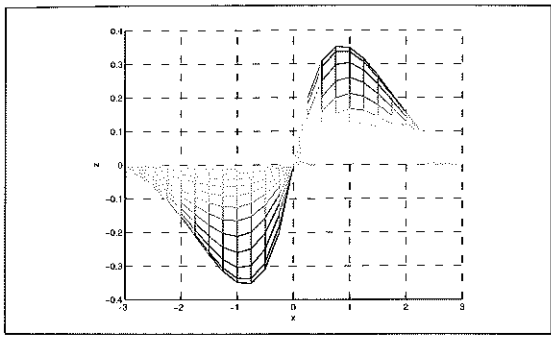
A continuación, en la Figura 9.5, se muestra un ejemplo de ángulos de visión que permiten una vista aérea de la representación gráfica, a partir del gráfico dibujado en la Figura 9.1.



```
t = 0:0.1:6*pi;
x = sqrt(t).*sin(2*t);
y = sqrt(t).*cos(2*t) ;
z = 0.5*t ;
plot3(x,y,z,'k','linewidth',1)
view(0,90)
grid on
xlabel('x'); ylabel('y'); zlabel('z')
```

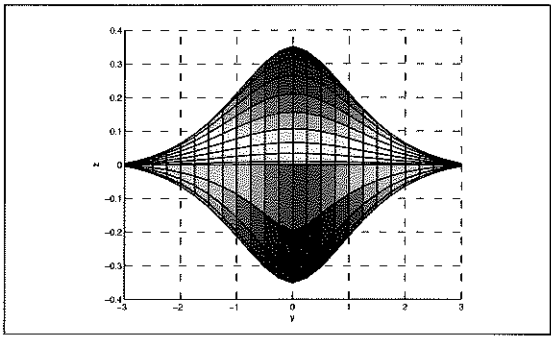
Figura 9.5: Vista aérea de la función $x = \sqrt{t} \operatorname{sen}(2t)$, $y = \sqrt{t} \cos(2t)$, $z = 0{,}5t$ para $0 \le t \le 6\pi$.

En las Figuras 9.6 y 9.7 se presentan ejemplos de proyecciones sobre los planos x-z e y-z, respectivamente. Estos gráficos muestran proyecciones de malla de la función dibujada en la Tabla 9.1.



```
x = -3:0.25:3;
y = -3:0.25:3;
[X, Y] = meshgrid(x,y);
Z = 1.8.^(-1.5*sqrt(X.^2 +
   Y.^2)).*cos(0.5*Y).*sin(X);
mesh(X,Y,Z);
view(0,0);
```

Figura 9.6: Proyección en el plano x-z de la función $z = 1{,}8^{-1{,}5\sqrt{x^2+y^2}} \text{ sen } (x) \cos (0{,}5y)$.



```
x = -3:0.25:3;
y = -3:0.25:3;
[X, Y] = meshgrid(x,y);
Z = 1.8.^(-1.5*sqrt(X.^2 +
   Y.^2)).*cos(0.5*Y).*sin(X);
surf(X,Y,Z);
view(90,0);
```

Figura 9.7: Proyección en el plano y-z de la función $z = 1{,}8^{-1{,}5\sqrt{x^2+y^2}} \text{ sen } (x) \cos (0{,}5y)$.

- El comando `view` también permite establecer una vista por defecto:

  `view(2)` establece por defecto la vista aérea, que es una proyección en el plano x-y con $az = 0°$ y $el = 90°$.

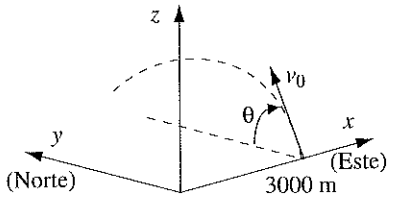
  `view(3)` establece la vistan estándar 3-D con ángulos $az = -37{,}5°$ y $el = 30°$.

- La dirección de visión se puede establecer también seleccionando un punto en el espacio desde donde se quiere ver el gráfico. En este caso, el comando `view` tendrá la sintaxis: `view([x,y,z])`, donde `x, y` y `z` son las coordenadas del punto. La dirección queda determinada por el punto en cuestión y el origen del sistema de coordenadas, independientemente de la distancia. Esto significa que, por ejemplo, la vista es la misma desde el punto [0, 0, 1] que desde el punto [10, 10, 10]. La vista aérea se puede establecer con [0, 0, 1]. Una vista lateral del plano x-z desde la dirección del eje y negativo se puede establecer con [0, –1, 0], y así sucesivamente.

9.5 Ejemplos de aplicaciones con MATLAB

Problema de ejemplo 9.1: Trayectoria 3-D de un proyectil

Se dispara un proyectil con una velocidad inicial de 250 m/s, con un ángulo relativo al suelo de $\theta = 65°$. El proyectil se dispara en dirección norte. Debido a los fuertes vientos que soplan hacia el oeste, el proyectil se mueve también en esta dirección a una velocidad constante de 30 m/s. Calcular y trazar la trayectoria del proyectil hasta que éste alcanza el suelo. A modo de comparación, trazar también (en la misma figura) la trayectoria que supuestamente debería haber seguido el proyectil si no hubiera intervenido el viendo.



Solución

Como se muestra en la figura adjunta, el sistema de coordenadas se ha establecido de forma que los ejes x e y apuntan en dirección este y norte, respectivamente. Por tanto, el movimiento del proyectil se puede analizar considerando la dirección vertical z, así como las dos componentes horizontales x e y. Como el proyectil se dispara hacia el norte, la velocidad inicial v_0 se puede descomponer en su componente horizontal y y su componente vertical z:

$$v_{0y} = v_0 \cos(\theta) \quad \text{y} \quad v_{0z} = v_0 \operatorname{sen}(\theta)$$

Además, a causa del viento el proyectil adquiere una velocidad constante en la dirección del eje x negativo, $v_x = -30$ m/s.

La posición inicial del proyectil (x_0, y_0, z_0) está en el punto (3000, 0, 0). En la dirección vertical, la velocidad y la posición del proyectil vienen dadas por:

$$v_z = v_{0z} - gt \quad \text{y} \quad z = z_0 + v_{0z}t - \frac{1}{2}gt^2$$

El tiempo que necesita el proyectil para alcanzar el punto más alto ($v_z = 0$) es $t_{hmax} = \dfrac{v_{0z}}{g}$.

El tiempo total de vuelo es dos veces ese tiempo, $t_{tot} = 2t_{hmax}$. En la dirección horizontal, la velocidad es constante (en ambas direcciones x e y), y la posición del proyectil viene dada por:

$$x = x_0 + v_x t \quad \text{y} \quad y = y_0 + v_{0y}t$$

El siguiente programa MATLAB resuelve las ecuaciones anteriores y nos da la solución al problema.

```
v0 = 250; g = 9.81; theta = 65;
x0 = 3000; vx = -30;
v0z = v0*sin(theta*pi/180);
v0y = v0*cos(theta*pi/180);
t = 2*v0z/g;
tplot = linspace(0,t,100);
```

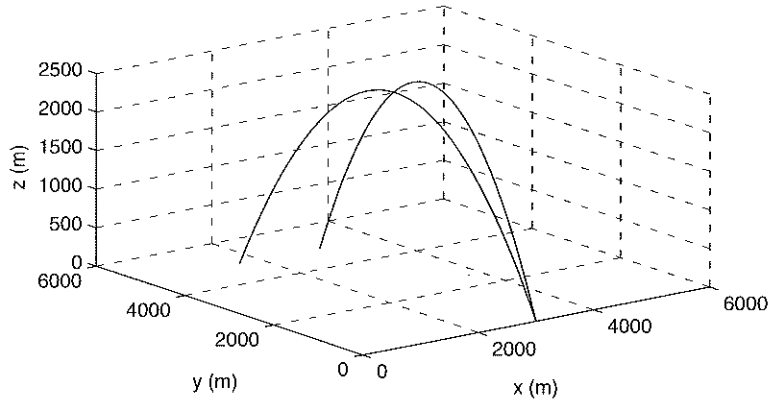
Se crea un vector con 100 elementos.

```
z = v0z*tplot - 0.5*g*tplot.^2;
y = v0y*tplot;
x = x0 + vx*tplot;
xnowind(1:length(y)) = x0;
plot3(x,y,z,'k-',xnowind,y,z,'k--');
grid on
axis([0 6000 0 6000 0 2500]);
xlabel('x(m)'); ylabel('y(m)'); zlabel('z(m)');
```

Se calculan las coordenadas x, y, z del proyectil en cada instante.

Coordenada x constante si no hubiera habido viento.

Representación gráfica en 3-D.



Problema de ejemplo 9.2: Potencial eléctrico de dos cargas puntuales

El potencial eléctrico V alrededor de una partícula cargada viene dado por:

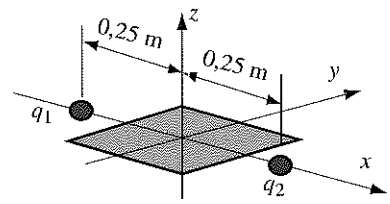
$$V = \frac{1}{4\pi\varepsilon_0}\frac{q}{r}$$

donde $\varepsilon_0 = 8{,}8541878 \times 10^{-12}\dfrac{\text{C}}{\text{N}\cdot\text{m}^2}$ es la permitividad (coeficiente dieléctrico) del vacío, q es la magnitud de la carga en culombios, y r es la distancia de la partícula en metros. El campo eléctrico de dos o más partículas se calcula por superposición. Por ejemplo, el potencial eléctrico en un punto debido a dos partículas vendrá dado por la expresión:

$$V = \frac{1}{4\pi\varepsilon_0}\left(\frac{q_1}{r_1} + \frac{q_2}{r_2}\right)$$

donde q_1, q_2, r_1 y r_2 con las cargas de las partículas y la distancia del punto a la partícula correspondiente, respectivamente.

Dos partículas con cargas $q_1 = 2 \times 10^{-10}$ C y $q_2 = 2 \times 10^{-10}$ C se sitúan en el plano x-y en las coordenadas (0,25, 0, 0) y (–0,25, 0, 0), respectivamente, tal y como se muestra en la figura adjunta. Calcular y representar el potencial eléctrico, debido a las dos partículas, en los puntos del plano x-y situados en el dominio $-0{,}2 \le x \le 0{,}2$ y $-0{,}2 \le y \le 0{,}2$ (las unidades del plano x-y están en metros). Hacer un gráfico que represente los puntos en el plano x-y y la magnitud del potencial eléctrico en el eje z.



Solución

El problema se resolverá siguiendo los pasos que se describen a continuación:

a) Se crea un rejilla en el plano x-y en el dominio $-0{,}2 \le x \le 0{,}2$ y $-0{,}2 \le y \le 0{,}2$.

b) Se calcula la distancia que hay entre cada punto de la rejilla y cada una de las cargas.

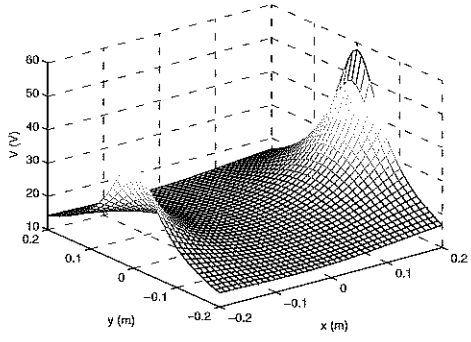
c) Se calcula el potencial eléctrico en cada punto.

d) Se representa el potencial eléctrico.

El siguiente fichero script muestra la resolución del problema siguiendo cada uno de los pasos descritos.

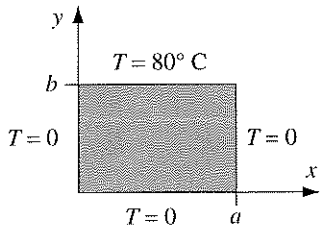
```
eps0 = 8.85e-12; q1 = 2e-10; q2 = 3e-10;
k = 1/(4*pi*eps0);
x = -0.2:0.01:0.2;
y = -0.2:0.01:0.2;
[X,Y] = meshgrid(x,y);                    % Se crea la rejilla en el plano x-y.
r1 = sqrt((X+0.25).^2+Y.^2);              % Se calcula la distancia r1 para cada punto de la rejilla.
r2 = sqrt((X-0.25).^2+Y.^2);              % Se calcula la distancia r2 para cada punto de la rejilla.
V = k*(q1./r1 + q2./r2);                  % Se calcula el potencial eléctrico V para cada punto de la rejilla.
mesh(X,Y,V);
xlabel('x(m)'); ylabel('y(m)'); zlabel('V(V)')
```

El gráfico generado por el programa es:



Problema de ejemplo 9.3: Conducción del calor en una placa rectangular

Tres de los cuatro lados de una placa rectangular ($a = 5$ m, $b = 4$ m) se mantienen a una temperatura $T = 0$ °C, y el lado restante a una temperatura $T_1 = 80$ °C, tal y como se muestra en la figura adjunta. Calcular y representar la distribución de la temperatura $T(x, y)$ en la placa.



Solución

La distribución de la temperatura, $T(x, y)$, en la placa se puede calcular resolviendo la ecuación del calor en dos dimensiones. Para las condiciones frontera dadas en el problema, $T(x, y)$ se puede expresar de forma analítica mediante la siguiente serie de Fourier (información extraída del libro de Erwin Kewyszing, "Advanced Engineering Mathematics", Hohn Wiley and Sons, 1993):

$$T(x, y) = \frac{4T_1}{\pi} \sum_{n=1}^{\infty} \frac{\text{sen}\left[(2n-1)\frac{\pi x}{a}\right] \text{senh}\left[(2n-1)\frac{\pi y}{a}\right]}{(2n-1) \quad \text{senh}\left[(2n-1)\frac{\pi b}{a}\right]}$$

A continuación se muestra el programa script que resuelve este problema. El programa sigue los siguientes pasos para la resolución:

a) Se crea una rejilla X, Y en el dominio $0 \le x \le a$ y $0 \le y \le b$. La longitud de la placa, a, se divide en 20 segmentos, y la anchura, b, en 16.

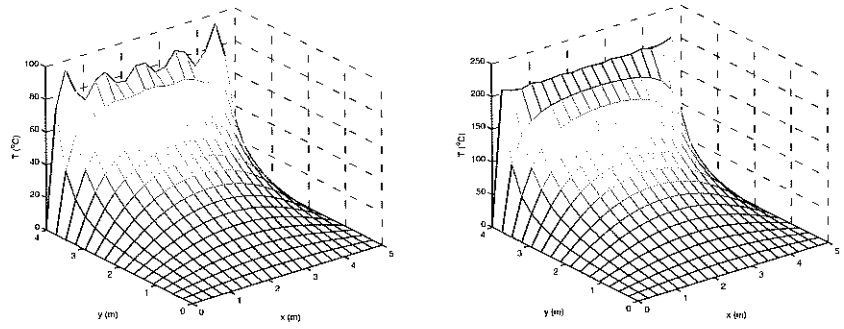
b) Se calcula la temperatura en cada punto de la malla. Los cálculos se harán punto a punto, utilizando un bucle anidado. En cada punto la temperatura se calculará sumando los k términos de la serie de Fourier.

c) Se representa un gráfico de superficie de T.

```
a = 5; b = 4; na = 20; nb = 16; T0 = 80; k = 5;
clear T
x = linspace(0,a,na);
y = linspace(0,b,nb);
[X,Y] = meshgrid(x,y);                    % Se crea la rejilla en el plano x-y.
for i = 1:nb                              % Primer bucle, con i como índice para recorrer las filas de la rejilla.
  for j = 1:na                            % Segundo bucle, con j como índice para recorrer las columnas de la rejilla.
    T(i,j) = 0;
    for n = 1:k                           % Tercer bucle, donde n es el enésimo término de la serie
      ns = 2*n-1;                         % de Fourier, y k es el número de términos de la serie.
      T(i,j) = T(i,j) + sin(ns*pi*X(i,j)/a).*sinh(ns*pi*Y(i,j)/a)/(sinh(ns*pi*b/a)*ns);
    end
```

```
      T(i,j) = T(i,j)*4*T0/pi;
    end
  end
  mesh(X,Y,T);
  xlabel('x(m)'); ylabel('y(m)'); zlabel('T(^oC)')
```

El programa se ejecuta dos veces. Primero utilizando 5 términos para la serie de Fourier ($k = 5$) y después utilizando 50 ($k = 50$). A continuación se muestran los gráficos de las dos ejecuciones.



La temperatura $y = 4$ debe ser uniforme y de 80 °C. Observe el efecto del número de términos (k) sobre la precisión en $y = 4$. El gráfico de la derecha por tanto es más preciso que el de la izquierda.

9.6 Problemas

1. La posición en función del tiempo de una partícula en movimiento viene dada por:

$$y = (2 + 4\cos(t))\cos(t)$$
$$x = (2 + 4\cos(t))\,\text{sen}(t)$$
$$z = t^2$$

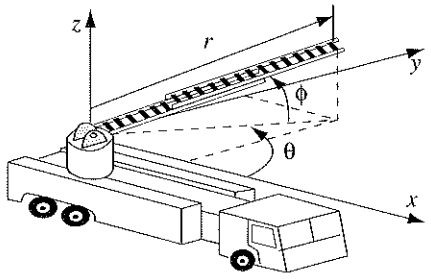
   Represente la posición de la partícula para $0 \leq t \leq 20$.

2. Una escalera de caracol se puede modelar utilizando las siguientes ecuaciones paramétricas:

$$x = R\cos\left(2\pi n\frac{t}{h}\right)$$
$$y = R\,\text{sen}\left(2\pi n\frac{t}{h}\right)$$
$$z = \frac{t}{h}$$

donde R es el radio de la escalera, h es la altura entre los pisos y n es el número de revoluciones de la escalera en cada piso. Sea un edificio de 2 pisos, con $h = 3$ m. Represente dos posibles escaleras de caracol para este edificio. Una con $R = 1{,}5$ m y $n = 3$, y otra con $R = 4$ m y $n = 2$. Represente las dos escaleras en el mismo gráfico.

3. La escalera de un camión de bomberos puede elevarse (incremento del ángulo ϕ), rotar sobre el eje z (incremento del ángulo θ), y extenderse (incremento de r). Inicialmente la escalera descansa sobre el vehículo: $\phi = 0$, $\theta = 0$ y $r = 8$ m. A continuación la escalera comienza a cambiar de posición con el tiempo, elevándose 8 grados/s, rotando 8 grados/s y extendiéndose 0,6 m/s. Calcule y represente gráficamente la posición del extremo final de la escalera durante 10 segundos.



4. Haga un gráfico de superficie 3-D y otro de contorno 3-D de la función $z = -\frac{x^2}{4} - \frac{y^2}{4}$ en el dominio $-4 \le x \le 4$ y $-4 \le y \le 4$.

5. Haga un gráfico de superficie 3-D y otro de contorno (ambos en la misma gráfica) de la función $z = (y + 3)^2 + 1{,}5x^2 - x^2y$ en el dominio $-3 \le x \le 3$ y $-3 \le y \le 3$.

6. La ley de los gases ideales relaciona presión, temperatura y volumen de un gas, de la forma:

$$P = \frac{nRT}{V}$$

donde P es la presión en Pa, n es el número de moles, $R = 8{,}31$ J/mol-K es la constante de los gases, T es la temperatura en grados K y V es el volumen en m^3.

Haga un gráfico 3-D que muestre la variación de la presión (variable dependiente, eje z) con respecto al volumen (variable independiente, eje x) y la temperatura (variable independiente, eje y) de un mol de gas. Los dominios del volumen y la temperatura son: $0{,}5 \times 10^{-3} \le V \le 2 \times 10^{-3}$ m^3, y $273 \le T \le 473$ K.

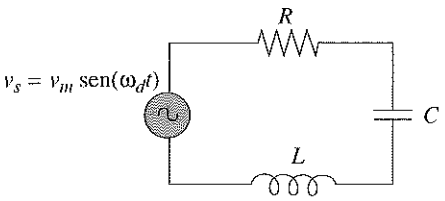
7. Las moléculas de un gas que se encuentran en el interior de un recipiente se mueven en todas las direcciones a diferentes velocidades. La ley de distribución de velocidades de Maxwell proporciona la distribución probabilística $P(v)$ en función de la temperatura y de la velocidad:

$$P(v) = 4\pi\left(\frac{M}{2\pi RT}\right)^{3/2} v^2 e^{(-Mv^2)/(2RT)}$$

donde M es la masa molar del gas en kg/mol, $R = 8{,}31$ J/mol-K es la constante de los gases, T es la temperatura en grados K y v es la velocidad de las moléculas en m/s.

Realice la representación 3-D de $P(v)$ en función de v y T, para $0 \le v \le 1000$ m/s y $70 \le T \le 320$ grados K, para moléculas de oxígeno (masa molar 0,032 kg/mol).

8. En la figura adjunta se muestra un circuito *RLC* con una fuente de voltaje alterna. El voltaje v_s de la fuente viene dado por la expresión $v_s = v_m \operatorname{sen}(\omega_d t)$, en donde $\omega_d = 2\pi f_d$, en la cual f_d es la frecuencia de excitación. La intensidad de corriente, I, en este circuito vendrá dada por:



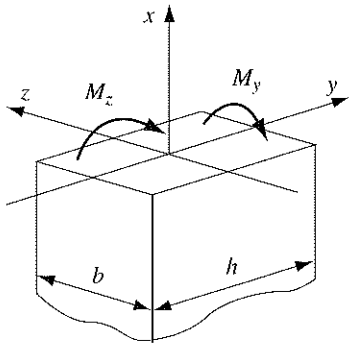
$$I = \frac{v_m}{\sqrt{R^2 + (\omega_d L - 1/(\omega_d C))^2}}$$

donde R y C representan el valor de la resistencia y la capacidad del condensador, respectivamente. Para un circuito como el de la figura, con $C = 15 \times 10^{-6}$ F, $L = 240 \times 10^{-3}$ H, y $v_m = 24$ V:

a) En un gráfico 3-D represente I (eje z) en función de ω_d (eje x) para $60 \le f \le 110$ Hz, y en función de R (eje y) para $10 \le R \le 40\ \Omega$.

b) Haga un gráfico que sea una proyección en el plano x-z. A partir de este gráfico estime la frecuencia natural del circuito (la frecuencia a la cual I es máxima). Compare la estimación con el valor calculado: $1/(2\pi\sqrt{LC})$.

9. La tensión normal, σ_{xx}, debida a los momentos de torsión M_z y M_y en un punto (y, z) en la sección transversal de una viga rectangular, viene dada por:



$$\sigma_{xx} = \frac{-M_z y}{I_{zz}} + \frac{-M_y z}{I_{yy}}$$

donde I_{zz} e I_{yy} son los momentos de inercia del área, definidos de la forma:

$$I_{zz} = \frac{1}{12} b h^3 \quad \text{e} \quad I_{yy} = \frac{1}{12} h b^3.$$

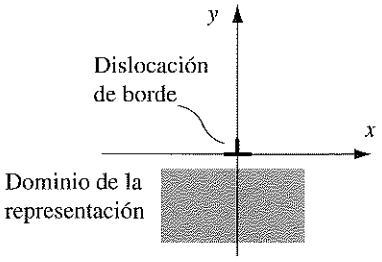
Determine y represente la tensión normal en la superficie de la sección transversal de la viga mostrada en la figura adjunta. Utilice los valores: $h = 40$ mm, $b = 30$ mm, $M_y = 2500$ N-m y $M_z = 3600$ N-m. Dibuje las coordenadas y y z en el plano horizontal, y la tensión normal en la dirección vertical.

10. Al defecto producido en la estructura de un cristal, en donde se ha perdido una fila de átomos, se le denomina dislocación de borde. El campo de esfuerzos alrededor de la dislocación de borde viene dada por:

$$\sigma_{xx} = \frac{-Gb}{2\pi(1-\nu)} \frac{y(3x^2 + y^2)}{(x^2 + y^2)^2}$$

$$\sigma_{yy} = \frac{Gb}{2\pi(1-\nu)} \frac{y(x^2 - y^2)}{(x^2 + y^2)^2}$$

$$\tau_{xy} = \frac{Gb}{2\pi(1-\nu)} \frac{x(x^2 - y^2)}{(x^2 + y^2)^2}$$



Represente las componentes de los esfuerzos (cada una en un gráfico separado) a causa de una dislocación de borde en aluminio, para la cual $G = 27{,}7 \times 10^9$ Pa, $b = 0{,}286 \times 10^{-9}$ m y $v = 0{,}334$. Represente los esfuerzos en el dominio $-5 \times 10^{-9} \leq x \leq 5 \times 10^{-9}$ m y $-5 \times 10^{-9} \leq y \leq -1 \times 10^{-9}$ m. Dibuje las coordenadas x e y en el plano horizontal, y los esfuerzos en la dirección vertical.

Capítulo 10
Aplicaciones de análisis numérico

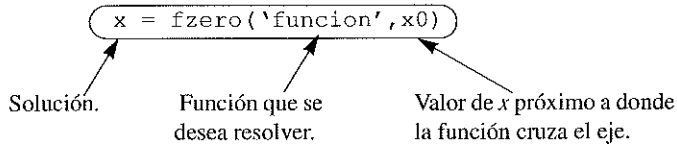
Los métodos numéricos se utilizan habitualmente para resolver problemas matemáticos formulados en diferentes campos científicos y técnicos, donde es difícil, o incluso imposible, obtener soluciones exactas a un problema dado. MATLAB posee distintas librerías de funciones pensadas para la resolución numérica de una gran variedad de problemas matemáticos. Este capítulo muestra cómo se usan algunas de las funciones más frecuentemente utilizadas para la resolución de problemas numéricos. Es importante destacar, no obstante, que el cometido de este capítulo es enseñar al lector cómo utilizar MATLAB para el propósito señalado, y no entrar en explicaciones sobre los diferentes métodos numéricos utilizados, cuya información puede encontrarse en cualquier otro libro relacionado más directamente con el análisis numérico.

En este capítulo se verá la resolución de ecuaciones con una incógnita, el cálculo de máximos y mínimos de una función, la integración numérica y la resolución de ecuaciones diferenciales ordinarias de primer orden.

10.1 Resolución de ecuaciones de una variable

Las ecuaciones de una variable se pueden escribir de la forma $f(x) = 0$. La solución a la ecuación será el valor x donde la función cruza el eje x (donde el valor de la función es cero), lo que significa que la función cambia de signo en x. La solución exacta es el valor de x para el cual el valor de la función es exactamente 0 (cero de una función). Si este valor no existe o es difícil de obtener, se puede calcular una solución numérica encontrando un x próximo al punto donde la función cambia de signo (cruza el eje x). Este cálculo se puede realizar mediante un proceso iterativo, donde en cada paso el programa calculará el valor de x próximo a la solución. El proceso iterativo termina cuando la diferencia de x entre dos iteraciones es menor que algún valor determinado. En general, una función puede tener ninguna, una, varias o un número infinito de soluciones.

En MATLAB, el cero de una función se puede obtener con el comando `fzero`, cuya sintaxis es:



Más detalles sobre los argumentos de la función `fzero`:

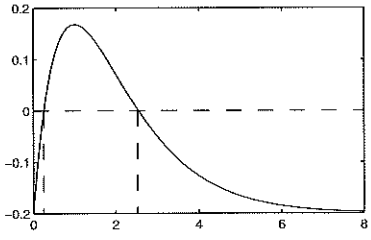
- `x` es la solución de la ecuación. Es un valor escalar.
- `'funcion'` es la función que se debe resolver. Este argumento se puede representar de tres formas distintas:
  1) La forma más sencilla es introducir la expresión matemática como si fuera una cadena.
  2) También se puede crear la función utilizando una función definida por el usuario en un fichero de función (véase Capítulo 6), e introducir luego el nombre de la función como cadena.
  3) La función se puede crear como función en línea (véase Sección 6.8), para luego introducir su nombre como cadena.
- La función se debe expresar en la forma estándar. Por ejemplo, si la función que hay que resolver es $xe^{-x} = 0{,}2$, la función se deberá escribir como $f(x) = xe^{-x} - 0{,}2 = 0$ o, lo que es lo mismo, se introduciría como cadena en el argumento `'funcion'` de la forma: `'x*exp(-x)-0.2'`.
- Cuando la función se introduce como cadena, ésta no puede incluir variables predefinidas. Por ejemplo, si la función introducida es $f(x) = xe^{-x} - 0{,}2$, no es posible definir $b = 0{,}2$ e introducir: `'x*exp(-x)-b'`.
- `x0` puede ser un escalar o un vector de dos elementos. Si se introduce como escalar, tiene que ser un valor x próximo al punto donde la función cruza el eje x. Si `x0` se introduce como vector, los dos elementos deben ser puntos opuestos de la solución tal que $f(x0(1))$ tenga un signo diferente a $f(x0(2))$. Cuando una función tiene más de una solución, cada solución se puede calcular por separado utilizando la función `fzero` e introduciendo valores `x0` que estén próximos a las soluciones.
- Una buena forma de averiguar cuándo una función tiene una solución es realizar una representación gráfica de la función. En muchas aplicaciones científicas se puede estimar el dominio de la solución. A menudo, cuando una función tiene más de una solución, sólo una de ellas tiene sentido con respecto al cálculo esperado.

Problema de ejemplo 10.1: Solución a una ecuación no lineal

Calcular la solución de la ecuación $xe^{-x} - 0{,}2$.

Solución

La ecuación se escribirá, primeramente, en forma estándar de función: $f(x) = xe^{-x} - 0{,}2$. En el gráfico adjunto se representa la función. En este gráfico puede verse cómo la función tiene una solución entre 0 y 1, y otra solución entre 2 y 3. El gráfico se obtiene tecleando en la Ventana de Comandos:




```
>> fplot('x*exp(-x) - 0.2',[0 8])
```

La solución de la función se calcula utilizando el comando `fzero` dos veces; una con el valor `x0` entre 0 y 1 (p. ej., `x0` = 0,7), y otra con `x0` entre 2 y 3, (p. ej., `x0` = 2,8):

```
>> x1 = fzero('x*exp(-x) - 0.2',0.7)
x1 =
   0.2592
```

La primera solución es 0,2592.

```
>> x1 = fzero('x*exp(-x) - 0.2',2.8)

x1 =
   2.5426
```

La segunda solución es 2,5426.

Más comentarios:

- El comando `fzero` busca los ceros de la función sólo donde la función cruza el eje x. El comando no encuentra ceros en puntos donde la función toca pero no acaba cruzando dicho eje.
- Si no se puede calcular la solución a una función, en el valor de salida x se devolverá `NaN`.
- El comando `fzero` tiene otras opciones (véase la ayuda de MATLAB). Dos de las más importantes son: `[x fval] = fzero('funcion',x0)` que asigna el valor de la función en `x` a la variable `fval`. Y `x = fzero('funcion',x0,optimset('display','iter'))`, que visualiza la salida de cada iteración durante el proceso de búsqueda de la solución para la función.
- Cuando la función se puede escribir en forma polinómica, la solución, o raíces del polinomio, se puede buscar directamente con el comando `roots`, tal y como se explicó en el Capítulo 8 (Sección 8.1.2).
- El comando `fzero` se puede utilizar también para encontrar el valor de x donde la función toma un valor concreto. Para hacer esto sólo hay que volver a escribir la función según el valor deseado para ésta. Por ejemplo, en la función del Problema de ejemplo 10.1, el primer valor de x donde la función toma el valor 0,1 se puede calcular resolviendo la ecuación $xe^{-x} - 0{,}2 = 0{,}1$, es decir, $xe^{-x} - 0{,}3 = 0$. A continuación se muestra el código asociado para encontrar esta solución:

```
>> x = fzero('x*exp(-x) - 0.3',0.5)

x =
   0.4894
```

10.2 Cálculo de un máximo o de un mínimo de una función

En determinadas ocasiones es necesario encontrar un mínimo o máximo local de una función de la forma $y = f(x)$. En Cálculo, el valor de x que se corresponde con un máximo o un mínimo local se determina igualando a cero la derivada de la función. Luego, este valor de x se sustituye en la función para obtener el valor de y. En MATLAB, el valor de x donde una función $f(x)$ de una sola variable tiene un mínimo en el intervalo $x_1 \leq x \leq x_2$, se puede calcular con el comando `fminbnd`, cuya sintaxis es:

```
x = fminbnd('funcion',x1,x2)
```

Valor de x donde la función tiene un mínimo. — Función. — Intervalo de x.

- La función se puede introducir en forma de cadena, o como el nombre de una función definida por el usuario en un fichero o el de una función en línea.

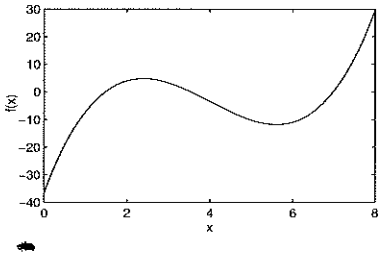
- El valor de la función en el punto mínimo también se puede visualizar utilizando esta otra sintaxis de la misma función:

```
[x fval] = fminbnd('funcion',x1,x2)
```

donde el valor de la función en el punto `x` se asigna a la variable `fval`.

- Dentro de un intervalo dado, el mínimo de una función puede ser uno de los extremos del intervalo o un punto interior del intervalo donde la pendiente de la función es cero (mínimo local). Cuando se ejecuta el comando `fminbnd`, MATLAB busca un mínimo local. Si se encuentra un mínimo local, su valor será comparado con el valor de la función en los puntos extremos del intervalo. MATLAB devuelve el punto con el valor del mínimo real del intervalo.

Por ejemplo, considere la función $f(x) = x^3 - 12x^2 + 40{,}25x - 36{,}5$, cuyo gráfico en el intervalo $0 \leq x \leq 8$ puede verse en la figura adjunta. Como se puede observar, existe un mínimo local de la función entre los valores 5 y 6, pero el mínimo absoluto está en $x = 0$. Utilizando la función `fminbnd` en el intervalo $3 \leq x \leq 8$ se puede encontrar la posición del mínimo local, así como el valor de la función en dicho punto:




```
>> [x fval] = fminbnd('x^3 - 12*x^2 + 40.25*x - 36.5',3,8)

x =
   5.6073

fval =
  -11.8043
```

El mínimo local se encuentra en el punto x = 5,6073.
El valor de la función en ese punto es –11,8043.

Si se cambia el intervalo a $0 \leq x \leq 8$, el programa retornará el valor del mínimo absoluto, y no del local:

```
>> [x fval] = fminbnd('x^3 - 12*x^2 + 40.25*x - 36.5',0,8)

x =
   0

fval =
  -36.5000
```

El mínimo se encuentra en el punto $x = 0$.
El valor de la función en ese punto es –36,5.

Para este intervalo, el valor del mínimo está en uno de los puntos extremos del intervalo ($x = 0$).

- El comando `fminbnd` también se puede utilizar para encontrar el máximo de una función. Para hacer esto se multiplica la función por –1 y se busca su mínimo. Por ejemplo, el máximo de la función $f(x) = xe^{-x} - 0{,}2$ (del Problema de ejemplo 10.1), en el intervalo $0 \leq x \leq 8$, se puede calcular buscando el mínimo de la función $f(x) = -xe^{-x} + 0{,}2$, como se muestra a continuación:

```
>> [x fval] = fminbnd('-x*exp(-x) + 0.2',0,8)

x =
   1.0000

fval =
  -0.1679
```

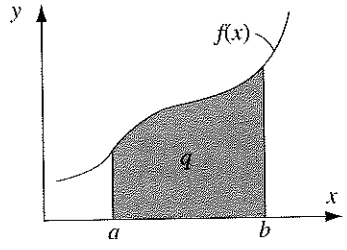
El máximo se encuentra en el punto $x = 1{,}0$.
El valor de la función en ese punto es $-0{,}1679$.

10.3 Integración numérica

La integración es una operación matemática muy habitual para resolver problemas en distintos campos científicos y técnicos. Las integrales se utilizan para calcular áreas y volúmenes, la velocidad a partir de la aceleración, el trabajo a partir de la fuerza y el desplazamiento, etc. La integración de funciones sencillas se puede realizar de forma analítica, pero muchas funciones son difíciles, e incluso imposibles, de integrar de forma analítica. En los cursos de cálculo el integrando (lo que se integra) viene dado normalmente por una función. En aplicaciones científicas y técnicas el integrando puede venir definido como una función o como un conjunto de puntos. Por ejemplo: una serie de puntos de medidas discretas sobre velocidad de flujo a partir de los cuales se puede calcular el volumen.

En esta sección se supone que el usuario posee conocimientos previos sobre integrales e integración. La integral definida de una función $f(x)$, en el intervalo desde a hasta b, tiene la forma:

$$q = \int_a^b f(x)\, dx$$



La función $f(x)$ es el integrando y los números a y b los límites de integración. De forma gráfica, el valor de la integral q es el área sombreada que se encuentra por debajo de la curva de la función y esta limitada por los límites a y b y el eje x. Cuando se calcula una integral definida de forma analítica, $f(x)$ es siempre una función. Cuando la integral se calcula numéricamente, $f(x)$ puede ser una función o una serie de puntos. En la integración numérica, el área total se obtiene dividiendo el área en secciones más pequeñas, calculando el área de cada sección y sumando posteriormente todas las áreas. Para este propósito se han desarrollado diferentes métodos numéricos. La diferencia entre uno y otro es la forma en que se divide el área en secciones y la forma en que se calcula el área de cada sección. El lector puede consultar libros sobre análisis numérico para obtener más detalles sobre estas técnicas.

A continuación se describe cómo se utilizan las tres funciones MATLAB para la integración. Estas funciones son `quad`, `quadl` y `trapz`. Los comandos `quad` y `quadl` se utilizan para integrales donde $f(x)$ es una función, mientras que la función `trapz` se utiliza cuando $f(x)$ viene dada en forma de puntos.

El comando `quad`

La sintaxis del comando `quad`, basado en el método adaptativo de integración de Simpson, es la siguiente:

```
q = quad('funcion',a,b)
```

Valor de la integral.

Función que será integrada.

Límites de integración.

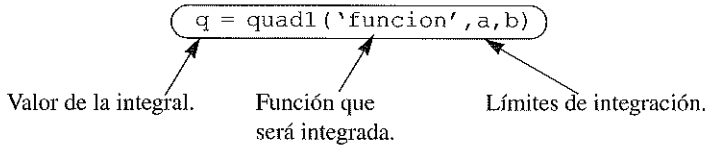
- La función se puede introducir como cadena o como el nombre de una función definida por el usuario o de una función en línea, al igual que el comando `fzero` (véase Sección 10.1 para más detalles).
- La función $f(x)$ se debe escribir para un argumento x que permita realizar operaciones elemento a elemento, de forma que se pueda calcular el valor de la función para cada elemento x.
- El usuario debe asegurarse de que la función no tiene ninguna asíntota vertical entre a y b.
- El comando `quad` calcula la integral con un error absoluto menor que 1,0e-6. Este número se puede cambiar añadiendo el parámetro opcional `tol` al comando `quad`, de la forma:

```
q = quad('funcion',a,b,tol)
```

donde `tol` es un número que define el error máximo. Asignando un valor grande a este número la integral tendrá menos precisión, pero su cálculo será más rápido.

El comando `quadl`*:*

El comando `quadl` (la última letra es una L minúscula) tiene la misma sintaxis que el comando `quad`:



Todos los comentarios referidos a la función quad son también válidos para el comando `quadl`. La diferencia entre estos dos comandos estriba en el método numérico utilizado para la integración. El comando `quadl` utiliza el método adaptativo de Lobatto, que es más eficiente cuando se requiere un nivel de precisión más alto para el cálculo de integrales.

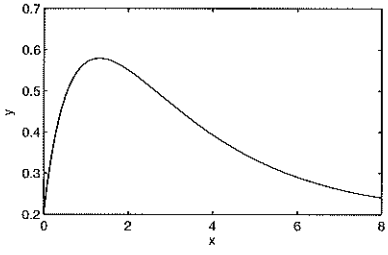
Problema de ejemplo 10.2: Integración numérica de una función

Utilizar integración numérica para calcular la siguiente integral:

$$\int_0^8 (xe^{-x^{0,8}} + 0{,}2)dx$$

Solución

En la figura adjunta se muestra un gráfico de la función para el intervalo $0 \leq x \leq 8$. La solución al problema utiliza el comando `quad` y muestra dos formas de introducir la función para el cálculo. La primera de ellas consiste en introducir directamente la expresión como argumento. En la segunda se utiliza el nombre de un fichero de función, creado previamente, para introducir la integral que se va a calcular .



A continuación se muestra cómo se utiliza el comando `quad` en la Ventana de Comandos para calcular el resultado directamente. Como puede comprobarse, la función se teclea utilizando operaciones elemento a elemento:

```
>> quad('x.*exp(-x.^0.8) + 0.2',0,8)

ans =
   3.1604
```

El segundo método consiste en crear un fichero de función que calcule la función que se va a integrar. El fichero de función (llamado `y = Cap10Ej2(x)`) es el siguiente:

```
function y = Cap10Ej2(x)
y = x.*exp(-x.^0.8) + 0.2;
```

Observe, nuevamente, que la función se escribe utilizando operaciones elemento a elemento, como si el argumento `x` fuera un vector. La integración se lleva a cabo en la Ventana de Comandos, tecleando dentro del comando `quad` el nombre del fichero de función en el argumento `'funcion'`:

```
>> q = quad('Cap10Ej2',0,8)

q =
   3.1604
```

El comando `trapz`*:*

Este comando se puede utilizar para integrar una función que se da en forma de datos o puntos. Este método utiliza integración por el método de los trapecios. La sintaxis de este comando es:

```
q = trapz(x,y)
```

donde `x` e `y` son vectores con las coordenadas x e y de los puntos que se van a integrar. Los dos vectores deben tener el mismo tamaño.

10.4 Ecuaciones diferenciales ordinarias

Las ecuaciones diferenciales juegan un papel esencial en las disciplinas científicas y técnicas, ya que la mayoría de los fenómenos físicos se pueden modelar utilizando este tipo de ecuaciones. Sólo un número limitado de ecuaciones diferenciales pueden resolverse analíticamente. Los métodos numéricos, por otro lado, pueden dar una solución aproximada de la mayoría de estas ecuaciones. Sin embargo, obtener una solución numérica no es una tarea sencilla, ya que no existe un método numérico general que pueda resolver todas las ecuaciones diferenciales existentes. MATLAB posee una librería con un gran número de herramientas que pueden ser utilizadas para la resolución de ecuaciones diferenciales. Para poder sacar el máximo provecho a estas herramientas, el usuario debe saber trabajar con ecuaciones diferenciales, así como estar familiarizado con los métodos apropiados para poder resolverlas.

En esta sección se describe en detalle cómo se utiliza MATLAB para resolver ecuaciones diferenciales ordinarias de primer orden. Se describirán los métodos numéricos utilizados para resolver este tipo de ecuaciones, aunque éstos no se explicarán desde el punto de vista matemático. Esta sección proporciona información para resolver ecuaciones de primer orden *no demasiado problemáticas*. Estos métodos proporcionan la pauta para resolver ecuaciones de orden superior y sistemas de ecuaciones más completos.

Una ecuación diferencial ordinaria (ODE: del inglés Ordinary Differential Equation) es una ecuación que contiene una variable independiente, una variable dependiente, así como derivadas de la variable dependiente. Las ecuaciones que se tratarán en esta sección son ecuaciones de primer orden, cuya forma es:

$$\frac{dy}{dx} = f(x, y)$$

donde x e y representan la variable independiente y dependiente, respectivamente. La solución es una función $y = f(x)$ que satisface la ecuación. En general, existen muchas funciones que dan solución a una determinada ODE, por lo que se necesita más información para calcular la solución a un problema específico. Esta información adicional es el valor de la función (variable dependiente) para ciertos valores de la variable independiente.

Pasos en la resolución de una ODE simple de primer orden:

De aquí en adelante se considerará t (el tiempo) como variable independiente. La razón es que en muchas aplicaciones el tiempo es casi siempre la variable independiente, así además la explicación será coherente con la ayuda que proporciona MATLAB en el menú **Help** (Ayuda).

Paso 1: Escribir el problema en la forma estándar

Esto es, escribir la ecuación en la forma:

$$\frac{dy}{dt} = f(t, y) \quad \text{para } t_0 \le t \le t_f, \quad \text{con } y = y_0 \quad \text{en} \quad t = t_0.$$

Como se puede observar, para resolver una ODE de primer orden es necesario conocer tres cosas importantes: una ecuación que da la expresión para la derivada y con respecto a t, el intervalo de la variable independiente y el valor inicial de y. La solución es el valor de y en función t, entre t_0 y t_f.

He aquí un problema de ejemplo:

$$\frac{dy}{dt} = \frac{t^3 - 2y}{t} \quad \text{para } 1 \le t \le 3, \quad \text{con} \quad y = 4{,}2 \quad \text{en} \quad t = 1.$$

Paso 2: Crear un fichero de función

Crear una función (en un fichero) que calcule $\frac{dy}{dt}$ para los valores dados de t e y. Para el problema anterior, el fichero de función sería el siguiente:

```
function dydt = ODEexp1(t,y)
dydt = (t^3 - 2*y)/t ;
```

Paso 3: Seleccionar un método para hallar la solución

Se trata de seleccionar el método para el cálculo de la solución. Hasta la fecha se han desarrollado multitud de métodos para calcular ODEs de primer orden. Algunos de estos métodos están disponibles en forma de función MATLAB. Típicamente, en estos métodos el intervalo de tiempo se divide en pequeños subintervalos o pasos. La solución comienza en el punto conocido y_0. Seguidamente, utili-

zando uno de los métodos de integración, se calcula el valor de y en cada paso. En la Tabla 10.1 se listan siete métodos distintos para resolver ODEs de primer orden, implementados en MATLAB mediante funciones. En la tabla se detalla además una breve descripción de cada método.

Tabla 10.1: Funciones MATLAB para la resolución de ODEs.

Nombre de la función	Descripción
`ode45`	Para ecuaciones no demasiado complejas. Obtiene la solución en un solo paso, ideal para intentar obtener una primera aproximación. Se basa en el método Runge-Kutta.
`ode23`	Para ecuaciones no demasiado complejas. Obtiene la solución en un solo paso. Está también basado en el método Runge-Kutta. Es más rápido pero menos preciso que el método `ode45`.
`ode113`	Para ecuaciones no demasiado complejas. Obtiene la solución en múltiples pasos.
`ode15s`	Para ecuaciones complejas. Obtiene la solución en varios pasos. Se utiliza cuando `ode45` falla.
`ode23s`	Para ecuaciones complejas. Obtiene la solución en un solo paso. Permite resolver algunas ecuaciones que no puede resolver `ode15`.
`ode23t`	Para ecuaciones de dificultad media.
`ode23tb`	Para ecuaciones complejas. A veces más eficiente que `ode15s`.

En general, los métodos de cálculo de ODEs se pueden dividir en dos grupos según su habilidad para resolver problemas fáciles o difíciles, y también en función de si se usan uno o varios pasos para la resolución de la ecuación. Las ecuaciones más difíciles son aquellas que incluyen componentes que varían de forma rápida y lenta, y requieren pequeños subintervalos de tiempo para obtener la solución. Los métodos que calculan la solución en un solo paso necesitan la información de un punto para obtener una solución próxima al punto en cuestión. Los métodos planificados en varios pasos utilizan información de varios puntos previos para encontrar la solución en el punto siguiente. Los detalles de cada uno de estos métodos se encuentran fuera del alcance de este libro.

Es necesario conocer de antemano qué método es el más apropiado para un problema específico. Para hacer esto se puede utilizar, como primera opción, el método `ode45`. Este método da, por lo general, buenos resultados para la mayoría de los problemas. Si no se obtiene ninguna solución adecuada debido a la dificultad de la ecuación, se puede utilizar, por ejemplo, el método `ode15s`.

Paso 4: Resolución de la ODE

El formato del comando que se utiliza para resolver ODEs para valores concretos es el mismo para todos los métodos vistos anteriormente. La sintaxis general es:

```
[t,y] = nombre_metodo('funcODE',tspan,y0)
```

Información adicional:

`nombre_metodo` Es el nombre del método (numérico) usado (por ejemplo: `ode45`, `ode23s`, etc.)

`'funcODE'` Es el nombre, en forma de cadena, de la función (fichero de función) que calcula $\frac{dy}{dt}$ para valores dados de t e y.

`tspan` Es un vector que especifica el intervalo de la solución. El vector debe tener al menos dos elementos, pero puede tener más. Si el vector tiene sólo dos elementos, los elementos deben ser `[t0,tf]`, que son el punto inicial y final, respectivamente, del intervalo de la solución. Sin embargo, el vector `tspan` puede tener puntos adicionales entre el primero y el último. El número de elementos de `tspan` afecta a la salida del comando. Véase `[t,y]` más abajo.

`y0` Es el valor inicial de y (valor de y en el primer punto del intervalo).

`[t,y]` Es la salida del comando, es decir, la solución a la ecuación diferencial. Los valores t e y son vectores columna. Los puntos primero y último representan el punto inicial y final del intervalo, respectivamente. El espaciado y el número de puntos dependen del vector de entrada `tspan`. Si `tspan` tiene dos elementos (puntos inicial y final), los vectores t e y contendrán la solución en cada paso de integración calculado por el método en cuestión. Por el contrario, si `tspan` tiene más de dos puntos (puntos adicionales entre el inicial y el final), los vectores t e y sólo contendrán la solución para esos puntos. El número de puntos en `tspan` no afecta a los pasos (subintervalos de tiempo) utilizados para la solución de la ecuación.

Por ejemplo, la solución al problema presentado en el Paso 1:

$$\frac{dy}{dt} = \frac{t^3 - 2y}{t} \quad \text{para } 1 \le t \le 3, \quad \text{con } y = 4{,}2 \quad \text{en} \quad t = 1.$$

se puede obtener de la forma:

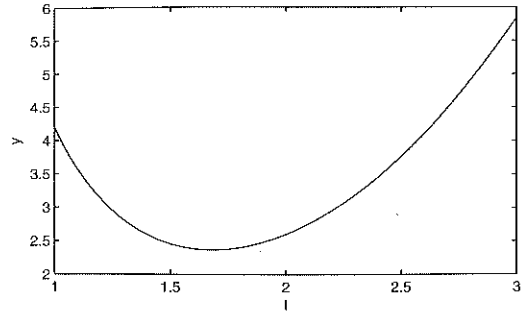
```
>> [t,y] = ode45('ODEexp1',[1:0.5:3],4.2)

t =
   1.0000
   1.5000
   2.0000
   2.5000
   3.0000

y =
   4.2000
   2.4528
   2.6000
   3.7650
   5.8444
```

Vector `tspan` — Valor inicial

Esta solución se ha calculado mediante el método `ode45`. El nombre de la función utilizada (la del Paso 2) es: `'ODEexp1'`. La solución comienza en $t = 1$ y finaliza en $t = 3$, con incrementos de 0,5 (según el vector `tspan`). Para visualizar la solución, el problema se resuelve de nuevo utilizando `tspan` con un espaciado menor. Esta solución se representa gráficamente mediante el comando `plot`:

```
>> [t,y] = ode45('ODEexp1',[1:0.01:3],4.2)
>> plot(t,y)
>> xlabel('t'), ylabel('y')
```

El gráfico que se crea es el siguiente:



10.5 Ejemplos de aplicaciones MATLAB

Problema de ejemplo 10.3: Ecuación de un gas

La ecuación de un gas ideal relaciona las magnitudes volumen (V en L), temperatura (T en grados K), presión (P en atm) y cantidad de gas (número n de moles) mediante la siguiente expresión:

$$p = \frac{nRT}{V}$$

donde $R = 0{,}08206$ L-atm/mol-K es la constante del gas.

La ecuación de van der Walls proporciona la relación entre estas magnitudes para un gas real:

$$\left(p + \frac{n^2 a}{V^2}\right)(V - nb) = nRT$$

donde a y b son constantes específicas para cada gas.

Utilizar la función `fzero` para calcular el volumen de 2 mol de CO_2 a una temperatura de 50 °C y una presión de 6 atm. Las constantes específicas para el CO_2 son: $a = 3{,}59$ L^2-atm/mol^2, y $b = 0{,}0427$ L/mol.

Solución

A continuación se muestra la solución escrita en un fichero script.

```
global P T n a b R
R = 0.08206;
P = 6; T = 323.2; n = 2; a = 3.59; b = 0.047;
Vest = n*R*T/P;                    Cálculo de un valor estimado para V.
V = fzero('Waals',Vest)
```

El programa calcula en primer lugar un valor estimado para el volumen utilizando la ecuación del gas ideal. Posteriormente se utiliza ese valor en el comando `fzero` para estimar la solución. La ecuación de van der Waals se representa en forma de fichero de función, con el nombre `Waals`:

```
function fofx = Waals(x)
global P T n a b R
fofx = (P + n^2*a/x^2)*(x - n*b) - n*R*T;
```

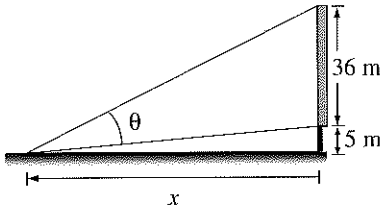
Para que el script y el fichero de función puedan ejecutarse correctamente, las variables `P`, `T`, `n` `a`, `b` y `R` deben ser declaradas como globales. Cuando se guarda el script (con el nombre Cap10ProbEj3) y se ejecuta en la Ventana de Comandos, éste nos proporciona el siguiente resultado:

```
>> Cap10ProbEj3
V =
  8.6613
```

El volumen del gas es 8,6613 L.

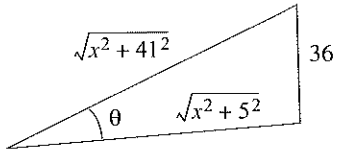
Problema de ejemplo 10.4: Ángulo máximo de visión

Para obtener el mejor ángulo de visión de una película, una persona tiene que sentarse a una distancia x de la pantalla, de forma que el ángulo de visión θ sea máximo. Calcular la distancia x para la cual el ángulo θ es máximo en función de los valores dados en la figura adjunta.



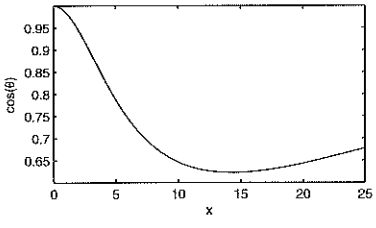
Solución

El problema se puede resolver encontrando una función para el ángulo θ en función de x, y después encontrando el valor x para el cual θ es máximo. En el triángulo que incluye el ángulo θ, uno de los lados tiene un valor conocido (la altura de la pantalla). Los otros dos lados se pueden representar en función de x, tal y como se muestra en la figura. Una forma de representar θ en función de x es mediante la ley de cosenos:



$$\cos(\theta) = \frac{(x^2 + 5^2) + (x^2 + 41^2) - 36^2}{2\sqrt{x^2 + 5^2}\sqrt{x^2 + 41^2}}$$

El ángulo θ estará comprendido entre 0 y $\pi/2$. Puesto que $\cos(0) = 1$, el valor del coseno disminuye cuando se incrementa el ángulo θ, el ángulo máximo se corresponderá con el menor valor de $\cos(\theta)$. A continuación se muestra una representación gráfica de $\cos(\theta)$ en función de x, donde se puede apreciar un mínimo entre 10 y 20. Los comandos utilizados para la representación son:



```
>> fplot('((x^2 + 5^2) + (x^2 + 41^2) - 36^2)/(2*sqrt(x^2 + 5^2)*sqrt(x^2 + 41^2))',[0 25])
>> xlabel('x'); ylabel('cos(\theta)')
```

El mínimo se puede calcular mediante la función `fminbnd`:

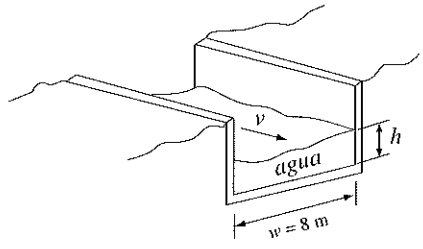
```
>> [x angulocos] = fminbnd('((x^2 + 5^2) + (x^2 + 41^2) - 36^2)/(2*sqrt(x^2 + 5^2)*sqrt(x^2 + 41^2))',10,20)
x =
   14.3178
angulocos =
   0.6225
>> angulo = angulocos*180/pi
angulo =
   35.6674
```

El mínimo se sitúa en $x = 14{,}3178$ m. En este punto, $\cos(\theta) = 0{,}6225$.

El valor del ángulo, en grados, es 35,6674°.

Problema de ejemplo 10.5: Flujo de agua de un río

Para estimar la cantidad de agua que fluye por un río durante un año, se toma una sección rectangular del río como la que se muestra en la figura adjunta. Al principio de cada mes (empezando desde el uno de enero) se mide tanto la altura h del agua como la velocidad v de la corriente. El primer día de la medida se considera como 1, y el último día (uno de enero del año siguiente) se corresponde con 366. A continuación se muestran los distintos valores medidos:



Día	1	32	60	91	121	152	182	213	244	274	305	335	366
h (m)	2,0	2,1	2,3	2,4	3,0	2,9	2,7	2,6	2,5	2,3	2,2	2,1	2,0
v (m/s)	2,0	2,2	2,5	2,7	5	4,7	4,1	3,8	3,7	2,8	2,5	2,3	2,0

Utilizar estos datos para calcular los caudales para cada uno de los datos y luego integrarlos para obtener una cantidad estimada del total de agua que circula por el río a lo largo del año.

Solución

El caudal Q (volumen de agua por segundo) para cada punto de la tabla anterior se obtiene multiplicando la velocidad del agua por el ancho y el alto de la sección de río considerada:

$$Q = vwh \ (\text{m}^3/\text{s})$$

El total de agua que fluye se puede estimar mediante la integral:

$$V = (60 \cdot 60 \cdot 24) \int_{t_1}^{t_2} Q \, dt$$

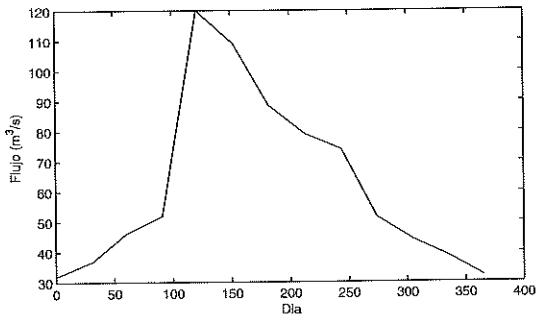
El caudal viene dado en metros cúbicos por segundo, lo que significa que el tiempo debe venir dado en segundos. Como los datos de la tabla anterior vienen en días, la integral se multiplicará por (60*60*24) segundos/día.

El siguiente programa, en forma de fichero script, resuelve el problema calculando Q e integrando, posteriormente, mediante la función `trapz`. El programa además genera un gráfico del caudal en función del tiempo.

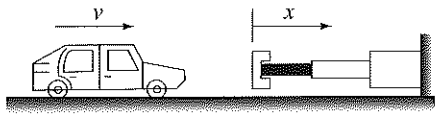
```
w = 8;
d = [1 32 60 91 121 152 182 213 244 274 305 335 366];
h = [2 2.1 2.3 2.4 3.0 2.9 2.7 2.6 2.5 2.3 2.2 2.1 2.0];
velocidad = [2 2.2 2.5 2.7 5 4.7 4.1 3.8 3.7 2.8 2.5 2.3 2];
Q = velocidad*w.*h;
Vol = 60*60*24*trapz(d,Q);
fprintf('La cantidad estimada de agua que fluye por el rio en un annio es de %g metros cubicos.',Vol);
plot(d,Q);
xlabel('Dia'), ylabel('Flujo (m^3/s)')
```

Cuando el fichero (guardado en disco como Cap10ProEj5) se ejecuta en la Ventana de Comandos, el gráfico y la cantidad estimada de agua se muestran como solución al problema:

```
>> Cap10ProEj5
La cantidad estimada de agua que fluye por el rio en un annio es de 2.03095e+009 metros cubicos.
```



Problema de ejemplo 10.6: Impacto de un coche contra una barrera de protección



Una barrera de protección se sitúa al final de un circuito con el objetivo de parar coches que han perdido el control. Esta barrera se ha diseñado de forma que la fuerza que la barrera aplica al coche viene dada en función de la velocidad v y del desplazamiento x de la parte frontal de la barrera, según la expresión:

$$F = Kv^3(x+1)^3$$

donde $K = 30$ s-kg/m^5 es una constante.

Un coche, con una masa m de 1500 kg, impacta contra la barrera de protección a una velocidad de 90 km/h. Calcular y representar la velocidad del coche en función de su posición para $0 \le x \le 3$ m.

Solución

La desaceleración del coche, una vez que éste ha impactado contra la barrera, se puede calcular utilizando la segunda ley de Newton:

$$ma = -Kv^3(x+1)^3$$

Esta ecuación se puede resolver para la aceleración a en función de v y x:

$$a = \frac{-Kv^3(x+1)^3}{m}$$

La velocidad, en función de x, se puede calcular sustituyendo la aceleración en la ecuación:

$$v\,dv = a\,dx$$

lo que resulta:

$$\frac{dv}{dx} = \frac{-Kv^2(x+1)^3}{m}$$

Esta última ecuación es una ecuación diferencia ordinaria de primer orden, que se resolverá en el intervalo $0 \le x \le 3$ con la condición inicial $v = 90$ km/h en $x = 0$.

La solución numérica para la ecuación diferencial se muestra a continuación en el siguiente fichero script:

```
global k m
k = 30; m = 1500; v0=90;
xspan = [0:0.2:3];          % Vector que especifica el intervalo de la solución.
v0mps = v0*1000/3600;       % Cambio de unidad a m/s para v0.
[x v] = ode45('barrera',xspan,v0mps)   % Resolución de la ecuación diferencial.
plot(x,v)
xlabel('x(m)'); ylabel('velocidad (m/s)')
```

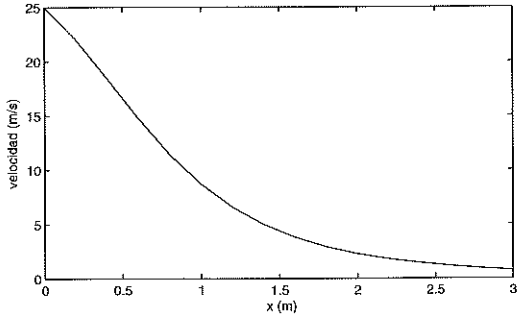
El fichero de función para la ecuación diferencial se llama `barrera`:

```
function dvdx = barrera(x,v)
global k m
dvdx = -(k*v^2*(x + 1)^3)/m;
```

Cuando se ejecuta el script (guardado en disco como Cap10ProEj6), se visualizan los vectores x y v en la Ventana de Comandos (se han puesto juntos para optimizar de espacio, aunque en la ejecución original aparecen uno detrás del otro):

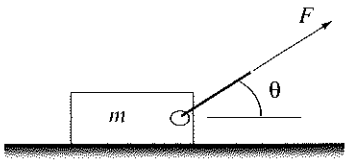
```
>> Cap10ProEj6
x =          v =
   0         25.0000
 0.2000      22.0420
 0.4000      18.4478
 0.6000      14.7561
 0.8000      11.4302
 1.0000       8.6954
 1.2000       6.5733
 1.4000       4.9793
 1.6000       3.7960
 1.8000       2.9220
 2.0000       2.2737
 2.2000       1.7886
 2.4000       1.4226
 2.6000       1.1435
 2.8000       0.9283
 3.0000       0.7607
```

También se visualiza el gráfico generado por el programa:



10.6 Problemas

1. Calcule la solución a la ecuación: $\cos(x) = 2x^3$
2. Calcule las primeras tres raíces positivas de la ecuación: $2\text{sen}(x) - \sqrt{x} = -2{,}5$.
3. Calcule las tres primeras raíces positivas de la ecuación: $4\cos(2x) - e^{0{,}5x} + 5 = 0$
4. Una caja con una masa $m = 20$ kg es arrastrada mediante una cuerda. La fuerza necesaria para mover la caja viene dada por la expresión:

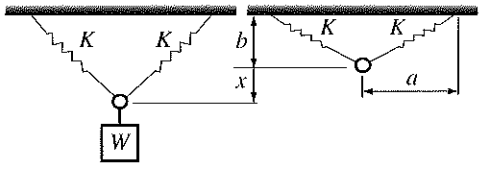


$$F = \frac{\mu mg}{\cos\theta + \mu\,\text{sen}\,\theta}$$

donde $\mu = 0{,}45$ es el coeficiente de rozamiento y $g = 9{,}81$ m/s^2. Calcule el ángulo θ si la fuerza de arrastre es de 92 N.

5. Una báscula se compone de dos muelles, tal y como se muestra en la figura adjunta. Inicialmente, los dos muelles se encuentran contraídos (no estirados). Cuando se cuelga un objeto del anillo, los muelles se estiran y el anillo se desplaza hacia abajo una distancia x. El peso de cada objeto se puede expresar en términos de esa distancia x, mediante la expresión:

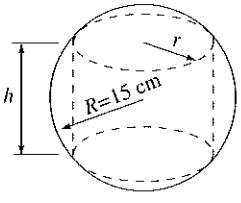
$$W = \frac{2K}{L}(L - L_0)(b + x)$$



donde $L_0 = \sqrt{a^2 + b^2}$ es la longitud inicial de un muelle, y $L = \sqrt{a^2 + (b + x)^2}$ es la longitud del muelle estirado.

Para una determinada báscula $a = 0{,}15$ m, $b = 0{,}05$ m, con constante para los muelles $K = 2800$ N/m, calcule la distancia x cuando un objeto de 250 N se cuelga a la báscula.

6. Considere de nuevo la caja que se arrastraba en el Problema 4. Calcule el ángulo θ para que la fuerza que se requiere para tirar de la caja sea mínima. ¿Cuál es el módulo de esta fuerza?
7. Una copa cilíndrica de papel se diseña para albergar 200 mL. Calcule el radio y la altura de la copa de manera que se utilice la mínima cantidad de papel para su fabricación.
8. Calcule las dimensiones (radio r y altura h) de un cilindro con el mayor volumen posible que se puede construir con una esfera de radio $R = 15$ cm.



9. La ley de radiación de Planck da la radiación espectral R en función de la longitud de onda λ y la temperatura T (en grados K):

$$R = \frac{2\pi c^2 h}{\lambda^5} \frac{1}{e^{(hc)/(\lambda kT)} - 1}$$

donde $c = 3{,}0 \times 10^8$ m/s es la velocidad de la luz, $h = 6{,}63 \times 10^{-34}$ J-s es la constante de Planck y $k = 1{,}38 \times 10^{-23}$ J/K es la constante de Boltzmann.

Represente R en función de λ para $0{,}2 \times 10^{-6} \leq \lambda \leq 6{,}0 \times 10^{-6}$ m a una temperatura de $T = 1500$ K. Calcule además la longitud de onda que proporciona la radiación espectral R máxima a esa temperatura.

10. Utilice MATLAB para calcular la siguiente integral:

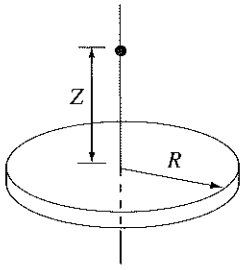
$$\int_0^5 \frac{1}{0{,}8x^2 + 0{,}5x + 2}\, dx$$

11. Utilice MATLAB para calcular la siguiente integral:

$$\int_0^{\pi} \cos^2(0{,}5x)\, \text{sen}^4(0{,}5x)\, dx$$

12. El campo eléctrico E debido a un disco circular cargado, en un punto situado a una distancia z a lo largo del eje del disco, viene dado por la expresión:

$$E = \frac{\sigma z}{4\varepsilon_0} \int_0^R (z^2 + r^2)^{-3/2} (2r)\, dr$$



donde σ es la densidad de carga, $\varepsilon_0 = 8{,}85 \times 10^{-12}$ C^2/N-m^2 es la permitividad (coeficiente dieléctrico del vacío), y R es el radio del disco. Calcule el campo eléctrico en un punto situado a 5 cm de un disco de 6 cm de radio y cargado con $\sigma = 300$ μC/m^2.

13. La variación de la aceleración de la gravedad g con respecto a la altitud viene dada por la expresión:

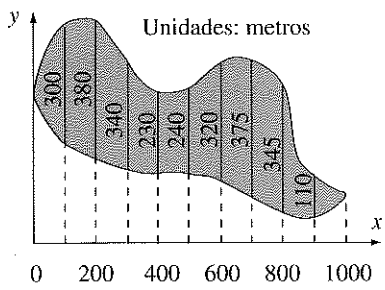
$$g = \frac{R^2}{(R + y)^2} g_0$$

donde $R = 6371$ km es el radio de la tierra, y $g_0 = 9{,}81$ m/s^2 es la aceleración de la gravedad al nivel del mar. La variación de la energía potencial gravitatoria, ΔU, de un objeto que se eleva desde la tierra viene dada por:

$$\Delta U = \int_0^h mg\, dy$$

Calcule la variación de la energía potencial de un satélite de masa $m = 500$ kg que se eleva desde la superficie de la tierra hasta una altura de 800 km.

14. El área de la superficie de un lago se estima midiendo el ancho del lago en intervalos de 100 m. Las medidas tomadas se muestran en el gráfico adjunto. Utilice integración numérica para estimar el área del lago.



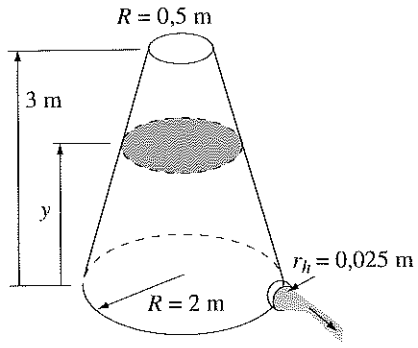
15. Resuelva y represente la solución de:

$$\frac{dy}{dx} = \frac{x^2 - 2x + 3}{y^2} \quad \text{para } 0{,}5 \le x \le 3, \quad \text{con } y(0, 5) = 2.$$

16. Resuelva y represente la solución de:

$$\frac{dy}{dx} = 0{,}2xy + 0{,}5y^2 \quad \text{para } 0{,}5 \le x \le 4, \quad \text{con } y(0) = -0{,}5.$$

17. Un depósito de agua con forma de cono truncado e invertido tiene un agujero circular en un lateral de su parte inferior, como se muestra en la figura adjunta. Según la ley de Torricelli, la velocidad v con la que el agua sale del depósito por el agujero viene dada por:



$$v = \sqrt{2gh}$$

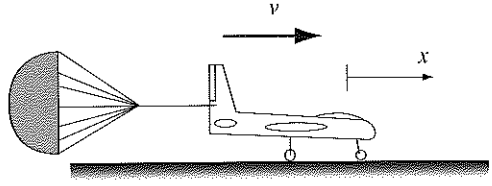
donde h es la altura del agua y $g = 9{,}81$ m/s^2. La relación que representa el cambio de la altura, y, del agua del depósito cuando ésta se sale, viene dada por:

$$\frac{dy}{dt} = \frac{\sqrt{2gyr_h^2}}{(2 - 0{,}5y)^2}$$

donde r_h es el radio del agujero.

Resuelva la ecuación diferencial para y. El valor inicial de la altura del agua es $y = 2$ m. Resuelva el problema para tiempos distintos, y encuentre el instante cuando $y = 0{,}1$ m. Represente y en función del tiempo.

18. Un aeroplano utiliza un paracaídas, así como otros medios de frenado, para detenerse en tierra después del aterrizaje. Su aceleración viene dada por $a = -0{,}0035v^2 - 3$ m/s^2. Como $a = \frac{dv}{dt}$, la relación que representa la variación de la velocidad viene dada por:



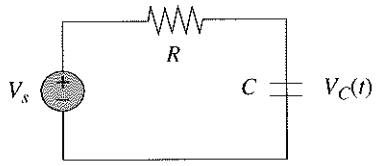
$$\frac{dv}{dt} = -0{,}0035v^2 - 3$$

Considere un aeroplano con una velocidad de 300 km/h que abre su paracaídas y comienza su desaceleración en el instante $t = 0$ s.

a) Resuelva la ecuación diferencial para calcular y representar la velocidad en función del tiempo, desde $t = 0$ s hasta que el aeroplano se detiene.

b) Utilice integración numérica para calcular la distancia x recorrida por el aeroplano en función del tiempo. Represente x en función del tiempo.

19. Un circuito RC incluye una fuente de voltaje v_s, una resistencia $R = 50\ \Omega$ y un condensador $C = 0{,}001$ F, tal y como se muestra en la figura adjunta. La ecuación diferencial que describe la respuesta del circuito es:



$$\frac{dv_c}{dt} + \frac{1}{RC}v_c = \frac{1}{RC}v_s$$

donde v_c es el voltaje del condensador. Inicialmente $v_s = 0$, y después, en $t = 0$, la fuente de voltaje se cambia. Calcule la respuesta del circuito para los tres casos siguientes:

a) $v_s = 12$ V para $t = 0$.

b) $v_s = 12\text{sen}(2 \cdot 60\ \pi t)$ V para $t = 0$.

c) $v_s = 12$ V para $0 \le t \le 0{,}01$ s, y después $v_s = 0$ para $t = 0{,}01$ s (pulso rectangular).

Cada caso corresponde a una ecuación diferencial distinta. La solución es el voltaje del condensador en función del tiempo. Resuelva cada caso para $0 \le t \le 0{,}2$ s. Represente además, para cada caso, v_s y v_c en función del tiempo (haga dos gráficos separados en la misma ventana).

Capítulo 11
Cálculo simbólico

Todas las operaciones matemáticas presentadas y efectuadas hasta el momento con MATLAB han sido de carácter numérico. Estas operaciones se calcularon escribiendo expresiones numéricas que podían contener diferentes números y variables con valores numéricos preasignados. Cuando MATLAB ejecuta una expresión numérica, el resultado es también numérico (un número o un array con números). Estos números pueden ser exactos o aproximados en coma flotante. Por ejemplo, tecleando 1/4 MATLAB devuelve 0,2500, un valor exacto, y tecleando 1/3 el sistema devuelve 0,3333, un valor aproximado.

En muchas aplicaciones matemáticas, científicas y técnicas se requieren operaciones simbólicas, es decir, operaciones matemáticas con expresiones que contienen variables simbólicas (variables que no tienen un valor numérico específico asignado cuando la operación se ejecuta). El resultado de tal operación es también una expresión matemática que contiene variables simbólicas. Un ejemplo sencillo de esto es despejar una variable de una ecuación algebraica de varias variables; por ejemplo si a, b y x son variables simbólicas y $ax - b = 0$, se desea calcular x a partir de a y b, cuyo resultado es $x = b/a$. Otros ejemplos de operaciones simbólicas son la resolución analítica de derivadas, integrales y ecuaciones diferenciales. Por ejemplo, la derivada de $2t^3 + 5t - 8$, con respecto a t, es $6t^2 + 5$.

MATLAB permite llevar a cabo diferentes tipos de operaciones simbólicas. La parte numérica de la operación simbólica se lleva a cabo de forma exacta, sin aproximar valores numéricos. Por ejemplo, el resultado de sumar $\frac{x}{4}$ y $\frac{x}{3}$ es $\frac{7}{12}x$ y no $0{,}5833x$.

MATLAB resuelve operaciones simbólicas a partir de una librería especial que debe estar instalada (Symbolic Math Toolbox). Esta librería es una colección de funciones MATLAB utilizadas para la resolución de operaciones simbólicas. Los comandos y funciones para las operaciones simbólicas tienen el mismo estilo y sintaxis que las operaciones numéricas. Las operaciones simbólicas son ejecutadas por Maple®, que es un software diseñado para este propósito. Maple está integrado dentro de MATLAB, y se activa de forma automática cuando se ejecuta una operación simbólica. Maple también existe como software independiente. Sin embargo, este software posee una estructura y comandos completamente diferentes a los de MATLAB. La librería de operaciones simbólicas está incluida en la versión para estudiantes de MATLAB. En la versión estándar de MATLAB, sin embargo, la librería debe ser comprada aparte. Para comprobar si la librería de operaciones simbólicas está instalada, el usuario puede teclear el comando `ver` en la Ventana de Comandos. La respuesta a este comando es la

información sobre la versión de MATLAB utilizada, así como un listado de las librerías (tool boxes) instaladas.

El punto de partida para comenzar a trabajar con operaciones simbólicas es el tratamiento de objetos simbólicos. Un objeto simbólico está compuesto por variables y números que, cuando se utilizan en operaciones matemáticas, indican a MATLAB que debe ejecutar la operación de forma simbólica. El usuario inicialmente define (crea) las variables simbólicas (objetos) necesarias, y después las utiliza para crear expresiones simbólicas. También se pueden utilizar expresiones simbólicas en operaciones numéricas, si fuera necesario.

La primera sección de este capítulo describe cómo se definen los objetos simbólicos y cómo se utilizan para crear expresiones simbólicas. La segunda sección muestra cómo se cambia la forma de expresiones ya creadas. Una vez que la expresión simbólica se crea, ésta se puede utilizar en operaciones matemáticas. MATLAB posee una gran colección de funciones para este propósito. Las cuatro próximas secciones (11.3 a 11.6) muestran cómo se utiliza MATLAB para resolver ecuaciones algebraicas. La Sección 11.7 explica cómo se representan gráficamente las expresiones simbólicas. Finalmente, en la Sección 11.8, se explicará cómo se utilizan las expresiones simbólicas en cálculos numéricos.

11.1 Objetos simbólicos y expresiones simbólicas

Un objeto simbólico puede ser una variable (sin un valor numérico preasignado), un número o una expresión obtenida a partir de variables y números simbólicos. Las expresiones simbólicas son expresiones matemáticas que contienen uno o más objetos simbólicos. A la hora de ser tecleados, las expresiones simbólicas tienen un aspecto similar al de las expresiones numéricas. Sin embargo, como estas expresiones contienen objetos simbólicos, éstas son ejecutadas por MATLAB de forma simbólica.

11.1.1 *Creación de objetos simbólicos*

Los objetos simbólicos pueden ser variables o números. Estos se pueden crear con los comandos `sym` y/o `syms`. Para crear un solo objeto simbólico se utiliza el comando `sym`:

```
nombre_objeto = sym('cadena')
```

donde `'cadena'` puede ser:

- Una letra o una combinación de varias letras (sin espacios). Por ejemplo: `'a'`, `'x'` o `'yad'`.
- Una combinación de letras y dígitos que comience por letra (sin espacios). Ejemplos: `'xh12'`, `'r2d2'`.
- Un número. Por ejemplo: `'15'` o `'4'`.

En los primeros dos casos (cuando la cadena es una letra o una combinación de varias letras y números), el objeto simbólico creado es una variable simbólica. En este caso es conveniente (aunque no necesario) dar al objeto el mismo nombre que la cadena. Por ejemplo, `a`, `bb` y `x` se pueden definir como variables simbólicas de la forma:

```
>> a = sym('a')          Se crea el objeto simbólico a.
a =
a                        El objeto simbólico se muestra al principio de la línea, sin sangrado.
>> bb = sym('bb')        Se crea el objeto simbólico bb.
```

```
bb =
bb
>> x = sym('x');
>>
```

El objeto simbólico se muestra al principio de la línea, sin sangrado.

La variable simbólica x se crea, pero no se visualiza. Para hacer esto sólo es necesario añadir un punto y coma al final.

El nombre del objeto simbólico puede ser diferente del nombre de la variable. Por ejemplo:

```
>> g = sym('gamma')
g =
gamma
```

El objeto simbólico se llama `gamma`, y el nombre del objeto es `g`.

Como se dijo anteriormente, los objetos simbólicos pueden ser también números. Los números no se deben teclear como cadenas. A continuación se verá cómo se utiliza el comando `sym` para crear objetos simbólicos a partir de los números 5 y 7, y después asignarlos a las variables *c* y *d*, respectivamente.

```
>> c = sym(5)
c =
5
>> d = sym(7)
d =
7
```

Se crea un objeto simbólico a partir del número 5, y se asigna a `c`.

El objeto simbólico se muestra al principio de la línea, sin sangrado.

Se crea un objeto simbólico a partir del número 7, y se asigna a `d`.

El objeto simbólico se muestra al principio de la línea, sin sangrado.

Como se muestra en el ejemplo anterior, cuando se crea un objeto simbólico y no se añade un punto y coma al final del comando, MATLAB muestra el nombre del objeto y el objeto propiamente dicho en las dos líneas siguientes. La visualización de un objeto simbólico comienza al principio de la línea, es decir, no se deja un espacio (sangrado) como en el caso de las variables numéricas. La diferencia se observa en el siguiente ejemplo, donde se crea una variable numérica;

```
>> e = 13
e =
   13
```

El número 13 se asigna a `e` (variable numerica).

El valor de la variable se muestra después de un espacio (sangrado).

Es posible crear más de una variable simbólica a la vez utilizando el comando `syms`:

```
syms nombre_variable nombre_variable nombre_variable ...
```

Este comando crea objetos simbólicos que tienen el mismo nombre que las variables simbólicas. Por ejemplo, las variables *y*, *z* y *d* se pueden crear de una vez como variables simbólicas tecleando:

```
>> syms y z d
>> y
y =
y
```

Las variables creadas con `syms` no se visualizan automáticamente. Para ver la variable es necesario teclear su nombre.

Cuando se ejecuta el comando syms, las variables se crean pero no se visualizan automáticamente, aún cuando no se ponga un punto y coma al final del comando.

11.1.2 *Creación de expresiones simbólicas*

Las expresiones simbólicas son expresiones matemáticas que contienen variables simbólicas. Una vez que las variables simbólicas han sido creadas, éstas se pueden utilizar para crear expresiones simbólicas. Las expresiones simbólicas son objetos simbólicos en sí. La forma de crear una expresión simbólica es:

```
Nombre_expresion = Expresion matematica
```

He aquí algunos ejemplos:

```
>> syms a b c x y
>> f = a*x^2 + b^x + c
f =
a*x^2+b^x+c
```

Define las variables simbólicas `a`, `b`, `c`, `x` e `y`.

Crea la expresión simbólica $ax^2 + bx + c$, y la asigna a `f`.

Se visualiza la expresión simbólica, sin sangrar.

Cuando se introduce una expresión simbólica que incluye operaciones matemáticas que se pueden ejecutar (suma, resta, multiplicación y división), MATLAB las ejecuta a medida que se crean. Por ejemplo:

```
>> g = 2*a/3 + 4*a/7 - 6.5*x + x/3 + 4*5/3 - 1.5
g =
26/21*a-37/6*x+31/6
```

Se introduce: $\frac{2a}{3} + \frac{4a}{7} - 6{,}5x + \frac{x}{3} + 4\frac{5}{3} - 1{,}5$.

Se visualiza: $\frac{26}{21}a - \frac{37}{6}x + \frac{31}{6}$,

Como puede comprobarse, las operaciones anteriores se realizan sin hacer aproximaciones numéricas. De esta forma MATLAB sumó $\frac{2}{3}a$ y $\frac{4}{7}a$ para obtener $\frac{26}{21}a$, y $-6{,}5x + \frac{x}{3}$ para obtener $\frac{37}{6}x$. Las operaciones con términos que sólo contienen números en las expresiones simbólicas se llevaron a cabo de forma exacta. Así pues, se ha sustituido $4 \cdot \frac{5}{3} + 1{,}5$ por $\frac{31}{6}$.

En el siguiente ejemplo se mostrará la diferencia entre los cálculos exactos y los aproximados, donde se lleva a cabo la misma operación matemática con variables simbólicas y variables numéricas.

```
>> a = sym(3); b = sym(5);
>> e = b/a + sqrt(2)
e =
5/3+2^(1/2)
>> c = 3; d = 5;
>> f = d/c + sqrt(2)
f =
   3.0809
```

Define a y b como representaciones simbólicas de 3 y 5, respectivamente.

Crea una expresión que incluye a y b.

El valor exacto de e se visualiza en forma de objeto simbólico (sin sangrar).

Se definen c y d como variables numéricas y se les asignan los valores 3 y 5, respectivamente.

Se crea una expresión que incluye c y d.

El valor aproximado de f se visualiza en forma de número (con sangría).

Una expresión se puede crear incluyendo objetos simbólicos y numéricos. Sin embargo, si la expresión incluye objetos simbólicos (uno o varios), todas las operaciones matemáticas se llevarán a cabo de forma exacta (sin calcular un valor numérico final para la expresión). Por ejemplo, si *c* se reemplaza por *a* en la última expresión del ejemplo anterior, el resultado será exacto:

```
>> f = d/a + sqrt(2)
f =
5/3+2^(1/2)
```

Comentarios adicionales sobre expresiones y objetos simbólicos:

- Las expresiones simbólicas pueden incluir variables numéricas obtenidas de la ejecución de expresiones numéricas. Cuando estas variables se introducen en expresiones simbólicas, se utilizará su valor exacto, incluso si la variable se visualizó anteriormente con su valor aproximado. Por ejemplo:

```
>> h = 10/3
h =
   3.3333
>> k = sym(5); m = sym(7);
>> p = k/m + h
p =
85/21
```

Se define h como 10/3 (variable numérica).

Se visualiza el valor aproximado de h (variable numérica).

Se definen los objetos simbólicos k y m que contienen 5 y 7, respectivamente.

h, k y m se utilizan en una expresión.

Se visualiza el valor exacto de p, en forma de objeto simbólico. Se usa el valor exacto de h para la obtención de p.

- Se puede utilizar el comando `double(S)` para convertir una expresión simbólica (objeto) `S`, escrita en forma exacta, a su forma numérica (número en coma flotante de doble precisión). A continuación se muestran dos ejemplos. En el primero se convierte `p` (del ejemplo anterior) a su forma numérica. En el segundo ejemplo se crea un objeto simbólico y después se convierte a su forma numérica.

```
>> pN = double(p)

pN =
   4.0476

>> y = sym(10)*cos(5*pi/6)

y =
-5*3^(1/2)

>> yN = double(y)

yN =
-8.6603
```

p se convierte a su forma numérica (y se asigna a pN).

Se crea la expresión simbólica y.

Se visualiza el valor exacto de y.

y se convierte a su forma numérica (y se asigna a yN).

- Un objeto simbólico puede ser una expresión simbólica compuesta por variables que no han sido previamente creadas como objetos simbólicos. Por ejemplo, la expresión cuadrática $ax^2 + bx + c$ se puede crear como un objeto simbólico llamado `f` utilizando el comando `sym`:

```
>> f = sym('a*x^2 + b*x + c')

f =
a*x^2+b*x+c
```

Es importante comprender que, en este caso, las variables a, b, c y x incluidas en el objeto no existen individualmente como objetos simbólicos independientes (la expresión completa es de por sí un solo objeto). Esto significa que es imposible realizar operaciones simbólicas asociadas con cada variable individual del objeto. Por ejemplo, no sería posible derivar `f` con respecto a x. Este caso difiere del ejemplo visto en la Sección 11.1.2, donde las variables se crearon primero de forma individual, como objetos simbólicos, y luego se utilizaron en la expresión cuadrática.

- Las expresiones simbólicas se pueden utilizar para crear nuevas expresiones simbólicas. Para hacer esto sólo hay que utilizar el nombre de la expresión existente en la nueva expresión que se va a crear. Por ejemplo:

```
>> syms x y

>> SA = x + y, SB = x - y

SA =
x+y

SB =
x-y

>> F = SA^2/SB^3 + x^2

F =
(x+y)^2/(x-y)^3+x^2
```

Se definen las variables simbólicas x e y

Se crean dos expresiones simbólicas, SA y SB.

$SA = x + y$

$SB = x - y$

Se crea una nueva expresión simbólica, F, utilizando SA y SB.

$$F = (SA^2)/(SB^2) + x^2 = \frac{(x+y)^2}{(x-y)^3} + x^2$$

11.1.3 *El comando* `findsym` *y las variables simbólicas por defecto*

El comando `findsym` se utiliza para encontrar las variables simbólicas utilizadas dentro una expresión simbólica. El formato de este comando es:

`findsym(S)` o `findsym(S,n)`

El comando `findsym` muestra, en orden alfabético, el nombre de todas las variables simbólicas (separadas por coma) que se encuentran en la expresión `S`. El comando `findsym(S,n)` muestra las `n` primeras variables simbólicas de la expresión `S`, en el orden en que aparecen dentro de la propia expresión. En las variables simbólicas compuestas por una letra, el orden por defecto comienza con *x*, y la secuencia según las siguientes letras en función de su proximidad a *x*. Si hay dos letras que son igualmente próximas a *x*, se seleccionará la que está después de *x*, en orden alfabético, como primera letra (*y* antes que *w*, y *z* antes que *v*). La variable simbólica por defecto de una expresión simbólica es la primera letra en el orden por defecto de la propia expresión. La variable simbólica por defecto de una expresión `S` se puede identificar tecleando `findsym(S,1)`. Por ejemplo:

```
>> syms x h w y d t          Se definen las variables simbólicas: x, h, w, y, d y t.
>> S = h*x^2 + d*y^2 + t*w^2  Se crea la expresión simbólica S.
S =
h*x^2+d*y^2+t*w^2
>> findsym(S)                Se utiliza el comando findsym(S).
ans =
d, h, t, w, x, y             Las variables simbólicas se visualizan en orden alfabético.
>> findsym(S,5)              Se utiliza el comando findsym(S,n) con n = 5.
ans =
x, y, w, t, h                Se visualizan cinco variables simbólicas en su orden por defecto.
>> findsym(S,1)              Se utiliza el comando finsym(S,n) con n = 1.
ans =
x                            Se visualiza la variable simbólica por defecto.
```

11.2 Modificación de expresiones simbólicas

Las expresiones simbólicas pueden ser creadas por el usuario, o también por MATLAB como resultado de otras operaciones simbólicas. Las expresiones creadas por MATLAB podrían no estar en su forma más simple, o en la forma en que el usuario desea. La forma de una expresión simbólica existente se puede modificar agrupando los términos de igual potencia, desarrollando los productos, sacando factores comunes, utilizando identidades matemáticas y trigonométricas, y mediante otras muchas operaciones. Las siguientes subsecciones describen varios de los comandos que pueden ser utilizados para cambiar o modificar la forma de una expresión simbólica existente.

11.2.1 *Los comandos* `collect`, `expand` *y* `factor`

Los comandos `collect`, `expand` y `factor` se pueden utilizar para realizar operaciones matemáticas varias:

El comando `collect`:

El comando `collect` agrupa los términos de igual potencia que se encuentran dentro de una expresión. En la nueva expresión, los términos se ordenarán en orden decreciente de potencia. Este comando tiene la siguiente sintaxis:

`collect(S)` o `collect(S,nombre_variable)`

donde `S` es la expresión. La forma `collect(S)` funciona mejor cuando la expresión tiene una única variable simbólica. Si la expresión tiene más de una variable, MATLAB agrupará los términos de una variable primero, después de una segunda variable, y así sucesivamente. El orden de las variables lo determina MATLAB. El usuario puede especificar la primera variable utilizando la otra sintaxis del comando; `collect(S,nombre_variable)`. Por ejemplo:

```
>> syms x y
>> S = (x^2 + x - exp(x))*(x + 3)
S =
(x^2+x-exp(x))*(x+3)
>> F = collect(S)
F =
x^3+4*x^2+(-exp(x)+3)*x-3*exp(x)
>> T = (2*x^2 + y^2)*(x + y^2 + 3)
T =
(2*x^2+y^2)*(x+y^2+3)
>> G = collect(T)
G =
2*x^3+(2*y^2+6)*x^2+y^2*x+y^2*(y^2+3)
>> H = collect(T,y)
H =
y^4+(2*x^2+x+3)*y^2+2*x^2*(x+3)
```

Se definen las variables simbólicas x e y.

Se crea la expresión simbólica S: $(x^2 + x - e^x)(x + 3)$.

Se utiliza el comando `collect(S)`.

MATLAB devuelve la expresión: $x^3 + 4x^2 + (-e^x + 3)x - 3e^x$.

Se define la expresión simbólica T: $(2x^2 + y^2) + (x + y^2 + 3)$.

Se utiliza el comando `collect(T)`.

MATLAB devuelve la expresión: $2x^3 + (2y^2 + 6)x^2 + xy^2 + y^2(y^2 + 3)$.

Se utiliza el comando `collect(T,y)`.

MATLAB devuelve el resultado: $y^4 + (2x^2 + x + 3)y^2 + 2x^2(x + 3)$.

Como puede comprobarse, cuando se utiliza el comando `collect(T)` la expresión se reformatea y se escribe de forma que las potencias de x se reducen. Por el contrario, cuando se utiliza el comando `collect(T,y)`, la expresión se reformatea y se escribe con el objetivo de reducir las potencias de y.

El comando `expand`*:*

El comando `expand` desarrolla expresiones (aplica la propiedad distributiva). Primero resuelve los productos de términos que incluyen sumas (de al menos un término), y posteriormente utiliza identidades trigonométricas, exponenciales y logarítmicas para desarrollar los términos correspondientes que incluyen dichas sumas. La sintaxis del comando es:

```
expand(S)
```

donde S es la expresión simbólica. He aquí dos ejemplos:

```
>> syms a x y
>> S = (x + 5)*(x - a)*(x + 4)
S =
(x+5)*(x-a)*(x+4)
>> T = expand(S)
T =
x^3+9*x^2-x^2*a-9*x*a+20*x-20*a
>> expand(sin(x - y))
ans =
sin(x)*cos(y)-cos(x)*sin(y)
```

Se definen las variables simbólicas a, x e y.

Se crea la expresión simbólica S: $(x + 5)(x - a)(x + 4)$.

Se utiliza el comando `expand`.

MATLAB devuelve la expresión: $x3 + 9x^2 - ax^2 - 9ax + 20x - 20a$.

Se utiliza el comando `expand` para desarrollar sen$(x - y)$.

MATLAB utiliza una identidad trigonométrica para desarrollar la expresión.

El comando `factor`*:*

El comando `factor` cambia (factoriza) una expresión en forma de polinomio y da como salida otra expresión simbólica compuesta por productos de polinomios de grado menor. La sintaxis de este comando es:

```
factor(S)
```

donde `S` es la expresión simbólica. Ejemplo:

```
>> syms x
>> S = x^3 + 4*x^2 - 11*x - 30
S =
x^3+4*x^2-11*x-30
>> factor(S)
ans =
(x+2)*(x-3)*(x+5)
```

Se define la variable simbólica x.

Se crea la expresión simbólica S: $x^3 + 4x^2 - 11x - 30$.

Se utiliza el comando `factor`.

MATLAB devuelve la expresión: $(x + 2)(x - 3)(x + 5)$.

11.2.2 *Los comandos* `simplify` *y* `simple`

Los comandos `simplify` y `simple` se utilizan para simplificar una expresión simbólica. El comando `simplify` utiliza reglas de simplificación predefinidas para generar una expresión mucho más simple que la original. El comando `simple` genera una expresión con el menor número de caracteres, aunque no existe garantía de que esta forma sea la más sencilla.

El comando `simplify`:

El comando `simplify` utiliza operaciones matemáticas (suma, multiplicación, reglas sobre fracciones, potencias, logaritmos, etc.), así como identidades funcionales y trigonométricas para generar la forma más simple posible de una expresión. El formato de este comando es el siguiente:

```
simplify(S)
```

donde `S` es el nombre de la expresión que se va a simplificar. También se puede introducir directamente la expresión como argumento del propio comando.

Veamos dos ejemplos:

```
>> syms x y
```
Se definen las variables simbólicas x e y.

```
>> S = x*(x*(x - 8) + 10) - 5
```
Se crea la expresión simbólica S: $x(x(x-8)+10)-5$.

```
S =
x*(x*(x-8)+10)-5
SA = simplify(S)
```
Se utiliza el comando `simplify` para simplificar S.

```
SA =
x^3-8*x^2+10*x-5
```
MATLAB simplifica la expresión y devuelve: $x^3 - 8x^2 + 10x - 5$.

```
>> simplify((x + y)/1/x + 1/y))
```
Se simplifica la expresión: $(x+y)/\left(\frac{1}{x}+\frac{1}{y}\right)$.

```
ans =
x*y
```
MATLAB simplifica la expresión y devuelve: xy.

El comando `simple`:

El comando `simple` encuentra una forma de la expresión de entrada con el menor número de caracteres. En muchos casos, esta forma es también la más simple. Cuando el comando se ejecuta, MATLAB crea varias formas de la expresión aplicando los comandos `collect`, `expand`, `factor`, `simplify`, así como otras funciones de simplificación no tratadas en este libro. Seguidamente, MATLAB devuelve la expresión más corta a partir de la original. El comando `simple` tiene la siguiente sintaxis:

```
F = simple(S)
```
Se asigna a F la forma más corta de S.

```
simple(S)
```
Se visualiza toda la secuencia de simplificación. La más corta se asigna a `ans`.

```
[F how] = simple(S)
```
Se asigna a F la forma más de corta de S. El nombre (cadena) del método de simplificación se asigna al parámetro de salida `how`.

La diferencia entre las distintas sintaxis de este mismo comando es, básicamente, la salida generada. Veamos unos ejemplos de utilización:

```
>> syms x
>> S = (x^3 - 4*x^2 + 16*x)/(x^3 + 64)
S =
(x^3-4*x^2+16*x)/(x^3+64)
>> F = simple(S)
F =
x(x+4)
>> [G how] = simple(S)
G =
x/(x+4)
how =
factor
```

Se define la variable simbólica x.

Se crea la expresión simbólica S: $\frac{x^3 - 4x^2 + 16x}{x^3 + 64}$.

Se simplifica S mediante el comando simple.

Se asigna a F la forma simple de S: $x / (x + 4)$.

Se utiliza el comando simple de nuevo sobre S, ahora con otra sintaxis.

Se asigna a G la forma simple de S: $x / (x + 4)$.

Se asigna 'factor' a la variable how, lo que indica que se ha utilizado la función factor para obtener en G la simplificación de le expresión simbólica S.

La forma `simple(S)` no se ha incluido como ejemplo debido a que la salida que proporciona el sistema es bastante larga. Para este caso, MATLAB visualizaría diez intentos diferentes para asignar la forma más simple de S a `ans`. El lector puede intentar ejecutar este comando y ver la salida generada.

11.2.3 *El Comando* `pretty`

El comando `pretty` visualiza una expresión simbólica de forma parecida a como ésta suele escribirse realmente (forma algebraica). Este comando tiene la forma:

`pretty(S)`

Por ejemplo:

```
>> syms a b c x
>> S = sqrt(a*x^2 + b*x + c)
S =
sqrt(a*x^2+b*x+c)
>> pretty(S)
                    2          1/2
          (a x + b x + c)
```

Se definen las variables simbólicas a, b, c y x.

Se crea la expresión simbólica S: $\sqrt{ax^2 + bx + c}$.

Visualiza la expresión simbólica anterior de forma parecida a como se escribe.

11.3 Resolución de ecuaciones algebraicas

Mediante la función `solve` se puede resolver una sola ecuación algebraica, en función de una de sus variables (si tiene más de una) o un sistema de ecuaciones de más de una variable.

Resolución de una sola ecuación:

Una ecuación algebraica puede tener una o más variables simbólicas. Si la ecuación tiene una variable, la solución será numérica. Si la ecuación tiene más de una variable simbólica, la solución se puede calcular para cualquiera de las variables en función de las restantes. La solución se obtiene utilizando el comando `solve`, cuya sintaxis es:

`h = solve(eq)` o `h = solve(eq,var)`

- El argumento `eq` puede ser el nombre de una expresión simbólica existente o una expresión tecleada directamente dentro del comando. Cuando se utiliza una expresión simbólica existente `S` para `eq`, o cuando la expresión `eq` no contiene el símbolo =, MATLAB resuelve la ecuación para `eq` = 0.
- Se pueden resolver ecuaciones de la forma $f(x) = g(x)$ tecleando la ecuación (incluido el signo =) como cadena dentro del argumento `eq`.
- Si la ecuación que se va a resolver tiene más de una variable, el comando `solve(eq)` opera en función de la variable simbólica por defecto (ver Sección 11.1.3). Para obtener la solución en función de cualquier otra variable, se utiliza la sintaxis `solve(eq,var)`, donde `var` es el nombre de la variable a partir de la cual se calcula la ecuación.
- Si el usuario teclea: `solve(eq)`, la solución se asignará a la variable `ans`.
- Si la ecuación tiene más de una solución, la salida `h` será un vector columna simbólico, donde cada elemento representará una solución. Los elementos de este vector son objetos simbólicos. Cuando se visualiza un array de objetos simbólicos, cada fila se encerrará entre corchetes (véanse los ejemplos que se muestran a continuación).

Veamos algunos ejemplos de utilización del comando `solve`.

```
>> syms a b x y z
```
Se definen las variables simbólicas `a`, `b`, `x`, `y` y `z`.

```
>> h = solve(exp(2*z) - 5)
```
Se utiliza el comando `solve` para resolver $e^{2z} - 5 = 0$.

```
h =
1/2*log(5)
```
La solución se asigna a `h`.

```
>> S = x^2 - x - 6
```
Se crea la expresión simbólica `S`: $x^2 - x - 6$.

```
S =
x^2-x-6
>> k = solve(S)
```
Se utiliza el comando `solve` para resolver: $x^2 - x - 6 = 0$.

```
k =
[-2]
[ 3]
```
La ecuación tiene dos soluciones. Éstas se asignan a `k`, que es un vector columna de objetos simbólicos.

```
>> solve('cos(2*y) + 3*sin(y) = 2')

ans =
[ 1/2*pi]
[ 1/6*pi]
[ 5/6*pi]

>> T = a*x^2 + 5*b*x + 20

T =
a*x^2+5*b*x+20

>> solve(T)

ans =
[ 1/2/a*(-5*b+(25*b^2-80*a)^(1/2))]
[ 1/2/a*(-5*b-(25*b^2-80*a)^(1/2))]

>> M = solve(T,a)

M =
-5*(b*x+4)/x^2
```

Se utiliza el comando `solve` para resolver: $\cos(2y) + 3\text{sen}(y) = 2$. La ecuación se teclea como cadena dentro del comando.

La solución se asigna a `ans`.

Se crea la expresión simbólica T: $ax^2 + 5bx + 20$.

Se utiliza `solve` para resolver $T = 0$.

La ecuación $T = 0$ se resuelve para la variable x, que es la variable por defecto.

Se utiliza el comando `solve` para resolver $T = 0$ respecto a la variable a.

Solución a la ecuación $T = 0$ para a.

- La ecuación que se va a resolver también se puede teclear en forma de cadena, sin incluir las variables en la ecuación como objetos simbólicos. Sin embargo, si la solución contiene variables (cuando la ecuación tiene más de una variable), las variables no podrán existir como variables simbólicas independientes. Por ejemplo:

```
>> ts = solve('4*t*h^2 + 20*t - 5*g')

ts =
5/4*g/(h^2+5)
```

Se intenta resolver la expresión: $4th^2 + 20t - 5g$. Las variables t, h y g no han sido creadas como objetos simbólicos.

MATLAB resuelve la ecuación $4th^2 + 20t - 5g$ para t.

La ecuación anterior también se puede resolver para otra variable distinta, por ejemplo:

```
>> gs = solve('4*t*h^2 + 20*t - 5*g','g')

gs =
4/5*t*h^2+4*t
```

Resolución de sistemas de ecuaciones:

El comando `solve` se puede usar también para resolver sistemas de ecuaciones. Si el número de ecuaciones y el de variables coinciden, la solución será numérica. Si el número de variables es mayor que el número de ecuaciones, la solución será simbólica para las variables deseadas, en función de las restantes. Un sistema de ecuaciones (dependiendo del tipo de ecuaciones) puede tener una o varias soluciones. Si el sistema tiene una solución, cada variable para la cual se resuelve el sistema tendrá un valor numérico (o expresión). Si el sistema tiene más de una solución, cada una de las variables podrá tener más de un valor.

El formato del comando `solve` para resolver un sistema de n ecuaciones es el siguiente:

```
salida = solve(eq1,eq2,...,eqn)
```

o

```
salida = solve(eq1,eq2,...,eqn,var1,var2,...,varn)
```

- Los argumentos `eq1, eq2,...,eqn` son las ecuaciones que hay que resolver. Cada argumento puede ser el nombre de una expresión simbólica existente, o una expresión introducida como cadena. Cuando se introduce una expresión simbólica `S` existente, la ecuación será `S = 0`. Cuando se introduce una expresión, como cadena, que no incluye el signo =, la ecuación será la expresión igualada a cero. Las ecuaciones que contengan el signo = se deben introducir obligatoriamente como cadenas.
- En el primer formato visto, si el número n de ecuaciones es igual al número de variables de la ecuación, MATLAB devuelve una solución numérica para todas las variables. Si el número de variables es mayor que el número n de ecuaciones, MATLAB proporciona una solución para las n variables en función del resto de ellas. Estas variables serán seleccionadas por MATLAB según el orden por defecto (ver Sección 11.1.3).
- Cuando el número de variables es mayor que el número n de ecuaciones, el usuario puede seleccionar las variables para las cuales se calculará la solución del sistema. Para hacer esto hay que usar la segunda sintaxis del comando `solve`, en donde se deben introducir los nombres de las variables implicadas: `var1, var2,...,varn`.

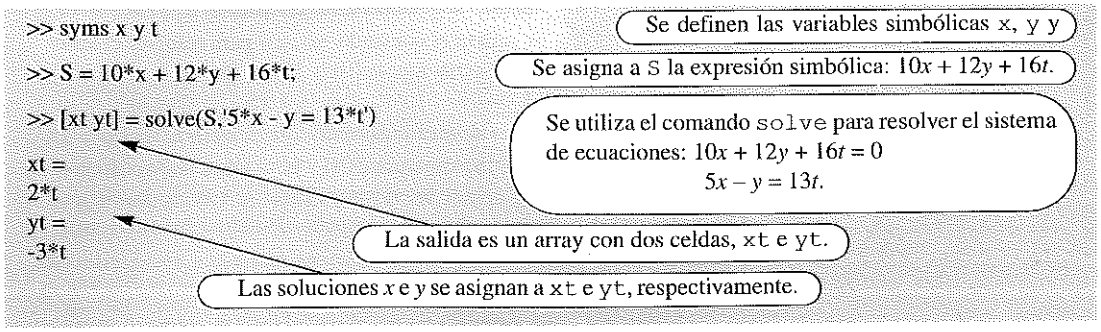
El `salida` de la función `solve` será la solución al sistema de ecuaciones. Esta salida puede tener dos formatos distintos: un array de celdas o una estructura MATLAB. Un array de celdas es un array en el cual cada elemento puede ser a su vez otro array. Una estructura es un array en el cual se accede a cada elemento o campo de forma textual, por el nombre del campo. Los campos de una estructura pueden ser arrays de diferentes tamaños y tipos. Los arrays de celdas y las estructuras no se tratarán en este libro, aunque se dará una pequeña explicación, seguidamente, para que el usuario pueda utilizar e interpretar los resultados del comando `solve`.

Cuando se tiene un array de celdas como salida del comando `solve`, el comando tendrá la siguiente forma (en el caso, por ejemplo, de un sistema de tres ecuaciones):

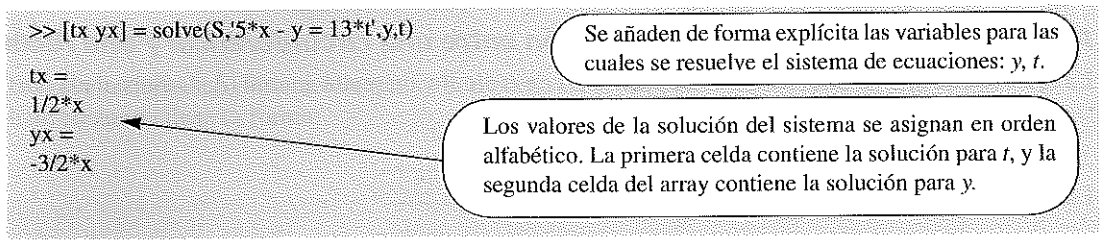
```
[varA,varB,varC] = solve(eq1,eq2,eq3)
```

- Cuando se ejecuta este comando, la solución se asignará a las variables `varA`, `varB`, `varC`, y éstas serán visualizadas con la solución calculada para cada una de ellas. Cada una de estas variables contendrá uno o varios valores (representados en forma de vector columna), dependiendo de si el sistema de ecuaciones tiene una o más soluciones.
- El usuario puede dar cualquier nombre a las variables `varA, varB, varC`. En realidad MATLAB asigna los valores de la solución del sistema en orden alfabético. Por ejemplo, si las variables para las cuales se ha resuelto el sistema son x, u y t, la solución para t se almacenará en `varA`, la solución para u se almacenará en `varB`, y la solución para x se almacenará en `varC`.

A continuación se muestra un ejemplo de uso del comando `solve` cuando se utilizan array de celdas en la salida del comando:



En el ejemplo anterior MATLAB resuelve el sistema de dos ecuaciones para x e y en función de t, ya que x e y son las dos primeras variables en el orden por defecto. El sistema, sin embargo, se puede resolver para otras variables distintas. Veamos a continuación un ejemplo donde el sistema se resuelve para y y t en función de x (utilizando la segunda forma del comando `solve`):



Cuando se utiliza una estructura como salida del comando `solve` esté tendrá la siguiente forma (en el caso de un sistema de tres ecuaciones):

```
AN = solve(eq1,eq2,eq3)
```

- `AN` es el nombre de la estructura.
- Cuando el comando se ejecuta, la solución se asigna a `AN`. MATLAB visualiza el nombre de la estructura y los nombres de los campos de la estructura, que se corresponden con los nombres de las variables que dan la solución al sistema de ecuaciones. El tamaño y el tipo de cada campo se visualizan a continuación del nombre del campo. Sin embargo, el contenido de cada campo, que se corresponde con la solución para una variable, no se visualiza.
- Para visualizar el contenido de un campo (la solución para una variable), el usuario debe teclear el nombre del campo, de la forma: `nombre_estructura.nombre_campo` (ver el ejemplo que se muestra a continuación).

A continuación se verá un ejemplo de resolución de un sistema de ecuaciones basado en un ejemplo anterior, utilizando esta vez una estructura para la salida.

```
>> syms x y t
>> S = 10*x + 12*y + 16*t
>> AN = solve(S,'5*x - y = 13*t')
AN =
    x: [1x1 sym]
    y: [1x1 sym]
>> AN.x
ans =
2*t
>> AN.y
ans =
-3*t
```

Se utiliza el comando `solve` para resolver el sistema de ecuaciones: $10x + 12y + 16t = 0$, $5x - y = 13t$.

MATLAB visualiza el nombre de la estructura (`AN`) y el nombre de los campos (`x` e `y`), junto con el tamaño y el tipo de los mismos. Éstos se corresponden con los nombres de las variables para las que se han resuelto el sistema de ecuaciones.

Acceso al campo `x` de la estructura.

Se visualiza el contenido del campo (solución para x).

Acceso al campo `y` de la estructura.

Se visualiza el contenido del campo (solución para y).

El Problema de ejemplo 11.1 muestra la resolución de un sistema de ecuaciones con dos soluciones.

Problema de ejemplo 11.1: Intersección entre una circunferencia y una recta

La ecuación de una circunferencia en el plano x-y con radio R y un punto central (2, 4) viene dada por:

$$(x - 2)^2 + (y - 4)^2 = R^2.$$

La ecuación de una recta en el plano viene dada por:

$$y = \frac{x}{2} + 1.$$

Calcular las coordenadas de los puntos (en función de R) donde la recta y la circunferencia se intersecan.

Solución

La solución se obtiene resolviendo el sistema de dos ecuaciones para x e y en función de R. Para mostrar la diferencia entre utilizar un array de celdas y una estructura, el sistema de ecuaciones se resolverá dos veces utilizando la función `solve`. La primera solución se corresponde con una salida de tipo array de celdas:

```
>> syms x y R
>> [xc, yc] = solve('(x - 2)^2 + (y - 4)^2 = R^2','y = x/2 + 1')
xc =
[14/5+2/5*(-16+5*R^2)^(1/2)]
[14/5-2/5*(-16+5*R^2)^(1/2)]
yc =
[12/5+1/5*(-16+5*R^2)^(1/2)]
[12/5-1/5*(-16+5*R^2)^(1/2)]
```

Las dos ecuaciones se teclean en el comando `solve`.

La salida `[xc, yc]` se corresponde con un array de celdas.

La salida es un array de dos celdas, llamadas `xc` e `yc`. Cada celda contiene dos soluciones, en forma de vector simbólico tipo columna.

En esta segunda solución se utiliza una estructura para la salida:

```
>> COORD = solve('(x - 2)^2 + (y - 4)^2 = R^2','y = x/2 + 1')
COORD =
   x: [2x1 sym]
   y: [2x1 sym]
COORD.x
ans =
[14/5+2/5*(-16+5*R^2)^(1/2)]
[14/5-2/5*(-16+5*R^2)^(1/2)]
COORD.y
ans =
[12/5+1/5*(-16+5*R^2)^(1/2)]
[12/5-1/5*(-16+5*R^2)^(1/2)]
```

La salida se introduce en una estructura llamada COORD.

La estructura COORD tiene dos campos (x e y). Cada campo es un vector simbólico de 2 × 1.

Se teclea el nombre del campo x.

Se visualiza el contenido del campo (solución para x).

Se teclea el nombre del campo y.

Se visualiza el contenido del campo (solución para y).

11.4 Derivación

El cálculo simbólico diferencial, o cálculo de derivadas, se lleva a cabo utilizando el comando `diff`. Este comando tiene la forma:

`diff(S)` o `diff(S,var)`

- `S` puede ser una expresión simbólica completa o el nombre de una expresión simbólica existente.
- En el comando `diff(S)`, si la expresión contiene una sola variable simbólica, el cálculo se llevará a cabo con respecto a esa variable. Si la expresión contiene más de una variable, el cálculo se llevará a cabo con respecto a la variable simbólica por defecto (ver Sección 11.13).
- El comando `diff(S,var)` se utiliza para calcular la derivada de una expresión con más de una variable simbólica. Este cálculo se lleva a cabo con respecto a la variable `var` indicada como parámetro.
- Las segundas derivadas (y otras de mayor orden) se pueden calcular mediante las sintaxis `diff(S,n)` o `diff(S,var,n)`, donde `n` es un número positivo. Por ejemplo, $n = 2$ para calcular la segunda derivada, $n = 3$ para calcular la tercera derivada, y así sucesivamente.

Veamos algunos ejemplos.

```
>> syms x y t
>> S = exp(x^4);
>> diff(S)
ans =
4*x^3*exp(x^4)
```

Se definen las variables simbólicas x, y y t.

Se crea la expresión simbólica: e^{x^4}.

Se calcula la derivada de S.

La solución es: $4x^3e^{x^4}$.

```
>> diff((1 - 4*x)^3)

ans =
-12*(1-4*x)^2

>> R = 5*y^2*cos(3*t)

>> diff(R)

ans =
10*y*cos(3*t)

>> diff(R,t)

ans =
-15*y^2*sin(3*t)

>> diff(S,2)

ans =
12*x^2*exp(x^4)+16*x^6*exp(x^4)
```

Se calcula la derivada de $(1-4x)^3$.

La solución es $-12(1-4x)^2$

Se crea la expresión simbólica $5y^2\cos(3t)$.

Se calcula la derivada de R.

MATLAB deriva R con respecto a y (variable simbólica por defecto). Se visualiza el resultado: $10y\cos(3t)$.

Se calcula la derivada de R con respecto a t.

La solución es $-15y^2\text{sen}(3t)$.

Se calcula la segunda derivada de S.

La solución es $12x^2e^{x^4} + 16x^6e^{x^4}$.

- Es posible utilizar también el comando `diff` introduciendo la ecuación que se va a derivar en forma de cadena, aunque se recuerda, al igual que en los comandos vistos anteriormente, que las variables simbólicas contenidas en la cadena se utilizan sólo para el cálculo, y no podrán ser utilizadas posteriormente como variables simbólicas independientes.

11.5 Integración

La integración simbólica, o cálculo de primitivas, se lleva a cabo utilizando el comando `int`. Este comando permite calcular integrales indefinidas y definidas. Para el cálculo de integrales indefinidas se utilizan las sintaxis:

`int(S)` o `int(S,var)`

- S puede ser una expresión simbólica o el nombre de una expresión simbólica existente.
- En el comando `int(S)`, si la expresión contiene una sola variable simbólica, el cálculo se llevará a cabo con respecto a esa variable. Si la expresión contiene más de una variable, la integración se realizará con respecto a la variable simbólica por defecto (ver Sección 11.1.3).
- En el comando `int(S,var)`, la integración se llevará a cabo con respecto a la variable `var`. Esta sintaxis se utiliza para integrar expresiones con más de una variable simbólica.

He aquí algunos ejemplos:

```
>> syms x y t

>> S = 2*cos(x) - 6*x

>> int(S)

ans =
2*sin(x)-3*x^2
```

Se definen las variables simbólicas x, y, t.

Se crea la expresión simbólica S: $2\cos(x) - 6x$.

Se calcula la integral de S.

Se visualiza la solución: $2\text{sen}(x) - 3x^2$.

```
>> int(x*sin(x))                                  Se calcula la integral de xsen(x).

ans =
sin(x)-x*cos(x)                                   Se visualiza la solución: sen(x) – xcos(x).

>> R = 5*y^2*cos(4*t);                            Se crea la expresión simbólica R: 5y^2cos(4t).

>> int(R)                                         Se calcula la integral de R.

ans =                                             MATLAB integra R con respecto a y (variable simbólica
5/3*y^3*cos(4*t)                                  por defecto). Se visualiza la solución: 5y^3cos(4t)/3.

>> int(R,t)                                       Se integra R con respecto a t.

ans =                                             Se visualiza la solución: 5y^2sen(4t)/4.
5/4*y^2*sin(4*t)
```

Para calcular integrales definidas se utilizan estas otras formas del comando `int`:

`int(S,a,b)` o `int(S,var,a,b)`

- donde a y b son los límites de integración. Estos límites pueden ser números o variables simbólicas.

Por ejemplo, el código necesario para calcular la integral definida $\int_0^{\pi}(\text{sen } y + 5y^2)dy$ mediante MATLAB es el siguiente:

```
>> syms y
>> int(sin(y) - 5*y^2,0,pi)
ans =
2-5/3*pi^3
```

- El comando `int` también permite teclear, en forma de cadena, la expresión que se va a integrar, sin haber declarado previamente las variables simbólicas que la expresión contiene. Sin embargo, estas variables no podrán ser utilizadas como variables simbólicas independientes en otras expresiones.
- La integración es a menudo un proceso difícil. Es posible que a veces ni siquiera exista una respuesta completamente cerrada para un problema dado. MATLAB podría no encontrar solución a la integración de una ecuación. Cuando esto sucede, MATLAB retorna `int(S)` junto con un mensaje del tipo: `Explicit integral could not be found` (no se puede encontrar una integral explícita).

11.6 Resolución de ecuaciones diferenciales ordinarias

Las ecuaciones diferenciales ordinarias (ODE: del inglés Ordinary Differential Equation) se pueden resolver simbólicamente con el comando `dsolve`. Este comando se utiliza para resolver una sola ecuación o bien un sistema de ecuaciones diferenciales. En este libro se tratará el caso de la resolución de una sola ecuación. En el Capítulo 10 se describe cómo resolver ODEs de primer orden de forma numérica. Para las explicaciones que se darán a continuación, se supone que el lector está familiarizado con las ecuaciones diferenciales. El propósito de esta sección es mostrar el uso de MATLAB para resolver dichas ecuaciones.

Una ecuación diferencial ordinaria de primer orden es una ecuación que contiene la derivada de la variable dependiente. Si t es la variable independiente, e y es la variable independiente, la ODE de primer orden se puede escribir de la forma:

$$\frac{dy}{dt} = f(t, y)$$

Una ODE de segundo orden contiene la segunda derivada de la variable dependiente (también puede contener su primera derivada). Su forma general es:

$$\frac{d^2y}{dt^2} = f(t, y, \frac{dy}{dt})$$

La solución para estas ecuaciones es una función $y = f(t)$ que satisface la ecuación. La solución puede ser general o particular. Una solución general contiene constantes. En la solución particular las constantes deben tener valores numéricos específicos tal que la solución satisfaga las condiciones iniciales específicas o condiciones en la frontera.

El comando `dsolve` se puede utilizar para obtener una solución general o, cuando la condición inicial o la condición en la frontera se han especificado, obtener una solución particular.

Solución general:

Para obtener una solución general, el comando `dsolve` debe tener la forma:

`dsolve('eq')` o `dsolve('eq','var')`

- `eq` es la ecuación a resolver. Debe ser introducida como cadena, incluso si la variables que contiene son objetos simbólicos.
- Las variables en la ecuación no tienen por qué haber sido creadas previamente como objetos simbólicos. De hecho, en la solución, las variables tampoco serán objetos simbólicos.
- Se puede usar cualquier letra (en mayúsculas o minúsculas) como variable dependiente, excepto D.
- En el comando `dsolve('eq')` MATLAB toma t como la variable independiente por defecto.
- En el comando `dsolve('eq','var')` el usuario puede indicar la variable independiente tecleándola en forma de cadena dentro el parámetro `var`.
- Cuando se teclea la ecuación, la letra `D` indica diferencial o derivada. Si y es la variable dependiente y t es la independiente, `Dy` significa $\frac{dy}{dt}$. Por ejemplo, la ecuación $\frac{dy}{dt} + 3y = 100$ se introduciría como: `'Dy + 3*y = 100'`.
- La segunda derivada se representa como `D2`, la tercera derivada como `D3`, y así sucesivamente. Por ejemplo, la ecuación $\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 5y = \text{sen } t$ se introduciría como: `'D2y + 3*Dy + 5*y = sin(t)'`.
- No es necesario haber definido previamente como variables simbólicas las variables que se introducen en el comando `dsolve`.
- En la solución que ofrece MATLAB, se utilizan `C1`, `C2`, `C3`, etc. como constantes de integración.

Por ejemplo, veamos el código correspondiente a una solución general para la ODE de primer orden $\frac{dy}{dt} = 4t + 2y$:

```
>> dsolve('Dy = 4*t + 2*y')

ans =
-2*t-1+exp(2*t)*C1
```

Se visualiza la solución: $y = -2t - 1 + C_1e^{2t}$.

A continuación veamos una solución general para la ODE de segundo orden $\frac{d^2x}{dt^2} + 2\frac{dx}{dt} + x = 0$:

```
>> dsolve('D2x + 2*Dx + x = 0')

ans =
C1*exp(-t)+C2*exp(-t)*t
```

Se visualiza la solución: $x = C_1e^{-1} + C_2te^{-t}$.

Los ejemplos siguientes muestran la resolución de ecuaciones diferenciales que contienen variables simbólicas, además de variables dependientes e independientes.

```
>> dsolve('Ds = a*x^2')

ans =
a*x^2*t+C1
>> dsolve('Ds = a*x^2','x')

ans =
1/3*a*x^3+C1
>> dsolve('Ds = a*x^2','a')

ans =
1/2*a^2*x^2+C1
```

La variable independiente es t (por defecto). MATLAB resuelve la ecuación: $\frac{ds}{dt} = ax^2$.

Se visualiza la solución: $s = ax^2t + C_1$.

La variable independiente se define como x. MATLAB resuelve la ecuación: $\frac{ds}{dx} = ax^2$.

Se visualiza la solución: $s = \frac{1}{3}ax^3 + C_1$.

La variable independiente se define como a. MATLAB resuelve la ecuación: $\frac{ds}{da} = ax^2$.

Se visualiza la solución: $s = \frac{1}{2}a^2x^2 + C_1$.

Solución particular:

Si las condiciones iniciales o en la frontera han sido especificadas, se puede obtener una solución particular para una ODE. Una ecuación de primer orden requiere de una condición, una de segundo orden

requiere dos condiciones, y así sucesivamente. Para obtener una solución particular se deben utilizar las siguientes sintaxis del comando `dsolve`:

Para ODEs de primer orden: `dsolve('eq','cond1','var')`

Para ODEs de orden superior: `dsolve('eq','cond1','cond2',...,'var')`

- Para resolver ecuaciones de orden superior se necesitan condiciones adicionales en la frontera que deben ser introducidas por parámetro. Si el número de condiciones es menor que el orden de la ecuación, MATLAB devolverá una solución que incluye las constantes de integración (`C1`, `C2`, `C3`, etc.).
- Las condiciones en la frontera se teclean en forma de cadena, de la siguiente manera:

Forma matemática	**Forma MATLAB**
$y(a) = A$	`'y(a)=A'`
$y'(\alpha) = A$	`'Dy(a)=A'`
$y''(a) = A$	`'D2y(a)=A'`

- El argumento `'var'` es opcional. Se puede utilizar para definir la variable independiente en la ecuación. Si no se introduce ningún valor se toma por defecto t.

Por ejemplo, veamos el código para resolver la ODE de primer orden $\frac{dy}{dt} + 4y = 60$, con una condición inicial $y(0) = 5$:

```
>> dsolve('Dy + 4*y = 60','y(0) = 5')
ans =
15-10*exp(-*t)
```

Se visualiza la solución: $y = 15 - 10e^{-4t}$.

A continuación se muestra, en este otro ejemplo, el código MATLAB para resolver la ODE de segundo orden $\frac{d^2y}{dt^2} - 2\frac{dy}{dt} + 2y = 0$, $y(0) = 1$, $\left.\frac{dy}{dt}\right|_{t=0} = 0$:

```
>> dsolve('D2y - 2*Dy + 2*y = 0','y(0) = 1','Dy(0) = 0')
ans =
-exp(t)*sin(t)+exp(t)*cos(t)
>> factor(ans)
ans =
-exp(t)*(sint(t)-cos(t))
```

Se visualiza la solución: $y = -e^t\text{sen}(t) + e^t\cos(t)$.

La solución se puede simplificar utilizando el comando `factor`.

Se visualiza la solución: $y = -e^t(\text{sen}(t) - \cos(t))$.

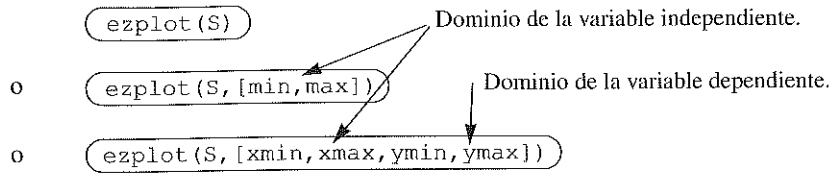
En el Problema de ejemplo 11.5 se pueden encontrar más casos de resolución de ecuaciones diferenciales.

Si MATLAB no puede encontrar una solución para la ecuación, éste devuelve un objeto simbólico vacío, seguido del mensaje: `'Warning: explicit solution could not be found'` (Aviso: podría no existir una solución explícita).

11.7 Representación gráfica de expresiones simbólicas

Cuando se trabaja con determinados problemas, puede ser necesario representar gráficamente una expresión simbólica. Esto se puede hacer mediante el comando `ezplot`. Si tenemos una expresión simbólica `S` que contiene una variable `var`, MATLAB considerará dicha expresión como una función *S*(*var*), de forma que `ezplot` creará un gráfico de *S*(*var*) frente a *var*. Para expresiones que contienen dos variables simbólicas, `var1` y `var2`, MATLAB considerará la expresión como una función de la forma *S*(*var*1, *var*2) = 0, de forma que el comando `ezplot` generará un gráfico de una variable frente a la otra.

Para realizar un gráfico de una expresión simbólica que contiene una o dos variables, se utilizará el comando `ezplot` de la forma:



- `S` es la expresión simbólica que se representará gráficamente. Se puede introducir directamente o bien puede ser el nombre de una expresión simbólica existente.
- Es posible introducir la expresión que se representará en forma de cadena, sin tener que crear los objetos simbólicos que contenga dicha expresión previamente.
- Si `S` contiene una sola variable simbólica, se creará un gráfico de *S*(*var*) frente a *var*, con los valores de *var* (variable independiente) en las abscisas (eje horizontal), y los valores de *S*(*var*) en las ordenadas (eje vertical).
- Si la expresión simbólica `S` contiene dos variables simbólicas, `var1` y `var2`, la expresión se interpretará como una función de la forma *S*(*var*1, *var*2) = 0. De esta forma, MATLAB creará un gráfico de una variable frente a la otra. La primera variable (en orden alfabético) se tomará como variable independiente. Por ejemplo, si las variables contenidas en *S* son *x* e *y*, entonces *x* será la variable independiente y será representada en el eje de abscisas, e *y* será la variable dependiente que se representará en el eje de ordenadas. Además, si las variables contenidas en *S* son *u* y *v*, *u* será la variable independiente, y *v* será en este caso la variable dependiente.
- En el comando `ezplot(S)`, si `S` tiene una variable (*S*(*var*)), el gráfico se representará sobre el dominio $-2\pi < var < 2\pi$ (dominio por defecto), y el rango será seleccionado directamente por MATLAB. Si `S` posee dos variable (*S*(*var*1, *var*2)), el gráfico se representará sobre el dominio $-2\pi < var1 < 2\pi$ y $-2\pi < var2 < 2\pi$.
- En el comando `ezplot(S,[min,max])`, el dominio para la variable independiente se define a través de `min` y `max`: *min* < *var* < *max*, siendo el rango seleccionado directamente por MATLAB.
- En el comando `ezplot(S,[xmin,xmax,ymin,ymax])`, el dominio de la variable independiente queda definido por `xmin` y `xmax`, y el de la dependiente por `ymin` e `ymax`.

El comando `ezplot` también se puede usar para representar una función que viene expresada en forma paramétrica. En este caso, se necesitan dos expresiones simbólicas, `S1` y `S2`, donde cada expresión vendrá dada en términos de la misma variable simbólica (parámetro independiente). Por ejemplo,

para representar y frente a x, donde $x = x(t)$ e $y = y(t)$, se utilizarán las siguientes formas del comando `ezplot`:

```
ezplot(S1,S2)
```

Dominio de la variable independiente.

o

```
explot(S1,S2,[min,max])
```

- `S1` y `S2` son expresiones simbólicas que contienen la misma variable simbólica, es decir, el mismo parámetro independiente. `S1` y `S2` pueden ser expresiones simbólicas introducidas directamente, o también los nombres de expresiones simbólicas existentes.
- El comando crea una representación gráfica de *S2*(*var*) frente a *S1*(*var*). La expresión simbólica introducida en primer lugar (`S1`, según el ejemplo anterior) se utilizará como eje horizontal, y la segunda (`S2`) como eje vertical.
- En el comando `ezplot(S1,S2)`, el dominio de la variable independiente es $0 < var < 2\pi$ (domino por defecto).
- En el comando `ezplot(S1,S2,[min,max])`, el dominio de la variable independiente queda definido a través de `min` y `max`: $min < var < max$.

Comentarios adicionales:

Una vez que se crea el gráfico, éste puede tener el formato deseado, al igual que los gráficos vistos en secciones anteriores (`plot` y `fplot`). Para ello se pueden utilizar los comandos ya vistos, o también utilizar el Editor Gráfico (ver Sección 5.4). Cuando se crea el gráfico, la expresión se visualiza automáticamente en la parte superior de éste. MATLAB posee funciones adicionales para dibujar gráficos polares bidimensionales, y también para dibujar gráficos en tres dimensiones. Para más información, el lector puede acudir a la ayuda de MATLAB, buscando "Symbolic Math Toolbox" (librería sobre matemática simbólica).

En la tabla 11.1 se muestran distintos ejemplos de la utilización del comando `ezplot`.

Tabla 11.1: Gráficos con el comando `ezplot`.

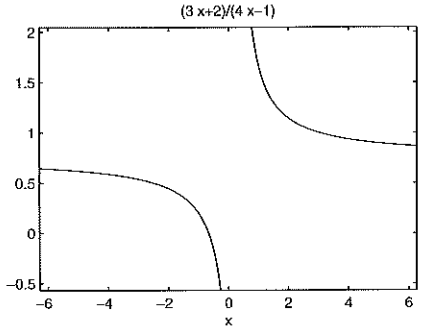
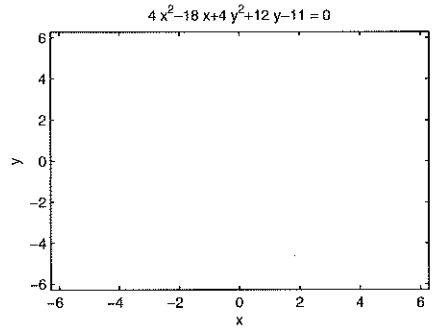
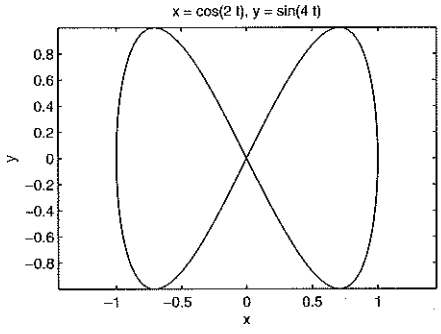
Comandos	Gráfico generado
>> syms x >> S = (3*x + 2)/(4*x - 1) S = (3*x+2)/(4*x-1) >> ezplot(S)	

Tabla 11.1: Gráficos con el comando `ezplot`. (*Continuación*)

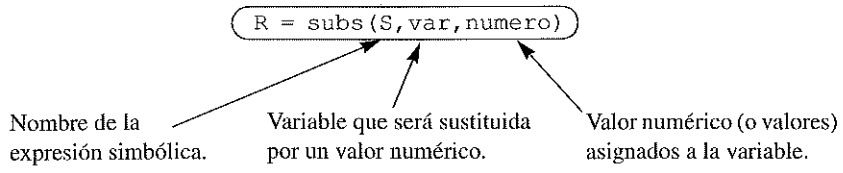
Comandos	Gráfico generado
>> syms x y >> S = 4*x^2 -18*x + 4*y^2 + 12*y - 11 >> S = 4*x^2-18*x+4*y^2+12*y-11 >> ezplot(S)	
>> syms t >> x = cos(2*t) x = cos(2*t) >> y = sin(4*t) y = sin(4*t) >> ezplot(x,y)	

11.8 Cálculo numérico mediante expresiones simbólicas

Una vez que MATLAB, a través de distintos cálculos, o el propio usuario ha creado una expresión simbólica, puede ser interesante asignar o sustituir las variables simbólicas por números, con el objetivo de realizar cálculos y obtener valores numéricos a partir de estas expresiones. Esto se puede hacer mediante el comando `subs`. El comando `subs` admite distintas formas, y puede ser utilizado de diferentes maneras. A continuación se describirán algunas de las formas que son más fáciles de usar, y más adecuadas quizás para la mayoría de las aplicaciones. En una de esta formas, la variable o las variables que serán sustituidas por un valor numérico, y el propio valor numérico, se teclean dentro del comando `subs`. En la otra forma, a la variable o a las variables se les asigna un valor numérico mediante otros comandos, y después se sustituyen las variables dentro de la expresión.

Sustitución de una variable simbólica por un valor numérico:

Una variable se puede sustituir por un valor numérico cuando la expresión simbólica tiene una o más variables simbólicas. En este caso, el comando `subs` tendrá la forma:



- `numero` puede ser un número (escalar), o un array con varios elementos (un vector o una matriz).
- El valor de `S` se calcula para cada valor de `numero`, y el resultado se asigna a `R`, que tendrá el mismo tamaño que `numero` (escalar, vector o matriz).
- Si `S` contiene una sola variable simbólica, la salida `R` será numérica. Si `S` contiene varias variables y sólo se sustituye una de ellas, entonces la salida `R` será una expresión simbólica.

Veamos un ejemplo con una expresión que contiene una variable simbólica:

```
>> syms x                                  Se define la variable simbólica x.
>> S = 0.8*x^3 + 4*exp(0.5*x)              Se crea la expresión simbólica S: 0,8x^3 + 4e^(0,5x).
S =
4/5*x^3+4*exp(1/2*x)
>> SD = diff(S)                            Se calcula la derivada de S.
SD =                                       Se asigna a SD la solución: 12x^2/5 + 2e^(0,5x).
12/5*x^2+2*exp(1/2*x)
>> subs(SD,x,2)                            Se asigna o sustituye x = 2 en SD.
ans =                                      Se visualiza el valor de SD.
   15.0366
>> SDU = subs(SD,x,[2:0.5:4])              Se asigna o sustituye x = [2, 2,5, 3, 3,5, 4] (vector) en SD.
SDU =
15.0366   21.9807   30.5634   40.9092   53.1781
                                           Se visualizan, en forma de vector, los valores de SD (asignados a SDU) para cada valor del vector x.
```

Como puede verse en el ejemplo anterior, cuando se calcula el valor numérico de la expresión simbólica la respuesta es numérica (se muestra el resultado del cálculo). Veamos otro ejemplo donde se opera con una expresión con más de una variable simbólica:

```
>> syms a g t v                            Se definen a, g, t y v como variables simbólicas.
>> Y = v^2*exp(a*t)/g                      Se crea la expresión simbólica v^2e^(at)/g y se asigna a Y.
Y =
v^2*exp(a*t)/g
>> subs(Y,t,2)                             Se utiliza el comando subs para sustituir t = 2 en SD.
ans =                                      Se visualiza la solución : v^2e^(2a)/g.
v^2*exp(a*2)/g
>> Yt = subs(Y,t,[2:4])                    Se utiliza el comando subs para sustituir el vector t = [2, 3, 4] en Y.
Yt =
[v^2*exp(2*a)/g, v^2*exp(3*a)/g, v^2*exp(4*a)/g]
                                           Se visualiza la solución, consistente en un vector con expresiones simbólicas como elementos, en función de cada uno de los valores de t introducidos.
```

Sustitución de dos o más variables simbólicas por un valor numérico:

Para sustituir más de una variable simbólica por uno o varios valores numéricos, se utiliza otra forma, ligeramente distinta, del comando `subs`:

```
R = subs(S,{var1,var2,…,varN},{numero1,numero2,...,numeroN})
```

- Nombre de la expresión simbólica.
- Variables simbólicas utilizadas en la sustitución.
- Valores numéricos que se asignarán o sustituirán en las variables simbólicas.

- Las variables `var1`, `var2`,..., `varN` son variables simbólicas de la expresión `S` que se pretenden sustituir por números. Las variables deben teclearse en forma de array de celdas (entre llaves { }). Los arrays de celdas son arrays donde cada elemento puede ser un número o una cadena.
- Los números `numero1`, `numero2`, ..., `numeroN` son los que se sustituirán en las variables simbólicas comentadas anteriormente. Deben teclearse también en forma de array de celdas, entre llaves { }. Los números pueden ser escalares, vectores o matrices. El primer elemento, `numero1`, se sustituirá en la variable `var1`. El segundo, `numero2` , se sustituirá en `var2`, y así sucesivamente.
- Si todos los números sustituidos son escalares, la salida será un número o una expresión simbólica (si alguna variable simbólica no se ha sustituido).
- Si al menos una variable se sustituye con un array de números, las operaciones se llevarán a cabo elemento a elemento, siendo la salida un array de números o expresiones simbólicas. Se debe tener en cuenta que los cálculos se llevarán a cabo elemento a elemento, incluso aunque la expresión `S` no se teclee utilizando la notación elemento a elemento propiamente dicha. Esto significa que todos los arrays sustituidos en las correspondientes variables simbólicas deben ser del mismo tamaño.
- Es posible sustituir arrays (del mismo tamaño) en algunas variables y escalares en otras. En este caso, MATLAB construye arrays del mismo tamaño para los escalares con los valores internos repetidos. El objetivo de esto es poder llevar a cabo las correspondientes operaciones elemento a elemento y producir un resultado en forma de array.

Veamos un ejemplo de sustitución de números sobre varias variables simbólicas:

```
>> syms a b c e x                          Se definen las variables simbólicas a, b, c, e y x.
>> S = a*x^e + b*x + c                     Se crea la expresión simbólica S: ax^e + bx + c.
S =
a*x^e+b*x+c
>> subs(S,{a,b,c,e,x},{5,4,-20,2,3})       Se sustituyen en S todas las variables por valores numéricos.
                                  Array de celdas   Array de celdas
   ans =
   37                                          Se visualiza el valor de S.
>> T = subs(S,{a,b,c},{6,5,7})             Se sustituyen en S las variables a, b y c.
```

```
T =
6*x^e+5*x+7                                   El resultado será una expresión, con variables simbólicas x y e.

>> R = subs(S,{b,c,e},{[2 4 6],9,[1 3 5]})   Se sustituyen en S un escalar para c y vectores para b y e.

R =
[a*x+2*x+9, a*x^3+4*x+9, a*x^5+6*x+9]         El resultado es un vector de expresiones simbólicas.

W = subs(S,{a,b,c,e,x},{[4 2 0],[2 4 6],[2 2 2],[1 3 5],[3 2 1]})   Se sustituyen en S todas las variables.

W =
20   26   8                                    El resultado es un vector de valores numéricos.
```

Un segundo método utilizado para asignar números en las variables de una expresión simbólica es asignar los valores numéricos a las variables, y después utilizar el comando `subs` con la siguiente sintaxis:

```
R = subs(S)  ← Nombre de la expresión simbólica.
```

Una vez que las variables simbólicas han sido redefinidas como numéricas, éstas no se podrán utilizar más como variables simbólicas. Veamos un ejemplo de uso del comando `subs`:

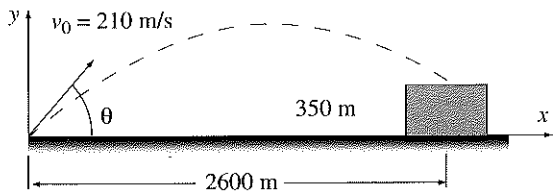
```
>> syms A c m x y                             Se definen A, c, m, x e y como variables simbólicas.

>> S = A*cos(m*x) + c*y                       Se crea la expresión simbólica A: Acos(mx) + cy.

S=
A*cos(m*x)+c*y

>> A = 10; m = 0.5; c = 3;                    Se asignan valores numéricos a las variables A, m y c.

>> subs(S)                                    Se utiliza el comando subs con la expresión S.

ans =
10*cos(1/2*x)+3*y                             Se sustituyen en S los valores numéricos de las variables A, m y c.

>> x = linspace(0,2*pi,4)                     Se asignan valores numéricos (vector) a la variable x.

>> T = subs(S)                                Se utiliza el comando subs con la expresión S.

T =
[10+3*y, 5+3*y, -5+3*y, -10+3*y]              Se sustituyen los valores numéricos de A, m, c y x.
                                              El resultado es un vector de expresiones simbólicas.
```

11.9 Ejemplos de aplicaciones MATLAB

Problema de ejemplo 11.2: Ángulo de lanzamiento de un proyectil

Un proyectil se lanza con una velocidad de 210 m/s y un ángulo θ. La trayectoria se calcula para que el cohete alcance un objetivo localizado a 2600 metros de distancia, y 350 metros por encima del punto de disparo.

a) Deducir la ecuación necesaria para calcular el ángulo θ para el cual el proyectil alcanza su objetivo.
b) Utilizar MATLAB para resolver la ecuación del apartado *a*.
c) Para el ángulo obtenido en el apartado *b*, utilizar el comando `ezplot` con el objetivo de representar gráficamente la trayectoria del proyectil.



Solución

a) El movimiento de un proyectil se puede modelar considerando las componentes vertical y horizontal. La velocidad inicial v_0 se puede descomponer en sus componentes horizontal y vertical, de la forma:

$$v_{0x} = v_0 \cos(\theta) \quad \text{y} \quad v_{0y} = v_0 \operatorname{sen}(\theta)$$

La velocidad es constante en la dirección horizontal, de forma que la posición del proyectil en función del tiempo vendrá dada por:

$$x = v_{0x}t$$

Si se sustituye $x = 2600$ m (la distancia horizontal que recorre el proyectil) y v_{0x} por $210\cos(\theta)$, entonces despejando t, se obtiene:

$$t = \frac{2600}{210\cos(\theta)}$$

En la dirección vertical, la posición del proyectil vendrá dada por:

$$y = v_{0y}t - \frac{1}{2}gt^2$$

Sustituyendo la coordenada vertical por $y = 350$ m, $210\operatorname{sen}(\theta)$ en v_{0y}, $g = 9{,}81$, y el valor anterior de t, tendremos:

$$350 = 210\operatorname{sen}(\theta)\frac{2600}{210\cos(\theta)} - \frac{1}{2}9{,}81\left(\frac{2600}{210\cos(\theta)}\right)^2$$

o

$$350 = 2600\tan(\theta) - \frac{1}{2}9{,}81\left(\frac{2600}{210\cos(\theta)}\right)^2$$

La solución a esta ecuación da el ángulo θ en el que se lanzó el proyectil.

b) Para resolver este apartado se puede utilizar el comando `solve` (en la Ventana de Comandos):

```
>> syms theta
>> Angle = solve('2600*tan(theta) - 0.5*9.81*(2600/(210*cos(theta)))^2 = 350')
```

```
Angle =
[ -2.68233984655772202568477886290671
[ -1.89623815635237707014882980262351
[ 1.24535449723741616831381358065601
[ .45925280703207121277786452037279]

>> Angle1 = Angle(3)*180/pi

Angle1 =
224.16380950273491029648644451808/pi

>> Angle1 = double(Angle1)

Angle1 =
  71.3536

>> Angle2 = Angle(4)*180/pi

Angle2 =
82.665505265772818300015613667102/pi

>> Angle2 = double(Angle2)

Angle2 =
26.3132
```

MATLAB visualiza cuatro soluciones. Las dos soluciones positivas son las más relevantes para este problema.

Se convierte la tercera de las soluciones obtenidas de radianes a grados.

MATLAB visualiza el resultado en forma de objeto simbólico, en términos de π.

Se utiliza el comando `double` para obtener el valor numérico de `Angle1`.

Se convierte la cuarta de las soluciones obtenidas de radianes a grados.

MATLAB visualiza el resultado en forma de objeto simbólico, en términos de π.

Se utiliza el comando `double` para obtener el valor numérico de `Angle2`.

c) De las soluciones calculadas en el apartado *b* se obtienen dos posibles ángulos y, por lo tanto, dos posibles trayectorias. Para dibujar la trayectoria, las coordenadas *x* e *y* del proyectil han de ser expresadas en términos de *t* (representación paramétrica):

$$x = v_0 \cos(\theta)t \quad \text{e} \quad y = v_0 \operatorname{sen}(\theta)t - \frac{1}{2}gt^2$$

El dominio de *t* va desde $t = 0$ hasta $t = \dfrac{2600}{210\cos(\theta)}$.

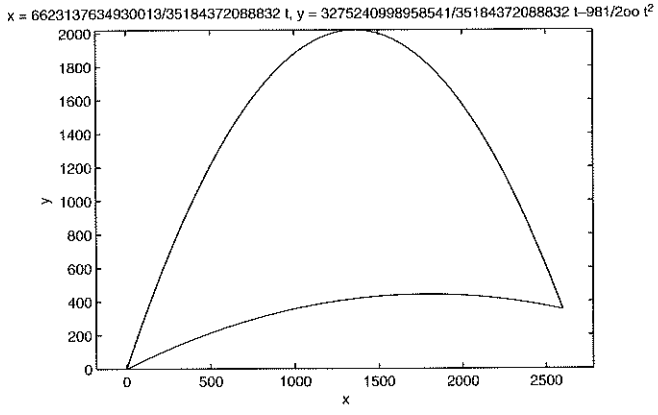
Estas ecuaciones se pueden utilizar en el comando `ezplot` para realizar el gráfico de la trayectoria del proyectil. Para ello se ha creado el siguiente script:

```
xmax = 2600; v0= 210; g = 9.81;
theta1 = 1.24535; theta2 = 0.45925;
t1 = xmax/(v0*cos(theta1));
t2 = xmax/(v0*cos(theta2));
syms t
X1 = v0*cos(theta1)*t;
X2 = v0*cos(theta2)*t;
```

Se asignan a `theta1` y `theta2` las soluciones calculadas en el apartado *b*.

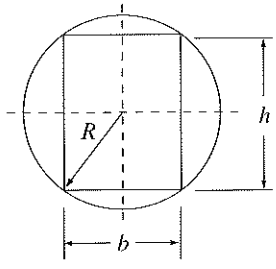
```
Y1 = v0*sin(theta1)*t - 0.5*g*t^2;
Y2 = v0*sin(theta2)*t - 0.5*g*t^2;
ezplot(X1,Y1,[0,t1])          Se dibuja una de las trayectorias.
hold on
ezplot(X2,Y2,[0,t2])          Se dibuja la otra trayectoria.
hold of
```

Cuando el programa se ejecuta, se genera el siguiente gráfico:



Problema de ejemplo 11.3: Resistencia a la flexión de una viga

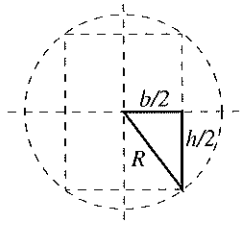
La resistencia a la flexión de una viga rectangular de ancho b y de altura h es proporcional al momento de inercia I, definido por $I = \frac{1}{12}bh^3$. Una viga rectangular se obtiene a partir de un tronco cilíndrico de radio r. Calcular los valores de b y h (en función R) que hacen que la viga tenga un momento de inercia I máximo.



Solución

El problema se resolverá a través de los siguientes pasos:

1) Se encontrará una ecuación que relacione R, h y b.
2) Se encontrará una expresión para I en función de h.
3) Se realizará la derivada de I con respecto a h.
4) La derivada se igualará a 0 para calcular h.
5) Se calculará el ancho b correspondiente.



El primer paso se llevará a cabo observando el triángulo de la figura adjunta. La relación entre R, h y b viene dada por el Teorema de Pitágoras: $\left(\frac{b}{2}\right)^2 + \left(\frac{h}{2}\right)^2 = R^2$. Despejando b de la ecuación se obtiene $b = \sqrt{4R^2 - h^2}$. El resto de los pasos se llevarán a cabo utilizando MATLAB:

```
>> syms b h R
>> b = sqrt(4*R^2 - h^2);
>> I = b*h^3/12
I =
1/12*(4*R^2-h^2)^(1/2)*h^3
>> ID = diff(I,h)
ID =
-1/12/(4*R^2-h^2)^(1/2)*h^4+1/4*(4*R^2-h^2)^(1/2)*h^2
>> hs = solve(ID,h)
hs =
[       0]
[       0]
[ 3^(1/2)*R]
[-3^(1/2)*R]
>> bs = subs(b,hs(3))
bs =
(R^2)^(1/2)
>> bss = simple(bs)
bss =
R
```

Se crea la expresión simbólica *b*.

Paso 2: Se crea la expresión simbólica *I*.

MATLAB sustituye *b* en *I*.

Paso 3: Se utiliza el comando `diff` para derivar *I* respecto a *h*.

Se visualiza la derivada de *I*.

Paso 4: Utilización del comando `solve` para resolver la ecuación $ID = 0$ en función de *h*. El resultado se asigna a *hs*.

MATLAB visualiza cuatro soluciones. La solución positiva $\sqrt{3}R$ es la importante para este problema.

Paso 5: Se utiliza el comando `subs` para calcular *b*, sustituyendo *h* por la solución en la expresión *b*.

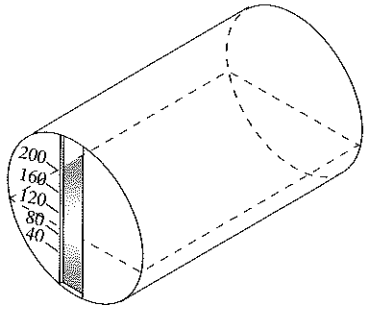
Se visualiza la respuesta para *b*. (La respuesta es *R*, pero MATLAB visualiza $(R^2)^{1/2}$.)

Se utiliza el comando `simple` para simplificar `bs`.

Se visualiza la respuesta simplificada de *b*.

Problema de ejemplo 11.4: Nivel de combustible de un depósito

Un depósito horizontal, como el que se muestra en la figura adjunta, se utiliza para almacenar combustible. El depósito tiene un diámetro de 6 m y una longitud de 8 m. La cantidad de combustible almacenada en el depósito se puede estimar mirando el nivel del líquido a través de una ventanilla de cristal situada en la parte frontal del depósito. En esta ventanilla viene marcada una escala con niveles de combustible correspondientes a 40, 80, 120, 160 y 200 mil litros. Calcular la posición vertical (medida desde el suelo) de las líneas de la escala de medición utilizada.

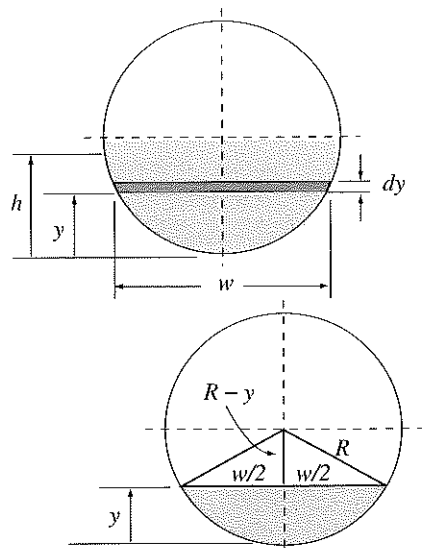


Solución

La relación entre el nivel del combustible y su volumen se puede expresar en forma de integral definida. Una vez efectuada la integración, se obtendrá una ecuación que representará el volumen del depósito en función de la altura del combustible. La altura para un volumen determinado se puede calcular resolviendo posteriormente la ecuación.

El volumen V de combustible se puede calcular multiplicando el área A de la sección transversal del combustible (el área sombreada de la figura) por la longitud del depósito L. El área de la sección transversal se puede calcular mediante la integral:

$$V = AL = L\int_0^h w\,dy$$



El ancho w de la superficie superior del combustible se puede expresar en función de y. Mediante el triángulo mostrado en la figura adjunta se puede establecer una relación entre las variables y, w y R a partir de la expresión:

$$\left(\frac{w}{2}\right)^2 + (R-y)^2 = R^2$$

Despejando w de esta ecuación se obtiene:

$$w = 2\sqrt{R^2 - (R-y)^2}$$

El volumen del combustible para una altura h se puede calcular sustituyendo w en la integral de la ecuación del volumen y llevando a cabo la integración. El resultado es una ecuación que nos proporcionará el volumen V en función de h. El valor de h para un V dado se obtendrá resolviendo dicha ecuación. A partir de los datos proporcionados en el problema, el valor de h debe ser calculado para los volúmenes 40, 80, 120, 160 y 200 mil litros. La solución se obtiene mediante MATLAB, a partir del siguiente fichero script:

```
R = 3; L = 8;
syms w y h
w = 2*sqrt(R^2 - (R - y)^2)
S = L*w
V = int(S,y,0,h)

Vscale = [40:40:200]
```

Se crea la expresión simbólica para *w*.

Se crea la expresión simbólica que se va a integrar.

Se utiliza el comando `int` para integrar `S`, desde 0 hasta *h*. El resultado nos da *V* en función de *h*.

Se crea un vector con los valores de *V*.

```
for i = 1:5
   Veq = V - Vscale(i);
   h_ans(i) = solve(Veq);
end

h_scale = double(h_ans)
```

En cada paso del bucle se calcula *h* para cada uno de los valores de *V*.

Se crea una ecuación para *h*.

Se resuelve la ecuación anterior.

`h_ans` es un vector (simbólico con números) con los valores de *h* que se corresponden con los valores de *V* en el vector `Vscale`.

Se utiliza el comando `double` para obtener valores numéricos a partir de los elementos de `h_ans`.

Cuando se ejecuta el script, la salida que se visualizará en la Ventana de Comandos será la siguiente:

```
>> w =
2*(6*y - y^2)^(1/2)

S =
16*(6*y-y^2)^(1/2)

V =
8*(6*h-h^2)^(1/2)*h-24*(6*h-h^2)^(1/2)+72*asin(-1+1/3*h)+36*pi

Vscale =
   40   80   120   160   200

h_scale =
   1.3972   2.3042   3.1439   3.9957   4.9608
```

Se visualiza la expresión simbólica *w*.

Se visualiza la expresión `S` que será integrada.

Se muestra el resultado de la integración: *V* en función de *h*.

Se muestran los valores de la escala de volúmenes de combustible *V*.

Se visualiza la posición de las líneas de la escala de medida.

En la solución mostrada, la unidad de longitud es el metro, por lo que la unidad para el volumen es el metro cúbico (1 m^3 = 1000 litros).

Problema de ejemplo 11.5: Cantidad de medicamento en el cuerpo

La cantidad de medicamento, *M*, presente en el organismo de una persona depende de la velocidad con que el organismo lo metaboliza, y del régimen de administración del medicamento a esa persona. La velocidad con que el medicamento se metaboliza es proporcional a la cantidad presente en el organismo. Esto se puede expresar con una ecuación diferencial para *M*, de la forma:

$$\frac{dM}{dt} = -kM + p$$

donde *k* es una constante de proporcionalidad y *p* es el régimen con que se administra a la persona.

a) Calcular *k* si la vida media del medicamento es de 3 horas.

b) Un paciente es ingresado en un hospital y se le administra el medicamento a razón de 50 mg por hora (inicialmente no existe medicamento en el organismo del paciente). Obtener una expresión para M en función del tiempo.

c) Representar M en función del tiempo para las primeras 24 horas.

Solución

a) La constante de proporcionalidad se puede calcular considerando el caso en el cual la medicación es metabolizada por el cuerpo y no se administra más medicamento. En ese caso, la ecuación diferencial será:

$$\frac{dM}{dt} = -kM$$

Esta ecuación diferencial se puede resolver con la condición inicial: $M = M_0$ en $t = 0$:

```
>> syms M M0 k t
>> Mt = dsolve('DM = -k*M','M(0) = M0')
Mt =
M0*exp(-k*t)
```

Se utiliza el comando `dsolve` para resolver: $\frac{dM}{dt} = -kM$.

Esta solución nos proporciona M en función del tiempo:

$$M(t) = M_0 e^{-kt}$$

Una vida media de tres horas significa que en $t = 3$ horas, $M(t) = \frac{1}{2}M_0$. Sustituyendo esta información en la solución se obtiene $0{,}5 = e^{-3k}$, donde la constante k se puede calcular resolviendo esta ecuación:

```
ks = solve('0.5 = exp(-k*3)')
ks =
.23104906018664843647241070715273
```

Se utiliza el comando `solve` para resolver la ecuación: $0{,}5 = e^{-3k}$,

b) Para este apartado la ecuación diferencial M tendrá la forma:

$$\frac{dM}{dt} = -kM + p$$

La constante k ya ha sido calculada en el apartado *a*, siendo $p = 50$ mg/hora. La condición inicial es que al principio no hay medicamento en el organismo de la persona, es decir, $M = 0$ en $t = 0$. La solución a esta ecuación utilizando MATLAB será:

```
>> syms p
>> Mtb = dsolve('DM = -k*M + p','M(0) = 0')
Mtb =
p/k-p/k*exp(-k*t)
```

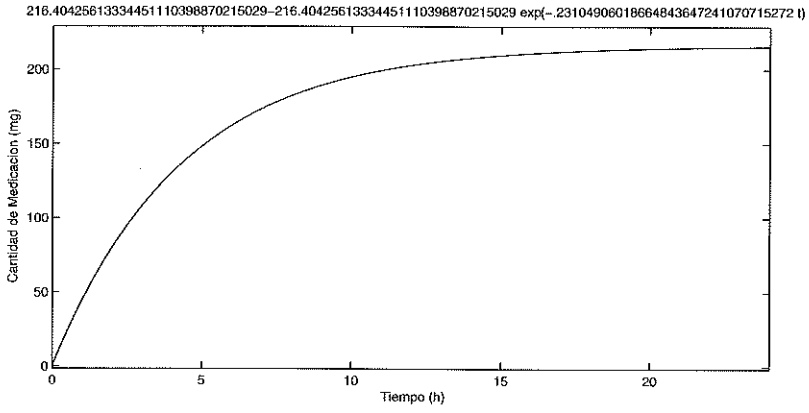
Se utiliza el comando `dsolve` para resolver la ecuación: $\frac{dM}{dt} = -kM + p$.

c) Para realizar una representación gráfica de `Mtb` en función del tiempo, para $0 \leq t \leq 24$, se utilizará el comando `ezplot` de la forma:

```
>> pgiven = 50;
>> Mtt = subs(Mtb,{p,k},{pgiven,ks})
Mtt =
216.404-216.404*exp(-.231049*t)
>> ezplot(Mtt,[0,24])
```

Se sustituyen los valores numéricos en p y k.

En la solución anterior la salida `Mtt` ha sido truncada, respecto al número total de decimales visualizados, por razones de espacio.



11.10 Problemas

1. Defina x como variable simbólica y cree dos expresiones simbólicas a partir de ella:

$$S_1 = x^3 - 9x^2 + 27x - 27 \qquad \text{y} \qquad S_2 = (x+3)^2 - x^2 - 5x - 12$$

Utilice operaciones simbólicas para simplificar las expresiones:

a) $S_1 \cdot S_2$.

b) $\dfrac{S_1}{S_2}$.

c) $S_1 + S_2$.

d) Utilice el comando `subs` para calcular el valor numérico del resultado obtenido en el apartado *c* para $x = 10$.

2. Defina y como variable simbólica y cree dos expresiones simbólicas a partir de ella:

$$S_1 = (\sqrt{y} + 2)^2 - 2(2\sqrt{y} + 1) \qquad \text{y} \qquad S_2 = y^2 - 2y + 4$$

Utilice operaciones simbólicas para simplificar las expresiones:

a) $S_1 \cdot S_2$.

b) $\dfrac{S_1}{S_2}$.

c) $S_1 + S_2$.

d) Utilice el comando `subs` para calcular el valor numérico del resultado obtenido en el apartado *c* para $x = 5$.

3. Defina w como variable simbólica y cree dos expresiones simbólicas a partir de ella:

$$S_1 = 2w^2 + 6w + 9 \qquad \text{y} \qquad S_2 = 2w^2 - 6w + 9$$

Utilice operaciones simbólicas para simplificar el producto: $S_1 \cdot S_2$.

4. Defina x como variable simbólica.

a) Demuestre que las raíces del polinomio $f(x) = x^5 + 6x^4 - 6x^3 - 64x^2 - 27x + 90$ son 1, –2, 3, –3 y –5. Utilice para ello el comando `factor`.

b) Obtenga la ecuación de un polinomio que tenga como raíces: $x = 6$, $x = -4$, $x = -1$ y $x = 2$.

5. Utilice los comandos de la Sección 11.2 para demostrar que:

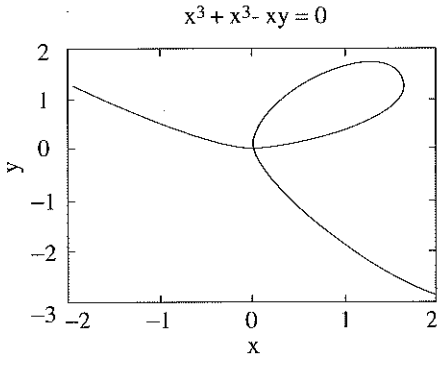
a) $\text{sen}(3x) = 3 \text{ sen } x - 4 \text{ sen}^3 x$.

b) $\text{sen } x \text{ sen } y = \dfrac{1}{2}[\cos(x - y) - \cos(s + y)]$.

c) $\cos(x + y + z) = \cos x \cos y \cos z - \text{sen } x \text{ sen } y \cos z - \text{sen } x \cos y \text{ sen } z - \cos x \text{ sen } y \text{ sen } z$

6. El *folium* de Descartes es el gráfico que se representa en la figura adjunta. Su ecuación viene dada, en forma paramétrica, por la expresión:

$$x = \frac{3t}{1 + t^3} \qquad \text{e} \qquad y = \frac{3t^2}{1 + t^3} \text{ para } t \neq -1$$

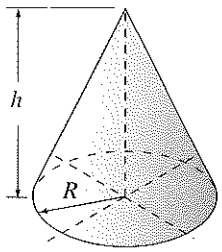


a) Utilice MATLAB para demostrar que esta ecuación se puede escribir también de la forma:

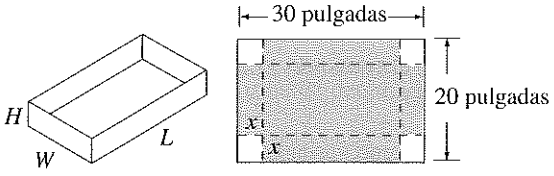
$$x^3 + y^3 = 3xy$$

b) Haga una representación gráfica de esta ecuación con el comando `ezplot`, en el dominio que se muestra en los ejes de la figura.

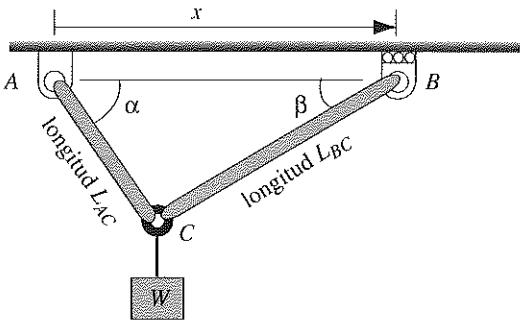
7. Un silo en forma de cono tiene una superficie de 280 m^2 y una altura h de 15 m. Calcule el radio R de su base. Escriba una ecuación para la superficie en función del radio y de la altura. Resuelva la ecuación para el radio, utilizando el comando `double`, para obtener un valor numérico.



8. Una caja se construye a partir de una pieza de cartón rectangular de dimensiones 20 por 30 pulgadas. Para ello se recortan los cuadrados sobrantes x en cada una de las esquinas, y se doblan hacia arriba los laterales tal y como muestra la figura adjunta. Calcule x de forma que el volumen de la caja sea de 1000 pulgadas^3 (hay dos posibilidades).



9. Un peso W cuelga de un anillo que a su vez esta sujeto por dos cables unidos a dos bisagras, tal y como se muestra la figura adjunta. La bisagra del punto A se encuentra fija, mientras que la bisagra del punto B se puede desplazar (sin fricción) en dirección horizontal. La fuerza en los cables F_{AC} y F_{BC} depende de la posición de la bisagra B (distancia x), y se puede calcular mediante las ecuaciones:



$$F_{AC} \cos \alpha = F_{BC} \cos\beta \text{ y } F_{AC} \operatorname{sen} \alpha + F_{BC} \operatorname{sen} \beta = W$$

a) Utilice MATLAB para obtener las expresiones de las fuerzas F_{AC} y F_{BC} en función de x, W y la longitud de los cables, L_{AC} y L_{BC}.

b) Utilice el comando `subs` para sustituir $W = 2000$ N, $L_{AC} = 0{,}3$ m y $L_{BC} = 0{,}5$ m en la expresión obtenida en el apartado *a*. Esto proporcionará la fuerza en los cables en función de la distancia x.

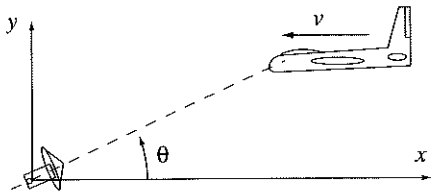
c) Utilice el comando `ezplot` para representar las fuerzas F_{AC} y F_{BC} (ambas en el mismo gráfico) en función de x, empezando en 0,4 m y acabando cerca de los 0,8 m. ¿Qué sucede cuando x se aproxima a 0,8 m?

10. La ecuación de una recta en el plano x-y viene dada por $y = 3x - 2$, y la ecuación de una elipse en el mismo plano viene dada por $16x^2 + 32x + 4y^2 - 24y = 52$.

a) Utilice el comando `ezplot` para representar la recta y la elipse en el mismo gráfico.

b) Calcule las coordenadas de intersección de la recta con la elipse.

11. Un radar de rastreo sigue automáticamente a un avión que vuela a una altitud constante de 5 km y a una velocidad constante de 540 km/h. El avión vuela a través de una ruta aérea que pasa exactamente por encima del radar. El radar comienza a rastrear el avión cuando está a 100 km de distancia de él.

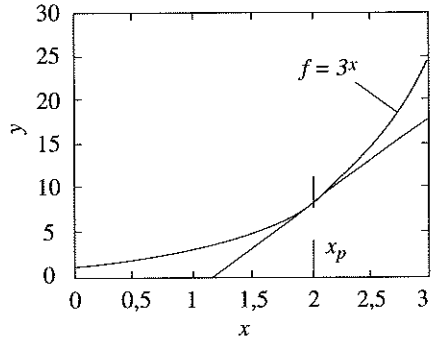


a) Obtenga una expresión para el ángulo θ de la antena el radar en función el tiempo.

b) Obtenga una expresión para la velocidad angular de la antena, $\frac{d\theta}{dt}$ en función del tiempo.

c) Haga dos gráficos en la misma ventana. Uno de θ frente al tiempo, y otro de $\frac{d\theta}{dt}$ frente al tiempo, considerando el ángulo en grados y el tiempo en minutos para $0 \leq t \leq 20$ min.

12. La función $f = 3x$ tiene una tangente en el punto x_p. Obtenga la ecuación de la recta tangente en función de x_p. La ecuación de la recta tangente tendrá la forma $y = mx + b$, donde m y b vendrán dados en función de x_p. Considere el caso en el que $x_p = 2$. Calcule el punto en el cual la tangente corta el eje x, y represente f y la tangente, ambas en el mismo gráfico. Para el dominio del gráfico, tome los valores de f tales que $0 \leq x \leq 3$. La tangente debe comenzar en el punto de intersección con el eje x y finalizar en $x = 3$.



13. Calcule la integral indefinida: $I = \int e^{2x}\sqrt{2 - e^{2x}}\, dx$.

14. La ecuación de una elipse es:

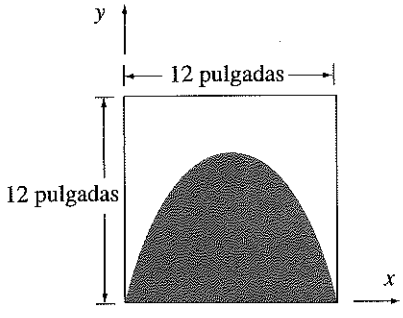
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Demuestre que el área A interior de la elipse viene dada por $A = \pi ab$.

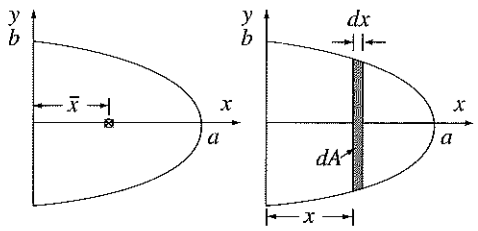
15. El diseño de una baldosa de cerámica es el que se muestra en la figura adjunta. El área sombreada está pintada de color rojo, y el resto de la baldosa es de color blanco. La línea de separación entre las áreas de color rojo y blanco se ajusta a la ecuación:

$$y = -kx^2 + 12kx$$

Calcule k para que las áreas blanca y roja coincidan.



16. Demuestre que la posición del centroide $\bar{x}$ del área semielíptica que se muestra en la figura adjunta es $\bar{x} = \frac{4a}{3\pi}$. La coordenada $\bar{x}$ se puede calcular mediante la expresión:



$$\bar{x} = \frac{\int x\, dA}{\int dA}$$

17. El valor *rms* de un voltaje AC viene dado por:

$$v_{rms} = \sqrt{\frac{1}{T}\int_0^T v^2(t')dt'}$$

donde T es el periodo de la onda.

a) Para un voltaje dado $v(t) = V\cos(\omega t)$. Demuestre que $v_{rms} = \frac{V}{\sqrt{2}}$ y que éste es independiente de ω. Tenga en cuenta que la relación entre el periodo T y la frecuencia ω es $T = \frac{2\pi}{\omega}$.

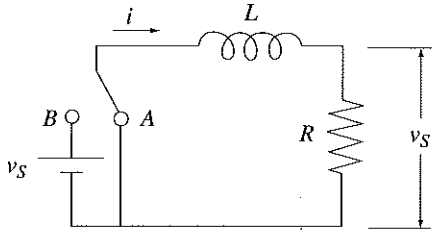
b) Un voltaje viene dado por $v(t) = 2{,}5\cos(350t) + 3\text{V}$. Calcule v_{rms}.

18. Calcule la solución a la ecuación diferencial:

$$\frac{dy}{dx} = 3y - 1{,}5yx$$

con la condición inicial $y(0) = 4$. Represente gráficamente la solución para $0 \geq x \geq 6$.

19. Una resistencia R de 0,4 Ω y un inductor L de 0,08 H se conectan en un circuito como el que se muestra en la figura adjunta. Inicialmente, el interruptor de la figura se conecta al punto A, de forma que no entra corriente al circuito. En el instante $t = 0$, el interruptor se mueve del punto A al punto B, de forma que la resistencia y el inductor quedan conectados al voltaje v_S ($v_S = 6$ V), y la corriente comienza a circular por el circuito. El interruptor permanece conectado al punto B hasta que el voltaje de la resistencia alcanza los 5 V. En ese instante (t_{BA}), el interruptor se lleva de nuevo al punto A.



La corriente i en el circuito se puede calcular a partir de las siguientes ecuaciones diferenciales:

$iR + L\frac{di}{dt} = v_S$ Desde el instante $t = 0$ hasta el instante en que el interruptor se mueve al punto A.

$iR + L\frac{di}{dt} = 0$ Desde el instante en que el interruptor se mueve de nuevo a A en adelante.

El voltaje a través de la resistencia, v_R viene dado por $v_R = iR$.

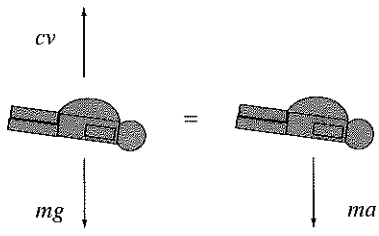
a) Obtenga una expresión para la corriente i en función de R, L, v_S y t, para $0 \le t \le t_{BA}$, resolviendo para ello la primera ecuación diferencial.

b) En la solución de i obtenida en el apartado anterior, sustituya R, L y v_S por sus valores y calcule el tiempo t_{BA} que tarda el voltaje de la resistencia en alcanzar los 5 V.

c) Obtenga una expresión para la corriente i en función de R, L y t, para $t_{BA} \le t$, resolviendo la segunda ecuación diferencial.

d) Haga dos gráficos en la misma ventana. Uno para v_R frente a t, para $0 \le t \le t_{BA}$, y el segundo para v_R frente a t, para $t_{BA} \le t \le 2t_{BA}$.

20. La velocidad de un paracaidista, mientras el paracaídas está aún cerrado, se puede modelar considerando que la resistencia del aire es proporcional a la velocidad. A partir de la segunda ley de Newton, la relación entre la masa m del paracaidista y su velocidad v viene dada por (la caída hacia abajo se considera positiva):



$$mg - cv = m\frac{dv}{dt}$$

donde c es la constante de resistencia aerodinámica y g es la constante gravitatoria $g = 9{,}81$ m/s^2.

a) Resuelva la ecuación anterior para v en función de m, g, c y t, considerando que la velocidad inicial del paracaidista es cero.

b) Se observa que 4 segundos después de que un paracaidista de 90 kg salta de un aeroplano, su velocidad es de 28 m/s. Calcule la constante c.

c) Haga un gráfico de la velocidad del paracaidista en función del tiempo, para $0 \le t \le 30$ segundos.

21. Calcule la solución general a la siguiente ecuación diferencial:

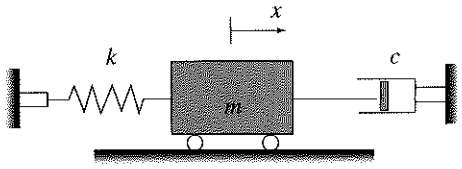
$$\frac{d^2y}{dx^2} + 3\frac{dy}{dx} - 2y = 0$$

Demuestre que la solución es correcta. Para ello obtenga la primera y segunda derivada de la solución y después sustituya el resultado en la ecuación.

22. Calcule la solución a la siguiente ecuación diferencial que satisface las condiciones iniciales dadas.

$$\frac{d^2y}{dx^2} - 4y = 5, \qquad y(0) = 0, \qquad \left.\frac{dy}{dx}\right|_{x=0} = 1$$

23. La vibración libre amortiguada se puede modelar considerando un bloque de masa m al que se le une un muelle y un dispositivo de amortiguamiento, tal y como se muestra en la figura adjunta. A partir de la segunda ley de Newton, el



desplazamiento x de la masa en función del tiempo se puede calcular resolviendo la siguiente ecuación diferencial:

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

donde k es la constante de elasticidad del muelle y c es el coeficiente de amortiguamiento del dispositivo de amortiguación. Si la masa se desplaza de su posición de equilibrio y después se libera, empezará a oscilar hacia delante y hacia atrás. La naturaleza de la oscilación depende del tamaño de la masa y de los valores de k y c.

Para un sistema como el que se muestra en la figura adjunta, $m = 10$ kg y $k = 28$ N/m. En el instante $t = 0$ la masa se desplaza hasta $x = 0{,}18$ m, y después se libera. Obtenga una expresión para el desplazamiento x y la velocidad v de la masa en función del tiempo. Considere los siguientes dos casos:

a) $c = 3$ N-s/m

b) $c = 50$ N-s/m.

Para cada caso, represente la posición x y la velocidad v frente al tiempo (los dos gráficos en la misma ventana). Para el caso *a*, considere: $0 \leq t \leq 20$ s, y para el caso *b*: $0 \leq t \leq 10$ s.

Apéndice: Resumen de caracteres, comandos y funciones

La siguientes tablas contiene los caracteres, comandos y funciones MATLAB que se han ido viendo a lo largo de este libro, ordenados por categorías.

Tabla A.1: Caracteres y operadores aritméticos.

Carácter	Descripción	Página
+	Suma.	10, 50
-	Resta.	10, 50
*	Multiplicación escalar y de arrays.	10, 51
.*	Multiplicación de arrays elemento a elemento.	58
/	División derecha.	10, 57
\	División izquierda.	10, 56
./	División derecha elemento a elemento.	58
.\	División izquierda elemento a elemento.	58
^	Exponenciación	10
.^	Exponenciación elemento a elemento.	58
:	Dos puntos; creación de vectores con elementos de igual espaciado, representación de rangos de elementos en arrays.	27, 33
=	Operador de asignación.	15
( )	Paréntesis; establece precedencia, encierra los argumentos de entrada en una función, acceso indexado a elementos de un array.	10, 32, 32, 139
[ ]	Corchetes; formación de arrays; encierra los argumentos de entrada y salida en funciones.	26, 27, 29, 139
,	Coma; separa los índices de acceso a un array y los argumentos de una función; permite separar comandos en la misma línea.	16, 32-35, 140
;	Punto y coma; evita la visualización de la ejecución de un comando; separa filas en la declaración de un array.	9, 29
'	Comilla simple; transpuesta de una matriz, creación de cadenas.	31, 42-44
...	Puntos suspensivos; continuación de una línea en la siguiente.	9
%	Porcentaje; crea comentarios, especifica un formato de salida.	9
<	Menor que.	160
>	Mayor que.	160

Tabla A.2: Operadores relacionales y lógicos.

Carácter	Descripción	Página
<=	Menor o igual que.	160
>=	Mayor o igual que.	160
==	Igual.	160
~=	No igual (diferente).	160
&	Y lógico.	163
|	O lógico.	163
~	NO.	163

Tabla A.3: Comandos de gestión de entorno de trabajo.

Carácter	Descripción	Página
`cd`	Cambia el directorio actual.	77
`clc`	Limpia la Ventana de Comandos.	9
`clear`	Borra todas las variables de memoria.	18
`clear x y z`	Borra las variables `x`, `y` y `z` de memoria.	18
`fclose`	Cierra un fichero.	90
`fopen`	Abre un fichero.	90
`global`	Declara variables globales.	142
`help`	Muestra ayuda de comandos MATLAB.	141
`lookfor`	Buscar una palabra determinada en la ayuda de MATLAB.	141
`who`	Muestra las variables actuales en memoria.	18, 38
`whos`	Muestra información de las variables actuales en memoria.	18, 38

Tabla A.4: Variables predefinidas.

Carácter	Descripción	Página
`ans`	Valor de la última expresión.	18
`eps`	La diferencia más pequeña entre dos números.	18
`i`	Raíz cuadrada de menos uno.	18
`inf`	Infinito.	18
`j`	Raíz cuadrada de menos uno (igual que i).	18
`NaN`	Del inglés Not a Number (resultado no numérico).	18
`pi`	Número π.	18

Tabla A.5: Formatos de visualización en la Ventana de Comandos.

Carácter	Descripción	Página
`format bank`	Dos dígitos decimales.	12
`format compact`	Elimina líneas en blanco.	12
`format long`	Formato de punto fijo con 14 dígitos decimales.	12
`format long e`	Notación científica con 15 dígitos decimales.	12
`format long g`	Los 15 dígitos fijos o en coma flotante de un número.	12
`format loose`	Añade líneas en blanco.	12
`format short`	Formato de punto fijo con 4 dígitos decimales.	12
`format short e`	Notación científica con 4 dígitos decimales.	12
`format short g`	Base de 5 dígitos fijos o en coma flotante.	12

Tabla A.6: Funciones matemáticas elementales.

Carácter	Descripción	Página
`abs`	Valor absoluto.	13
`exp`	Exponencial.	13
`factorial`	Función factorial.	13
`log`	Logaritmo natural o neperiano.	13
`log10`	Logaritmo en base 10.	13
`sqrt`	Raíz cuadrada.	13

Tabla A.7: Funciones trigonométricas.

Carácter	Descripción	Página	Carácter	Descripción	Página
`acos`	Arcocoseno (coseno inverso).	15	`cos`	Coseno.	14
`acot`	Arcocotangente (tangente inversa).	15	`cot`	Tangente.	14
`asin`	Arcoseno (seno inverso).	15	`sin`	Seno.	14
`atan`	Arcotangente (tangente inversa).	15	`tan`	Tangente.	14

Tabla A.8: Funciones hiperbólicas.

Carácter	Descripción	Página	Carácter	Descripción	Página
`cosh`	Coseno hiperbólico.	15	`sinh`	Seno hiperbólico.	15
`coth`	Cotangente hiperbólica.	15	`tanh`	Tangente hiperbólica.	15

Tabla A.9: Redondeo.

Carácter	Descripción	Página
`ceil`	Redondea hacia el infinito.	14
`fix`	Redondea hacia cero.	14
`floor`	Redondea hacia menos infinito.	14
`rem`	Devuelve el resto de la división de dos números.	14
`round`	Redondea al entero más próximo.	14
`sign`	Devuelve el signo.	14

Tabla A.10: Creación de arrays.

Carácter	Descripción	Página
`diag`	Crea una matriz diagonal a partir de un vector. Crea un vector a partir de la diagonal de una matriz.	39
`eye`	Crea una matriz identidad (matriz con unos en la diagonal y ceros en el resto de las posiciones).	30, 54
`linspace`	Crea un vector con espaciado constante.	28
`ones`	Crea un matriz de unos.	30
`rand`	Crea un arrays con números aleatorios.	64
`randn`	Crea un matriz con números aleatorios con distribución normal.	64
`randperm`	Crea un vector a partir de la permutación de números enteros.	64
`zeros`	Crea un array de ceros.	30

Tabla A.11: Manipulación de arrays.

Carácter	Descripción	Página
`length`	Número de elementos de un vector.	39
`reshape`	Redimensiona una matriz.	39
`size`	Tamaño de una matriz.	39

Tabla A.12: Funciones relacionadas con arrays.

Carácter	Descripción	Página
`cross`	Calcula el producto cruzado de dos vectores.	63
`det`	Calcula el determinante.	56, 63
`dot`	Calcula el producto escalar de dos vectores.	52, 63
`inv`	Calcula la inverso de una matriz.	55, 63
`max`	Retorna el valor máximo.	62
`mean`	Calcula el valor medio.	62
`median`	Calcula el valor mediano.	63
`min`	Retorna el valor mínimo.	62
`sort`	Ordena los elementos en orden ascendente.	63
`std`	Calcula la desviación estándar.	63
`sum`	Calcula la suma de elementos.	63

Tabla A.13: Entrada y salida.

Carácter	Descripción	Página
`disp`	Visualiza una salida.	82
`fprintf`	Visualiza o guarda una salida.	85-92
`input`	Pide al usuario una entrada por teclado.	80
`uiimport`	Inicializa el asistente de importación de datos.	94
`xlsread`	Importa datos en formato Excel.	93
`xlswrite`	Exporta datos en formato Excel.	94

Tabla A.14: Gráficos bidimensionales.

Carácter	Descripción	Página
`bar`	Crea un gráfico de barras verticales.	122
`barh`	Crea un gráfico de barras horizontales.	122
`fplot`	Representa gráficamente una función.	112
`hist`	Crea un histograma.	123-126
`hold off`	Finaliza `hold on`.	114
`hold on`	Mantiene un gráfico abierto para seguir añadiendo gráficas.	114
`line`	Añade gráficas a un gráfico existente.	115
`loglog`	Crea un gráfico con escala logarítmica en ambos ejes.	121
`pie`	Crea un gráfico de tarta o circular.	123
`plot`	Crea un gráfico.	106-112
`polar`	Crea un gráfico en coordenadas polares.	126
`semilogx`	Crea un gráfico con escala logarítmica en el eje *x*.	121
`semilogy`	Crea un gráfico con escala logarítmica en el eje *y*.	121
`stairs`	Crea un gráfico de escalera.	122
`stem`	Crea un gráfico de tallo o líneas verticales.	123

Tabla A.15: Gráficos tridimensionales.

Carácter	Descripción	Página
`bar3`	Crea un gráfico de barras tridimensional.	239
`contour`	Crea un gráfico de contorno bidimensional.	238
`contour3`	Crea un gráfico de contorno tridimensional.	238
`cylinder`	Dibuja un cilindro.	239
`mesh`	Crea un dibujo de malla.	235, 236
`meshc`	Crea un dibujo de malla con contorno.	237
`meshgrid`	Crea una rejilla para representar un gráfico tridimensional.	234
`meshz`	Crea un gráfico de mallas con cortinas.	237
`pie3`	Crea un gráfico de tarta tridimensional.	240
`plot3`	Crea un gráfico en tres dimensiones.	231
`scatter3`	Crea un gráfico de dispersión.	240
`sphere`	Dibuja una esfera.	239
`stem3`	Crea un gráfico de tallo tridimensional.	239
`surf`	Crea un gráfico de superficie.	235, 237
`surfc`	Crea un gráfico de superficie con contorno.	237
`surfl`	Crea un gráfico de superficie con iluminación.	237
`waterfall`	Crea un gráfico de malla con efecto catarata.	238

Tabla A.16: Formato de gráficos.

Carácter	Descripción	Página
`axis`	Establece los límites de los ejes.	119
`colormap`	Establece el color.	236
`grid`	Activa la rejilla de un gráfico.	120, 236
`gtext`	Añade texto a un gráfico.	117
`legend`	Añade una leyenda a un gráfico.	117
`subplot`	Crea múltiples gráficos en la misma página.	127
`text`	Añade texto a un gráfico.	117
`title`	Añade el título a un gráfico.	117
`view`	Controla el ángulo de visión de un gráfico en tres dimensiones.	240
`xlabel`	Añade una etiqueta al eje *x*.	117
`ylabel`	Añade una etiqueta el eje *y*.	117

Tabla A.17: Funciones matemáticas (crear, evaluar y resolver).

Carácter	Descripción	Página
`feval`	Evalúa el valor de una función matemática.	149
`fminbnd`	Calcula el mínimo de una función.	253
`fzero`	Calcula la solución de una ecuación de una variable.	251
`inline`	Crea una función en línea.	146

Tabla A.18: Integración numérica.

Carácter	Descripción	Página
`quad`	Integra una función.	255
`quadl`	Integra una función.	256
`trapz`	Integra una función.	257

Tabla A.19: Resolución de ecuaciones diferenciales ordinarias.

Carácter	Descripción	Página
`ode113`	Resuelve una ecuación diferencial ordinaria de primer orden.	259
`ode15s`	Resuelve una ecuación diferencial ordinaria de primer orden.	259
`ode23`	Resuelve una ecuación diferencial ordinaria de primer orden.	259
`ode23s`	Resuelve una ecuación diferencial ordinaria de primer orden.	259
`ode23t`	Resuelve una ecuación diferencial ordinaria de primer orden.	259
`ode23tb`	Resuelve una ecuación diferencial ordinaria de primer orden.	259
`ode45`	Resuelve una ecuación diferencial ordinaria de primer orden.	259

Tabla A.20: Funciones lógicas.

Carácter	Descripción	Página
`all`	Determina si todos los elementos de un array son distintos de cero.	166
`and`	Y lógico.	165
`any`	Determina si algún elemento de un array es distinto de cero.	166
`find`	Encuentra los índices de ciertos elementos de un vector.	166
`not`	NO lógico.	165
`or`	O lógico.	165
`xor`	O exclusivo lógico.	165

Tabla A.21: Comandos de control de flujo.

Carácter	Descripción	Página
`break`	Termina la ejecución de un bucle.	185
`continue`	Termina una iteración dentro de un bucle.	185
`else`	Ejecución condicional de comandos.	170,
`elseif`	Ejecución condicional de comandos.	171
`end`	Terminación de bucles y sentencias condicionales.	168, 174, 176, 180
`for`	Repite la ejecución de un grupo de comandos.	176
`if`	Ejecución condicional de comandos.	168
`switch`	Escoge entre distintos casos o valores de una expresión.	172
`while`	Repite la ejecución de un grupo de comandos.	180

Tabla A.22: Funciones polinómicas.

Carácter	Descripción	Página
`conv`	Multiplica polinomios.	205
`deconv`	Divide polinomios.	205
`poly`	Calcula los coeficientes de un polinomio.	204
`polyder`	Calcula la derivada de un polinomio.	206
`polyval`	Calcula el valor de un polinomio.	202
`roots`	Calcula las raíces de un polinomio.	203

Tabla A.23: Curvas de ajuste e interpolación.

Carácter	Descripción	Página
`interp1`	Interpolación unidimensional.	215
`polyfit`	Polinomio de ajuste a partir de una serie de puntos.	209

Tabla A.24: Cálculo simbólico.

Carácter	Descripción	Página
`collect`	Agrupa los términos de una expresión.	278
`diff`	Calcula la derivada de una ecuación.	287
`double`	Convierte número simbólicos a su forma númerica.	275
`dsolve`	Resueve una ecuación diferencial ordinaria.	290
`expand`	Desarrolla una expresión.	279
`ezplot`	Crea un gráfico a partir de una expresión.	293
`factor`	Factoriza una expresión en polinomios de grado menor.	279
`findsym`	Visualiza las variables simbólicas de una expresión.	277
`int`	Calcula la integral de una expresión.	288
`pretty`	Visualiza una expresión en forma algebraica.	281
`simple`	Encuentra una expresión con el menor número de caracteres.	280
`simplify`	Simplifica una expresión.	280
`solve`	Resuelve una ecuación o un sistema de ecuaciones.	282
`subs`	Substituye números en expresiones simbólicas.	297
`sym`	Crea un objeto simbólico.	272
`syms`	Crea varios objetos simbólicos con una sola declaración.	273

Respuestas de problemas seleccionados

Capítulo 1

2. *a*) 1,7584e+003
   b) 1,0174e+007

4. *a*) 1,4395
   b) 0,2325

6. *a*) 629,1479
   b) 2,0279

8. $r = 4{,}3718$ in., $A = 240{,}1759$ in^2.

12. *a*) $b = 17{,}8885$ cm
    b) $\alpha = 31{,}5881°$

14. $d = 2{,}5725$

16. *a*) 1233,82 €
    b) 1301,68 €
    c) 1202,00 €

18. 707.95

Capítulo 3

2. y = 1,0e+004 *
   –0,5954 – 0,1000 – 0,0116 – 0,0008
   – 0,0000 0 0,0000 0,0008 0,0116
   0,10000 0,5954 2,7000

4. z = 6,7415 5,4965 5,2579 5,2421
   5,2415

6. y = 2,0000 2,5937 2,7048 2,7156
   2,7169 2,7176 2,7179 2,7181

8. *a*) Sn = 0,6883
   b) Sn = 0,6926
   c) Sn = 0,6931

12.

5	9957
10	19611
15	28670
20	36857
25	43925
30	49657
35	53881
40	56468
45	57339
50	56468
55	53881
60	49657
65	43925
70	36857
75	28670
80	19611
85	9957

14.

Resistencia	Corriente(A)
R_1	0,6750
R_2	1,1036
R_3	0,8011
R_4	0,3025
R_5	0,9324
R_6	1,2349
R_7	1,9533
R_8	1,3610
R_9	3,3163
R_{10}	2,8877

Capítulo 4

2. theta = 37,2750 28,1630 22,0930
   18,0097 15,1373

4.

theta	tiempo (h)
0	4,0857
10,000	3,9469
20,000	3,8407
30,000	3,7650
40,000	4,7232
50,000	3,7276
60,000	3,8126

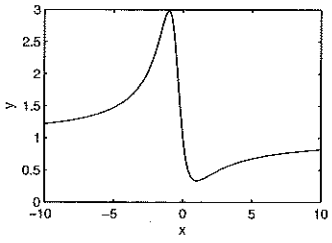
6.

Tiempo (s)	Distancia (m)	Velocidad (m/s)
0	0	0
1,0000	0,7750	1,5500
2,0000	3,1000	3,1000
3,0000	6,9750	4,6500
4,0000	12,4000	6,2000
5,0000	19,3750	7,7500
6,0000	27,9000	9,3000
7,0000	37,9750	10,8500
8,0000	49,6000	12,4000
9,0000	62,7750	13,9500
10,0000	77,5000	15,5000

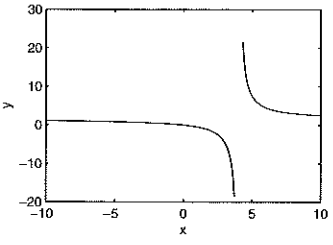
8. $a = 2b = 3{,}5$
$c = 4{,}2 \qquad b = 7$

Capítulo 5

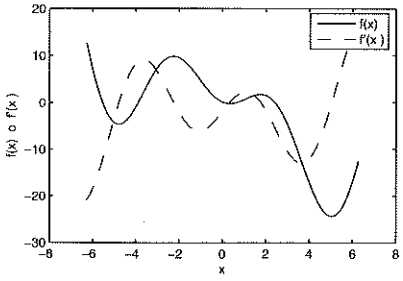
2.



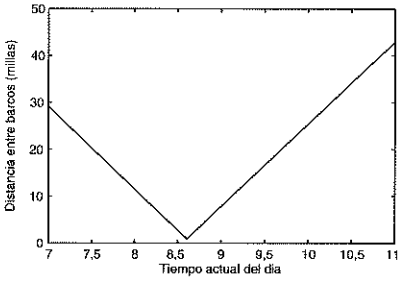
4.



6.

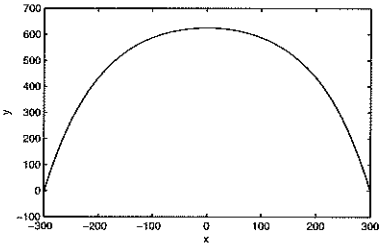


8.

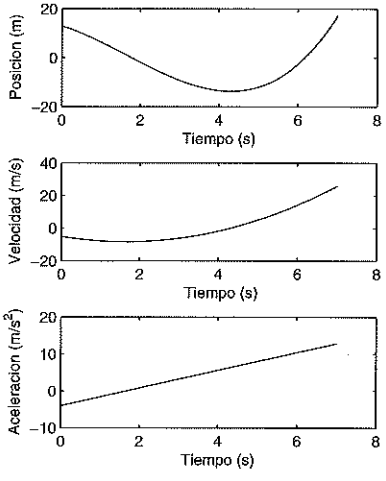


Visibilidad inferior a 8 millas, desde las 8,1 AM hasta las 9 AM,

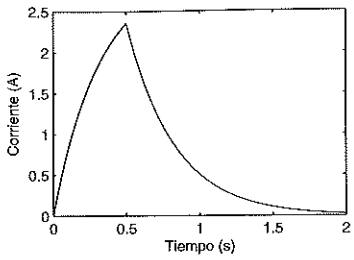
10.



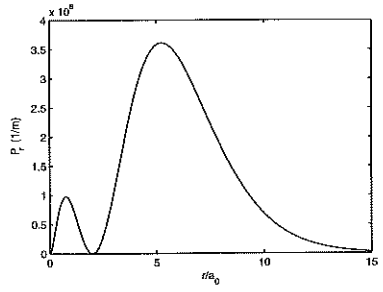
12.



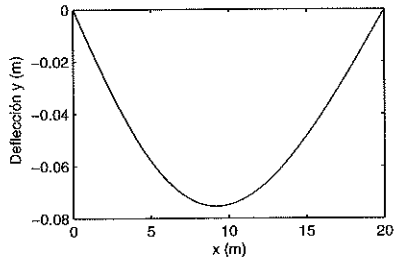
14.



16.



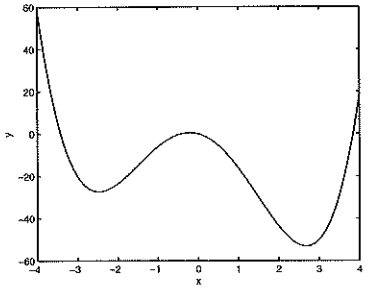
18.



Capítulo 6

2. *a*) $y(-3) = -20{,}1$, $y(5) = 237{,}5$

b)



4. *a*) (3, 21)
   b) (1, 2)

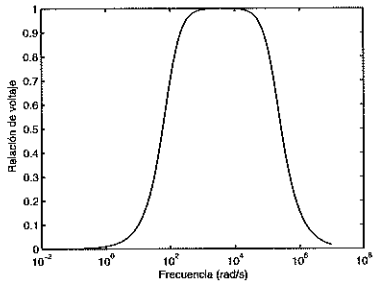
6. 56,5 N-m

8. *a*) –0,7730 0,5797 0,2577
   b) 0,7730 –0,5797 –0,2577
   c) 0,7071 0 –0,7071

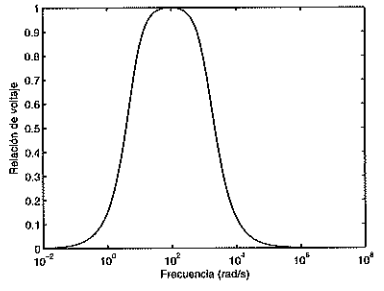
10. *a*) 75,6000
    b) 81,6000
    55,8000
    68,8000
    83,4000

14. *a*) 179,1974 MPa –691974 MPa
    b) –6,7199 psi –21,2801 psi

16. *a*)



b)



Capítulo 7

2. *a*) $y = 1$
   b) $y = 0$
   c) $y = 10$

4. $y = [2\ 1\ 4]$

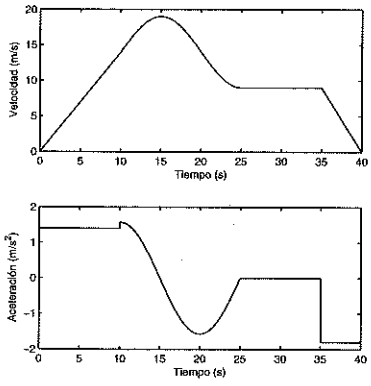
6. *a*) Nueva York 37,6774 °F
   Anchorage 33,1290 °F
   b) Nueva York 17
   Anchorage 13
   c) 11 días, en días: 1 7 9 14 15 18 19 21 22 25 26
   d) 1 día: el 23.
   e) 16 días: 7 8 9 13 14 15 16 17 18 19 20 23 24 25 26 27

8. *a*) La ecuación tiene dos raíces
   x1 = 0,345208
   x2 = –4,345208

b) La ecuación no tiene raíces reales
c) La ecuación tiene una raíz:
$x = -0{,}333333$

18. (11,3099, 15,2971)
(120,2564, 13,8924)
(207,8973, 19,2354)
(–33,0239, 11,9269)

20.



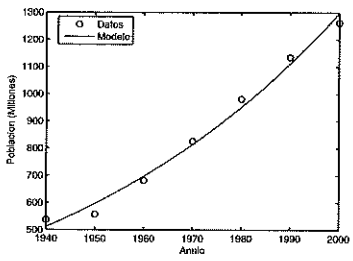
Capítulo 8

2. $3x^2 + 7x - 5$

4. $x = 0{,}0022785$ m

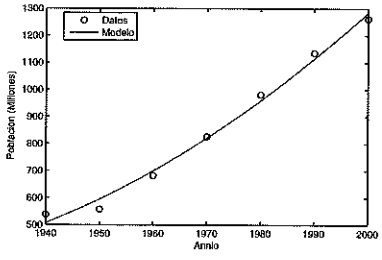
8. $a = 7{,}5$ m, $b = 5$ m

10. *a*)



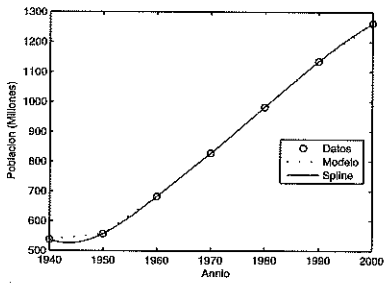
Población
1955 = 644,7597 millones

b)



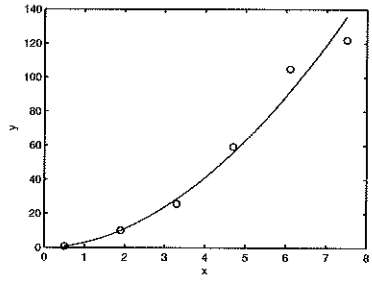
Población 1955 = 645,1964 millones

c)

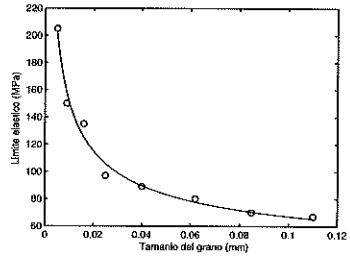


Población 1955L = 619,5 millones
Población 1955S = 613,058 millones

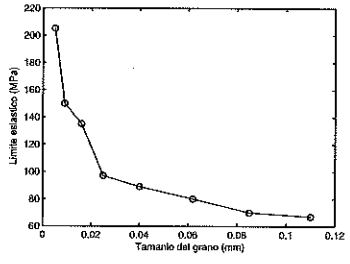
12.



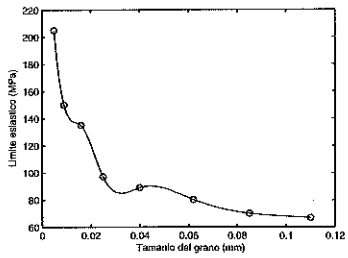
14. *a*) $k = 12{,}2603$ MPa $\sqrt{\text{mm}}$
$\sigma_0 = 28{,}2938$ MPa
$\sigma_y = 83{,}1237$ MPa para $d = 0{,}05$ mm



b) $\sigma_y = 84{,}9091$ MPa para $d = 0{,}05$ mm

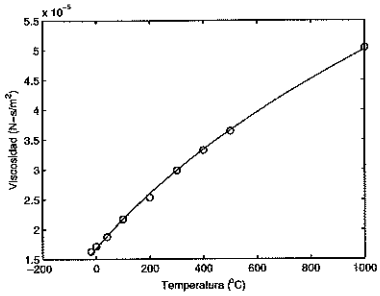


c) $\sigma_y = 88{,}5457$ MPa para $d = 0{,}05$ mm



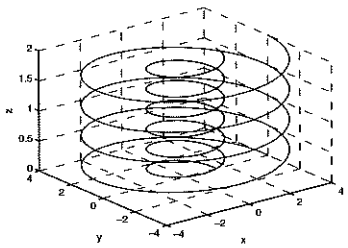
16. $C = 1{,}5682\text{e-}6 \dfrac{\text{kg}}{\text{m} \cdot \text{s} \cdot \text{K}^{1/2}}$

S = 148,1622

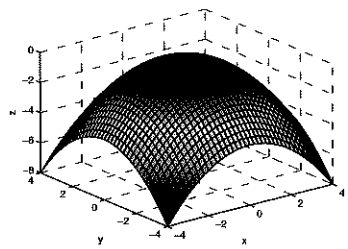


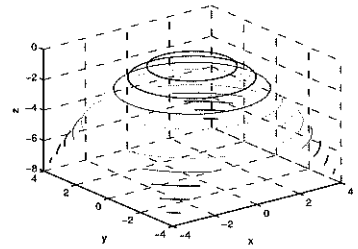
Capítulo 9

2.

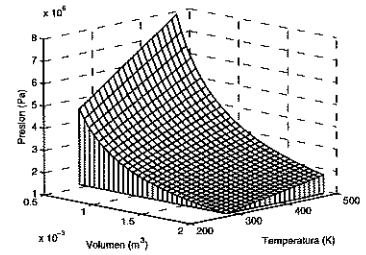


4.

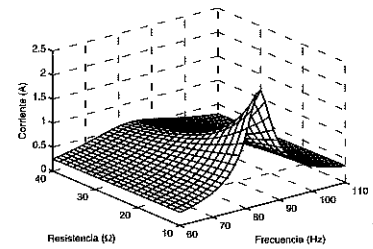




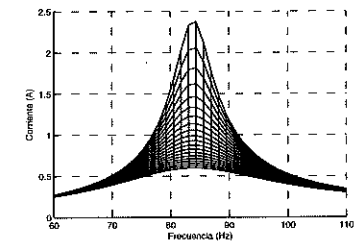
6.



8. *a*)

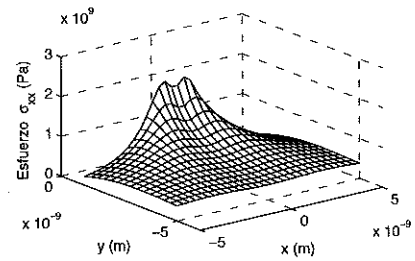


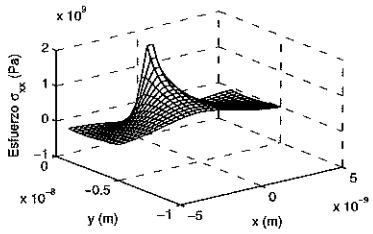
b)

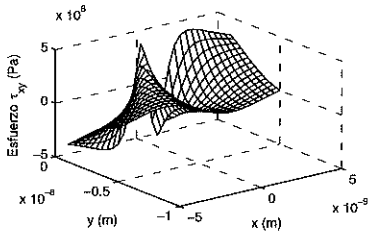


$1/(2\pi\sqrt{LC}) = 83{,}883$ Hz

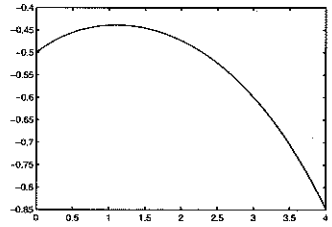
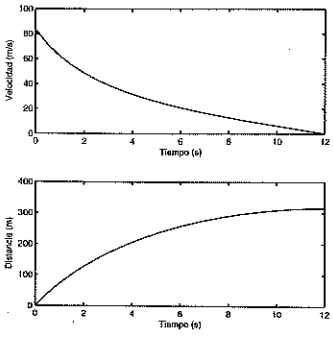
10.



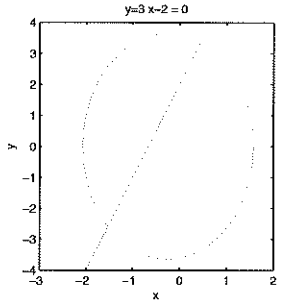




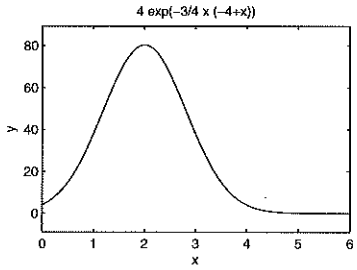
Capítulo 10

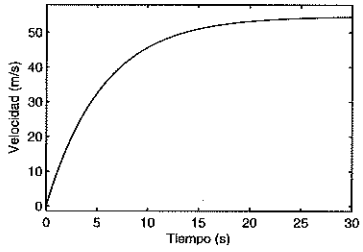
2. 3,4664, 6,2869, 9,1585
4. 53,1653°
6. 24,2280°, 80,5135 N
8. r = 12,2474 cm, h = 17,3205 cm
10. 0,8774
12. E = 6,0986e+006 N/C
14. 264000 m^2
16. 
18. 

Capítulo 11

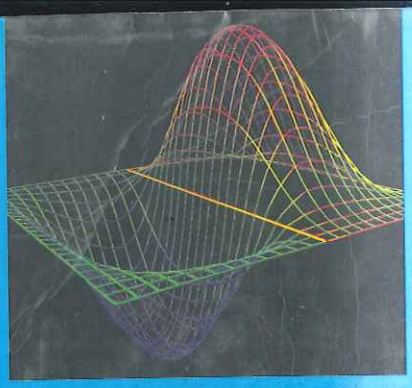
2. *a*) y^3 + 8
   b) (y+2)/(y^2–2*y+4)
   c) –y+6+y^2
   d) 26
4. *a*) (x–1)*(x–3)*(x+5)*(x+3)*(x+2)
   b) x^4–3*x^3–24*x^2+28*x+48
8. 5 in., 2,9289 in.
10. *a*) 

   b) x = (5/13)±(1/13) $\sqrt{142}$
   c) y = (–11/13) ±(3/13) $\sqrt{142}$
12. (–1+2*log(3))/log(3)
18. y = 4*exp(–3/4*x*(–4+x))



20. *a*) g/c*m–exp(–c/m*t)*g/c*m
   b) 16,1489 kg/s
   c) 

Índice alfabético



Amos Gilat

Matlab®

Una introducción con ejemplos prácticos

Este libro ofrece una guía práctica para el estudiante, profesor, científico, ingeniero o, simplemente, cualquier lector interesado en el software MATLAB® que quiera adentrarse paulatinamente en el manejo y comprensión de la nueva versión 7 de este programa científico. Comenzando con un repaso de los aspectos más básicos, el libro cubre gran parte de lo que un usuario de MATLAB® necesita para aplicarlo de forma efectiva en cualquier campo de las ciencias: desde operaciones aritméticas simples con escalares, hasta la creación y uso de *arrays*, gráficos en dos y tres dimensiones, curvas de ajuste e interpolación, programación, aplicaciones en el cálculo numérico, etc. El libro contiene explicaciones detalladas de cada uno de los comandos de MATLAB®, con sus correspondientes ejemplos y tutoriales, que pueden ser seguidos fácilmente por el lector. De esta manera se pretende que el texto sea también una poderosa herramienta para el autoaprendizaje. Con el objetivo de instruir al lector en el uso de MATLAB®, todos los capítulos contienen ejemplos resueltos paso a paso, y, además, se proponen muchos otros para que desarrolle su habilidad en el uso de este programa.

EDITORIAL REVERTÉ S.A.
www.reverte.com

ISBN-10: 84-291-5035-8
ISBN-13: 978-84-291-5035-3