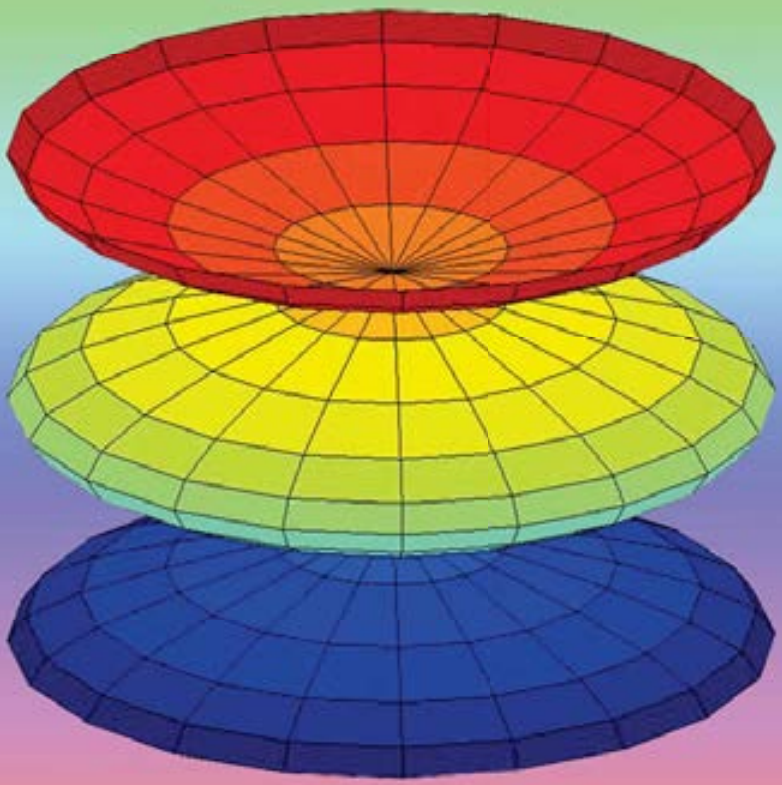
Matlab®

y sus Aplicaciones en las Ciencias y la Ingeniería



PEARSON
Prentice Hall

César Pérez

MATLAB y sus aplicaciones en las ciencias y la ingeniería

MATLAB y sus aplicaciones en las ciencias y la ingeniería

César Pérez López

Universidad Complutense de Madrid

* * * *

Instituto de Estudios Fiscales

Prentice Hall

Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima • Montevideo
San Juan • San José • Santiago • São Paulo • White Plains

Datos de catalogación bibliográfica

CÉSAR PÉREZ LÓPEZ
MATLAB y sus aplicaciones en las ciencias y la ingeniería

PEARSON EDUCACIÓN, S.A., Madrid, 2002

ISBN: 84-205-3537-0
Materia: Estadística 519

Formato 170 × 240 Páginas: 632

CÉSAR PÉREZ LÓPEZ
MATLAB y sus aplicaciones en las ciencias y la ingeniería

ISBN: 84-205-3537-0
Depósito legal: M.

PRENTICE HALL es un sello editorial autorizado de PEARSON EDUCACIÓN, S.A.

Equipo editorial:
Editora: Isabel Capella
Técnico editorial: Marta Caicoya
Equipo de producción:
Dirección: José Antonio Clares
Técnico: Diego Marín
Diseño de cubierta: Equipo de diseño de PEARSON EDUCACIÓN, S.A.
Impreso por: Gráficas ROGAR

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Este libro ha sido impreso con papel y tintas ecológicos

CONTENIDO

Introducción

MATLAB es un entorno de computación técnica que posibilita la ejecución del cálculo numérico y simbólico de forma rápida y precisa, acompañado de características gráficas y de visualización avanzadas aptas para el trabajo científico y la ingeniería. MATLAB es un entorno interactivo para el análisis y el modelado que implementa más de 500 funciones para el trabajo en distintos campos de la ciencia.

Por otra parte, MATLAB presenta un lenguaje de programación de muy alto nivel basado en vectores, arrays y matrices.

Además, el entorno básico de MATLAB se complementa con una amplia colección de toolboxes que contienen funciones específicas para determinadas aplicaciones en diferentes ramas de las ciencias y la ingeniería.

La arquitectura de MATLAB es abierta y ampliamente extensible, permitiendo la relación con Excel, C, Fortran y otras aplicaciones externas muy utilizadas e importantes. Entre otras cosas, el código escrito en lenguaje de MATLAB puede ser traducido a C de forma inmediata.

MATLAB también permite la operatividad entre plataformas posibilitando trabajar con distintos sistemas operativos y relacionar el trabajo realizado en las distintas plataformas.

MATLAB es un software en continuo crecimiento y muy adaptable a los avances científicos y al trabajo en laboratorios I+D, que resuelve los problemas que presenta la ingeniería en el desarrollo de productos innovadores.

En el campo de las *Comunicaciones,* MATLAB permite realizar modelado y diseño de sistemas DSP, trabajar con sistemas conmutados, con telefonía fija/móvil o ADSL y con modelado de canal/emisor/receptor.

En el campo de los *Periféricos para ordenadores*, MATLAB dispone de drivers para discos, de periféricos de control para posición/velocidad y de instrumentación.

En el campo *Aeroespacial/Defensa,* MATLAB permite trabajar en sistemas radar, unidades de seguimiento y rastreo, aviónica, modelado y control de sistemas de potencia y guiado, y navegación y control.

En el campo de la *Automoción,* MATLAB posibilita aplicaciones para trabajar en la ingeniería de control, sistemas de suspensión, sistemas ABS y diseño de bloques de embrague.

Pero MATLAB tampoco olvida otros campos importantes como el de las *Finanzas cuantitativas*, pudiendo utilizarse como un entorno de cálculo para el análisis de datos, para la valoración y análisis de opciones e instrumentos financieros, para la optimización de carteras y análisis de riesgos y para el desarrollo de modelos y su validación. Asimismo, MATLAB se puede utilizar como un entorno de desarrollo de aplicaciones de renta fija, de opciones derivadas, de distribución de activos/gestión de cartera y de gestión de riesgo y reporting.

Materias como la *Estadística*, el *Álgebra lineal*, el *Análisis matemático*, el *Análisis numérico*, el *Análisis de series temporales,* las *Bases de datos* y la *Geometría* encuentran en el módulo básico de MATLAB y en sus toolboxes adicionales una herramienta esencial para su desarrollo.

También MATLAB, a través de Simulink, permite diseñar sistemas dinámicos sencillos o complejos y realizar modelado y simulación mediante un lenguaje agradable basado en diagramas de bloques. Admite sistemas en tiempo continuo, sistemas de control y control inteligente, y aplicaciones de procesado de señal digital y comunicaciones.

En este libro se comenzará tratando el módulo básico de MATLAB y sus aplicaciones en materias como la programación, el análisis matemático, el álgebra lineal y el cálculo numérico. A continuación se analizan los toolboxes más interesantes (matemática simbólica, estadística, optimización, etc.). Posteriormente se abordará el trabajo con los toolboxes más útiles en sistemas de control y otras aplicaciones de MATLAB en el campo de la ingeniería. Al final de cada capítulo se presentan ejemplos prácticos totalmente resueltos que aclaran los conceptos y amplían el campo de aplicación de MATLAB.

1

El software MATLAB y sus componentes

1.1 El software básico MATLAB y sus herramientas adicionales

MATLAB, en su contenido básico, es un entorno integrado de trabajo que permite el análisis y la computación matemáticos interactivos de forma sencilla con más de 500 funciones matemáticas, estadísticas y técnicas implementadas, así como la visualización a través de todo tipo de gráficos de ingeniería y científicos.

También es posible con el módulo básico de MATLAB el desarrollo de algoritmos a través de un lenguaje propio de programación que resulta ser abierto (integrable con C, Excel y Bases de Datos), extensible (a través de las funcionalidades que aportan las librerías especializadas complementarias) y de sintaxis similar al C (pero sin las dificultades técnicas de programación que presenta C).

Asimismo, MATLAB integra un conjunto importante de herramientas básicas adicionales muy útiles, entre las que destacan las siguientes:

MATLAB Report Generador

Permite la creación de informes estándar y personalizados de los algoritmos desarrollados en MATLAB. Los informes pueden ejecutar comandos de MATLAB a medida que se van procesando, lo cual nos ofrece la posibilidad de documentar nuestras pruebas con MATLAB a medida que las realizamos.

MATLAB Compiler Suite: MATLAB Compiler, MATLAB C/C++ Math Library, MATLAB C/C++ Graphics Library

Convierte, de forma automática, los ficheros de MATLAB que contienen los algoritmos que hemos desarrollado (en lenguaje de MATLAB) a código C y C++, bien para poder distribuir aplicaciones independientes o para mejorar el rendimiento del algoritmo. Se trata por tanto de un ***generador de código C*** a partir de los programas de MATLAB.

MATLAB Runtime Server

Permite distribuir cualquier aplicación desarrollada con MATLAB de forma sencilla mediante un runtime.

MATLAB Web Server

Permite incorporar funciones de MATLAB (matemáticas y gráficas) a nuestras aplicaciones web.

Matriz VB

Librería para poder utilizar desde Visual Basic las funciones matemáticas y gráficas que incorpora MATLAB.

MATLAB Excel Builder

Permite transformar algoritmos desarrollados en MATLAB a funciones de Excel y usar estas funciones desde Excel sin necesidad de tener MATLAB.

1.2 Toolboxes de MATLAB de aplicación en matemática general

Una faceta muy importante de MATLAB son los ***toolboxes*** añadidos, que consisten en paquetes de ampliación al software básico y que son aplicables a determinados campos de la ciencia. A continuación se relacionan los toolboxes más interesantes de MATLAB.

Symbolic Math

Permite integrar la expresión y el cálculo simbólicos (cálculo, transformadas, álgebra lineal, ecuaciones) al entorno de cálculo y visualización de MATLAB.

Extended Symbolic Math

Incluye todas las características de Symbolic Math, proporciona soporte completo para la programación en Maple y permite el acceso completo a las librerías matemáticas de Maple.

Database Toolbox

Permite directamente desde MATLAB consultar e intercambiar datos con las bases de datos ODBC/JDBC más populares (Oracle, Sybase SQL Server, Sybase SQL Anywhere, Microsoft SQL Server, Microsoft Access, Informix and Ingres) de forma dinámica, preservándolos durante el intercambio y simultáneamente con más de una base de datos.

Excel Link Toolbox

Integra toda la potencia de MATLAB con Microsoft Excel permitiendo la transferencia de datos en los dos sentidos, ejecutando cualquier función de MATLAB desde una fórmula de Excel o utilizando Excel desde MATLAB como editor de vectores.

Statistics Toolbox

Funciones y herramientas interactivas para el análisis de datos históricos, el modelado y simulación de sistemas y para el desarrollo de algoritmos estadísticos. Soporta 20 distribuciones de probabilidad, incorpora el control estadístico de procesos, el diseño de experimentos, estadística descriptiva, análisis clúster, etc.

Optimization Toolbox

Proporciona diversos algoritmos y técnicas para solucionar problemas de optimización no lineales, tanto generales como a gran escala.

Spline Toolbox

Mediante un interface gráfico proporciona potentes funciones para el ajuste de datos, visualización, interpolación y extrapolación mediante técnicas spline.

Partial Differential Equation Toolbox

De aplicación en la solución de problemas en muchos campos de la física y la ingeniería: transferencia de calor, flujo en medios porosos, medios conductores, cálculo de esfuerzos y fatigas en estructuras, campos magnéticos, etc. Usa el método FEM (Método de los Elementos Finitos que incorpora el algoritmo de triangulación de Delaunay) para solucionar ecuaciones diferenciales parciales.

Neural Network Toolbox

Proporciona las versiones más comunes de paradigmas y algoritmos para el diseño y simulación de redes neuronales. Incluye bloques de Simulink para poder usar esta toolbox en aplicaciones de control y simulación de sistemas. Incluye ejemplos de control predictivo y control adaptativo. Entre las aplicaciones más comunes de las redes neuronales tenemos las técnicas de clasificación, predicción, filtrado, optimización, reconocimiento de patrones, aproximación a funciones, interpretación y clasificación de imágenes.

1.3 Toolboxes de MATLAB de adquisición de datos

Dentro de los toolboxes de MATLAB tienen un papel importante los relativos a la ***adquisición de datos*** de otras aplicaciones. En esta categoría destacan los siguientes:

Data Acquisition Toolbox

Permite el control y la comunicación con una gran variedad de dispositivos de adquisición de datos estándares en la industria (National Instruments, Agilent, Computer Boards, etc.). Incluye kit de adaptación para el desarrollo de interfaces para nuevos dispositivos.

Instrument Control Toolbox

Permite la comunicación con instrumentos (analizadores de espectro, osciloscopios, generadores de funciones) y dispositivos externos. Soporta los protocolos de comunicación GPIB (IEEE-488, HPIB) y VISA (Serial, GPIB, VXI, GPIB-VXI) y proporciona soporte avanzado de puerto serie (RS-232, RS-422, RS485).

Curve Fitting Toolbox

Mediante un interface gráfico podemos realizar ajustes de curvas, visualizando y preprocesando los datos y usando una amplia gama de modelos y métodos de ajuste.

1.4 Toolboxes de MATLAB para procesado de señales

Otra campo importante en el que MATLAB presenta herramientas es el ***procesado de señales***. Destacan los siguientes toolboxes:

Signal Processing Toolbox

Conjunto de funciones para analizar, manipular y visualizar señales y sistemas lineales. Incorpora un interface para diseñar y analizar de forma interactiva filtros digitales (FIR and IIR).

Filter Desgin Toolbox

Complementa la Signal Processing Toolbox añadiendo técnicas avanzadas de filtros digitales para aplicaciones complejas de DSP en tiempo real. También proporciona funciones para simplificar el diseño de filtros de punto fijo y para el análisis de los efectos de "quantization".

Communications Toolbox

Conjunto de funciones para MATLAB que facilitan el diseño de algoritmos y componentes de sistemas de comunicaciones.

Wavelet Toolbox

Funciones basadas en el análisis wavelet para analizar y procesar señales, imágenes (señal bidimensional) y series temporales; son adecuadas para el estudio de señales con características no estacionarias o transitorias en las que el análisis del tiempo en que dichas señales experimentan los cambios es primordial (para el estudio de este tipo de señales no es adecuado el análisis de Fourier). Estas aplicaciones son muy convenientes para eliminación de ruidos y ecos, compresión de imágenes y vídeo.

System Identification Toolbox

Proporciona herramientas para crear modelos matemáticos de sistemas dinámicos, de los cuales desconocemos su comportamiento, a partir de los datos de entrada observados y de los de salida. De aplicación en una gran variedad de campos.

1.5 Toolboxes de MATLAB para procesado de imágenes

También en el área del procesado de imágenes MATLAB presenta los siguientes toolboxes:

Image Processing Toolbox

Entorno interactivo que proporciona un conjunto de herramientas para el análisis y procesado de imágenes con un amplio abanico de aplicaciones. Algunas de estas herramientas son: operaciones geométricas, análisis de imagen, eliminación de ruidos, filtros lineales, filtros 2-D, transformaciones, ROI (Region-of-Interest), operaciones binarias, conversiones de color, procesado por "neighborhood" y por bloques, etc.

Mapping Toolbox

Conjunto de herramientas para el análisis y visualización de información gráfica de tipo geográfico.

1.6 Toolboxes de MATLAB en el área financiera

Las finanzas es otro campo en el que MATLAB ha desarrollado toolboxes. Los más importantes son los siguientes:

Financial Toolbox

Proporciona las herramientas básicas para finanzas cuantitativas y prototipaje analítico aplicables a optimización de portfolios y análisis de riesgos, cálculos de precios y sensibilidades según diversos modelos, análisis de volatilidad (ARCH/GARCH), análisis básico de series temporales, etc.

Financial Time Series Toolbox

Conjunto de herramientas para el análisis de series temporales en mercados financieros: análisis y transformación de datos, análisis técnico (osciladores, índices, estocásticos, indicadores), visualización.

Financial Derivatives Toolbox

Permite la creación y gestión de portafolios con diversos instrumentos financieros, así como calcular sus precios y sensibilidades. Proporciona análisis "hedging".

Garch Toolbox

Proporciona un entorno de cálculo integrado para trabajar con el modelo GARCH de volatilidad. Usa un modelo compuesto ARMAX/GARCH para simulaciones, previsiones, estimación de parámetros de series temporales, etc.

Datafeed Toolbox

Permite desde MATLAB el acceso a los servicios de datos financieros (Bloomberg, Interactive Data, Yahoo Finance) para su descarga y posterior análisis en MATLAB.

1.7 Simulación de sistemas con SIMULINK y sus herramientas adicionales

Otro de los campos tratados especialmente por MATLAB, y de gran aplicación en la ingeniería, es la automatización del diseño mediante ***simulación de sistemas dinámicos***. Los productos más importantes en el área de simulación de sistemas son:

Simulink

Es un entorno gráfico interactivo para el modelado, análisis y simulación de una gran variedad de sistemas dinámicos (discretos, analógicos e híbridos) mediante la utilización de diagramas de bloques. Permite la incorporación de los algoritmos y controles que se hayan desarrollado en C previamente a la utilización de Simulink. Trabaja totalmente integrado con MATLAB.

Stateflow

Es un entorno gráfico interactivo para el modelado de la lógica de sistemas dinámicos basados en eventos (temporales o de estado). Se basa en la teoría de máquinas de estado finito y utiliza diagramas de transición de estado para expresar la lógica del sistema y diagramas de control de flujo. Trabaja perfectamente integrado con Simulink.

Simulink Report Generator

Permite la documentación automática mediante la creación de diversos informes de los modelos desarrollados en Simulink.

Simulink Performance Tools

Es un conjunto de 4 herramientas que gestionan y optimizan el rendimiento de Simulink en las simulaciones de modelos de gran escala: Simulink Accelerator, Simulink Model Profiler, Simulink Model Differencing y Simulink Model Coverage.

Requirements Management Interface

Nos permite coordinar, registrar e implementar los cambios en el diseño de especificaciones a lo largo del ciclo de desarrollo. Esta herramienta nos permite asociar los requerimientos del proyecto con modelos de Simulink, diagramas de Stateflow y algoritmos de MATLAB.

Virtual Reality

Permite la creación de escenas e imágenes en movimiento en un entorno de realidad virtual de 3 dimensiones. Desde MATLAB y Simulink podemos representar y visualizar en este entorno de realidad virtual el modelo que estamos simulando e interactuar con él, bien desde Simulink o bien desde el propio entorno de realidad virtual.

1.8 Blocksets de SIMULINK

Al igual que MATLAB presentaba toolboxes adicionales con extensiones del programa general aplicadas a diversos campos, Simulink también presenta aplicaciones adicionales con extensiones de simulación de sistemas denominadas ***blocksets***. A continuación se relacionan los más interesantes:

DSP Blockset

Proporciona un conjunto de bloques para Simulink que son el fundamento para el diseño de muchas aplicaciones de procesado de señales digitales (DSP) como procesado básico de señal, estimación espectral, diseño de filtros. Todos los bloques soportan simulación por muestreo y por frames.

Communications Blockset

Conjunto de más de 150 bloques para Simulink para diseñar de forma completa y simular sistemas de comunicaciones.

CDMA Reference Blockset

Conjunto de bloques de Simulink para crear y simular modelos de sistema de comunicaciones inalámbricos bajo el estándar IS-95A.

Fixed-Point Blockset

Permite emular la aritmética de punto fijo cuando diseñamos y simulamos sistemas dinámicos o filtros digitales que al final serán implementados en targets digitales de punto fijo.

Dials & Gauges Blockset

Monitoriza señales y parámetros de simulación mediante elementos gráficos (instrumentos de aspecto real). Nos permite añadir a los modelos de Simulink estos elementos gráficos y así visualizar el entorno que estamos modelando.

Nonlinear Control Design Blockset

Proporciona una aproximación al diseño de sistemas de control basada en una optimización que ajusta los parámetros de acuerdo con unas restricciones en la respuesta transitoria temporal del sistema fijadas por el usuario.

Power System Blockset

Permite modelar y simular en Simulink sistemas eléctricos de potencia (generación, transmisión, distribución) y su control (motores, transformadores, tiristores, diodos, etc.)

Sym Mechanics Blockset

Permite modelar y simular de forma sencilla en Simulink los componentes de un sistema mecánico, ver y animar su movimiento, estudiar su cinemática y dinámica (directa e inversa), etc.

1.9 Generación de código de SIMULINK

La generación de código en el campo de la simulación es otra de las facetas tratadas por MATLAB. Entre los generadores de código tenemos:

Real-Time Workshop

Genera código C en tiempo real a partir de los modelos realizados en Simulink, lo que nos permite realizar prototipaje rápido, acelerar las simulaciones o realizar simulaciones en tiempo real.

Real-Time Workshop Embedded Coder

Genera código C en tiempo real optimizado en velocidad de ejecución y con unos mínimos requerimientos de memoria para usarlo en sistemas “embedded” de tiempo real. Este código puede ser descargado directamente al procesador target. El código generado es comparable al código optimizado escrito a mano.

Stateflow Coder

Genera código C en tiempo real a partir de los diagramas de transición realizados en Stateflow.

1.10 Implementación en targets

El desarrollo de tarjetas (*targets*) para relacionar MATLAB con aplicaciones de otros sistemas ha registrado un avance importante en las últimas versiones del programa. La ***implementación en targets*** ha originado módulos como los siguientes:

Developer's Kit for Texas Instruments DSP

Este software facilita el diseño, análisis e implementación de aplicaciones para DSPs de Texas Instruments al integrar MATLAB, Simulink y Real-Time Workshop con el software (Code Composer Studio, RTDX) y targets (C5000, C6000, C6701EVM, DSKs) de Texas Instruments.

Motorola DSP Developer's Kit.

Este software integra MATLAB y Simulink con el software de Motorola (Motorola's Suite 56) para el desarrollo de aplicaciones basadas en DSPs de Motorola (familias 56300 y 56600).

Xilinx's FPGA System Generator for Simulink

Este software permite el desarrollo de aplicaciones DSP de alto rendimiento para los FPGAs de Xilinx (Spartan II y Virtex/E) usando MATLAB y Simulink (Xilinx Blockset). Entre otras prestaciones genera código VHDL a partir de los modelos de Simulink.

1.11 Prototipaje

En el campo del ***prototipaje*** MATLAB dispone de las siguientes aplicaciones:

Real-Time Windows Target

Permite ejecutar los modelos de Simulink y Stateflow en tiempo real en un PC con Windows. Durante la ejecución podemos comunicarnos con una amplia variedad de tarjetas I/O (más de 100), lo cual nos permite controlar sensores, actuadores y otros dispositivos para poder experimentar, desarrollar y testear nuestros sistemas tiempo-real.

xPC Target

Permite añadir bloques I/O a Simulink, generar código con Real-Time Workshop y descargar este código en un segundo PC que ejecuta el kernel de xPC Target. Es ideal para prototipaje rápido. Permite la ejecución de modelos en tiempo real en un segundo PC (como target) sin necesidad de Windows. Con esta solución, el PC que actúa como host y el que actúa como target se mantienen comunicados durante la ejecución en tiempo real.

xPC Target Embedded Option

Esta opción nos permite que el modelo desarrollado se ejecute en el PC que actúa como target sin necesidad de que esté concectado al PC host. Esta opción es la adecuada cuando nuestro modelo está finalizado y lo entregamos para su funcionamiento en producción.

1.12 Análisis y diseño de sistemas de control

El ***análisis y diseño de los sistemas de control*** es otro de los campos de la ingeniería que cubre MATLAB con diversos toolboxes. Entre los más importantes tenemos los siguientes:

Control System Toolbox

Proporciona un entorno interactivo y gráfico para el modelado, análisis y diseño de sistemas de control, utilizando tanto los métodos clásicos como los modernos: root locus, loop shaping, pole placement y LQR/LQG regulation. Soporta sistemas LTI, SISO, MIMO y diversos métodos de conversión.

Fuzzy Logic Toolbox

Herramienta interactiva para el desarrollo de algoritmos de lógica difusa aplicable a reconocimiento e identificación de imágenes con patrones difusos, procesado de señales y desarrollo de procesos inteligentes y adaptativos.

Robust Control Toolbox

Herramientas para el diseño y modelado de sistemas de control multivariable mediante técnicas H∞ donde la robustez del sistema es un factor crítico.

μ-Analysis and Synthesis Toolbox

Algoritmos para μ aplicables al análisis de rendimiento y de robustez en sistemas con incertidumbres en el modelado y parametrización.

LMI Control Toolbox

Algoritmos de optimización para la resolución de inecuaciones matriciales lineales (LMI) presentes en control robusto, en control multiobjetivo, etc.

Model Predictive Control Toolbox

Completo conjunto de herramientas para implantar estrategias de control predictivo.

Model-Based Calibration Toolbox

Conjunto de herramientas para la calibración de sistemas motrices complejos. Cubre las tres fases de la calibración: planificación de la prueba física en el dinamómetro, utilización de los datos medidos para producir una serie de modelos de respuesta que reflejen el comportamiento del motor y empleo de estos modelos para calibrar el sistema motriz.

2

Instalación y entorno de trabajo de MATLAB

2.1 Requisitos mínimos

En cuanto al hardware, el programa exige para el correcto funcionamiento unas características mínimas muy básicas, que actualmente están al alcance de cualquier computador de sobremesa. Basta con disponer de un PC-Compatible con microprocesador Pentium o AMD para trabajar sobre Windows 95, 98, Me, NT, XP o Windows 2000, con tarjeta gráfica VGA y monitor color (opcional para gráficos en alta resolución). También son imprescindibles 64 megabytes de memoria RAM (recomendables 128 megabytes), disco duro con un espacio libre de un gigabyte si se va a utilizar todo el sistema (si se utilizan solamente módulos aislados, el propio programa de instalación indica el espacio en disco necesario como mínimo), un ratón y unidad de CD-ROM.

En cuanto al software, el programa exige disponer del sistema operativo Windows 95, 98, Me, NT, XP o Windows 2000. También existen versiones del programa para UNIX (Sun Solaris, HP UX, HP 700, Compaq Alpha), Linux y MAC. Si se va a utilizar MATLAB Notebook, es necesario disponer de Microsoft Word 7.0 o versión superior (Office 2000 o XP). Si se quieren construir ficheros MEX propios es necesario disponer de Microsoft Visual C/C++ (versión 5.0 o superior), de Borland C++ (versión 5.0 o superior), de Borland C++ Builder (versión 3.0 o superior), de Compaq Visual Fortram 5.0 (o superior) o de Lcc 2.4 (propio de MATLAB). También es necesario disponer de Adobe Acrobat Reader si se quiere ver la ayuda en línea de MATLAB en formato PDF.

MATLAB también puede ser operativo en una red con protocolo TCP/IP.

2.2 Instalación de MATLAB

Para instalar el programa en Windows colocamos el CD en la unidad correspondiente del computador. Al cerrar la unidad lectora, automáticamente se inicia el proceso de instalación con la pantalla temporal de la Figura 2-1. Durante unos segundos el programa prepara el proceso de instalación y a continuación presenta la pantalla de la figura Figura 2-2.

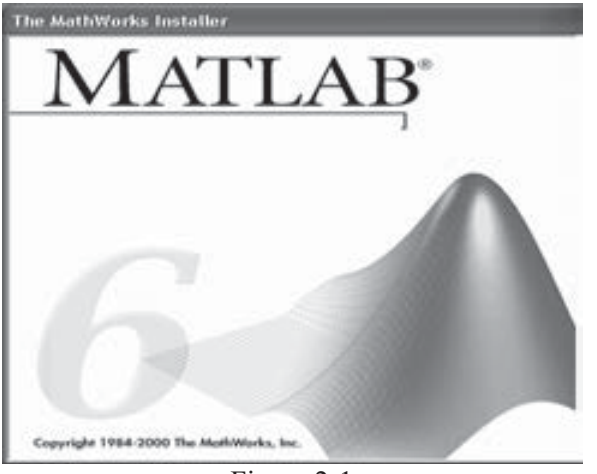


Figura 2-1

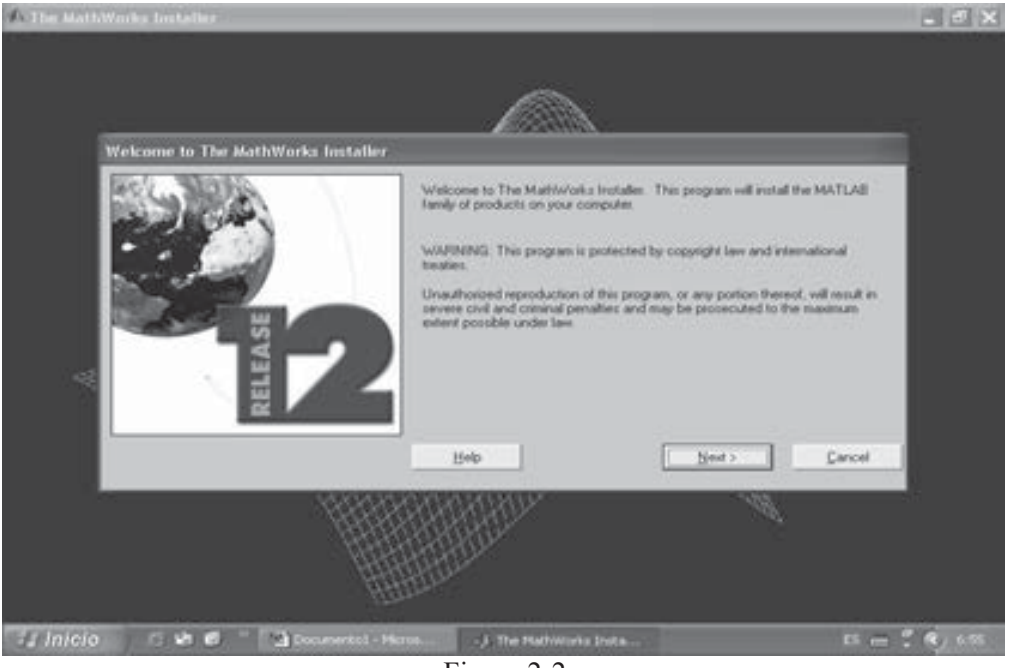


Figura 2-2

Al pulsar *Next* se presenta la pantalla de la Figura 2-3, en la que se introduce la clave PLP del CD-ROM suministrada por Math Works. Después de introducir la clave, se pulsa *Next* y se obtiene la pantalla de licencia de la Figura 2-4. Si se está de acuerdo con las condiciones de la licencia, se pulsa *Yes* y se obtiene la pantalla de identificación de la Figura 2-5, en la que se introduce el nombre del usuario y la compañía. Una vez introducidos los datos adecuados, se pulsa *Next* y se obtiene la pantalla de la Figura 2-6, que permite seleccionar el directorio en el que se instalará MATLAB, opciones de instalación adicional de la documentación, opciones de idioma y productos componentes de MATLAB a instalar. En la parte derecha de la Figura 2-6 se informa del espacio disponible en disco y del espacio requerido para instalar los productos seleccionados. Al pulsar *Next*, el programa de instalación crea el directorio seleccionado para MATLAB (Figura 2-7).

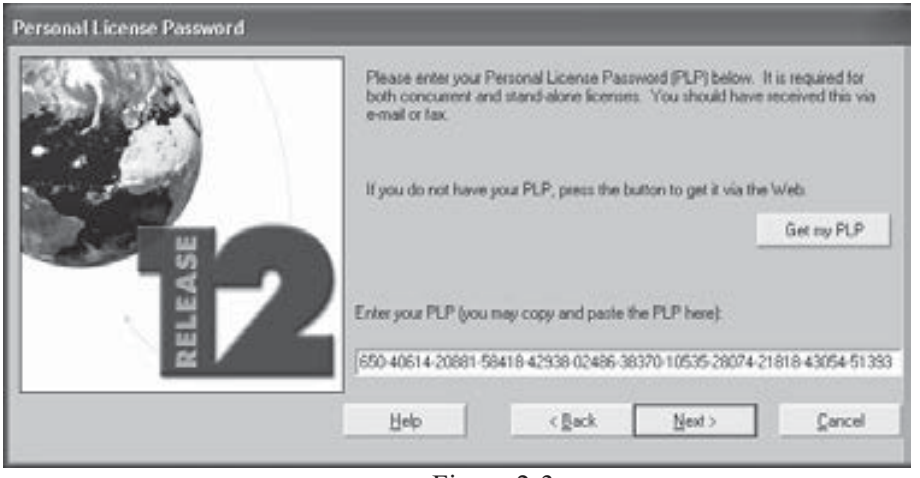


Figura 2-3

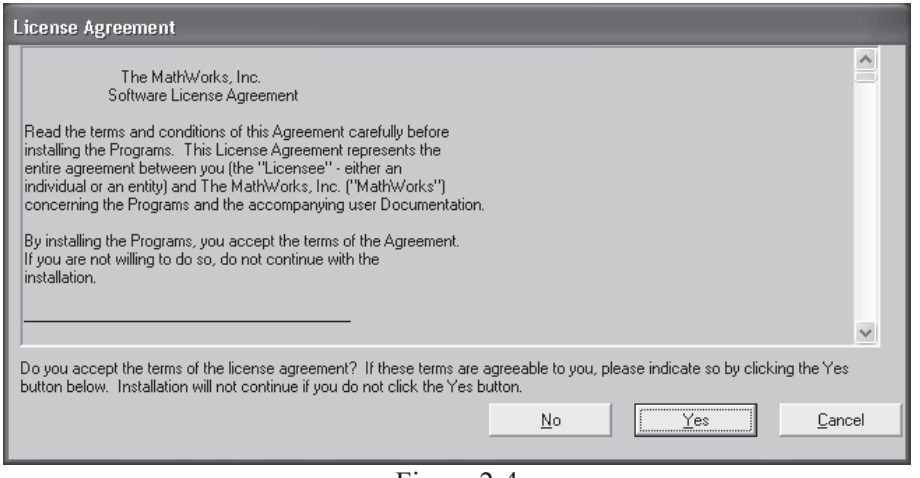


Figura 2-4

Figura 2-5

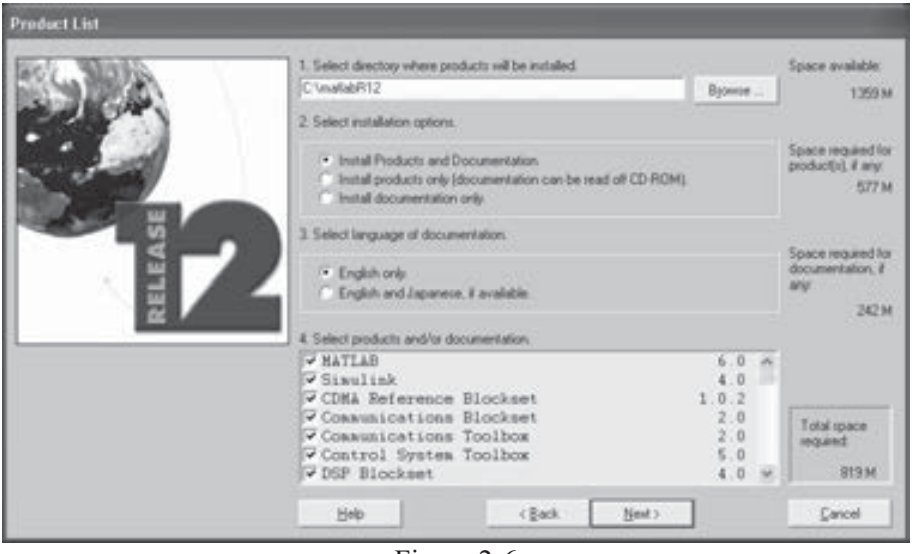


Figura 2-6

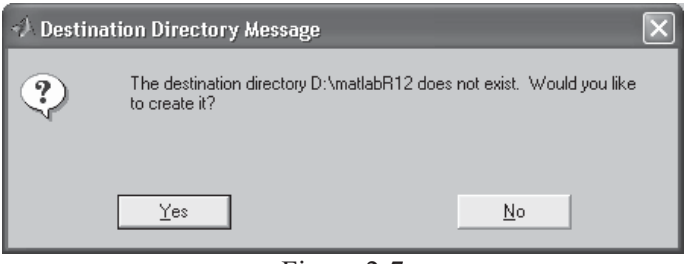


Figura 2-7

Una vez aceptado el directorio, se pulsa *Yes* en la Figura 2-7 y comienza el proceso de instalación (Figura 2-8). Automáticamente van apareciendo pantallas que informan de la instalación de las diversas componentes de MATLAB (Figuras 1-9 a 1-17).

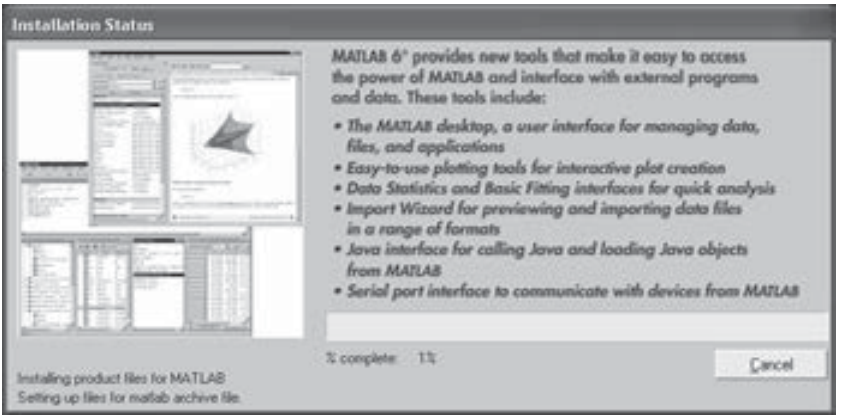


Figura 2-8

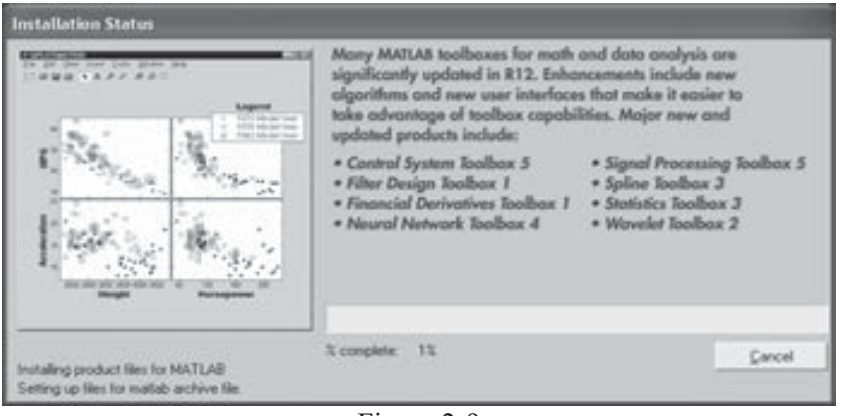


Figura 2-9

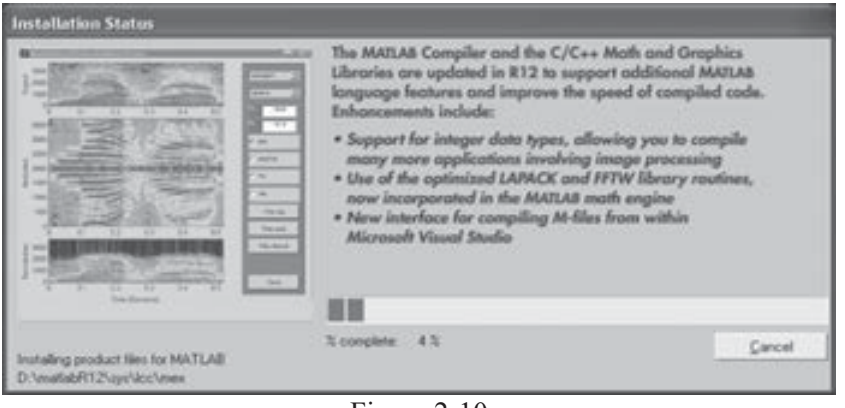


Figura 2-10

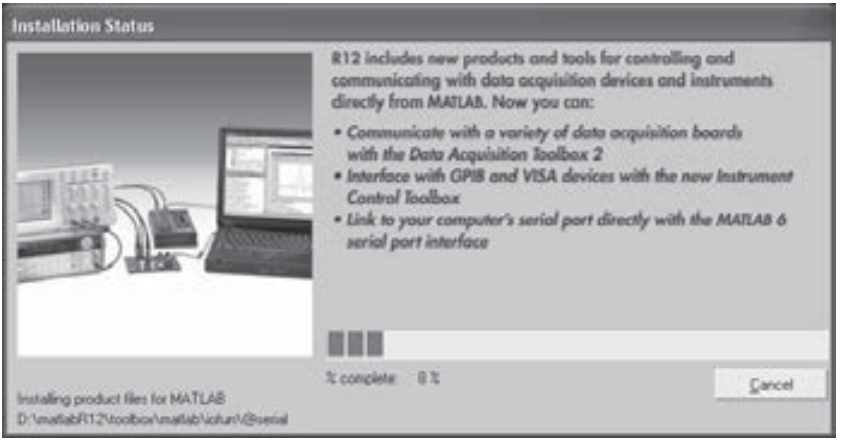


Figura 2-11

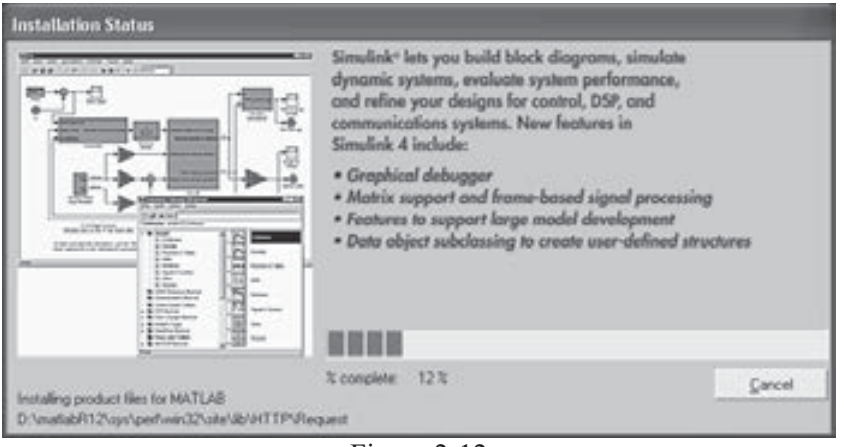


Figura 2-12

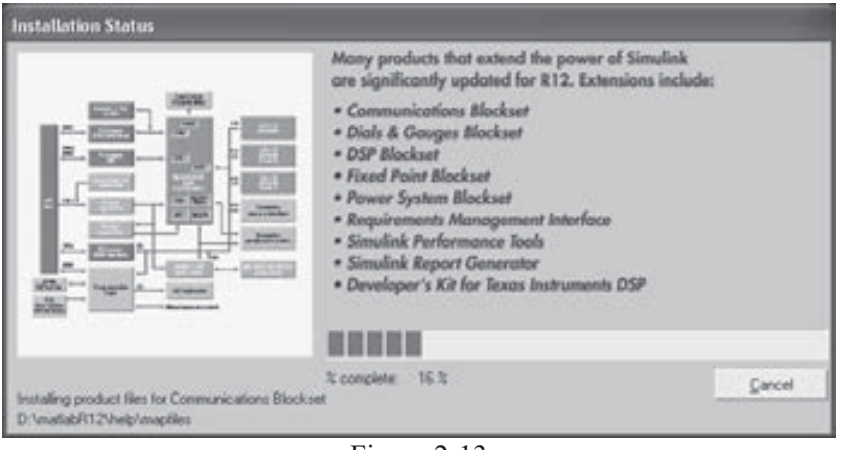


Figura 2-13

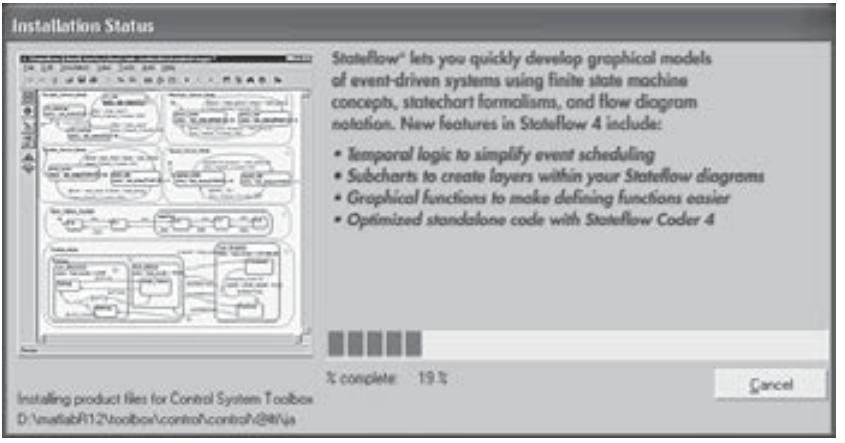


Figura 2-14

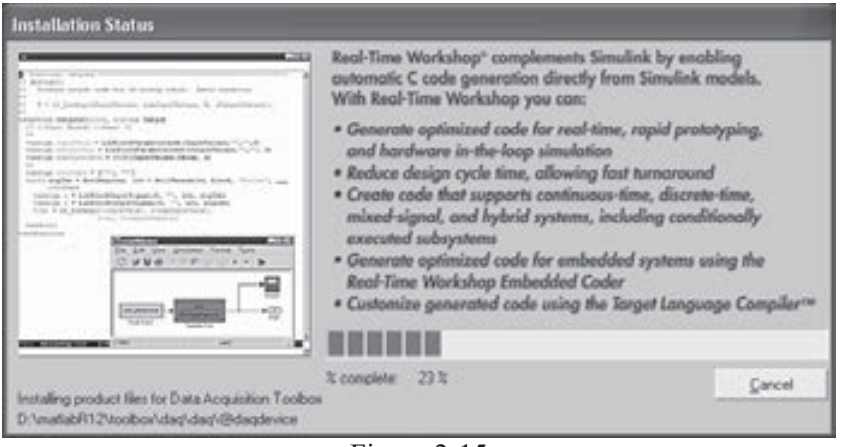


Figura 2-15

Figura 2-16

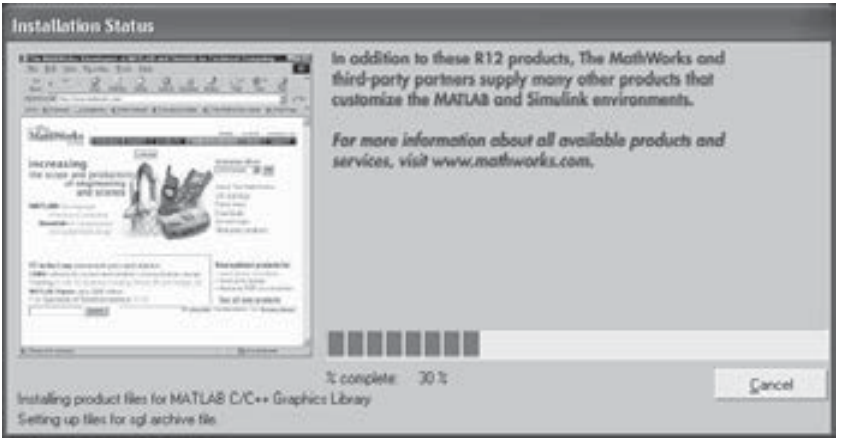


Figura 2-17

En caso de haber seleccionado la instalación de la documentación de MATLAB, será necesario introducir el CD número 2 del programa para instalar en disco la citada documentación (Figura 2-18). Al pulsar *OK* se instalan los archivos de ayuda y se obtiene la pantalla informativa de la Figura 2-19. Al pulsar *Next* se obtiene la pantalla de la Figura 2-20, que indica la finalización del proceso de instalación con éxito y que da la opción de rearrancar el computador en este instante para memorizar la configuración de MATLAB o hacerlo más tarde.

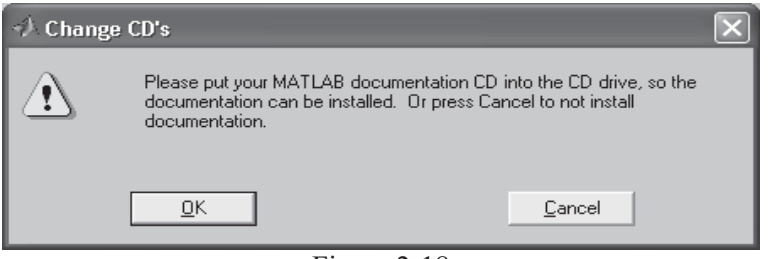


Figura 2-18

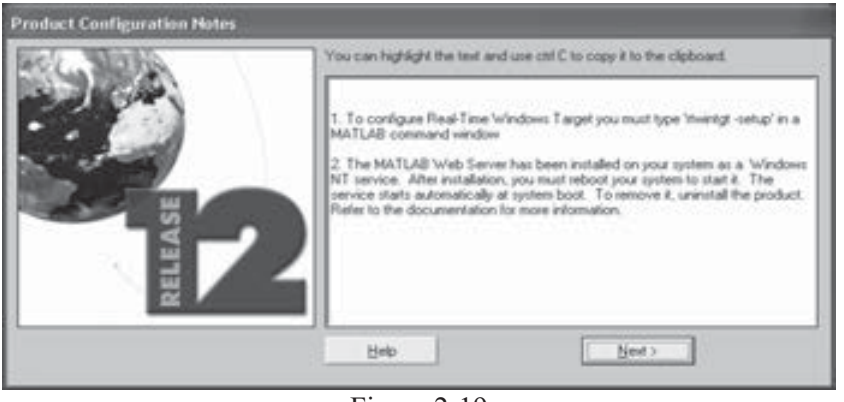


Figura 2-19

Figura 2-20

Una vez concluida la tarea de instalación de MATLAB y reiniciado el sistema, se dispone ya del programa instalado al elegir el botón *Inicio* y seleccionar *Programas*, tal y como se indica en la Figura 2-21. También se observa un acceso directo a MATLAB en el escritorio, que se crea automáticamente (Figura 2-22).

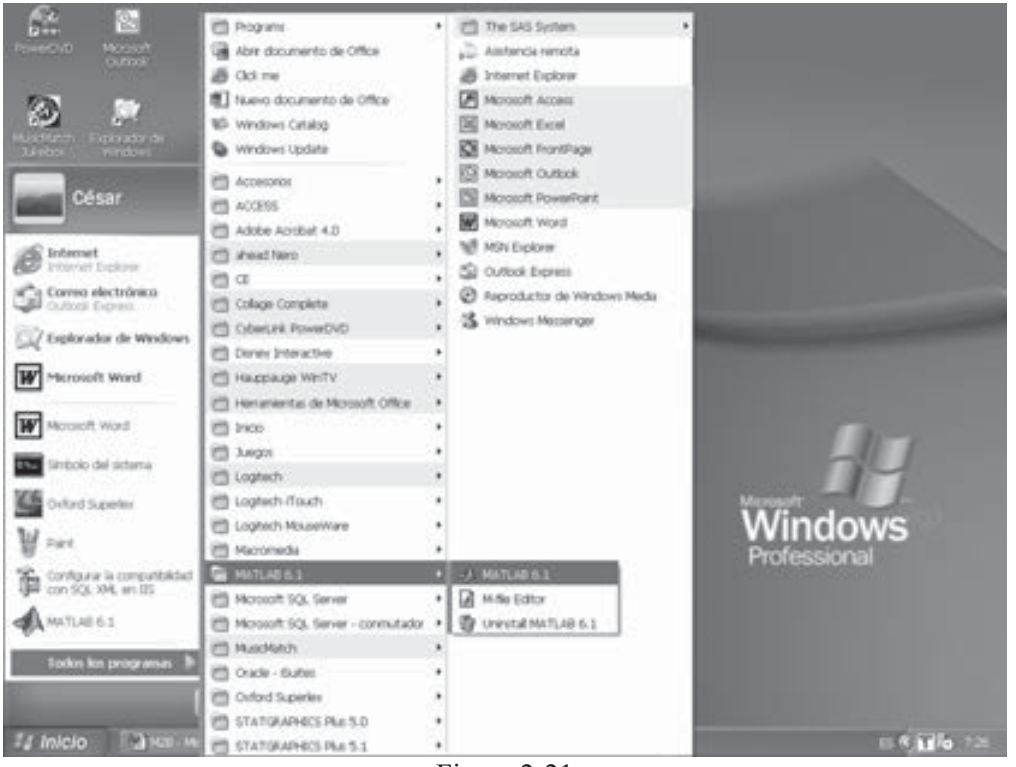


Figura 2-21

Figura 2-22

2.3 Comenzando con MATLAB en Windows

Para comenzar con MATLAB, basta hacer doble clic en el icono de acceso directo al programa situado en el Escritorio de Windows (Figura 2-22). Alternativamente, si no existe icono de acceso directo en el escritorio, lo más común y sencillo para ejecutar el programa es elegir la opción *Programas* del menú *Inicio* de Windows y seleccionar la opción *MATLAB Release 12→ MATLAB R12* (Figura 2-21). De cualquiera de las formas que ejecutemos MATLAB, el programa presenta la pantalla temporal de la Figura 2-23 y a continuación la pantalla inicial del programa de la Figura 2-24.

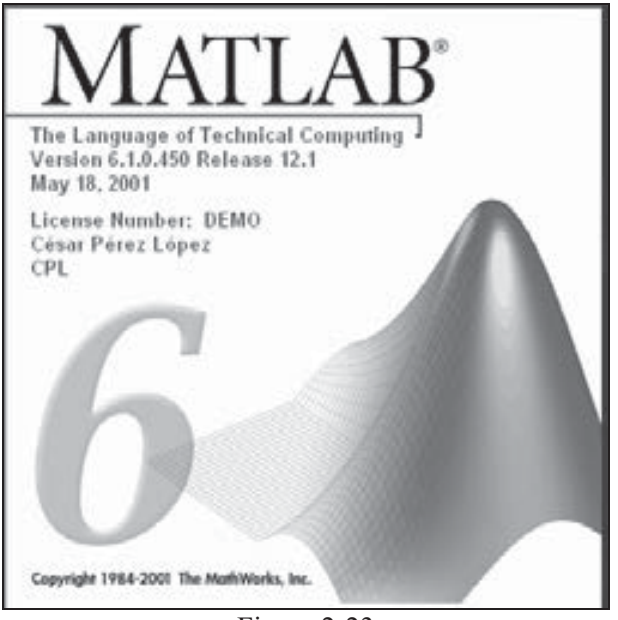


Figura 2-23

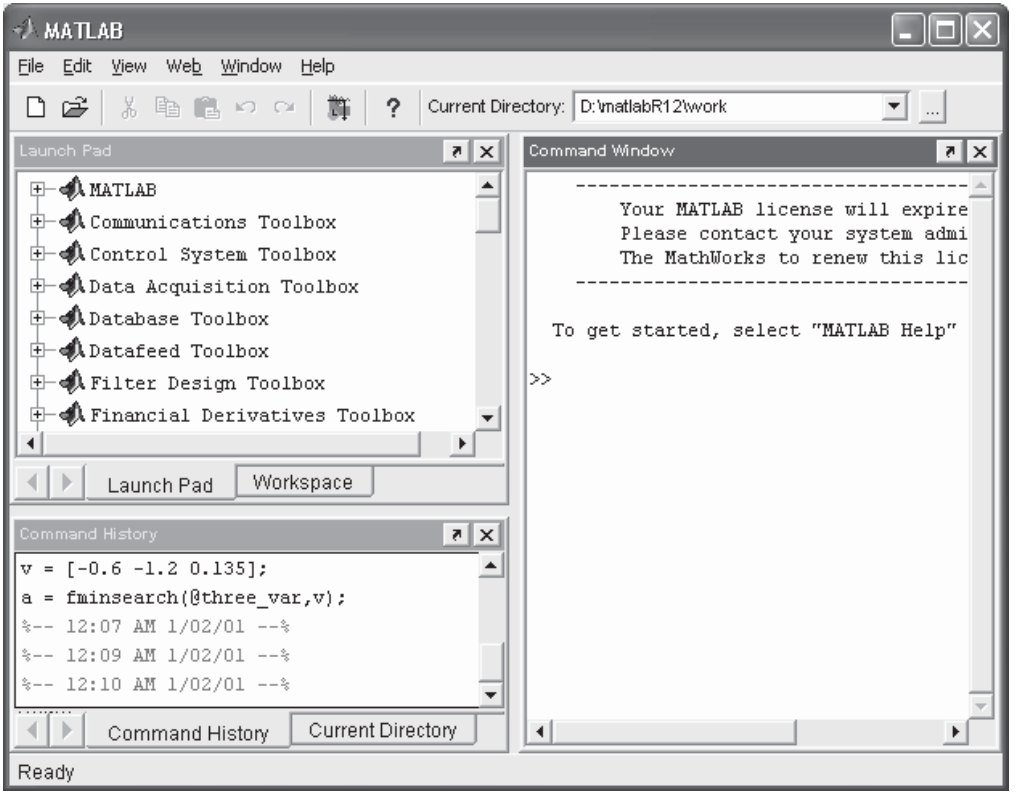


Figura 2-24

2.4 Entorno de trabajo de MATLAB

La pantalla de entrada de MATLAB (Figura 2-24) constituye el marco de trabajo general del programa. Los elementos más importantes de esta pantalla de inicio de MATLAB (Figura 2-25) son los siguientes:

- *Command Window (ventana de comandos):* Ejecuta las funciones MATLAB.
- *Command History (historial de comandos):* Presenta una historia de todas las funciones introducidas en la ventana de comandos y permite copiarlas y ejecutarlas.
- *Launch Pad:* Ejecuta herramientas y documentación de acceso para todos los productos de MathWorks instalados actualmente en el computador.
- *Current Directory (directorio actual):* Muestra ficheros MATLAB y ejecuta operaciones de ficheros tales como abrir y buscar contenido.

- *Help (ayuda):* Muestra y busca documentación para la familia completa de productos MATLAB.
- *Workspace (espacio de trabajo):* Muestra y realiza cambios en el contenido del espacio de trabajo.
- *Array Editor:* Muestra contenido de arrays en formato de tabla y edita sus valores.
- *Editor/Debugger:* Crea, edita y comprueba M-ficheros (ficheros que contienen sintaxis de funciones MATLAB).

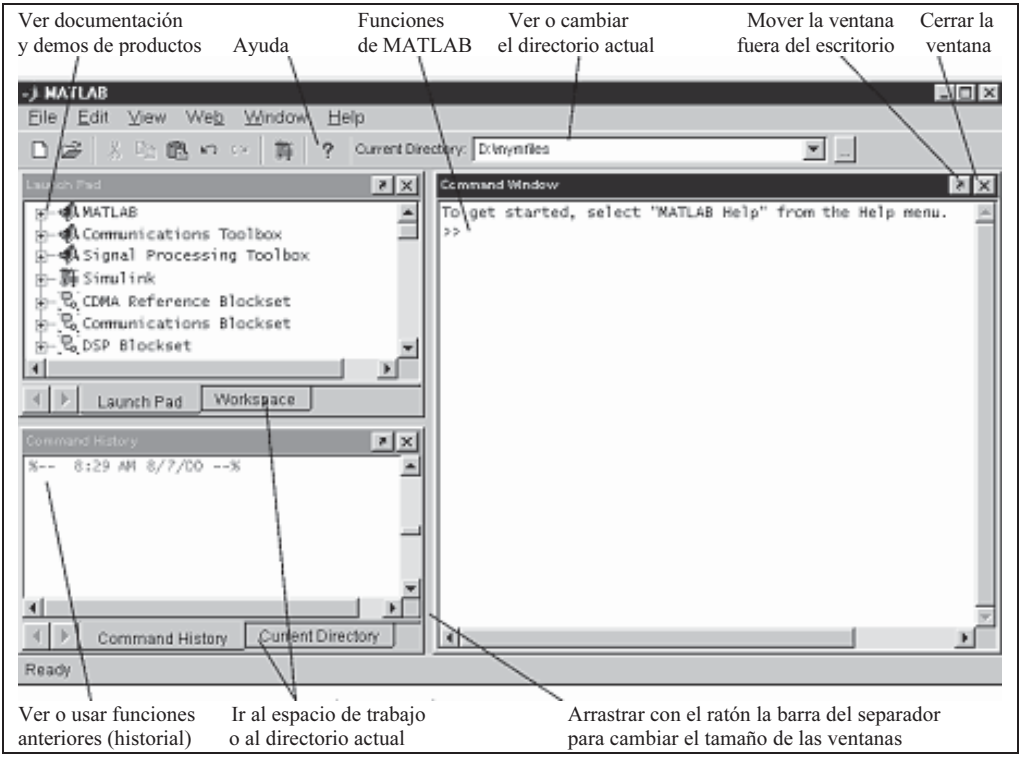


Figura 2-25

La ventana de comandos de MATLAB

La ventana de comandos (Figura 2-26) es el camino principal para comunicarse con MATLAB. Aparece en el escritorio cuando se inicia MATLAB y se utiliza para ejecutar funciones y todo tipo de operaciones. Las entradas a ejecutar se escriben a continuación del prompt >> y, una vez completadas, se pulsa *Enter*. En la primera línea de la Figura 2-27 se define una matriz y, al pulsar *Enter,* se obtiene como salida la propia matriz.

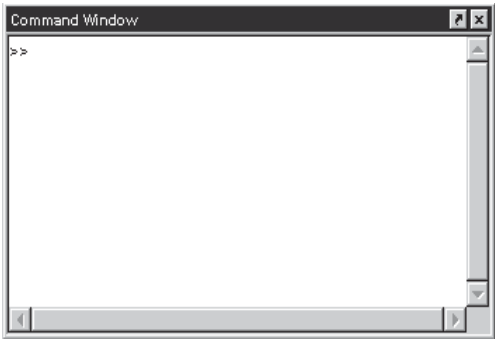


Figura 2-26

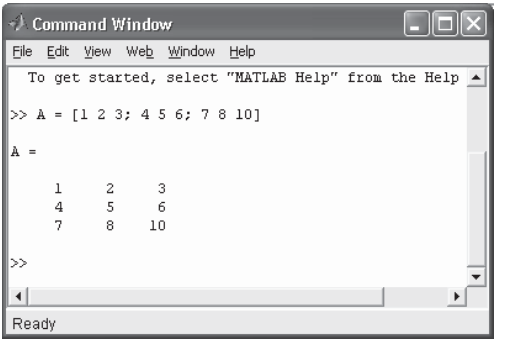


Figura 2-27

Sobre la ventana de comandos es posible evaluar los ya ejecutados anteriormente. Para ello basta seleccionar la zona de sintaxis a evaluar, hacer clic con el botón derecho del ratón y elegir la opción *Evaluate Selection* del menú emergente resultante (Figuras 1-28 y 1-29). La opción *Open Selection* de este mismo menú permite abrir en el *Editor/Debugger* un M-fichero previamente seleccionado en la ventana de comandos (Figuras 1-30 y 1-31).

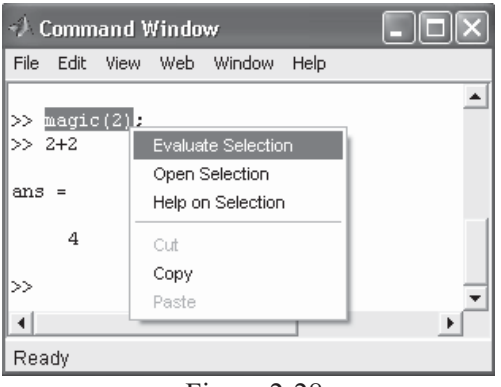


Figura 2-28

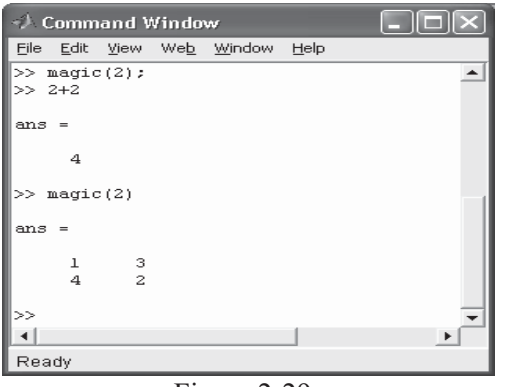


Figura 2-29

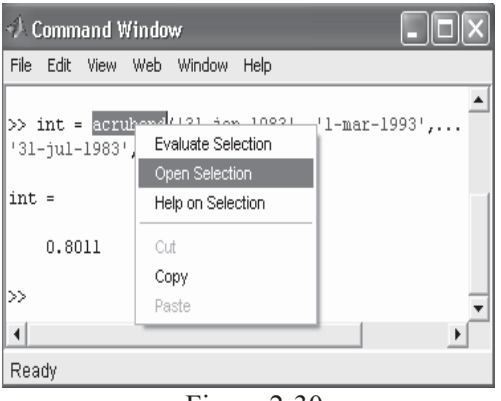


Figura 2-30

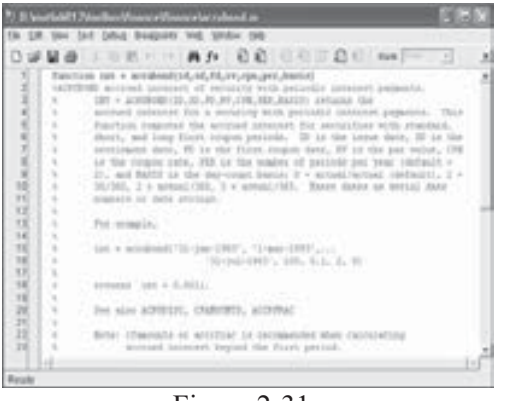


Figura 2-31

MATLAB es sensible al uso de mayúsculas y minúsculas, pero permite situar o no espacios en blanco antes y después del signo menos, de los dos puntos y de los paréntesis. MATLAB también permite escribir varias entradas sobre la misma línea, pero separadas por punto y coma (Figura 2-32). Las entradas se ejecutan todas secuencialmente según están colocadas en la línea, pero sólo se ofrece la salida de la última, siempre y cuando no finalice también en punto y coma. Cualquier entrada que tenga punto y coma detrás se ejecuta al pulsar *Enter*, pero no se ve su salida.

Las entradas largas que no quepan en una línea pueden continuarse en la línea siguiente situando puntos suspensivos al final de la línea (Figura 2-33).

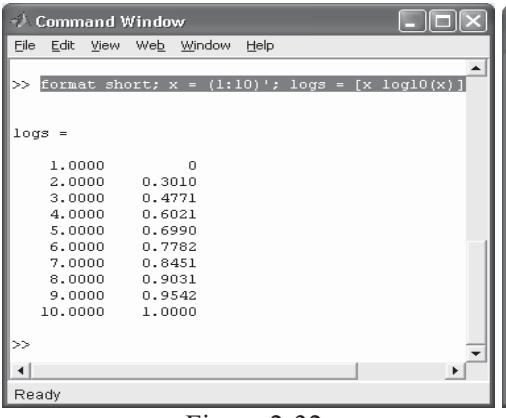


Figura 2-32

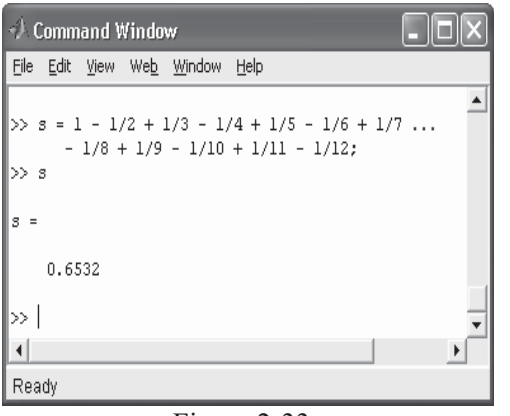


Figura 2-33

La opción *Clear Command Window* del menú *Edit* (Figura 2-34) permite limpiar la ventana de comandos. El comando *clc* también realiza esta función (Figura 2-35). Del mismo modo, las opciones *Clear Command History* y *Clear Workspace* del menú *Edit* permiten limpiar la ventana historial y la del espacio de trabajo.

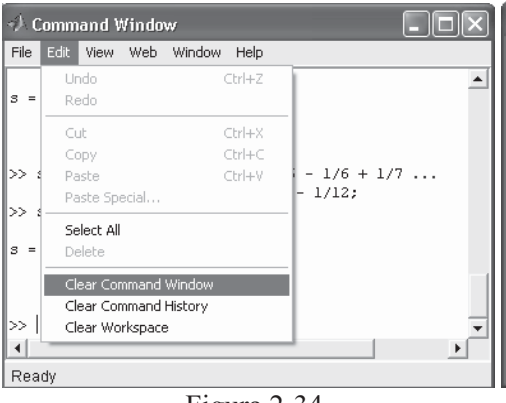


Figura 2-34

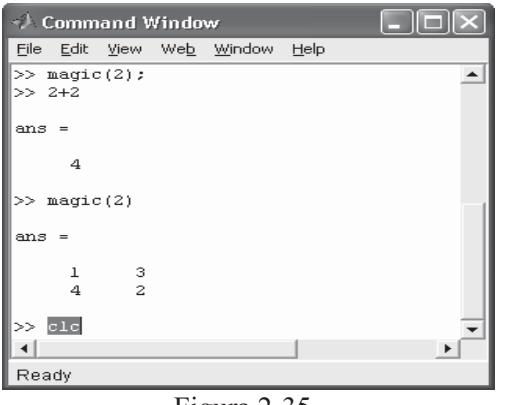


Figura 2-35

Algunas entradas aparecen en diferentes colores en la ventana de comandos para ayudar a interpretar mejor determinados elementos como instrucciones *if/else*, cadenas, etc. Algunas de las reglas existentes para los colores son las siguientes:

1. Las cadenas aparecen en color púrpura mientras se teclean. Al finalizarlas correctamente (con la comilla de cierre) se vuelven de color marrón.
2. La sintaxis de control de flujo aparece en color azul. Todas las líneas entre la apertura y el cierre de las funciones de control de flujo aparecen correctamente sangradas.
3. Los paréntesis, corchetes y llaves se mantienen brevemente iluminados hasta que no se finalice la escritura de su contenido. Esto permite ver si se cierran correctamente o no en las expresiones matemáticas.
4. Al introducir el símbolo % que precede a un comentario en la ventana de comandos la escritura es de color verde.
5. Los comandos de sistema tales como ! aparecen en color dorado.
6. Los errores aparecen en color rojo.

A continuación se presenta una lista de teclas, flechas y sus combinaciones que pueden utilizarse en la ventana de comandos.

Tecla	*Tecla de control*	*Operación*
↑	**Ctrl+p**	*Llama a la última entrada submitida.*
↓	**Ctrl+n**	*Llama a la línea siguiente.*
←	**Ctrl+b**	*Mueve un carácter hacia atrás.*
→	**Ctrl+f**	*Mueve un carácter hacia adelante.*
Ctrl+→	**Ctrl+r**	*Mueve una palabra hacia la derecha.*
Ctrl+←	**Ctrl+l**	*Mueve una palabra hacia la izquierda.*
Home	**Ctrl+a**	*Mueve al comienzo de la línea.*
End	**Ctrl+e**	*Mueve al final de la línea.*
Esc	**Ctrl+u**	*Borra la línea.*
Delete	**Ctrl+d**	*Borra el carácter en el que está el cursor.*
Backspace	**Ctrl+h**	*Borra el carácter anterior al cursor.*
	Ctrl+k	*Borra hasta el final de línea.*
Shift+home		*Ilumina hasta el comienzo de la línea.*
Shift+end		*Ilumina hasta el final de la línea.*

Para introducir comentarios explicativos basta comenzarlos con el símbolo % en cualquier punto de una línea. El resto de la línea debe utilizarse para comentario (Figura 2-36).

Para ejecutar M-ficheros (ficheros que contienen código en lenguaje de MATLAB) se sigue el mismo camino que para ejecutar cualquier otro comando o función. Basta con teclear el nombre del M-fichero (con sus argumentos, si es necesario) en la ventana de comandos y pulsar *Enter* (Figura 2-37). Para ver cada función de un M-fichero según se ejecuta, basta con submitir antes *echo on*. Para interrumpir la ejecución de un M-fichero se utiliza *Ctrl+c* o *Ctrl+Break*.

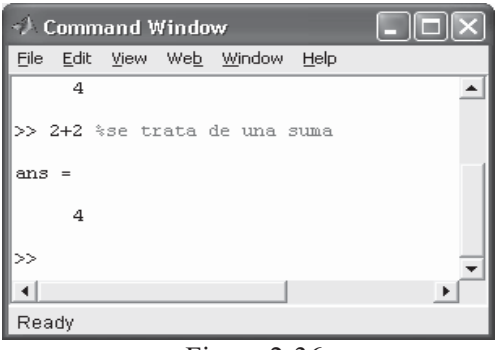


Figura 2-36

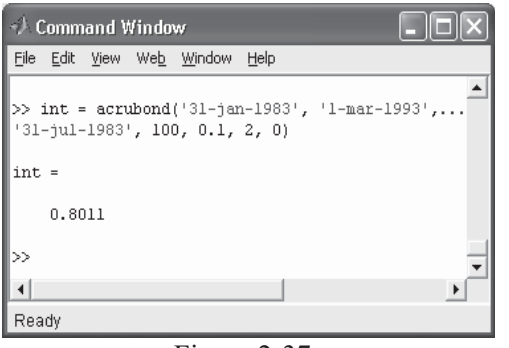


Figura 2-37

Comandos de escape y salida al entorno DOS

Existen tres formas de pasar desde la ventana de comandos de MATLAB al entorno del sistema operativo MS-DOS para ejecutar tareas temporales.

El comando *!orden_dos* introducido en la ventana de comandos permite ejecutar la orden especificada en ambiente MATLAB. La Figura 2-38 hace referencia a la ejecución del comando *!dir*. El mismo efecto se consigue con el comando *dos orden_dos* (Figura 2-39).

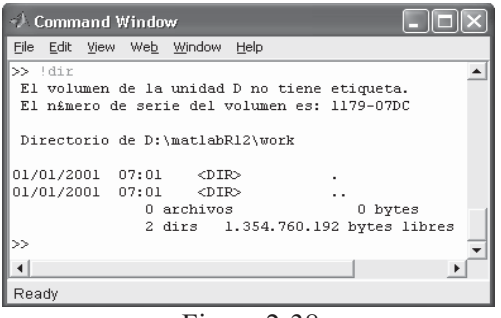


Figura 2-38

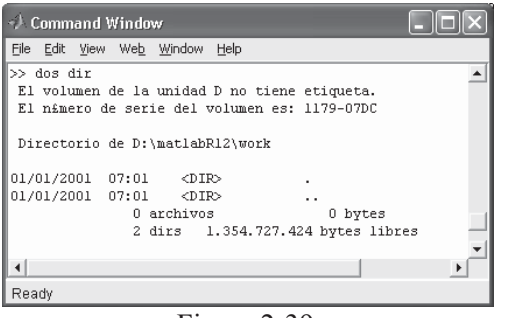


Figura 2-39

El comando *!orden_dos &* se utiliza para ejecutar la orden del DOS en modo background. La orden se ejecuta abriendo una ventana de ambiente DOS sobre la ventana de comandos de MATLAB (Figura 2-40). Para volver a ambiente MATLAB basta con pulsar con el ratón en cualquier zona de la ventana de comandos o cerrar la ventana del DOS con su botón cerrar ✖ o con el comando *Exit*.

Con los tres comandos anteriores no sólo pueden ejecutarse comandos del DOS, sino también todo tipo de ficheros ejecutables o tareas batch. Para salir definitivamente de MATLAB basta con teclear *quit* o *exit* en la ventana de comandos y a continuación pulsar *Enter*. También puede usarse la opción *Exit MATLAB* del menú *File* (Figura 2-41).

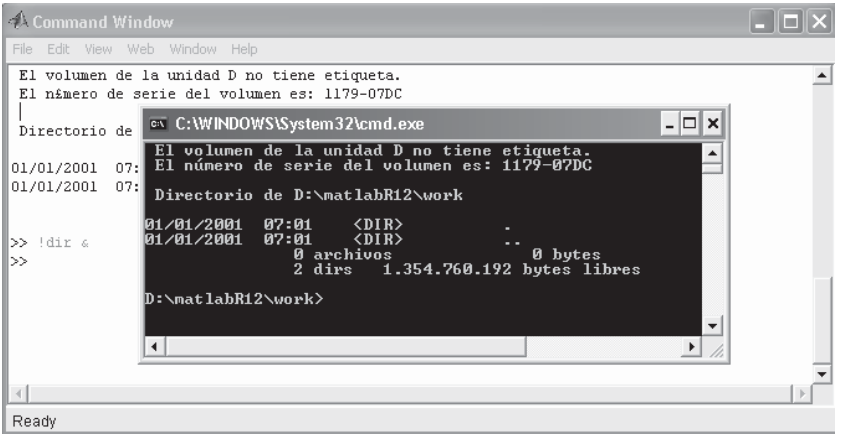


Figura 2-40

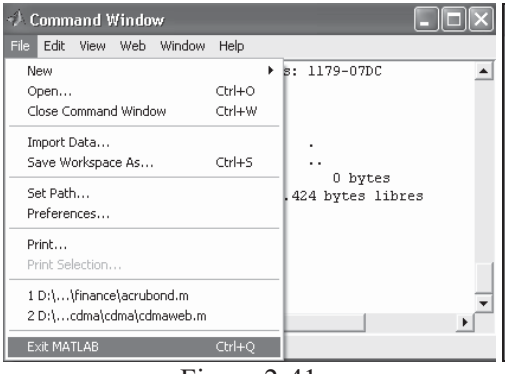


Figura 2-41

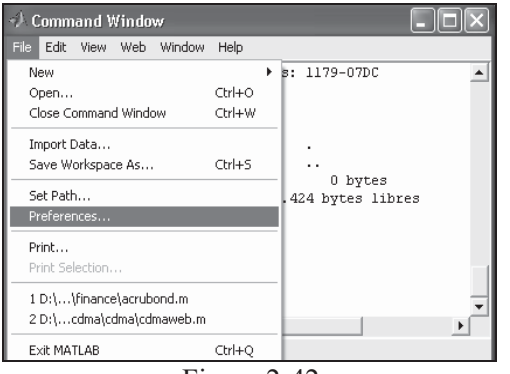


Figura 2-42

Preferencias para la ventana de comandos

La opción *Preferences* del menú *File* (Figura 2-42) permite establecer determinadas características para el trabajo en la ventana de comandos. Para ello basta con utilizar las opciones de la ventana *Command Window Preferences* de la Figura 2-43.

La primera zona que aparece en la ventana *Command Window Preferences* es *Text display,* que especifica cómo aparece la salida en la ventana de comandos. Sus opciones son las siguientes:

- *Numeric format:* Especifica el formato de los valores numéricos en la ventana de comandos (Figura 2-44). Afecta sólo a la forma de mostrar los números, pero no a los cálculos ni a la forma de guardarlos. Los posibles formatos se presentan en la tabla siguiente:

Formato	***Resultado***	***Ejemplo***
`+`	*+, -, blanco*	`+`
`bank`	*Fijo*	`3.14`

`compact`	*Suprime el exceso de líneas mostradas en la pantalla para presentar así una salida más compacta*	`theta = pi/2`
		`theta=`
		`1.5708`

`hex`	*Hexadecimal*	`400921fb54442d18`
`long`	*15 dígitos con punto fijo*	`3.14159265358979`
`long e`	*15 dígitos con punto flotante*	`3.141592653589793e+00`
`long g`	*El mejor de los dos anteriores*	`3.14159265358979`

`loose`	*Añade líneas para hacer la salida más legible. Lo cantrario de* `compact`	`theta = pi/2`
		
		`theta=`
		
		`1.5708`

`rat`	*Razón de enteros pequeños*	`355/113`
`short`	*5 dígitos con punto fijo*	`3.1416`
`short e`	*5 dígitos con punto flotante*	`3.1416e+00`
`short g`	*El mejor de los dos anteriores*	`3.1416`

- *Numeric display:* Regula el espaciado de la salida en la ventana de comandos. Para suprimir líneas en blanco se usa `compact`. Para mostrar líneas en blanco se usa `loose`.

- *Spaces per tab:* Regula el número de espacios asignados al tabulador cuando se muestra la salida (el valor por defecto res 4).

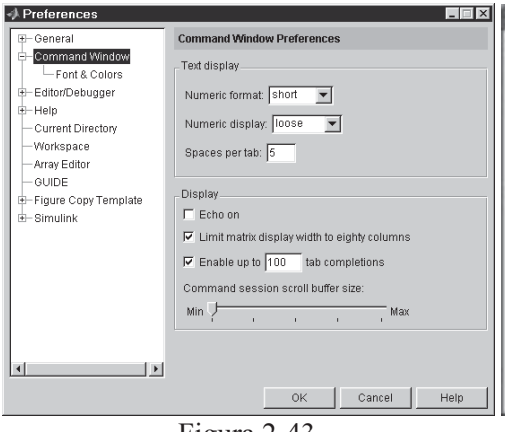


Figura 2-43

Figura 2-44

La segunda zona que aparece en la ventana *Command Window Preferences* es *Display,* que especifica el tamaño del buffer y la posibilidad de mostrar o no en pantalla las ejecuciones de todos los comandos incluidos en M-ficheros. Sus opciones son las siguientes:

- *Echo on:* Si se activa esta casilla, se muestran en pantalla las ejecuciones de todos los comandos incluidos en M-ficheros.
- *Limit matrix display width to eighty columns:* Si se activa esta casilla, MATLAB muestra sólo 80 columnas en las salidas matriciales, sea cual sea la anchura de la ventana de comandos. Si no se marca esta casilla, las salidas matriciales ocuparán todo el ancho actual de la ventana de comandos.
- *Enable up to n tab completions:* Se marca esta casilla si se quiere utilizar finalización mediante tabulador cuando se teclean funciones en la ventana de comandos. Es necesario introducir un límite de finalizaciones por encima del cual MATLAB no muestra la lista de finalizaciones.
- *Command session scroll buffer size:* Sitúa el tamaño del búfer que mantiene la lista de comandos previos ejecutados para ser utilizados por el comando *recall.*

En MATLAB también es posible situar preferencias de fuentes y colores para la ventana de comandos. Para ello basta desdoblar la subopción *Font & Colors* que cuelga de *Command Windows* (Figura 2-45). En la zona *Fuentes* se señala *Use desktop font* si se quiere utilizar la misma fuente que la especificada para *General Font & Colors preferences*. Para usar una fuente diferente se utiliza el botón *Use custom font* y en las tres casillas situadas inmediatamente debajo se elige la fuente deseada (Figura 2-46), su estilo (Figura 2-47) y su tamaño. En la zona *Sample* se muestra un ejemplo de la fuente elegida. En la zona *Colors* se puede elegir el color del texto (*Text color*) según la Figura 2-48 y el color del fondo (background color). Si se señala la casilla *Syntax highlighting* se obtiene color destacado. El botón *Set Colors* se utiliza para chequear el color seleccionado.

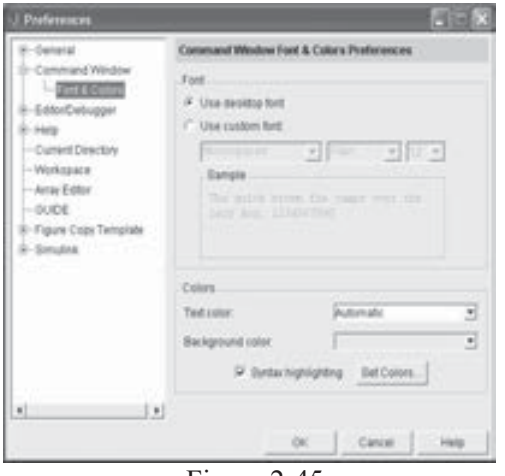


Figura 2-45

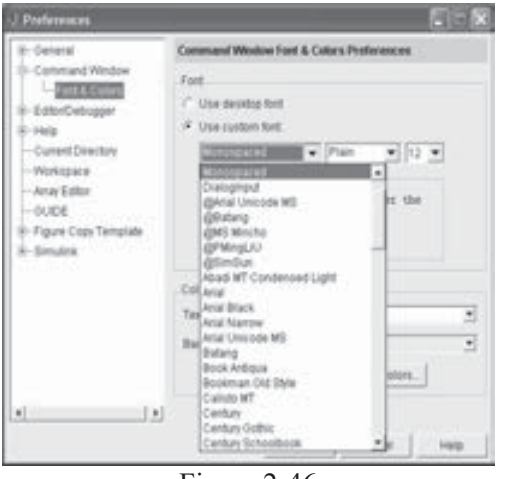


Figura 2-46

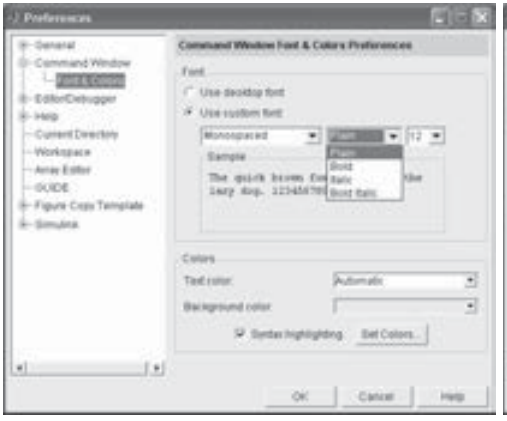


Figura 2-47

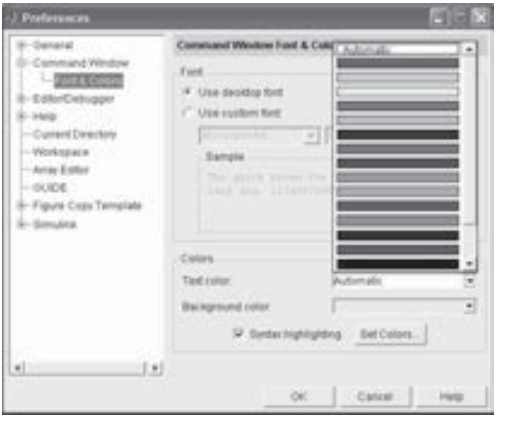


Figura 2-48

Para *mostrar la ventana de comandos separada del escritorio de MATLAB* basta hacer clic en el botón situado en su esquina superior derecha. Para retornar la ventana a su sitio en el escritorio, se utiliza la opción *Dock Command Window* del menú *View* (Figura 2-49).

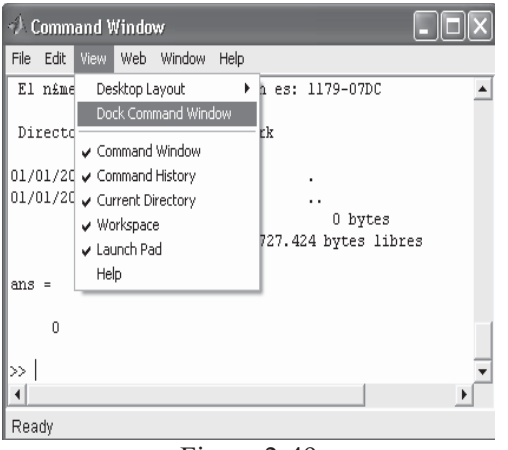


Figura 2-49

Ventana de historial de comandos

La ventana *Command History* (Figura 2-50) aparece cuando se inicia MATLAB, situada en la parte inferior derecha del escritorio de MATLAB de la Figura 2-24. La ventana *Command History* muestra una relación de las funciones utilizadas recientemente en la ventana de comandos (Figura 2-50). También muestra un indicador de comienzo de sesión. Para mostrar esta ventana separada del escritorio de MATLAB basta hacer clic en el botón situado en su esquina superior derecha. Para retornar la ventana a su sitio en el escritorio se utiliza la opción *Dock Command Window* del menú *View*. Este método de separación y acoplamiento de ventanas es común para todas las ventanas de MATLAB.

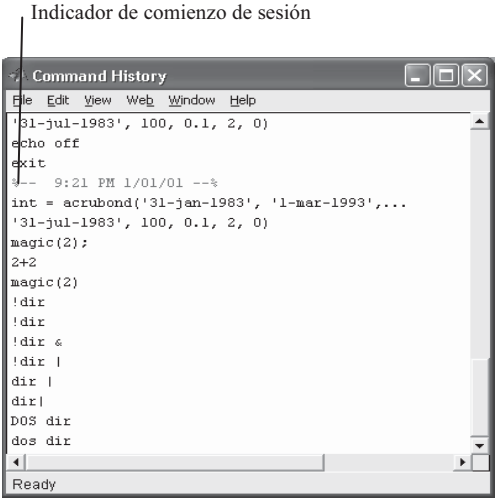


Figura 2-50

Si seleccionamos una o varias líneas de la ventana Historial de comandos y hacemos clic con el botón derecho del ratón sobre la selección, obtenemos el menú emergente de la Figura 2-51, cuyas opciones permiten copiar la selección al portapapeles (*Copy*), evaluar la selección en la ventana de comandos (*Evaluate Selection*), crear un M-fichero con la sintaxis seleccionada (*Create M-File*), borrar la selección (*Delete Selection*), borrar hasta la selección (*Delete to Selection*) y borrar todo el historial (*Delete Entire History*).

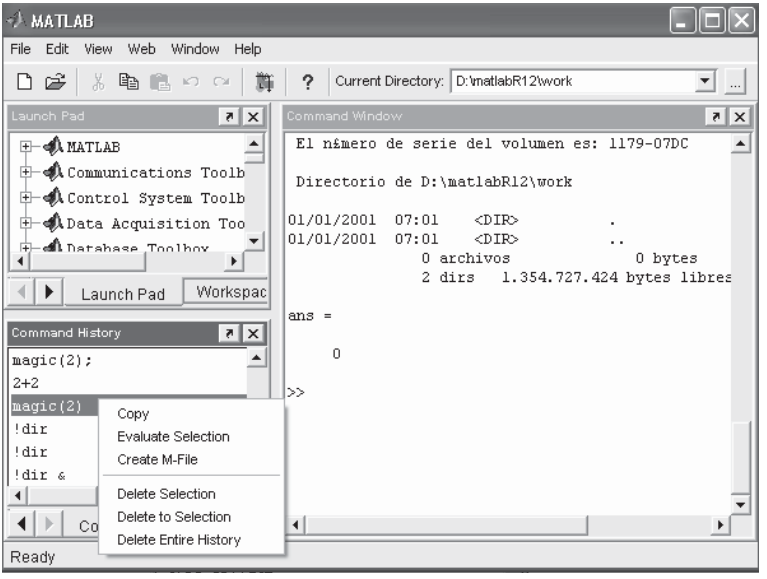


Figura 2-51

Ventana Launch Pad

La ventana *Launch Pad* (situada por defecto en la esquina superior izquierda del escritorio de MATLAB) permite obtener ayuda y ver demostraciones de los productos instalados, así como ir a otras ventanas del escritorio y visitar los sitios Web de MathWorks (Figura 2-52).

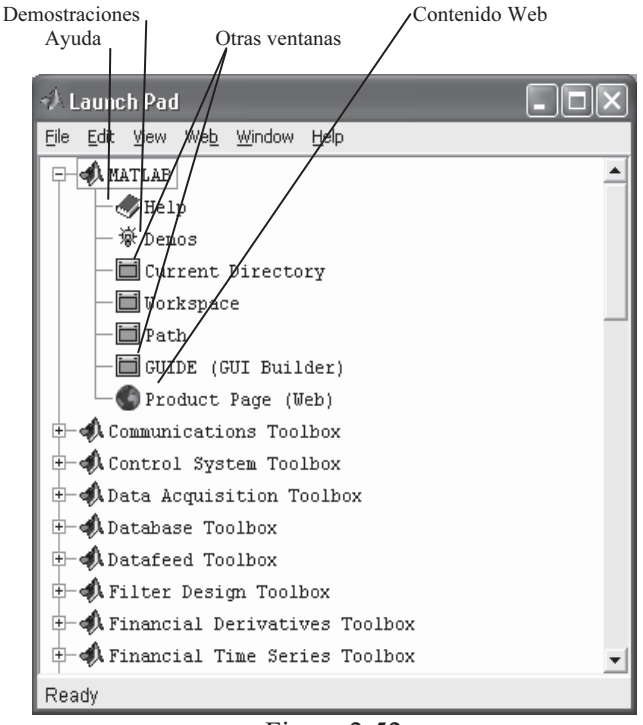


Figura 2-52

Ventana de directorio actual

La ventana de directorio actual se sitúa en la esquina inferior izquierda del escritorio de MATLAB y se obtiene haciendo clic sobre la etiqueta *Current Directory* situada en la parte inferior izquierda del escritorio (Figura 2-53). Su función es ver, abrir y hacer cambios en los ficheros del entorno de MATLAB. Para mostrar esta ventana separada del escritorio de MATLAB (Figura 2-54) basta hacer clic en el botón situado en su esquina superior derecha. Para retornar la ventana a su sitio en el escritorio se utiliza la opción *Dock Command Window* del menú *View*.

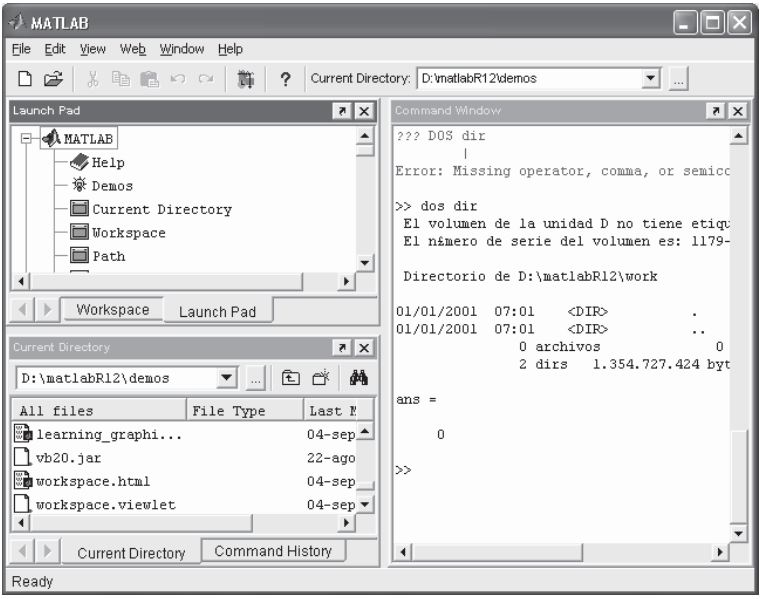


Figura 2-53

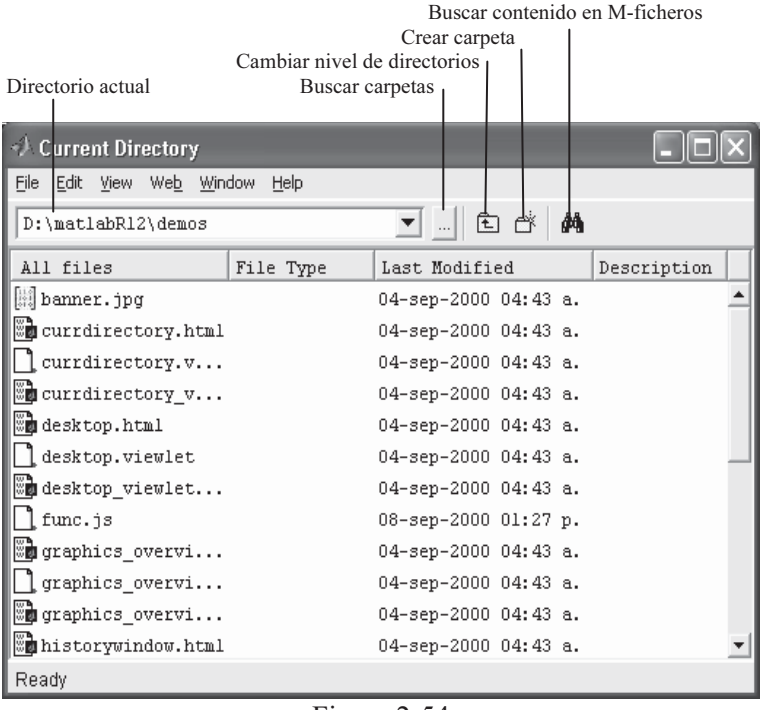


Figura 2-54

Es posible situar preferencias en la ventana de directorio actual mediante la opción *Preferences* del menú File (Figura 2-55). Se obtiene la ventana *Current Directory Preferentes* de la Figura 2-56. En el campo *History* se fija el número de directorios recientes a salvar para el historial. En el campo *Browser display options* se fijan las características a mostrar en los ficheros (tipo de fichero, fecha de la última modificación y descripciones y comentarios de los M-ficheros).

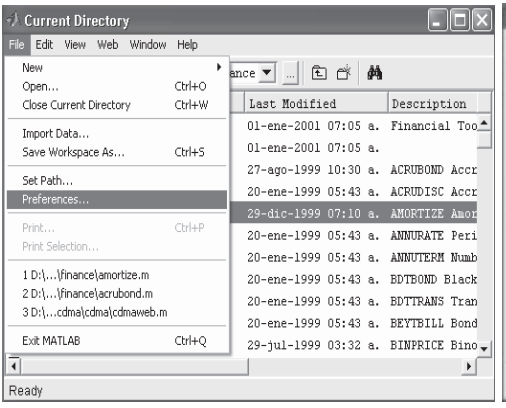


Figura 2-55

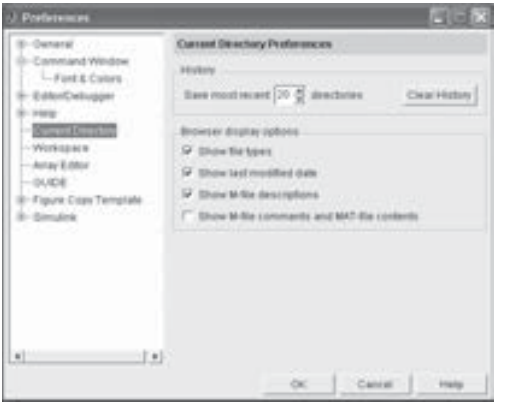


Figura 2-56

Si se selecciona cualquier fichero de la ventana *Current Directory* y se hace clic sobre él con el botón izquierdo del ratón se obtiene el menú emergente de la Figura 2-57, cuyas opciones permiten abrir el fichero (*Open*), ejecutarlo (*Run*), ver ayuda (*View Help*), abrirlo como texto (*Open as Text*), importar datos (*Import Data*), crear nuevos ficheros, M-ficheros o carpetas (*New*), renombrarlo, borrarlo, cortarlo, copiarlo o pegarlo, pasarle filtros y añadirlo al camino actual.

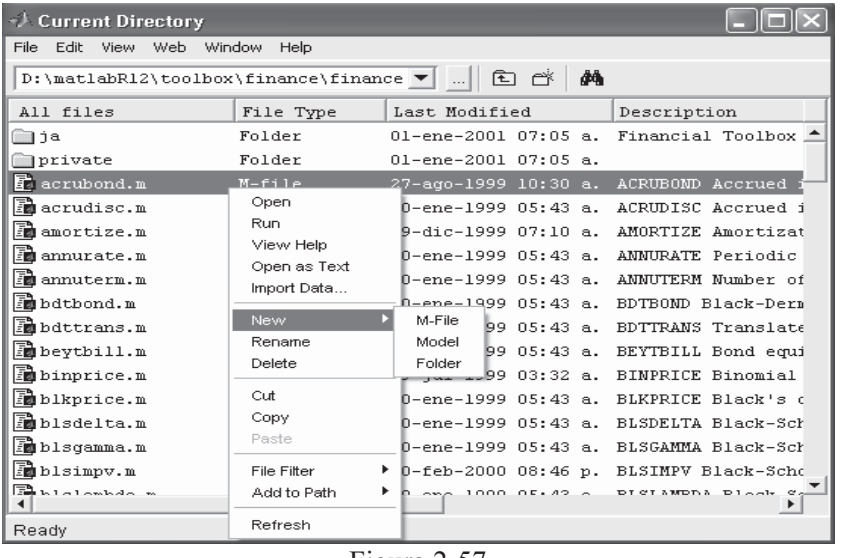


Figura 2-57

Navegador de la ayuda

El navegador de la ayuda de MATLAB (Figura 2-58) se obtiene haciendo clic en el botón de la barra de herramientas o utilizando la función `helpbrowser` en la ventana de comandos.

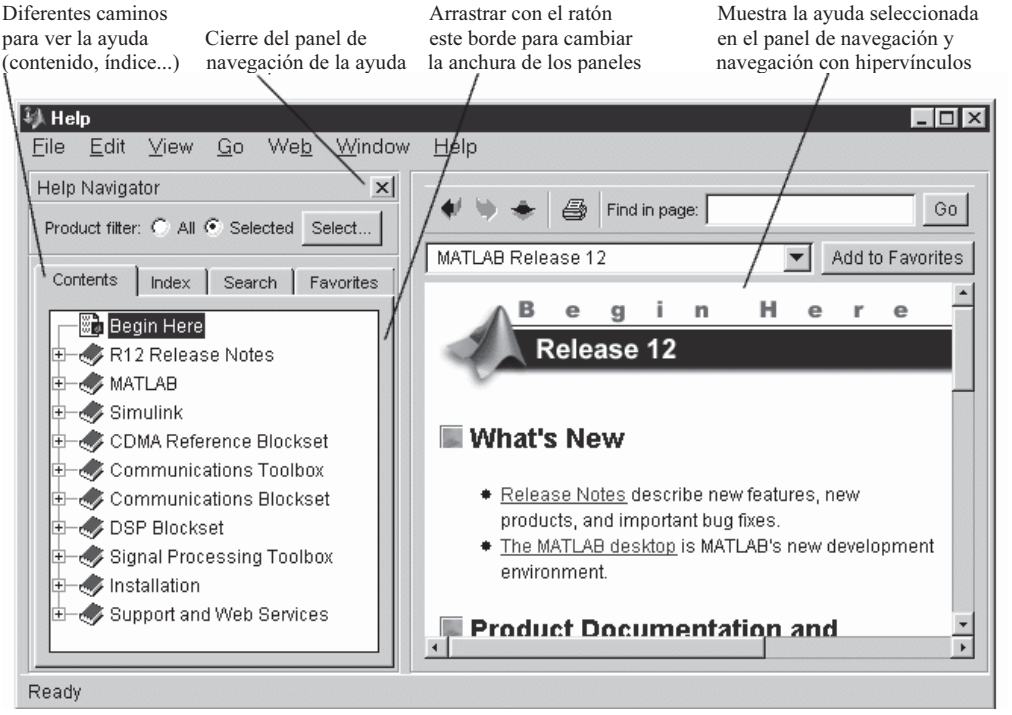


Figura 2-58

Ventana de espacio de trabajo

La ventana de espacio de trabajo (*Workspace*) se sitúa en la esquina superior izquierda del escritorio de MATLAB y se obtiene haciendo clic sobre la etiqueta *Work Space* situada debajo de ella (Figura 2-59). Su función es ver las variables almacenadas en memoria. Para cada variable se muestra su nombre, tipo, tamaño y clase, tal y como se indica en la Figura 2-60. Para mostrar esta ventana separada del escritorio de MATLAB (Figura 2-60) basta hacer clic en el botón situado en su esquina superior derecha. Para retornar la ventana a su sitio en el escritorio se utiliza la opción *Dock Command Window* del menú *View*.

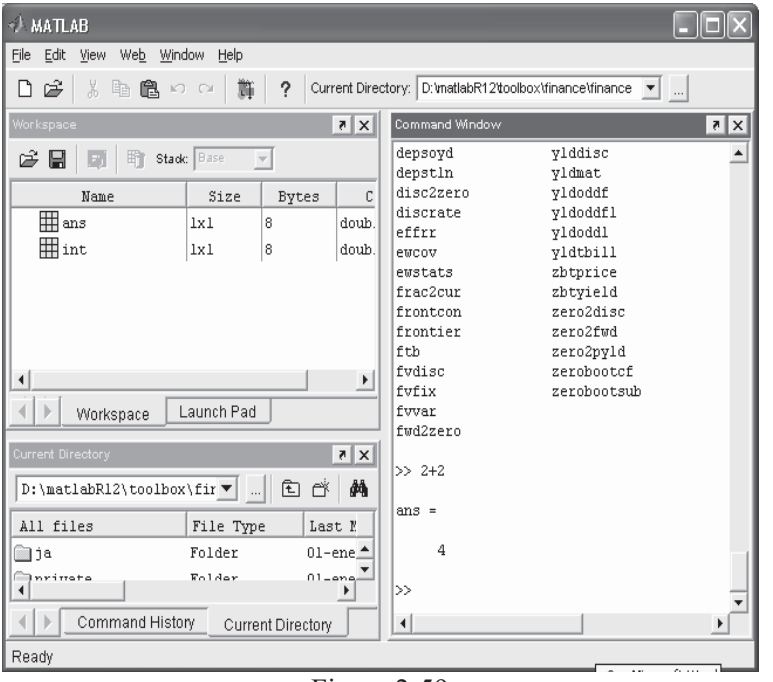


Figura 2-59

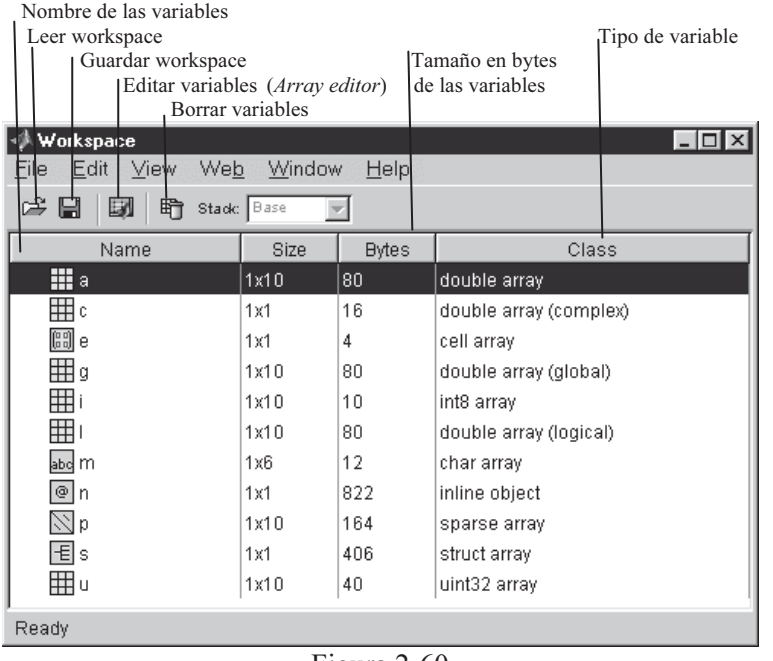


Figura 2-60

Un elemento importante de la ventana *Workspace* es el *Array editor*, que permite editar arrays numéricos y cadenas. En la Figura 2-61 se muestran los elementos del *Array editor*.

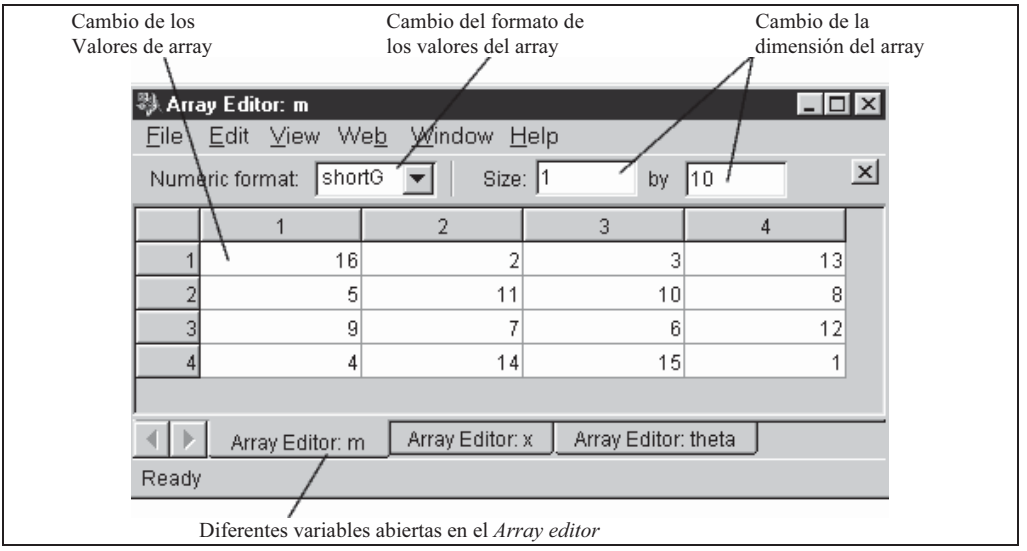


Figura 2-61

Es posible situar preferencias en la ventana de especio de trabajo mediante la opción *Preferences* del menú *File*. Se obtiene la ventana *Preferentes* de la Figura 2-62. En el campo *History* se fija el número de directorios recientes a salvar para el historial. En el campo *Font* se fijan las fuentes tal y como ya se indicó en las preferencias relativas a la ventana de comandos, y en el botón *Confirm Delettion of Variables* se elige la confirmación o no del borrado de variables.

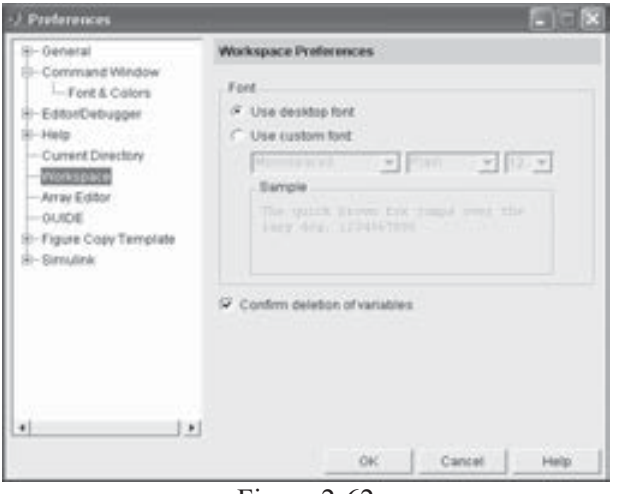


Figura 2-62

2.5 Editor y Debugger de M-ficheros

Para crear un nuevo M-fichero en el *Editor/Debugger* basta hacer clic en el botón de la barra de herramientas de MATLAB o seleccionar *File→ New→ M-file* en el escritorio de MATLAB (Figura 2-63). El *Editor/Debugger* se abre con un fichero en blanco en el cual crearemos el M-fichero, es decir, un fichero con código de programación MATLAB (Figura 2-65). El comando *Edit* en la ventana de comandos también abre el *Editor/Debugger*. Para abrir un M-fichero ya existente se utiliza *File→ Open* en el escritorio de MATLAB (Figura 2-63). También se puede utilizar el comando *Open* en la ventana de comandos.

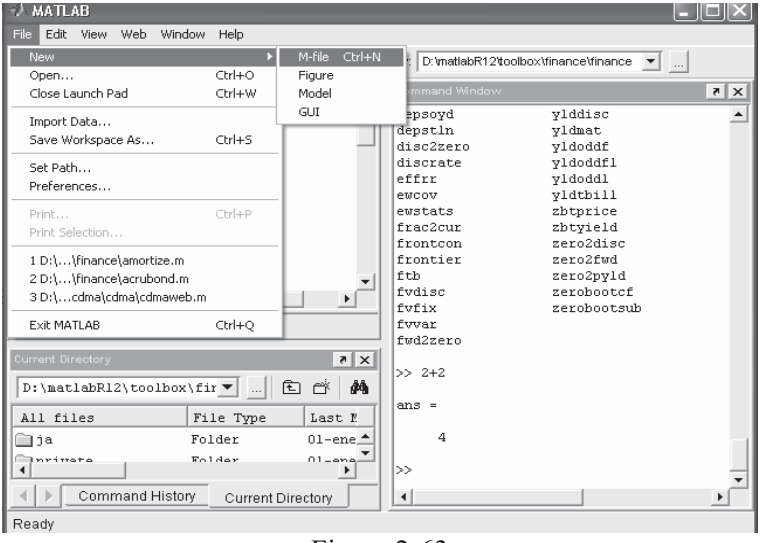


Figura 2-63

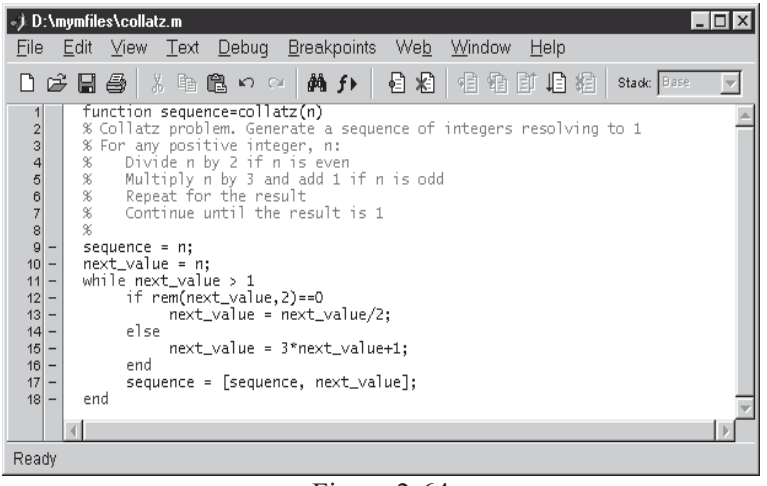


Figura 2-64

Igualmente se puede abrir el *Editor/Debugger* haciendo clic con el botón derecho del ratón en el interior de la ventana *Current Directory* y eligiendo *New → M-file* en el menú emergente resultante (Figura 2-65). La opción *Open* de este menú abre un M-fichero existente. Se pueden abrir varios M-ficheros simultáneamente, en cuyo caso aparecerán en ventanas diferentes (Figura 2-66).

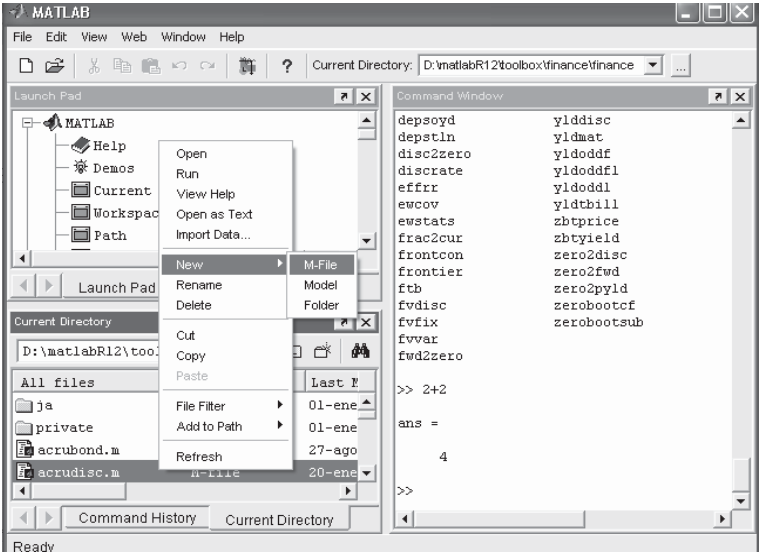


Figura 2-65

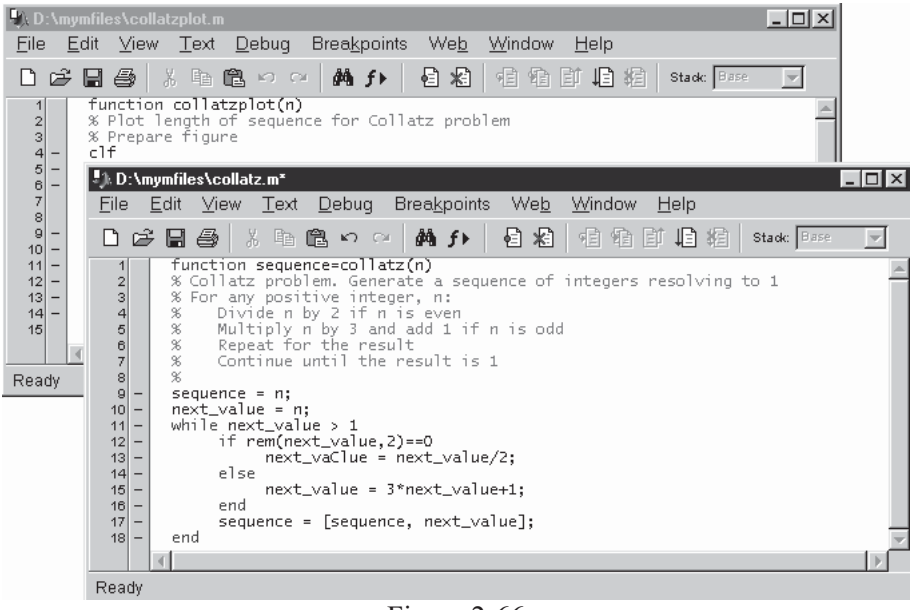


Figura 2-66

La Figura 2-67 muestra la función de la barra de iconos del *Editor/Debugger*.

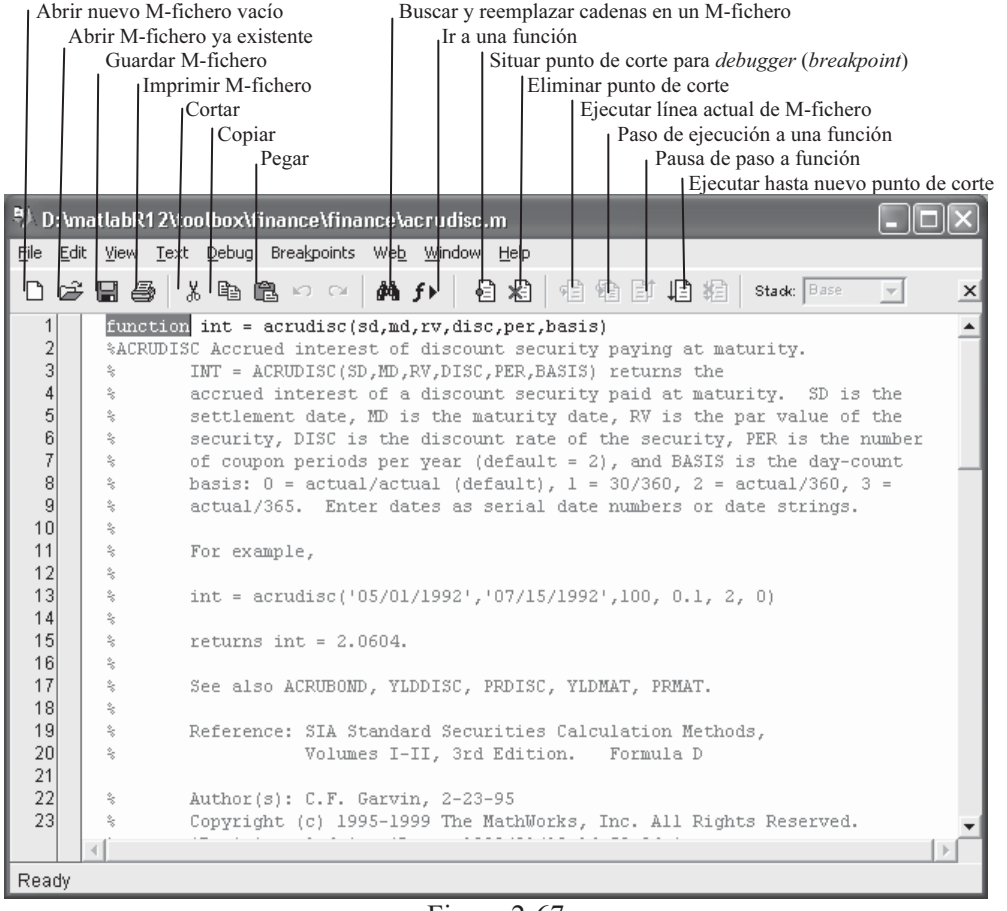


Figura 2-67

2.6 Ayuda en MATLAB

MATLAB dispone de un sistema de ayuda en línea bastante eficiente. La primera de las herramientas a tener en cuenta es el navegador de la ayuda (Figura 2-68), al que se accede mediante el icono ? o tecleando *helpbrowser* en la ventana de comandos (en el menú *View* debe estar señalizada la opción *Help Browser*). En el panel de la izquierda del navegador de la ayuda se selecciona el tema, y en el panel de la derecha se presenta la ayuda relativa al tema seleccionado, siendo posible el uso de hipervínculos para navegar por su contenido. La parte izquierda del panel de navegación de la ayuda presenta en su zona superior una barra con las opciones *Content* (ayuda por contenido), *Index* (ayuda por índice alfabético), *Search* (buscar ayuda por tema) y *Favorites* (temas de ayuda favoritos).

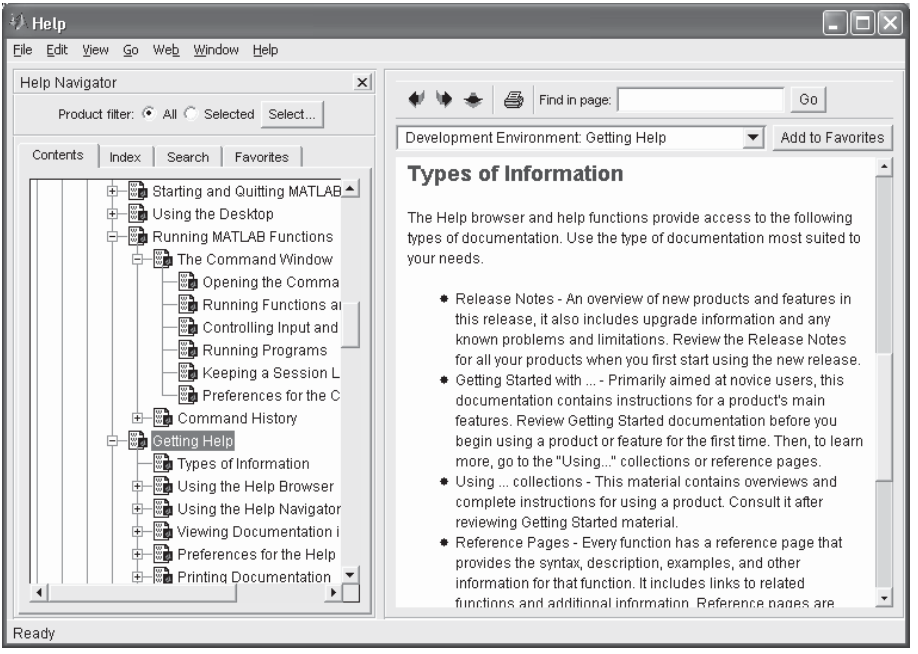


Figura 2-68

Otro camino muy importante para obtener ayuda en MATLAB es el uso de sus funciones de ayuda. En el cuadro siguiente se presentan estas funciones.

Función	***Descripción***
`doc función`	*Muestra la página de referencia en el navegador de la ayuda para la función especificada, mostrando sintaxis, descripción, ejemplos y enlaces con otras funciones relacionadas.*
`docopt`	*Se usa esta función para mostrar la localización de los ficheros de ayuda en plataformas UNIX que no soportan interfaces Java.*
`help función`	*Muestra sobre la ventana de comandos descripción y sintaxis de la función especificada.*
`helpbrowser`	*Abre el navegador de la ayuda.*
`helpdesk`	*Abre el navegador de la ayuda. Se mantiene por compatibilidad.*
`helpwin ó helpwin tema`	*Muestra en el navegador de la ayuda una lista de todas las funciones de MATLAB o las relativas al tema especificado.*
`lookfor texto`	*Muestra en el navegador de la ayuda todas las funciones que contienen el texto especificado como parte de la función.*
`web url`	*Abre en el navegador Web la URL especificada por defecto como relativa a la ayuda Web de MATLAB.*

A continuación se presentan algunos ejemplos de funciones de ayuda:

```
>> help
HELP topics:
MATLAB\general       -  General purpose commands.
MATLAB\ops           -  Operators and special characters.
MATLAB\lang          -  Programming language constructs.
MATLAB\elmat         -  Elementary matrices and matrix manipulation.
MATLAB\elfun         -  Elementary math functions.
MATLAB\specfun       -  Specialized math functions.
MATLAB\matfun        -  Matrix functions - numerical linear algebra.
MATLAB\datafun       -  Data analysis and Fourier transforms.
MATLAB\audio         -  Audio support.
MATLAB\polyfun       -  Interpolation and polynomials.
MATLAB\funfun        -  Function functions and ODE solvers.
MATLAB\sparfun       -  Sparse matrices.
MATLAB\graph2d       -  Two dimensional graphs.
MATLAB\graph3d       -  Three dimensional graphs.
MATLAB\specgraph     -  Specialized graphs.
MATLAB\graphics      -  Handle Graphics.
MATLAB\uitools       -  Graphical user interface tools.
MATLAB\strfun        -  Character strings.
MATLAB\iofun         -  File input/output.
MATLAB\timefun       -  Time and dates.
MATLAB\datatypes     -  Data types and structures.
.....................................................................................
```

Vemos que el comando *help* muestra una lista de los directorios del programa y de su contenido.

```
>> help sin
 SIN    Sine.
    SIN(X) is the sine of the elements of X.
 Overloaded methods
    help sym/sin.m
```

En este caso observamos que el comando *help* muestra ayuda sobre la función especificada.

```
>> lookfor inverse

INVHILB Inverse Hilbert matrix.
ACOS    Inverse cosine.
ACOSH   Inverse hyperbolic cosine.
ACOT    Inverse cotangent.
ACOTH   Inverse hyperbolic cotangent.
.....................................................
```

En este ejemplo hemos buscado las funciones que tratan sobre *inverse*.

3

Variables, números, operadores y funciones

3.1 Variables

MATLAB no requiere ningún tipo de comando para declarar variables. Sencillamente crea la variable mediante asignación directa de su valor. Por ejemplo:

```
>> v=3

v =

     3
```

La variable *v* valdrá 3 mientras no se cambie su valor mediante una nueva asignación. Una vez declarada la variable podemos utilizarla en cálculos.

```
>> v^3

ans =

    27

>> v+5

ans =

     8
```

El valor asignado a una variable es permanente, hasta que no se cambie de forma expresa o hasta que no se salga de la presente sesión de MATLAB.

Si ahora escribimos:

```
>> v=3+7

v =

    10
```

la variable *v* pasa a valer 10 a partir de este momento, tal y como se observa en el cálculo siguiente:

```
>> v^4

ans =

       10000
```

Los nombres de las variables comienzan por una letra seguida de cualquier número de letras, dígitos o subrayados, teniendo presente que MATLAB sólo utiliza los primeros 31 caracteres del nombre de la variable. También es muy importante señalar que MATLAB es sensible a mayúsculas y minúsculas; por lo tanto, una variable con mayúsculas es distinta a la misma variable con minúsculas.

Variables vectoriales

Para representar a un vector de *n* elementos se puede definir en MATLAB una variable de las siguientes formas:

V = [v1, v2, v3, ...,vn]

V = [v1 v2 v3 ..... vn]

Cuando se aplican la mayoría de los comandos y funciones de MATLAB sobre una variable vectorial el resultado que se obtiene es la aplicación del comando o función sobre cada elemento del vector:

```
>> vector1=[1,4,9,2.25,1/4]

vector1 =

    1.0000    4.0000    9.0000    2.2500    0.2500
```

```
>> sqrt(vector1)

ans =

    1.0000    2.0000    3.0000    1.5000    0.5000
```

Existen diferentes formas de definir una variable vectorial sin necesidad de explicitar entre corchetes todos sus elementos separados por comas o espacios en blanco. Se presentan en la tabla siguiente

variable=[a:b]	*Define el vector cuyos primero y último elementos son a y b, respectivamente, y los elementos intermedios se diferencian en una unidad*
variable=[a:s:b]	*Define el vector cuyos primero y último elementos son a y b, y los elementos intermedios se diferencian en la cantidad s especificada por el incremento*
variable=linespace[a,b,n]	*Define el vector cuyos primero y último elementos son a y b, y que tiene en total n elementos uniformemente espaciados entre sí*
variable=logespace[a,b,n]	*Define el vector cuyos primero y último elementos son los especificados y que tiene en total n elementos en escala logarítmica uniformemente espaciados entre sí*

A continuación se presentan algunos ejemplos:

```
>> vector2=[5:5:25]

vector2 =

     5    10    15    20    25
```

Hemos obtenido los números entre 5 y 25 separados por 5 unidades.

```
>> vector3=[10:30]

vector3 =

  Columns 1 through 13

    10    11    12    13    14    15    16    17    18    19
    20    21    22

  Columns 14 through 21

    23    24    25    26    27    28    29    30
```

Hemos obtenido los números entre 10 y 30 separados una unidad.

```
>> vector4=linspace(10,30,6)

vector4 =

    10    14    18    22    26    30
```

Hemos obtenido 6 números entre 0 y 20 igualmente espaciados.

```
>> vector5=logspace(10,30,6)

vector5 =

  1.0e+030 *

    0.0000    0.0000    0.0000    0.0000    0.0001    1.0000
```

Hemos obtenido 6 números entre antilogaritmo decimal de 0 y antilogaritmo decimal de 2, con una separación logarítmica uniforme.

En MATLAB también existe la posibilidad de considerar vectores fila y vectores columna. Un vector columna se obtiene separando sus elementos por punto y coma, o también transponiendo un vector fila mediante una comilla simple situada al final de su definición.

```
>> a=[10;20;30;40]

a =

    10
    20
    30
    40

>> a=(10:14);b=a'

b =

    10
    11
    12
    13
    14

>> c=(a')'

c =

    10    11    12    13    14
```

Asimismo podemos seleccionar un elemento de un vector o un subconjunto de elementos. La tabla siguiente presenta las reglas:

x(n)	*Devuelve el enésimo elemento del vector x*
x(a:b)	*Devuelve los elementos del vector x situados entre el a-ésimo y el bésimo, ambos inclusive*
x(a:p:b)	*Devuelve los elementos del vector x situados entre el a-ésimo y el bésimo, ambos inclusive, pero separados de p en p unidades (a>b)*
x(b:-p:a)	*Devuelve los elementos del vector x situados entre el b-ésimo y el a-ésimo, ambos inclusive, pero separados de p en p unidades y empezando por el bésimo (b>a)*

Veamos algunos ejemplos:

```
>> x=(1:10)
x =
     1     2     3     4     5     6     7     8     9    10
>> x(6)
ans =
     6
```

Hemos obtenido el sexto elemento del vector *x*.

```
>> x(4:7)
ans =
     4     5     6     7
```

Hemos obtenido los elementos del vector *x* situados entre el cuarto y el séptimo, ambos inclusive.

```
>> x(2:3:9)
ans =
     2     5     8
```

Hemos obtenido los elementos del vector *x* situados entre el segundo y el noveno, ambos inclusive, pero separados de tres en tres unidades.

```
>> x(9:-3:2)
ans =
     9     6     3
```

Hemos obtenido los elementos del vector *x* situados entre el noveno y el segundo, ambos inclusive, pero separados de tres en tres unidades y empezando por el noveno.

Variables matriciales

En MATLAB se definen las matrices introduciendo entre corchetes todos sus vectores fila separados por punto y coma. Los vectores se pueden introducir separando sus componentes por espacios en blanco o por comas, tal y como ya sabemos. Por ejemplo, una variable matricial de dimensión 3x3 se puede introducir de las dos siguientes formas:

M = [a_{11} a_{12} a_{13};a_{21} a_{22} a_{23};a_{31} a_{32} a_{33}]

M = [a_{11} ,a_{12} ,a_{13};a_{21} ,a_{22} ,a_{23};a_{31} ,a_{32} ,a_{33}]

De forma semejante se definiría una variable matricial de dimensión *mxn*. Una vez que una variable matricial ha sido definida, MATLAB habilita muchos caminos para insertar, extraer, renumerar y manipular en general sus elementos. La tabla siguiente presenta diferentes posibilidades de definición de variables matriciales.

A(m,n)	*Define el elemento (m,n) de la matriz A (fila m y columna n)*
A(a:b,c:d)	*Define la submatriz de A formada por las filas que hay entre la a-ésima y la b-ésima y por las columnas que hay entre la c-ésima y la d-ésima*
A(a:p:b,c:q:d)	*Define la submatriz de A formada por las filas que hay entre la a-ésima y la b-ésima tomándolas de p en p, y por las columnas que hay entre la c-ésima y la d-ésima tomándolas de q en q*
A([a b],[c d])	*Define la submatriz de A formada por la intersección de las filas a-ésima y b-ésima y las columnas c-ésima y d-ésima*
A([a b c ...], [e f g ...])	*Define la submatriz de A formada por la intersección de las filas a, b, c, ... y las columnas e, f, g, ...*
A(:,c:d)	*Define la submatriz de A formada por todas las filas de A y por las columnas que hay entre la c-ésima y la d-ésima*
A(:,[c d e ...])	*Define la submatriz de A formada por todas las filas de A y por las columnas c, d, e, ...*
A(a:b,:)	*Define la submatriz de A formada por todas las columnas de A y por las filas que hay entre la a-ésima y la b-ésima*
A([a b c ...],:)	*Define la submatriz de A formada por todas las columnas de A y por las filas a, b, c, ...*
A(a,:)	*Define la fila a-ésima de la matriz A*
A(:,b)	*Define la columna b-ésima de la matriz A*
A(:)	*Define un vector columna cuyos elementos son las columnas de A situadas por orden una debajo de otra*
A(:,:)	*Equivale a toda la matriz A*
[A,B,C,...]	*Define la matriz formada por las submatrices A, B, C, ...*
S_A = [ ]	*Borra la submatriz de la matriz A, S_A y devuelve el resto*

diag(v)	*Crea una matriz diagonal con el vector v en la diagonal*
diag(A)	*Extrae la diagonal de la matriz A como vector columna*
eye(n)	*Crea la matriz identidad de orden n*
eye(m,n)	*Crea la matriz de orden mxn con unos en la diagonal principal y ceros en el resto*
zeros(m,n)	*Crea la matriz nula de orden mxn*
ones(m,n)	*Crea la matriz de orden mxn con todos sus elementos 1*
rand(m,n)	*Crea una matriz aleatoria uniforme de orden mxn*
randn(m,n)	*Crea una matriz aleatoria normal de orden mxn*
flipud(A)	*Devuelve la matriz cuyas filas están colocadas en orden inverso (de arriba abajo) a las filas de A*
fliplr(A)	*Devuelve la matriz cuyas columnas están colocadas en orden inverso (de izquierda a derecha) a las de A*
rot90(A)	*Rota 90 grados la matriz A*
reshape(A,m,n)	*Devuelve la matriz de orden mxn extraída de la matriz A tomando elementos consecutivos de A por columnas*
size(A)	*Devuelve el orden (tamaño) de la matriz A*
find(cond$_A$)	*Devuelve los elementos de A que cumplen la condición*
length(v)	*Devuelve la longitud del vector v*
tril(A)	*Devuelve la parte triangular inferior de la matriz A*
triu(A)	*Devuelve la parte triangular superior de la matriz A*
A'	*Devuelve la matriz transpuesta de A*
inv(A)	*Devuelve la matriz inversa de A*

Veamos algunos ejemplos:

Consideramos en primer lugar la matriz *2x3* cuyas filas son los 6 primeros impares consecutivos:

```
>> A=[1 3 5;7 9 11]

A =

     1     3     5
     7     9    11
```

Ahora vamos a anular el elemento *(2,3)*, o sea, su último elemento:

```
>> A(2,3)=0

A =

     1     3     5
     7     9     0
```

A continuación consideramos la matriz *B* transpuesta de *A*:

```
>> B=A'

B =

     1     7
     3     9
     5     0
```

Ahora construimos una matriz *C*, formada por la matriz *B* y la matriz identidad de orden 3 adosada a su derecha:

```
>> C=[B eye(3)]

C =

     1     7     1     0     0
     3     9     0     1     0
     5     0     0     0     1
```

Vamos a construir una matriz *D* extrayendo las columnas impares de la matriz *C*, una matriz *E* formada por la intersección de las dos primeras filas de *C* y sus columnas tercera y quinta, y una matriz *F* formada por la intersección de las dos primeras filas y las tres últimas columnas de la matriz *C*:

```
 >> D=C(:,1:2:5)

D =

     1     1     0
     3     0     0
     5     0     1

>> E=C([1 2],[3 5])

E =

     1     0
     0     0


>> F=C([1 2],3:5)

F =

     1     0     0
     0     1     0
```

Ahora construimos la matriz diagonal *G* tal que los elementos de su diagonal principal son los mismos que los de la diagonal principal de *D*:

```
>> G=diag(diag(D))

G =

     1     0     0
     0     0     0
     0     0     1
```

A continuación construimos la matriz *H*, formada por la intersección de la primera y tercera filas de *C* y sus segunda, tercera y quinta columnas:

```
>> H=C([1 3],[2 3 5])

H =

     7     1     0
     0     0     1
```

Ahora construimos una matriz *I* formada por la matriz identidad de orden *5x4* y las matrices nula y unidad del mismo orden adosadas a su derecha. A continuación extraemos la primera fila de *I* y, por último, formamos la matriz *J* con las filas impares y las columnas pares de *Y* y calculamos su orden (tamaño).

```
>> I = [eye(5,4) zeros(5,4) ones(5,4)]

ans =

     1     0     0     0     0     0     0     0     1     1     1     1
     0     1     0     0     0     0     0     0     1     1     1     1
     0     0     1     0     0     0     0     0     1     1     1     1
     0     0     0     1     0     0     0     0     1     1     1     1
     0     0     0     0     0     0     0     0     1     1     1     1

>> I(1,:)

ans =

     1     0     0     0     0     0     0     0     1     1     1     1

>> J=I(1:2:5,2:2:12)

J =

     0     0     0     0     1     1
     0     0     0     0     1     1
     0     0     0     0     1     1

>> size(J)

ans =

     3     6
```

A continuación construimos una matriz aleatoria *K* de orden *3x4* e invertimos primero el orden de sus filas, después el orden de sus columnas y a continuación el orden de sus filas y columnas a la vez. Por último, hallamos la matriz *L* de orden *4x3* cuyas columnas resultan de tomar los elementos de las columnas de *K* consecutivamente.

```
>> K=rand(3,4)

K =

    0.5269    0.4160    0.7622    0.7361
    0.0920    0.7012    0.2625    0.3282
    0.6539    0.9103    0.0475    0.6326

>> K(3:-1:1,:)

ans =

    0.6539    0.9103    0.0475    0.6326
    0.0920    0.7012    0.2625    0.3282
    0.5269    0.4160    0.7622    0.7361

>> K(:,4:-1:1)

ans =

    0.7361    0.7622    0.4160    0.5269
    0.3282    0.2625    0.7012    0.0920
    0.6326    0.0475    0.9103    0.6539

>> K(3:-1:1,4:-1:1)

ans =

    0.6326    0.0475    0.9103    0.6539
    0.3282    0.2625    0.7012    0.0920
    0.7361    0.7622    0.4160    0.5269

>> L=reshape(K,4,3)

L =

    0.5269    0.7012    0.0475
    0.0920    0.9103    0.7361
    0.6539    0.7622    0.3282
    0.4160    0.2625    0.6326
```

Variables carácter

Una variable carácter (cadena) es sencillamente una cadena de caracteres incluidos entre comillas simples que MATLAB trata de forma vectorial. La sintaxis general para la declaración de variables carácter en MATLAB es la siguiente:

c = 'cadena de caracteres'

Entre los comandos de MATLAB que manejan variables carácter tenemos los siguientes:

abs('cadena_caracteres')	*Devuelve el vector de caracteres ASCII equivalentes a cada carácter de la cadena*
setstr(vector_numérico)	*Devuelve la cadena de caracteres ASCII equivalentes a los elementos del vector*
str2mat(t1,t2,t3, ...)	*Forma la matriz cuyas filas son las cadenas de caracteres t1, t2, t3, ... , respectivamente*
str2num('cadena')	*Convierte la cadena de caracteres en su valor numérico exacto utilizado por MATLAB*
num2str(número)	*Convierte el número exacto en su cadena de caracteres equivalente con la precisión fijada*
int2str(entero)	*Convierte el número entero en cadena*
sprintf('formato', A)	*Convierte la matriz numérica exacta A a una cadena con el formato especificado*
sscanf('cadena', 'formato')	*Convierte la cadena a valor numérico con el formato especificado*
dec2hex(entero)	*Convierte el entero decimal a su cadena equivalente en hexadecimal*
hex2dec('cadena_hex')	*Convierte la cadena hexadecimal en el número entero equivalente*
hex2num('cadena_hex')	*Convierte la cadena hexadecimal en el número IEEE en punto flotante equivalente*
lower('cadena')	*Convierte la cadena a minúsculas*
upper('cadena')	*Convierte la cadena a mayúsculas*
strcmp(c1,c2)	*Compara las cadenas s1 y s2 y devuelve 1 si son iguales y 0 en caso contrario*
strcmp(c1,c2,n)	*Compara las cadenas s1 y s2 y devuelve 1 si son iguales sus n primeros caracteres y 0 en caso contrario*
strrep(c, 'exp1', 'exp2')	*Reemplaza en la cadena c, exp2 por exp1*
findstr(c, 'exp')	*Da los lugares que ocupa exp en la cadena c*
isstr(expresión)	*Da 1 si la expresión es cadena y 0 si no lo es*
ischar(expresión)	*Da 1 si la expresión es cadena y 0 si no lo es*
strjust(cadena)	*Ajusta la cadena a la derecha*

blanks(n)	*Genera una cadena de n blancos*
deblank(cadena)	*Sustituye los blancos de la cadena por nulos*
eval(expresión)	*Ejecuta la expresión aunque sea una cadena*
disp('cadena')	*Muestra la cadena (o matriz) tal y como se ha escrito y continúa el proceso de MATLAB*
input('cadena')	*Muestra la cadena en pantalla y MATLAB espera la presión de una tecla para continuar*

Veamos algunos ejemplos:

```
>> hex2dec('3ffe56e')

ans =

    67102062
```

MATLAB ha convertido en decimal una cadena en hexadecimal.

```
>> dec2hex(1345679001)

ans =

50356E99
```

El programa ha convertido un número decimal a cadena hexadecimal.

```
>> sprintf(' %f',[1+sqrt(5)/2,pi])

ans =

 2.118034 3.141593
```

Se han convertido a cadena (con la precisión por defecto) las componentes numéricas exactas de un vector.

```
>> sscanf('121.00012', '%f')

ans =

  121.0001
```

Se ha pasado a formato numérico exacto (con la precisión por defecto) una cadena de caracteres numéricos. Al estudiar posteriormente el comando *fscanf*, en este mismo capítulo, se verán los posibles valores de los formatos.

```
>> num2str(pi)

ans =

3.142
```

Se ha convertido el número exacto π a cadena de caracteres.

```
>> str2num('15/14')

ans =

    1.0714
```

Se ha convertido una cadena a su valor numérico exacto con la precisión prefijada (en este caso la precisión por defecto).

```
>> setstr(32:126)

ans =

!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\
]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

Hemos obtenido los caracteres ASCII asociados a los números enteros entre 32 y 126.

```
>> abs('{]}><#¡¿?°ª')

ans =

123    93   125    62    60    35   161   191    63   186   170
```

Hemos obtenido los números enteros correspondientes a los caracteres ASCII especificados en el argumento de *abs*.

```
>> lower('ABCDefgHIJ')

ans =

abcdefghij
```

Se ha traducido a minúsculas un texto.

```
>> upper('abcd eFGHi jKlMn')

ans =

ABCD EFGHI JKLMN
```

Se ha traducido a mayúsculas un texto.

```
>> str2mat('El Mundo','El País','Diario 16','ABC')

ans =
```

```
El Mundo
El País
Diario 16
ABC
```

Se ha formado una matriz de texto cuyas filas son las cadenas introducidas como argumentos de *str2mat*.

```
>> disp('este texto aparecerá en la pantalla')

este texto aparecerá en la pantalla
```

Se ha obtenido por pantalla el texto argumento del comando *disp*.

```
>> c='esto es un buen ejemplo';
>> strrep(c, 'buen', 'mal')

ans =

esto es un mal ejemplo
```

Se ha sustituido *bien* por *mal* en la cadena *c* y a continuación se verán los lugares que ocupa la expresión *es* dentro de la cadena *c*.

```
>> findstr(c, 'es')

ans =

     1     6
```

3.2 Números

En MATLAB se puede trabajar con diferentes tipos de números y expresiones numéricas, que abarcan todo el campo de los números enteros, racionales, reales y los números complejos, y se utilizan en argumentos de funciones.

Las operaciones aritméticas en MATLAB se definen de acuerdo con las convenciones matemáticas estándar. MATLAB es un programa interactivo que permite realizar de manera sencilla gran variedad de operaciones matemáticas. MATLAB asume las operaciones aritméticas habituales de suma, diferencia, producto, división y potencia, con la jerarquía habitual entre ellas:

x + y	*Suma*
x - y	*Diferencia*
x * y o x y	*Producto*
x/y	*División*
x ^y	*Potencia*

Para sumar dos números teclee simplemente el primer número, un signo más (+) y el segundo número. Puede incluir tranquilamente espacios antes y después del signo más para que el *input* sea más fácil de leer.

```
>> 2+3

ans =

     5
```

Podemos realizar el cálculo de una potencia directamente.

```
>> 100^50

ans =

  1.0000e+100
```

A diferencia de una calculadora, cuando trabaja con enteros, MATLAB muestra el resultado exacto incluso cuando tiene más dígitos de los que cabrían a lo ancho de la pantalla. MATLAB devuelve el valor exacto de *100^50* si se utiliza la función vpa.

```
>> vpa '99^50'

ans =

.60500606713753665044791996801256e100
```

Al combinar varias operaciones en una misma instrucción han de tenerse en cuenta los criterios de prioridad habituales entre ellas, que determinan el orden de evaluación de la expresión. Véase el siguiente ejemplo:

```
>> 2*3^2+(5-2)*3

ans =

    27
```

Teniendo en cuenta la prioridad de los operadores, el primero en ser evaluado es el de potenciación. El orden de evaluación normal puede alterarse agrupando expresiones entre paréntesis.

Además de estos operadores aritméticos, MATLAB está dotado de un conjunto de funciones básicas y el usuario puede definir también sus propias funciones. Tanto los operadores como las funciones que lleva incorporadas MATLAB pueden ser aplicadas sobre constantes simbólicas o números.

Una de las primeras aplicaciones de MATLAB es su uso para la realización de operaciones aritméticas como si se tratara de una calculadora convencional, pero con una importante diferencia sobre ella: la precisión en el cálculo. Las operaciones son realizadas bien en forma exacta o bien especificando el usuario el grado de precisión que desea. Esta precisión ilimitada en el cálculo es la característica que diferencia a MATLAB de otros programas de cálculo numérico, donde la longitud de palabra con que trabaja el ordenador determina la precisión, con lo que se convierte en algo inherente al hardware y no modificable.

La exactitud de la salida de los resultados de las operaciones con MATLAB puede relajarse utilizando técnicas especiales de aproximación al resultado exacto con un determinado grado de precisión. MATLAB representa los resultados con exactitud, pero aunque internamente siempre trabaja con cálculos exactos para no arrastrar errores de redondeo, pueden habilitarse diferentes formatos de representación aproximada, que en ocasiones facilitan la interpretación de los resultados. A continuación se citan los comandos que permiten aproximaciones numéricas:

format long	*Ofrece los resultados con 16 cifras decimales*
format short	*Ofrece los resultados con 4 cifras decimales. Se trata del formato por defecto de MATLAB*
format long e	*Ofrece los resultados con 16 decimales más la potencia de 10 necesaria*
format short e	*Ofrece los resultados con 4 decimales más la potencia de 10 necesaria*
format long g	*Ofrece los resultados en formato largo óptimo*
format short g	*Ofrece los resultados en formato corto óptimo*
format bank	*Ofrece los resultados con 2 cifras decimales*
format rat	*Ofrece los resultados en forma de número racional aproximado*
format +	*Ofrece el signo de los resultados (+, -) e ignora la parte imaginaria de los números complejos*
format hex	*Ofrece los resultados en el sistema hexadecimal*
vpa 'operaciones' n	*Ofrece el resultado de las operaciones con n dígitos decimales exactos*
numeric('expr')	*Ofrece el valor de la expresión de forma numérica aproximada según el formato actual activo*
digits(n)	*Ofrece los resultados con n dígitos exactos*

Mediante *format* obtenemos una aproximación numérica de 174/13 de la manera especificada en el formato:

```
>> 174/13

ans =
```

```
   13.3846
>> format long; 174/13

ans =

  13.38461538461539

>> format long e; 174/13

ans =

    1.338461538461539e+001

>> format short e; 174/13

ans =

  1.3385e+001

>> format long g; 174/13

ans =

          13.3846153846154

>> format short g; 174/13

ans =

       13.385

>> format bank; 174/13

ans =

         13.38

>> format hex; 174/13

ans =

   402ac4ec4ec4ec4f
```

Ahora vamos a ver ejemplos del cálculo del valor de *sqrt(17)* con las cifras decimales de precisión que nos apetezcan:

```
>> vpa '174/13' 10

ans =

13.38461538
```

```
 >> vpa '174/13' 15

ans =

13.3846153846154

>> digits(20); vpa '174/13'

ans =

13.384615384615384615
```

Números enteros

En MATLAB todas las operaciones usuales con números enteros se realizan de forma exacta, independientemente del tamaño que tenga la salida del resultado. Si queremos que el resultado de una operación aparezca en pantalla con un determinado número de cifras exactas, utilizamos el comando de cálculo simbólico ***vpa*** (*variable precission arithmetic*), cuya sintaxis ya conocemos.

Por ejemplo, al calcular 6 elevado a la potencia 400 con 450 cifras exactas se obtiene como sigue:

```
>> vpa '6^400' 450

ans =

182179771682187282513946871240893712673389715281747606674596975
493339599720905327003028267800766283867331479599455916367452421
574456059646801054954062150177042349998869907885947439947961712
484067309738073652485056311556920850878594283008099992731076250
733948404739350551934565743979678824151197232629947748581376.
```

El resultado de la operación es exacto, siempre que aparezca un punto al final del resultado. En este caso no son necesarias 450 cifras para expresar el resultado de la operación propuesta, con lo cual la solución es exacta. Si se requiere un número más pequeño de cifras exactas que las que realmente tiene el resultado, MATLAB redondea por la cifra pedida y completa el resultado con potencias de 10. Por ejemplo, vamos a realizar el cálculo anterior solamente con 50 cifras exactas.

```
>> vpa '6^400' 50

ans =

.18217977168218728251394687124089371267338971528175e312
```

Funciones con números enteros y divisibilidad

Existen varias funciones en MATLAB con argumento entero, la mayoría de las cuales son relativas a divisibilidad. Entre las funciones con argumento entero más típicas destacan las siguientes:

rem(n,m)	*Resto de la división de n entre m (función válida para n y m reales)*
sign(n)	*Signo de n (1 si n>0 , -1 si n<0, n real)*
max(n1,n2)	*Máximo de los números n1 y n2*
min(n1,n2)	*Mínimo de los números n1 y n2*
gcd(n1,n2)	*Máximo común divisor de n1 y n2*
lcm(n1,n2)	*Mínimo común múltiplo de n1 y n2*
factorial(n)	*Factorial de n (n(n-1)(n-2)...1)*
factor(n)	*Descompone n en factores primos*

A continuación se presentan algunos ejemplos.

Resto de la división de 17 entre 3:

```
>> rem(17,3)

ans =

     2
```

Resto de la división de 4.1 entre 1.2:

```
>> rem(4.1,1.2)

ans =
    0.5000
```

Resto de la división entre -4.1 y 1.2:

```
>> rem(-4.1,1.2)
ans =
   -0.5000
```

Máximo común divisor de 1.000, 500 y 625:

```
>> gcd(1000,gcd(500,625))
ans =
        125.00
```

Mínimo común múltiplo entre 1.000, 500 y 625:

```
>> lcm(1000,lcm(500,625))

ans =

       5000.00
```

Sistemas de numeración

MATLAB permite trabajar con sistemas de numeración de base cualquiera, siempre y cuando se disponga del *Toolbox* extendido de matemática simbólica. Además, permite expresar todo tipo de números en las diferentes bases. Para ello implementa las siguientes funciones:

dec2base(decimal,n_base)	*Convierte el número decimal (base 10) especificado a la nueva base n_base dada*
base2dec(numero,B)	*Convierte el número dado en base B a decimal*
dec2bin(decimal)	*Convierte el número decimal especificado a base 2 (binario)*
dec2hex(decimal)	*Convierte el número decimal especificado a base 16 (hexadecimal)*
bin2dec(binario)	*Convierte el número binario especificado a base decimal*
hex2dec(hexadecimal)	*Convierte el número base 16 especificado a base decimal*

A continuación se presentan algunos ejemplos.

Se representa en base 10 el número 100101 en base 2.

```
>> base2dec('100101',2)

ans =

         37.00
```

Se representa en base 10 el número hexadecimal FFFFAA00.

```
>> base2dec('FFFFAA0',16)

ans =

  268434080.00
```

Se halla en base 10 el resultado de la operación FFFAA2 + FF –1

```
>> base2dec('FFFAA2',16)+base2dec('FF',16)-1

ans =

   16776096.00
```

Números reales

Como bien sabemos, los números reales son la unión disjunta de los números racionales y los números irracionales. Un número racional es de la forma *p/q*, donde *p* es un entero y *q* otro entero. Luego los racionales son números que se pueden representar como el cociente de un entero dividido por otro entero. La forma en que MATLAB trata los racionales es distinta a la de la mayoría de calculadoras. Si se pide a una calculadora que calcule la suma *1/2+1/3+1/4*, la mayoría devolverá algo como *1.0833*, que no es más que una aproximación al resultado.

Los números racionales son cocientes de enteros, y MATLAB también puede trabajar con ellos en modo exacto, de manera que el resultado de expresiones en las que intervienen números racionales es siempre otro número racional o entero. Para ello es necesario activar el formato racional con el comando *format rat*. Pero MATLAB también devuelve aproximaciones decimales de los resultados si el usuario así lo desea, activando cualquier otro tipo de formato (por ejemplo, *format short* o *format long*). La operación propuesta anteriormente a la calculadora la resuelve MATLAB en modo exacto de la siguiente forma:

```
>> format rat
>> 1/2+1/3+1/4

ans =

    13/12
```

A diferencia de las calculadoras, al hacer operaciones con números racionales el resultado siempre se puede conseguir exacto. Por ello, mientras MATLAB esté tratando con racionales como cocientes de enteros, los mantiene en esta forma. De esta manera, no se arrastran errores de redondeo en los cálculos con fracciones, que pueden llegar a ser muy graves, como demuestra la Teoría de errores. Nótese que, una vez habilitado el formato racional, cuando se pide a MATLAB que sume dos racionales, devuelve un racional como cociente de enteros y así lo representa simbólicamente. Una vez habilitado el formato de trabajo racional, las operaciones con racionales serán exactas hasta que no se habilite otro formato distinto, en cuyo caso también podemos obtener aproximaciones decimales a los números racionales.

Un número con coma flotante, o sea, un número con punto decimal, se interpreta como exacto siempre que esté habilitado el formato racional. Si hay un número con una coma flotante en la expresión, MATLAB trata toda la expresión como racional exacta y representa el resultado en números racionales. A su vez, si existe un número irracional en una expresión racional, MATLAB lo hace corresponder a una fracción para trabajar en formato racional.

```
>> format rat
>> 10/23 + 2.45/44

ans =

  1183/2412
```

El otro subconjunto fundamental dentro de los números reales es el de los números irracionales, que por su especial naturaleza siempre han generado dificultades en los procesos de cálculo numérico. La imposibilidad de representar un irracional de forma exacta en modo numérico (usando las diez cifras del sistema de numeración decimal) es la causa de la mayoría de los problemas. MATLAB representa los resultados con la mayor precisión que puede o con la precisión requerida por el usuario. Los irracionales no se pueden representar exactamente como la razón entre dos enteros. Si se pide la raíz cuadrada del número 17, MATLAB devuelve la cantidad 5.1962 para el formato por defecto.

```
>> sqrt(27)

ans =

    5.1962
```

Existe un grupo importante de números irracionales y reales en general que por su utilización muy común merecen trato especial. MATLAB incorpora los siguientes:

pi	*Número π = 3,1415926*
exp(1)	*Número e = 2,7182818*
inf	*Infinito (por ejemplo 1/0)*
NaN	*Indeterminación (por ejemplo 0/0)*
realmin	*Menor número real positivo utilizable*
realmax	*Mayor número real positivo utilizable*

A continuación se ilustran estos números con salidas de MATLAB.

```
>> format long
>> pi

ans =

   3.14159265358979

>> exp(1)

ans =
```

```
   2.71828182845905
>> 1/0
Warning: Divide by zero.

ans =

   Inf

>> 0/0
Warning: Divide by zero.

ans =
   NaN

>> realmin

ans =
    2.225073858507201e-308

>> realmax

ans =
    1.797693134862316e+308
```

Funciones con argumento real

El conjunto de los números reales es la unión disjunta del conjunto de los números racionales y del conjunto de los números irracionales. Como a su vez el conjunto de los números racionales contiene al conjunto de los números enteros, todas las funciones aplicables a números reales serán válidas también para números irracionales, racionales y enteros. MATLAB dispone de una gama muy completa de funciones predefinidas, la mayoría de las cuales se estudian en capítulos posteriores de este libro. Dentro del grupo de funciones con argumento real que ofrece MATLAB, las más importantes son las siguientes:

Funciones trigonométricas

Función	*Inversa*
sin(x)	**asin(x)**
cos(x)	**acos(x)**
tan(x)	**atan(x) y atan2(x)**
csc(x)	**acsc(x)**
sec(x)	**asec(x)**
cot(x)	**acot(x)**

Funciones hiperbólicas

Función	*Inversa*
sinh(x)	**asinh(x)**
cosh(x)	**acosh(x)**
tanh(x)	**atanh(x)**
csch(x)	**acsch(x)**
sech(x)	**asech(x)**
coth(x)	**acoth(x)**

Funciones exponenciales y logarítmicas

Función	*Significado*
exp(x)	*Función exponencial en base e (e^x)*
log(x)	*Función logaritmo en base e de x*
log10(x)	*Función logaritmo en base 10 de x*
log2(x)	*Función logaritmo en base 2 de x*
pow2(x)	*Función potencia de base 2 de x*
sqrt(x)	*Función raíz cuadrada de x*

Funciones específicas de variable numérica

Función	*Significado*
abs(x)	*Valor absoluto del real x*
floor(x)	*El mayor entero menor o igual que el real x*
ceil(x)	*El menor entero mayor o igual que el real x*
round(x)	*El entero más próximo al real x*
fix(x)	*Elimina la parte decimal del real x*
rem(a,b)	*Da el resto de la división entre los reales a y b*
sign(x)	*Signo del real x (1 si x>0 , -1 si x<0)*

Veamos algunos ejemplos:

```
>> sin(pi/2)

ans =

     1

>> asin(1)

ans =
```

```
   1.57079632679490
>> log(exp(1)^3)

ans =

   3.00000000000000
```

El significado de la función *round* se ilustra en los dos casos siguientes:

```
>> round(2.574)

ans =

     3

>> round(2.4)

ans =

     2
```

El significado de la función *ceil* se ilustra en los dos casos siguientes:

```
>> ceil(4.2)

ans =

     5

>> ceil(4.8)

ans =

     5
```

El significado de la función *floor* lo vemos en los dos ejemplos siguientes:

```
>> floor(4.2)

ans =

     4

>> floor(4.8)

ans =

     4
```

La función *fix* se limita a eliminar la parte decimal de un número real:

```
» fix(5.789)

ans =

     5
```

Números complejos

El trabajo en el campo de los números complejos está perfectamente implementado en MATLAB. Siguiendo la convención de que todas las funciones empiezan con minúscula, una *i* o una *j* minúsculas representan el *número imaginario* $\sqrt{-1}$, que es el valor clave en todo el análisis de variable compleja. Sobre los números complejos pueden ser aplicados los operadores habituales, además de algunas funciones específicas. Tanto la parte real como la parte imaginaria de los números complejos pueden ser constantes simbólicas o cualquier número real, y las operaciones con ellos se realizan siempre en modo exacto, a no ser que en alguna intervenga una aproximación decimal, en cuyo caso se devuelve una aproximación del resultado. Como la unidad imaginaria se representa mediante los símbolos *i* o *j*, los números complejos se expresan en la forma *a+bi* o *a+bj*. Es notorio el hecho de no necesitar el símbolo del producto (el asterisco) antes de la unidad imaginaria:

```
>> (1-5i)*(1-i)/(-1+2i)

ans =

  -1.6000 + 2.8000i

>> format rat
>> (1-5i)*(1-i)/(-1+2i)

ans =

    -8/5      +   14/5i
```

Funciones con argumento complejo

El trabajo con variable compleja es muy importante en análisis matemático y en sus aplicaciones en ramas importantes de la ingeniería. MATLAB no solamente implementa la posibilidad de operar con números complejos, sino que además incorpora varias funciones con variable compleja. A continuación se presenta un resumen de las más importantes.

Funciones trigonométricas

Función	*Inversa*
sin(Z)	**asin(Z)**
cos(Z)	**acos(Z)**
tan(Z)	**atan(Z) y atan2(Z)**
csc(Z)	**acsc(Z)**
sec(Z)	**asec(Z)**
cot(Z)	**acot(Z)**

Funciones hiperbólicas

Función	*Inversa*
sinh(Z)	**asinh(Z)**
cosh(Z)	**acosh(Z)**
tanh(Z)	**atanh(Z)**
csch(Z)	**acsch(Z)**
sech(Z)	**asech(Z)**
coth(Z)	**acoth(Z)**

Funciones exponenciales y logarítmicas

Función	*Significado*
exp(Z)	*Función exponencial en base e (e^Z)*
log(Z)	*Función logaritmo en base e de Z*
log10(Z)	*Función logaritmo en base 10 de Z*
log2(Z)	*Función logaritmo en base 2 de Z*
pow2(Z)	*Función potencia de base 2 de Z*
sqrt(Z)	*Función raíz cuadrada de Z*

Funciones específicas para la parte real e imaginaria

Función	*Significado*
floor(Z)	*Aplica la función floor a real(Z) y a imag(Z)*
ceil(Z)	*Aplica la función ceil a real(Z) y a imag(Z)*
round(Z)	*Aplica la función round a real(Z) y a imag(Z)*
fix(Z)	*Aplica la función fix a real(Z) y a imag(Z)*

Funciones específicas para la parte real e imaginaria

Función	*Significado*
abs(Z)	*Módulo del complejo Z*
angle(Z)	*Argumento del complejo Z*
conj(Z)	*Conjugado del complejo Z*
real(Z)	*Parte real del complejo Z*
imag(Z)	*Parte imaginaria del complejo Z*

A continuación se presentan algunos ejemplos de operaciones con números complejos.

```
>> round(1.5-3.4i)

ans =

      2       -    3i

>> real(i^i)

ans =

    0.2079


>> (2+2i)^2/(-3-3*sqrt(3)*i)^90

ans =

 -1.0502e-085 +7.4042e-070i

>> sin(1+i)

ans =

  1.2985 + 0.6350i
```

Funciones elementales que admiten como argumento un vector complejo *v*

MATLAB es un software que maneja perfectamente el cálculo vectorial y matricial. Su propio nombre, *laboratorio matricial*, ya da idea de su potencia para el trabajo con vectores y matrices. MATLAB permite trabajar con funciones de variable compleja, pero además esta variable puede ser vectorial e incluso matricial. A continuación se presenta una tabla con las funciones de variable compleja vectorial que incorpora MATLAB.

max(V)	*Mayor componente (para complejos se calcula max(abs(V)))*
min(V)	*Menor componente (para complejos se calcula min(abs(V)))*
mean(V)	*Media de las componentes de V*
median(V)	*Mediana de las componentes de V*
std(V)	*Desviación típica de las componentes de V*
sort(V)	*Ordena de forma ascendente las componentes de V. Para complejos hace la ordenación según los valores absolutos*
sum(V)	*Suma las componentes de V*
prod(V)	*Multiplica los elementos de V, con lo que* ***n! = prod(1:n)***
cumsum(V)	*Da el vector de sumas acumuladas de V*
cumprod(V)	*Da el vector de productos acumulados de V*
diff(V)	*Da el vector de primeras diferencias de V (V_t - V_{t-1})*
gradient(V)	*Aproxima el gradiente de V*
del2(V)	*Laplaciano de V (discreto de 5 puntos)*
fft(V)	*Transformada discreta de Fourier de V*
fft2(V)	*Transformada discreta bidimensional de Fourier de V*
ifft(V)	*Inversa de la transformada discreta de Fourier de V*
ifft2(V)	*Inversa de la transformada 2-D discreta de Fourier de V*

Estas funciones también admiten como argumento una matriz compleja, en cuyo caso el resultado es un vector fila cuyas componentes son los resultados de aplicar la función a cada columna de la matriz.

Veamos algunos ejemplos:

```
>> V=2:7, W=[2-i  4i  5+3i]

V =

     2     3     4     5     6     7

W =

   2.0000 - 1.0000i         0 + 4.0000i   5.0000 + 3.0000i

>> diff(V),diff(W)

ans =

   1     1     1     1     1

ans =

  -2.0000 + 5.0000i   5.0000 - 1.0000i
```

```
>> cumprod(V),cumsum(V)

ans =

  2           6          24         120         720        5040

ans =

  2     5     9    14    20    27

>> cumsum(W), mean(W), std(W), sort(W), sum(W)

ans =

   2.0000 - 1.0000i   2.0000 + 3.0000i   7.0000 + 6.0000i

ans =

   2.3333 + 2.0000i

ans =

    3.6515

ans =

   2.0000 - 1.0000i        0 + 4.0000i   5.0000 + 3.0000i

ans =

   7.0000 + 6.0000i

>> mean(V), std(V), sort(V), sum(V)

ans =

    4.5000

ans =

    1.8708

ans =

     2     3     4     5     6     7

ans =

    27
```

```
>> fft(W), ifft(W), fft2(W)

ans =

   7.0000 + 6.0000i   0.3660 - 0.1699i  -1.3660 - 8.8301i

ans =

   2.3333 + 2.0000i  -0.4553 - 2.9434i   0.1220 - 0.0566i

ans =

   7.0000 + 6.0000i   0.3660 - 0.1699i  -1.3660 - 8.8301i
```

Funciones elementales que admiten como argumento una matriz compleja *Z*

• ***Trigonométricas***	
sin(Z)	*Función seno*
sinh(Z)	*Función seno hiperbólico*
asin(Z)	*Función arcoseno*
asinh(Z)	*Función arcoseno hiperbólico*
cos(Z)	*Función coseno*
cosh(Z)	*Función coseno hiperbólico*
acos(Z)	*Función arcocoseno*
acosh(Z)	*Función arcocoseno hiperbólico*
tan(Z)	*Función tangente*
tanh(Z)	*Función tangente hiperbólica*
atan(Z)	*Función arcotangente*
atan2(Z)	*Función arcotangente en el cuarto cuadrante*
atanh(Z)	*Función arcotangente hiperbólica*
sec(Z)	*Función secante*
sech(Z)	*Función secante hiperbólica*
asec(Z)	*Función arcosecante*
asech(Z)	*Función arcosecante hiperbólica*
csc(Z)	*Función cosecante*
csch(Z)	*Función cosecante hiperbólica*
acsc(Z)	*Función arcocosecante*
acsch(Z)	*Función arcocosecante hiperbólica*
cot(Z)	*Función cotangente*
coth(Z)	*Función cotangente hiperbólica*
acot(Z)	*Función arcocotangente*
acoth(Z)	*Función arcocotangente hiperbólica*

• ***Exponenciales***	
exp(Z)	*Función exponencial de base e*
log(Z)	*Función logaritmo neperiano*
log10(Z)	*Función logaritmo decimal*
sqrt(Z)	*Función raíz cuadrada*
• ***Complejas***	
abs(Z)	*Módulo o valor absoluto*
angle(Z)	*Argumento*
conj(Z)	*Complejo conjugado*
imag(Z)	*Parte imaginaria*
real(Z)	*Parte real*
• ***Numéricas***	
fix(Z)	*Elimina las partes decimales*
floor(Z)	*Redondea los decimales al menor entero más cercano*
ceil(Z)	*Redondea los decimales al mayor entero más cercano*
round(Z)	*Efectúa el redondeo común de decimales*
rem(Z1, Z2)	*Resto de la división de los términos de Z1 y Z2*
sign(Z)	*Función signo*
• ***Matriciales***	
expm(Z)	*Función exponencial matricial por defecto*
expm1(Z)	*Función exponencial matricial en M-fichero*
expm2(Z)	*Función exponencial matricial vía series de Taylor*
expm3(Z)	*Función exponencial matricial vía autovalores*
logm(Z)	*Función logarítmica matricial*
sqrtm(Z)	*Función raíz cuadrada matricial*
funm(Z,'función')	*Aplica la función a la matriz Z*

Veamos algunos ejemplos:

```
>> A=[7 8 9; 1 2 3; 4 5 6], B=[1+2i 3+i;4+i,i]

A =

     7     8     9
     1     2     3
     4     5     6

B =

   1.0000 + 2.0000i   3.0000 + 1.0000i
   4.0000 + 1.0000i         0 + 1.0000i
```

```
>> sin(A), sin(B), exp(A), exp(B), log(B), sqrt(B)

ans =

    0.6570    0.9894    0.4121
    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794

ans =

   3.1658 + 1.9596i   0.2178 - 1.1634i
  -1.1678 - 0.7682i         0 + 1.1752i

ans =

  1.0e+003 *

    1.0966    2.9810    8.1031
    0.0027    0.0074    0.0201
    0.0546    0.1484    0.4034

ans =

  -1.1312 + 2.4717i  10.8523 +16.9014i
  29.4995 +45.9428i   0.5403 + 0.8415i

ans =

   0.8047 + 1.1071i   1.1513 + 0.3218i
   1.4166 + 0.2450i         0 + 1.5708i

ans =

   1.2720 + 0.7862i   1.7553 + 0.2848i
   2.0153 + 0.2481i   0.7071 + 0.7071i
```

Las funciones exponencial, raíz cuadrada y logaritmo usadas anteriormente se aplican elemento a elemento a la matriz y no tienen nada que ver con las funciones matriciales exponenciales y logarítmicas que se usan a continuación.

```
>> expm(B), logm(A), abs(B), imag(B)

ans =

 -27.9191 +14.8698i -20.0011 +12.0638i
 -24.7950 +17.6831i -17.5059 +14.0445i
```

```
ans =

   11.9650   12.8038  -19.9093
  -21.7328  -22.1157   44.6052
   11.8921   12.1200  -21.2040

ans =

    2.2361    3.1623
    4.1231    1.0000

ans =

     2     1
     1     1

>> fix(sin(B)), ceil(log(A)), sign(B), rem(A,3*ones(3))

ans =

   3.0000 + 1.0000i        0 - 1.0000i
  -1.0000                  0 + 1.0000i

ans =

     2     3     3
     0     1     2
     2     2     2

ans =

   0.4472 + 0.8944i   0.9487 + 0.3162i
   0.9701 + 0.2425i        0 + 1.0000i

ans =

     1     2     0
     1     2     0
     1     2     0
```

Números aleatorios

MATLAB trata perfectamente la generación automática de números aleatorios. Proporciona la función *rand* para generar números aleatorios distribuidos uniformemente y la función *randn* para generar números aleatorios distribuidos normalmente. Las funciones más interesantes de MATLAB que generan números aleatorios se presentan en la tabla siguiente.

rand	*Devuelve un número decimal aleatorio distribuido uniformemente en el intervalo [0,1]*
rand(n)	*Devuelve una matriz de dimensión nxn cuyos elementos son números decimales aleatorios distribuidos uniformemente en el intervalo [0,1]*
rand(m,n)	*Devuelve una matriz de dimensión mxn cuyos elementos son números decimales aleatorios distribuidos uniformemente en el intervalo [0,1]*
rand(size(A))	*Devuelve una matriz del mismo tamaño que la matriz A y cuyos elementos son números decimales aleatorios distribuidos uniformemente en el intervalo [0,1]*
rand('seed')	*Devuelve el valor actual de la semilla generadora de los números aleatorios uniformes*
rand('seed',n)	*Coloca en la cantidad n el valor actual de la semilla generadora de los números aleatorios uniformes*
randn	*Devuelve un número decimal aleatorio distribuido según una normal de media 0 y varianza 1*
randn(n)	*Devuelve una matriz de dimensión nxn cuyos elementos son números decimales aleatorios distribuidos según una normal de media 0 y varianza 1*
randn(m,n)	*Devuelve una matriz de dimensión mxn cuyos elementos son números decimales aleatorios distribuidos según una normal de media 0 y varianza 1*
randn(size(A))	*Devuelve una matriz del mismo tamaño que la matriz A y cuyos elementos son números decimales aleatorios distribuidos según una normal de media 0 y varianza 1*
randn('seed')	*Devuelve el valor actual de la semilla generadora de los números aleatorios normales*
randn('seed',n)	*Coloca en la cantidad n el valor actual de la semilla generadora de los números aleatorios uniformes*

Veamos algunos ejemplos:

```
>> [rand, rand(1), randn, randn(1)]

ans =

    0.9501    0.2311   -0.4326   -1.6656

>> [rand(2), randn(2)]

ans =

    0.6068    0.8913    0.1253   -1.1465
    0.4860    0.7621    0.2877    1.1909

>> [rand(2,3), randn(2,3)]

ans =

    0.3529    0.0099    0.2028   -0.1364    1.0668   -0.0956
    0.8132    0.1389    0.1987    0.1139    0.0593   -0.8323
```

3.3 Operadores

MATLAB dispone de operadores para denotar las operaciones aritméticas, lógicas, relacionales, condicionales y de estructura.

Operadores aritméticos

Existen en MATLAB dos tipos de operaciones aritméticas: las operaciones aritméticas matriciales, que se rigen por las reglas del álgebra lineal, y las operaciones aritméticas con vectores, que se realizan elemento a elemento. Los operadores involucrados se presentan en la tabla siguiente.

Operador	***Función que desempeña***
+	*Suma de escalares, vectores o matrices*
-	*Resta de escalares, vectores o matrices*
*	*Producto de escalares o de matrices*
.*	*Producto de escalares o de vectores*
\	*A\B = inv(A)*B, siendo A y B matrices*
.\	*A.\B = [B(i,j)/A(i,j)], siendo A y B vectores [dim(A)=dim(B)]*
/	*Cociente escalar o B/A = B*inv(A), siendo A y B matrices*
./	*A./B = [A(i,j)/B(i,j)], siendo A y B vectores [dim(A)=dim(B)]*
^	*Potencia de escalares o potencia escalar de matriz (M^p)*
.^	*Potencia de vectores (A.^B = [A(i,j)$^{B(i,j)}$], A y B vectores)*

Las operaciones matemáticas simples entre escalares y vectores aplican el escalar a todos los elementos del vector según la operación definida, y las operaciones simples entre vectores se realizan elemento a elemento. A continuación se presenta una especificación más amplia de estos operadores:

a = {a1, a2, ..., an}, b = {b1, b2, ..., bn} c = escalar	
a + c = [a1+c a2+c ..., an+c]	*Suma de un escalar y un vector*
a * c = [a1*c a2*c ... an*c]	*Producto de un escalar por un vector*
a + b = [a1+b1 a2+b2 ... an+bn]	*Suma de dos vectores*
a. * b = [a1*b1 a2*b2 ... an*bn]	*Producto de dos vectores*
a. / b = [a1/b1 a2/b2 ... an/bn]	*Cociente a la derecha de dos vectores*
a. \ b = [a1\b1 a2\b2 ... an\bn]	*Cociente a la izquierda de dos vectores*
a.^c = [a1^c a2^c ... an^c]	*Vector elevado a escalar*
c.^a = [c^a1 c^a2 ... c^an]	*Escalar elevado a vector*
a.^b = [a1^b1 a2^b2 ... an^bn]	*Vector elevado a vector*

Hay que tener presente que los vectores han de ser de la misma longitud y que en el producto, cociente y potencia el primer operando va seguido de un punto.

A continuación se presenta un ejemplo que involucra este tipo de operadores.

```
>> X=[5,4,3]; Y=[1,2,7]; a=X+Y, b=X-Y, c=X.*Y, d=2.*X,...
e=2./X, f=2.\Y, g=X./Y, h=Y.\X, i=X.^2, j=2.^X, k=X.^Y

a =

     6     6    10

b =

     4     2    -4

c =

     5     8    21

d =

    10     8     6

e =

    0.4000    0.5000    0.6667

f =

    0.5000    1.0000    3.5000

g =

    5.0000    2.0000    0.4286

h =

    5.0000    2.0000    0.4286

i =

    25    16     9

j =

    32    16     8

k =

    5          16        2187
```

Las operaciones anteriores tienen todas sentido, ya que las variables operandos son vectores de la misma dimensión en todos los casos, con lo que las operaciones se realizarán elemento a elemento (para la suma y la diferencia no hay distinción entre vectores y matrices, pues son operaciones idénticas en los dos casos).

En cuanto a los operadores con variables matriciales, las más importantes se especifican a continuación:

A+B, A-B, A*B	*Suma, resta y producto de matrices*
A\B	*Si A es cuadrada, A\B=inv(A)*B. Si A no es cuadrada, A\B es la solución en el sentido de mínimos cuadrados del sistema AX=B*
B/A	*Coincide con (A' \ B')'*
$\mathbf{A^n}$	*Coincide con A*A*A*...*A n veces (n escalar)*
$\mathbf{p^A}$	*Realiza el cálculo sólo si p es un escalar*

Veamos algunos ejemplos:

```
>> X=[5,4,3]; Y=[1,2,7]; l=X'*Y, m=X*Y', n=2*X, o=X/Y, p=Y\X

l =
     5    10    35
     4     8    28
     3     6    21
m =
    34
n =
    10     8     6
o =
    0.6296
p =
         0         0         0
         0         0         0
    0.7143    0.5714    0.4286
```

Todas las operaciones anteriores están definidas de forma matricial con la dimensión adecuada. No olvidemos que un vector es un caso particular de matriz, pero para operar con él de forma matricial (no elemento a elemento) es necesario respetar las reglas de dimensionalidad para operaciones matriciales. Las operaciones vectoriales *X.'*Y* y *X.*Y'* no tienen sentido, ya que se están operando vectores de distinta dimensión. Las operaciones matriciales *X*Y*, *2/X*, *2\Y*, *X^2*, *2^X* y *X^Y* no tienen sentido, ya que se cometen errores de dimensionalidad con las matrices.

A continuación se presentan más ejemplos sobre operadores matriciales.

```
>> M=[1,2,3;1,0,2;7,8,9]

M =

     1     2     3
     1     0     2
     7     8     9

>> B=inv(M), C=M^2, D=M^(1/2), E=2^M

B =

   -0.8889    0.3333    0.2222
    0.2778   -0.6667    0.0556
    0.4444    0.3333   -0.1111

C =

    24    26    34
    15    18    21
    78    86   118

D =

   0.5219 + 0.8432i   0.5793 - 0.0664i   0.7756 - 0.2344i
   0.3270 + 0.0207i   0.3630 + 1.0650i   0.4859 - 0.2012i
   1.7848 - 0.5828i   1.9811 - 0.7508i   2.6524 + 0.3080i

E =

  1.0e+003 *

    0.8626    0.9568    1.2811
    0.5401    0.5999    0.8027
    2.9482    3.2725    4.3816
```

Operadores relacionales

MATLAB también ofrece símbolos para denotar las operaciones relacionales. Los operadores relacionales ejecutan comparaciones elemento a elemento entre dos matrices y devuelven una matriz del mismo tamaño cuyos elementos son ceros si la correspondiente relación es cierta, o unos si la correspondiente relación es falsa. Los operadores relacionales también permiten comparar escalares con vectores o matrices, en cuyo caso se compara el escalar con todos los elementos de la matriz. A continuación se presenta una tabla con estos operadores.

<	*Menor (para complejos sólo afecta a partes reales)*
<=	*Menor o igual (sólo afecta a partes reales)*
>	*Mayor (sólo afecta a partes reales)*
>=	*Mayor o igual (sólo afecta a partes reales)*
x == y	*Igualdad (afecta a los números complejos)*
x ~= y	*Desigualdad (afecta a los números complejos)*

Operadores lógicos

MATLAB ofrece símbolos para denotar las operaciones lógicas. Los operadores lógicos ofrecen un camino para combinar o negar expresiones relacionales. La tabla siguiente presenta este tipo de operadores.

~A	*Negación lógica (NOT) o complementario de A*
A & B	*Conjunción lógica (AND) o intersección de A y B*
A | B	*Disyunción lógica (OR) o unión de A y B*
xor(A,B)	*OR exclusivo (XOR) o diferencia simétrica de A y B (vale 1 si A o B, pero no ambos, son 1)*

Veamos algunos ejemplos:

```
>> A=2:7;P=(A>3)&(A<6)

P =

     0     0     1     1     0     0
```

Devuelve 1 cuando *A* es mayor que 3 y menor que 6, y devuelve 0 en caso contrario.

```
>> X=3*ones(3,3); X>= [7 8 9; 4 5 6; 1 2 3]

ans =

     0     0     0
     0     0     0
     1     1     1
```

Los elementos de la matriz *X* que son mayores o iguales que los de la matriz *[7 8 9; 4 5 6; 1 2 3]* se corresponden con un 1 en la matriz respuesta. El resto de los elementos se corresponden con un 0.

Funciones lógicas

MATLAB implementa funciones lógicas cuya salida es del tipo verdadero (valor 1) o falso (valor 0). La tabla siguiente presenta las más importantes.

exist(A)	*Chequea si la variable o función A existe (devuelve 0 si A no existe y un número entre 1 y 5, según el tipo, si existe)*
any(V)	*Devuelve 0 si todos los elementos del vector V son nulos y devuelve 1 si algún elemento de V es no nulo*
any(A)	*Devuelve 0 para cada columna de la matriz A con todos los elementos nulos y devuelve 1 para cada columna de la matriz A con alguno de sus elementos no nulo*
all(V)	*Devuelve 1 si todos los elementos del vector V son no nulos y devuelve 0 si algún elemento de V es nulo*
all(A)	*Devuelve 1 para cada columna de la matriz A con todos los elementos no nulos y devuelve 0 para cada columna de la matriz A con alguno de sus elementos nulo*
find(V)	*Devuelve los lugares (o índices) que ocupan los elementos no nulos del vector V*
isnan(V)	*Devuelve 1 para los elementos de V que son indeterminados y devuelve 0 para los que no lo son*
isinf(V)	*Devuelve 1 para los elementos de V que son infinitos y devuelve 0 para los que no lo son*
isfinite(V)	*Devuelve 1 para los elementos de V que son finitos y devuelve 0 para los que no lo son*
isempty(A)	*Devuelve 1 si A es una matriz vacía y devuelve 0 en otro caso (una matriz vacía es la que tiene una de sus dimensiones 0)*
issparse(A)	*Devuelve 1 si A es una matriz por cajas y devuelve 0 en otro caso*
isreal(V)	*Devuelve 1 si todos los elementos de V son reales y 0 en otro caso*
isprime(V)	*Devuelve 1 para todos los elementos de V que son primos y devuelve 0 para los elementos de V no primos*
islogical(V)	*Devuelve 1 si V es un vector lógico y 0 en otro caso*
isnumeric(V)	*Devuelve 1 si V es un vector numérico y 0 en otro caso*
ishold	*Devuelve 1 si retienen las propiedades del gráfico actual para el siguiente y sólo se añaden las nuevas y 0 en caso contrario*
isieee	*Devuelve 1 para computadores IEEE*
isstr(S)	*Devuelve 1 si S es una cadena y 0 en caso contrario*
ischart(S)	*Devuelve 1 si S es una cadena y 0 en caso contrario*
isglobal(A)	*Devuelve 1 si A es una variable global y 0 en otro caso*
isletter(S)	*Devuelve 1 si S es una letra del alfabeto y 0 en otro caso*
isequal(A,B)	*Devuelve 1 si las matrices o vectores A y B son iguales y 0 en otro caso*
ismember(V,W)	*Devuelve 1 para cada elemento de V que está en W y 0 para cada elemento de V que no está en W*

A continuación se presentan algunos ejemplos sobre las funciones lógicas definidas.

```
>> V=[1,2,3,4,5,6,7,8,9], isprime(V), isnumeric(V), all(V), any(V)
V =
     1     2     3     4     5     6     7     8     9
ans =
     0     1     1     0     1     0     1     0     0
ans =
     1
ans =
     1
ans =
     1
>> B=[Inf, -Inf, pi, NaN], isinf(B), isfinite(B), isnan(B), isreal(B)
B =
       Inf      -Inf    3.1416       NaN
ans =
     1     1     0     0
ans =
     0     0     1     0
ans =
     0     0     0     1
ans =
     1
>> ismember([1,2,3],[8,12,1,3]),A=[2,0,1];B=[4,0,2];isequal(2*A,B)
ans =
     1     0     1
ans =
     1
```

Ejercicio 3-1. Hallar las combinaciones sin repetición de 30 elementos tomados de 12 en 12, el resto de la división de 2^{134} entre 3, la descomposición en factores primos del número 18900, el factorial de 200 y el número N que al dividirlo por 16,24,30 y 32 da de resto 5.

```
 >> factorial(30)/(factorial(12)*factorial(30-12))
ans =
  8.6493e+007
```

A continuación se utiliza el comando *vpa* para presentar el resultado exacto.

```
>> vpa 'factorial(30)/(factorial(12)*factorial(30-12))' 15
ans =
86493225.
```

```
>> rem(2^134,3)
ans =
     0
```

```
>> factor(18900)
ans =
     2     2     3     3     3     5     5     7
```

```
>> factorial(100)
ans =
  9.3326e+157
```

Seguidamente se utiliza el comando *vpa* para presentar el resultado exacto.

```
>> vpa 'factorial(100)' 160
ans =
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000.
```

N-5 será el mínimo común múltiplo de 16, 24, 30 y 32.

```
>> lcm(lcm(16,24),lcm(30,32))
ans =
   480
```

Luego N = 480+5=485

Ejercicio 3-2. Obtener en base 5 el resultado de la operación definida mediante $a25aaff6_{16} + 6789aba_{12} + 35671_8 + 1100221_3$ - 1250. Obtener también en base 13 el resultado de la operación $(666551_7)\ (aa199800a_{11})+(fffaaa125_{16}) / (33331_4 +6)$

El resultado de toda la primera operación en base 10 se calcula como sigue:

```
>> base2dec('a25aaf6',16)+base2dec('6789aba',12)+...
   base2dec('35671',8)+base2dec('1100221',3)-1250

ans =

   190096544
```

Pero todavía falta pasar el resultado decimal anterior a base 5.

```
>> dec2base(190096544,5)

ans =

   342131042134
```

Luego, el resultado final de la primera operación en base 5 es 342131042134.

El resultado de toda la segunda operación en base 10 se calcula como sigue:

```
>> base2dec('666551',7)*base2dec('aa199800a',11)+...
   79*base2dec('fffaaa125',16)/(base2dec('33331',4)+6)

ans =

   2.7537e+014
```

Ahora transformamos el resultado a base 13.

```
>> dec2base(275373340490852,13)

ans =

   BA867963C1496
```

Ejercicio 3-3. Obtener, en base 13, el resultado de la operación siguiente: $(666551_7)\ (aa199800a_{11}) + (fffaaa125_{16}) / (33331_4 + 6)$

En primer lugar, realizamos la operación en base 10:

Una forma más directa de hacer todo lo anterior es:

```
>> base2dec('666551',7)*base2dec('aa199800a',11)+…
   79*base2dec('fffaaa125',16)/(base2dec('33331',4)+6)

ans =

    2.753733404908515e+014
```

Ahora transformamos el resultado a base 13.

```
>> dec2base(275373340490852,13)

ans =

BA867963C1496
```

Ejercicio 3-4. Dados los números complejos X=2+2i e Y=-3-3√3 i, calcular Y^3, X^2/Y^{90}, $Y^{1/2}$, $Y^{3/2}$ y log(X)

```
>> X=2+2*i;Y=-3-3*sqrt(3)*i;
>> Y^3

ans =

   216

>> X^2/Y^90

ans =

    -1.050180953422426e-085 +7.404188256695968e-070i

>> sqrt(Y)

ans =

  1.22474487139159 - 2.12132034355964i

>> sqrt(Y^3)

ans =

  14.69693845669907

>> log(X)

ans =

  1.03972077083992 + 0.78539816339745i
```

Ejercicio 3-5. Calcular el valor de las siguientes operaciones con números complejos:

$$\frac{i^8 - i^{-8}}{3-4i} + 1,\ i^{Sen(1+i)},\ (2+Ln(i))^{\frac{1}{i}},\ (1+i)^i,\ i^{Ln(1+i)},\ \left(1+\sqrt{3}i\right)^{1-i}$$

```
>> (i^8-i^(-8))/(3-4*i) + 1

ans =

     1

>> i^(sin(1+i))

ans =

 -0.16665202215166 + 0.32904139450307i

>> (2+log(i))^(1/i)

ans =

  1.15809185259777 - 1.56388053989023i

>> (1+i)^i

ans =

  0.42882900629437 + 0.15487175246425i

>> i^(log(1+i))

ans =

  0.24911518828716 + 0.15081974484717i

>> (1+sqrt(3)*i)^(1-i)

ans =

  5.34581479196611 + 1.97594883452873i
```

Ejercicio 3-6. Calcular parte real, parte imaginaria, módulo y argumento de los siguientes complejos:

$$i^{3+i},\ \left(1+\sqrt{3}i\right)^{1-i},\ i^{i^i},\ i^i$$

```
>> Z1=i^3*i; Z2=(1+sqrt(3)*i)^(1-i); Z3=(i^i)^i;Z4=i^i;

>> format short

>> real([Z1 Z2 Z3 Z4])
```

```
ans =

    1.0000    5.3458    0.0000    0.2079

>> imag([Z1 Z2 Z3 Z4])

ans =

         0    1.9759   -1.0000         0

>> abs([Z1 Z2 Z3 Z4])

ans =

    1.0000    5.6993    1.0000    0.2079

>> angle([Z1 Z2 Z3 Z4])

ans =

         0    0.3541   -1.5708         0
```

Ejercicio 3-7. Generar una matriz cuadrada de orden 4 cuyos elementos sean números aleatorios uniformes [0,1]. Generar otra matriz cuadrada de orden 4 cuyos elementos sean números aleatorios normales [0,1]. Observar las semillas generadoras actuales, cambiarlas al valor ½ y volver a generar las dos matrices de números aleatorios.

```
>> rand(4)

ans =

    0.9501    0.8913    0.8214    0.9218
    0.2311    0.7621    0.4447    0.7382
    0.6068    0.4565    0.6154    0.1763
    0.4860    0.0185    0.7919    0.4057

>> randn(4)

ans =

   -0.4326   -1.1465    0.3273   -0.5883
   -1.6656    1.1909    0.1746    2.1832
    0.1253    1.1892   -0.1867   -0.1364
    0.2877   -0.0376    0.7258    0.1139

>> rand('seed')

ans =

   931316785
```

```
>> randn('seed')

ans =

   931316785

>> randn('seed',1/2)
>> rand('seed',1/2)
>> rand(4)

ans =

    0.2190    0.9347    0.0346    0.0077
    0.0470    0.3835    0.0535    0.3834
    0.6789    0.5194    0.5297    0.0668
    0.6793    0.8310    0.6711    0.4175

>> randn(4)

ans =

    1.1650   -0.6965    0.2641    1.2460
    0.6268    1.6961    0.8717   -0.6390
    0.0751    0.0591   -1.4462    0.5774
    0.3516    1.7971   -0.7012   -0.3600
```

Ejercicio 3-8. Dadas las variables vectoriales a=[π,2π,3π,4π,5π] y b=[e,2e,3e,4e,5e], calcular c= Sen(a)+b, d=Cos(a), e=Ln(b), f=c*d, g=c/d, h=d^2, i=d^2-e^2 y j=3d^3-2e^2 .

```
>>
a=[pi,2*pi,3*pi,4*pi,5*pi],b=[exp(1),2*exp(1),3*exp(1),4*exp(1),
    5*exp(1)],c=sin(a)+b,d=cos(a),e=log(b),f=c.*d,g=c./d,
    h=d.^2, i=d.^2-e.^2, j=3*d.^3-2*e.^2

a =

    3.1416    6.2832    9.4248   12.5664   15.7080


b =

    2.7183    5.4366    8.1548   10.8731   13.5914


c =

    2.7183    5.4366    8.1548   10.8731   13.5914
```

```
d =

    -1     1    -1     1    -1


e =

    1.0000    1.6931    2.0986    2.3863    2.6094


f =

   -2.7183    5.4366   -8.1548   10.8731  -13.5914


g =

   -2.7183    5.4366   -8.1548   10.8731  -13.5914


h =

     1     1     1     1     1


i =

         0   -1.8667   -3.4042   -4.6944   -5.8092


j =

   -5.0000   -2.7335  -11.8083   -8.3888  -16.6183
```

Ejercicio 3-9. Dada una M matriz cuadrada aleatoria uniforme de orden 3, obtener su inversa, su transpuesta y su diagonal. Transformarla en una matriz triangular inferior y en otra superior y rotarla 90 grados. Obtener la suma de los elementos de la primera fila y la suma de los elementos de la diagonal. Extraer la submatriz cuya diagonal son los elementos a_{11} y a_{22} y extraer también la submatriz cuyos elementos de la diagonal son a_{11} y a_{33}.

```
>> M=rand(3)

M =

    0.6868    0.8462    0.6539
    0.5890    0.5269    0.4160
    0.9304    0.0920    0.7012
```

```
>> A=inv(M)

A =

   -4.1588    6.6947   -0.0934
    0.3255    1.5930   -1.2487
    5.4758   -9.0924    1.7138

>> B=M'

B =

    0.6868    0.5890    0.9304
    0.8462    0.5269    0.0920
    0.6539    0.4160    0.7012

>> V=diag(M)

V =

    0.6868
    0.5269
    0.7012

>> TI=tril(M)

TI =

    0.6868         0         0
    0.5890    0.5269         0
    0.9304    0.0920    0.7012

>> TS=triu(M)

TS =

    0.6868    0.8462    0.6539
         0    0.5269    0.4160
         0         0    0.7012

>> TR=rot90(M)

TR =

    0.6539    0.4160    0.7012
    0.8462    0.5269    0.0920
    0.6868    0.5890    0.9304
```

```
>> s=M(1,1)+M(1,2)+M(1,3)

s =

    2.1869

>> sd=M(1,1)+M(2,2)+M(3,3)

sd =

    1.9149

>> SM=M(1:2,1:2)

SM =

    0.6868    0.8462
    0.5890    0.5269

>> SM1=M([1 3], [1 3])

SM1 =

    0.6868    0.6539
    0.9304    0.7012
```

Ejercicio 3-10. Dada la matriz M cuadrada compleja de orden 3, obtener su cuadrado, su raíz cuadrada y sus exponenciales de bases 2 y -2.

$$M = \begin{bmatrix} i & 2i & 3i \\ 4i & 5i & 6i \\ 7i & 8i & 9i \end{bmatrix}$$

```
>>  M=[i 2*i 3*i; 4*i 5*i 6*i; 7*i 8*i 9*i]

M =

        0 + 1.0000i         0 + 2.0000i         0 + 3.0000i
        0 + 4.0000i         0 + 5.0000i         0 + 6.0000i
        0 + 7.0000i         0 + 8.0000i         0 + 9.0000i

>> C=M^2

C =

   -30   -36   -42
   -66   -81   -96
  -102  -126  -150
```

```
>> D=M^(1/2)

D =

   0.8570 - 0.2210i   0.5370 + 0.2445i   0.2169 + 0.7101i
   0.7797 + 0.6607i   0.9011 + 0.8688i   1.0224 + 1.0769i
   0.7024 + 1.5424i   1.2651 + 1.4930i   1.8279 + 1.4437i

>> 2^M

ans =

   0.7020 - 0.6146i  -0.1693 - 0.2723i  -0.0407 + 0.0699i
  -0.2320 - 0.3055i   0.7366 - 0.3220i  -0.2947 - 0.3386i
  -0.1661 + 0.0036i  -0.3574 - 0.3717i   0.4513 - 0.7471i

>> (-2)^M

ans =

  17.3946 -16.8443i   4.3404 - 4.5696i  -7.7139 + 7.7050i
   1.5685 - 1.8595i   1.1826 - 0.5045i  -1.2033 + 0.8506i
 -13.2575 +13.1252i  -3.9751 + 3.5607i   6.3073 - 6.0038i
```

Ejercicio 3-11. Dada la matriz compleja M del ejercicio anterior, obtener su logaritmo neperiano elemento a elemento y su exponencial de base elemento a elemento. Realizar también las operaciones matrices e^M y Ln(M).

```
>> M=[i 2*i 3*i; 4*i 5*i 6*i; 7*i 8*i 9*i]

>> log(M)

ans =

        0 + 1.5708i   0.6931 + 1.5708i   1.0986 + 1.5708i
   1.3863 + 1.5708i   1.6094 + 1.5708i   1.7918 + 1.5708i
   1.9459 + 1.5708i   2.0794 + 1.5708i   2.1972 + 1.5708i

>> exp(M)

ans =

   0.5403 + 0.8415i  -0.4161 + 0.9093i  -0.9900 + 0.1411i
  -0.6536 - 0.7568i   0.2837 - 0.9589i   0.9602 - 0.2794i
   0.7539 + 0.6570i  -0.1455 + 0.9894i  -0.9111 + 0.4121i
```

```
>> logm(M)

ans =

  -5.4033 - 0.8472i  11.9931 - 0.3109i  -5.3770 + 0.8846i
  12.3029 + 0.0537i -22.3087 + 0.8953i  12.6127 + 0.4183i
  -4.7574 + 1.6138i  12.9225 + 0.7828i  -4.1641 + 0.6112i

>> expm(M)

ans =

   0.3802 - 0.6928i  -0.3738 - 0.2306i  -0.1278 + 0.2316i
  -0.5312 - 0.1724i   0.3901 - 0.1434i  -0.6886 - 0.1143i
  -0.4426 + 0.3479i  -0.8460 - 0.0561i  -0.2493 - 0.4602i
```

Ejercicio 3-12. Dado el vector complejo V=[i,1-i,1+i], hallar la media, mediana, desviación típica, varianza, suma, producto, máximo y mínimo de sus elementos, así como su gradiente, la transformada discreta de Fourier y su inversa.

```
>> [mean(V),median(V),std(V),var(V),sum(V),prod(V),max(V),min(V)]'

ans =
   0.6667 - 0.3333i
   1.0000 + 1.0000i
   1.2910
   1.6667
   2.0000 - 1.0000i
        0 - 2.0000i
   1.0000 + 1.0000i
        0 - 1.0000i

>> gradient(V)

ans =
   1.0000 - 2.0000i   0.5000                  0 + 2.0000i

>> fft(V)
ans =

   2.0000 + 1.0000i  -2.7321 + 1.0000i   0.7321 + 1.0000i

>> ifft(V)

ans =
   0.6667 + 0.3333i   0.2440 + 0.3333i  -0.9107 + 0.3333i
```

Ejercicio 3-13. Dadas las matrices:

$$A=\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\\ 0 & 0 & 1\end{bmatrix} \quad B=\begin{bmatrix}i & 1-i & 2+i\\ 0 & -1 & 3-i\\ 0 & 0 & -i\end{bmatrix} \quad C=\begin{bmatrix}1 & 1 & 1\\ 0 & sqrt(2)i & -sqrt(2)i\\ 1 & -1 & -1\end{bmatrix}$$

Calcular AB - BA , $A^2 + B^2 + C^2$, ABC, sqrt(A)+ sqrt(B) - sqrt(C), e^A (e^B+ e^C), sus inversas y sus transpuestas. Comprobar también que al multiplicar cualquier matriz por su inversa se obtiene la matriz identidad

```
>> A=[1 1 0;0 1 1;0 0 1]; B=[i 1-i 2+i;0 -1 3-i;0 0 -i];
   C=[1 1 1; 0 sqrt(2)*i -sqrt(2)*i;1 -1 -1];

>> M1=A*B-B*A

M1 =

        0             -1.0000 - 1.0000i   2.0000
        0                   0             1.0000 - 1.0000i
        0                   0                   0

>> M2=A^2+B^2+C^2

M2 =

   2.0000             2.0000 + 3.4142i   3.0000 - 5.4142i
        0 - 1.4142i  -0.0000 + 1.4142i   0.0000 - 0.5858i
        0             2.0000 - 1.4142i   2.0000 + 1.4142i

>> M3=A*B*C

M3 =

   5.0000 + 1.0000i  -3.5858 + 1.0000i  -6.4142 + 1.0000i
   3.0000 - 2.0000i  -3.0000 + 0.5858i  -3.0000 + 3.4142i
        0 - 1.0000i        0 + 1.0000i        0 + 1.0000i

>> M4=sqrtm(A)+sqrtm(B)-sqrtm(C)

M4 =

   0.6356 + 0.8361i  -0.3250 - 0.8204i   3.0734 + 1.2896i
   0.1582 - 0.1521i   0.0896 + 0.5702i   3.3029 - 1.8025i
  -0.3740 - 0.2654i   0.7472 + 0.3370i   1.2255 + 0.1048i
```

```
>> M5=expm(A)*(expm(B)+expm(C))

M5 =

  14.1906 - 0.0822i   5.4400 + 4.2724i  17.9169 - 9.5842i
   4.5854 - 1.4972i   0.6830 + 2.1575i   8.5597 - 7.6573i
   3.5528 + 0.3560i   0.1008 - 0.7488i   3.2433 - 1.8406i

>> inv(A)

ans =

     1    -1     1
     0     1    -1
     0     0     1

>> inv(B)

ans =

        0 - 1.0000i  -1.0000 - 1.0000i  -4.0000 + 3.0000i
        0            -1.0000             1.0000 + 3.0000i
        0                  0                  0 + 1.0000i

>> inv(C)

ans =

   0.5000                  0             0.5000
   0.2500                  0 - 0.3536i  -0.2500
   0.2500                  0 + 0.3536i  -0.2500

>> [A*inv(A) B*inv(B) C*inv(C)]

ans =

     1     0     0     1     0     0     1     0     0
     0     1     0     0     1     0     0     1     0
     0     0     1     0     0     1     0     0     1

>> A'

ans =

     1     0     0
     1     1     0
     0     1     1
```

```
>> B'

ans =

        0 - 1.0000i         0                   0
   1.0000 + 1.0000i   -1.0000                   0
   2.0000 - 1.0000i    3.0000 + 1.0000i        0 + 1.0000i

>> C'

ans =

   1.0000                   0              1.0000
   1.0000                   0 - 1.4142i   -1.0000
   1.0000                   0 + 1.4142i   -1.0000
```

4

Funciones del entorno de desarrollo de MATLAB

4.1 Comandos de propósito general

MATLAB dispone de un grupo de comandos denominados de propósito general que pueden ser clasificados en varias categorías. Estas categorías, de acuerdo con la función esencial de los comandos podrían ser:

- Comandos que manejan variables en el espacio de trabajo.
- Comandos que trabajan con ficheros y el entorno operativo.
- Comandos que manejan funciones.
- Comandos que controlan la ventana *Command Window.*
- Comandos de comienzo y salida de MATLAB.

Comandos que manejan variables en el espacio de trabajo

MATLAB permite definir y manejar las variables, así como almacenarlas en ficheros, de modo muy simple. Cuando se realizan cálculos extensos, es conveniente dar nombres a resultados intermedios. Estos resultados intermedios se asignan a variables para hacer más fácil su uso. La definición de variables ya ha sido tratada en el capítulo anterior, pero es conveniente recordar que el valor asignado a una variable es permanente, hasta que no se cambie expresamente o hasta que no se salga de la presente sesión de MATLAB.

En MATLAB existe un grupo de comandos que manejan variables y que se presentan en la tabla siguiente:

clear	*Borra todas las variables del espacio de trabajo*
clear(v1,v2, ..., vn)	*Borra las variables numéricas especificadas*
clear('v1', 'v2', ..., 'vn')	*Borra las variables cadena especificadas*
disp(X)	*Muestra un array sin incluir su nombre*
length(X)	*Muestra la longitud del vector X y, si X es una matriz o array, muestra su mayor dimensión*
load	*Lee todas las variables del fichero MATLAB.mat*
load fichero	*Lee todas las variables del fichero .mat especificado*
load fichero X Y Z	*Lee las variables X, Y, Z del fichero .mat especificado*
load fichero -ascii	*Lee el fichero como ASCII sea cual sea su extensión*
load fichero -mat	*Lee el fichero como .mat sea cual sea su extensión*
S = load(...)	*Devuelve el contenido de un fichero .mat en la variable S*
memory	*Ayuda cuando hay limitaciones de memoria*
mlock	*Previene el borrado del M-fichero en memoria*
munlock	*Permite el borrado del M-fichero en memoria*
openvar('v')	*Abre la variable v del espacio de trabajo en el Array Editor, para edición gráfica*
pack	*Comprime la memoria del espacio de trabajo*
pack fichero	*Usa fichero como temporal para guardar las variables*
pack 'fichero'	*Forma funcional de pack*
save	*Guarda las variables del espacio de trabajo en el fichero binario MATLAB.mat en el directorio actual*
save fichero	*Guarda las variables del espacio de trabajo en el fichero fichero.mat en el directorio actual.* ***Un fichero .mat tiene formato específico de MATLAB***
save fichero v1 v2 ...	*Guarda las variables del espacio de trabajo v1, v2, ... en el fichero fichero.mat*
save ... opción	*Guarda las variables del espacio de trabajo en el formato especificado por opción*
save('fichero', ...)	*Forma funcional de save*
saveas(h,'f.ext')	*Salva la figura o modelo h como el fichero f.ext*
saveas(h,'f','formato')	*Salva la figura o modelo h como el fichero f con el formato especificado*
d = size(X)	*Devuelve en un vector los tamaños de cada dimensión*
[m,n] = size(X)	*Devuelve las dimensiones de la matriz X como dos variables de nombres m y n*
[d1,d2,d3,...,dn] = size(X)	*Devuelve las dimensiones del array X como variables de nombres d1, d2, ..., dn*

who	*Lista las variables del workspace*
whos	*Lista las variables del workspace con sus tamaños y tipos*
who('global')	*Lista las variables en el workspace global*
whos('global')	*Lista las variables en el workspace global con sus tamaños y tipos*
who('-file','fichero')	*Lista las variables en el fichero .mat especificado*
whos('-file','fichero')	*Lista las variables en el fichero .mat especificado y sus tamaños y tipos*
who('var1','var2',...)	*Lista las variables cadena del workspace especificadas*
who('-file','filename', 'var1','var2',...)	*Lista las variables cadena especificadas en el fichero .mat dado*
s = who(...)	*Almacena en s las variables listadas*
s = whos(...)	*Almacena en s las variables con sus tamaños y tipos*
who -file filename var1 var2 ...	*Lista las variables numéricas especificadas en el fichero .mat dado*
whos -file filename var1 var2 ...	*Lista las variables numéricas especificadas en el fichero .mat dado con sus tamaños y tipos*
workspace	*Muestra un navegador para gestionar el workspace*

El comando *save*, que guarda en fichero variables del espacio de trabajo, admite las siguientes opciones:

Opción	***Modo de almacenaminto de los datos***
-append	*Las variables se añaden al final del fichero*
-ascii	*Las variables se almacenan en un fichero en formato ASCII de 8 dígitos*
-ascii -double	*Las variables se almacenan en un fichero en formato ASCII de 16 dígitos*
-ascii -tabs	*Las variables se almacenan en un fichero en formato ASCII de 8 dígitos delimitado por tabuladores*
-ascii -double -tabs	*Las variables se almacenan en un fichero en formato ASCII de 16 dígitos delimitado por tabuladores*
-mat	*Las variables se almacenan en un fichero .mat binario con formato MATLAB (MAT-file)*
-v4	*Las variables se almacenan en un fichero MATLAB version 4*

El comando *save* es el instrumento esencial para guardar datos en ficheros *.mat* tipo MATLAB (sólo legibles por el programa MATLAB) y en ficheros tipo ASCII (legibles por cualquier aplicación). Por defecto, el almacenamiento de las variables suele realizarse en ficheros con formato MATLAB *.mat*. Para almacenar variables en ficheros con formato ASCII es necesario utilizar opciones.

Como primer ejemplo consideramos una variable *A* equivalente a la inversa de una matriz cuadrada aleatoria de orden *5* y una variable *B* equivalente a la inversa de 2 veces la matriz de unos de orden 5 menos la matriz identidad de orden 5.

```
>> A=inv(rand(3))

A =
          1.67         -0.12         -0.93
         -0.42          1.17          0.20
         -0.85         -1.00          1.71

>> B=inv(2*ones(3)-eye(3))

B =
         -0.60          0.40          0.40
          0.40         -0.60          0.40
          0.40          0.40         -0.60
```

Ahora utilizamos los comandos *who* y *whos* para ver las variables del espacio de trabajo, respectivamente, como simple listado y con su tipo y tamaño.

```
>> who

Your variables are:

A  B

>> whos

  Name        Size              Bytes  Class

  A           3x3                  72  double array
  B           3x3                  72  double array

Grand total is 18 elements using 144 bytes
```

Si sólo queremos información de la variable *A* hacemos lo siguiente:

```
>> who A

Your variables are:

A

>> whos A

  Name        Size              Bytes  Class

  A           3x3                  72  double array

Grand total is 9 elements using 72 bytes
```

Ahora vamos a guardar las variables *A* y *B* en un fichero ASCII con 8 dígitos de precisión y de nombre *matriz.asc*. Además, para comprobar que el fichero ASCII se ha generado, usamos la orden *dir* y vemos que nuestro fichero existe. Por último, comprobaremos el contenido de nuestro fichero, usando la orden *type* del sistema operativo DOS para ver que efectivamente el contenido son los elementos de las dos matrices con 8 dígitos de precisión, situadas una a continuación de la otra

```
>> save matriz.asc A B -ascii
>> dir

.            ..           matriz.asc

>> type matriz.asc

  1.6740445e+000 -1.1964440e-001 -9.2759516e-001
 -4.1647244e-001  1.1737582e+000  2.0499870e-001
 -8.5035677e-001 -1.0006147e+000  1.7125190e+000
 -6.0000000e-001  4.0000000e-001  4.0000000e-001
  4.0000000e-001 -6.0000000e-001  4.0000000e-001
  4.0000000e-001  4.0000000e-001 -6.0000000e-001
```

Los ficheros generados con el comando *save* se almacenan por defecto (si no se especifica otra cosa) en el subdirectorio \MATLAB\BIN\ .

Guardar todas las variables del *workspace* con el comando *save* a un fichero binario con formato MATLAB es equivalente a utilizar la subopción *Save Workspace As.* de la opción *Archivo* del menú general de MATLAB.

Una vez guardadas en fichero las variables podemos borrar el espacio de trabajo mediante el comando *clear*.

```
>> clear
```

A continuación, para ilustrar el comando *load*, vamos a leer el fichero ASCII *matriz..asc* guardado anteriormente. MATLAB leerá todo el fichero ASCII como una única variable cuyo nombre es el del fichero, como se comprueba con el comando *whos*.

```
>> load matriz.asc
>> whos

  Name         Size          Bytes  Class

  matriz       6x3             144  double array

Grand total is 18 elements using 144 bytes
```

Ahora comprobamos que MATLAB ha leído los datos con la misma estructura matricial 6x3 con que los había guardado. Las cuatro primeras filas correspondían a la variable *A* y las cuatro últimas a la variable *B*.

```
>> matriz

matriz =

          1.67         -0.12         -0.93
         -0.42          1.17          0.20
         -0.85         -1.00          1.71
         -0.60          0.40          0.40
          0.40         -0.60          0.40
          0.40          0.40         -0.60
```

Ahora podemos utilizar los comandos de manejo de variables matriciales para volver a definir las variables *A* y *B* a partir de la variable *matriz*:

```
>> A=matriz(1:3,1:3)

A =

          1.67         -0.12         -0.93
         -0.42          1.17          0.20
         -0.85         -1.00          1.71

>> B=matriz(4:6,1:3)

B =

         -0.60          0.40          0.40
          0.40         -0.60          0.40
          0.40          0.40         -0.60
```

Comandos que trabajan con ficheros y el entorno operativo

En MATLAB existe un grupo de comandos que permiten trabajar con ficheros, permitiendo el análisis de su información, copiarlos, borrarlos, editarlos, guardarlos y otras operaciones. Asimismo, estos comandos también permiten interrelacionar el entorno MATLAB con el entorno DOS, para dar cabida al trabajo con comandos del sistema operativo DOS dentro de la ventana de comandos del programa MATLAB.

A continuación se presenta una relación de este tipo de comandos.

beep	*Produce un pitido*
cd directorio	*Cambia desde el directorio actual al directorio de trabajo dado*
copyfile f1 f2	*Copia el fichero (o directorio) origen f1 al fichero destino f2*
delete fichero	*Borra el fichero (u objeto gráfico) especificado*
diary('fichero')	*Escribe las entradas y salidas de la sesión actual en el fichero*
dir	*Muestra los ficheros del directorio actual*
dos comando	*Ejecuta un comando de DOS y devuelve el resultado*
edit M-fichero	*Edita un M-fichero*
[path,name,ext,ver] = fileparts('fichero')	*Devuelve el camino, el nombre, la extensión y la versión del fichero especificado*
filebrowser	*Muestra los ficheros del directorio actual en un navegador*
fullfile('d1', 'd2', ..., 'f')	*Forma un fichero completo con los directorios y fichero dados*
info toolbox	*Muestra información sobre el toolbox especificado*
[M,X,J] =inmem	*Devuelve M-ficheros, MEX-ficheros y clases Java en memoria*
ls	*Lista el directorio actual en UNIX*
MATLABroot	*Devuelve el nombre del directorio en que está instalado MATLAB*
mkdir directorio	*Construye un nuevo directorio*
open('fichero')	*Abre el fichero dado*
pwd	*Muestra el directorio actual*
tempdir	*Devuelve el nombre del directorio temporal del sistema*
nombre=tempname	*Asigna nombre único al directorio temporal*
unix comando	*Ejecuta un comando UNIX y devuelve el resultado*
! comando	*Ejecuta un comando del sistema operativo*

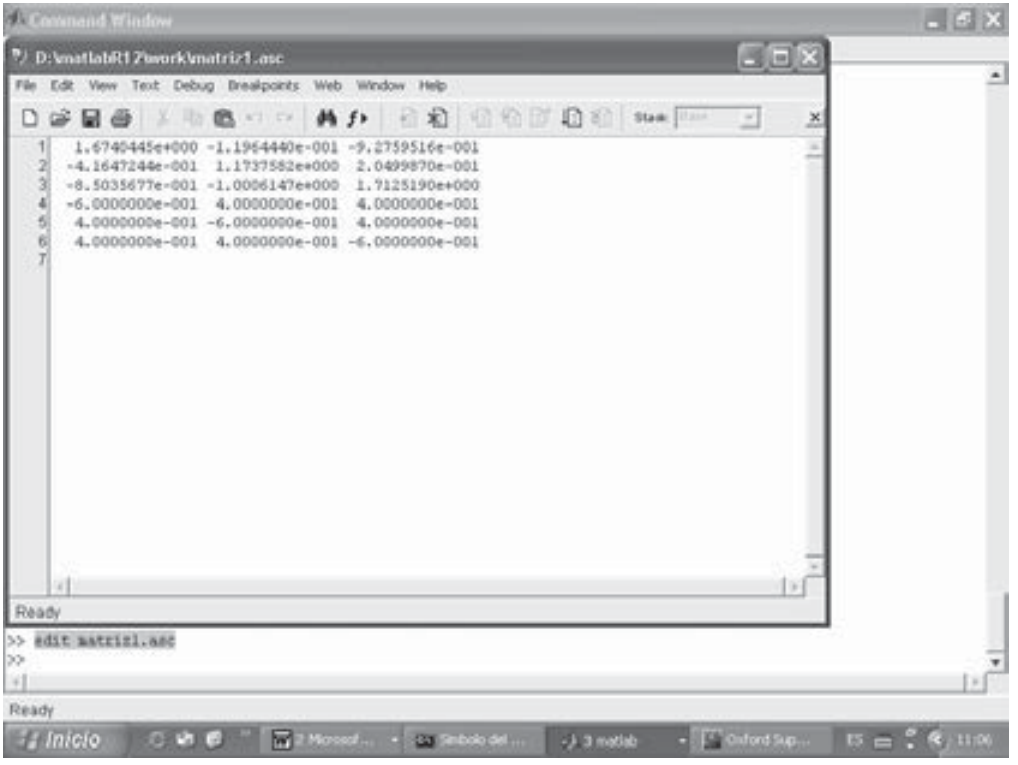
Veamos algunos ejemplos:

```
>> dir

.            ..           matriz.asc

>> !dir

 El volumen de la unidad D no tiene etiqueta.
 El n£mero de serie del volumen es: 1179-07DC

 Directorio de D:\MATLABR12\work

01/01/2001  07:01    <DIR>          .
01/01/2001  07:01    <DIR>          ..
02/01/2001  03:27               300 matriz.asc
               1 archivos            300 bytes
               2 dirs   1.338.146.816 bytes libres

>> !type matriz.asc
```

```
  1.6740445e+000 -1.1964440e-001 -9.2759516e-001
 -4.1647244e-001  1.1737582e+000  2.0499870e-001
 -8.5035677e-001 -1.0006147e+000  1.7125190e+000
 -6.0000000e-001  4.0000000e-001  4.0000000e-001
  4.0000000e-001 -6.0000000e-001  4.0000000e-001
  4.0000000e-001  4.0000000e-001 -6.0000000e-001
```

```
>> tempdir
ans =
C:\DOCUME~1\CPL\CONFIG~1\Temp\

>> MATLABroot
ans =
D:\MATLABR12

>> pwd
ans =
D:\MATLABR12\work

>> cd ..
>> pwd
ans =

D:\MATLABR12

>> cd work
>> pwd
ans =
D:\MATLABR12\work

>> copyfile matriz.asc matriz1.asc
>> dir
.            ..           matriz.asc   matriz1.asc
>> dos dir

 El volumen de la unidad D no tiene etiqueta.
 El n£mero de serie del volumen es: 1179-07DC

 Directorio de D:\MATLABR12\work

01/01/2001  07:01    <DIR>          .
01/01/2001  07:01    <DIR>          ..
02/01/2001  03:27               300 matriz.asc
02/01/2001  03:27               300 matriz1.asc
               2 archivos            600 bytes
               2 dirs   1.338.130.432 bytes libres
```

Un comando importante que permite la edición directa sobre una ventana de cualquier M-fichero es *edit*. La figura siguiente presenta la ventana de edición para el fichero *matriz1.asc*.



Comandos que manejan funciones

Existe en MATLAB un grupo de comandos que manejan funciones mostrando ayuda sobre las mismas, ofreciendo acceso a su información y generando informes. A continuación se presenta un listado de este tipo de comandos.

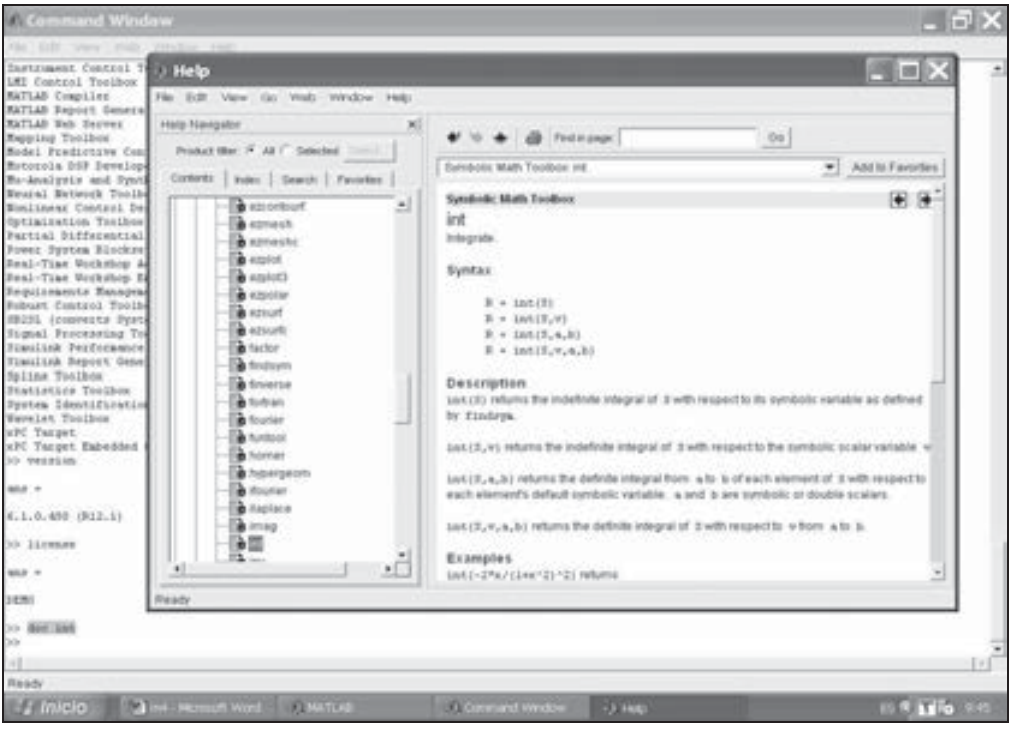
addpath('dir', 'dir2', ...)	*Añade los directorios al path de búsqueda de MATLAB*
doc **doc fichero** **doc toolbox/** **doc toolbox/función**	*Muestra documentación HTML en el panel de ayuda para las funciones MATLAB en la ventana de comandos, para M-ficheros especificados, para el contenido de un toolbox o para las funciones especificadas de un toolbox.*
help **help fichero** **help toolbox/** **help toolbox/función**	*Muestra ayuda para las funciones MATLAB en la ventana de comandos común, para M-ficheros especificados, para el contenido de un toolbox o para las funciones especificadas de un toolbox.*

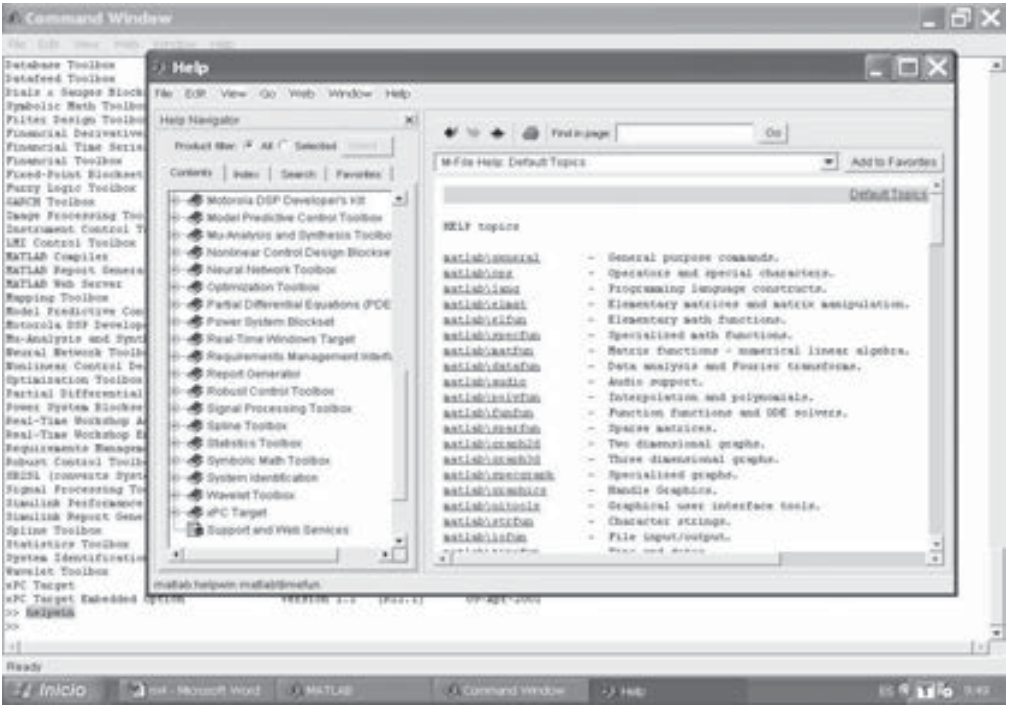
helpbrowser	*Muestra el navegador de toda la ayuda de MATLAB*
helpdesk	*Muestra el navegador de ayuda situado en la página de inicio*
helpwin	*Muestra ayuda para todas las funciones de MATLAB*
docopt	*Muestra la localización del fichero de ayuda en UNIX*
genpath	*Genera una cadena de path*
lasterr	*Devuelve el último mensaje de error*
lastwarn	*Devuelve el último mensaje de emergencia*
license	*Muestra el número de licencia de MATLAB*
lookfor tema	*Búsqueda de todas las funciones relacionadas con el tema*
partialpath	*Sitúa el path parcialmente en el directorio actual*
path	*Muestra el path completo de MATLAB*
pathtool	*Muestra el path completo de MATLAB en modo ventana*
profile	*Inicia la utilidad profiler, para depurar y optimizar código de M-ficheros*
profreport	*Genera un informe de perfil en formato HTML y suspende la utilidad de ventanas profiler*
rehash	*Refresca funciones y ficheros de caches de sistema*
rmpath directorio	*Remueve el directorio del path de MATLAB*
support	*Abre la página Web de MathWorks*
type fichero	*Lista el contenido del fichero*
ver (o ver toolbox)	*Muestra la version de MATLAB, Simulink y toolboxes*
version	*Da el número de versión de MATLAB*
web url	*Sitúa el navegador en la dirección Web indicada*
what	*Lista los ficheros específicos de MATLAB (.m, .mat, .mex, .mdl y .p) en el directorio actual*
whatsnew	*Muestra ficheros de ayuda con novedades de MATLAB y sus toolboxes*
which función **which fichero**	*Localiza funciones* *Localiza ficheros*

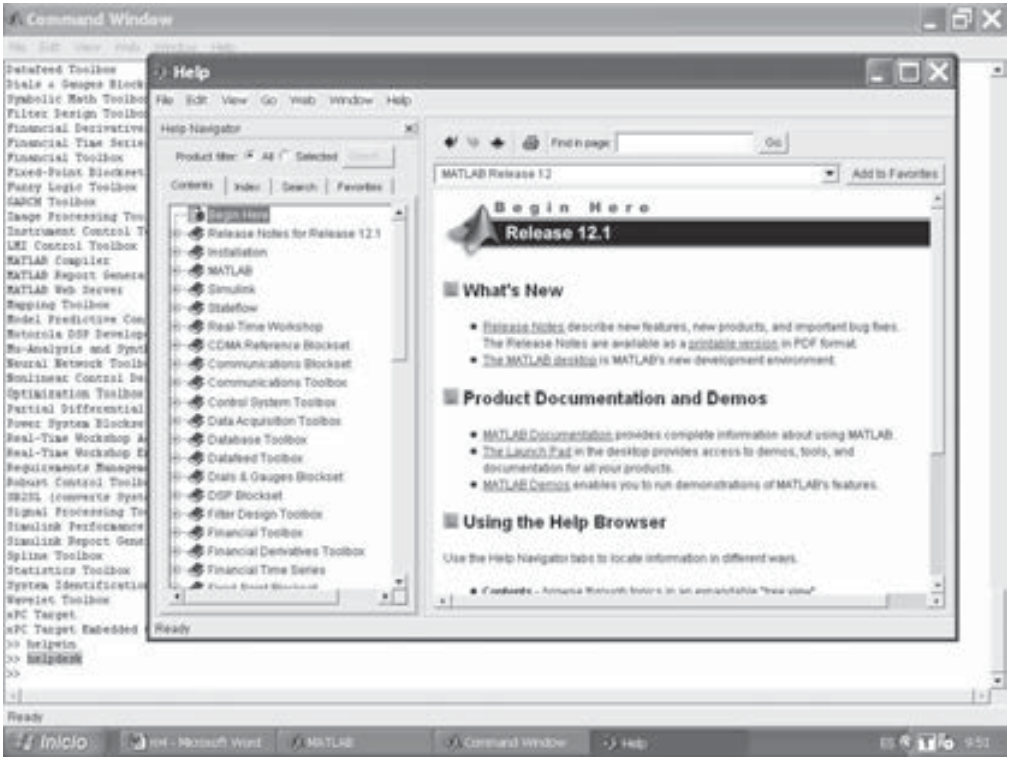
Veamos algunos ejemplos:

```
>> version

ans =

6.1.0.450 (R12.1)

>> license

ans =

DEMO
```

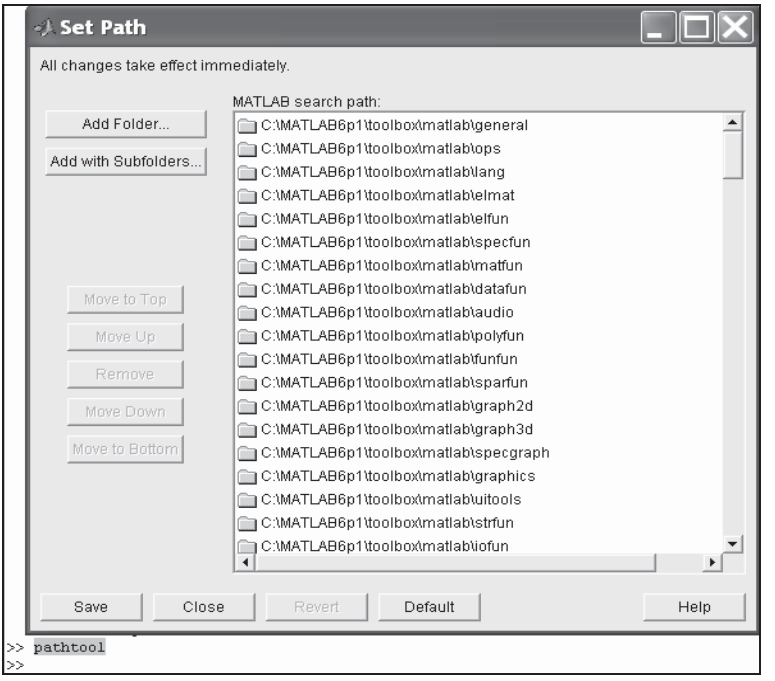
```
>> help toolbox\symbolic

  Symbolic Math Toolbox.
  Version 2.1.2    (R12.1)  11-Sep-2000

  New Features.
  Readme      - Overview of the new features in/changes made to
                the Symbolic and Extended Symbolic Math Toolboxes.

  Calculus.
    diff         - Differentiate.
    int          - Integrate.
    limit        - Limit.
    taylor       - Taylor series.
    jacobian     - Jacobian matrix.
    symsum       - Summation of series.

  Linear Algebra.
    diag         - Create or extract diagonals.
    triu         - Upper triangle.
    tril         - Lower triangle.
    inv          - Matrix inverse.
    det          - Determinant.
    rank         - Rank.
    rref         - Reduced row echelon form.
    null         - Basis for null space.
    colspace     - Basis for column space.
    eig          - Eigenvalues and eigenvectors.
    svd          - Singular values and singular vectors.
    jordan       - Jordan canonical (normal) form.
    poly         - Characteristic polynomial.
    expm         - Matrix exponential.

>> help int

 --- help for sym/int.m ---

 INT    Integrate.
    INT(S) is the indefinite integral of S with respect to its symbolic
      variable as defined by FINDSYM. S is a SYM (matrix or scalar).
      If S is a constant, the integral is with respect to 'x'.
    INT(S,v) is the indefinite integral of S with respect to v. v is a
      scalar SYM.
    INT(S,a,b) is the definite integral of S with respect to its
      symbolic variable from a to b. a and b are each double or
      symbolic scalars.
    INT(S,v,a,b) is the definite integral of S with respect to v
      from a to b.

    Examples:
      syms x alpha u t;
      int(1/(1+x^2))           returns     atan(x)
      int(sin(alpha*u),alpha)  returns     -cos(alpha*u)/u
      int(4*x*t,x,2,sin(t))    returns      2*sin(t)^2*t-8*t
```

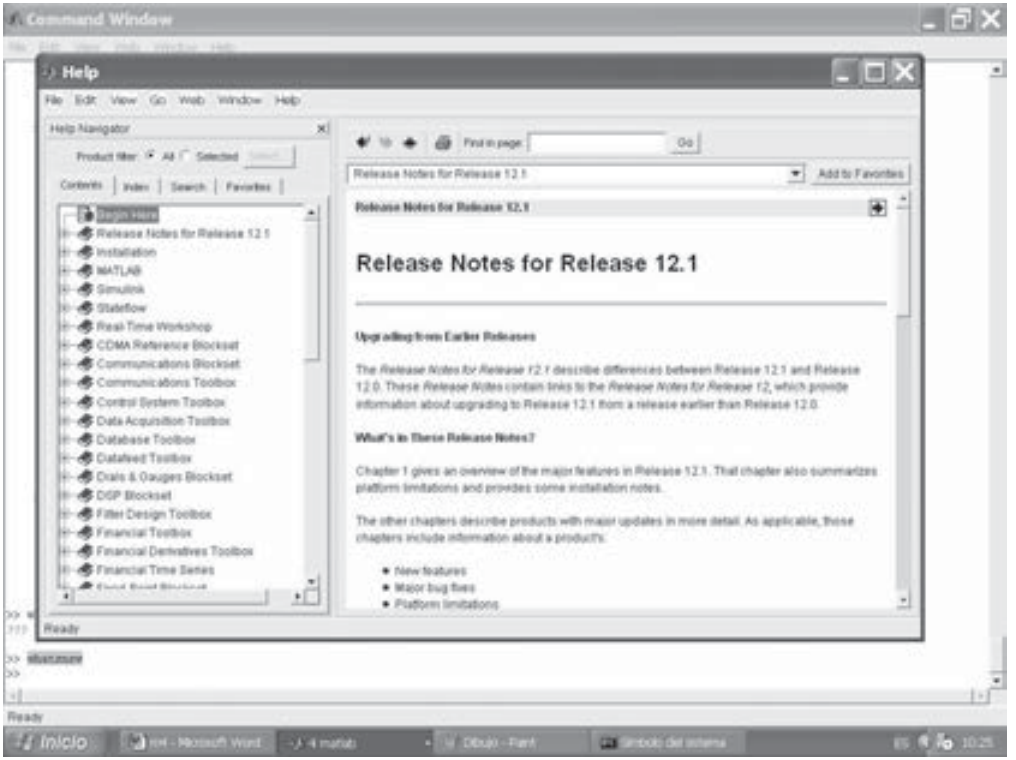

Command Window
Help
Help Navigator
Contents
Index
Search
Favorites
Symbolic Math Toolbox int
Add to Favorites
Symbolic Math Toolbox
int
Integrate.
Syntax
R = int(S)
R = int(S,v)
R = int(S,a,b)
R = int(S,v,a,b)
Description
Examples
Ready


Command Window
Help
Help Navigator
Contents
Index
Search
Favorites
M-File Help: Default Topics
Add to Favorites
Default Topics
HELP topics
matlab\general - General purpose commands.
matlab\ops - Operators and special characters.
matlab\lang - Programming language constructs.
matlab\elmat - Elementary matrices and matrix manipulation.
matlab\elfun - Elementary math functions.
matlab\specfun - Specialized math functions.
matlab\matfun - Matrix functions - numerical linear algebra.
matlab\datafun - Data analysis and Fourier transforms.
matlab\audio - Audio support.
matlab\polyfun - Interpolation and polynomials.
matlab\funfun - Function functions and ODE solvers.
matlab\sparfun - Sparse matrices.
matlab\graph2d - Two dimensional graphs.
matlab\graph3d - Three dimensional graphs.
matlab\specgraph - Specialized graphs.
matlab\graphics - Handle Graphics.
matlab\uitools - Graphical user interface tools.
matlab\strfun - Character strings.
matlab\iofun - File input/output.
Ready




```
>> lookfor GALOIS

GFADD Add polynomials over a Galois field.
GFCONV Multiply polynomials over a Galois field.
GFCOSETS Produce cyclotomic cosets for a Galois field.
GFDECONV Divide polynomials over a Galois field.
GFDIV Divide elements of a Galois field.
GFFILTER Filter data using polynomials over a prime Galois field.
GFLINEQ Find a particular solution of Ax = b over a prime Galois
field.
GFMINPOL Find the minimal polynomial of an element of a Galois field.
GFMUL Multiply elements of a Galois field.
GFPLUS Add elements of a Galois field of characteristic two.
GFPRIMCK Check whether a polynomial over a Galois field is primitive.
GFPRIMDF Provide default primitive polynomials for a Galois field.
GFPRIMFD Find primitive polynomials for a Galois field.
GFRANK Compute the rank of a matrix over a Galois field.
GFROOTS Find roots of a polynomial over a prime Galois field.
GFSUB Subtract polynomials over a Galois field.
GFTUPLE Simplify or convert the format of elements of a Galois field.
```


Set Path
All changes take effect immediately.
MATLAB search path:
Add Folder...
Add with Subfolders...
Move to Top
Move Up
Remove
Move Down
Move to Bottom
C:\MATLAB6p1\toolbox\matlab\general
C:\MATLAB6p1\toolbox\matlab\ops
C:\MATLAB6p1\toolbox\matlab\lang
C:\MATLAB6p1\toolbox\matlab\elmat
C:\MATLAB6p1\toolbox\matlab\elfun
C:\MATLAB6p1\toolbox\matlab\specfun
C:\MATLAB6p1\toolbox\matlab\matfun
C:\MATLAB6p1\toolbox\matlab\datafun
C:\MATLAB6p1\toolbox\matlab\audio
C:\MATLAB6p1\toolbox\matlab\polyfun
C:\MATLAB6p1\toolbox\matlab\funfun
C:\MATLAB6p1\toolbox\matlab\sparfun
C:\MATLAB6p1\toolbox\matlab\graph2d
C:\MATLAB6p1\toolbox\matlab\graph3d
C:\MATLAB6p1\toolbox\matlab\specgraph
C:\MATLAB6p1\toolbox\matlab\graphics
C:\MATLAB6p1\toolbox\matlab\uitools
C:\MATLAB6p1\toolbox\matlab\strfun
C:\MATLAB6p1\toolbox\matlab\iofun
Save
Close
Revert
Default
Help
>> pathtool
>>


Command Window
Help
Release Notes for Release 12.1
Ready

```
>> what

M-files in the current directory C:\MATLAB6p1\work

cosint

>> which sinint

C:\MATLAB6p1\toolbox\symbolic\sinint.m
```

Comandos que controlan la ventana Command Window

Existe un grupo de comandos en MATLAB que controlan la salida de la información en la ventana de comandos. Se presentan en la siguiente tabla:

clc	*Borra la ventana de comandos*
echo	*Permite ver (echo on) u ocultar (echo off) las líneas de código de un M-fichero durante su ejecución*
format tipo	*Controla el formato de la salida en la ventana de comandos*
home	*Mueve el cursor a la esquina superior izquierda de la ventana de comandos*
more	*Permite ver la salida en la ventana de comandos página a página*

Los tipos posibles para el comando *format* se presentan a continuación:

Tipo	***Resultado***	***Ejemplo***
+	*+, -, blanco*	`+`
bank	*Fijo para dólares y céntimos*	`3.14`
compact	*Suprime el exceso de líneas en la salida. Contrasta con* `loose`.	`theta = pi/2`
hex	*Formato hexadecimal*	`400921fb54442d18`
long	*Punto fijo de 15 dígitos*	`3.14159265358979`
long e	*Punto flotante de 15 dígitos*	`3.141592653589793e+00`
long g	*El mejor de 15 dígitos (punto fijo o flotante)*	`3.14159265358979`
loose	*Añade líneas a la salida para que sea más legible. Contrasta con* `compact`.	`theta=` `1.5708`
rat	*Formato racional*	`355/113`
short	*Punto fijo de 5 dígitos*	`3.1416`
short e	*Punto flotante de 5 dígitos*	`3.1416e+00`
short g	*El mejor de 5 dígitos (punto fijo o flotante)*	`3.1416`

Comandos de comienzo y salida de MATLAB

Los comandos de comienzo y salida de MATLAB son los siguientes:

finish	*Finalización de M-fichero*
exit	*Finalización de MATLAB*
MATLAB	*Comienzo de MATLAB (solo en UNIX)*
MATLABrc	*Comienzo de M-fichero*
quit	*Finalización de MATLAB*
startup	*Comienzo de M-fichero*

4.2 Comandos de entrada/salida de ficheros

MATLAB dispone de un grupo de comandos denominados de entrada/salida que actúan sobre ficheros y que permiten abrir y cerrar ficheros, leer y escribir en ficheros, controlar la posición en un fichero y exportar e importar datos. La tabla siguiente resume este tipo de comandos de MATLAB. Su sintaxis completa se ofrecerá en los párrafos siguientes.

Abriendo y cerrando ficheros	
fclose	*Cierra uno o más ficheros*
fopen	*Abre un fichero u obtiene información acerca de ficheros abiertos*
Entrada/salida sin formato	
fread	*Lee datos binarios de un fichero*
fwrite	*Escribe datos binarios en un fichero*
Entrada/salida con formato	
fgetl	*Devuelve la siguiente línea de un fichero como una cadena sin fin de líneas*
fgets	*Devuelve la siguiente línea de un fichero como una cadena con fin de líneas*
fprintf	*Escribe datos formateados en un fichero*
fscanf	*Lee datos formateados de un fichero*
Controlando la posición en un fichero	
feof	*Testea el final de fichero*
ferror	*Pregunta a MATLAB por los errors en la entrada o salida de fichero*
frewind	*Relee un fichero abierto*
fseek	*Sitúa el indicador de posición en un fichero*
ftell	*Obtiene la situación del indicador de posición en un fichero*
Conversión de cadenas	
sprintf	*Escribe datos formateados como una cadena*
sscanf	*Lee cadenas bajo control de formato*

Funciones de entrada/salida especializadas	
dlmread	*Lee ficheros con formato ASCII delimitado*
dlmwrite	*Escribe ficheros con formato ASCII delimitado*
hdf	*Interfaz HDF*
imfinfo	*Devuelve información acerca de ficheros gráficos*
imread	*Lee imágenes de ficheros gráficos*
imwrite	*Escribe una imagen en un fichero gráfico*
strread	*Lee datos formateados de una cadena*
textread	*Lee datos formateados de un fichero texto*
wk1read	*Lee datos de ficheros WK1 de hoja de cálculo Lotus123*
wk1write	*Escribe datos en ficheros WK1 de hoja de cálculo Lotus123*

Abriendo y cerrando ficheros

El proceso a seguir para leer o escribir datos en un fichero cualquiera (que no tiene por qué ser de formatos ASCII o MATLAB), comienza por utilizar el comando *fopen* para abrirlo. Después, con el fin de realizar las operaciones de lectura o escritura en él, se usarán los comandos correspondientes de lectura y escritura (*fload*, *fwrite*, *fscanf*, *fprintf*, etc.). Por último, se utilizará el comando *fclose* para cerrar el fichero. El fichero que se abre puede ser nuevo o puede existir previamente (con la finalidad de ampliar su contenido o leerlo).

El comando *fopen* devuelve un identificador de fichero que consiste en un entero no negativo asignado por el sistema operativo al fichero que se abre. El indicador es realmente una referencia para el manejo del fichero abierto que posteriormente será utilizada para leerlo (comando *read*), escribir en él (comando *write*) o cerrarlo (comando *close*). Si el fichero no se abre correctamente, *fopen* devuelve -1 como identificador de fichero. Como identificador de fichero genérico suele utilizarse *fid*. La sintaxis de los comandos *fopen* y *fclose* admite las modalidades que se describen a continuación.

fid = fopen('fichero')	*Abre el fichero especificado ya existente*
fid = fopen('fichero', 'permiso')	*Abre el fichero para el tipo de permiso dado*
[fid, message] = fopen('fichero', 'permiso', arquitectura')	*Abre el fichero para el permiso dado y con el formato numérico de la arquitectura dado*
fids = fopen('all')	*Devuelve un vector columna con los identificadores de todos los ficheros abiertos*
[filename, permission, arquitectura] = fopen(fid)	*Devuelve el nombre del fichero, el tipo de permiso y el formato numérico de la arquitectura especificada referentes al fichero cuyo identificador es fid*
fclose(fid)	*Cierra el fichero de identificador fid si está abierto; devuelve 1 si el proceso se ha realizado con éxito y 0 en caso contrario*
fclose('all')	*Cierra todos los ficheros abiertos; devuelve 1 si el proceso es correcto y 0 si no lo es*

Los tipos posibles de permisos son los siguientes:

'r'	*Abre el fichero ya existente para lectura (es el permiso por defecto)*
'r+'	*Abre el fichero ya existente para lectura y escritura*
'w'	*Crea el fichero nuevo y lo abre para escritura, y si ya existe un fichero con ese nombre, lo borra y lo abre de nuevo vacío*
'w+'	*Crea el fichero nuevo para lectura y escritura, y si ya existe un fichero con ese nombre, lo borra y lo abre de nuevo vacío*
'a'	*Crea el fichero nuevo y lo abre para escritura, y si ya existe un fichero con ese nombre, añade el contenido nuevo al final del ya existente*
'a+'	*Crea el fichero nuevo y lo abre para lectura y escritura, y si ya existe un fichero con ese nombre, añade el contenido nuevo al final de la información existente*
'A'	*Abre el fichero en drivers de cinta*
'W'	*Escribe en el fichero en drivers de cinta*

Los tipos posibles de arquitecturas para el formato numérico son los siguientes:

'native' o 'n'	*Formato numérico de la máquina actual*
'ieee-le' o 'l'	*Formato pequeño IEEE de punto flotante*
'ieee-be' o 'b'	*Formato grande IEEE de punto flotante*
'vaxd' o 'd'	*Formato de punto flotante VAX D*
'vaxg' o 'g'	*Formato de punto flotante VAX G*
'cray' o 'c'	*Formato grande de punto flotante tipo Cray*
'ieee-le.l64' o 'a'	*Formato pequeño IEEE de punto flotante y 64 bits de longitud de datos*
'ieee-be.l64' o 's'	*Formato grande IEEE de punto flotante y 64 bits de longitud de datos*

El hecho de poder abrir un fichero según el formato numérico de una determinada arquitectura permite su uso en distintas plataformas MATLAB.

Leyendo y escribiendo ficheros binarios

La lectura y escritura en ficheros binarios se realiza a través de los comandos *fwrite* y *fread*. El comando *fwrite* se utiliza para escribir datos binarios en un fichero previamente abierto con el comando *fopen*. El comando *fread* se utiliza para leer datos de un fichero binario previamente abierto con el comando *fopen.* Su sintaxis es la siguiente:

fwrite(fid,A, precision)	*Escribe los elementos especificados en A (que en general es una matriz) en el fichero de identificador fid (previamente abierto) con la precisión especificada*
A = fread(fid)	*Lee los datos del fichero binario abierto con identificador fid y los escribe en la matriz A, que por defecto será un vector columna*
[A, count] = fread(fid,size, 'precision')	*Lee los datos del fichero de identificador fid con la dimensión especificada en size y la precisión dada por precision, y los escribe en la matriz A de tamaño size y cuyo número total de elementos es count*

El tamaño *size* es opcional. Si *size* toma el valor *n*, *fread* lee los *n* primeros datos del fichero (por columnas y en orden) como un vector columna, *A*, de longitud *n*. Si *size* toma el valor *inf*, *fread* lee todos los datos del fichero por columnas y en orden, para formar un único vector columna *A* (se trata del valor por defecto). Si *size* toma el valor *[m,n]*, *fread* lee *mxn* elementos del fichero por columnas y en orden, hasta completar la matriz *A* de dimensión *mxn*. Si no hay suficientes elementos en el fichero se completa la matriz con los ceros que sean necesarios.

El argumento *precision*, relativo a la precisión numérica de la máquina con la que trabajamos, puede presentar diferentes valores. Por similitud con los lenguajes de programación C y Fortran, MATLAB acepta los tipos de formato para la precisión numérica de ambos lenguajes, además de los suyos propios. A continuación se presenta una tabla con los posibles valores de *precision*.

MATLAB	***C o Fortran***	***Interpretación***
'schar'	**'signed char'**	*Carácter con signo; 8 bits*
'uchar'	**'unsigned char'**	*Carácter sin signo; 8 bits*
'int8'	**'integer*1'**	*Entero; 8 bits*
'int16'	**'integer*2'**	*Entero; 16 bits*
'int32'	**'integer*4'**	*Entero; 32 bits*
'int64'	**'integer*8'**	*Entero; 64 bits*
'uint8'	**'integer*1'**	*Entero sin signo; 8 bits*
'uint16'	**'integer*2'**	*Entero sin signo; 16 bits*
'uint32'	**'integer*4'**	*Entero sin signo; 32 bits*
'uint64'	**'integer*8'**	*Entero sin signo; 64 bits*
'float32'	**'real*4'**	*Punto flotante; 32 bits*
'float64'	**'real*8'**	*Punto flotante; 64 bits*
'double'	**'real*8'**	*Punto flotante; 64 bits*

Los formatos de la siguiente tabla también son soportados por MATLAB, pero no hay garantía de que se utilice el mismo tamaño en todas las plataformas.

MATLAB	***C or Fortran***	***Interpretación***
'char'	**'char*1'**	*Carácter; 8 bits*
'short'	**'short'**	*Entero; 16 bits*
'int'	**'int'**	*Entero; 32 bits*
'long'	**'long'**	*Entero; 32 o 64 bits*
'ushort'	**'unsigned short'**	*Entero sin signo; 16 bits*
'uint'	**'unsigned int'**	*Entero sin signo; 32 bits*
'ulong'	**'unsigned long'**	*Entero sin signo; 32 o 64 bits*
'float'	**'float'**	*Punto flotante; 32 bits*
'intN'		*Entero con signo de ancho N bits (1≤N≤64)*
'ubitN'		*Entero sin signo de ancho N bits (1≤N≤64)*

Cuando se leen y se guardan formatos suele utilizarse el símbolo de implicación tal y como se ilustra en los ejemplos siguientes:

'uint8=>uint8'	*Lee enteros sin signo de 8-bits y los guarda en un array de enteros sin signo de 8 bits*
'*uint8'	*Versión abreviada del ejemplo anterior*
'bit4=>int8'	*Lee enteros con signo de 4 bits 4-bits empaquetados en bytes y los guarda en un array de enteros de 8 bits. Cada entero de 4 bits se convierte en un entero de 8 bits*
'double=>real*4'	*Lee en doble precisión de punto flotante y guarda en un array de reales de 32-bits en punto flotante.*

Como primer ejemplo podemos ver el contenido del fichero *fclose.m* mediante el comando *type* como sigue:

```
>> type fclose.m

%FCLOSE Close file.
%   ST = FCLOSE(FID) closes the file with file identifier FID,
%   which is an integer obtained from an earlier FOPEN.  FCLOSE
%   returns 0 if successful and -1 if not.
%
%   ST = FCLOSE('all') closes all open files, except 0, 1 and 2.
%
%   See also FOPEN, FREWIND, FREAD, FWRITE.
%   Copyright 1984-2001 The MathWorks, Inc.
%   $Revision: 5.8 $  $Date: 2001/04/15 12:02:12 $
%   Built-in function.
```

Sería equivalente al comando *type* anterior la apertura del fichero con *fopen*, seguida de la lectura de su contenido con *fread* y de la presentación del mismo con la función *char*.

```
>> fid = fopen('fclose.m','r');
>> F = fread(fid);
>> s = char(F')

s =

%FCLOSE Close file.
%   ST = FCLOSE(FID) closes the file with file identifier FID,
%   which is an integer obtained from an earlier FOPEN.  FCLOSE
%   returns 0 if successful and -1 if not.
%
%   ST = FCLOSE('all') closes all open files, except 0, 1 and 2.
%
%   See also FOPEN, FREWIND, FREAD, FWRITE.
%   Copyright 1984-2001 The MathWorks, Inc.
%   $Revision: 5.8 $  $Date: 2001/04/15 12:02:12 $
%   Built-in function.
```

En el ejemplo siguiente creamos un fichero binario de nombre *id4.bin* que contiene los 16 elementos de la matriz identidad de orden 4 almacenados en enteros de 4 bytes (64 bytes en total). Primeramente abriremos el fichero que ha de contener a la matriz con permiso para lectura y escritura y a continuación escribimos en él la matriz con el formato adecuado. Por último, cerramos el fichero abierto.

```
>> fid = fopen('id4.bin','w+')

fid =

     5

>> fwrite(fid,eye(4),'integer*4')

ans =

    16

>> fclose(5)

ans =

     0
```

En el ejemplo anterior, al abrir el fichero, el sistema le ha asignado el identificador 5. Después de la escritura de la matriz en el fichero es necesario cerrarlo con el comando *fclose* utilizando su indicador. Como la respuesta es cero, el cierre ha tenido éxito.

Si ahora queremos ver el contenido del fichero binario recién grabado, lo abrimos con permiso de lectura mediante el comando *fopen* y leemos sus elementos con *fread* con la misma estructura matricial y el mismo formato con que fue guardado.

```
>> fid = fopen('id4.bin','r+')
fid =
     5

>> [R,count]=fread(5,[4,4],'integer*4')

R =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

count =
    16
```

Leyendo y escribiendo ficheros ASCII de texto con formato

Es posible escribir texto formateado a medida en un fichero previamente abierto con el comando *fopen* (o en la propia pantalla) mediante el comando *fprintf*. Por otra parte, es posible mediante el comando *fscanf* leer datos con formato de un fichero previamente abierto con el comando *fopen*. La sintaxis es la siguiente:

fprintf(fid, 'formato', A,...)	*Escribe los elementos especificados en A (que en general es una matriz) en el fichero de identificador fid (previamente abierto) con el formato especificado en formato*
fprintf('formato', A,...)	*Realiza la escritura por la pantalla*
[A, count] = fscanf(fid, 'formato')	*Lee los datos con el formato dado del fichero abierto con identificador fid y los escribe en la matriz A, que por defecto será un vector columna*
[A, count] = fscanf(fid, 'formato', size)	*Lee los datos del fichero de identificador fid con la dimensión especificada en size y el formato dado, y los escribe en la matriz A de tamaño size y cuyo número de elementos es count*

El argumento *formato* consiste en una cadena formada por caracteres de escape (precedidos del carácter "\") y por caracteres de conversión según los diferentes formatos (precedidos del carácter "%").

Los posibles caracteres de escape son los siguientes:

\n	*Se ejecuta el paso a nueva línea*
\t	*Se ejecuta un tabulador horizontal*
\b	*Se ejecuta un paso hacia atrás de un solo carácter (backspace), borrando el contenido del carácter actual en caso de que exista*
\r	*Se ejecuta un retorno de carro*
\f	*Se ejecuta un salto de página (form feed)*
****	*Se ejecuta el carácter backslash*
\'	*Se ejecuta el carácter comilla simple*

Los posibles caracteres de conversión son los siguientes:

%d	*Enteros en el sistema decimal*
%o	*Enteros en el sistema octal*
%x	*Enteros en el sistema hexadecimal*
%u	*Enteros sin signo en el sistema decimal*
%f	*Reales de punto fijo*
%e	*Reales de punto flotante*
%g	*Utiliza d, e o f, seleccionando el de mayor precisión en el mínimo espacio*
%c	*Caracteres individuales*

%s	*Cadena de caracteres*
%E	*Reales de punto flotante con la letra E en mayúsculas*
%X	*Notación hexadecimal con mayúsculas*
%G	*Formato %g con letras mayúsculas*

Cuando se trabaja con enteros, se utilizan los caracteres de conversión en la forma *%nv* (*n* es el número de cifras del entero y *v* es el carácter de conversión cuyos valores pueden ser *d*, *o*, *x* o *u*). Por ejemplo, el formato *%7x* indica un entero hexadecimal con 7 cifras.

Cuando se trabaja con reales, se utilizan los caracteres de conversión en la forma *%n.mv* (*n* es el número total de cifras del número real, incluido el punto decimal, *m* es el número de decimales del real y *v* es el carácter de conversión cuyos valores pueden ser *f*, *e* o *g*). Por ejemplo, el formato *%6.2f* indica un número real de punto fijo que tiene 6 cifras en total (incluido el punto) y que presenta 2 cifras decimales.

Cuando se trabaja con cadenas, se utilizan los caracteres de conversión en la forma *%na* (*n* es el número total de caracteres de la cadena y *a* es el carácter de conversión cuyos valores pueden ser *c* o *s*). Por ejemplo, el formato *%8s* indica una cadena con 8 caracteres.

Además, se admiten los caracteres de escape y de conversión del lenguaje C (ver manuales de C al efecto).

En el comando *fscanf* el tamaño *size* es opcional. Si *size* toma el valor *n*, *fscanf* lee los *n* primeros datos del fichero (por columnas y en orden) como un vector columna *A* de longitud *n*. Si *size* toma el valor *inf*, *fread* lee todos los datos del fichero por columnas y en orden, para formar un único vector columna *A* (se trata del valor por defecto). Si *size* toma el valor *[m,n]*, *fread* lee *mxn* elementos del fichero por columnas y en orden, hasta completar la matriz *A* de dimensión *mxn*. Si no hay suficientes elementos en el fichero, se completa la matriz con los ceros que sean necesarios. El argumento *format* toma los mismos valores que para el comando *fprintf*.

Para la lectura en ficheros ASCII existen otros dos comandos, *fgetl* y *fgets*, que presentan como cadena el contenido de las diferentes líneas de un fichero de texto. Su sintaxis es la siguiente:

fgetl(fid)	*Lee los caracteres del fichero de texto con identificador fid, línea a línea e ignorando los retornos de carro, y los devuelve como cadena*
fgets(fid)	*Lee los caracteres del fichero de texto de identificador fid, línea a línea incluyendo los retornos de carro, y los devuelve como cadena*
fgets(fid,nchar)	*Devuelve al menos nchar caracteres de la siguiente línea*

Como ejemplo creamos un fichero ASCII de nombre *exponen.txt,* que contiene los valores de la función exponencial para valores de la variable entre 0 y 1 separados una décima.

El formato del texto en el fichero ha de consistir en dos columnas de números reales de punto flotante, de tal forma que en la primera aparezcan los valores de la variable y en la segunda los correspondientes valores de la función exponencial. Por último, se escriben los comandos de forma que el contenido del fichero se liste en la pantalla.

```
>> x = 0:.1:1;
>> y= [x; exp(x)];
>> fid=fopen('exponen.txt','w');
>> fprintf(fid,'%6.2f  %12.8f\n', y);
>> fclose(fid)

ans =

     0
```

Ahora se presenta la información directamente por pantalla sin necesidad de guardarla en disco:

```
>> x = 0:.1:1;
>> y= [x; exp(x)];
>> fprintf('%6.2f  %12.8f\n', y)

  0.00    1.00000000
  0.10    1.10517092
  0.20    1.22140276
  0.30    1.34985881
  0.40    1.49182470
  0.50    1.64872127
  0.60    1.82211880
  0.70    2.01375271
  0.80    2.22554093
  0.90    2.45960311
  1.00    2.71828183
```

A continuación se lee el fichero ASCII de nombre *exponen.txt* recién generado, de modo que el formato del texto ha de consistir en dos columnas de números reales con la máxima precisión en el mínimo espacio, de tal forma que en la primera aparezcan los valores de la variable y en la segunda los correspondientes valores de la función exponencial.

```
>> fid=fopen('exponen.txt');
>> a = fscanf(fid,'%g  %g', [2 inf]);
>> a = a'
```

```
a =

         0    1.0000
    0.1000    1.1052
    0.2000    1.2214
    0.3000    1.3499
    0.4000    1.4918
    0.5000    1.6487
    0.6000    1.8221
    0.7000    2.0138
    0.8000    2.2255
    0.9000    2.4596
    1.0000    2.7183
```

Seguidamente abrimos el fichero *exponent.txt* y leemos su contenido línea a línea con el comando *fgetl*.

```
>> fid=fopen('exponen.txt');
>> linea1=fgetl(fid)

linea1 =

  0.00    1.00000000

>> linea2=fgetl(fid)

linea2 =

  0.10    1.10517092
```

A continuación se utiliza el comando *sprintf* para producir una salida de variable cadena que presenta el texto según el formato introducido y el valor de un número áureo.

```
>> S = sprintf('Un número áureo es %6.3f', (1+sqrt(5))/2)

S =

Un número áureo es  1.618
```

Por último se obtiene un vector columna de dos elementos con las aproximaciones de los números irracionales *e* y π.

```
>> S = '2.7183 3.1416';
>> A = sscanf(S,'%f')

A =

    2.7183
    3.1416
```

Control sobre la posición en un fichero

Los comandos *fseek, ftell, feof, frewind* y *ferror* controlan la posición en los ficheros. El comando *fseek* permite situar el indicador de posición en un fichero previamente abierto para la realización sobre él de operaciones de entrada y salida. El comando *ftell* devuelve la situación actual del indicador de posición dentro de un fichero. El comando *feof* indica si el indicador de posición está situado al final del fichero. El comando *frewind* coloca el indicador de posición al principio del fichero. Su sintaxis es la siguiente y el comando *ferror* devuelve un mensaje que informa acerca del error cometido en cualquier operación de entrada o salida sobre un fichero previamente abierto con *fopen.* La sintaxis de estos comandos es la siguiente:

fseek(fid, n, 'origin')	*Sitúa el indicador en la posición resultante de moverse n bytes desde el origen indicado por origin dentro del fichero de identificador fid previamente abierto con fopen. Si n>0, el indicador de posición se mueve n bytes hacia adelante en dirección al final del fichero. Si n<0, el indicador de posición se mueve n bytes hacia atrás en dirección al principio del fichero. Si n=0, el indicador de posición no cambia. Los valores que puede tomar el argumento origin son: 'bof' o -1 (el origen está en el comienzo del fichero), 'cof' o 0 (el origen está en la posición corriente del indicador) y eof' o 1 (el origen está en el del fichero final*
n = ftell(fid)	*Devuelve el número de bytes que hay desde el comienzo del fichero cuyo identificador es fid (previamente abierto con fopen) hasta la situación actual del indicador de posición*
feof(fid)	*Devuelve 1 si el indicador de posición está situado al final del fichero con identificador fid (previamente abierto) y 0 en caso contrario*
frewind(fid)	*Sitúa el indicador de posición del fichero con identificador fid (previamente abierto) a su principio*
ferror(fid) salida	*Devuelve un mensaje explicativo del error cometido en la más reciente operación de entrada sobre el fichero con identificador fid previamente abierto*
[message, errnum]= ferror(fid)	*Además del mensaje de error, devuelve su número. Si el número no es cero, es segura la existencia de un error en la más reciente operación de entrada o salida sobre el fichero*

Como ejemplo escribimos los números enteros de dos bytes desde el 1 hasta el 5 en un fichero binario de nombre *cinco.bin*. A continuación se comprueba la situación actual del indicador de posición en el fichero y se mueve 6 bytes hacia adelante, comprobando que la operación ha sido correcta. Posteriormente se mueve el indicador de posición 4 bytes hacia atrás y se constata sobre qué cifra ha quedado situado.

```
>> A=[1:5];
fid=fopen('cinco.bin','w');
fwrite(fid,A,'short');
fclose(fid);
fid=fopen('cinco.bin','r');
n=ftell(fid)

n =

     0
```

Como el número de bytes que hay desde el principio del fichero hasta la situación actual del indicador de posición ha resultado ser *n=0*, evidentemente el indicador de posición se encuentra situado al principio del fichero, o sea, en el primer valor, que es el 1. Otra forma de ver que el indicador de posición se encuentra sobre el 1 es utilizar el comando *fread* para leer solamente el primer elemento del fichero binario *cinco.bin*:

```
>> fid=fopen('cinco.bin','r');
principio=fread(fid,1,'short')

principio =

     1
```

Ahora vamos a mover el indicador de posición 6 bytes hacia adelante y a comprobar su nueva situación:

```
>> fid=fopen('cinco.bin','r');
fseek(fid,6,'bof');
n=ftell(fid)

n =

     6

>> principio=fread(fid,1,'short')

principio =

     4
```

Hemos visto que el indicador de posición se ha movido 6 bytes a la derecha para situarse sobre el elemento 4 (hay que tener presente que cada elemento del fichero ocupa 2 bytes). Ahora vamos a mover el indicador de posición 4 unidades a la izquierda y a constatar sobre qué elemento ha quedado situado:

```
>> fseek(fid,-4,'cof');
n=ftell(fid)

n =

     4

>> principio=fread(fid,1,'short')

principio =

     3
```

Finalmente, el indicador de posición ha quedado situado a 4 bytes del inicio del fichero, o sea, sobre el elemento 3 (no olvidemos que cada elemento del fichero ocupa 2 bytes).

Exportación e importación de datos a Lotus 123, a ASCII delimitado y a formatos cadena y gráfico

Existe un grupo de comandos en MATLAB que permiten exportar e importar datos entre Lotus 123 y MATLAB. También hay otro grupo de comandos que permiten exportar e importar datos entre ficheros ASCII con delimitadores y MATLAB. La tabla siguiente presenta estos comandos.

A = wk1read(fichero)	*Lee el fichero de hoja cálculo Lotus 123 de nombre fichero.wk1y lo importa como la matriz A de MATLAB cuyas filas y columnas son las de la hoja de cálculo*
A = wk1read(fichero,F,C)	*Lee el fichero de hoja cálculo Lotus 123 de nombre fichero.wk1 a partir de la fila F y la columna C, y lo importa como la matriz A de MATLAB cuyas filas y columnas son las de la hoja de cálculo*
A = wk1read(fichero,F,C,R)	*Lee el rango R de datos del fichero de hoja cálculo Lotus 123 de nombre fichero.wk1 a partir de la fila F y la columna C, y lo importa como la matriz A de MATLAB cuyas filas y columnas son las de la hoja de cálculo*
A = wk1write(fichero, M)	*Escribe la matriz M de MATLAB como el fichero de hoja cálculo Lotus 123 de nombre fichero.wk1 cuyas filas y columnas son las de la matriz M*
A=wk1write(fichero,M,F,C)	*Escribe la matriz M de MATLAB como el fichero de hoja cálculo Lotus 123 de nombre fichero.wk1 cuyas filas y columnas son las de la matriz M empezando en la fila F y la columna C*
M = dlmread(fichero, D)	*Lee el fichero formateado especificado cuyos datos están separados por el delimitador D y lo devuelve como la matriz M*

M = dlmread(fichero, D, F,C)	*Lee el fichero especificado cuyos datos están separados por el delimitador D y lo devuelve como la matriz M que comienza en la fila F y en la columna C*
M = dlmwrite(fichero, M,D)	*Escribe la matriz M en el fichero formateado especificado, cuyos datos están separados por el delimitador D*
M = dlmwrite(fichero,D,F,C)	*Escribe la matriz M, empezando en la fila F y la columna C, en el fichero formateado especificado, cuyos datos están separados por el delimitador D*
A = imread(fichero,*fmt*)	*Lee la imagen en formato gráfico fmt del fichero dado en escala de grises o color verdadero*
[X,map] = imread(fichero,*fmt*)	*Lee la imagen en formato gráfico fmt del fichero dado indexado en X y con sus colores asociados en map*
[...] = imread(fichero)	*Intenta inferir el formato del fichero desde su contenido*
[...] = imread(...,idx) **(CUR, ICO, y TIFF sólo)**	*Lee una imagen de orden idx en un fichero TIFF, CUR o ICO*
[...] = imread(...,ref) **(HDF sólo)**	*Lee la imagen de número idx en un fichero HDF*
[...] = imread(...,'Colorfondo',BG) **(PNG sólo)**	*Lee una imagen con color de fondo e intensidad de escala de grises dadas*
[A,map,alpha] = imread(fichero, fmt...)	*Lee una imagen en formato gráfico fmt del fichero dado aplicando máscara de transparencia*
[A,map,alpha] = imread(...) **(PNG sólo)**	*Devuelve la máscara de transparencia*
imwrite(A,fichero,*fmt*)	*Escribe la imagen en formato gráfico fmt en el fichero dado en escala de grises o color verdadero*
imwrite(X,map,fichero,*fmt*)	*Escribe la imagen indexada en X y su mapa de colores asociado map en el fichero dado con formato gráfico fmt*
imwrite(...,filename)	*Escribe la imagen en el fichero dado infiriendo el formato de la extensión del nombre del fichero*
imwrite(...,Param1,Val1, Param2,Val2...)	*Especifica los parámetros de control de varias características del fichero de salida*
info = imfinfo(fichero,*fmt*)	*Da información del fichero gráfico con formato fmt*
A = strread('C')	*Lee datos numéricos de la cadena C*
A = strread('C','',N)	*Lee N líneas de datos numéricos de la cadena C*
A = strread('C','',p,valor,...)	*Lee la cadena C a medida según el parámetro p y el valor*
A = strread('str','',N,p,valor,...)	*Lee N filas de C a medida según el parámetro p y el valor*
[A,B,C,...]=strread('C','formato')	*Lee la cadena C con el formato especificado*
[A,B,C,...] = strread ('C','format0',N)	*Lee N líneas de la cadena C con el formato especificado*
[A,B,C,...] = strread ('C','formato',p,value,...)	*Lee la cadena C con el formato especificado según el parámetro p y el valor*
[A,B,C,...]=strread ('C','format',N,param,value,...)	*Lee N líneas de la cadena C con el formato especificado según el parámetro p y el valor*

[A,B,C,...] = textread('fichero','formato')	*Lee datos del fichero de texto usando el formato dado*
[A,B,C,...] = textread('fichero','format',N)	*Lee datos del fichero texto usando el formato dado N veces*
[...] = textread(...,'p','value',...)	*Lee datos a medida utilizando el parámetro y valor especificados*

Los posibles valores del formato gráfico de fichero *fmt* se presentan en la tabla siguiente:

Formato	***Tipo de fichero***
'bmp'	*Windows Bitmap (BMP)*
'cur'	*Windows Cursor resources (CUR)*
'hdf'	*Hierarchical Data Format (HDF)*
'ico'	*Windows Icon resources (ICO)*
'jpg' or 'jpeg'	*Joint Photographic Experts Group (JPEG)*
'pcx'	*Windows Paintbrush (PCX)*
`png'	*Portable Network Graphics (PNG)*
'tif' or 'tiff'	*Tagged Image File Format (TIFF)*
'xwd'	*X Windows Dump (XWD)*

La tabla siguiente presenta los tipos de imagen que *imread* puede leer.

Formato	***Variantes***
BMP	*1-bit, 4-bit, 8-bit y 24-bit en imágenes sin comprimir; 4-bit y 8-bit en imágenes comprimidas (RLE)*
CUR	*1-bit, 4-bit y 8-bit en imágenes sin comprimir*
HDF	*8-bit en conjuntos de datos imagen con o sin mapa de colores asociado; 24-bit y 8-bit en conjuntos de datos imagen*
ICO	*1-bit, 4-bit, y 8-bit en imágenes sin comprimir*
JPEG	*Cualquier imagen baselina JPEG (8 o 24-bit); imágenes JPEG con alguna extensión usada comúnmente*
PCX	*1-bit, 8-bit, and 24-bit en imágenes*
PNG	*Cualquier imagen PNG, incluyendo 1-bit, 2-bit, 4-bit, 8-bit, y 16-bit en imágenes de escalas de grises; 8-bit y 16-bit en imágenes indexadas; 24-bit y 48-bit en imágenes RGB*
TIFF	*Cualquier imagen baselina TIFF, incluyendo 1-bit 8-bit y 24-bit en imágenes sin comprimir; 1-bit, 8-bit, 16-bit y 24-bit en imágenes comprimidas; 1-bit en imágenes comprimidas con CCITT; también 16-bit en escala de grises, 16-bit indexado y 48-bit en imágenes RGB*
XWD	*1-bit y 8-bit ZPixmaps; XYBitmaps; 1-bit XYPixmaps*

La tabla siguiente presenta todos los formatos que admiten los comandos *strread* y *testread.*

Formato	***Acción***	***Salida***
Literals (caracteres)	*Ignora los caracteres de correspondencia*	*Ninguna*
%d	*Lee un valor entero con signo*	*Doble array*
%u	*Lee un valor entero*	*Doble array*
%f	*Lee un valor en punto flotante*	*Doble array*
%s	*Lee con separación de espacio en blanco*	*Celda array de cadenas*
%q	*Lee una cadena entre comillas dobles*	*Celda array de cadenas. Sin incluir dobles comillas*
%c	*Lee carcateres incluyendo el espacio en blanco*	*Array carácter*
%[...]	*Lee la cadena más larga conteniendo los caracteres especificados entre corchetes*	*Celda array de cadenas*
%[^...]	*Lee la cadena más larga no vacía conteniendo los caracteres no especificados entre corchetes*	*Celda array de cadenas*
%*... en lugar de %	*Ignora la correspondencia entre caracteres especificada por * *	*Sin salida*
%w... en lugar de %	*Lee la anchura de campo especificada por w. El formato %f soporta %w.pf, donde w es la anchura de campo y p es la precisión*	

Los posibles pares (parámetro, valor) a utilizar como opciones a medida en los comandos *strread* y *testread* se presentan en el cuadro siguiente:

Parámetro	***Valor***	***Acción***
whitespace	*Cualquiera de la siguiente lista*	*Characters, *, como espacio en blanco. Por defecto es \b\r\n\t.*
	b *f* *n* *r* *t* ** *\" or "* *%%*	*Retroceso* *Forma del identificador* *Nueva línea* *Retorno de carro* *Tabulador horizontal* *Backslash (retroceder hacia atrás un espacio)* *Marca con comillas simples* *Signo de porcentaje*
delimiter	*Carácter delimitador*	*Especifica el carácter delimitador*
expchars	*Carácter exponente*	*Por defecto es* `eEdD`.

bufsize	*Entero positivo*	*Máxima longitud de cadena en bytes (4095)*
headerlines	*Entero positivo*	*Ignora el número especificado de líneas al comienzo del fichero*
commentstyle	*MATLAB*	*Ignora caracteres después de* `%`
commentstyle	*shell*	*Ignora caracteres después de* #
commentstyle	*c*	*Ignora caracteres entre* `/* y */`
commentstyle	*c++*	*Ignora caracteres después de* `//`

Como primer ejemplo leemos la información del fichero *canoe.tif*.

```
>> info = imfinfo('canoe.tif')
info =
                    Filename:
'C:\MATLAB6p1\toolbox\images\imdemos\canoe.tif'
                FileModDate: '25-Oct-1996 23:10:40'
                    FileSize: 69708
                      Format: 'tif'
               FormatVersion: []
                       Width: 346
                      Height: 207
                    BitDepth: 8
                   ColorType: 'indexed'
             FormatSignature: [73 73 42 0]
                   ByteOrder: 'little-endian'
              NewSubfileType: 0
               BitsPerSample: 8
                 Compression: 'PackBits'
   PhotometricInterpretation: 'RGB Palette'
                 StripOffsets: [9x1 double]
             SamplesPerPixel: 1
                RowsPerStrip: 23
             StripByteCounts: [9x1 double]
                 XResolution: 72
                 YResolution: 72
              ResolutionUnit: 'Inch'
                    Colormap: [256x3 double]
         PlanarConfiguration: 'Chunky'
                  TileWidth: []
                 TileLength: []
                TileOffsets: []
             TileByteCounts: []
                Orientation: 1
                  FillOrder: 1
           GrayResponseUnit: 0.0100
             MaxSampleValue: 255
             MinSampleValue: 0
               Thresholding: 1
```

En el siguiente ejemplo se lee la sexta imagen del fichero *flowers.tif*.

```
>> [X,map] = imread('flowers.tif',6);
```

En el siguiente ejemplo se lee la cuarta imagen de un fichero HDF.

```
>> info = imfinfo('skull.hdf');
[X,map] = imread('skull.hdf',info(4).Reference);
```

En el siguiente ejemplo se lee una imagen PNG de 24 bits con transparencia completa.

```
>> bg = [255 0 0];
A = imread('image.png','BackgroundColor',bg);
```

A continuación se presenta un ejemplo con *sprintf* y con *strread*.

```
>> s = sprintf('a,1,2\nb,3,4\n');
[a,b,c] = strread(s,'%s%d%d','delimiter',',')

a =
    'a'
    'b'
b =
     1
     3
c =
     2
     4
```

Si el fichero *misdatos.dat* tiene como primera línea *Sally Type1 12.34 45 Yes,* realizaremos una lectura de esta primera columna en formato libre.

```
>> [names,types,x,y,answer] = textread('misdatos.dat','%s %s %f ...
    %d %s',1)

names =
          'Sally'
types =
          'Type1'
x =
        12.34000000000000
y =
          45
answer =
          'Yes'
```

A continuación utilizamos el comando *strread*.

```
>> s = sprintf('a,1,2\nb,3,4\n');
[a,b,c] = strread(s,'%s%d%d','delimiter',',')

a =

    'a'
    'b'


b =

     1
     3

c =

     2
     4
```

4.3 Funciones de procesamiento de sonido

El módulo básico de MATLAB dispone de un grupo de funciones que leen y escriben ficheros de sonido. Estas funciones se presentan en la tabla siguiente:

Funciones generales de sonido	
μ=lin2mu(y)	*Convierte señal lineal de audio de amplitud -1≤y≤1 a señal μ-codificada de audio con 0≤μ≤255*
Y=mu2lin(μ)	*Convierte señal de audio μ-codificada (0≤μ≤255) a señal lineal de audio (-1≤y≤1)*
sound(y,Fs) **sound(y)** **sound(y,Fs,b)**	*Convierte la señal del vector y en sonido con frecuencia de muestreo Fs* *Convierte la señal del vector y en sonido con tasa de muestreo 8192 Hz* *Usa b bits/muestra al convertir y en sonido con frecuencia de muestreo Fs*
Funciones específicas de estaciones SPARC	
auread('f.au') **[y,Fs,bits] =** **uread('f.au')**	*Lee el fichero de sonido NeXT/SUN f.au* *Da la tasa de muestreo en Hz y el número de bits por muestra usados para codificar los datos en el fichero f.au*
auwrite(y, 'f.au') **auwrite(y,Fs, 'f.au')**	*Escribe el fichero de sonido NeXT/SUN f.au* *Escribe el fichero de sonido f.au y especifica la tasa de muestreo en Hertz*

Funciones de sonido .WAV	
wavplay(y,Fs)	*Reproduce la señal de audio almacenada en el vector y con tasa de muestreo Fs*
wavread('f.wav') **[y,Fs,bits] =** **wavread('f.wav')**	*Lee el fichero de sonido f.wav* *Devuelve la tasa de muestreo Fs y el número de bits por muestra al leer el fichero de sonido f.wav*
wavrecord(n, fs)	*Graba n muestras de señal digital de audio muestreada a tasa fs*
wavwrite(y, 'f.wav') **wavwrite(y,Fs, 'f.wav')**	*Escribe el fichero de sonido f.wab* *Escribe el fichero de sonido f.wab con tasa de muestreo Fs*

Ejercicio 4-1. Se considera la matriz mágica de orden 5, y se trata de escribir su inversa en un fichero binario de nombre magica.bin.

Comenzamos hallando la matriz del problema:

```
>> M=magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

A continuación abrimos un fichero de nombre *magica.bin*, con permiso de lectura y escritura para guardar en él la matriz *M*. Utilizamos el permiso 'w+' porque el fichero no existe previamente y hay que crearlo nuevo a la vez que se abre, y además vamos a necesitar escribir en él (ya que el fichero no existe previamente, también se podría utilizar el permiso 'a+').

```
>> fid=fopen('magica.bin','w+')

fid =

     3
```

El sistema le ha asignado el identificador 3 a nuestro fichero, y a continuación escribimos en él la matriz *M*.

```
>> fwrite(3,M)

ans =

    16
```

Ya hemos escrito la matriz *M* en el fichero binario *magica.bin* de identificador 3. MATLAB devuelve el número de elementos que tiene el fichero, que en este caso son 25. A continuación cerramos el fichero y la información se graba en disco.

```
>> fclose(3)

ans =

     0
```

Como la respuesta es cero la grabación al cierre ha sido correcta y el fichero recién creado aparecerá ya en el directorio activo.

```
>> dir

.             ..            cinco.bin     cosint.m      exponen.txt
 id4.bin       magica.bin
```

Puede verse el fichero recién creado en el directorio activo con sus propiedades.

```
>> !dir

El volumen de la unidad C no tiene etiqueta.
El nfmero de serie del volumen es: 1059-8290

 Directorio de C:\MATLAB6p1\work

03/01/2001  19:50    <DIR>          .
03/01/2001  19:50    <DIR>          ..
10/06/2000  23:41               457 cosint.m
10/01/2001  22:14                64 id4.bin
10/01/2001  23:17               231 exponen.txt
11/01/2001  00:12                10 cinco.bin
12/01/2001  23:09                16 magica.bin
               5 archivos            778 bytes
               2 dirs  18.027.282.432 bytes libres
```

Ejercicio 4-2. Considerar la matriz identidad de orden 4 y grabarla en un fichero binario con formato de punto flotante de 32 bits. Recuperar posteriormente dicho fichero y leer su contenido en forma matricial (tal y como se había grabado). Después añadirle a la matriz anterior una columna de unos y grabarla como fichero binario con el mismo nombre inicial. Volver a leer el fichero binario para comprobar su contenido definitivo.

Comenzamos generando la matriz identidad de orden 4:

```
>> I=eye(4)

I =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Abrimos un fichero binario de nombre *id4.bin*, en el que vamos a guardar la matriz *I*, con permiso de escritura:

```
>> fid=fopen('id4.bin','w+')

fid =

     3
```

Grabamos la matriz *I* en el fichero previamente abierto con formato de punto flotante de 32 bits:

```
>> fwrite(3,I,'float32')

ans =

    16
```

Una vez grabados los 16 elementos de la matriz, cerramos el fichero:

```
>> fclose(3)

ans =

     0
```

Para leer el contenido del fichero grabado anteriormente lo abrimos con permiso de lectura:

```
>> fid=fopen('id4.bin','r+')

fid =

     3
```

Ahora realizamos la lectura de los 16 elementos del fichero abierto con la misma estructura matricial y el mismo formato con que fue guardado.

```
>> [R,count]=fread(3,[4,4],'float32')

R =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1


count =

    16
```

Una vez comprobado el contenido, cerramos el fichero:

```
>> fclose(3)

ans =

     0
```

A continuación abrimos el fichero con el permiso adecuado de escritura para poder añadir información sin perder la ya existente:

```
>> fid=fopen('id4.bin','a+')

fid =

     3
```

Ahora añadimos al fichero una columna de unos al final de su contenido y posteriormente lo cerramos:

```
>> fwrite(3,[1 1 1 1]','float32')

ans =

     4

>> fclose(3)

ans =

     0
```

Ahora abrimos el fichero con permiso de lectura para ver su contenido:

```
>> fid=fopen('id4.bin','r+')

fid =

     3
```

Por último leemos los 20 elementos del fichero en la forma matricial adecuada y comprobamos que se le ha añadido una columna de unos al final:

```
>> [R,count]=fread(3,[4,5],'float32')

R =

     1     0     0     0     1
     0     1     0     0     1
     0     0     1     0     1
     0     0     0     1     1


count =

    20
```

Ejercicio 4-3. Generar un fichero ASCII de nombre log.txt que contenga los valores de la función logaritmo neperiano para valores de la variable entre 1 y 2 separados una décima. El formato del texto en el fichero ha de consistir en dos columnas de números reales de punto flotante, de tal forma que en la primera aparezcan los valores de la variable y en la segunda los correspondientes valores de la función exponencial. Por último, escribir los comandos de forma que el contenido del fichero se liste en la pantalla.

```
>> x = 1:.1:2;
y= [x; log(x)];
fid=fopen('log.txt','w');
fprintf(fid,'%6.2f  %12.8f\n', y);
fclose(fid)

ans =

     0
```

Ahora vamos a ver cómo puede presentarse la información directamente por pantalla sin necesidad de guardarla en disco:

```
>> x = 1:.1:2;
y= [x; log(x)];
fprintf('%6.2f  %12.8f\n', y)
```

```
  1.00    0.00000000
  1.10    0.09531018
  1.20    0.18232156
  1.30    0.26236426
  1.40    0.33647224
  1.50    0.40546511
  1.60    0.47000363
  1.70    0.53062825
  1.80    0.58778666
  1.90    0.64185389
  2.00    0.69314718
```

Ejercicio 4-4. Leer el fichero ASCII de nombre log.txt generado en el ejercicio anterior. El formato del texto ha de consistir en dos columnas de números reales con la máxima precisión en el mínimo espacio, de tal forma que en la primera aparezcan los valores de la variable y en la segunda los correspondientes valores de la función exponencial.

```
>> fid=fopen('log.txt');
a = fscanf(fid,'%g  %g', [2 inf]);
a = a'

a =

    1.0000         0
    1.1000    0.0953
    1.2000    0.1823
    1.3000    0.2624
    1.4000    0.3365
    1.5000    0.4055
    1.6000    0.4700
    1.7000    0.5306
    1.8000    0.5878
    1.9000    0.6419
    2.0000    0.6931

>> fclose(fid);
```

5

Funciones matemáticas del módulo básico de MATLAB

5.1 Funciones matemáticas elementales

El grupo de funciones elementales del módulo básico de MATLAB que se presentan en la tabla siguiente ya fueron estudiadas en el capítulo relativo a la definición de variables (tanto escalares como vectoriales y matriciales, y tanto reales como complejas). No obstante, sólo a modo de resumen,se vuelven a presentar aquí.

abs	*Módulo o valor absoluto*
acos, acosh	*Arco coseno y arco coseno hiperbólico*
acot, acoth	*Arco cotangente y arco cotangente hiperbólico*
acsc, acsch	*Arco cosecante y arco cosecante hiperbólico*
angle	*Argumento*
asec, asech	*Arco secante y arco secante hiperbólico*
asin, asinh	*Arco seno y arco seno hiperbólico*
atan, atanh	*Arco tangente y arco tangente hiperbólico*
atan2	*Arco tangente en el cuarto cuadrante*
ceil	*Redondeo al entero más próximo*
complex	*Forma un complejo con sus componentes real e imaginaria*
conj	*Complejo conjugado*
cos, cosh	*Coseno y coseno hiperbólico*
cot, coth	*Cotangente y cotangente hiperbólica*
csc, csch	*Cosecante y cosecante hiperbólica*
exp	*Exponencial*
fix	*Elimina la parte decimal*

floor	*Mayor entero menor o igual que un real dado*
gcd	*Máximo común divisor*
imag	*Parte imaginaria de un número complejo*
lcm	*Mínimo común múltiplo*
log	*Logaritmo natural (neperiano)*
log2	*Logaritmo en base 2*
log10	*Logaritmo decimal*
mod	*Módulo*
nchoosek	*Coeficiente binomial*
real	*Parte real de un número complejo*
rem	*Resto de la división*
round	*Redondeo al entero más cercano*
sec, sech	*Secante y secante hiperbólica*
sign	*Función signo*
sin, sinh	*Seno y seno hiperbólico*
sqrt	*Raíz cuadrada*
tan, tanh	*Tangente y tangente hiperbólica*

Como primer ejemplo calculamos las combinaciones sin repetición de 10 elementos tomados de 4 en 4.

```
>> nchoosek(10,4)

ans =

        210.00
```

A continuación se calcula el seno y el coseno de los ángulos desde 0 a 2π de $\pi/2$ en $\pi/2$.

```
>> sin(0:pi/2:2*pi)

ans =

     0          1.00          0.00         -1.00         -0.00

>> cos(0:pi/2:2*pi)

ans =

  1.00          0.00         -1.00         -0.00          1.00
```

Posteriormente se prueban algunas propiedades de las funciones exponencial y logarítmica.

```
>> exp(2*pi*i)

ans =
   1.0000 - 0.0000i

>> exp(log(2))

ans =
          2.00

>> 2*exp(i*pi)

ans =
  -2.0000 + 0.0000i

>> 2*(cos(pi)+i*sin(pi))

ans =
  -2.0000 + 0.0000i

>> log(3+2*i)

ans =
   1.2825 + 0.5880i
```

Por último se prueban algunas propiedades de las funciones trigonométricas e hiperbólicas.

```
>> sin(pi)^2+cos(pi)^2

ans =
     1

>> (exp(5)+exp(-5))/2

ans =
   74.2099

>> cosh(5)

ans =
   74.2099
```

```
>> cosh(5)^2-sinh(5)^2
ans =
    1.0000
>> 1+tan(pi/4)^2
ans =
    2.0000
>> sec(pi/4)^2
ans =
    2.0000
```

5.2 Funciones matemáticas especiales

Existe en el módulo básico de MATLAB un grupo importante de funciones especializadas con variable real y compleja que se presentan en la siguiente tabla:

airy(Z)=airy(0,Z) **airy(2,Z)**	*Soluciones linealmente independientes de la ecuación diferencial w''-zw=0, w=w(Z) (funciones de Airy de clases 1 y 2)*
airy(1,Z) y airy(3,Z)	*Primeras derivadas de las funciones de Airy de clases 1 y 2*
besselj(k,Z) **bessely(k,Z)**	*Funciones de Bessel de 1ª y 2ª clase soluciones de la ecuación de Bessel $Z^2w''+Zw'+(Z^2-k^2)w=0$, w=w(Z)*
besseli(k,Z) **besselk(k,Z)**	*Funciones de Bessel de 1ª y 2ª clase modificadas soluciones de la ecuación de Bessel $Z^2w''+Zw'+(Z^2+k^2)w=0$, w=w(Z)*
besselh(k,1,Z) **besselh(k,2,Z)**	*Funciones de Bessel de 3ª clase (o de Hankel), que cumplen:* *besselh(k,1,Z)=besselj(k,Z)+ibessely(k,Z)* *besselh(k,2,Z)=besselj(k,Z)-ibessely(k,Z)*
gamma(A)	*Función gamma* $\Gamma(a)=\int_0^\infty e^{-t}t^{a-1}dt$
gammainc(X,A)	*Función gamma incompleta* $\Gamma(x,a)=\frac{1}{\Gamma(a)}\int_0^x e^{-t}t^{a-1}dt$
gammaln **(A y X vectores)**	*Logaritmo de la función gamma* $Log\,\Gamma(a)$
beta(Z,W)	*Función beta* $\beta(z,w)=\int_0^1(1-t)^{w-1}t^{z-1}dt=\frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)}$
betainc(X,Z,W)	*Beta incompleta* $I_x(z,w)=\frac{1}{\beta(z,w)}\int_0^x(1-t)^{w-1}t^{z-1}dt$
betaln(X,Z,W)	*Logaritmo de la función beta* $Log\,\beta(z,w)$

[SN,CN,DN] = ellipj(U,M)	*Funciones elípticas de Jacobi* $u = \int_0^{\Phi} \frac{1}{\sqrt{1-m\sin^2(\Phi)}} d\Phi$ $SN(u) = sin(\phi),\ CN(u) = cos(\phi),\ DN(u) = \sqrt{1-m\sin^2(\Phi)}$
K = ellipke(M) **[K,E] = ellipke(M)**	*Integrales elípticas completas de 1ª (K) y 2ª (M) clase* $k(m) = \int_0^1 \sqrt{\frac{1-t^2}{1-mt^2}} dt = \int_0^{\pi/2} \frac{1}{\sqrt{1-m\sin^2(\theta)}} d\theta = F(\pi/2, m)$ $e(m) = \int_0^1 \sqrt{\frac{1-mt^2}{1-t^2}} dt = \int_0^{\pi/2} \sqrt{1-m\sin^2(\theta)} d\theta = E(\pi/2, m)$
erf(X) = función de error **erfc(X) = coplementaria de función de error** **erfcx(X) = complementaria escalada de error** **erfinv(X) = inversa de error**	$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt = 2F_{N(0,1/2)}(x)$ *F = distribución normal* $erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt = 1 - erf(x)$ $erfcx(x) = e^{x^2} erfc(x) \cong \frac{1}{\sqrt{\pi}} \frac{1}{x}$ $x = erfinv(y) \Leftrightarrow y = erf(x)$
expint(X)	*Exponencial integral* $expint(x) = \int_x^{\infty} \frac{e^{-t}}{t} dt$
factorial(n)	*Función factorial n!= n(n-1)(n-2)…3.2.1*
P = legendre(n,X) **S = legendre(n,X,'sch')**	*Funciones asociadas de Legendre* $P_{m,n}(x) = (-1)^m (1-x^2)^{m/2} P_n^{m)}(x)$ *con* $P_n(x) = [(x^2-1)^n]^{n)} / (n!2^n)$ *Funciones de Smith seminormalizadas* $S_{m,n}(x) = (-1)^m (2(n-m)!/(n+m)!)^{1/2} P_{m,n}(x)$ $m>0$
pow2(Y) **pow2(F,Y)**	2^y $f*2^y$
[N,D] = rat(X) **S = rats(X)**	*Devuelve los arrays N y D tales que N./D aproxima X con una tolerancia por defecto de 1.e-6*norm(X(:),1)* *Devuelve X en formato racional*

Es necesario precisar que todas las variables que aparecen en letras mayúsculas son variables reales vectoriales, salvo la variable Z, que es vectorial compleja. Las letras minúsculas designan variables escalares reales.

Como primer ejemplo calculamos los valores de la función de distribución de una normal (0,1/2) en los números entre 0 y 1 distanciados 1/4.

```
>> erf(0:1/4:1)

ans =

     0          0.28          0.52          0.71          0.84
```

A continuación calculamos los valores de la función gamma en los cuatro primeros números pares.

```
>> gamma([2 4 6 8])

ans =

          1.00          6.00        120.00       5040.00
```

Teniendo presente el resultado anterior comprobamos que $\Gamma(a) = (a-1)!$ para los cuatro primeros pares

```
>> [factorial(1),factorial(3),factorial(5),factorial(7)]

ans =

          1.00          6.00        120.00       5040.00
```

A continuación, para $z = 3$ y $w = 5$, comprobaremos que:

beta(z,w) = exp(gammaln(z)+gammaln(w)-gammaln(z+w))
betaln(z,w) = gammaln(z)+gammaln(w)-gammaln(z+w)

```
>> beta(3,5)

ans =

          0.01

>> exp(gammaln(3)+gammaln(5)-gammaln(3+5))

ans =

          0.01

>> betaln(3,5)
```

```
ans =

         -4.65

>> gammaln(3)+gammaln(5)-gammaln(3+5)

ans =

         -4.65
```

También para los valores de *z=3* y *w=5* comprobaremos que:

beta(z,w)=Γ(z)Γ(w)/ Γ(z+w)

```
>> gamma(3)*gamma(5)/gamma(3+5)

ans =

          0.01

>> beta(3,5)

ans =

          0.01
```

Ahora se calcula una matriz de polinomios de Legendre.

```
>> legendre(2,(6:8))

ans =

  1.0e+002 *

   0.5350             0.7300             0.9550
        0 - 1.0649i        0 - 1.4549i        0 - 1.9049i
  -1.0500            -1.4400            -1.8900
```

Ahora calculamos el valor de la función de Airy en el vector complejo *Z=(i,1+i,2i).* Asimismo comprobamos para ese mismo vector complejo *Z* y *k=1*que:

*besselh(k,1,Z)=besselj(k,Z)+i*bessely(k,Z)* y *besselh(k,2,Z)=besselj(k,Z)-i*bessely(k,Z)*

```
>> airy([i,1+i,2*i])
ans =
   0.3315 - 0.3174i   0.0605 - 0.1519i  -0.1096 - 0.9116i
```

```
>> besselh(1,1,[i,1+i,2*i])

ans =

  -0.3832 - 0.0000i  -0.0156 - 0.2927i  -0.0890 - 0.0000i

>> besselj(1,[i,1+i,2*i])+i*bessely(1,[i,1+i,2*i])

ans =

  -0.3832            -0.0156 - 0.2927i  -0.0890

>> besselh(1,2,[i,1+i,2*i])

ans =

   0.3832 + 1.1303i   1.2440 + 1.0227i   0.0890 + 3.1813i

>> besselj(1,[i,1+i,2*i])-i*bessely(1,[i,1+i,2*i])

ans =

   0.3832 + 1.1303i   1.2440 + 1.0227i   0.0890 + 3.1813i
```

5.3 Funciones para conversión de sistemas de coordenadas

MATLAB permite trabajar con distintos sistemas de coordenadas e implementa funciones para hacer conversiones de coordenadas de unos sistemas a otros. Como los sistemas de coordenadas menos habituales son las cilíndricas y las esféricas, a continuación se presenta una breve aclaración teórica sobre estos dos sistemas.

En un sistema de *coordenadas cilíndricas*, un punto P del espacio se representa por una tripleta ordenada (r, θ, z), donde:

r es la distancia del origen *(O)* a la proyección de P *(P')* en el plano XY
θ es el ángulo entre el eje X y el segmento OP'
z es la distancia PP'

En un sistema de *coordenadas esféricas*, un punto P del espacio se representa por una tripleta ordenada (ρ, θ, ϕ), donde:

ρ es la distancia de P al origen
θ es el mismo ángulo que el usado en coordenadas cilíndricas
ϕ es el ángulo entre el eje Z positivo y el segmento OP

Fácilmente se llega a las ecuaciones de conversión siguientes:

Cilíndricas a rectangulares:

$$x = r\cos\theta$$
$$y = r\,\text{sen}\,\theta$$
$$z = z$$

Rectangulares a cilíndricas:

$$r = \sqrt{x^2 + y^2}$$
$$\theta = arctg\frac{y}{x}$$
$$z = z$$

Esféricas a rectangulares:

$$x = \rho\ sen\phi\cos\theta$$
$$y = \rho\ sen\phi\ sen\theta$$
$$z = \rho\ \cos\theta$$

Rectangulares a esféricas:

$$\rho = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = arctg\frac{y}{x}$$

A continuación se presenta la sintaxis de las funciones que ofrece MATLAB en su módulo básico para transformación de coordenadas.

[THETA,RHO,Z] = cart2pol(X,Y,Z) **[THETA,RHO] = cart2pol(X,Y)**	*Transforma cartesianas a cilíndricas* *Transforma cartesianas a polares*
[THETA,PHI,R] = cart2sph(X,Y,Z)	*Transforma cartesianas a esféricas*
[X,Y,Z] = pol2cart(THETA,RHO,Z) **[X,Y] = pol2cart(THETA,RHO)**	*Transforma cilíndricas a cartesianas* *Transforma polares a cartesianas*
[x,y,z] = sph2cart(THETA,PHI,R)	*Transforma esféricas a cartesianas*

En el ejemplo siguiente se transforma el punto (π,1,2) de coordenadas cilíndricas a cartesianas.

```
>> [X,Y,Z]=pol2cart(pi,1,2)

X =

    -1
```

```
Y =

  1.2246e-016

Z =

     2
```

A continuación se transforma el punto (1,1,1) de coordenadas cartesianas a esféricas y cilíndricas.

```
>> [X,Y,Z]=cart2sph(1,1,1)

X =

    0.7854

Y =

    0.6155

Z =

    1.7321

>> [X,Y,Z]=cart2pol(1,1,1)

X =

    0.7854

Y =

    1.4142

Z =

     1
```

En el ejemplo siguiente se transforma a cartesianas el punto $(2,\pi/4)$ en polares.

```
>> [X,Y] = pol2cart(2,pi/4)

X =

   -0.3268

Y =

    0.7142
```

5.4 Funciones de análisis de datos y análisis estadístico básico

El módulo básico de MATLAB dispone de un grupo de funciones relativas al análisis de datos en general (estadística descriptiva), correlación y diferencias finitas. La tabla siguiente presenta este tipo de funciones.

Análisis de datos	
cumprod(A)	*Vector de productos acumulados de las columnas de la matriz A*
cumprod(A,2)	*Vector de productos acumulados de las filas de la matriz A*
max(A)	*Vector con los valores máximos de las filas de la matriz A*
max(A,[ ],2)	*Vector con los valores máximos de las columnas de la matriz A*
max(A,B)	*Matriz con los máximos de los términos correspondientes de A y B*
mean(A)	*Vector con las medias de las columnas de la matriz A*
mean(A,2)	*Vector con las medias de las filas de la matriz A*
median(A)	*Vector con las medianas de las columnas de la matriz A*
median(A,2)	*Vector con las medianas de las filas de la matriz A*
min(A)	*Vector con los valores mínimos de las filas de la matriz A*
min(A,[ ],2)	*Vector con los valores mínimos de las columnas de la matriz A*
min(A,B)	*Matriz con los mínimos de los términos correspondientes de A y B*
std(A)	*Vector con las cuasidesviaciones típicas de las columnas de la matriz A*
std(A,1)	*Vector con las desviaciones típicas de las columnas de la matriz A*
std(A,0,2)	*Vector con las cuasidesviaciones típicas de las filas de la matriz A*
std(A,1,2)	*Vector con las desviaciones típicas de las filas de la matriz A*
var(A)	*Vector con las cuasivarianzas de las columnas de la matriz A*
var(A,1)	*Vector con las varianzas de las columnas de la matriz A*
var(A,W)	*Vector con las varianzas de las columnas de la matriz A ponderadas por las frecuencias absolutas incluidas en el vector W*
sum(A)	*Vector con las sumas de las columnas de la matriz A*
sum(A,2)	*Vector con las sumas de las filas de la matriz A*
perms(V)	*Matriz cuyas filas son las posibles permutaciones de los n elementos del vector fila V*
primes(n)	*Genera un vector con los números primos menores o iguales que n*
factor(n)	*Genera un vector con los factores primos de n*
prod(V)	*Producto de los elementos del vector V*
prod(A)	*Vector cuyos elementos son los productos de las columnas de la matriz A*
prod(A,2)	*Vector cuyos elementos son los productos de las filas de la matriz A*
sort(V)	*Ordena los elementos de V en orden ascendente*
sort(A)	*Ordena cada columna de la matriz A*
sort(A,2)	*Ordena cada fila de la matriz A*
sortrows(A)	*Ordena las filas de A como un grupo en orden ascendente*
sortrows(A,V)	*Ordena la matriz A según los números de columna del vector V*

Diferencias finitas	
del2(U)	*Laplaciana discreta del vector U* $(d^2u/dx^2 + d^2/dy^2)/4)$
del2(U,h)	*Laplaciana discreta del vector U con intervalo h*
del2(U,hx,hy)	*Laplaciana discreta del array rectangular U con intervalos hx y hy*
del2(U,hx,hy,…)	*Laplaciana discreta del array multidimensional U con intervalos h*
FX = gradient(F)	*Gradiente numérico de la matriz F en la dirección de la columna x (dF/dx)*
[FX,FY] = gradient(F)	*Gradiente numérico de la matriz F en las direcciones de la columna x (dF/dx) y de la fila y (dF/dx)*
[Fx,Fy,Fz,...] = gradient(F)	*Gradiente numérico de la matriz F en las direcciones de todas las componentes (dF/dx, dF/dx, dF/dz,…)*
Y = diff(X)	*Vector de diferencias del vector X. Si X es una matriz diff(X) devuelve los vectores de diferencias por filas*
Y = diff(X,*n*)	*Diferencias sucesivas de orden n*
Correlación	
corrcoef(X)	*Matriz de correlaciones, donde X es una matriz cuyas columnas son variables y cuyas filas son los valores de estas variables*
corrcoef(x,y)	*Coeficiente de correlación de los vectores columna x e y*
cov(X)	*Matriz de covarianzas, donde X es una matriz cuyas columnas son variables y cuyas filas son los valores de estas variables*
cov(x,y)	*Covarianza de los vectores columna x e y*

A continuación se presentan varios ejemplos:

```
>> A=[1 2 3; 4 5 6; 7 8 9];
>> cumprod(A)

ans =

     1     2     3
     4    10    18
    28    80   162

>> cumprod(A,2)

ans =

     1     2     6
     4    20   120
     7    56   504

>> max(A)

ans =

     7     8     9
```

```
>> max(A,[],2)

ans =

     3
     6
     9

>> mean(A)

ans =

     4     5     6

>> mean(A,2)

ans =

     2
     5
     8

>> std(A)

ans =

     3     3     3

>> std(A,1)

ans =

    2.4495    2.4495    2.4495

>> std(A,0,2)

ans =

     1
     1
     1

>> std(A,1,2)

ans =

    0.8165
    0.8165
    0.8165
```

```
>> prod(A)

ans =

    28    80   162

>> prod(A,2)

ans =

     6
   120
   504

>> sort(A)

ans =

     1     2     3
     4     5     6
     7     8     9

>> [Ax,Ay]=gradient(A)

Ax =

     1     1     1
     1     1     1
     1     1     1

Ay =

     3     3     3
     3     3     3
     3     3     3

>> diff(A)

ans =

     3     3     3
     3     3     3

>> corrcoef(A)

ans =

     1     1     1
     1     1     1
     1     1     1

>> cov(A)

ans =

     9     9     9
     9     9     9
     9     9     9
```

Ejercicio 5-1. El número de pétalos de 13 flores de una determinada especie es el siguiente: 8, 10, 6, 5, 8, 11, 8, 10, 7, 10, 7, 10 y 9. Calcular la media, la varianza y el coeficiente de variación.

Definimos el vector *V* cuyos componentes son los números de pétalos y a continuación calculamos la media varianza y coeficiente de variación con las funciones adecuadas de MATLAB.

```
>> V=[8, 10, 6, 5, 8, 11, 8, 10, 7, 10, 7, 10, 9];
>> Media=mean(V)

Media =

    8.3846

>> Varianza=var(V,1)

Varianza =

    3.0059

>> Coeficiente_de_variacion=std(V,1)/mean(V)

Coeficiente_de_variacion =

    0.2068
```

El coeficiente de variación se ha calculado como cociente entre la desviación típica y la media.

Ejercicio 5-2. Dada la variable NP cuyos datos son 8, 10, 6, 5, 8, 11, 8, 10, 7, 10, 7, 10 y 9, obtener sus diferencias sucesivas de órdenes 1 y 2.

Comenzamos definiendo un vector *NP* que contiene los valores dados y a continuación utilizamos adecuadamente la función *diff* de MATLAB.

```
>> NP=[8, 10, 6, 5, 8, 11, 8, 10, 7, 10, 7, 10, 9];
>> Diferencias_sucesivas_orden1=diff(NP)

Diferencias_sucesivas_orden1 =

  2    -4    -1     3     3    -3     2    -3     3    -3     3    -1

>> Diferencias_sucesivas_orden2=diff(NP,2)

Diferencias_sucesivas_orden2 =

    -6     3     4     0    -6     5    -5     6    -6     6    -4
```

Ejercicio 5-3. Sea una variable aleatoria Y cuya función de densidad está dada por f(y) = 6y (1-y) si 0 < y <1, y f(y) = 0 en otro caso. Calcular P(0,5 < Y < 8).

La variable aleatoria *Y* se ajusta a una distribución beta de parámetros *z*=2, *w*=2, ya que:

$$\frac{1}{\beta(z,w)} = 6$$

```
>> 1/beta(2,2)

ans =

    6.0000
```

. El problema nos pide:

$$P(0{,}5 < Y < 0{,}8) = P(Y < 0{,}8) - P(Y \le 0{,}5) = I_{0,8}(2,2) - I_{0,5}(2,2)$$

o lo que es lo mismo:

$$\frac{1}{\beta(2,2)}\int_0^{0,8}(1-t)t\,dt - \frac{1}{\beta(2,2)}\int_0^{0,5}(1-t)t\,dt = F(0{,}8) - F(0{,}5)$$

donde *F* es la función de distribución de una variable beta(2,2).

Este valor se puede calcular utilizando la función *betainc* de MATLAB como sigue:

```
>> betainc(0.8,2,2)-betainc(0.5,2,2)

ans =

    0.3960
```

Ejercicio 5-4. Con los datos de la economía española correspondientes al Producto Interior Bruto a precios de mercado en pesetas constantes de 1980 que se presentan a continuación:

1970	*10.595,1*	*1977*	*14.685,4*
1971	*11.111,5*	*1978*	*14.934,4*
1972	*11.948,6*	*1979*	*15.003,1*
1973	*12.948,0*	*1980*	*15.209,1*
1974	*13.715,1*	*1981*	*15.195,9*
1975	*13.803,4*	*1982*	*15.379,2*
1976	*14.184,7*	*1983*	*15.679,7*

calcular las tasas de variación interanual del PIBpm en pesetas constantes de 1980.

Las tasas de variación interanual del *PIBpm* (variable *TVPIBpm*) se calculan mediante la fórmula *TVPIBpm*= (*PIBpm,t / PIBpm,t*-1) - 1)*100, donde la serie *PIBpm,t* es la dada y la serie *PIBpm,t-1* es la dada desplazada un año hacia atrás. Teniendo presente que hay que considerar ambas series con el mismo número de elementos, pueden expresarse en MATLAB como sigue:

```
>> PIBpmt=[11111.5, 11948.6, 12948, 13715.1, 13803.4, 14184.7,
14685.4, 14934.4, 15000.31, 15209.1, 15195.9, 15379.2, 15679.7]

PIBpmt =

  1.0e+004 *

  1.1112    1.1949    1.2948    1.3715    1.3803    1.4185    1.4685
  1.4934    1.5000    1.5209    1.5196    1.5379    1.5680

>> PIBpmt_1=[10595.1, 11111.5, 11948.6, 12948, 13715.1, 13803.4,
14184.7, 14685.4, 14934.4, 15000.31, 15209.1, 15195.9, 15379.2]

PIBpmt_1 =

  1.0e+004 *

    1.0595    1.1112    1.1949    1.2948    1.3715    1.3803    1.4185
    1.4685    1.4934    1.5000    1.5209    1.5196    1.5379
```

El cálculo de la serie de tasas de variación puede calcularse en MATLAB utilizando la siguiente expresión:

```
>> TVPIBpm=(PIBpmt./PIBpmt_1-ones(1,13))*100

TVPIBpm =

    4.8740    7.5336    8.3642    5.9245    0.6438    2.7624    3.5299
    1.6956    0.4413    1.3919   -0.0868    1.2062    1.9539
```

Ahora colocamos los vectores en modo columna para una mejor interpretación de los resultados. Es necesario tener en cuenta que para abarcar varios decimales significativos en la salida es conveniente utilizar el formato adecuado de MATLAB, en nuestro caso el formato largo. La sintaxis de MATLAB que presenta las tres columnas con cada una de las series puede ser la siguiente:

```
>> format long

>> [PIBpmt',PIBpmt_1',TVPIBpm']
```

```
ans =

  1.0e+004 *

   1.11115000000000   1.05951000000000   0.00048739511661
   1.19486000000000   1.11115000000000   0.00075336363227
   1.29480000000000   1.19486000000000   0.00083641598179
   1.37151000000000   1.29480000000000   0.00059244670992
   1.38034000000000   1.37151000000000   0.00006438159401
   1.41847000000000   1.38034000000000   0.00027623628961
   1.46854000000000   1.41847000000000   0.00035298596375
   1.49344000000000   1.46854000000000   0.00016955615782
   1.50003100000000   1.49344000000000   0.00004413300836
   1.52091000000000   1.50003100000000   0.00013919045673
   1.51959000000000   1.52091000000000  -0.00000867901454
   1.53792000000000   1.51959000000000   0.00012062464217
   1.56797000000000   1.53792000000000   0.00019539377861
```

Ejercicio 5-5. Un agente de seguros vende pólizas a 5 individuos, todos de la misma edad. De acuerdo con las tablas actuariales, la probabilidad de que un individuo con esa edad viva 30 años más es de 3/5. Determinar la probabilidad de que dentro de 30 años vivan:

a) al menos 3 individuos;
c) como mucho, 2.

Como quiera que dentro de 30 años la circunstancia de cada individuo será que viva o que no viva, y al menos una de las dos se ha de presentar, la situación para cada individuo se ajusta a una variable de Bernoulli con probabilidad de éxito (vivir 30 años más) igual a 3/5 = 0,6. Al considerar los 5 individuos, estamos ante una variable aleatoria X binomial con $n = 5$ y $p = 0.6$, $X = B(5, 0{,}6)$. Si llamamos $F(x)$ a la función de distribución de $X = B(5, 0{,}6)$ en el punto x, los apartados del problema se calculan como sigue:

a) Habrá que calcular $P(X \geq 3)$ o, lo que es lo mismo, $1\text{-}p(X < 3) = 1\text{-}F(2)$.
b) Habrá que calcular $P(X \leq 2)$ o, lo que es lo mismo, $F(2)$.

La primera probabilidad es equivalente a:

$$1 - F(2) = 1 - \binom{5}{2} 0{,}6^2 0{,}4^3 - \binom{5}{1} 0{,}6^1 0{,}4^4 - \binom{5}{0} 0{,}4^5$$

que puede calcularse mediante MATLAB como sigue:

```
>> 1-nchoosek(5,2)*0.6^2*0.4^3-nchoosek(5,1)*0.6*0.4^4-
   nchoosek(5,0)*0.4^5

ans =

   0.68256000000000
```

La segunda probabilidad es equivalente a:

$$F(2)=\binom{5}{2}0{,}6^2 0{,}4^3+\binom{5}{1}0{,}6^1 0{,}4^4+\binom{5}{0}0{,}4^5$$

que puede calcularse mediante MATLAB como sigue:

```
>> nchoosek(5,2)*0.6^2*0.4^3+nchoosek(5,1)*0.6*0.4^4+
   nchoosek(5,0)*0.4^5

ans =

   0.31744000000000
```

Ejercicio 5-6. El número medio de automóviles que llega a una estación de suministro de gasolina es de 210 por hora. Si dicha estación puede atender a un máximo de 10 automóviles por minuto, determinar la probabilidad de que en un minuto dado lleguen a la estación de suministro más automóviles de los que puede atender.

El número aleatorio de automóviles que llegan a la estación de servicio en un minuto puede representarse por una variable X de Poisson de parámetro $m = 210/60 = 3{,}5$, ya que m es el número medio de llegadas por minuto (teníamos 210 llegadas a la hora).

La probabilidad que vamos a obtener vendrá dada por $P(X > 10)$, ya que para que lleguen a la estación más automóviles por minuto de los que se puedan atender es necesario que lleguen más de 10 por minuto. Pero $P(X > 10) = 1 - P(X \leq 10) = 1 - F(10)$, siendo F la función de distribución de una variable aleatoria de Poisson de parámetro 3,5.

Para calcular la probabilidad pedida tendremos en cuenta que:

$$1-F(10)=1-\sum_{k=0}^{10}\frac{e^{-3{,}5}3{,}5^k}{k!}$$

que puede calcularse en MATLAB como sigue:

```
>> 1-exp(-3.5)*(1+3.5+3.5^2/factorial(2)+3.5^3/factorial(3)+
3.5^4/factorial(4)+3.5^5/factorial(5)+3.5^6/factorial(6)+
3.5^7/factorial(7)+3.5^8/factorial(8)+3.5^9/factorial(9)+
3.5^10/factorial(10))

ans =

   0.00101939443762
```

Ejercicio 5-7. La proporción de individuos de una población con renta superior a los dos millones de pesetas es de 0,005%. Determinar la probabilidad de que entre 5.000 individuos consultados haya como mucho 2 con ese nivel de renta, supuesto que todos los consultados respondan.

Teóricamente, la variable X definida por el número de individuos, de entre los 5.000 que tienen un nivel de renta superior a los dos millones de pesetas es una variable binomial con $n = 5.000$ y $p = 0{,}00005$. La probabilidad pedida sería $P(X \leq 2)$. Pero este cálculo con números tan extremos es molesto.

Ahora bien, habida cuenta de que p es muy pequeño ($p = 0{,}00005$) y de que n es muy grande ($n = 5.000$), y además $m = np = 0{,}25 < 5$ y $p < 0{,}1$, ya podemos aproximar la variable $X = B(5.000, 0{,}00005)$ por una variable de Poisson de parámetro $m = np = 5.000 * 0{,}00005 = 0{,}25$. Luego $P(X \leq 2)$ lo podemos calcular mediante la distribución de Poisson de parámetro 0,25 como $F(2)$, es decir:

$$F(2) = \sum_{k=0}^{2} \frac{e^{-0,25} 0,25^k}{k!}$$

que puede calcularse en MATLAB como sigue:

```
>> exp(-0.25)*(1+0.25+0.25^2/factorial(2))

ans =

   0.99783850331024
```

Ejercicio 5-8. Supongamos que la variable aleatoria Y tiene una función de densidad dada por: $f(y) = ky^3 e^{-y/2}$ con $y > 0$ y $f(y) = 0$ si $y \leq 0$.

1) Hallar k para que f sea una función de densidad.
2) Calcular $P(2 < Y < 4)$.

La función de densidad de esta variable tiene la forma de una gamma con parámetros $a = 4$ y $p = 1/2$, para lo cual el valor de k ha de ser el siguiente: $k = (1/\Gamma(4))(1/2)^4 = (1/3!)(1/2)^4 = 1/96$.

La función de distribución de una variable aleatoria *gamma* de parámetros *a* y *p* en el punto *x*, $F_{\gamma(a,p)}(x)$ se puede calcular a través de la función *gammainc* de MATLAB mediante la siguiente relación:

$$F_{\gamma(a,p)}(x) = \frac{p^a}{\Gamma(a)} \int_0^x e^{-pt} t^{a-1} dt = \frac{1}{\Gamma(a)} \int_0^{px} e^{-u} u^{a-1} du = \Gamma(px, a) = gammainc(px, a)$$

Sencillamente se ha realizado el cambio de variable $pt=u$.

Por lo tanto, ya podemos escribir que $P(2 < Y < 4) = P(Y < 4) - P(Y \leq 2) = F(4) - F(2)$, siendo $F(y)$ la función de distribución de una gamma de parámetros $a = 4$ $p = 1/2$. La sintaxis MATLAB que resuelve el problema es la siguiente:

```
>> gammainc(1/2*4,4)-gammainc(1/2*2,4)

ans =

    0.1239
```

Ejercicio 5-9. En un proceso de fabricación se sabe que el número aleatorio de unidades defectuosas producidas diariamente viene dado por una variable de Poisson de parámetro 10. Determinar la probabilidad de que en 150 días el número de unidades defectuosas producidas supere 1.480.

Sea X_i la variable definida por el número de piezas defectuosas producidas en el día *i*. X_i es una variable de Poisson de parámetro 10. El número de piezas defectuosas producidas en 150 días vendrá dado por la variable $Y=\sum X_i$ con i=1...150, que será una variable de Poisson de parámetro 1.500. El problema nos pide $P(Y > 1.480)$, valor que podría calcularse con la distribución de Poisson, pero al ser el parámetro tan alto, utilizamos la aproximación a la distribución normal.

Las variables X_i tienen $E(X_i) = 10$ y $V(X_i) = 10$ por ser de Poisson con $\lambda = 10$. Por el teorema central del límite, $Y=\sum X_i$ puede aproximarse por una $N(150E(X_i), (150\ V(X_i))^{1/2}) = N(1500, 1500^{1/2})$.

El problema nos pide $P(Y > 1.480) = 1 - P(Y \leq 1480) = 1 - P(X \leq -20/1500^{1/2}) = 1-F_{N(0,1)}(-20/1500^{1/2}) = F_{N(0,1)}(20/1500^{1/2})$. X es una variable normal (0,1).

La función de distribución de una variable aleatoria normal (0,1) en el punto *x*, $F_{N(0,1)}(x)$ se puede calcular a través de la función *erf* de MATLAB mediante la siguiente relación:

$$F_{N(0,1)}(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{t^2}{2}}dt=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{0} e^{-\frac{t^2}{2}}dt+\frac{1}{\sqrt{2\pi}}\int_{0}^{x} e^{-\frac{t^2}{2}}dt=\frac{1}{2}+\frac{1}{\sqrt{2\pi}}\int_{0}^{x} e^{-\frac{t^2}{2}}dt$$

Si ahora realizamos el cambio de variable $t = \sqrt{2}u$ se puede escribir:

$$F_{N(0,1)}(x)=\frac{1}{2}+\frac{1}{\sqrt{\pi}}\int_{0}^{x/\sqrt{2}} e^{-u^2}du=\frac{1}{2}+\frac{erf(x/\sqrt{2})}{2}$$

Por lo tanto, ya podemos escribir que:

$$F_{N(0,1)}\left(\frac{20}{\sqrt{1500}}\right)=\frac{1}{2}+\frac{erf(20/\sqrt{3000})}{2}$$

Los cálculos pueden realizarse mediante la siguiente sintaxis MATLAB:

```
>> 1/2+erf(20/sqrt(3000))/2

ans =

    0.6972
```

Ejercicio 5-10. Si la variable aleatoria Y es normal (2,3), calcular P(-0,15≤Y≤2,123) y P(Y≤3).

Comenzaremos calculando la segunda probabilidad, que es más sencilla.

$$P(Y\le 3)=P\left(\frac{Y-2}{3}\le\frac{3-2}{3}\right)=F_{N(0,1)}\left(\frac{1}{3}\right)=\frac{1}{2}+\frac{erf\left(1/\left(3\sqrt{2}\right)\right)}{2}$$

El cálculo se realiza con MATLAB así:

```
>> 1/2+erf(1/(3*sqrt(2)))/2
ans =
    0.6306
```

La segunda probabilidad se calcula como sigue:

$$P(-0{,}15 \le Y \le 2{,}123) = P\left(\frac{-0{,}15-2}{3} \le \frac{Y-2}{3} \le \frac{2{,}123-2}{3}\right) = F_{N(0,1)}\left(\frac{0{,}123}{3}\right) - F_{N(0,1)}\left(\frac{-2{,}15}{3}\right)$$

$$= F_{N(0,1)}\left(\frac{0{,}123}{3}\right) + F_{N(0,1)}\left(\frac{2{,}15}{3}\right) - 1 = \frac{1}{2} + \frac{erf\left(0{,}123/\left(3\sqrt{2}\right)\right)}{2} + \frac{1}{2} + \frac{erf\left(2{,}15/\left(3\sqrt{2}\right)\right)}{2} - 1$$

Esta expresión se calcula en MATLAB como sigue:

```
>> 1/2+erf(0.123/(3*sqrt(2)))/2+1/2+erf(2.15/(3*sqrt(2)))/2-1

ans =

    0.2796
```

Ejercicio 5-11. Se lanzan consecutivamente 6 dados perfectos. Calcúlese la probabilidad de obtener los 6 números distintos.

El número de casos favorables vendrá dado por las permutaciones sin repetición de 6 elementos, que se calculan mediante la expresión *factorial*(6).

El número de casos posibles vendrá dado por las variaciones con repetición de 6 elementos tomados de 6 en 6. O sea 6^6.

Luego la probabilidad pedida se podrá calcular utilizando la expresión *factorial*(6)/(6^6). Los cálculos en MATLAB se realizan como sigue:

```
>> factorial(6)/(6^6)

ans =

    0.0154
```

Ejercicio 5-12. Una urna contiene 8 bolas blancas y 7 negras y hacemos una extracción de 2 bolas. En el supuesto de que hemos visto que una de estas bolas es negra, ¿cuál es la probabilidad de que la otra también lo sea?

Sea *A* el suceso de *que al extraer dos bolas, una al menos sea negra* y sea *B* el suceso de *que al extraer dos bolas las dos sean negras*. El suceso intersección de *A* con *B* coincide con *B* y su probabilidad se halla como cociente de casos favorables entre casos posibles de la forma:

$$\binom{7}{2} : \binom{15}{2}$$

Este valor se calcula con la expresión MATLAB siguiente:

```
>> B=nchoosek(7,2)/nchoosek(15,2)
```

```
B =

    0.2000
```

Por otra parte, la probabilidad de *A* será la de que las dos bolas sean negras, o una negra y una blanca, que se calculará mediante la suma de dos probabilidades ajustables a la distribución hipergeométrica (se trata de la probabilidad de una unión disjunta de sucesos, cuyo resultado es la suma de probabilidades). La probabilidad de *A* será la siguiente:

$$\frac{\binom{7}{1}\cdot\binom{8}{1}}{\binom{15}{2}}+\frac{\binom{7}{2}\cdot\binom{8}{0}}{\binom{15}{2}}$$

Este valor se calcula mediante la expresión MATLAB siguiente:

```
>> A=nchoosek(7,1)*nchoosek(8,1)/nchoosek(15,2)+…
   nchoosek(7,2)*nchoosek(8,0)/nchoosek(15,2)

A =

    0.7333
```

La probabilidad pedida, que es la probabilidad condicionada del suceso *B* por el suceso *A*, será el cociente de las dos probabilidades anteriores.

```
>> B/A

ans =

    0.2727
```

Para la última fórmula se ha tenido en cuenta que $p(B/A) = p(A \cap B)/p(A) =$ 0,2 / 0,73 = 0,27

Ejercicio 5-13. Tenemos 6 cajas que contienen 12 tornillos cada una (entre buenos y malos). Una tiene 8 buenos y 4 defectuosos. Dos de ellas contienen 6 buenos y 6 malos, y las otras tres cajas contienen 4 buenos y 8 malos. Se elige una caja al azar y se extraen 3 tornillos, sin reemplazamiento; de éstos, 2 son buenos y 1 defectuoso. ¿Cuál es la probabilidad de que la caja elegida contenga 6 tornillos buenos y 6 malos?

Sean A_1, A_2 y A_3 los sucesos consistentes en que las cajas elegidas contengan 8 buenos y 4 malos, 6 buenos y 6 malos, y 4 buenos y 8 malos, respectivamente, y sea B el suceso de que al extraer 3 tornillos de una caja resulten 2 buenos y 1 malo.

Por el teorema de Bayes la probabilidad condicionada pedida será:

$$P = P(A_2 / B) = \frac{P(B / A_2)P(A_2)}{\sum P(B / A_i)P(A_i)}$$

Por otra parte, $P(A_1)$=1/6, $P(A_2)$=2/6 y $P(A_3)$=3/6.

Observamos que las probabilidades condicionadas son hipergeométricas cuyos valores son:

$$P(B / A_1) = \frac{\binom{8}{2}.\binom{4}{1}}{\binom{12}{3}}, \quad P(B / A_2) = \frac{\binom{6}{2}.\binom{6}{1}}{\binom{12}{3}}, \quad P(B / A_3) = \frac{\binom{4}{2}.\binom{8}{1}}{\binom{12}{3}}$$

Si a las probabilidades anteriores les llamamos *P*1, *P*2 y *P*3, respectivamente, podemos realizar su cálculo con MATLAB como sigue:

```
>> P1=nchoosek(8,2)*nchoosek(4,1)/nchoosek(12,3)

P1 =

    0.5091

>> P2=nchoosek(6,2)*nchoosek(6,1)/nchoosek(12,3)

P2 =

    0.4091

>> P3=nchoosek(4,2)*nchoosek(8,1)/nchoosek(12,3)

P3 =

    0.2182
```

Ya podemos calcular la probabilidad pedida *P* como sigue:

```
>> P=P2*(2/6)/(P1*(1/6)+P2*(2/6)+P3*(3/6))

P =

    0.4128
```

Ejercicio 5-14. Tenemos una sucesión de variables aleatorias independientes X_1, X_2, ..., X_n con la misma función de densidad: $f(x) = 8\,e^{-8x}$ si $x>0$ y $f(x)=0$ si $x \leq 0$. Si consideremos 100 de estas variables, calcúlese:

1) probabilidad de que su suma sea menor o igual que 10;
2) probabilidad de que la suma esté entre 11 y 13.

Por la forma de su función de densidad, estas variables son exponenciales de parámetro p=8. Por otra parte, la suma de 100 variables exponenciales independientes de parámetro 8 será una gamma (100,8).

Sea $Y = X_1+X_2+ \ldots +X_{100}$. Y es una variable aleatoria gamma (100,8).

1) El problema nos pide $P(Y \leq 10) = F(10)$, donde F es la función de distribución de una gamma (100,8).

2) El problema nos pide $P(11< Y <13)$, siendo Y una gamma (100,8).

$P(11< Y <13) = P(Y <13) - P(Y \leq 11) = F(13) - F(11)$, donde F es la función de distribución de una variable gamma (100,8).

Para realizar el cálculo se utiliza la siguiente expresión MATLAB:

```
>> gammainc(8*13,100)-gammainc(8*11,100)
ans =
    0.5541
```

Ejercicio 5-15. Un distribuidor mayorista de gasolina dispone de tanques de almacenaje que contienen una cantidad fija de gasolina y que se llenan cada lunes. La proporción de esta reserva que se vende durante la semana es de sumo interés para el distribuidor. Mediante observaciones al cabo de muchas semanas se descubrió que se podría representar el modelo de esta proporción mediante una distribución beta con z = 4 y w = 2. Encontrar la probabilidad de que el mayorista venda al menos el 90% de su reserva durante una semana dada.

Se trata de calcular $P(Y > 0{,}9)$, donde Y es una distribución beta(4,2). Tenemos $P(Y > 0{,}9) = 1\text{-}P(Y \leq 0{,}9) = 1\text{-}F(0{,}9)$, donde F es la función de distribución de una variable aleatoria beta(4,2). Este valor se calcula mediante la expresión MATLAB siguiente:

```
>> 1-betainc(0.9,4,2)
ans =
    0.0815
```

Ejercicio 5-16. Calcúlense las siguientes probabilidades:

1) $P(\chi^2(10) > 5{,}031)$
2) $P(6{,}821 < \chi^2(15) < 15{,}13)$
3) $P(\chi^2(150) > 128)$
4) $P(50 < \chi^2(65) < 60)$

Nota: Los números entre paréntesis son los grados de libertad.

1) $p(\chi^2(10) > 5{,}031) = 1 - p(\chi^2(10) \leq 5{,}031,10)$, que se calcula con MATLAB de la siguiente forma:

```
>> 1-gammainc(1/2*5.031,10/2)

ans =

    0.8891
```

Se ha utilizado que una chi-cuadrado con *n* grados de libertad equivale a una gamma (*n*/2,1/2).

2) $p(6{,}821 < \chi^2(15) < 15{,}13) = p(\chi^2(15) < 15{,}13) - p(\chi^2(15) < 6{,}821)$, que se calcula con MATLAB de la siguiente forma:

```
>> gammainc(1/2*15.13,15/2)-gammainc(1/2*6.821,15/2)

ans =

    0.5203
```

3) $p(\chi^2(150) > 128) = p(\sqrt{2\chi^2(150)} > \sqrt{2*128} = p(Y > 16)$, siendo Y una $N(\sqrt{2*150-1},1)$. En este caso hemos aproximado la chi-cuadrado por la normal, ya que los grados de libertad son muy elevados (>30). El valor se calcula con MATLAB de la siguiente forma:

```
>> 1-(1/2+erf((16-sqrt(2*150-1))/sqrt(2))/2)

ans =

    0.9018
```

4) Se tiene que $p(50 < \chi^2(65) < 60) = p(\chi^2(65) < 60) - p(\chi^2(65) < 50) = p(\sqrt{2\chi^2(65)} < \sqrt{2*60}) - p(\sqrt{2\chi^2(65)} < \sqrt{2*50}) = p(Y < \sqrt{120}) - p(Y < 10)$, siendo Y una $N(\sqrt{2*65-1},1)$. El valor se calcula con MATLAB de la siguiente forma:

```
>> 1/2+erf((sqrt(120)-sqrt(2*65-1))/sqrt(2))/2
   -(1/2+erf((10-sqrt(2*65-1))/sqrt(2))/2)

ans =

    0.2561
```

Ejercicio 5-17. Dadas cuatro variables aleatorias independientes N(0,5), W, X, Y, Z se definen las siguientes variables aleatorias:

$$S = 2W + 3X - Y + Z + 30, \quad T = W^2 + X^2 + Y^2 + Z^2, \quad U = \sqrt{T/4}$$

Calcular:

1) P(S < 42)
2) P(T < 48,075)
3) P(U < 6,973)

1) La variable S es una normal de media 2*0+3*0-0+0+30=30 y de varianza 4*25+9*25+25+25=375, luego S es $N(30, \sqrt{375})$. Luego $p(S < 42)$ se calculará mediante la expresión MATLAB siguiente:

```
>> 1/2+erf((42-30)/sqrt(2*375))/2

ans =

    0.7323
```

2) $T/25 = (W/5)^2 + (X/5)^2 + (Y/5)^2 + (Z/5)^2$ es una suma de 4 cuadrados de N(0,1), esto es, es una chi-cuadrado con cuatro grados de libertad, luego $p(T < 48{,}075) = p(T/25 < 48{,}075/25) = p(\chi^2(4) < 48{,}075/25)$, valor que se halla mediante la expresión MATLAB siguiente:

```
>> gammainc(1/2*(48.075/25),2)

ans =

    0.2501
```

3) La variable $4U^2/25 = (W/5)^2 + (X/5)^2 + (Y/5)^2 + (Z/5)^2$ es una chi-cuadrado con cuatro grados de libertad, luego $p(U < 6.973) = p(U^2 < 6{,}973^2) = p(4U^2/25 < 4*6{,}973^2/25) = p(\chi^2(4) < 4*6{,}973^2/25)$, valor que se calculará mediante la expresión MATLAB siguiente:

```
>> gammainc((1/2)*(4*6.973^2/25),2)

ans =

    0.9000
```

Ejercicio 5-18. La cantidad de tornillos por segundo producidos por una máquina sigue una distribución exponencial de parámetro p=1/4. Calcular la probabilidad de que la máquina produzca más de 4 tornillos en un segundo determinado.

Sea *X* la variable aleatoria que representa la cantidad de tornillos producidos por la máquina en un segundo determinado. La variable *X* es una exponencial de parámetro ¼ o, lo que es lo mismo, podemos decir que la variable *X* es una gamma (1,1/4).

El problema nos pide $P(X>4) = 1-P(X<=4) = 1-F(4)$, donde *F* es la función de distribución de una variable gamma(1,1/4) o exponencial(1/4). Este valor se hallará mediante la expresión MATLAB:

```
>> 1-gammainc(4*1/4,1)

ans =

    0.3679
```

Ejercicio 5-19. Comprobar que para x=2 la función: $\frac{\sqrt{2}Sen(x)}{\sqrt{\pi}\,x^{3/2}} - \frac{\sqrt{2}Cos(x)}{\sqrt{\pi}\sqrt{x}}$ ***es una solución de la ecuación diferencial*** $4x^2y''+4xy'+(4x^2-9)y=0$.

La ecuación diferencial puede escribirse en la forma $x^2y''+xy'+(x^2-(3/2)^2)y = 0$. Nos encontramos ante una ecuación diferencial de segundo grado y segundo orden tipo *J* de Bessel. Para resolver el problema veremos que efectivamente el valor de la función trigonométrica en el punto $x=2$ coincide con la solución de la ecuación diferencial en ese punto. Utilizaremos la siguiente sintaxis MATLAB:

```
>> y=(sqrt(2)*sin(2))/(sqrt(pi)*2^(3/2))-
   (sqrt(2)*cos(2))/(sqrt(pi)*sqrt(2))

y =

    0.4913

>> x=besselj(3/2,2)

x =

    0.4913
```

Ejercicio 5-20. Hallar en x=1 una solución particular de la ecuación diferencial $y''- xy=0$.

Nos encontramos ante una ecuación diferencial tipo Airy. Por lo tanto hallamos su solución en el punto x=1 mediante la siguiente sintaxis MATLAB:

```
>> x=airy(1)

x =

    0.1353
```

Ejercicio 5-21. Resolver las integrales: a = $\int_0^\infty x^3 e^{-x}\,dx$ ***y b =*** $\int_0^\infty x^2 e^{-x^3}\,dx$

Se tiene que $a = \Gamma(4)$.

Por otra parte, para el cambio de variable $x^3 = t \Rightarrow b = \int_0^\infty \frac{1}{3} e^{-t}\,dt = \Gamma(1)/3$.

Por lo tanto, para el cálculo de a y b usaremos la siguiente sintaxis MATLAB:

```
>> a=gamma(4)

a =

     6

>> b=(1/3)*gamma(1)

b =

    0.3333
```

Ejercicio 5-22. ¿Cuántos triángulos distintos pueden formarse con los vértices de un dodecágono?

Se trata de calcular el número de combinaciones de 12 elementos tomados de 3 en 3. Para ello se usa la siguiente sintaxis:

```
>> a=nchoosek(12,3)

a =

   220
```

Ejercicio 5-23. Probar para n=100 y p=30 que: $\binom{n}{p}+\binom{n}{p+1}=\binom{n+1}{p+1}$

El problema puede resolverse comprobando la igualdad de ambos miembros mediante la siguiente sintaxis:

```
>> format long
>> a=nchoosek(100,30)+nchoosek(100,30+1)

a =

    9.569697812847435e+025

>> b=nchoosek(100+1,30+1)

b =

    9.569697812847439e+025
```

Ejercicio 5-24. Resolver las siguientes integrales:

$$\int_0^{\infty} \frac{x^3}{e^x} dx\,, \quad \int_0^5 \frac{x^3}{e^x} dx$$

Como $\Gamma(p) = \int_0^{\infty} x^{p-1} e^{-x} dx$, la primera integral se resuelve de la forma:

```
>> gamma(4)

ans =

     6
```

Como $\Gamma(x,p) = \frac{1}{\Gamma(p)} \int_0^x t^{p-1} e^{-t} dt = gammainc(x,p)$, la segunda integral se resuelve de la forma siguiente:

```
>> gamma(4)*gammainc(5,4)

ans =

    4.4098
```

Ejercicio 5-25. Resolver las integrales siguientes:

$$\int_0^1 x^4 (1-x)^3 dx \qquad \int_0^5 \frac{x^4}{(1-x)^{-3}} dx\,, \quad \int_0^8 \frac{\sqrt[4]{2-\sqrt[3]{x}}}{\sqrt{x}} dx$$

$$\beta(p,q)=\int_0^1 x^{p-1}(1-x)^{q-1}dx \quad \text{y} \quad betainc(z,w)=\frac{1}{\beta(p,q)}\int_0^1 t^{z-1}(1-t)^{w-1}dt,$$

con lo que la primera integral se resuelve como:

```
>> beta(5,4)*betainc(1/2,5,4)

ans =

    0.0013
```

Para la segunda integral hacemos el cambio de variable $x^{1/3} = 2t$, con lo que la integral se convierte en la siguiente:

$$6\cdot\int_0^1 \frac{\sqrt{t}\,(1-t)^{\frac{1}{4}}\,dt}{2^{4/3}}$$

cuyo valor se calcula mediante la expresión MATLAB:

```
>> 6*2^(3/4)*beta(3/2,5/4)

ans =

    5.0397
```

Ejercicio 5-26. Hallar el valor de las integrales siguientes:

$$\int_3^\infty \frac{1}{\sqrt{6x^3-37x^2+72x-45}}dx \quad y \quad \int_0^1 \frac{x^2+1}{\sqrt{x^4-5x^2+4}}dx$$

Las primitivas correspondientes a integrales elípticas son de difícil cálculo algebraico. El hecho de estandarizar su resolución permite hallar el valor de múltiples tipos de integrales, que por cambio de variable pueden reducirse a integrales elípticas. En los ejemplos veremos varios ejercicios de este tipo. MATLAB habilita funciones simbólicas que calculan los valores de las integrales elípticas de primer, segundo y tercer orden.

$$k(m)=\int_0^1 \sqrt{\frac{1-t^2}{1-mt^2}}dt=\int_0^{\pi/2}\frac{1}{\sqrt{1-m\sin^2(\theta)}}d\theta=F(\pi/2,m)$$

$$e(m)=\int_0^1 \sqrt{\frac{1-mt^2}{1-t^2}}dt=\int_0^{\pi/2}\sqrt{1-m\sin^2(\theta)}d\theta=E(\pi/2,m)$$

La función [*K*,*E*] = *ellipke*(*m*) calcula las dos integrales anteriores.

Para la primera integral (irracional) del problema, como el polinomio subradical es de tercer grado, se hace el cambio $x = a + t^2$, siendo a una de las raíces del polinomio subradical. Tomamos la raíz $x = 3$ y hacemos el cambio $x = 3 + t^2$, con lo que obtenemos la integral:

$$\frac{1}{3}\sqrt{6}\cdot\int_0^{\infty}\frac{1}{\left(t^2+4/3\right)\left(t^2+3/2\right)}\,dt$$

Hagamos ahora el cambio $t=(2/\sqrt{3})\ tan\ u$, con lo que la integral se transforma en la elíptica completa de primer orden:

$$2\cdot\int_0^{\frac{1}{2}\pi}\frac{1}{\sqrt{9-Sen^2(u)}}\,du=\frac{2}{3}\cdot\int_0^{\frac{1}{2}\pi}\frac{1}{\sqrt{1-\frac{1}{9}Sen^2(u)}}\,du$$

cuyo valor se calcula mediante la expresión:

```
>> (2/3)*ellipke(1/9)
ans =
   1.07825782374982
```

Para la segunda integral hacemos el cambio $x = sen\ t$ y tendremos:

$$5\int_0^2\frac{1}{\sqrt{4-sen^2\ t}}\,dt-\int_0^2\sqrt{4-sen^2\ t}\,dt=\frac{5}{2}\int_0^2\frac{1}{\sqrt{1-\frac{1}{4}sen^2\ t}}\,dt-2\int_0^2\sqrt{1-\frac{1}{4}sen^2\ t}\,dt$$

Hemos reducido el problema a dos integrales elípticas, la primera completa de primera especie y la segunda completa de segunda especie, cuyo valor puede calcularse mediante la expresión:

```
>> [K,E]=ellipke(1/4)
K =
   1.68575035481260
E =
   1.46746220933943
>> I=(5/2)*K-2*E
I =
   1.27945146835264
```

Ejercicio 5-27. Calcular la longitud de un período completo de la sinusoide y=3sen(2x).

La longitud de esta curva vendrá dada por la fórmula:

$$4\int_0^{\frac{1}{4}\pi}\sqrt{1-\left(\frac{\partial}{\partial x}y(x)\right)^2}\,dt=\int_0^{\pi/2}\sqrt{1+36\cdot Cos^2(2x)}\,dx=2\int_0^{\pi/2}\sqrt{37-36\cdot Sen^2(t)}\,dt$$

En el último paso hemos hecho el cambio de variable $2x=t$, utilizando además que $cos^2(t)=1-sin^2(t)$. El valor de la integral puede calcularse ahora mediante:

```
>> [K,E]=ellipke(36/37)

K =

   3.20677433446297

E =

   1.03666851510702

>> 2*sqrt(37)*E

ans =

  12.61161680006573
```

Ejercicio 5-28. Calcular la integral $\displaystyle\int_{-2}^{\infty}3\frac{e^{-2t}}{t}\,t\;dt$

Se trata de una integral de tipo exponencial. Realizamos el cambio de variable $v=2t$ y obtenemos la integral equivalente:

$$3\int_{-4}^{\infty}\frac{e^{-t}}{t}dt$$

que se resuelve mediante la expresión MATLAB:

```
>> 3*expint(-4)

ans =

-58.89262341016866 - 9.42477796076938i
```

Suele utilizarse solamente la parte real del resultado anterior.

Ejercicio 5-29. Sean las variables X, Y y Z cuyos valores son los siguientes:

X	*Y*	*Z*
2	*4*	*2*
3	*5*	*4*
6	*10*	*6*
8	*11*	*7*
10	*15*	*10*

a) Calcular la media, la mediana, la desviación típica, la varianza y el coeficiente de variación para las tres variables, y hallar un intervalo de confianza para la media basado en cada variable con un nivel de confianza del 95%.
b) Determínese la matriz de correlaciones, deduciendo de ella el grado de dependencia entre las variables.
c) Determínese la matriz de covarianzas e interpretar el resultado.

Comenzamos introduciendo las variables X, Y y Z como columnas de la matriz *A*.

```
>> A=[2,3,6,8,10;4,5,10,11,15;2,4,6,7,10]'
A =
     2     4     2
     3     5     4
     6    10     6
     8    11     7
    10    15    10
```

Para calcular los estadísticos pedidos usamos la siguiente sintaxis MATLAB:

```
>> mean(A)
ans =
    5.8000    9.0000    5.8000
>> median(A)
ans =
     6    10     6
>> std(A,1)
ans =
    2.9933    4.0497    2.7129
>> var(A,1)
ans =
    8.9600   16.4000    7.3600
```

```
>> std(A,1)./mean(A)

ans =

    0.5161    0.4500    0.4677
```

Para calcular los intervalos de confianza usamos la sintaxis siguiente:

```
>> [mean(A)-2*std(A,1),mean(A)+2*std(A,1)]

ans =

   -0.1867    0.9006    0.3741   11.7867   17.0994   11.2259
```

Los intervalos serán: [-0,187, 117867], [0,9006, 17,0994] y [0,3741, 11,2259].

Para hallar la matriz de correlaciones usamos la siguiente sintaxis MATLAB:

```
>> corrcoef(A)

ans =

    1.0000    0.9899    0.9802
    0.9899    1.0000    0.9830
    0.9802    0.9830    1.0000
```

En la salida anterior se observa que el coeficiente de correlación entre X e Y es 0,9899, entre X y Z 0,9802 y entre Y y Z 0,9830, lo que indica la fuerte dependencia lineal positiva entre cada par de variables.

Para hallar la matriz de covarianzas usamos la siguiente sintaxis MATLAB:

```
>> cov(A)

ans =

   11.2000   15.0000    9.9500
   15.0000   20.5000   13.5000
    9.9500   13.5000    9.2000
```

A la vista de los resultados podemos decir que, como todas las covarianzas son positivas, los pares de variables varían en el mismo sentido alrededor de sus medias.

Ejercicio 5-30. Para un determinado país, sean las variables R=renta personal en unidades monetarias y H=número de personas que se van de vacaciones al extranjero y cuyos valores son los siguientes:

R	*H*
1.000	30
1.100	50
1.300	60
1.500	70
1.800	90
2.000	100
2.200	110
2.500	120

justificar si puede aceptarse una dependencia estadística lineal entre R y H.

Comenzamos introduciendo las variables *R* y *H* como vectores columna y a continuación se calcula el coeficiente de correlación como sigue:

```
>> R=[1000,1100,1300,1500,1800,2000,2200,2500]';
>> H=[30,50,60,70,90,100,110,120]';
>> corrcoef(R,H)

ans =

    1.0000    0.9862
    0.9862    1.0000
```

Se observa que el coeficiente de correlación entre *R* y *H* es $r = 0{,}9862$, lo que asegura la fuerte dependencia lineal positiva entre las dos variables.

6

Álgebra lineal numérica

6.1 Matrices numéricas

MATLAB permite trabajar con matrices numéricas de forma muy cómoda y extensa. No obstante, se trata de un programa especializado en cálculo matricial. La tabla que se presenta a continuación muestra las funciones matrices numéricas que pueden realizarse con el módulo básico de MATLAB.

expm(A)	*e^A calculada a través de autovalores*
expm1(A)	*e^A calculada a través de aproximantes de Padé*
expm2(A)	*e^A calculada a través de series de Taylor*
expm3(A)	*e^A calculada a través de la condición de la matriz de autovectores*
logm(A)	*Logaritmo neperiano de la matriz A*
sqrtm(A)	*Raíz cuadrada de la matriz cuadrada A*
funm(A, 'función')	*Aplica la función a la matriz cuadrada A*
transpose(A) o A'	*Matriz transpuesta de A*
inv(A)	*Matriz inversa de la matriz cuadrada A (A^{-1})*
det(A)	*Determinante de la matriz cuadrada A*
rank(A)	*Rango de la matriz A*
trace(A)	*Suma de los elementos de la diagonal de A*
svd(A)	*Vector V de valores singulares de A, que son las raíces cuadradas de los autovalores de la matriz simétrica A'A*
[U,S,V]=svd(A)	*Da la matriz diagonal S de valores singulares de A (ordenados de mayor a menor) y las matrices U y V tales que A=U*S*V'*

rcond(A)	*Recíproco de la condición de la matriz A*
norm(A)	*Norma de A o 2-norma (mayor valor singular de la matriz A)*
norm(A,1)	*1-norma de A (mayor suma de las columnas de A)*
norm(A,inf)	*Norma infinita de A (mayor suma de las filas de A)*
norm(A,'fro')	*F-norma de A, definida por sqrt(sum(diag(A'A)))*
cond(A) o cond(A,2) **cond(A,1)** **cond(A,inf)** **cond(A,fro)**	*Da la condición de la matriz A (cociente entre el mayor y el menor valor singular de A). Condición según la 2-norma* *Condición de A según la 1-norma* *Condición de A según la norma infinita* *Condición de A según la F-norma (norma de Frobenius)*
Z=null(A)	*Da una base ortonormal del núcleo de A (Z'Z=I). El número de columnas de Z es la nulidad de A*
Q=orth(A) ***(Q'Q=I).***	*Da una base ortonormal para el rango de A. Las columnas de Q generan el mismo espacio que las columnas de A, y el número de columnas de Q es el rango de A*
subspace(A,B)	*Da el ángulo entre los subespacios especificados por las columnas de A y de B. Si A y B son vectores, da el ángulo formado por ambos*
rref(A)	*Da la matriz reducida escalonada de Gauss-Jordan por filas de A. El número de filas no nulas de rref(A) es el rango de A*
A^p	*Matriz A elevada a la potencia escalar p*
p^A	*Escalar p elevado a la matriz A*

Como primer ejemplo consideramos una matriz cuadrada de orden 4 formada por números aleatorios distribuidos uniformemente en el intervalo [0,1] y calculamos su determinante, rango, inversa, transpuesta, traza y condición.

```
>> A=rand(4)

A =

    0.9501    0.8913    0.8214    0.9218
    0.2311    0.7621    0.4447    0.7382
    0.6068    0.4565    0.6154    0.1763
    0.4860    0.0185    0.7919    0.4057

>> det(A)

ans =

    0.1155

>> rank(A)

ans =

     4
```

```
>> inv(A)

ans =

    2.2631   -2.3495   -0.4696   -0.6631
   -0.7620    1.2122    1.7041   -1.2146
   -2.0408    1.4228    1.5538    1.3730
    1.3075   -0.0183   -2.5483    0.6344

>> A'

ans =

    0.9501    0.2311    0.6068    0.4860
    0.8913    0.7621    0.4565    0.0185
    0.8214    0.4447    0.6154    0.7919
    0.9218    0.7382    0.1763    0.4057

>> trace(A)

ans =

    2.7334

>> cond(A)

ans =

   12.7005
```

A continuación consideramos una matriz compleja B y calculamos B^3, 3^B, e^B, $ln(B)$ y $\sqrt{B}$.

```
>> B=[i,2i,3i;1+i,0,i;1,2,1-i]

B =

        0 + 1.0000i         0 + 2.0000i         0 + 3.0000i
   1.0000 + 1.0000i         0                   0 + 1.0000i
   1.0000              2.0000              1.0000 - 1.0000i

>> B^3

ans =

 -15.0000 + 4.0000i -14.0000           -20.0000 - 6.0000i
 -10.0000 + 5.0000i  -8.0000 + 6.0000i  -8.0000 + 1.0000i
        0 + 8.0000i  -4.0000 +12.0000i  -3.0000 +14.0000i
```

```
>> 3^B

ans =

  -4.8310 + 0.1746i  -5.3632 + 0.6815i  -5.0531 - 1.1152i
  -2.7685 + 0.8853i  -3.0042 + 0.6888i  -3.5365 + 0.3409i
  -0.0866 + 3.1091i  -0.1737 + 2.7987i  -1.2642 + 2.6024i

>> expm(B)

ans =

  -3.7156 + 0.9975i  -4.3209 + 1.6058i  -4.2517 + 0.2072i
  -2.0052 + 1.2226i  -2.1593 + 1.0291i  -2.8797 + 0.9181i
   0.5008 + 2.5777i   0.5409 + 2.2781i  -0.3326 + 2.1998i

>> logm(B)

ans =

   0.1213 - 0.7024i   0.1425 + 1.3871i  -1.0825 + 1.4441i
   0.1054 + 1.1681i   0.5049 - 1.2596i   0.2305 + 0.9892i
   0.9210 - 0.0031i   0.6595 + 0.6307i   1.2182 - 1.5014i

>> sqrtm(B)

ans =

   0.7128 - 0.0137i   0.1275 + 0.7962i  -0.3363 + 1.2680i
   0.3733 + 0.6374i   0.8162 - 0.4642i   0.1735 + 0.4555i
   0.4817 - 0.0899i   0.7339 + 0.2591i   1.2865 - 0.7852i
```

En el siguiente ejemplo se calcula la forma reducida escalonada por filas de Gauss-Jordan para la matriz mágica de orden 4.

```
>> A = magic(4), R = rref(A)

A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
R =
     1     0     0     1
     0     1     0     3
     0     0     1    -3
     0     0     0     0
```

En el siguiente ejemplo se calcula la raíz cuadrada y el seno de la matriz *A* anterior utilizando el comando *funm*.

```
>> funm(A,'sqrt')

ans =

   3.7584 - 0.2071i  -0.2271 + 0.4886i   0.3887 + 0.7700i   1.9110 - 1.0514i
   0.2745 - 0.0130i   2.3243 + 0.0306i   2.0076 + 0.0483i   1.2246 - 0.0659i
   1.3918 - 0.2331i   1.5060 + 0.5498i   1.4884 + 0.8666i   1.4447 - 1.1833i
   0.4063 + 0.4533i   2.2277 - 1.0691i   1.9463 - 1.6848i   1.2506 + 2.3006i

>> funm(A,'sin')

ans =

    0.5199   -0.2036   -0.1520    0.3648
   -0.0486    0.2615    0.2098    0.1064
    0.1581    0.0548    0.0031    0.3131
   -0.1003    0.4165    0.4682   -0.2553
```

Valores propios, vectores propios y descomposición de matrices

El trabajo con valores propios y vectores propios, así como los métodos de descomposición de matrices, son esenciales para el trabajo en álgebra matricial numérica. El módulo básico de MATLAB permite trabajar con estas materias mediante varias funciones. Las más importantes se exponen a continuación.

eig(A)	*Halla los autovalores de la matriz cuadrada A*
[V,D] = eig(A)	*Halla la matriz diagonal D de autovalores de A y una matriz V cuyas columnas son los autovectores correspondientes, cumpliéndose que A*V=V*D*
eig(A,B)	*Devuelve un vector con los autovalores generalizados de las matrices cuadradas A y B. Los autovalores generalizados de A y B son las raíces del polinomio en λ det(λ*C - A)*
[V,D] = eig(A,B)	*Halla la matriz diagonal D de autovalores generalizados de A y B y una matriz V cuyas columnas son los autovectores correspondientes, cumpliéndose A*V=B*V*D*
[AA, BB, Q, Z, V] = qz(A,B)	*Calcula las matrices triangulares superiores AA y BB y las matrices Q y Z tales que Q*A*Z = AA y Q*B*Z = BB, y da la matriz V de autovectores generalizados de A y B. Los autovalores generalizados son los elementos de la diagonal de AA y BB, de tal modo que se tiene la igualdad siguiente: A*V*diag(BB) = B*V*diag(AA)*
[T,B] = balance(A)	*Encuentra una matriz de transformación T tal que B = T\A*T tiene autovalores aproximados a los de A. La matriz B se llama matriz balanceada de la matriz A*

balance(A)	*Calcula la matriz B balanceada de la matriz A. Su uso esencial es aproximar los autovalores de A cuando son difíciles de calcular. Se tiene que eig(A)=eig(balance(A))*
[V,D] = cdf2rdf(V,D)	*Transforma la salida compleja [V,D] del comando eig a forma real. Cada autovalor complejo en la diagonal de D origina una submatriz 2x2 en la forma real de la matriz D*
[U,T] = schur(A)	*Halla una matriz T y una matriz unitaria U tales que A = U*T*U' and U'*U = eye(U). Si A es compleja, T es una matriz triangular superior con los autovalores de A en la diagonal. Si A es real, la matriz T tiene los autovalores de A en la diagonal, y los correspondientes autovalores complejos se corresponden con los bloques diagonales 2x2 de la matriz T*
schur(A)	*Devuelve la matriz T del apartado anterior solamente*
[U,T] = rsf2csf(U,T)	*Convierte a compleja la salida [U,T] del comando schur*
[P,H] = hess(A)	*Devuelve la matriz unitaria P y la matriz de Hessenberg H tales que A = P*H*P' y P'*P =eye(size(P))*
[L,U] = lu(A)	*Descompone la matriz A en el producto A=L*U (descomposición LU de A), siendo U una matriz triangular superior y L una matriz pseudotriangular inferior (triangularizable mediante permutación)*
[L,U,P] = lu(A)	*Da una matriz triangular inferior L, una matriz triangular superior U y una matriz de permutación P tales que P*A=L*U*
R = chol(A)	*Devuelve la matriz triangular superior R tal que R'*R=A (descomposición de Cholesky de A), en caso de que A sea definida positiva. Si A no es definida positiva devuelve un error*
[Q,R] = qr(A)	*Devuelve la matriz triangular superior R de la misma dimensión que A, y la matriz ortogonal Q tales que A=Q*R (descomposición QR de A). Esta descomposición puede aplicarse a matrices no cuadradas*
[Q,R,E] = qr(A)	*Devuelve la matriz triangular superior R de la misma dimensión que A, la matriz permutación E y la matriz ortogonal Q tales que A*E=Q*R*
jordan(A) **[V,J]=jordan(A)**	*Halla la matriz canónica de Jordan J de la matriz A (J tiene los autovalores de A en la diagonal)* *Halla la matriz canónica de Jordan J de la matriz A y la matriz de paso V cuyas columnas son los autovectores de A cumpliéndose que $V^{-1}*A*V=J$*
condeig(A)	*Vector con los números de condición para los autovalores de A*
[V,D,s]=condeig(A)	*Equivale a [V,D]=eig(A) y s=condeig(A)*
[U,V,X,C,S]= gsvd(A,B) **[U,V,X,C,S]= gsvd(A,B,0)**	*Da las matrices unitarias U y V, la matriz cuadrada X y las matrices diagonales no negativas C y S tales que A = U*C*X', B = V*S*X' y C'*C + S'*S = I. A es (m,p), B es (n,p), U es (M,M), V es (n,n), X es (p,q) y q=min(m+n,p)* *Resultado anterior cuando m o n≥p. U y V tienen a lo sumo p columnas y C y S tienen a lo sumo p filas*

sigma = gsvd(A,B)	*Devuelve el vector de valores singulares generalizados sqrt(diag(C'*C)./diag(S'*S)).*
X = pinv(A)	*Devuelve la matriz X (pseudoinversa de A), de la misma dimensión que A' tal que A*X*A=A y X*A*X=X, siendo A*X y X*A matrices hermitianas*
hess(A)	*Devuelve la matriz de Hessenberg H*
poly(A)	*Devuelve el polinomio característico de la matriz A*
poly(V)	*Devuelve un vector cuyas componentes son los coeficientes del polinomio cuyas raíces son los elementos del vector V*
vander(C)	*Devuelve la matriz de Vandermonde A tal que su j-ésima columna es A(:,j) = C^{n-j}*

Como primer ejemplo consideramos la matriz aleatoria normal cuadrada de orden 3 y calculamos la matriz diagonal *D* con los autovalores de *A* y la matriz *V* cuyas columnas son sus autovectores, y si la salida es compleja se transforma a real. Hallamos también su matriz balanceada y las formas real y compleja de su descomposición de Schur. Por último encontramos los coeficientes del polinomio característico de la matriz *A*.

```
>> A=randn(3)

A =

   -0.4326    0.2877    1.1892
   -1.6656   -1.1465   -0.0376
    0.1253    1.1909    0.3273

>> [V,D] = eig(A)

V =

   0.2827             0.4094 - 0.3992i   0.4094 + 0.3992i
   0.8191            -0.0950 + 0.5569i  -0.0950 - 0.5569i
  -0.4991             0.5948             0.5948

D =

  -1.6984                  0                  0
        0            0.2233 + 1.0309i         0
        0                  0            0.2233 - 1.0309i

>> [V,D] = cdf2rdf(V,D)

V =

    0.2827    0.4094   -0.3992
    0.8191   -0.0950    0.5569
   -0.4991    0.5948         0
```

```
D =

   -1.6984         0         0
         0    0.2233    1.0309
         0   -1.0309    0.2233

>> [T,B] = balance(A)

T =

     1     0     0
     0     1     0
     0     0     1

B =

   -0.4326    0.2877    1.1892
   -1.6656   -1.1465   -0.0376
    0.1253    1.1909    0.3273

>> [U,T] = schur(A)

U =

    0.2827    0.2924    0.9136
    0.8191   -0.5691   -0.0713
   -0.4991   -0.7685    0.4004

T =

   -1.6984    0.2644   -1.2548
         0    0.2233    0.7223
         0   -1.4713    0.2233

>> [U,T] = rsf2csf(U,T)

U =

   0.2827            -0.7482 + 0.1678i   0.2395 - 0.5242i
   0.8191             0.0584 - 0.3266i  -0.4661 + 0.0409i
  -0.4991            -0.3279 - 0.4410i  -0.6294 - 0.2298i

T =

  -1.6984             1.0277 + 0.1517i   0.2165 + 0.7201i
        0             0.2233 + 1.0309i   0.7490 - 0.0000i
        0                  0             0.2233 - 1.0309i
```

```
>> poly(A)

ans =

    1.0000    1.2517    0.3540    1.8895
```

En el ejemplo siguiente se calcula la matriz triangular superior *R* de la misma dimensión que la matriz *A* del ejemplo anterior, la matriz permutación *E* y la matriz ortogonal *Q* tales que *A*E=Q*R* y se comprueba el resultado.

```
>> [Q,R,E] = qr(A)

Q =

   -0.2507    0.4556   -0.8542
   -0.9653   -0.0514    0.2559
    0.0726    0.8887    0.4527

R =

    1.7254    1.1211   -0.2380
         0    1.2484    0.8346
         0         0   -0.8772

E =

     1     0     0
     0     1     0
     0     0     1

>> A*E

ans =

   -0.4326    0.2877    1.1892
   -1.6656   -1.1465   -0.0376
    0.1253    1.1909    0.3273

>> Q*R

ans =

   -0.4326    0.2877    1.1892
   -1.6656   -1.1465   -0.0376
    0.1253    1.1909    0.3273
```

Se observa que se cumple *A*E=Q*R*.

Ahora consideramos la matriz *B* de Hessenberg de *A* y calculamos la matriz diagonal *D* de autovalores generalizados de *A* y *B*, y una matriz *V* cuyas columnas son los autovectores correspondientes, cumpliéndose *A*V=B*V*D*. Asimismo calculamos el vector de valores singulares generalizados de *A* y *B*.

```
>> B=hess(A)

B =

   -0.4326   -0.1976    1.2074
    1.6703   -1.2245    0.1544
         0   -1.0741    0.4053

>> [V,D] = eig(A,B)

V =

    0.0567    1.0000    1.0000
   -0.0376   -0.4998    0.5297
   -1.0000    0.4172    0.3785

D =

    1.0000         0         0
         0   -0.4722         0
         0         0   -2.1176

>> A*V

ans =

   -1.2245   -0.0803    0.1699
   -0.0137   -1.1082   -2.2872
   -0.3649   -0.3334    0.8801

>> B*V*D

ans =

   -1.2245   -0.0803    0.1699
   -0.0137   -1.1082   -2.2872
   -0.3649   -0.3334    0.8801

>> sigma = gsvd(A,B)

sigma =

    0.2874
    1.0000
    3.4799
```

En el ejemplo siguiente consideramos la matriz cuadrada 3x3 cuyas filas son los vectores (1,5,-2) (-7,3,1) y (2,2,-2) y realizamos las descomposiciones de Schur, LU, QR, Cholesky, Hessenberg y valores singulares comprobando que los resultados son correctos. Se halla también la matriz pseudoinversa de *A*.

En primer lugar, hallamos la descomposición de Schur, comprobando que el resultado es correcto.

```
>> A=[1,5,-2;-7,3,1;2,2,-2];
>> [U,T] = schur(A)

U =

   -0.0530   -0.8892   -0.4544
   -0.9910   -0.0093    0.1337
    0.1231   -0.4573    0.8807

T =

    2.4475   -5.7952   -4.6361
    5.7628    0.3689    2.4332
         0         0   -0.8163
```

Ahora comprobamos que *U*T*U'=A* y que *U*U'=eye*(3):

```
>> [U*T*U', U*U']

ans =

    1.0000    5.0000   -2.0000            1.0000    0.0000    0.0000
   -7.0000    3.0000    1.0000            0.0000    1.0000    0.0000
    2.0000    2.0000   -2.0000            0.0000    0.0000    1.0000
```

Ahora hallamos las descomposiciones LU, QR, Cholesky, Hesenberg y valores singulares, comprobando los resultados para cada caso:

```
» [L,U,P] = lu(A)

L =

    1.0000         0         0
   -0.1429    1.0000         0     Matriz triangular inferior
   -0.2857    0.5263    1.0000

U =

   -7.0000    3.0000    1.0000
         0    5.4286   -1.8571     Matriz triangular superior
         0         0   -0.7368
```

```
P =

     0     1     0
     1     0     0
     0     0     1

>> [P*A, L*U]

ans =

    -7     3     1        -7     3     1
     1     5    -2         1     5    -2   Tenemos que P*A=L*U
     2     2    -2         2     2    -2

>> [Q,R,E] = qr(A)

Q =

   -0.1361   -0.8785   -0.4579
    0.9526   -0.2430    0.1831
   -0.2722   -0.4112    0.8700

R =

   -7.3485    1.6330    1.7691
         0   -5.9442    2.3366    Matriz triangular superior
         0         0   -0.6410

E =

     1     0     0
     0     1     0
     0     0     1

>> [A*E,Q*R]

ans =

    1.0000   5.0000  -2.0000      1.0000    5.0000   -2.0000
   -7.0000   3.0000   1.0000     -7.0000    3.0000    1.0000
    2.0000   2.0000  -2.0000      2.0000    2.0000   -2.0000

      Luego, A*E=Q*R.

>> R = chol(A)

??? Error using ==> chol
Matrix must be positive definite.
```

Se obtiene mensaje de error porque la matriz no es definida positiva.

```
>> [P,H] = hess(A)

P =

    1.0000         0         0
         0   -0.9615    0.2747
         0    0.2747    0.9615

H =

    1.0000   -5.3571   -0.5494
    7.2801    1.8302   -2.0943
         0   -3.0943   -0.8302

>> [P*H*P', P'*P]

ans =

    1.0000    5.0000   -2.0000    1.0000         0         0
   -7.0000    3.0000    1.0000         0    1.0000         0
    2.0000    2.0000   -2.0000         0         0    1.0000
```

Luego $PHP'=A$ y $P'P$=I.

```
>> [U,S,V]=svd(A)

U =

   -0.1034   -0.8623    0.4957
   -0.9808    0.0056   -0.1949
    0.1653   -0.5064   -0.8463

S =

    7.8306         0         0
         0    6.2735         0     Matriz diagonal
         0         0    0.5700

V =

    0.9058   -0.3051    0.2940
   -0.3996   -0.8460    0.3530
   -0.1411    0.4372    0.8882

>> U*S*V'

ans =

    1.0000    5.0000   -2.0000
```

```
   -7.0000    3.0000    1.0000    Vemos que USV'=A
    2.0000    2.0000   -2.0000
```

Ahora calculamos la matriz pseudoinversa de *A*:

```
>> X = pinv(A)

X =

    0.2857   -0.2143   -0.3929
    0.4286   -0.0714   -0.4643
    0.7143   -0.2857   -1.3571

>> [A*X*A, X*A*X]

ans =

    1.0000    5.0000   -2.0000    0.2857   -0.2143   -0.3929
   -7.0000    3.0000    1.0000    0.4286   -0.0714   -0.4643
    2.0000    2.0000   -2.0000    0.7143   -0.2857   -1.3571
```

Luego, se cumple que *AXA=A* y *XAX=X*.

Matrices dispersas y especiales

Se denominan matrices dispersas aquellas que poseen un número importante de ceros de modo que sea posible trabajar con ellas de forma ventajosa aprovechando su reducido número de elementos no nulos. El módulo básico de MATLAB presenta varios comandos para trabajar eficientemente con matrices dispersas. Por otro lado, existen en MATLAB comandos que permiten trabajar con determinados tipos específicos de matrices especiales como las de Hadamard, Hankel, Hilbert, Pascal, Rosser, Toeplitz y otras. A continuación se presentan estos comandos:

S = sparse(i,j,s,m,n,nzmax) i, j y s vectores	*Crea una matriz dispersa S de dimensión mxn con espacio para nzmax elementos no nulos. El vector i contiene las componentes i-ésimas de los elementos no nulos de S, el vector j contiene sus correspondientes componentes j-ésimas y el vector s contiene los valores*
S=sparse(i,j,s,m,n)	*Crea la matriz dispersa S usando nzmax=length(s)*
S=sparse(i,j,s)	*Crea la matriz dispersa S con m=max(i) y n=max(j)*
S=sparse(A)	*Convierte la matriz completa A a la dispersa S*
A=full(S)	*Convierte la matriz dispersa S en la completa A*
S=spconvert(D)	*Convierte un fichero ASCII exterior leído con nombre D a la matriz dispersa S*
(i,j)=find(A)	*Devuelve los índices de filas y columnas no cero de la matriz A*
B=spdiags(A,d)	*Construye la matriz dispersa deducida de A cuya diagonal son de los elementos del vector d*
S=speye(m,n)	*Construye la matriz dispersa identidad mxn*

S=speye(n)	*Construye la matriz dispersa identidad cuadrada de orden n*
R=sprandn(S)	*Genera una matriz dispersa aleatoria de valores normales (0,1) con la misma estructura de la matriz dispersa S*
R=sprandsym(S)	*Genera una matriz simétrica aleatoria de valores normales (0,1) cuyo triángulo inferior y diagonal tienen la misma estructura que en S*
r=sprank(S)	*Da el rango estructural de la matriz dispersa S*
n=nnz(S)	*Da el número de elementos no nulos de la matriz dispersa S*
V=nonzeros(S)	*Da un vector columna completo de elementos no nulos en A ordenados por columnas*
k=nzmax(S)	*Da la cantidad de almacenamiento ocupada por los elementos no nulos de la matriz dispersa S. Se tiene que nzmax(S)=prod(size(S))*
s=spalloc(m,n, nzmax)	*Crea espacio en memoria para una matriz dispersa de dimensión mxn*
R=spones(S)	*Reemplaza los elementos no nulos de la matriz dispersa S con unos.*
n=condest(S)	*Devuelve la 1-norma de la matriz dispersa S*
m=normest(S)	*Devuelve la 2-norma de la matriz dispersa S*
issparse(A)	*Devuelve 1 si la matriz A es dispersa, y 0 en otro caso*
spfun('función',S)	*Aplica la función sólo a los elementos no nulos de S*
spy(S)	*Grafica el patrón de dispersión de la matriz S*
colamd(S)	*Vector que aproxima la permutación columna de mínimo grado de la matriz dispersa S*
colmmd(S)	*Permutación columna de mínimo grado de S*
symamd(S)	*Permutación aproximada simétrica de mínimo grado*
symmmd(S)	*Permutación simétrica de mínimo grado*
symrcm(S)	*Permutación simétrica inversa de Cuthill-McKee*
colperm(S)	*Permutación columna*
randperm(n)	*Permutación aleatoria del vector (1:n)*
dmperm(A)	*Permutación Dulmage-Mendelsohn de la matriz reducible A*
eigs(A)	*Vector con los seis mayores autovalores de A*
eigs(A,B)	*Resuelve la ecuación A*V == B*V*D con B simétrica*
eigs(A,k)	*Vector con los k mayores autovalores de A*
eigs(A,B,k)	*Devuelve las k mayores soluciones de A*V == B*V*D*
eigs(A,k,σ)	*Vector con los k mayores autovalores de A basados en σ, donde σ puede valer 'lm' (mayor magnitud), 'sm' (menor magnitud), 'lr' (mayor parte real), 'sr' (menor parte real), 'li' (mayor parte imaginaria) y 'si' (menor parte imaginaria)*
eigs(A,B,k, σ)	*Da las k mayores soluciones de A*V == B*V*D según σ*
[V,D] = eigs(A,...)	*Da la matriz diagonal D de autovalores de A y la matriz V cuyas columnas son los respectivos autovectores*
svds(A)	*Da los 5 mayores valores singulares de A*
svds(A,k)	*Da los k mayores valores singulares de A*

svds(A,k,0) **[U,S,V] = svds(A,...)**	*Da los k mayores valores singulares de A usando eigs* *Da las matrices U(m,k) con columnas ortonormales, S(k,k) diagonal y V(n,k) con columnas ortonormales*
[L,U,P] = luinc(A)	*Factorización LU incompleta de A*
cholinc(A)	*Factorización incompleta de Cholesky de A*
luinc(A)	*Factorización LU incompleta de A*
H=hadamard(n)	*Matriz con valores 1 o -1 tal que H'*H = n*eye(n)*
hankel(V)	*Matriz cuya primera columna es el vector V y cuyos elementos son cero por debajo de la primera antidiagonal. La matriz Hankel(C,R) tiene como primera columna el vector C y como última fila el vector R*
hilb(n)	*Matriz de Hilbert de orden n tal que Aij=1/(i+j-1)*
invhilb(n)	*Inversa de la matriz de Hilbert de orden n*
magic(n)	*Matriz mágica de orden n. Sus elementos son los enteros desde 1 hasta n^2 con iguales sumas de filas y columnas*
pascal(n)	*Matriz de Pascal de orden n (simétrica, definida positiva y basada en el triángulo de Pascal)*
rosser	*Matriz 8x8 con un autovalor doble, otro nulo, tres casi iguales, otro muy pequeño y dos opuestos*
toeplitz(C,R)	*Matriz de Toeplitz (no simétrica con el vector C de primera columna y el vector R como primera fila)*
vander(C)	*Matriz de Vandermonde cuya penúltima columna es el vector C. Además, A(:,j) = C^{n-j}*
wilkinson(n)	*Matriz de Wilkinson (simétrica tridiagonal con pares de autovalores cercanos pero no iguales)*
compan(P)	*Matriz del polinomio de coeficientes P*

Como primer ejemplo se crea una matriz dispersa cuadrada de orden 4 cuyos términos (2,3), (1,4), (4,1), (3,2), (3,1), (4,2) y (4,3) sean -4, 9, 12, -3, 6, 1 y 15, respectivamente, y se presenta dicha matriz de forma completa.

```
>> S=sparse([2,1,4,3,3,4,4],[3,4,1,2,1,2,3],[-4,9,12,-3,6,1,15])

S =
   (3,1)         6
   (4,1)        12
   (3,2)        -3
   (4,2)         1
   (2,3)        -4
   (4,3)        15
   (1,4)         9

>> full(S)

ans =
     0     0     0     9
     0     0    -4     0
     6    -3     0     0
    12     1    15     0
```

A continuación se halla la norma, condición, rango estructural y número de elementos no nulos de la matriz *S*, así como la matriz dispersa cuyos elementos no nulos son las raíces cuadradas de los elementos de *S*.

```
>> m=normest(S)
m =
   19.8297
>> n=condest(S)
n =
   20.0179
>> r=sprank(S)
r =
     4
>> n=nnz(S)
n =
     7
>> spfun('sqrt',S)
ans =
   (3,1)       2.4495
   (4,1)       3.4641
   (3,2)            0 + 1.7321i
   (4,2)       1.0000
   (2,3)            0 + 2.0000i
   (4,3)       3.8730
   (1,4)       3.0000
>> full(ans)
ans =
        0                   0                   0              3.0000
        0                   0                    0 + 2.0000i         0
   2.4495                   0 + 1.7321i         0                   0
   3.4641              1.0000              3.8730                   0
```

Seguidamente se calculan los 6 mayores valores propios de las matrices mágica, de Pascal, de Hadamard y de Hilbert de orden 16, respectivamente.

```
>> format long
>> M=magic(16);
>> P=pascal(16);
>> H=hadamard(16);
>> HI=hilb(16);
>> [eigs(M),eigs(P),eigs(H),eigs(HI)]

ans =

  1.0e+008 *

  0.00002056000000             2.06096881237987      0.00000004      0.00000001860036
  0.00000295025423             0.03091667679027      0.00000004      0.00000000440131
 -0.00000295025423             0.00100814646024      0.00000004      0.00000000061114
 -0.00000000000000             0.00005409112411     -0.00000004      0.00000000006298
  0.00000-0.00000i             0.00000429846732     -0.00000004      0.00000000000515
  0.00000+0.00000i             0.00000048334533     -0.00000004      0.00000000000034
```

6.2 Soluciones de ecuaciones y sistemas

MATLAB permite resolver ecuaciones y sistemas utilizando los comandos que se presentan a continuación:

solve('ecuación','x')	*Resuelve la ecuación en la variable x*
syms x ; solve(ecu(x),x)	*Resuelve la ecuación ecu(x) en la variable x*
solve('ec1,ec2,...,ecn', 'x1, x2,...,xn')	*Resuelve n ecuaciones simultáneas ec1,...,ecn (sistema en las variables x1,...,xn)*
X = linsolve(A,B)	*Resuelve A*X=B para una matriz cuadrada A, siendo B y X matrices*
x = nnls(A,b)	*Resuelve A*x=b en el sentido de mínimos cuadrados, siendo x un vector (x≥0)*
x = lscov(A,b,V)	*Da x (vector) que minimiza (A*x-b)'*inv(V)*(A*x- b)*
roots(V)	*Da las raíces del polinomio cuyos coeficientes son las componentes del vector V*
X = A\B	*Resuelve el sistema A*X=B*
X = A/B	*Resuelve el sistema X*A=B*
roots(A)	*Da las raíces del polinomio cuyos coeficientes construyen el vector A*
poly(V)	*Da el polinomio cuyas raíces son el vector V*
x = lscov(A,b,V) **[x,dx] = lscov(A,b,V)**	*Da el vector x tal que Ax=b+e con e→N(0,V)* *Da además el error estándar de x (dx)*
x = bicg(A,b) **bicg(A,b,tol)** **bicg(A,b,tol,maxit)**	*Intenta resolver el sistema Ax=b por el método de los gradientes biconjugados* *Resuelve Ax=b especificando la tolerancia* *Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*

bicg(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
bicg(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
bicg(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,f] = bicg(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
x = bicgstab(A,b)	*Intenta resolver el sistema Ax=b por el método de los gradientes biconjugados estabilizado*
bicgstab(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
bicgstab(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
bicgstab(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
bicgstab(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
bicgstab(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,f] = bicgstab(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,f,relres] = bicgstab(A,b,...)	*Devuelve también el residuo relativo norm(b-A*x)/norm(b)*
[x,f,relres,iter] = bicgstab(A,b,...)	*Devuelve también el número de iteraciones*
x = cqs(A,b)	*Intenta resolver el sistema Ax=b por el método de los gradientes conjugados cuadráticos*
cqs(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
cqs(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
cqs(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
cqs(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
cqs(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,f] = cqs(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,f,relres] = cqs(A,b,...)	*Devuelve también el residuo relativo norm(b-A*x)/norm(b)*
[x,f,relres,iter] = cqs(A,b,...)	*Devuelve también el número de iteraciones*

x = pcg(A,b)	*Intenta resolver el sistema Ax=b por el método del gradiente conjugado precondicionado*
pcg(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
pcg(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
pcg(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
pcg(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
pcg(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,f] = pcg(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,f,relres] = pcg(A,b,...)	*Devuelve también el residuo relativo norm(b-A*x)/norm(b)*
[x,f,relres,iter] = pcg(A,b,...)	*Devuelve también el número de iteraciones*
x = qmr(A,b)	*Intenta resolver el sistema Ax=b por el método residual cuasiminimal*
qmr(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
qmr(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
qmr(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
qmr(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
qmr(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,f] = qmr(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,f,relres] = qmr(A,b,...)	*Da también el residuo relativo norm(b-A*x)/norm(b)*
[x,f,relres,iter] = qmr(A,b,...)	*Devuelve también el número de iteraciones*
x = gmres(A,b)	*Intenta resolver el sistema Ax=b por el método residual mínimo generalizado*
gmres(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
gmres(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
gmres(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
gmres(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
gmres(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*

[x,f] = gmres(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,f,relres] = gmres(A,b,...)	*Devuelve también el residuo relativo norm(b-A*x)/norm(b)*
[x,f,relres,iter] = gmres(A,b,...)	*Devuelve también el número de iteraciones*
x = lsqr(A,b)	*Intenta resolver el sistema Ax=b por el método del gradiente conjugado en ecuaciones normales*
lsqr(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
lsqr(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
lsqr(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
lsqr(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
lsqr(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,f] = lsqr(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,f,relres] = lsqr(A,b,...)	*Devuelve también el residuo relativo norm(b-A*x)/norm(b)*
[x,f,relres,iter] = lsqr(A,b,...)	*Devuelve también el número de iteraciones*
x = minres(A,b)	*Intenta resolver el sistema Ax=b por el método residual mínimo*
minres(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
minres(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
minres(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
minres(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
minres(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,f] = minres(A,b,...)	*Resuelve el sistema y f indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,f,relres] = minres(A,b,...)	*Devuelve también el residuo relativo norm(b-A*x)/norm(b)*
[x,f,relres,iter] = minres(A,b,...)	*Devuelve también el número de iteraciones*
x = symmlq(A,b)	*Intenta resolver el sistema Ax=b por el método LQ simétrico*

symmlq(A,b,tol)	*Resuelve Ax=b especificando la tolerancia*
symmlq(A,b,tol,maxit)	*Resuelve Ax=b especificando la tolerancia y el número máximo de iteraciones*
symmlq(A,b,tol,maxit,M)	*Resuelve el sistema inv(M)*A*x = inv(M)*b*
symmlq(A,b,tol,maxit,M1,M2)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2*
symmlq(A,b,tol,maxit,M1,M2,x0)	*Resuelve el sistema inv(M)*A*x = inv(M)*b con M=M1*M2 y valor inicial x0*
[x,flag] = symmlq(A,b,...)	*Resuelve el sistema e indica el resultado (0=convergencia, 1=no convergencia, 2=convergencia condicionada, 3=estancamiento y 4=números muy extremos)*
[x,flag,relres] = symmlq(A,b,...)	*Devuelve también el residuo relativo norm(b-A*x)/norm(b)*
[x,flag,relres,iter] = symmlq(A,b,...)	*Devuelve también el número de iteraciones*
x = lsqnonneg(C,d)	*Da x que minimiza norm(C*x-d) sujeto a x≥0 por mínimos cuadrados (C y d reales)*
x = lsqnonneg(C,d,x0)	*Da x que minimiza norm(C*x-d) sujeto a x≥0, pero con x=x0≥0 como valor inicial*
x = lsqnonneg(C,d,x0,opt)	*Da x que minimiza norm(C*x-d) sujeto a x≥0, con x=x0≥0 como valor inicial y con la opción opt. Las opciones son TolX para la tolerancia y Display para mostrar la salida ('off' no muestra la salida, 'final' muestra el final de la salida y 'notify' muestra la salida sólo si no hay convergencia)*
[x,resnorm] = lsqnonneg(...)	*Da la solución y el valor de la 2-norma cuadrada del residuo norm(C*x-d)^2*
[x,resnorm,residual] = lsqnonneg(...)	*Da la solución y el residuo C*x-d*
[x,resnorm,residual,f] = lsqnonneg(...)	*Da la solución , el residuo C*x-d y un valor f positivo o nulo según converja o no la solución*
[x,resnorm,residual,f,out,lambda] = lsqnonneg(...)	*Da la solución, el residuo C*x-d, el valor f , el algoritmo usado y el vector de los multiplicadores de Lagrange lambda*
x=fzero(función,x0)	*Halla un cero de la función cerca de x0*
[x, feval]=fzero(fun,x0)	*Da también el valor de la función objetivo en x*
[x, feval,f]=fzero(fun,x0)	*Si f>0 hay en x y en caso contrario no*
S = spaugment(A,c)	*Crea la matriz dispersa S = [c*I A; A' 0]. Se tiene que r = b - A*x , S * [r/c; x] = [b; 0] para el problema de mínimos cuadrados: min norm(b - A*x)*

Como primer ejemplo hallamos las raíces de la ecuación $2x^3+11x^2+12x-9=0$. Como se trata de un polinomio utilizamos *roots* como sigue:

```
>> roots([2, 11, 12, -9])

ans =

   -3.0000
   -3.0000
    0.5000
```

La ecuación anterior también puede resolverse de la siguiente forma:

```
>> solve('2*x^3+11*x^2+12*x-9','x')

ans =

[ 1/2]
[  -3]
[  -3]
```

La ecuación $xsen(x) = 1/2$ puede resolverse en entornos de 2, 4 y 6 como sigue:

```
>>[fzero('x*sin(x)-1/2',2),fzero('x*sin(x)-1/2',4),fzero('x*sin(x)-1/2',6)]

ans =

    0.7408    2.9726    6.3619
```

El sistema de ecuaciones $x+y+z=1$, $3x+y=3$, $x-2y-z=0$ puede resolverse de la forma siguiente:

```
>> [x,y,z]=solve('x+y+z=1', '3*x+y=3', 'x-2*y-z=0','x','y','z')

x =

4/5

y =

3/5

z =

-2/5
```

La sintaxis también podía haberse escrito de la siguiente forma:

```
>> [x,y,z]=solve('x+y+z=1, 3*x+y=3, x-2*y-z=0','x,y,z')

x =

4/5

y =

3/5

 z =

-2/5
```

También es posible utilizar la siguiente sintaxis:

```
>> A=[1,1,1;3,1,0;1,-2,-1]; B=[1,3,0]'; linsolve(A,B)

ans =

[  4/5]
[  3/5]
[ -2/5]
```

O incluso puede utilizarse la sintaxis siguiente:

```
>> A\B

ans =

    0.8000
    0.6000
   -0.4000
```

También puede resolverse el sistema por métodos aproximados (aunque en este caso no es necesario), como por ejemplo el método del gradiente conjugado en ecuaciones normales. La sintaxis será la siguiente:

```
>> lsqr(A,B)

lsqr stopped at iteration 3 without converging to the desired
tolerance 1e-006
because the maximum number of iterations was reached.
The iterate returned (number 3) has relative residual 0.084

ans =

    0.8558
    0.3542
   -0.0448
```

6.3 Espacios vectoriales y aplicaciones lineales

Es posible trabajar con el módulo básico de MATLAB en tareas relativas a espacios vectoriales, aplicaciones lineales, formas bilineales, formas cuadráticas, etc. La potencia del cálculo matricial permite el trabajo fluido en este campo. Los comandos más importantes relacionados con esta materia se presentan a continuación:

N= null(A) **N=null(A, 'r')**	*Genera una base ortonormal para el núcleo de A (de la aplicación lineal de matriz A) obtenida mediante la descomposición por valores singulares, y el número de columnas de N es la dimensión del núcleo de A (nulidad de A)* *Genera una base racional para el núcleo de A obtenida mediante el método de la reducida escalonada*
Q=orth(A)	*Genera una base ortonormal para el rango de A, es decir, que las columnas de Q generan el mismo espacio que las columnas de A, y el número de columnas de Q es el rango de A*
colspace(A)	*Devuelve una base para las columnas de A*
dot(A,B)	*Da el producto escalar de los vectores A y B*
cross(A,B)	*Da el producto vectorial de los vectores A y B*

Como primer ejemplo consideramos la matriz mágica de orden 4 y calculamos una base ortonormal y otra racional para su núcleo.

```
>> null(magic(4))

ans =

    0.2236
    0.6708
   -0.6708
   -0.2236

>> null(magic(4),'r')

ans =

    -1
    -3
     3
     1
```

A continuación calculamos una base ortonormal para las columnas de la matriz anterior.

```
>> colspace(sym(magic(4)))

ans =

[1,  0,  0]
[0,  1,  0]
[0,  0,  1]
[1,  3, -3]
```

En el ejemplo siguiente se halla una base ortonormal para el rango de la matriz anterior.

```
>> orth(magic(4))

ans =

   -0.5000    0.6708    0.5000
   -0.5000   -0.2236   -0.5000
   -0.5000    0.2236   -0.5000
   -0.5000   -0.6708    0.5000
```

En el ejemplo siguiente se calculan los productos escalar y vectorial de los vectores (1,1,-1) y (1,1,1).

```
>> dot([1,1,-1],[1,1,1])

ans =
     1

>> cross([1,1,-1],[1,1,1])

ans =
     2    -2     0
```

6.4 Trabajando con polinomios

MATLAB contiene comandos para realizar las operaciones estándar con polinomios, tales como búsqueda de sus raíces, evaluación, diferenciación, interpolación y ajuste. Los comandos (funciones) más importantes disponibles en el módulo básico de MATLAB para el trabajo con polinomios se presentan a continuación.

q=conv(u,v)	*Da los coeficientes del polinomio producto de los dos polinomios cuyos coeficientes vienen dados por los vectores u y v*
[q,r] = deconv(v,u)	*Da los polinomios cociente y resto de la división entre los polinomios u y v. Se tiene v = conv(u,q)+r*
p = poly(r)	*Da los coeficientes del polinomio p cuyas raíces son el vector r*
k = polyder(p) **k = polyder(a,b)** **[q,d] = polyder(b,a)**	*Da los coeficientes del polinomio k derivada del polinomio p* *Da los coeficientes de la derivada del producto de a y b* *Da el numerador q y el denominador d de la derivada de a/b*
p = polyfit(x,y,n) **[p,S] = polyfit(x,y,n)** **[p,S,u] = polyfit(x,y,n)**	*Polinomio de grado n que ajusta los puntos (x,y)* *Polinomio de grado n que ajusta los puntos (x,y) con varianza S* *Polinomio de grado n que ajusta los puntos (x,y) con varianza S y media u para el error del ajuste*
y = polyval(p,x) **y = polyval(p,x,[ ],u)** **[y,delta] = polyval(p,x,S)** **[y,delta] = polyval(p,x,S,u)**	*Evalúa el polinomio p en x* *Evalúa el polinomio p en x con media u para el error* *Evalúa el polinomio p en x con varianza S para el error y con un error y±delta en el resultado* *Evalúa el polinomio p en x con media u y varianza S para el error y con un error y±delta en el resultado*
Y = polyvalm(p,X)	*Evalúa el polinomio p en la variable matricial X*
[r,p,k] = residue(b,a) **[b,a] = residue(r,p,k)**	*Da residuos, polos y términos directos de la expansión racional de b/a* $\frac{b(s)}{a(s)} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \cdots + \frac{r_n}{s-p_n} + k(s)$ *Convierte la expansión racional a polinomios de coeficientes b y a*
r = roots(c)	*Da el vector columna r de raíces del polinomio con coeficientes c*

Como primer ejemplo calculamos las raíces del polinomio x^3-6x^2-72x-27.

```
>> p = [1 -6 -72 -27];r=roots(p)

r =

   12.1229
   -5.7345
   -0.3884
```

A continuación se evalúa el polinomio x^3-6x^2-72x-27 en la matriz de Pascal de orden 4 y en el escalar 10.

```
>> Y=polyval(p,X)

Y =
```

```
        -104        -104        -104        -104
        -104        -187        -270        -347
        -104        -270        -459        -347
        -104        -347        -347        4133

>> polyval(p,10)

ans =

  -347
```

En el ejemplo siguiente se considera un vector de puntos x igualmente espaciados en el intervalo [0, 2,5], se evalúa la función *erf(x)* en estos puntos y se calculan los coeficientes aproximados del polinomio de grado 6 que ajusta los puntos $(x, erf(x))$.

```
>>  x = (0: 0.1: 2.5)'; y = erf(x);p = polyfit(x,y,6)

p =

0.0084   -0.0983    0.4217   -0.7435    0.1471    1.1064    0.0004
```

A continuación se calcula la derivada del producto de polinomios dado por $(3x^2+6x+9)(x^2+2x)$ y también de su cociente.

```
>> a = [3 6 9];
b = [1 2 0];
k = polyder(a,b)

k =

    12    36    42    18
```

La derivada del producto es el polinomio $12x^3+36x^2+42x+18$

```
>> [q,d] = polyder(b,a)

q =

    18    18

d =

     9    36    90   108    81
```

La derivada del cociente es $\dfrac{18x+18}{9x^4+36x^3+90x^2+108x+81}$

6.5 Interpolación polinómica

MATLAB permite trabajar con una cantidad importante de técnicas de interpolación que permiten realizar ajustes rápidos que ocupan poca memoria. A continuación se presentan los comandos del módulo básico de MATLAB que permiten la interpolación:

Yi = interp1(X,Y,Xi)	*Da el vector Yi tal que (Xi,Yi) es el conjunto total de puntos hallados por interpolación unidimensional del conjunto de puntos dado (X,Y)*
Yi = interp1(Y,Xi)	*Supone que X=1:N, siendo N la longitud de Y*
Yi = interp1(X,Y,Xi,método)	*Realiza la interpolación mediante el método dado, que puede ser nearest (vecino más cercano), linear (lineal), cubic (cúbica de Hermite), v5cubic (cúbica de MATLAB 5), spline y pchip (cúbica de Hermite)*
Yi = interp1(X,Y,Xi,método,ext)	*Adicionalmente realiza extrapolación para el valor de x=ext externo al rango de variación de x usado para la interpolación*
Zi = interp2(X,Y,Z,Xi,Yi)	*Da el vector Zi tal que (Xi,Yi, Zi) es el conjunto total de puntos hallados por interpolación bidimensional del conjunto de puntos dados (X,Y,Z)*
Zi = interp2(Z,Xi,Yi)	*Supone que X=1:n e Y=1:m y (n,m) = tamaño de Z*
Zi = interp2(Z,n)	*Interpolación recursiva bidimensional n veces*
Zi = interp2(X,Y,Z,Xi,Yi,*method*)	*Interpolación mediante los métodos nearest (vecino más cercano), linear (lineal), cubic (cúbica de Hermite) y spline*
Vi = interp3(X,Y,Z,V,Xi,Yi,Zi)	*Da el vector Vi tal que (Xi,Yi, Zi) es el conjunto total de puntos hallados por interpolación tridimensional del conjunto de puntos resultante de aplicar la función tridimensional V a los puntos (X,Y,Z)*
Vi = interp3(V,Xi,Yi,Zi)	*Supone que X=1:n, Y=1:m, Z=1:p y (n,m,p) = tamaño de V*
Vi = interp3(V,n)	*Interpolación recursiva tridimensional n veces*
Vi = interp3(...,método)	*Interpolación mediante los métodos especificados*
Y = interpft(X,n)	*Interpolación unidimesional usando el método FFT. Da el vector Y que contiene los valores de la función periódica X muestreada en n puntos igualmente espaciados. El vector original X se transforma al dominio de frecuencias de Fourier mediante la transformada rápida de Fourier FFT*
y = interpft(x,n,dim)	*Opera a lo largo de la dimensión especificada*

Vi = interpn(X,Y,Z,…V,Xi,Yi,Zi…)	*Da el vector Vi tal que (Xi,Yi, Zi,…) es el conjunto total de puntos hallados por interpolación n-dimensional del conjunto de puntos resultante de aplicar la función tridimensional V a los puntos (X,Y,Z,…)*
Vi = interpn(V,Xi,Yi,Zi)	*Supone que X=1:n, Y=1:m, Z=1:p,... y (n,m,p,…) = tamaño de V*
Vi = interpn(V,n)	*Interpolación recursiva n-dimensional n veces*
Vi = interpn(...,método)	*Interpolación mediante los métodos especificados*
Yi = pchip(X,Y,Xi)	*Da el vector Yi tal que (Xi,Yi) es el conjunto total de puntos hallados por interpolación cúbica polinomial de Hermite a trozos (PCHIP) del conjunto de puntos dado (X,Y)*
pp = pchip(X,Y)	*Realiza interpolación en puntos intermedios*
Yi = spline(X,Y,Xi)	*Da el vector Yi tal que (Xi,Yi) es el conjunto total de puntos hallados por interpolación cúbica spline del conjunto de puntos dado (X,Y)*
pp = spline(X,Y)	*Realiza interpolación en puntos intermedios*
Zi = griddata(X,Y,Z,Xi,Yi)	*Ajusta una superficie de la forma Z=f(X,Y) en el espacio de los vectores (X,Y,Z). El vector Zi determina los puntos de interpolación (Xi,Yi,Zi) entre los puntos dados (X,Y,Z). Se utiliza un método de la distancia inversa para interpolar*
[Xi,Yi,Zi] = griddata(X,Y,Z,Xi,Yi)	*Además de Xi, da los vectores fila y columna Yi y Zi*
[...] = griddata(...,método)	*Interpolación mediante los métodos especificados*
W = griddata3(X,Y,Z,V,Xi,Yi,Zi)	*Ajusta una hipersuperficie de la forma W=f(X,Y,Z) en el espacio de los vectores (X,Y,Z,V). El vector W determina los puntos de interpolación (Xi,Yi,Zi,Vi) entre los puntos dados (X,Y,Z,V). Se utiliza un método de la distancia inversa para interpolar*
W = griddata3(...,'método')	*Interpolación mediante los métodos especificados*
Yi = griddatan(X,Y,Xi)	*Ajusta una superficie de dimensión mayor o igual que 3 de la forma Y=f(X)*
yi = griddatan(...,'método')	*Interpolación mediante los métodos especificados*
plot(X,Y,S)	*Grafica los puntos (X,Y) con las opciones definidas en S (dígitos entre comillas, el primero de los cuales fija el color de la línea del gráfico y el segundo fija el carácter a usar en el graficado)*
plot3(X,Y,S)	*Grafica los puntos (X,Y,Z) con las opciones definidas en S (similar al comando anterior)*
[X,Y] = meshgrid(x,y)	*Define la matriz de puntos de la superficie z=f(x,y)*
[X,Y,Z] = meshgrid(x,y,z)	*Define la matriz de puntos volumétrica w=f(x,y,z)*
mesh(x,y,z,c)	*Define malla y color para una superficie paramétrica*
slice(X,Y,Z,V,Xi,Yi,Zi)	*Dibuja cortes según las direcciones X, Y, Z en el volumen V a través de la superficie (Xi,Yi,Zi)*

Como primer ejemplo calculamos el polinomio interpolador de segundo grado que pasa por los puntos (-1,4), (0,2) y (1,6) en el sentido de mínimos cuadrados.

```
>> x=[-1,0,1];y=[4,2,6];p=polyfit(x,y,2)

p =
    3.0000    1.0000    2.0000
```

El polinomio interpolador pedido resulta ser $3x^2+x+2$.

En el ejemplo siguiente obtenemos 25 puntos de aproximación por interpolación de la función paramétrica $X=Cos(t)$, $Y=Sen(t)$, $Z=Tan(t)$ para valores de t entre 0 y $\pi/6$ igualmente espaciados, sobre el conjunto de puntos definido para valores de t en el intervalo (i, $\pi/6$) con $0 \leq i \leq 6$.

```
>> t=0:pi/150:pi/6;
>> x=cos(t);y=sin(t);z=tan(t) ;
>> xi=cos(t);yi=sin(t);
>> zi=griddata(x,y,z,xi,yi) ;
>> puntos=[xi',yi',zi']

puntos =
    1.0000         0         0
    0.9998    0.0209    0.0209
    0.9991    0.0419    0.0419
    0.9980    0.0628    0.0629
    0.9965    0.0837    0.0840
    0.9945    0.1045    0.1051
    0.9921    0.1253    0.1263
    0.9893    0.1461    0.1477
    0.9860    0.1668    0.1691
    0.9823    0.1874    0.1908
    0.9781    0.2079    0.2126
    0.9736    0.2284    0.2345
    0.9686    0.2487    0.2568
    0.9632    0.2689    0.2792
    0.9573    0.2890    0.3019
    0.9511    0.3090    0.3249
    0.9444    0.3289    0.3482
    0.9373    0.3486    0.3719
    0.9298    0.3681    0.3959
    0.9219    0.3875    0.4204
    0.9135    0.4067    0.4452
    0.9048    0.4258    0.4706
    0.8957    0.4446    0.4964
    0.8862    0.4633    0.5228
    0.8763    0.4818    0.5498
    0.8660    0.5000    0.5774
```

A continuación hallamos y representamos (Figura 6-1) 40 puntos de interpolación (x,y) según la función $y=Sen(x)$ para valores de x igualmente espaciados entre 0 y 10.

```
>> x = 0:10;
y = sin(x);
xi = 0:.25:10;
yi = interp1(x,y,xi);
plot(x,y,'o',xi,yi)
```

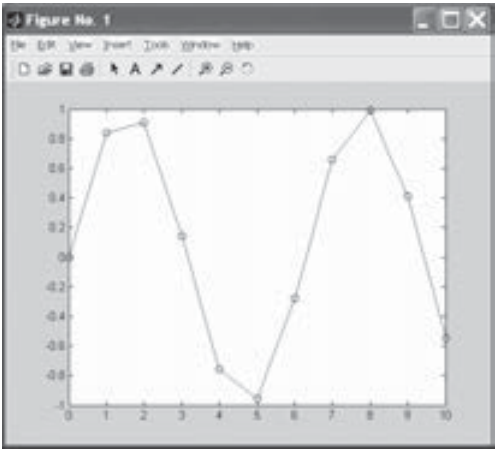


Figura 6-1

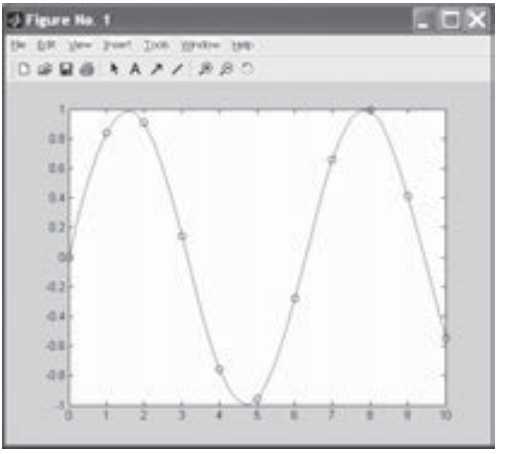


Figura 6-2

Seguidamente resolvemos el problema anterior mediante interpolación *spline* (Figura 6-2) y se observa que este tipo de interpolación es más fino.

```
>> x = 0:10;
y = sin(x);
xx = 0:.25:10;
yy = spline(x,y,xx);
plot(x,y,'o',xx,yy)
```

En el ejemplo siguiente consideramos una muestra de 100 puntos aleatorios *x, y, z* en el intervalo [-2,2] y graficamos la superficie de interpolación $z=F(x,y)$ sobre la malla regular definida por 16 puntos igualmente espaciados entre -2 y 2.

```
>> rand('seed',0)
x = rand(100,1)*4-2; y = rand(100,1)*4-2;
z = x.*exp(-x.^2-y.^2);

>> ti = -2:.25:2;
[XI,YI] = meshgrid(ti,ti);
ZI = griddata(x,y,z,XI,YI);
```

```
>> mesh(XI,YI,ZI), hold
plot3(x,y,z,'o'), hold off
Current plot held
```

Se obtiene la superficie de interpolación de la Figura 6-3.

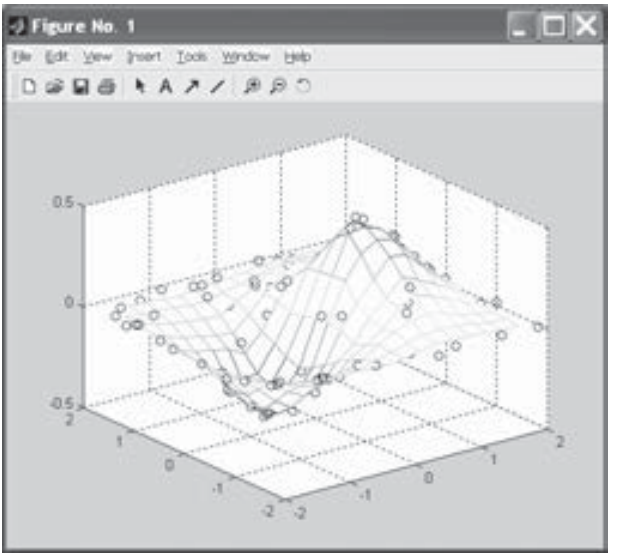


Figura 6-3

Ejercicio 6-1. Dada la matriz de Hilbert de orden 3 H calcular transpuesta, inversa, determinante, rango, traza, valores singulares, condición, norma, H^3, 3^H, e^H y Ln(H).

```
>> format rat
>> H=hilb(3)

H =

      1           1/2          1/3
     1/2          1/3          1/4
     1/3          1/4          1/5

>> transpuesta=transpose(H)

transpuesta =

      1           1/2          1/3
     1/2          1/3          1/4
     1/3          1/4          1/5

>> inversa=inv(H)
```

```
inversa =

      9          -36           30
    -36          192         -180
     30         -180          180

>> determinante=det(H)

determinante =

     1/2160

>> rango=rank(H)

rango =

      3

>> vsingulares=svd(H)

vsingulares =

   745/529
   389/3180
    26/9675

>> condicion=cond(H)

condicion =

 27775/53

>> normas=[norm(H),norm(H,1),norm(H,inf),norm(H,'fro')]

normas =

   745/529       11/6         11/6        581/411

>> H^3

ans =

    86/45      1529/1440     344/461
  1529/1440    1277/2160    1951/4691
   344/461     1951/4691     728/2487

>> 3^H
```

```
ans =

  1497/419     1059/776      575/613
  1059/776     3011/1651    3585/5998
   575/613     3585/5998     943/651

>> expm(H)

ans =

  1431/454     6250/5497    1464/1877
  6250/5497     766/453      295/587
  1464/1877     295/587     1088/789

>> logm(H)

ans =

 -1589/3230     751/588      473/1489
   751/588   -17033/4826    2417/1079
   473/1489    2417/1079    -479/131
```

Ejercicio 6-2. Dada la matriz siguiente:

$$\begin{pmatrix} 2/3 & 2/5 & 2/7 & 2/9 & 2/11 \\ 2/5 & 2/7 & 2/9 & 2/11 & 2/13 \\ 2/7 & 2/9 & 2/11 & 2/13 & 2/15 \\ 2/9 & 2/11 & 2/13 & 2/15 & 2/17 \\ 2/11 & 2/13 & 2/15 & 2/17 & 2/19 \end{pmatrix}$$

hallar sus autovalores, autovectores, polinomio característico y números de condición de los autovalores. Diagonalizar la matriz simétrica A=[3,-1,0;-1,2,-1;0,-1,3] y calcular una matriz semejante a la dada.

```
>> M=[2/3,  2/5,  2/7,  2/9, 2/11; 2/5, 2/7,  2/9, 2/11, 2/13; 2/7,
2/9,  2/11,  2/13,  2/15; 2/9,  2/11,  2/13,  2/15,  2/17;  2/11,  2/13,
2/15, 2/17, 2/19]

M =

     2/3          2/5          2/7          2/9          2/11
     2/5          2/7          2/9          2/11         2/13
     2/7          2/9          2/11         2/13         2/15
     2/9          2/11         2/13         2/15         2/17
     2/11         2/13         2/15         2/17         2/19
```

```
>> format short
>> [V,E]=eig(M)

V =

   -0.0102    0.0697    0.2756   -0.6523    0.7026
    0.1430   -0.4815   -0.7052    0.1593    0.4744
   -0.5396    0.6251   -0.2064    0.3790    0.3629
    0.7526    0.2922    0.2523    0.4442    0.2954
   -0.3490   -0.5359    0.5661    0.4563    0.2496

E =

    0.0000         0         0         0         0
         0    0.0001         0         0         0
         0         0    0.0059         0         0
         0         0         0    0.1244         0
         0         0         0         0    1.2423
```

Los autovectores son las columnas de la matriz V y los autovalores son los elementos de la diagonal de la matriz E.

```
>> poly(M)

ans =

    1.0000   -1.3728    0.1628   -0.0009    0.0000   -0.0000
```

El polinomio característico de la matriz M resulta ser:

$$x^5 - 1.3728\,x^4 + 0{,}1628\,x^3 - 0{,}0009\,x^2$$

Los valores singulares se calculan mediante:

```
>> svd(M)

ans =

    1.2423
    0.1244
    0.0059
    0.0001
    0.0000
```

En cuanto a los números de condición, de una forma más completa, podemos calcular la matriz V cuyas columnas son los autovectores de A, la matriz diagonal D cuyos elementos diagonales son los autovalores de A y el vector S de números de condición de los autovalores de A. El cálculo se realiza como sigue:

```
>> [V,D,S]=condeig(M)

V =
   -0.0102    0.0697    0.2756   -0.6523    0.7026
    0.1430   -0.4815   -0.7052    0.1593    0.4744
   -0.5396    0.6251   -0.2064    0.3790    0.3629
    0.7526    0.2922    0.2523    0.4442    0.2954
   -0.3490   -0.5359    0.5661    0.4563    0.2496

D =
    0.0000         0         0         0         0
         0    0.0001         0         0         0
         0         0    0.0059         0         0
         0         0         0    0.1244         0
         0         0         0         0    1.2423

S =
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

Para diagonalizar la matriz *A* calculamos la matriz diagonal *J* semejante a *A*, que tendrá los valores propios de *A* en la diagonal y la matriz de paso *V*. Para ello usamos el comando *[V,J]=jordan(A)*.

```
>> A=[3,-1,0;-1,2,-1;0,-1,3]

A =
     3    -1     0
    -1     2    -1
     0    -1     3

>> [V,J]=jordan(A)

V =
    0.1667    0.5000    0.3333
    0.3333         0   -0.3333
    0.1667   -0.5000    0.3333

J =
     1     0     0
     0     3     0
     0     0     4
```

Las matrices A y J resultan ser semejantes, ya que existe la matriz de paso V que cumple la relación V^{1} *A*$V = J$:

```
>> inv(V)*A*V

ans =

    1.0000         0         0
         0    3.0000         0
         0         0    4.0000
```

Ejercicio 6-3. Consideremos la matriz :

$$\boldsymbol{M} = \begin{bmatrix} 1 & -1 & 3 \\ -1 & i & -1-2i \\ i & 1 & i-2 \end{bmatrix}$$

Calcular sus autovalores, sus autovectores, su matriz balanceada con sus autovalores y su polinomio característico

```
>> A=[1,-1,3;-1,i,-1-2i;i,1,i-2];
>> [V,D] = eig(A)

V =

   0.9129             0.1826 + 0.5477i  -0.1826 + 0.3651i
  -0.2739 - 0.0913i   0.5477 - 0.1826i   0.3651 - 0.7303i
  -0.0913 + 0.2739i  -0.1826 - 0.5477i   0.1826 - 0.3651i

D =

   1.0000 + 1.0000i        0                   0
        0             -2.0000 + 1.0000i        0
        0                  0                   0
```

Vemos que los autovalores de A son *1+i*, *-2+i* y *0*, y los autovectores son las columnas de la matriz V. Ahora calculamos la matriz balanceada de A y veremos que sus autovalores coinciden con los de A:

```
>> balance(A)

ans =

   1.0000            -1.0000             1.5000
  -1.0000                  0 + 1.0000i  -0.5000 - 1.0000i
        0 + 2.0000i   2.0000            -2.0000 + 1.0000i
```

```
>> eig(balance(A))
ans =

 1.0000 + 1.0000i
-2.0000 + 1.0000i
      0
```

Calculamos ahora el polinomio característico de A:

```
>> p=poly(A)
p =

1.0000            1.0000 - 2.0000i  -3.0000 - 1.0000i        0
```

Luego el polinomio característico es $x^3 + x^2 - 2ix^2 - 3x - ix$.

Ejercicio 6-4. Comprobar que la matriz de Rosser tiene un autovalor doble, otro nulo, tres casi iguales, otro muy pequeño y dos opuestos. Además, comprobar que la matriz de Wilkinson de orden 8 tiene pares de autovalores cercanos pero no iguales.

```
>> [eig(rosser),eig(wilkinson(8))]

ans =

  1.0e+003 *

   -1.0200   -0.0010
   -0.0000    0.0002
    0.0001    0.0011
    1.0000    0.0017
    1.0000    0.0026
    1.0199    0.0028
    1.0200    0.0042
    1.0200    0.0043
```

Ejercicio 6-5. Resolver las ecuaciones siguientes:

$x^{3/2}\log(x) = x\log(x^{3/2})$, sqrt[1-x]+sqrt[1+x] = a, $x^4-1=0$ y Sen(z)=2

```
>> s1=solve('x^(3/2)*log(x)=x*log(x)^(3/2)')
s1 =
[ -lambertw(-1)]
[             1]
```

```
>> s2=solve('sqrt(1-x)+sqrt(1+x)=a','x')

s2 =

[  1/2*a*(-a^2+4)^(1/2)]
[ -1/2*a*(-a^2+4)^(1/2)]

>> s3=solve('x^4-1=0')

s3 =

[  1]
[ -1]
[  i]
[ -i]

>> s4=solve('sin(Z)=2')

s4 =

asin(2)
```

La solución de la primera ecuación se interpreta mejor pasándola a formato numérico como sigue:

```
>> numeric(s1)

ans =

   0.3181 - 1.3372i
   1.0000
```

Ejercicio 6-6. Resolver el sistema de dos ecuaciones siguiente:

$$cos(x/12)/exp(x^2/16)=y$$
$$-5/4 + y = sen(x^{3/2})$$

```
>> [x,y]=solve('cos(x/12)/exp(x^2/16)=y','-5/4+y=sin(x^(3/2))')

x =

-.18864189802267887925036526820236-
.34569744170126319331033283636228*i

y =

5/4+sin((-.14259332915370291604677691198415-
.51515304994330991250882243014347e-2*i)*3^(1/2))
```

Ejercicio 6-7. Estudiar y resolver el sistema:

$$\begin{aligned} x + 2y + 3z &= 6 \\ x + 3y + 8z &= 19 \\ 2x + 3y + z &= -1 \\ 5x + 6y + 4z &= 5 \end{aligned}$$

```
>> A=[1,2,3;1,3,8;2,3,1;5,6,4]

A =

     1     2     3
     1     3     8
     2     3     1
     5     6     4

>> B=[1,2,3,6;1,3,8,19;2,3,1,-1;5,6,4,5]

B =

     1     2     3     6
     1     3     8    19
     2     3     1    -1
     5     6     4     5

>> [rank(A), rank(B)]

ans =

     3     3

>> b=[6,19,-1,5]

b =

     6    19    -1     5
```

Vemos que el rango de *A* y el de *B* coinciden y su valor es 3, que es igual al número de incógnitas del sistema (3). Por lo tanto, el sistema tendrá solución única (compatible determinado *con distinto número de ecuaciones e incógnitas*). Con el comando *linsolve* puede obtenerse la única solución:

```
>> X=linsolve(A,b')

X =

[ 1]
[-2]
[ 3]
```

También puede resolverse el sistema de la siguiente forma:

```
>> A\b'

ans =

    1.0000
   -2.0000
    3.0000
```

Ejercicio 6-8. Estudiar y resolver el sistema:

$$\begin{aligned} 2x + y + z + t &= 1 \\ x + 2y + z + t &= 1 \\ x + y + 2z + t &= 1 \\ x + y + z + 2t &= 1 \end{aligned}$$

```
>> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];
>> B=[2,1,1,1,1;1,2,1,1,1;1,1,2,1,1;1,1,1,2,1];
>> [rank(A), rank(B)]

ans =

     4     4

>> b=[1,1,1,1]';
```

Vemos que las matrices *A* y *B* (ampliada) tienen rango 4, que, además, coincide con el número de incógnitas. Luego, el sistema tiene solución única (es compatible determinado *con el mismo número de ecuaciones y de incógnitas*). Para calcular su solución se utiliza cualquiera de los múltiples comandos vistos.

```
>> x = nnls(A,b)

x =

    0.2000
    0.2000
    0.2000
    0.2000

>> x = bicg(A,b)
bicg converged at iteration 1 to a solution with relative
residual 0

x =

    0.2000
    0.2000
    0.2000
    0.2000
```

```
>> x = bicgstab(A,b)
bicgstab converged at iteration 0.5 to a solution with
relative residual 0

x =

    0.2000
    0.2000
    0.2000
    0.2000

>> x = pcg(A,b)
pcg converged at iteration 1 to a solution with relative
residual 0

x =

    0.2000
    0.2000
    0.2000
    0.2000

>> x = qmr(A,b)
qmr converged at iteration 1 to a solution with relative
residual 0

x =

    0.2000
    0.2000
    0.2000
    0.2000

>> gmres(A,b)
gmres converged at iteration 1 to a solution with relative
residual 0

ans =

    0.2000
    0.2000
    0.2000
    0.2000

>> x = lsqr(A,b)
lsqr converged at iteration 2 to a solution with relative
residual 0

x =

    0.2000
    0.2000
    0.2000
    0.2000
```

Ejercicio 6-9. Dada la matriz A siguiente:

$$\begin{pmatrix} 2 & 3 & 4 & -1 & 1 \\ 3 & 4 & 7 & -2 & -1 \\ 1 & 3 & -1 & 1 & 8 \\ 0 & 5 & 5 & -1 & 4 \end{pmatrix}$$

obtener la dimensión de la variedad lineal engendrada por los vectores que forman sus filas y hallar una base de dicha variedad.

La dimensión de la variedad lineal será el rango de la matriz *A*.

```
>> A=[2,3,4,-1,1;3,4,7,-2,-1;1,3,-1,1,8;0,5,5,-1,4]

A =

     2     3     4    -1     1
     3     4     7    -2    -1
     1     3    -1     1     8
     0     5     5    -1     4

>> rank(A)

ans =

     3
```

El rango de la matriz es 3, luego la dimensión pedida es 3.

Para hallar una base se considera cualquier menor de orden 3 distinto de cero de la matriz. Los vectores que tengan componentes incluidas en ese menor formarán una base.

```
>> det([2,3,4;3,4,7;0,5,5])

ans =

   -15
```

Luego, una base de la variedad lineal generada será el conjunto de vectores *{{2,3,4,-1,1},{3,4,7,-2,-1},{0,5,5,-1,4}}*.

Ejercicio 6-10. Dada la matriz A siguiente:

$$\begin{pmatrix} 2 & 3 & -1 \\ 0 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

estudiar si los vectores que forman sus filas constituyen una base de R3 y, en caso positivo, hallar las componentes del vector b=(3,5,1) en dicha base.

```
>> A=[2,3,-1;0,0,1;2,1,0];
>> det(A)

ans =

     4
```

Como tenemos tres vectores de un espacio de dimensión 3, formarán una base si su determinante es distinto de cero (linealmente independientes). Luego realmente forman una base.

Las componentes del vector (3,5,1) en esa base se hallan planteando:

```
>> b=[3,5,1]';
>> inv(A)*b

ans =

   -1.2500
    3.5000
    5.0000
```

Ejercicio 6-11. Consideremos las bases B y B1 del espacio vectorial real tridimensional: B={{1,0,0},{-1,1,0},{0,1,-1}} y B1={{1,0,-1},{2,1,0},{-1,1,1}}. Hallar la matriz del cambio de base de B a B1 y calcular las componentes del vector {2,1,3} en base B, en la base B1.

```
>> B=[1,0,0;-1,1,0;0,1,-1];
>> B1=[1,0,-1;2,1,0;-1,1,1];
>> A=inv(B1')*B'

A =

   -0.5000    1.5000    2.5000
    0.5000   -0.5000   -0.5000
   -0.5000    1.5000    1.5000
```

Ya tenemos la matriz del cambio de base. Ahora hallamos las componentes del vector [2,3,1] en la base *B1*.

```
>> inv(B1')*B'*[2,1,3]'

ans =

     8
    -1
     5
```

Ejercicio 6-12. Hallar el producto mixto de los vectores (1,0,0), (1,1,0) y (0,1,1). Hallar también el área del triángulo de vértices (0,0), (5,1) y (3,7).

```
>> dot([1,1,2],cross([0,1,0],[0,1,1]))

ans =

     1
```

El producto mixto se ha calculado a partir del producto escalar y del producto vectorial.

```
>> (1/2)*det([0,0,1;5,1,1;3,7,1])

ans =

    16
```

Se ha aplicado la conocida fórmula del área de un triángulo en función de las coordenadas de sus vértices.

Ejercicio 6-13. Dada la aplicación lineal cuya matriz es la siguiente:

$$\begin{pmatrix} 0 & -3 & -1 & -3 & -1 \\ -3 & 3 & -3 & -3 & -1 \\ 2 & 2 & -1 & 1 & 2 \end{pmatrix}$$

hallar una base de su núcleo y la imagen de los vectores (4,2,0,0,6) y (1,2,-1,-2,3) mediante la aplicación lineal.

```
>> A=[0,-3,-1,-3,-1;-3,3,-3,-3,-1;2,2,-1,1,2] ;
>> null(A)

ans =
```

```
    0.5540   -0.0044
    0.3397    0.2209
    0.1882    0.6693
   -0.1631   -0.5801
   -0.7181    0.4083
```

Ya tenemos una base del núcleo formada por las dos columnas de la salida anterior, con lo que la dimensión del núcleo será 2.

Para hallar la imagen de los vectores dados mediante nuestra aplicación lineal hacemos lo siguiente:

```
>> A*[4,2,0,0,-6]'

ans =

     0
     0
     0

>> A*[1,2,-1,-2,3]'

ans =

    -2
     9
    11
```

Ejercicio 6-14. Consideramos la aplicación lineal f entre dos subespacios vectoriales U (contenido en R^3) y V (contenido en R^4), de tal forma que para cualquier punto (a,b,c) de U se tiene:

$$f(ab,c) = (a,b,c)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Hallar el núcleo y las dimensiones del núcleo y la imagen de f.

```
>> A=([1,0,0;0,0,0;0,0,1;0,0,0]);
```

El núcleo es el conjunto de vectores de *U* con imagen nula en *V*:

```
>> null(A)
```

```
ans =

     0
     1
     0
```

Con lo que el núcleo será el conjunto de vectores $\{0,b,0\}$ con b variando en U. Además, evidentemente el núcleo tiene de dimensión 1, ya que hemos visto que una base es el vector $\{0,1,0\}$.

```
>> rank(A)

ans =

     2
```

La dimensión de la imagen de f será 2, ya que la suma de las dimensiones del núcleo y la imagen ha de ser la dimensión de U (que es 3, por ser un subespacio de R^3). Por otra parte, la dimensión de la imagen de f ha de coincidir con el rango de la matriz de la aplicación lineal, que es 2. Los dos vectores columna que contienen a los elementos del menor no nulo que define el rango de la matriz formarán una base de la imagen de f.

```
>> det([1,0;0,1])

ans =

     1
```

Luego una base de la imagen de f será $\{\{1,0,0,0\},\{0,0,1,0\}\}$.

Ejercicio 6-15. Clasificar la forma bilineal f:UxV→R y la forma cuadrática g:U→R definidas de la siguiente forma:

$$f[(a,b,c),(d,e,f)] = (a,b,c)\begin{pmatrix} 1 & -2 & 0 \\ 0 & 0 & 4 \\ -1 & 0 & 3 \end{pmatrix}\begin{pmatrix} d \\ e \\ f \end{pmatrix}$$

$$g(a,b,c) = (a,b,c)\begin{pmatrix} 1 & -1 & 3 \\ -1 & 1 & -3/2 \\ 3 & -3/2 & 4 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

```
>> A=[1,-2,0;0,0,4;-1,0,-3]

A =

     1    -2     0
     0     0     4
    -1     0    -3

>> det(A)

ans =

     8
```

Como el determinante de la matriz de *f* es no nulo, la forma bilineal es regular no degenerada.

```
>> B=[1,-1,3;-1,1,-3/2;3,-3/2,4]

B =

    1.0000   -1.0000    3.0000
   -1.0000    1.0000   -1.5000
    3.0000   -1.5000    4.0000
```

Para clasificar la forma cuadrática calculamos sus determinantes diagonales.

```
>> det(B)

ans =

   -2.2500

>> det([1,-1;-1,1])

ans =

     0
```

Resulta que la forma cuadrática es semidefinida negativa.

Pero la clasificación también puede hacerse a través de los autovalores de la matriz de la forma cuadrática.

Una forma cuadrática es definida positiva si, y sólo si, todos sus autovalores son positivos estrictamente. Una forma cuadrática es definida negativa si, y sólo si, todos sus autovalores son negativos estrictamente.

Una forma cuadrática es semidefinida positiva si, y sólo si, todos sus autovalores son no negativos. Una forma cuadrática es semidefinida negativa si, y sólo si, todos sus autovalores son no positivos.

Una forma cuadrática es indefinida si existen autovalores positivos y negativos.

```
>> eig(B)

ans =

   -0.8569
    0.4071
    6.4498
```

Hay autovalores positivos y negativos, con lo que la forma cuadrática es indefinida.

Ejercicio 6-16. Dada la forma cuadrática g:U→R definida de la siguiente forma:

$$g(a,b,c)=(a,b,c)\begin{pmatrix}1&0&0\\0&2&2\\0&2&2\end{pmatrix}\begin{pmatrix}a\\b\\c\end{pmatrix}$$

clasificarla y hallar su ecuación reducida, su rango y su signatura.

Para clasificar la forma cuadrática calculamos sus determinantes diagonales.

```
>> G=[1,0,0;0,2,2;0,2,2]
G =
     1     0     0
     0     2     2
     0     2     2

>> det(G)
ans =
     0

>> det([1,0;0,2])
```

```
ans =

     2
```

La forma cuadrática es degenerada, semidefinida positiva.

Para hallar la ecuación reducida diagonalizamos su matriz y hallamos la expresión utilizando la matriz diagonal.

```
>> J=jordan(G)

J =

     0     0     0
     0     1     0
     0     0     4
```

La ecuación reducida de la forma cuadrática será:

$$h(x,y,z)=(x,y,z)\begin{pmatrix}0&0&0\\0&1&0\\0&0&4\end{pmatrix}\begin{pmatrix}x\\y\\z\end{pmatrix}=x^2+4z^2$$

```
>> rank(J)

ans =

     2
```

El rango de la forma cuadrática es 2, ya que el rango de su matriz es 2. La signatura también es 2, ya que el número de términos positivos en la diagonal de la matriz diagonalizada es 2.

Ejercicio 6-17. Consideramos el vector t que representa los años entre 1900 y 1990 (de 10 en 10) y el vector p con las poblaciones de Estados Unidos en esos años.

p = [75.995 91.972 105.711 123.203 131.669 150.697 179.323 203.212 226.505 249.633]

1) Inferir la población de Estados Unidos para 1975 mediante una función de interpolación.
2) Representar el polinomio interpolador spline entre los años 1900 y 2000 de año en año para los valores de p dados.

```
>> t = 1900:10:1990;
p = [75.995  91.972  105.711  123.203  131.669...
     150.697  179.323  203.212  226.505  249.633];
```

```
>> interp1(t,p,1975)

ans =

  214.8585
```

La población prevista para Estados Unidos en 1975 según la función de interpolación es 214,8585.

La función de interpolación *spline* (Figura 6-3) se representa como sigue:

```
>> x = 1900:1:2000;
 y = interp1(t,p,x,'spline');
 plot(t,p,'o',x,y)
```

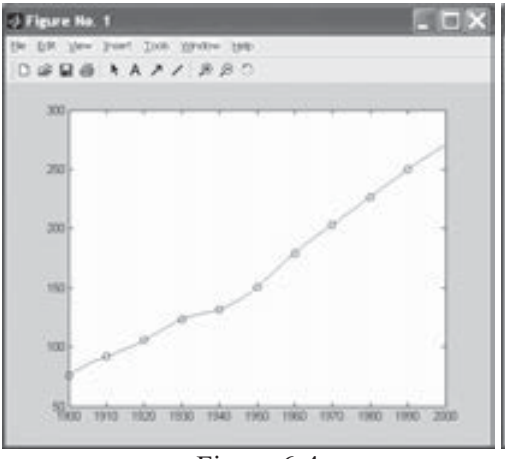


Figura 6-4

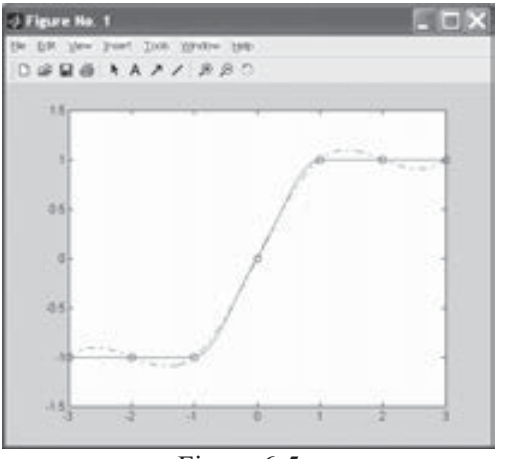


Figura 6-5

Ejercicio 6-18. Representar la diferencia entre el polinomio interpolador cúbico hermitiano a trozos y el polinomio interpolador spline cuando x y t varían entre -3 y 3 (t varía de décima en décima) e y = [-1 -1 -1 0 1 1 1].

```
>> x = -3:3;
y = [-1 -1 -1 0 1 1 1];
t = -3:.01:3;
p = pchip(x,y,t);
s = spline(x,y,t);
plot(x,y,'o',t,p,'-',t,s,'-.')
```

La representación gráfica se observa en la Figura 6-5.

Ejercicio 6-19. Representar 48 puntos de interpolación bidimensional entre los puntos (x,y,z) dados por los valores que toma la función peaks (predefinida en MATLAB) para 20 valores de x e y igualmente espaciados entre 0 y 2 y definiendo la matriz de puntos de la superficie z=f(x,y).

Comenzamos definiendo la matriz de puntos y a continuación se realiza la interpolación. Por último se hace una representación gráfica a medida (Figura 6-6).

```
>> [X,Y] = meshgrid(-3:.25:3);
Z = peaks(X,Y);
[XI,YI] = meshgrid(-3:.125:3);
ZI = interp2(X,Y,Z,XI,YI);
mesh(X,Y,Z), hold, mesh(XI,YI,ZI+15)
hold off
axis([-3 3 -3 3 -5 20])
```

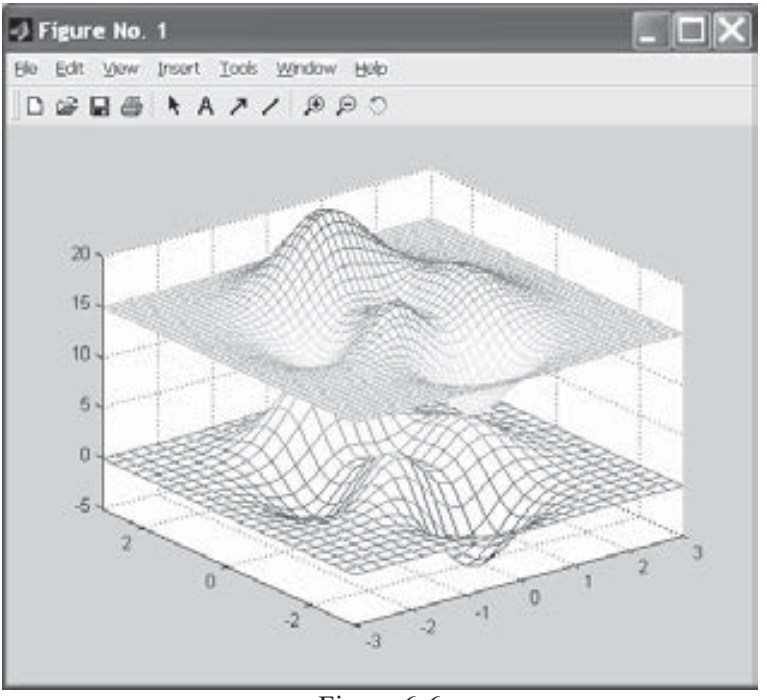


Figura 6-6

Ejercicio 6-20. Representar la matriz de puntos volumétrica w=f(x,y,z) de interpolación tridimensional entre los puntos (x,y,z,t) dados por los valores que toma la función flor (predefinida en MATLAB) para valores de x entre 0,1 y 10 separados un cuarto de punto y para valores de y, z entre -3 y 3 separados también un cuarto de punto. Dibujar cortes según las direcciones X, Y, Z en el volumen V a través de la superficie (Xi,Yi,Zi).

La representación pedida (Figura 6-7) se obtiene a partir de la sintaxis siguiente:

```
>> [x,y,z,v] = flow(10);
[xi,yi,zi] = meshgrid(.1:.25:10, -3:.25:3, -3:.25:3);
vi = interp3(x,y,z,v,xi,yi,zi);
slice(xi,yi,zi,vi,[6 9.5],2,[-2 .2]), shading flat
```

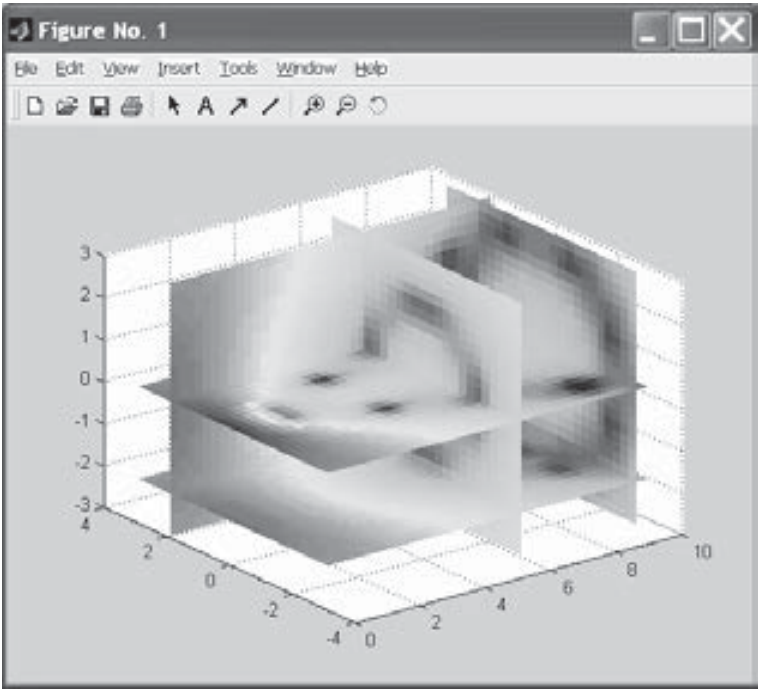


Figura 6-7

7

Representación geométrica: curvas y superficies

7.1 Graficando datos

El módulo básico de MATLAB ofrece una gama amplia de opciones a la hora de realizar representaciones gráficas. Permite realizar gráficos de curvas planas y superficies, posibilitando la agrupación y la superposición. También es posible trabajar colores, rejillas, marcos, etc., en los gráficos. Las representaciones de funciones pueden realizarse en coordenadas implícitas, explícitas y paramétricas. MATLAB es, por tanto, un software matemático con elevadas prestaciones gráficas, lo que le distingue de muchos otros paquetes de cálculo simbólico. También permite MATLAB realizar gráficos de barras, líneas, estrellas, histogramas, poliedros, mapas geográficos y animaciones. La creación de un gráfico suele acoplarse a los siguiente pasos:

Paso	***Ejemplo***
Preparar los datos	`x = 0:0.2:12; y1 = bessel(1,x); y2 = bessel(2,x); y3 = bessel(3,x);`
Elegir ventana y situar posición	`figure(1); subplot(2,2,1)`
Usar función de gráfico	`h = plot(x,y1,x,y2,x,y3);`
Elegir características de líneas y marcadores (anchura, colores, ...)	`set(h,'LineWidth',2,{'LineStyle'}; set(h,{'Color'},{'r';'g';'b'})`
Situar límites de ejes, marcas y mallas	`axis([0 12 -0.5 1]) grid on`
Situar anotaciones, etiquetas y leyendas	`xlabel('Time') ylabel('Amplitude') legend(h,'First','Second','Third') title('Bessel Functions') [y,ix] = min(y1);`
Exportar el gráfico	`print -depsc -tiff -r200 migrafico`

7.2 Gráficos básicos 2D: barras, sectores, histogramas, racimo, error y flechas

A continuación se presentan los comandos más importantes de MATLAB que permiten realizar gráficos bidimensionales básicos. Es de resaltar la posibilidad de representar funciones tanto en coordenadas explícitas como paramétricas y polares.

bar(Y)	*Gráfico de barras relativo al vector de frecuencias Y. Si Y es matriz se obtiene el gráfico de barras múltiple para cada fila de Y*
bar(x,Y)	*Gráfico de barras relativo al vector de frecuencias Y siendo x un vector que define los espacios en el eje X para situar las barras*
bar(...,anchura)	*Gráfico con anchura de las barras dada. Por defecto, la anchura es 0,8 y la anchura 1 provoca que las barras se toquen*
bar(...,'e*stilo*')	*Gráfico con estilo para las barras dado. Los estilos son 'group' (estilo por defecto con barras verticales agrupadas) y 'stack' (barras apiladas). Si la matriz Y es (m,n), el gráfico agrupado tiene m grupos de n barras verticales*
bar(...,color)	*Las barras son todas del color especificado (r=rojo, g=verde, b=azul, c=cyan, m=magenta y=yellow, k=black y w=white)*
barh(...)	*Gráficos de barras horizontales*
hist(Y)	*Dibuja el histograma relativo al vector de frecuencias Y usando 10 rectángulos iguales de igual base. Si Y es una matriz, se construye un histograma para cada una de sus columnas.*
hist(Y,x)	*Dibuja el histograma relativo al vector de frecuencias Y usando tantas cajas como elementos tiene el vector x y centrando cada caja en los sucesivos valores de x*
hist(Y,k)	*Dibuja el histograma relativo al vector de frecuencias Y usando tantas cajas como indica el escalar k.*
[n,x] = hist(...)	*Devuelve los vectores n y x con las frecuencias asignadas a cada caja del histograma y los valores en los que se centran las cajas*
pie(X)	*Realiza el gráfico de sectores relativo al vector de frecuencias X*
pie(X,Y)	*Realiza el gráfico de sectores relativo al vector de frecuencias X desplazando hacia fuera los sectores en los que Yi ≠0*
errorbar(x,y,e)	*Realiza el gráfico del vector x contra el vector y con los errores especificados en el vector e. Pasando por cada punto (x_i,y_i) se dibuja una línea vertical de longitud $2e_i$ cuyo centro es el punto (x_i,y_i)*
stem(Y)	*Dibuja el gráfico de racimo relativo al vector Y. Cada punto de Y es unido al eje x por una línea vertical*
stem(X,Y)	*Dibuja el gráfico de racimo relativo al vector Y cuyos elementos son dados a través del vector X*
stairs(Y)	*Dibuja el gráfico escalonado relativo al vector Y*

stairs(X,Y)	*Gráfico escalonado del vector Y con elementos a través del vector X*
rose(Y)	*Dibuja el histograma angular relativo al vector Y, de ángulos en radianes utilizando 20 radios iguales*
rose(Y,n)	*Dibuja el histograma angular del vector Y usando n radios iguales*
rose(X,Y)	*Dibuja el histograma angular relativo al vector Y utilizando radios que miden lo especificado en los elementos del vector X*
compas(Z)	*Realiza un diagrama de flechas que salen del origen y cuya magnitud y dirección vienen determinadas por el módulo y el argumento de los números complejos componentes del vector Z. La flecha relativa al complejo Zi une el origen con el afijo de Zi*
compas(X,Y)	*Equivale a compas(X+i*Y)*
compas(Z,S) o compas(X,Y,S)	*Especifica en S el tipo de línea a usar en las flechas*
feather(Z) o feather(X,Y) o feather(Z,S) o feather(X,Y,S)	*Es igual que compas, con la única diferencia de que el origen de las flechas no está en el origen de coordenadas, sino que salen de puntos igualmente espaciados de una línea horizontal*
legend ('leyenda1', 'leyenda2', ..., 'leyendan')	*Sitúa las leyendas dadas en n gráficos consecutivos*

Como primer ejemplo representamos un gráfico de barras (Figura 7-1) definidas por la función e^{-x*x} cuando x varía en (-3,3) de dos en dos décimas.

```
>> x = -2.9:0.2:2.9;
bar(x,exp(-x.*x))
```

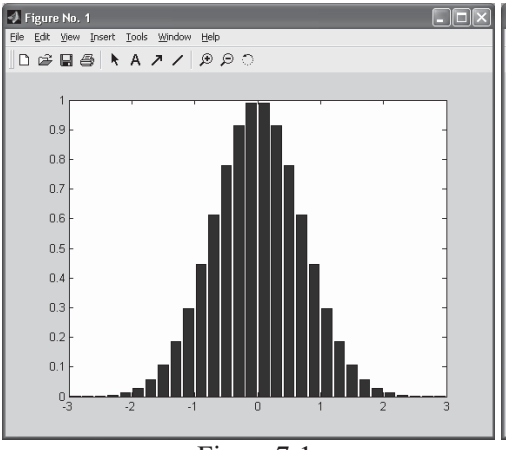


Figura 7-1

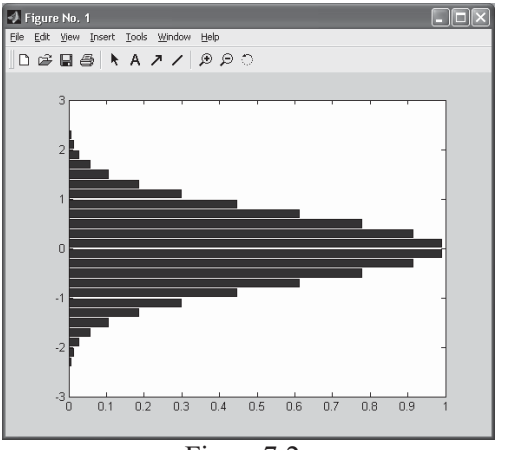


Figura 7-2

A continuación representamos el mismo gráfico para barras horizontales (Figura 7-2).

```
>> x = -2.9:0.2:2.9;
>> barh(x,exp(-x.*x))
```

En el ejemplo siguiente se grafican los 5 grupos de 3 barras correspondientes a una matriz aleatoria (5,3), tanto en modo agrupado (Figura 7-3), como apilado (Figura 7-4).

```
>> Y = round(rand(5,3)*10);
>> bar(Y,'group')
>> bar(Y,'stack')
```

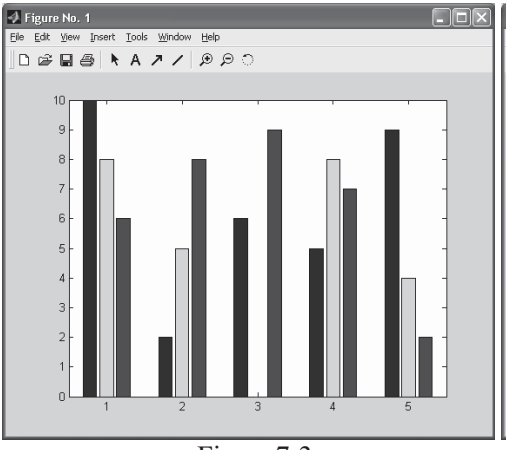


Figura 7-3

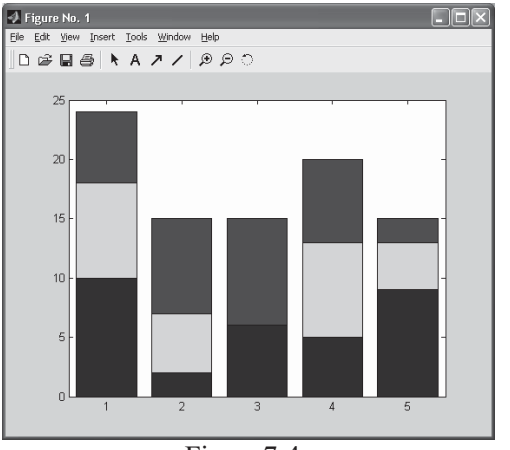


Figura 7-4

En el ejemplo siguiente se muestran las barras apiladas horizontalmente (Figura 7-5) y también el gráfico de barras verticales con anchura 1,5.

```
>> Y = round(rand(5,3)*10);
>> barh(Y,'stack')
>> bar(Y,1.5)
```

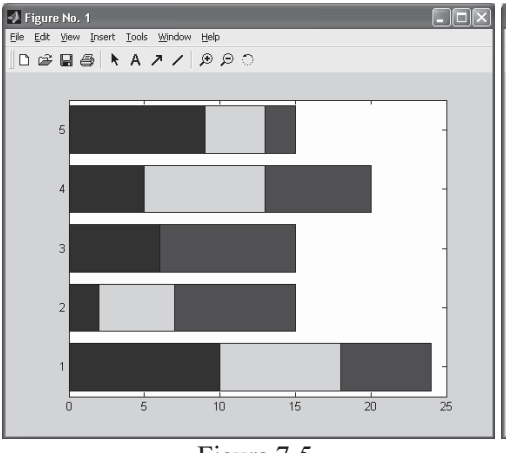


Figura 7-5

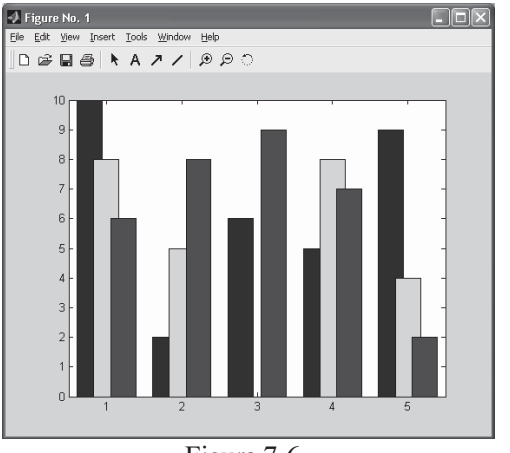


Figura 7-6

A continuación consideramos un vector con 10.000 puntos aleatorios normales (0,1) y graficamos el histograma de frecuencias relativo a dichos puntos (Figura 7-7).

```
>> y = randn(10000,1);
>> hist(y)
```

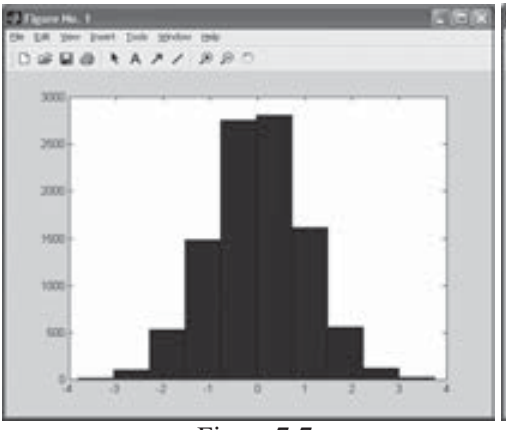


Figura 7-7

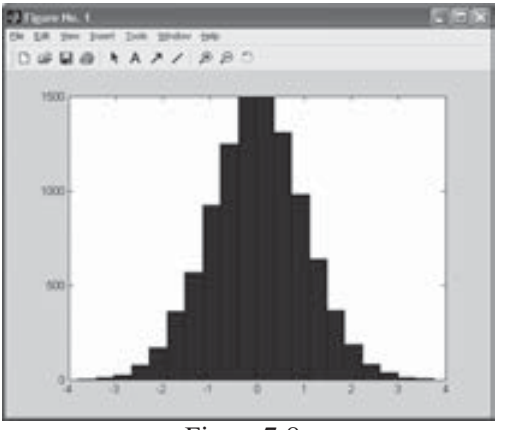


Figura 7-8

También se puede graficar el histograma anterior con 20 cajas (Figura 7-8) mediante la sintaxis siguiente:

```
>> y = randn(10000,1);
>> hist(y,20)
```

A continuación representamos un gráfico de sectores para los valores especificados en el vector *x* (Figura 7-9).

```
>> x = [1 3 0.5 2.5 2];
>> pie(x)
```

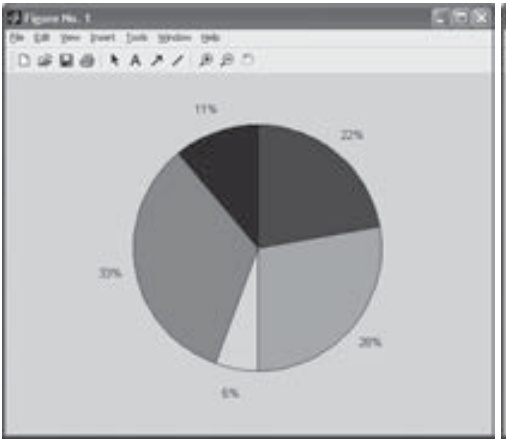


Figura 7-9

Figura 7-10

En la representación siguiente se presenta el mismo gráfico de sectores anterior, pero con el segundo sector desplazado (Figura 7-10).

```
>> x = [1 3 0.5 2.5 2];
>> y = [0 1 0 0 0];
>> plot(x,y)
```

A continuación se presenta el gráfico escalonado relativo a la función e^{-x*x} para puntos de x entre -3 y 3 separados por una décima (Figura 7-11).

```
>> x=-3:0.1:3;
>> stairs(x,exp(-x.^2))
```

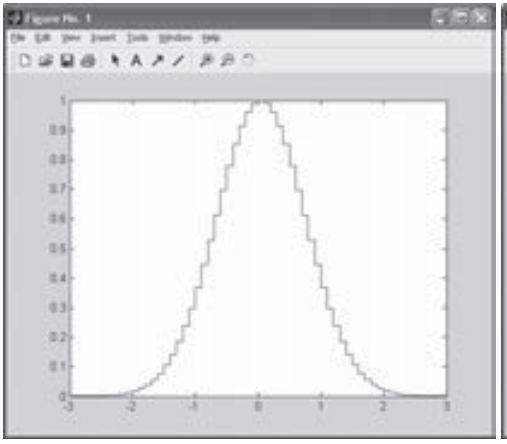


Figura 7-11

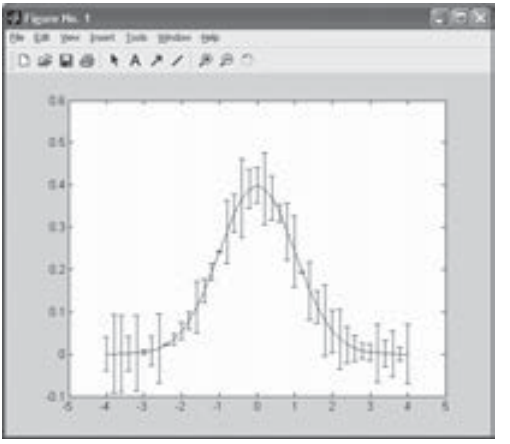


Figura 7-12

Seguidamente se representa (Figura 7-12) un gráfico de errores para la función de densidad de una distribución normal (0,1), con la variable definida en 40 puntos entre -4 y 4, siendo definidos los errores por 40 valores aleatorios uniformes (0,10).

```
>> x  = -4:.2:4;
y=(1/sqrt(2*pi))*exp(-(x.^2)/2);
e=rand(size(x))/10;
errorbar(x,y,e)
```

En el ejemplo siguiente representamos un gráfico de racimo (Figura 7-13) relativo a 50 números aleatorios normales (0,1).

```
>> y=randn(50,1);
>> stem(y)
```

A continuación se presenta un histograma angular (Figura 7-14) para 1.000 ángulos múltiplos de π según una razón de multiplicidad aleatoria normal (0,1).

```
>> y=randn(1000,1)*pi;
>> rose(y)
```

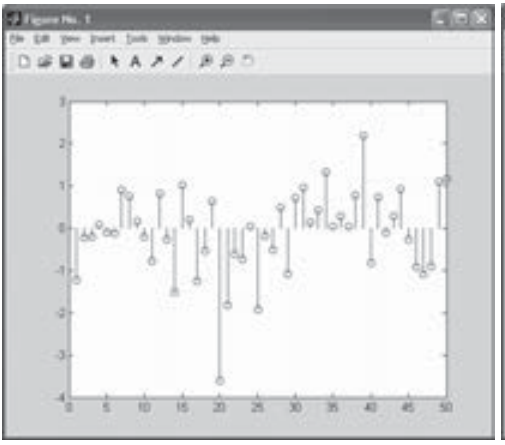


Figura 7-13

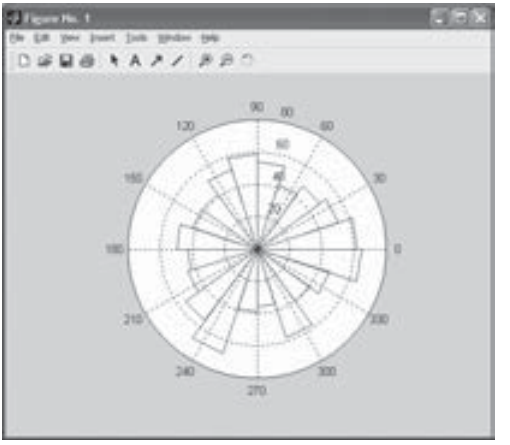


Figura 7-14

Ahora se presenta un gráfico de flechas con centro en el origen, correspondiente a los autovalores de una matriz cuadrada aleatoria normal (0,1) de tamaño 20x20 (Figura 7-15).

```
>> z=eig(randn(20,20));
>> compass(z)
```

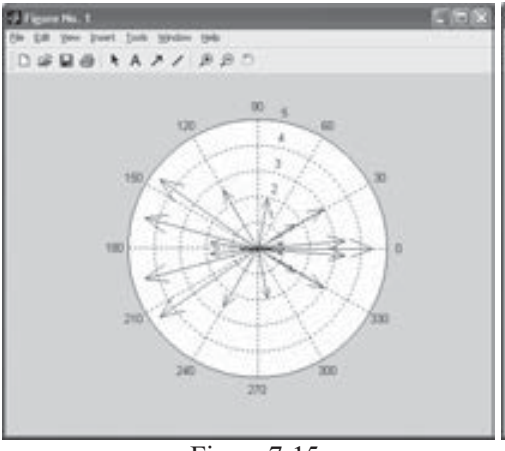


Figura 7-15

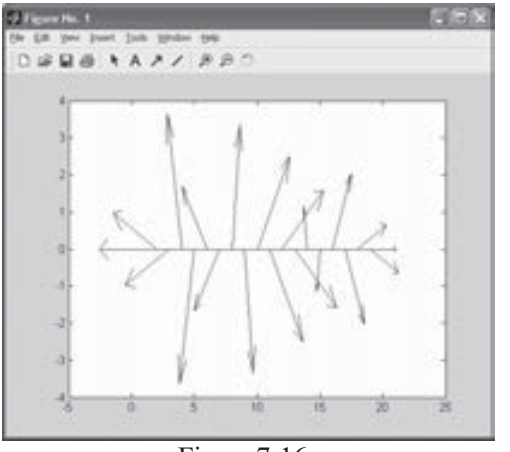


Figura 7-16

A continuación se representa el gráfico de flechas del ejemplo anterior, pero con el origen de las flechas en una recta horizontal (Figura 7-16).

```
>> z=eig(randn(20,20));
>> feather(z)
```

7.3 Gráficos 2D: curvas en explícitas, implícitas, paramétricas y polares

Los comandos más importantes de MATLAB para representar curvas en dos dimensiones se presentan a continuación.

plot(X,Y)	*Dibuja el conjunto de puntos (X,Y), donde X e Y son vectores fila. Para graficar una función y=f(x) es necesario conocer un conjunto de puntos (X,f(X)), para lo que hay que definir inicialmente un intervalo de variación vectorial X para la variable x. X e Y pueden ser matrices de la misma dimensión, en cuyo caso se hace una gráfica por cada par de filas y sobre los mismos ejes. Para valores complejos de X e Y se ignoran las partes imaginarias. Para x = x(t) e y = y(t) con la variación de t dada, grafica la curva paramétrica plana especificada*
plot(Y)	*Grafica los elementos del vector Y contra sus índices, es decir, da la gráfica del conjunto de puntos (t,y_t) t=1,2,...n (n=length(Y)). Es útil para graficar series temporales. Si Y es una matriz, plot(Y) realiza un gráfico para cada columna de Y, presentándolos todos sobre los mismos ejes. Si los componentes del vector Y son complejos, plot(Y) es equivalente a plot(real(Y),imag(Y)).*
plot(X,Y,S)	*Gráfica de plot(X,Y) con las opciones definidas en S. Usualmente, S se compone de dos dígitos entre comillas simples, el primero de los cuales fija el color de la línea del gráfico y el segundo el carácter a usar en el graficado. Los valores posibles de colores y caracteres son, respectivamente, los siguientes: y (amarillo), m (magenta), c (cyan), r (rojo), g (verde), b (azul), w (blanco), k (negro), . (puntos), o (círculos), x (x-marcas), + (signo más), - (sólido), * (estrellas), : (dos puntos), -. (guiones y punto) y – (semisólido)*
plot(X1,Y1,S1,X2,Y2,S2,…	*Combina, sobre los mismos ejes, los gráficos definidos para las tripletas (Xi,Yi,Si). Se trata de una forma de representar varias funciones sobre el mismo gráfico*
fplot('f',[xmin, *x*max])	*Grafica la función en el intervalo de variación de x dado*
fplot('f',[xmin, *x*max, ymin, ymax], S)	*Grafica la función en los intervalos de variación de x e y dados, con las opciones de color y caracteres dadas por S*
fplot('[f1,f2,...,fn]',[xmin, *x*max, ymin, ymax], S)	*Grafica las funciones f1, f2, ..., fn sobre los mismos ejes en los intervalos de variación de x e y especificados y con las opciones de color y caracteres definidas en S*

fplot('f',[xmin, *x*max],…,t)	*Grafica f con la tolerancia t*
fplot('f',[xmin, *x*max],…,n)	*Grafica f con la tolerancia t como n+1 puntos como mínimo*
ezplot('f', [xmin *x*max])	*Grafica la función en el intervalo de variación de x dado*
ezplot('f',[xmin, *x*max, ymin, ymax])	*Grafica la función en los intervalos de variación de x e y dados*
ezplot(x,y)	*Grafica la curva paramétrica plana x = x(t)* e *y = y(t) sobre el dominio 0 < t < 2π*
ezplot('f', [xmin *x*max])	*Grafica la curva paramétrica plana x = x(t)* e *y = y(t) sobre el dominio xmin < t < xmax*
ezplot('f')	*Grafica la curva f en coordenadas implícitas en [-2π,2π]*
loglog(X,Y)	*Realiza gráficos similares a plot(X,Y), pero con escala logarítmica en los dos ejes*
semilogx(X,Y)	*Realiza gráficos similares a plot(X,Y), pero con escala logarítmica en el eje X y escala normal en el eje Y*
semilogy(X,Y)	*Realiza gráficos similares a plot(X,Y), pero con escala logarítmica en el eje Y y escala normal en el eje X*
fill(X,Y,C)	*Dibuja el polígono compacto cuyos vértices son los pares de componentes (Xi,Yi) de los vectores columna X e Y. C es un vector de la misma dimensión de X e Y, que contiene los colores Ci de cada punto (Xi,Yi). Los valores de Ci pueden ser: 'r','g','b', 'c','m','y','w','k', cuyos significados ya conocemos. Si C es un solo carácter, se pintarán todos los puntos del polígono del color correspondiente al carácter. Si X e Y son matrices de la misma dimensión, se representarán simultáneamente varios polígonos correspondientes a cada par de vectores columna (X.j,Y.j). En este caso, C puede ser un vector fila cuyos elementos Cj determinan el color único de cada polígono correspondiente al par de vectores columna (X.j,Y.j). C puede ser también una matriz de la misma dimensión que X e Y, en cuyo caso sus elementos determinan los colores de cada punto (Xij,Yij) del conjunto de polígonos.*
fill(X1,Y1,C1,...)	*Dibuja el polígono compacto cuyos vértices vienen dados por los puntos (Xi, Yi, Ci).*
polar(α,r)	*Dibuja la curva en coordenadas polares r=r(α)*
polar(α,r,S)	*Dibuja la curva en coordenadas polares r=r(α) con el estilo de líneas dado por S*

Como primer ejemplo realizamos la gráfica de la función $f(x) = Sen(x)\,e^{-0,4x}$ en el intervalo [0,10]. La Figura 7-17 presenta el resultado relativo a la siguiente sintaxis:

```
>> x=0:0.05:10;
>> y=sin(x).*exp(-0.4*x);
>> plot(x,y)
```

La representación de la curva anterior (Figura 7-18) también puede realizarse mediante la sintaxis siguiente:

```
>> ezplot('sin(x)*exp(-0.4*x)',[0,10])
```

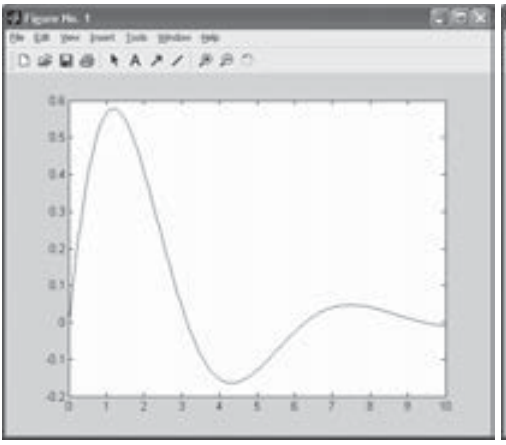


Figura 7-17

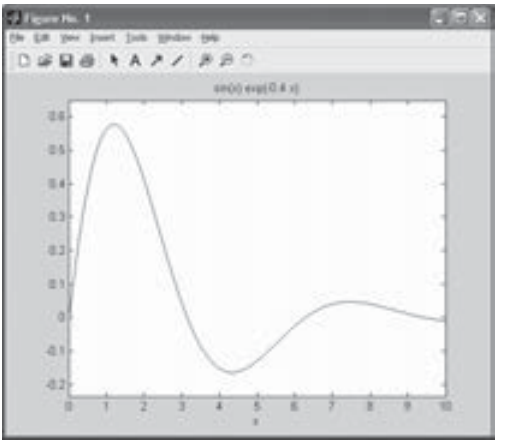


Figura 7-18

Asimismo se obtendría la misma representación gráfica mediante el comando siguiente:

```
>> fplot('sin(x)*exp(-0.4*x)',[0,10])
```

A continuación se representa (Figura 7-19) la curva en implícitas $x^2 - y^4 = 0$ en [-2π,2π] mediante la sintaxis siguiente:

```
>> ezplot('x^2-y^4')
```

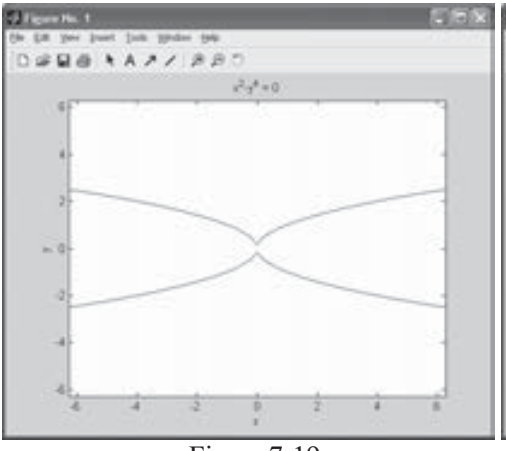


Figura 7-19

Figura 7-20

Seguidamente se grafica la curva en paramétricas $x(t) = 4cos(t)\text{-}cos(4t)$, $y(t) = 4sen(t)\text{-}sen(4t)$ para t variando entre 0 y 2 π (Figura 7-20). La sintaxis es la siguiente:

```
>> t=0:0.1:2*pi;
>> x=4*cos(t)-cos(4*t);
>> y=4*sin(t)-sin(4*t);
>> plot(x,y)
```

El mismo gráfico (Figura 7-21) podría haberse obtenido mediante la sintaxis siguiente:

```
>> t=0:0.1:2*pi;
>> x=4*cos(t)-cos(4*t);
>> y=4*sin(t)-sin(4*t);
>> ezplot('4*cos(t)-cos(4*t)','4*sin(t)-sin(4*t)', [0,2*pi])
```

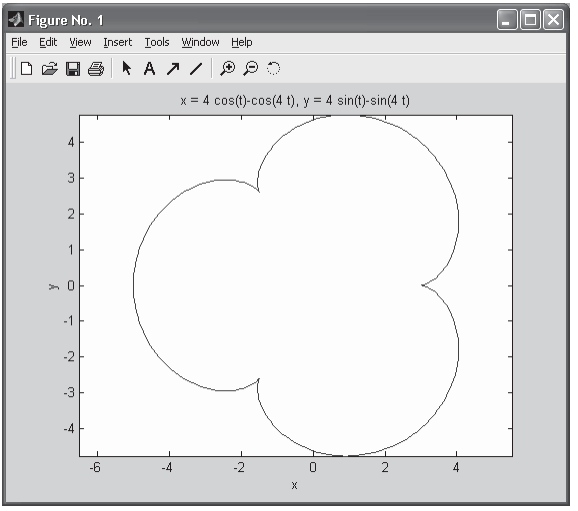


Figura 7-21

A continuación representamos sobre los mismos ejes (Figura 7-22) las curvas $Sen(x)$, $Sen(2x)$ y $Sen(3x)$. La sintaxis es la siguiente:

```
>> fplot('[sin(x), sin(2*x), sin(3*x)]', [0,2*pi])
```

Podemos intentar representar cada curva anterior mediante distinto trazo (Figura 7-23) mediante la sintaxis siguiente:

```
>> fplot('[sin(x), sin(2*x), sin(3*x)]', [0,2*pi],'-','o','*')
```

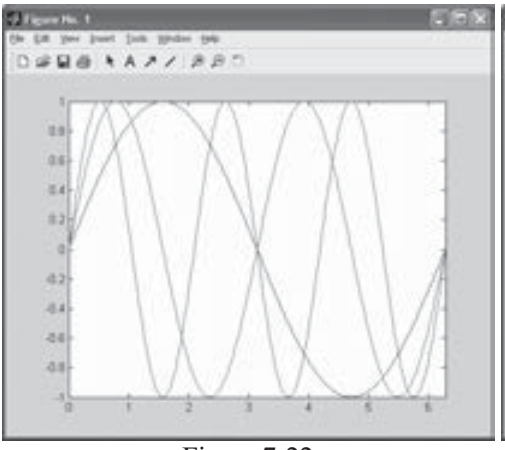


Figura 7-22

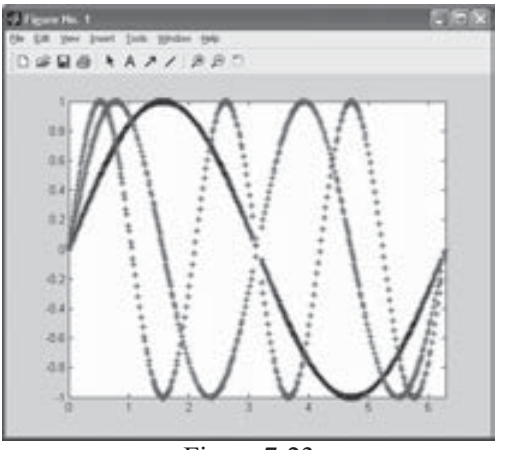


Figura 7-23

A continuación representamos la curva en polares $r = sen(2a)\ Cos(2a)$ para a entre 0 y 2π (Figura 7-24).

```
>> t=0:0.1:2*pi;
>> r=sin(2*t).*cos(2*t);
>> polar(t,r)
```

También se podría representar la curva polar anterior con un trazo discontinuo de color rojo (Figura 7-25) mediante la sintaxis siguiente:

```
>> t=0:0.1:2*pi;
>> r=sin(2*t).*cos(2*t);
>> polar(t,r,'--r')
```

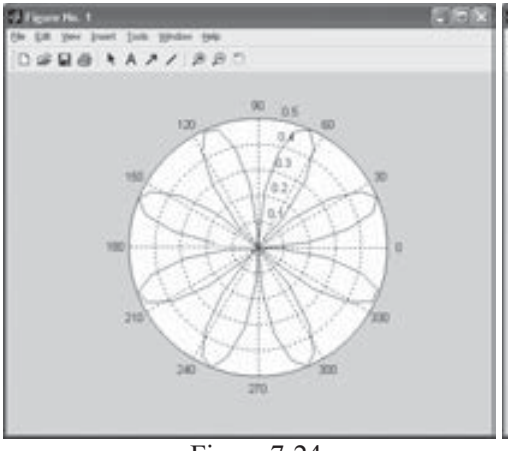


Figura 7-24

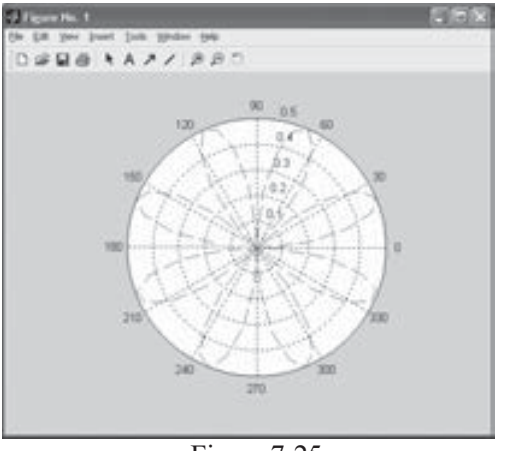


Figura 7-25

7.4 Títulos, etiquetas y colocación

MATLAB permite manejar correctamente las anotaciones sobre los gráficos y los ejes mediante colocación adecuada de títulos, etiquetas, leyendas, rejillas, etc. También permite situar gráficos en subzonas. Los comandos más usuales son los siguientes:

title('texto')	*Añade el texto como título del gráfico en la parte superior del mismo en gráficos 2-D y 3-D*
xlabel('texto')	*Sitúa el texto al lado del eje x en gráficos 2-D y 3-D*
ylabel('texto')	*Sitúa el texto al lado del eje y en gráficos 2-D y 3-D*
zlabel('texto')	*Sitúa el texto al lado de eje z en un gráfico 3-D*
clabel(C,h)	*Rota etiquetas y las sitúa en el interior de las líneas de contorno*
clabel(C,h,v)	*Crea etiquetas sólo para los niveles de contorno dados por el vector v y las rota y sitúa en el interior de las líneas de contorno*
datetick(eje)	*Etiqueta las marcas del eje especificado ('x', 'y' o 'z') basándose en el máximo y el mínimo del eje especificado*
datetick(eje, fecha)	*Etiqueta las marcas del eje especificado ('x', 'y' o 'z') con el formato de fecha dado (un entero entre 1 y 28)*
legend('cadena1', 'cadena2',...)	*Sitúa las leyendas especificadas por las cadenas en n gráficos consecutivos*
legend(h, 'cadena1', 'cadena2',...)	*Sitúa las leyendas especificadas por las cadenas en los gráficos manejados según el vector h*
legend('off'),	*Elimina las leyendas de los ejes actuales*
text(x,y,'texto')	*Sitúa el texto en el punto (x,y) dentro del gráfico 2-D*
text(x,y,z,'texto')	*Sitúa el texto en el punto (x,y,z) en el gráfico 3-D*
gtext('texto')	*Permite situar el texto en un punto seleccionado con el ratón dentro de un gráfico 2-D*
grid	*Sitúa rejillas en los ejes de un gráfico 2-D o 3-D. La opción grid on coloca las rejillas y greed off las elimina. La opción grid permuta entre on y off*
hold	*Permite mantener el gráfico existente con todas sus propiedades, de modo que el siguiente gráfico que se realice se sitúe sobre los mismos ejes y se superponga al existente. La opción hold on activa la opción y hold off la elimina. La opción hold permuta entre on y off. Válido para 2-D y 3-D*
axis([xmin xmax ymin ymax zmin zmax]	*Sitúa los valores máximo y mínimo para los ejes X, Y y Z en el gráfico corriente*
axis('auto')	*Sitúa los ejes en la escala automática por defecto (la dada por xmin = min(x), xmax = max(x) e y libre)*
axis(axis)	*Congela el escalado de ejes en los límites corrientes, de tal forma que al situar otro gráfico sobre los mismos ejes (con hold en on) la escala no cambie*

V=axis	*Da el vector V de 4 elementos, conteniendo la escala del gráfico corriente*
axis('xy')	*Sitúa coordenadas cartesianas con el origen en la parte inferior izquierda del gráfico*
Axis('tight')	*Sitúa los límites de los ejes en el rango de los datos*
axis('ij')	*Sitúa coordenadas con el origen en la parte superior izquierda del gráfico*
axis('square')	*Convierte el rectángulo de graficado en un cuadrado, con lo que las figuras se abomban*
axis('equal')	*Sitúa el mismo factor de escala para ambos ejes*
axis('normal')	*Elimina las opciones square y equal*
axis('off')	*Elimina las etiquetas y marcas de los ejes y las rejillas, manteniendo el título del gráfico y los textos situados en él con text y gtext*
axis('on')	*Coloca de nuevo las etiquetas, marcas y rejillas de los ejes*
subplot(m,n,p)	*Divide la ventana gráfica en mxn subventanas y coloca el gráfico corriente en la ventana p-ésima, empezando a contar por la parte superior izquierda y de izquierda a derecha hasta acabar la línea, para pasar a la siguiente (Figura 7-26)*
plotyy(X1,Y1,X2,Y2)	*Etiqueta la gráfica de X1 contra Y1 en la izquierda, y la gráfica de X2 contra Y2 en la derecha.*
plotyy(X1,Y1,X2,Y2, 'función')	*Igual que el comando anterior, pero la función puede ser plot, loglog, semilogx, semilogy, ítem o cualquier h=function(x,y).*
plotyy(X1,Y1,X2,Y2, 'función1','funcción2')	*Igual que el comando anterior, pero usando cada función fi para el par (XiYi)*

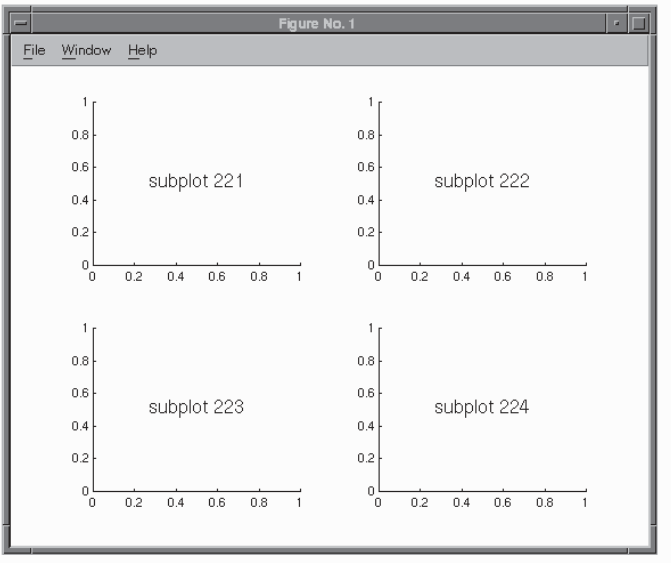


Figura 7-26

A continuación representamos sobre los mismos ejes (Figura 7-27) las curvas *Sen*(*x*), *Sen*(2*x*) y *Sen*(3*x*) etiquetadas adecuadamente. La sintaxis es la siguiente:

```
>> fplot('[sin(x), sin(2*x), sin(3*x)]', [0,2*pi],'-','o','*');
>> legend('Sen(x)', 'Sen(2x)', 'Sen(3x)')
```

También es posible situar las etiquetas sobre cada gráfica (Figura 7-28) mediante la sintaxis siguiente:

```
>> fplot('[sin(x), sin(2*x), sin(3*x)]', [0,2*pi],'-','o','*');
>> text(3,0.8,'y=sen(2x)')
>> text(1,-0.6,'y=sen(3x)')
>> text(1.5,1,'y=sen(x)')
```

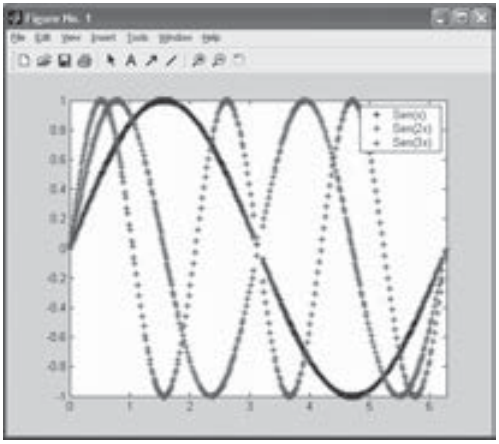


Figura 7-27

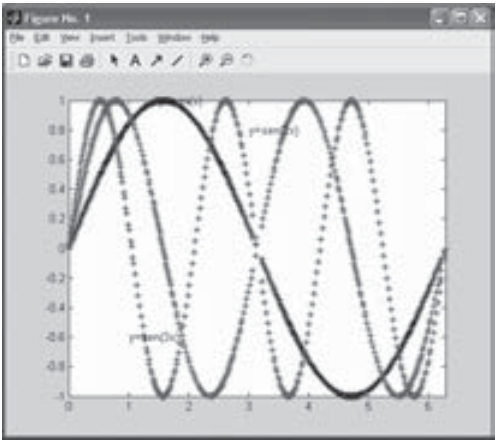


Figura 7-28

En el ejemplo siguiente se representan sobre los mismos ejes las gráficas de las funciones $y=sen(x^2)$ e $y=log(sqrt(x))$, colocándose el texto de cada ecuación adecuadamente dentro del gráfico, así como el titular del gráfico y de los dos ejes (Figura 7-29).

```
>> x=linspace(0,2,30);
y=sin(x.^2);
plot(x,y)
text(1,0.8, 'y=sin(x^2)')
hold on
z=log(sqrt(x));
plot(x,z)
text(1,-0.1, 'y=log(sqrt(x))')
xlabel('Eje X');
ylabel('Eje Y');
title('Gráfico senoidal y logarítmico');
```

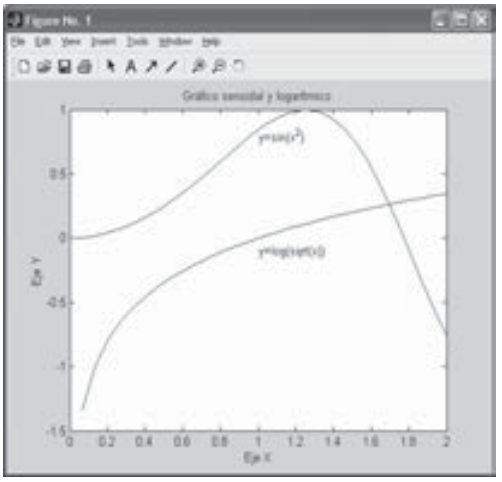


Figura 7-29

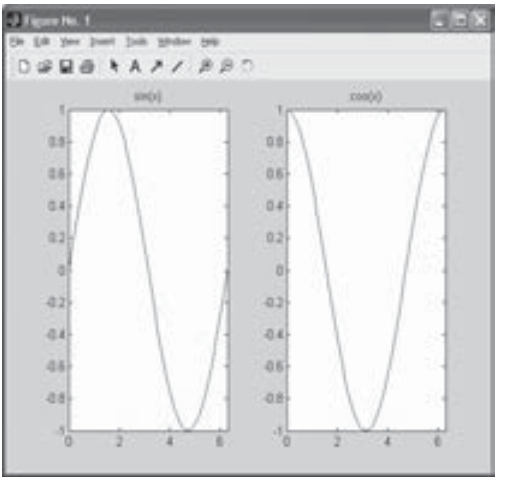


Figura 7-30

En el ejemplo siguiente se presentan en el mismo gráfico (Figura 7-30) las gráficas de las funciones *Sen*(*x*) y *Cos*(*x*), colocadas horizontalmente una al lado de la otra con sus nombres y con el eje *x* tomando valores entre 0 y 2π y el eje *y* tomando valores entre -1 y 1.

```
>> x=(0:.1:4*pi);
y=sin(x);
z=cos(x);
subplot(1,2,1);
plot(x,y), axis([0 2*pi -1 1]), title('sin(x)')
subplot(1,2,2);
plot(x,z), axis([0 2*pi -1 1]), title('cos(x)')
```

A continuación se realiza la representación anterior, pero de modo vertical, situando una figura debajo de la otra y con rejillas en los ejes (Figura 7-31).

```
>> x=(0:.1:4*pi);
y=sin(x);
z=cos(x);
subplot(2,1,1);
plot(x,y), axis([0 2*pi -1 1]), title('sin(x)'), grid
subplot(2,1,2);
plot(x,z), axis([0 2*pi -1 1]), title('cos(x)'), gris
```

En el ejemplo siguiente se presentan en el mismo gráfico (Figura 7-32) las gráficas de las funciones *Sen*(*x*), *Cos*(*x*), *Cosec*(*x*) y *Sec* (*x*), colocadas en una matriz de cuatro gráficos, de forma que debajo de cada función esté su inversa para *x* variando en [-2π, 2π].

```
>> subplot(2,2,1);
ezplot('sin(x)',[-2*pi 2*pi])
subplot(2,2,2);
ezplot('cos(x)',[-2*pi 2*pi])
subplot(2,2,3);
ezplot('csc(x)',[-2*pi 2*pi])
subplot(2,2,4);
ezplot('sec(x)',[-2*pi 2*pi])
```

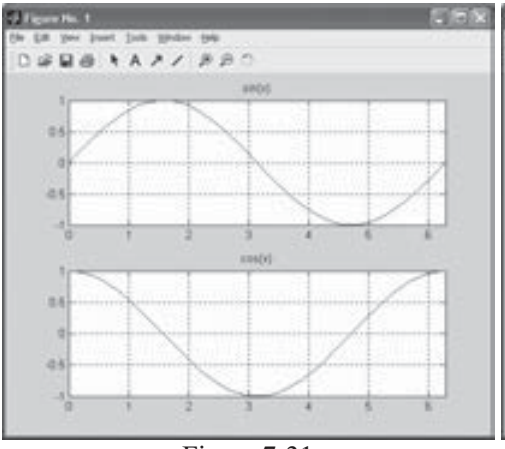


Figura 7-31

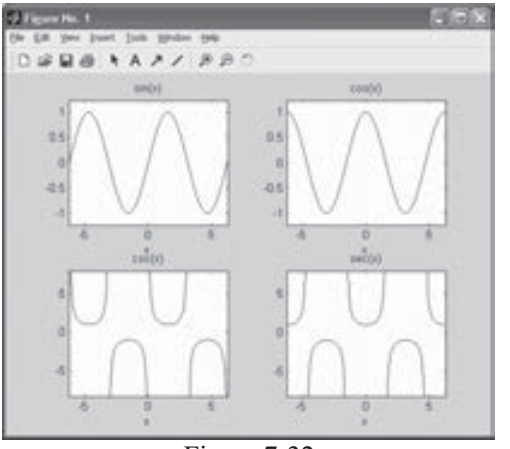


Figura 7-32

A continuación se presenta sobre el mismo gráfico (Figura 7-33) la función $y = abs(e^{-1/2\ x}\ Sen(5x))$ representada en escala normal, en escala logarítmica y en escala semilogarítmica.

```
>> x=0:0.01:3;
y=abs(exp(-0.5*x).*sin(5*x));
subplot(2,2,1)
plot(x,y)
title('normal')
hold on
subplot(2,2,2)
loglog(x,y)
title('logarítmica')
subplot(2,2,3)
semilogx(x,y)
title('semilogarítmica en eje X')
subplot(2,2,4)
semilogy(x,y)
title('semilogarítmica en eje Y')
```

Seguidamente se representa un octógono regular (Figura 7-34), cuyos vértices están definidos por los pares de valores ($Sen(t)$, $Cos(t)$), para valores de t variando entre $\pi/8$ y $15\pi/8$ separados entre sí $2\pi/8$. Se usa solamente el color verde y se sitúa el texto octogono en el punto (-1/4,0) del interior de la figura.

```
>> t=(pi/8:2*pi/8:15*pi/8)';
x=sin(t); y=cos(t);
fill(x,y,'g')
axis('square')
text(-0.25,0,'OCTÓGONO')
```

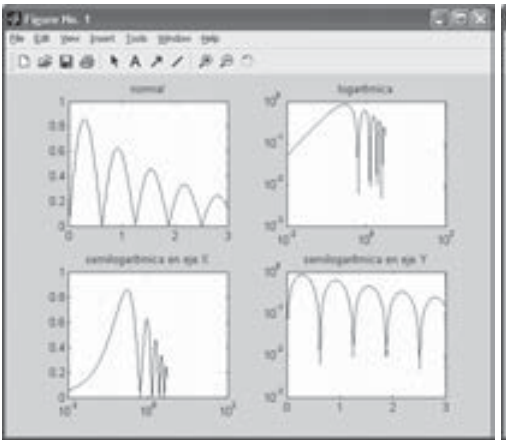


Figura 7-33

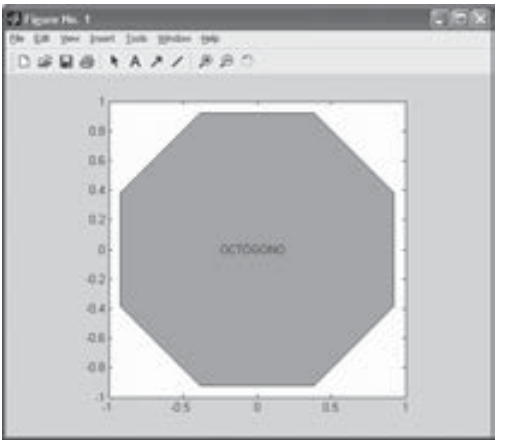


Figura 7-34

Por último se representa un octógono regular (Figura 7-33), cuyos vértices están definidos por los pares de valores (*Sen*(*t*), *Cos*(*t*)), para valores de *t* variando entre $\pi/8$ y $15\pi/8$ separados entre sí $2\pi/8$. Se usa solamente el color verde y se sitúa el texto octogono en el punto (-1/4,0) del interior de la figura.

7.5 Gráficos de líneas 3D

El módulo básico de MATLAB permite realizar gráficos en tres dimensiones, tanto de líneas como de mallas y superficies. También permite utilizar coordenadas paramétricas y explícitas. Los pasos que se siguen en general para realizar un gráfico tridimensional se presentan en la siguiente tabla:

Paso	***Ejemplo***
1. Preparar los datos	`Z = peaks(20);`
2. Seleccionar ventana y posición	`figure(1)` `subplot(2,1,2)`
3. Usar función gráfica 3-D	`h = surf(Z);`

4. Situar color y sombra	`colormap hot`
	`shading interp`
	`set(h,'EdgeColor','k')`

5. Añadir iluminación	`light('Position',[-2,2,20])`
	`lighting phong`
	`material([0.4,0.6,0.5,30])`
	`set(h,'FaceColor',[0.7 0.7 0],...`
	`'BackFaceLighting','lit')`

6. Situar punto de vista	`view([30,25])` `set(gca,'CameraViewAngleMode','Manual')`
7. Situar límites y marcas en los ejes	`axis([5 15 5 15 -8 8])` `set(gca'ZTickLabel','Negative||Positive')`
8. Situar ratio de aspecto	`set(gca,'PlotBoxAspectRatio',[2.5 2.5 1])`

9. Situar anotaciones y leyendas, gráfico y ejes	`xlabel('X Axis')`
	`ylabel('Y Axis')`
	`zlabel('Function Value')`
	`title('Peaks')`

10. Imprimir el gráfico	`set(gcf,'PaperPositionMode','auto')` `print -dps2`

En el cuadro siguiente se presentan los comandos de MATLAB más comunes en la representación de gráficos de líneas 3-D.

plot3(X,Y,Z)	*Dibuja el conjunto de puntos (X,Y,Z), donde X, Y y Z son vectores fila. X, Y y Z pueden ser coordenadas paramétricas o matrices de la misma dimensión, en cuyo caso se hace una gráfica por cada tripleta de filas y sobre los mismos ejes. Para valores complejos de X, Y y Z se ignoran las partes imaginarias*
plot3(X,Y,Z,S)	*Gráfica de plot(X,Y,Z) con las opciones definidas en S. Usualmente S se compone de dos dígitos entre comillas simples, el primero de los cuales fija el color de la línea del gráfico y el segundo el carácter a usar en el graficado. Los valores posibles de colores y caracteres se han descrito ya al explicar el comando plot*
plot3(X1,Y1,Z1,S1, X2,Y2,Z2,S2, X3,Y3,Z3,S3,...)	*Combina, sobre los mismos ejes, los gráficos definidos para las tripletas (Xi,Yi,Zi,Si). Se trata de una forma de representar varias funciones sobre el mismo gráfico*
fill3(X,Y,Z,C)	*Dibuja el polígono compacto cuyos vértices son las tripletas de componentes (Xi,Yi,Zi) de los vectores columna X, Y y Z. C es un vector de la misma dimensión de X, Y y Z, que contiene los colores Ci de cada punto (Xi,Yi,Zi). Los valores de Ci pueden ser 'r','g','b', 'c','m','y','w','k', cuyos significados ya conocemos. Si C es un solo carácter, se pintarán todos los puntos del polígono del color correspondiente al carácter. Si X, Y y Z son de la misma dimensión, se representarán simultáneamente varios polígonos correspondientes a cada tripleta de vectores columna (X,j,Y,j,Z,j). En este caso, C puede ser un vector fila cuyos elementos Cj determinan el color único de cada polígono correspondiente a la tripleta de vectores columna (X,j,Y,j,Z,j). C puede ser también una matriz de la misma dimensión que X, Y y Z, en cuyo caso sus elementos determinan los colores de cada punto (Xijk,Yijk,Zijk) del conjunto de polígonos*
fill3(X1,Y1,Z1,C1, X2,Y2, Z2, C2,...)	*Dibuja el polígono compacto cuyos vértices vienen dados por los puntos (Xi, Yi, Zi, Ci)*

Como primer ejemplo representamos la hélice paramétrica $x(t) = Sen(t)$, $y(t) = Cos(t)$, $z(t) = t$ para valores de t entre 0 y 10π separados $\pi/50$ (Figura 7-35).

```
>> t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
grid on
axis square
```

A continuación representamos un polígono compacto (Figura 7-36) mediante la sintaxis siguiente:

```
>> X = [0 1 1 2;1 1 2 2;0 0 1 1];
Y = [1 1 1 1;1 0 1 0;0 0 0 0];
Z = [1 1 1 1;1 0 1 0;0 0 0 0];
C = [0.5000 1.0000 1.0000 0.5000;
     1.0000 0.5000 0.5000 0.1667;
     0.3330 0.3330 0.5000 0.5000];
fill3(X,Y,Z,C)
```

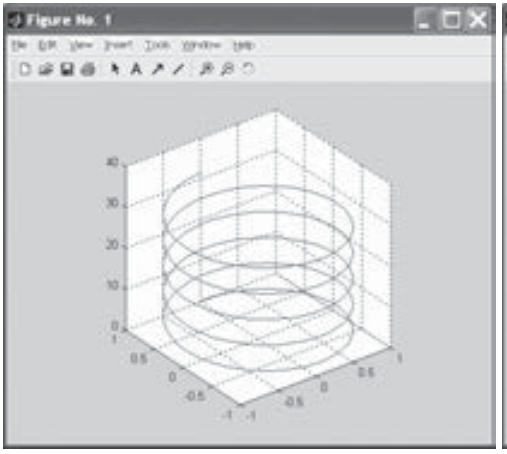


Figura 7-35

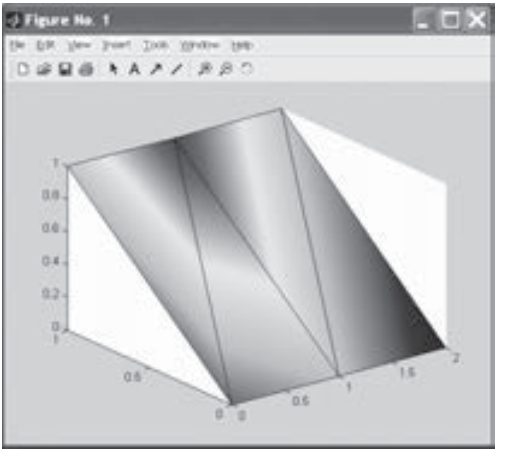


Figura 7-36

Seguidamente se representan sobre los mismos ejes las dos hélices paramétricas $x(t) = Sen(t)$, $y(t) = Cos(t)$, $z(t) = t$ y $x(t) = Cos(t)$, $y(t) = Sen(t)$, $z(t) = t$ para valores de t entre 0 y 10π separados $\pi/50$ (Figura 7-37).

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t,'-',cos(t),sin(t),t,'*')
```

En el ejemplo siguiente se presenta un polígono compacto de color rojo (Figura 7-38).

```
>> x=cos(0:0.01:8*pi);
y=sin(0:0.01:8*pi);
z=0:0.01:8*pi;
fill3(x,y,z,'r')
```

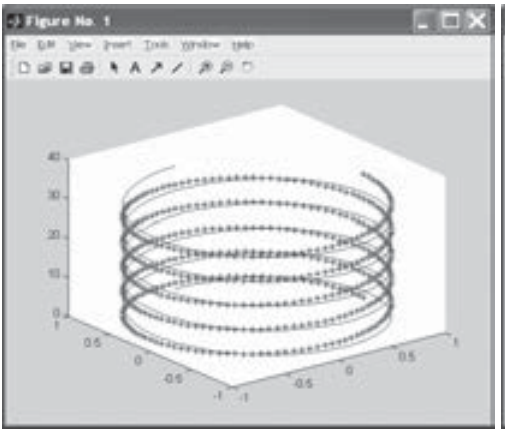


Figura 7-37

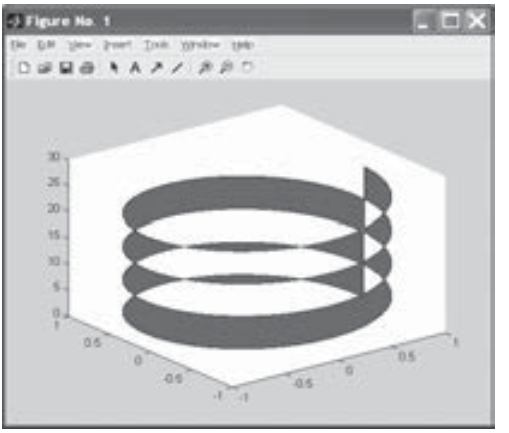


Figura 7-38

7.6 Formas geométricas 3D especiales

MATLAB dispone de comandos para representar formas geométricas especiales en tres dimensiones, como cilindros, esferas, gráficos de barras, secciones, tallos, cascada, etc. La sintaxis de estos comandos se presenta en la tabla siguiente:

bar3(Y)	*Gráfico de barras relativo al vector de frecuencias Y. Si Y es matriz se obtiene el gráfico de barras múltiple para cada fila de Y*
bar3(x,Y)	*Gráfico de barras relativo al vector de frecuencias Y siendo x un vector que define los espacios en el eje X para situar las barras*
bar3(...,anchura)	*Gráfico con anchura de las barras dada. Por defecto, la anchura es 0,8 y la anchura 1 provoca que las barras se toquen*
bar3(...,'estilo')	*Gráfico con estilo para las barras dado. Los estilos son 'detached' (estilo por defecto) 'grouped' (estilo con barras verticales agrupadas) y 'stacked' (barras apiladas).*
bar3(...,color)	*Las barras son todas del color especificado (r=rojo, g=verde, b=azul, c=cyan, m=magenta y=yellow, k=black y w=white)*
comet3(z) **comet3(x,y,z)** **comet3(x,y,z,p)**	*Gráfico de cometa relativo al vector z* *Gráfico de cometa paramétrico (x(t), y(t), z(t))* *Gráfico de cometa con cuerpo de longitud p*length(y)*
[X,Y,Z] = cylinder **[X,Y,Z] = cylinder(r)** **[X,Y,Z] = cylinder(r,n)** **cylinder(...)**	*Da las coordenadas del cilindro unidad* *Da las coordenadas del cilindro generado por la curva r* *Da las coordenadas del cilindro generado por la curva r con n puntos en la circunferencia sección horizontal alineado con el eje Z (n = 20 por defecto)* *Grafica los cilindros anteriores*
sphere	*Grafica la esfera unidad usando 20x20 caras*
sphere(n)	*Genera una esfera usando nxn caras*
[X,Y,Z] = sphere(n)	*Da las coordenadas de la esfera en tres matrices (n+1)x(n+1)*

slice(V,sx,sy,sz)	*Dibuja cortes a lo largo de las direcciones x, y , z en el volumen V (array mxnxp) definidos por los vectores (sx,sy,sz)*
slice(X,Y,Z,V,sx,sy,sz)	*Dibuja cortes definidos por los vectores (sx,sy,sz) en el volumen V definido por los arrays tridimensionales (X,Y,Z)*
slice(V,XI,YI,ZI)	*Dibuja cortes en el volumen V definidos por las matrices (XI,YI,ZI) que generan una superficie*
slice(X,Y,Z,V,XI,YI,ZI)	*Dibuja cortes definidos por las matrices (XI,YI,ZI) que generan una superficie en el volumen V definido por los arrays tridimensionales (X,Y,Z)*
slice(...,'método')	*Dibuja cortes según el método de interpolación especificado (linear para lineal, cubic para cúbica y nearest para vecino)*
stem3(Z)	*Dibuja la secuencia Z como un gráfico de tallos en el plano (x,y)*
stem3(X,Y,Z)	*Dibuja la secuencia Z en los valores especificados por X e Y*
stem3(...,'fill')	*Rellena de color los círculos de las puntas de los tallos*
stem3(...,S)	*Realiza el gráfico de tallos con las especificaciones de S (color, ...)*
waterfall(X,Y,Z)	*Dibuja un gráfico de cascada según los valores de X,Y,Z*
waterfall(Z)	*Supone* `x = 1:size(Z,1) e y = 1:size(Z,1).` *Z*
waterfall(...,C)	*Dibuja el gráfico en cascada con el mapa de colores C*
quiver3(X,Y,Z,U,V,W)	*Dibuja los vectores de componentes (u,v,w) en los puntos (x,y,z)*

Como primer ejemplo se presenta un gráfico con varios tipos de gráficos de barras tridimensionales como subgráficos (Figura 7-39).

```
>> Y= cool(7);
subplot(3,2,1)
bar3(Y,'detached')
title('SEPARADO')

subplot(3,2,2)
bar3(Y,0.25,'detached')
title('ANCHURA = 0.25')

subplot(3,2,3)
bar3(Y,'grouped')
title('AGRUPADO')

subplot(3,2,4)
bar3(Y,0.5,'grouped')
title('ANCHURA = 0.5')

subplot(3,2,5)
bar3(Y,'stacked')
title('APILADO')

subplot(3,2,6)
bar3(Y,0.3,'stacked')
title('ANCHURA = 0.3')
```

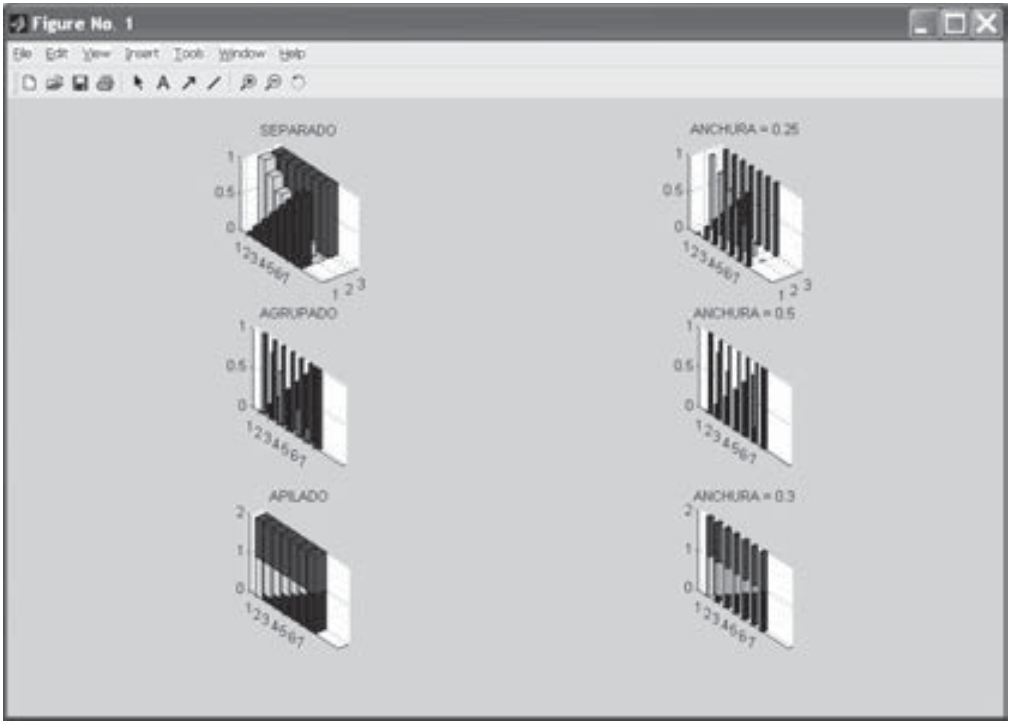


Figura 7-39

En el ejemplo siguiente se crea un gráfico de cometa tridimensional (Figura 7-40).

```
>> t = -10*pi:pi/250:10*pi;
comet3((cos(2*t).^2).*sin(t),(sin(2*t).^2).*cos(t),t);
```

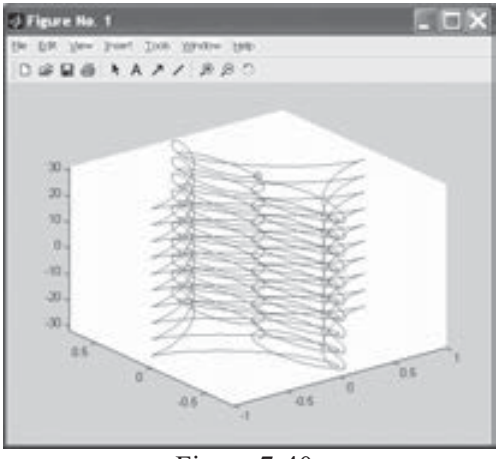


Figura 7-40

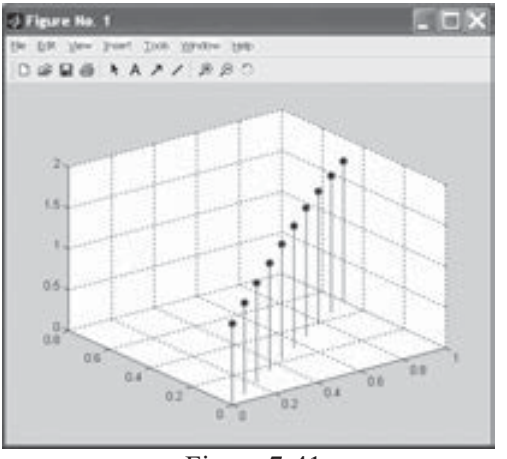


Figura 7-41

A continuación se presenta la sintaxis que crea un gráfico de tallos para visualizar una función de dos variables (Figura 7-41).

```
>> X = linspace(0,1,10);
Y = X./2;
Z = sin(X) + cos(Y);
stem3(X,Y,Z,'fill')
```

Seguidamente se dibuja la esfera unidad con ejes iguales (Figura 7-42).

```
>> sphere
axis equal
```

Por ultimo se dibuja el cilindro unidad con ejes cuadrados (Figura 7-43).

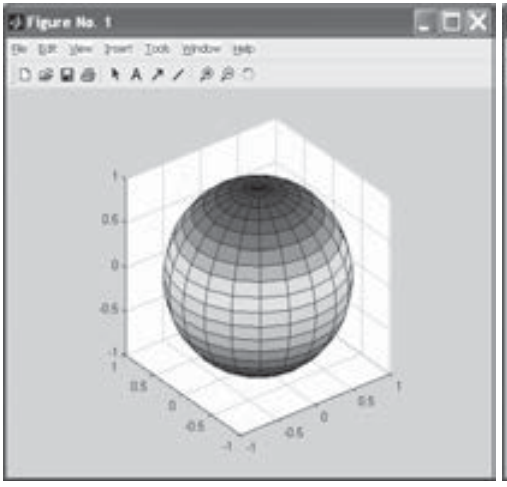


Figura 7-42

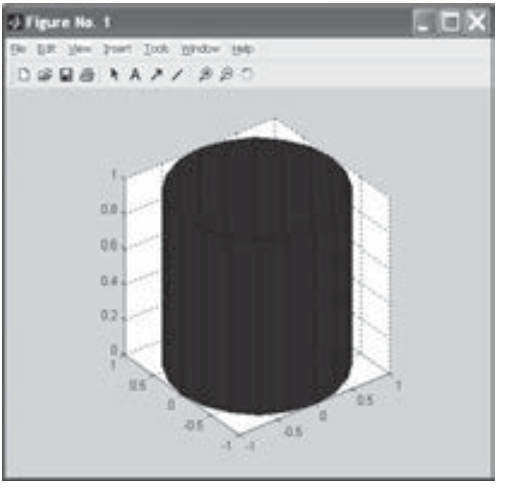


Figura 7-43

8.7 Superficies explícitas y paramétricas, mallas y contornos (curvas de nivel)

Los gráficos de superficie permiten obtener representaciones densas de figuras tridimesionales y en especial de funciones de dos variables. El primer paso para representar una función de dos variables *z=f(x,y)* mediante su gráfico de superficie es utilizar el comando *meshgrid*, que básicamente define la matriz de puntos *(X,Y)* sobre los cuales se evalúa la función de dos variables para hacer su representación gráfica. El segundo paso es utilizar los comandos disponibles al efecto que vienen dados por el comando *surf* y sus variantes.

Un gráfico tridimensional de malla viene definido por una función *z=f(x,y)*, de tal forma que los puntos de la superficie se representan sobre una rejilla, resultado de levantar los valores de *z* dados por *f(x,y)* sobre los correspondientes puntos del plano *(x,y)*. El aspecto de un gráfico de malla es como una red de pesca, con los puntos de la superficie sobre los nudos de la red. Realmente, es un gráfico de superficie cuyo grafo tiene forma de red. Para representar un gráfico de malla se utiliza el comando *mesh* y sus variantes.

Otra forma de visualizar funciones de dos variables consiste en utilizar las llamadas curvas de nivel o el sistema de planos acotados. Estas curvas se caracterizan porque son puntos *(x,y)* sobre las cuales el valor de *f(x,y)* es constante. Así, por ejemplo, en los mapas del tiempo las curvas de nivel que representan puntos de la misma temperatura se llaman isotermas, y las curvas de nivel de igual presión, isobaras. Mediante las curvas de nivel, que representan alturas [valores de *f*(*x,y*)] iguales, se pueden describir superficies en el espacio. Así, dibujando diferentes curvas de nivel correspondientes a alturas constantes se puede describir un mapa de líneas de nivel de la superficie, que MATLAB denomina *gráfico de contorno*. Los gráficos de contorno pueden representarse en dos y tres dimensiones. Un mapa que muestre regiones de la superficie terrestre, cuyas curvas de nivel representen la altura sobre el nivel del mar, se llama mapa topográfico. En estos mapas se observa, por tanto, la variación de *z = f(x,y)* con respecto a *x* y a *y*. Cuando el espacio entre las curvas de nivel es grande, significa que la variación de la variable *z* es lento, mientras que un espacio pequeño indica un cambio rápido de *z*. Los comandos que utiliza MATLAB para la representación de gráficos de contorno (curvas de nivel) son *contour* y sus variantes.

MATLAB también permite representar superficies paramétricas cuyas componentes dependen de parámetros de variación especificada. Para ello se pueden utilizar los comandos *surf* y *mesh*, definiendo adecuadamente las variables *x*, *y* y *z*. Las superficies en coordenadas implícitas, cilíndricas y esféricas son representables en MATLAB parametrizándolas previamente. En cuanto a las superficies de revolución, son siempre parametrizables, lo que permite también su representación gráfica con MATLAB.

En la tabla siguiente se presenta la sintaxis de los comandos de MATLAB citados en este apartado.

[X,Y] = meshgrid(x,y)	*Transforma el campo de definición dado de las variables x e y de la función a representar z=f(x,y) en argumentos matriciales utilizables por los comandos surf y mesh para obtener gráficos de superficie y malla, respectivamente*
surf(X,Y,Z,C)	*Representa la superficie explícita z=f(x,y) o la paramétrica x=x(t,u), y=y(t,u), z=z(t,u), realizando el dibujo con los colores especificados en C. El argumento C se puede ignorar*
surfc(X,Y,Z,C)	*Representa la superficie explícita z=f(x,y) o la paramétrica x=x(t,u), y=y(t,u), z=z(t,u), junto con el gráfico de contorno correspondiente (curvas de nivel proyectadas sobre el plano XY)*

surfl(X,Y,Z)	*Representa la superficie explícita z=f(x,y) o la paramétrica x=x(t,u), y=y(t,u), z=z(t,u), con el dibujo con sombreado*
mesh(X,Y,Z,C)	*Representa la superficie explícita z=f(x,y) o la paramétrica x=x(t,u),y=y(t,u), z=z(t,u), dibujando las líneas de la rejilla que componen la malla con los colores especificados en C (opcional)*
meshz(X,Y,Z,C)	*Representa la superficie explícita z=f(x,y) o la paramétrica x=x(t,u),y=y(t,u), z=z(t,u) con una especie de cortina o telón en la parte inferior*
meshc(X,Y,Z,C)	*Representa la superficie explícita z=f(x,y) o la paramétrica x=x(t,u),y=y(t,u), z=z(t,u) junto con el gráfico de contorno correspondiente (curvas de nivel proyectadas sobre el plano XY)*
contour(Z)	*Dibuja el gráfico de contorno (curvas de nivel) para la matriz Z. El número de líneas de contorno a utilizar se elige automáticamente*
contour(Z,n)	*Dibuja el gráfico de contorno (curvas de nivel) para la matriz Z usando n líneas de contorno*
contour(x,y,Z,n)	*Dibuja el gráfico de contorno (curvas de nivel) para la matriz Z usando en los ejes X e Y el escalado definido por los vectores x e y (n líneas de contorno)*
contour3(Z), contour3(Z,n) y contour3(x,y,Z,n)	*Dibujan los gráficos de contorno en 3 dimensiones*
contourf(…)	*Dibuja un gráfico de contorno y rellena las áreas entre las isolíneas*
pcolor(X,Y,Z)	*Dibuja un gráfico de contorno (curvas de nivel) para la matriz (X,Y,Z) utilizando una representación basada en densidades de colores. Suele denominarse gráfico de densidad*
trimesh(Tri,X,Y,Z,C)	*Muestra triángulos definidos en la matriz Tri como una malla. Cada fila de la matriz Tri define una cara triangular simple y C define los colores como en surf. El argumento C es opcional.*
trisurf(Tri,X,Y,Z,C)	*Muestra triángulos definidos en la matriz Tri como una superficie. Cada fila de la matriz Tri define una cara triangular simple y C define los colores como en surf. El argumento C es opcional.*

Como primer ejemplo representamos la superficie (Figura 7-44), cuya ecuación explícita es la siguiente:

$$z = \frac{Sen\left(\sqrt{x^2 + y^2}\right)}{\sqrt{x^2 + y^2}}$$

```
>> [X,Y]=meshgrid(-7.5:.5:7.5);
Z=sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
surf(X,Y,Z)
```

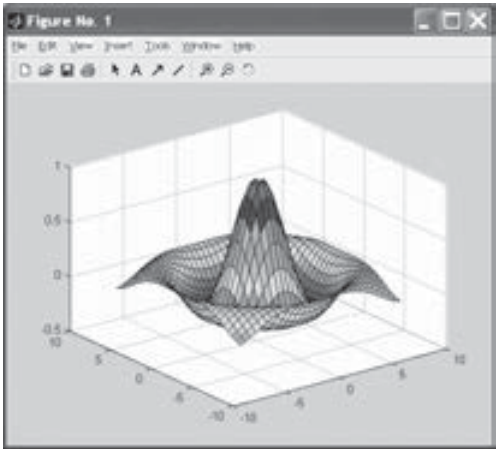


Figura 7-44

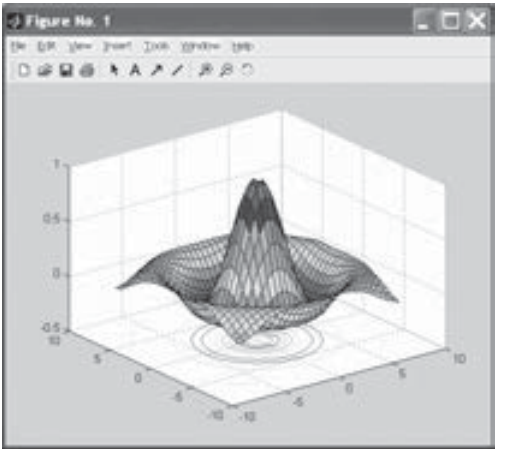


Figura 7-45

A continuación representamos la superficie anterior acompañada de sus curvas de nivel (Figura 7-45).

```
>> [X,Y]=meshgrid(-7.5:.5:7.5);
Z=sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
surfc(X,Y,Z)
```

Ahora representamos el gráfico de malla relativo a la superficie anterior (Figura 7-46).

```
>> [X,Y]=meshgrid(-7.5:.5:7.5);
Z=sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
mesh(X,Y,Z)
```

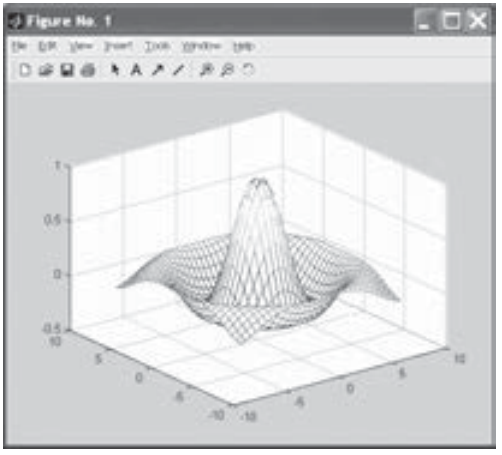


Figura 7-46

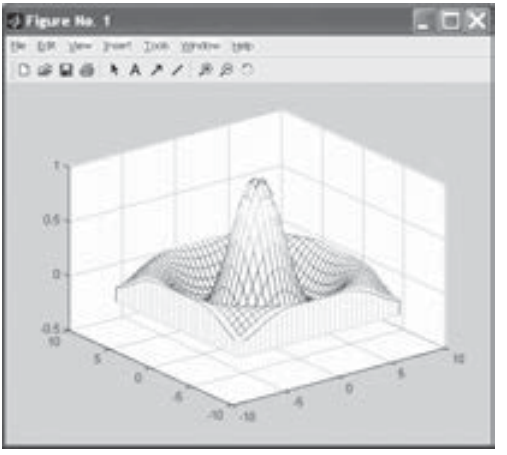


Figura 7-47

A continuación representamos el gráfico de malla anterior con la opción de cortina o telón inferior (Figura 7-47).

```
>> [X,Y]=meshgrid(-7.5:.5:7.5);
Z=sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
meshz(X,Y,Z)
```

El siguiente gráfico realiza las curvas de nivel bidimensionales para la superficie anterior (Figura 7-48).

```
>> [X,Y]=meshgrid(-7.5:.5:7.5);
Z=sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
contour(Z)
```

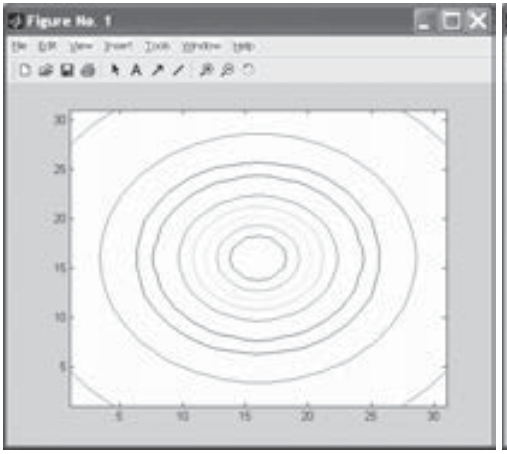


Figura 7-48

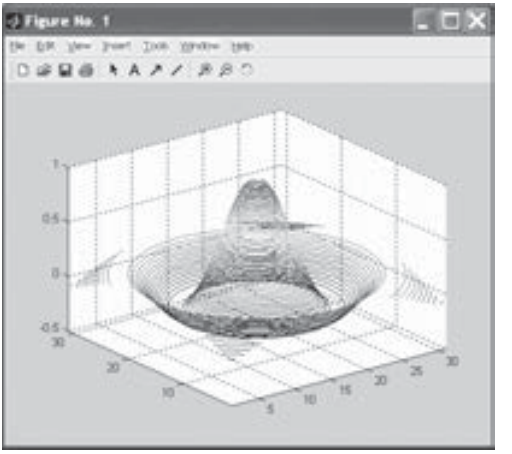


Figura 7-49

A continuación se realizan las curvas de nivel tridimensionales para la superficie anterior (Figura 7-49).

```
>> [X,Y]=meshgrid(-7.5:.5:7.5);
Z=sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
contour3(Z,50)
```

Posteriormente representamos el cilindro (Figura 7-50) en coordenadas paramétricas siguiente: $x(t) = t, y(t) = Sen(t), z(t) = u$ cuando t varía en $[0,2\pi]$ y u varía en $[0,4]$.

```
>> t=(0:0.1:2*pi)';
r=(0:0.1:4);
X=sin(t)*ones(size(r));
Y=cos(t)*ones(size(r));
Z=ones(1,size(t))'*r;
surf(X,Y,Z)
```

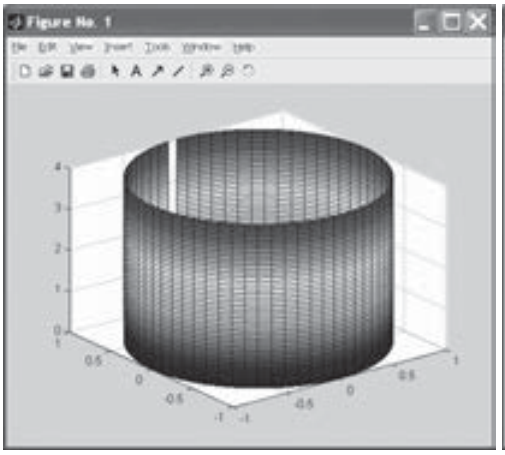


Figura 7-50

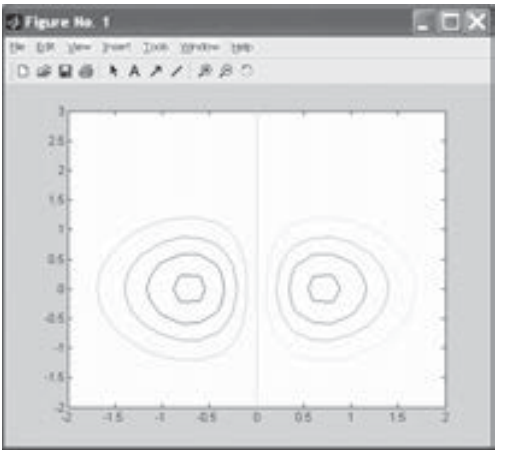


Figura 7-51

Sucesivamente se realiza un gráfico de contorno (Figura 7-51) para la superficie $z = xe^{-x^2-y^2}$.

```
>> [X,Y] = meshgrid(-2:.2:2,-2:.2:3);
Z = X.*exp(-X.^2-Y.^2);
>> contour(X,Y,Z);
```

Podemos aumentar el número de líneas del contorno anterior (Figura 7-52).

```
>> [X,Y] = meshgrid(-2:.2:2,-2:.2:3);
Z = X.*exp(-X.^2-Y.^2);
contour(X,Y,Z,50)
```

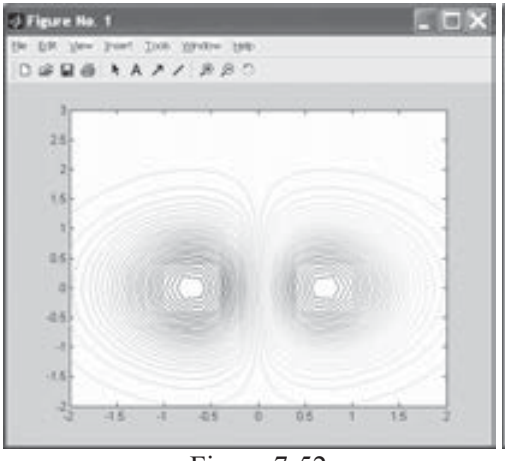


Figura 7-52

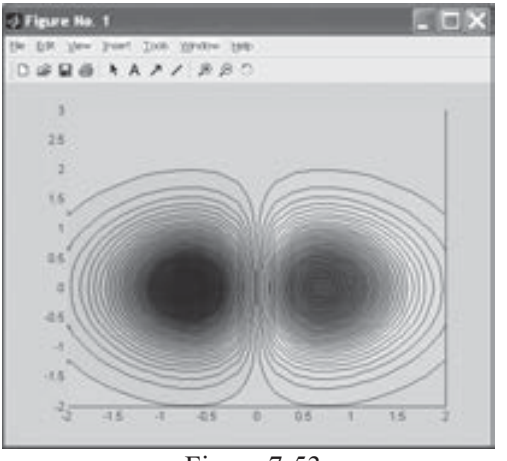


Figura 7-53

También podemos conseguir un gráfico de contorno relleno (Figura 7-53).

```
>> [X,Y] = meshgrid(-2:.2:2,-2:.2:3);
Z = X.*exp(-X.^2-Y.^2);
contourf(X,Y,Z,50)
```

7.8 Opciones de manejo de gráficos 3D

MATLAB dispone de un grupo de comandos que permiten cambiar la apariencia de un gráfico (sombreado, escala de sus ejes, colores, líneas ocultas, el punto de vista desde el que se observa, etc.). Por otra parte, las características gráficas tratadas hasta ahora pertenecen a la interfaz gráfica de alto nivel de MATLAB. Sin embargo, existen comandos de bajo nivel (*Handle Graphics*) que permiten crear y manipular como objetos las figuras, ejes, líneas, superficies, imágenes, texto, menús y otros objetos gráficos. A continuación se presentan algunos de los comandos relativos a estos temas:

axis([xmin xmax ymin ymax zmin zmax])	*Sitúa los intervalos de variación de los ejes en los valores indicados. También acepta las opciones 'ij', 'square', 'equal', etc, idénticas a las ya vistas para dos dimensiones*
view([x,y,z])	*Sitúa el punto de vista de la figura en el punto de coordenadas cartesianas (x,y,z)*
view([az, el])	*Sitúa el ángulo de vista de la figura en el punto de azimut (rotación horizontal) 'az' y elevación (elevación vertical) 'el'*
hidden	*Controla la presencia de las líneas ocultas en el gráfico. Dichas líneas aparecen con hidden on y desaparecen con hidden off*
shading	*Controla el tipo de sombreado de una superficie creada con los comandos surf, mesh, pcolor, fill y fill3. La opción shading flat sitúa un sombreado suave, la opción shading interp sitúa un sombreado denso y la opción shading faceted (opción por defecto) sitúa un sombreado normal*
colormap(M)	*Sitúa la matriz M como el mapa corriente de colores. M debe tener tres columnas y contener valores sólo entre 0 y 1. También puede ser una matriz cuyas filas sean vectores RGB del tipo [r g b]. Existen en MATLAB matrices M ya definidas, que son las siguientes: bone(p), contrast(p), cool(p), copper(p), flag(p), gray(p), hsv(p), hot(p), jet(p), pink(p), prism(p) y white(p). Todas las matrices tienen 3 columnas y p filas. Por ejemplo, la sintaxis colormap(hot(8)) sitúa la matriz hot(8) como el mapa corriente de colores (sistema completo de colores de la figura actual)*
brighten(p)	*Ajusta la iluminación de la figura. Si 0<p<1, la figura será brillante, y si -1<p<0, la figura será oscura. La variación de p es el intervalo (-1,1), y a medida que los valores de p se acercan a -1, la figura se oscurece, mientras que a medida que los valores de p se acercan a 1, la figura se ilumina*

image(A)	*Produce una imagen bidimensional con brillos proporcionales a los elementos de la matriz A, y se usa para mostrar fotografías y dibujos adaptados a los brillos dados en la matriz A. Cada elemento (m,n) de la matriz A afecta a una celda del dibujo*
pcolor(A)	*Produce una figura bidimensional con colores proporcionales a los elementos de la matriz A, y se utiliza para mostrar objetos matemáticos abstractos con colores variados. Cada elemento (m,n) de la matriz A afecta a una rejilla de la figura*
caxis([cmin cmax])	*Sitúa los valores mínimo y máximo de la escala de colores (definida por colormap e intrínsecamente relacionada con las divisiones que se hacen en los ejes, vía las rejillas) para un gráfico. Por lo tanto, permite utilizar sólo un subconjunto de colores del definido por colormap para la figura*
figure(h) o h=figure	*Crea la figura como un objeto de nombre h, y la sitúa como figura corriente. Se utiliza el comando gcf(h) para referir cualquier propiedad a la figura h. El comando close(h) cierra la figura h. El comando whitebg(h) cambia el color del fondo de la figura h. El comando clf cierra la figura corriente. El comando graymon sitúa la escala de grises. El comando newplot determina los ejes para hacer una nueva figura. El comando refresh redibuja la figura*
axes(e) o e=axes	*Crea los ejes como un objeto de nombre e en la figura corriente. Se utiliza el comando gca(e) para referir cualquier propiedad a los ejes e. Se utiliza el comando cla para borrar todos los objetos referentes a los ejes corrientes*
l=line(x,y) o l=line(x,y,z)	*Crea, como un objeto de nombre l, la línea que une los puntos (X,Y) en el plano o (X,Y,Z) en el espacio*
p=patch(X,Y,C) o patch(X,Y,Z,C)	*Crea un área poligonal opaca que está definida por el conjunto de puntos (X,Y) en el plano o (X,Y,Z) en el espacio, y cuyo color está dado por C, como un objeto de nombre p*
s=surface(X,Y,Z,C)	*Crea la superficie paramétrica definida por X, Y y Z y cuyo color está dado por C como un objeto de nombre s*
i=image(C)	*Crea la imagen definida por los colores dados en la matriz C como un objeto de nombre i*
t=text(x,y,'cadena') o t=text(x,y,z,'cadena')	*Crea el texto definido por la cadena, localizado en el punto (x,y) del plano, o en el punto (x,y,z) del espacio*
set(h, 'propiedad1', 'propiedad2', ...)	*Sitúa las propiedades especificadas en el objeto h (gca para límites de ejes, gcf para colores, gco, gcbo, gcbd, etc.)*
get(h, 'propiedad')	*Devuelve el valor corriente de la propiedad dada para el objeto h*
object=gco	*Devuelve el objeto actual de la figura actual*
rotate(h, [a, e], α, [p,q,r])	*Rota el objeto h un ángulo α, según los ejes de azimut a y elevación e, siendo el origen el punto (p,q,r)*
reset(h)	*Actualiza todas las propiedades asignadas al objeto h y coloca sus propiedades por defecto*
delete(h)	*Borra el objeto h*

Las propiedades más típicas a situar en los objetos son las siguientes:

Objeto	*Propiedades*	*Posibles valores*
Figure	*Color (color de fondo)*	*'y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'*
	ColorMap (color de mapa)	*hot(p), gray(p), pink(p), ...*
	Position (posición en pantalla)	*[left, botton, width, height]*
	Name (nombre)	*cadena con el nombre*
	MinColorMap (n.º mín. de color)	*n.º mínimo de colores para el mapa*
	NextPlot (modo del gráf. siguien.)	*new, add, replace*
	NumberTitle (n.º en la figura)	*on, off*
	Units (unidades de medida)	*pixels, inches, centimeters, points*
	Resize (tamaño figura con ratón)	*on (se puede cambiar), off (no)*
Axes	*Box (caja para el gráfico)*	*on, off*
	Color (color de los ejes)	*'y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'*
	GridLineStyle (línea para malla)	*'-', '--', ':', '-.'*
	Position (posición en pantalla)	*[left, botton, width, height]*
	TickLength (long. entre marcas)	*un valor numérico*
	TickDir (dirección de las marcas)	*in, out*
	Units (unidades de medida)	*pixels, inches, centimeters, points*
	View (punto de vista)	*[azimut, elevación]*
	FontAngle (ángulo de fuente)	*normal, italic, oblique*
	FontName (nombre de fuente)	*texto con el nombre de la fuente*
	FontSize (tamaño de fuente)	*valor numérico*
	FontWeight (peso de la fuente)	*light, normal, demi, bold*
	DrawMode (modo de dibujo)	*normal, fast (rápido)*
	Xcolor, Ycolor, Zcolor (color ejes)	*[min, max]*
	XDir,YDir, ZDir (dirección ejes)	*normal (crecen de izq. a der.), reverse*
	XGrid,YGrid, Zgrid (rejillas)	*on, off*
	XLabel,YLabel, Zlabel (etiquetas)	*cadena con el texto de las etiquetas*
	XLim,YLim,Zlim (valores límites)	*[min, max] (intervalo de variación)*
	XScale,YScale,ZScale (escalas)	*linear (lineal), log(logarítmica)*
	XTick,YTick,ZTick (marcas)	*[m1,m2,...] (situación marcas en eje)*
Line	*Color (color de la línea)*	*'y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'*
	LineStyle (estilo de línea)	*'-', '--', ':', '-.', '+', '*', '.', 'x'*
	LineWidth (anchura de línea)	*valor numérico*
	Visible (línea visible o no en pant.)	*on, off*
	Xdata,Ydata,Zdata (coordenad.)	*conjunto de coordenadas de la línea*
Text	*Color (color de texto)*	*'y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'*
	FontAngle (ángulo de fuente)	*normal, italic, oblique*
	FontName (nombre de fuente)	*texto con el nombre de la fuente*
	FontSize (tamaño de fuente)	*valor numérico*
	FontWeight (peso de la fuente)	*light, normal, demi, bold*
	HorizontalAlignment (ajuste hor.)	*left, center, right*
	VerticalAlignment (ajuste vert.)	*top, cap, middle, baseline, bottom*
	Position (posición en pantalla)	*[x, y, z] (punto de situación)*

	Rotation (orientación del texto)	*0, ±90, ±180, ±270*
	Units (unidades de medida)	*pixels, inches, centimeters, points*
	String (cadena de texto a situar)	*cadena con el texto*
Surface	*Cdata (color de cada punto)*	*matriz de colores*
	Edgecolor (color de rejillas)	*'y', 'm',..., none, flat, interp*
	Facecolor (color de las caras)	*'y', 'm',..., none, flat, interp*
	LineStyle (estilo de línea)	*'-', '--', ':', '-.', '+', '*', '.', 'x'*
	LineWidth (anchura de línea)	*valor numérico*
	MeshStyle (líneas en filas y col.)	*row, colum, both*
	Visible (línea visible o no en pant.)	*on, off*
	Xdata,Ydata,Zdata (coordenad.)	*conjunto de coordenadas de la superficie*
Patch	*Cdata (color de cada punto)*	*matriz de colores*
	Edgecolor (color de los ejes)	*'y', 'm',..., none, flat, interp*
	Facecolor (color de las caras)	*'y', 'm',..., none, flat, interp*
	LineWidth (anchura de línea)	*valor numérico*
	Visible (línea visible o no en pant.)	*on, off*
	Xdata,Ydata,Zdata (coordenad.)	*conjunto de coordenadas de la superficie*
Image	*Cdata (color de cada punto)*	*matriz de colores*
	Xdata,Ydata (coordenadas)	*conjunto de coordenadas de la imagen*

Como primer ejemplo consideramos la superficie $z = x^2 - y^2$ en [-2,2]x[-2,2] y la representamos con iluminación fuerte, sombreado denso y colores grisáceos (Figura 7-54).

```
>> [X,Y]=meshgrid(-2:0.05:2);
Z=X.^2-Y.^2;
surf(X,Y,Z),shading interp,brighten(0.75),colormap(gray(5))
```

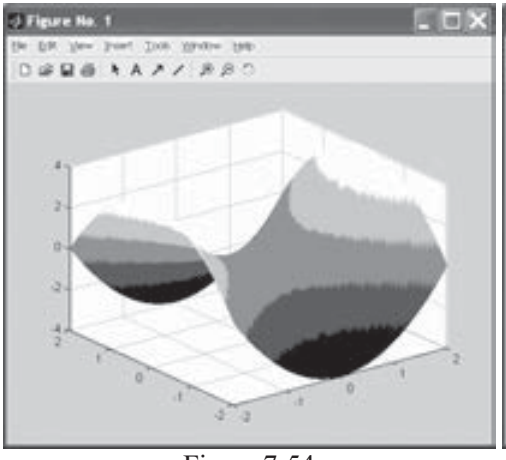


Figura 7-54

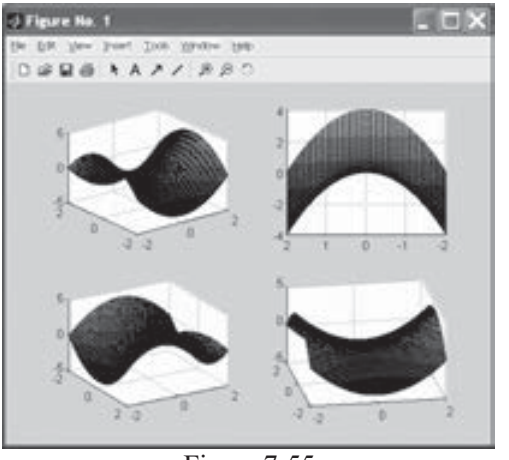


Figura 7-55

A continuación representamos sobre los mismos ejes la curva enfocada desde cuatro puntos de vista distintos y con el sombreado por defecto (Figura 7-55).

```
>> [X,Y]=meshgrid(-2:0.05:2);
Z=X.^2-Y.^2;
subplot(2,2,1)
surf(X,Y,Z)
subplot(2,2,2)
surf(X,Y,Z),view(-90,0)
subplot(2,2,3)
surf(X,Y,Z),view(60,30)
subplot(2,2,4)
surf(X,Y,Z),view(-10,30)
```

Seguidamente se lee un fichero y se utilizan propiedades adecuadas para generar el gráfico vistoso de la Figura 7-56.

```
>> load clown
surface(peaks,flipud(X),...
        'FaceColor','texturemap',...
        'EdgeColor','none',...
        'CDataMapping','direct')
colormap(map)
view(-35,45)
```

A continuación se usan distintos sombreados para una esfera (Figura 7-57).

```
>> subplot(3,1,1)
sphere(16)
axis square
shading flat
title('Sombreado suave')

subplot(3,1,2)
sphere(16)
axis square
shading faceted
title('Sombreado normal')

subplot(3,1,3)
sphere(16)
axis square
shading interp
title('Sombreado denso')
```

En el ejemplo siguiente se cambia el ratio de aspecto para la esfera unidad (Figura 7-58).

```
>> sphere
set(gca,'DataAspectRatio',[1 1 1],...
        'PlotBoxAspectRatio',[1 1 1],'ZLim',[-0.6 0.6])
```

Ahora situamos el color del fondo de la figura corriente en blanco (Figura 7-59).

```
>> set(gcf, 'Color', 'w')
```

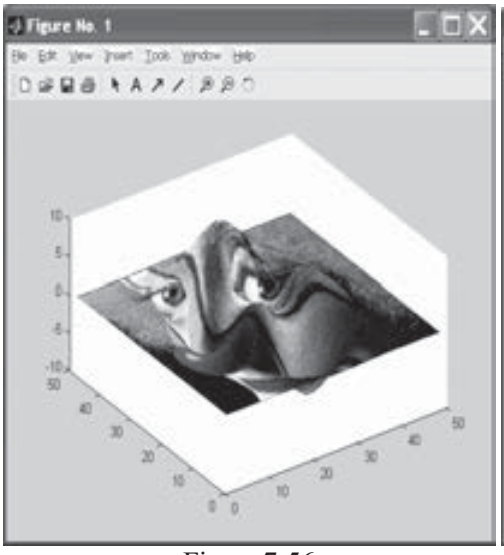


Figura 7-56

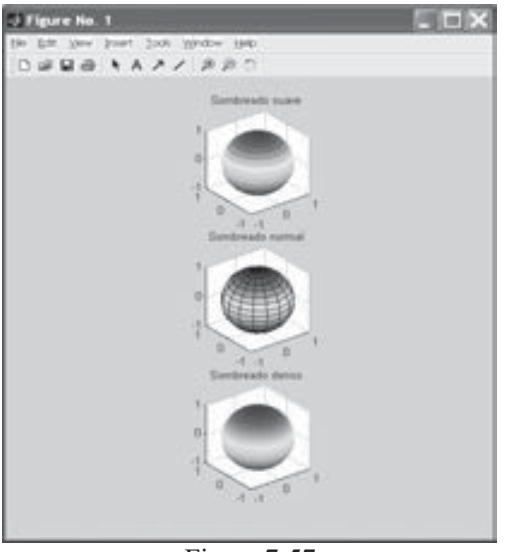


Figura 7-57

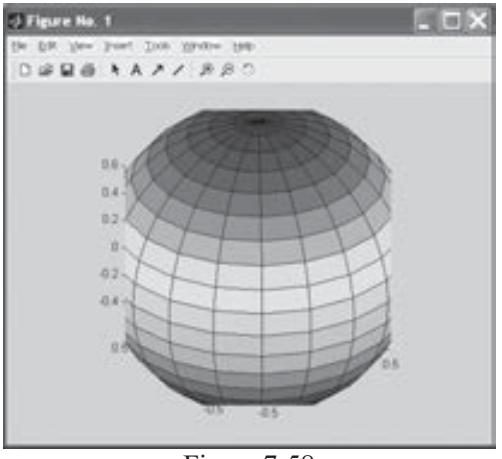


Figura 7-58

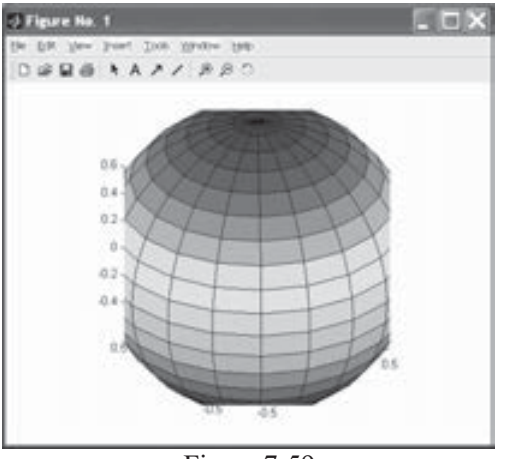


Figura 7-59

En el ejemplo siguiente (Figura 7-60) representamos una superficie (función *peaks* predefinida en MATLAB similar a una gaussiana bidimensional con cambio de origen y escala).

```
>> surf(peaks(20))
```

A continuación rotamos la figura anterior 180º alrededor del eje X (Figura 7-61).

```
>> h = surf(peaks(20));
rotate(h,[1 0 0],15)
```

Posteriormente cambiamos el centro y rotamos la superficie inicial 45º en la dirección del eje *z* (Figura 7-62).

```
>> h = surf(peaks(20));
zdir = [0 0 1];
center = [10 10 0];
rotate(h,zdir,45,center)
```

En el ejemplo siguiente definimos varios ejes en una ventana simple (Figura 7-63).

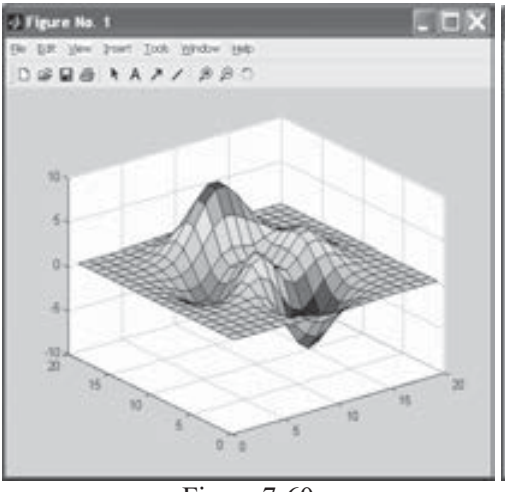


Figura 7-60

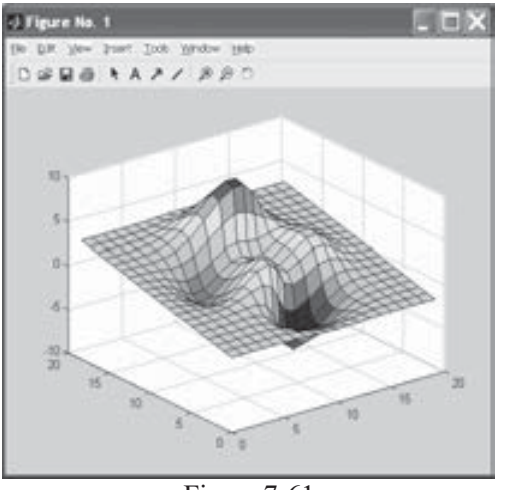


Figura 7-61

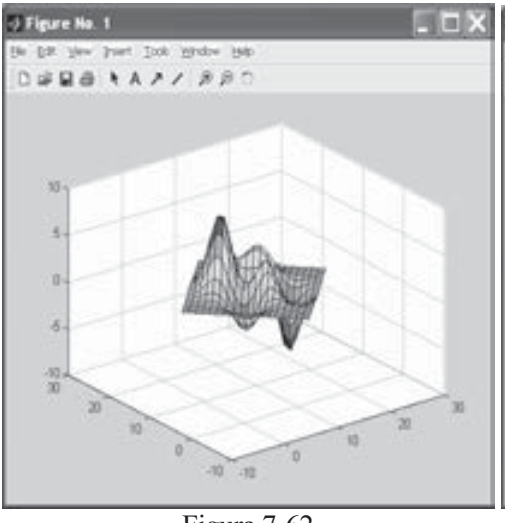


Figura 7-62

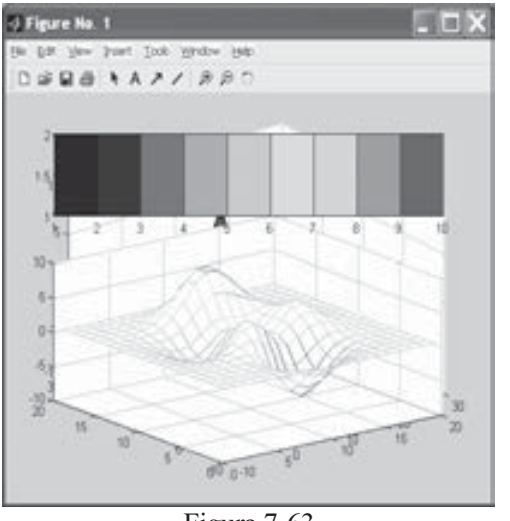


Figura 7-63

Seguidamente dotamos de iluminación especial, sombreado denso gris y variación de ejes en [-3,3]x[-3,3]x[-8,8] a la superficie *peaks* (Figura 7-64).

```
>> [x,y] = meshgrid(-3:1/8:3);
z = peaks(x,y);
surfl(x,y,z);
shading interp
colormap(gray);
axis([-3  3  -3  3  -8  8])
```

Después cambiamos punto de vista, malla y color a la superficie iluminada anterior (Figura 7-65).

```
>> view([10 10])
grid on
hold on
surfl(peaks)
shading interp
colormap copper
hold off
```

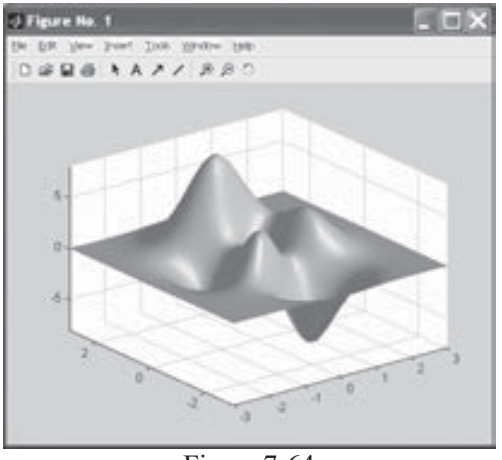


Figura 7-64

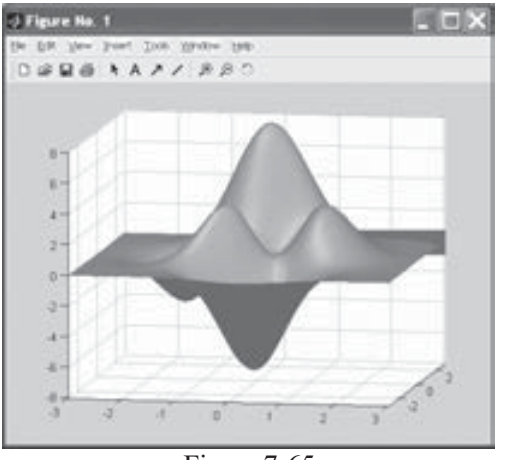


Figura 7-65

A continuación se presentan dos ejemplos de gráficos de triángulos (Figuras 7-66 y 7-67).

```
>> x = rand(1,50);
y = rand(1,50);
z = peaks(6*x-3,6*x-3);
tri = delaunay(x,y);
trimesh(tri,x,y,z)
```

```
>> x = rand(1,50);
y = rand(1,50);
z = peaks(6*x-3,6*x-3);
tri = delaunay(x,y);
trisurf(tri,x,y,z)
```

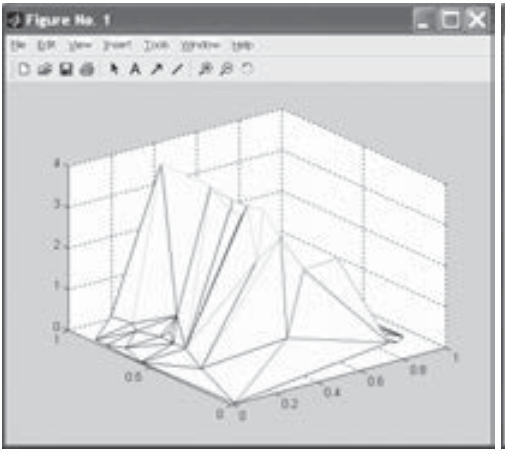


Figura 7-66

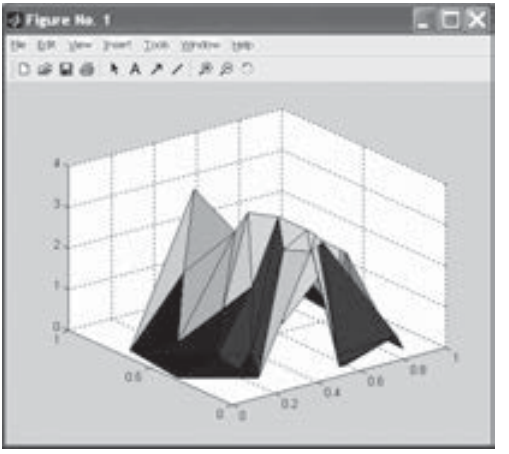


Figura 7-67

7.9 Visualización de volúmenes

MATLAB dispone de un grupo de comandos que permiten representar distintos tipos de volúmenes. A continuación se presenta la sintaxis de los más importantes.

coneplot(X,Y,Z,U,V,W,Cx,Cy,Cz) **coneplot(U,V,W,Cx,Cy,Cz)** **coneplot(...,s)** **coneplot(...,color)** **coneplot(...,'quiver')** **coneplot(...,'método')** **coneplot(X,Y,Z,U,V,W,'nointerp')** **h = coneplot(...)**	*Grafica vectores velocidad como conos en campos vectoriales 3-D. (X,Y,Z) definen las coordenadas del vector campo. (U,V,W) definen el vector campo. (Cx, Cy, Cz) definen la localización de los conos en el vector campo. Además pueden definirse como argumentos color, método de interpolación, etc.*
contourslice(X,Y,Z,V,Sx,Sy,Sz) **contourslice(X,Y,Z,V,Xi,Yi,Zi)** **contourslice(V,Sx,Sy,Sz),** **contourslice(V,Xi,Yi,Zi)** **contourslice(...,n)** **contourslice(...,cvals)** **contourslice(...,[cv cv])** **contourslice(...,'método')** **h = contourslice(...)**	*Dibuja contornos en cortes planos de volúmenes según los vectores (Sx, Sy, Sz). X, Y, Z definen las coordenadas para el volumen V. Xi,Yi,Zi definen la superficie a lo largo de la cual se dibuja el contorno a través del volumen V. Se puede usar un método de interpolación y un número n de líneas de contorno por plano. El valor cv indica contorno simple por plano*

[curlx,curly,curlz,cav] = curl(X,Y,Z,U,V,W) **[curlx,curly,curlz,cav] = curl(U,V,W)** **[curlz,cav]= curl(X,Y,U,V)** **[curlz,cav]= curl(U,V)** **[curlx,curly,curlz] = curl(...),** **[curlx,curly] = curl(...)** **cav = curl(...)**	*Halla el rotacional curl y la velocidad angular cav de un campo vectorial. Los arrays (X,Y,Z) definen las coordenadas para el vector (U,V,W) del campo. En el caso de curl(U,V,W) se tiene que [X Y Z] = meshgrid(1:n,1:m,1:p) con [m,n,p] = size(V)*
div = divergence(X,Y,Z,U,V,W) **div = divergence(U,V,W)** **div = divergence(X,Y,U,V)** **div = divergence(U,V)**	*Halla la divergencia de un campo vectorial*
interpstreamspeed(X,Y,Z,U,V,W,vertices) **interpstreamspeed(U,V,W,vertices)** **interpstreamspeed(X,Y,Z,speed,vertices)** **interpstreamspeed(speed,vertices)** **interpstreamspeed(X,Y,U,V,vertices)** **interpstreamspeed(U,V,vertices)** **interpstreamspeed(X,Y,speed,vertices)** **interpstreamspeed(speed,vertices)** **interpstreamspeed(...,sf)** **vertsout = interpstreamspeed(...)**	*Interpola vértices de magnitudes de campos vectoriales*
fvc = isocaps(X,Y,Z,V,isovalor) **fvc = isocaps(V,isovalue)** **fvc = isocaps(...,'*enclose*')** **fvc = isocaps(...,'*whichplane*')** **[f,v,c] = isocaps(...)** **isocaps(...)**	*Halla isosuperficies del volumen V en el valor isovalor. X, Y, Z dan las coordenadas del volumen V. Si no se da (X,Y,Z) se supone [X,Y,Z] = meshgrid(1:n,1:m,1:p) con [m,n,p] = size(V). El valor enclose (cierre) puede ser above 8 (superior) o below (inferior). El valor whichplane indica el plano de dibujo (all, xmin, xmax, ymin, ymax, zmin, o zmax)*
nc = isocolors(X,Y,Z,C,vertices) **nc = isocolors(X,Y,Z,R,G,B,vertices)** **nc = isocolors(C,vertices)** **nc = isocolors(R,G,B,vertices)** **nc = isocolors(...,PatchHandle)** **isocolors(...,PatchHandle)**	*Halla los colores de los vértices de las isosuperficies usando los valores de los colores dados en C o RGB. Pueden usarse los vértices identificados en PatchHandle*
n = isonormals(X,Y,Z,V,vertices) **n = isonormals(V,vertices)** **n = isonormals(V,p),** **n = isonormals(X,Y,Z,V,p)** **n = isonormals(...,'negate')** **isonormals(V,p), isonormals(X,Y,Z,V,p)**	*Computa normales de vértices de isosuperficies. La opción negate cambia la dirección de las normales y p es un identificador del objeto vértice. Si no aparece el argumento n, se dibujan las isonormales*

fv = isosurface(X,Y,Z,V,isovalor) **fv = isosurface(V,isovalue)** **fv = isosurface(X,Y,Z,V),** **fv = isosurface(X,Y,Z,V)** **fvc = isosurface(...,colors)** **fv = isosurface(...,'noshare')** **fv = isosurface(...,'verbose')** **[f,v] = isosurface(...)** **isosurface(...)**	*Extrae datos de isosuperficies del volumen V en el valor especificado en isovalor. El argumento noshare indica la no creación de vértices compartidos y verbose imprime mensajes de proceso*
reducepatch(p,r) **nfv = reducepatch(p,r)** **nfv = reducepatch(fv,r)** **reducepatch(...,'fast')** **reducepatch(...,'verbose')** **nfv = reducepatch(f,v,r)** **[nf,nv] = reducepatch(...)**	*Reduce el número de caras del objeto con identificador p con un factor de reducción r. El argumento fase indica que no se computan vértices compartidos; nf y nv indica que se devuelven caras y vértices, y nfv indica un factor de reducción de 0,5*
[nx,ny,nz,nv] = **reducevolume(X,Y,Z,V,[Rx,Ry,Rz])** **[nx,ny,nz,nv] =** **reducevolume(V,[Rx,Ry,Rz])** **nv = reducevolume(...)**	*Reduce el número de elementos en el conjunto de datos del volumen V cuyas coordenadas son los arrays (X,Y,Z). Se retienen [Rx,Ry,Rz]) en las direcciones x,y,z respectivamente*
shrinkfaces(p,sf) **nfv = shrinkfaces(p,sf)** **nfv = shrinkfaces(fv,sf)** **shrinkfaces(p), shrinkfaces(fv)** **nfv = shrinkfaces(f,v,sf)** **[nf,nv] = shrinkfaces(...)**	*Contrae el tamaño de las caras del objeto con identificador p según un factor de contracción sf. Si se usa fv, se utilizan las caras y vértices de la estructura fv.*
W = smooth3(V) **W = smooth3(V,'*filter*')** **W = smooth3(V,'*filter*',size)** **W = smooth3(V,'*filter*',size,sd)**	*Suavizado de los datos del volumen V Suavizado con filtro gaussian o box. Se puede usar un escalar de suavizado (size) y una desviación típica (sd)*
XY = stream2(x,y,u,v,startx,starty) **XY = stream2(u,v,startx,starty)**	*Halla líneas de corriente 2D del vector de datos (u,v) cuyas coordenadas se definen por los arrays (x,y)*
XYZ = **stream3(X,Y,Z,U,V,W,startx,starty,startz)** **XYZ = stream3(U,V,W,startx,starty,startz)**	*Halla líneas de corriente 3D del vector de datos (u,v,w) cuyas coordenadas se definen por los arrays (x,y,z)*
h = streamline(X,Y,Z,U,V,W,startx,starty,startz) **h = streamline(U,V,W,startx,starty,startz)** **h = streamline(XYZ)** **h = streamline(X,Y,U,V,startx,starty)** **h = streamline(U,V,startx,starty)** **h = streamline(XY)**	*Dibuja líneas de corriente para vectores de datos 2D o 3D. (X,Y,Z) son las coordenadas de (U,V,W) y (startx,starty,startz) indican las posiciones de comienzo de las líneas*

streamparticles(vertices) **streamparticles(vertices,n)**	*Dibuja partículas del vector del campo de datos y las representa por marcadores en vértices 2D o 3D.*
streamribbon(X,Y,Z,U,V,W,startx,starty,startz) **streamribbon(U,V,W,startx,starty,startz)** **streamribbon(vertices,X,Y,Z,cav,speed)** **streamribbon(vertices,cav,speed)** **streamribbon(vertices,twistangle)** **streamribbon(...,width), h = streamribbon(...)**	*Dibuja cintas del vector de datos de volumen (U,V,W) cuyas coordenadas son los arrays (X,Y,Z), (startx,starty,startz) indican las posiciones de comienzo de las cintas*
streamslice(X,Y,Z,U,V,W,startx,starty,startz) **streamslice(U,V,W,startx,starty,startz)** **streamslice(X,Y,U,V)** **streamslice(U,V)** **streamslice(...,density)** **streamslice(...,'arrowmode')** **streamslice(...,'method')** **h = streamslice(...)** **[vertices arrowvertices] = streamslice(...)**	*Dibuja líneas de corriente igualmente espaciadas con flechas de dirección del vector de datos de volumen (U,V,W) cuyas coordenadas son los arrays (X,Y,Z), (startx,starty,startz) indican las posiciones de comienzo de las líneas. Density es un número positivo que modifica el espaciado de las líneas, arrowsmode puede valer arrows o noarrows para situar o no puntas en las flechas, y método indica el tipo de interpolación (linear, cubic o nearest)*
streamtube(X,Y,Z,U,V,W,startx,starty,startz) **streamtube(U,V,W,startx,starty,startz)** **streamtube(vertices,X,Y,Z,divergence)** **streamtube(vertices,divergence)** **streamtube(vertices,width)** **streamtube(vertices)** **streamtube(...,[scale n])** **h = streamtube(...)**	*Dibuja tubos del vector de datos de volumen (U,V,W) cuyas coordenadas son los arrays (X,Y,Z), (startx,starty,startz) indican las posiciones de comienzo de los tubos. Puede ocurrir que los vértices y la divergencia sean dados y whith indica la anchura de los tubos (que puede estar afectada por un parámetro de escala n)*
fvc = surf2patch(h) **fvc = surf2patch(Z)** **fvc = surf2patch(Z,C)** **fvc = surf2patch(X,Y,Z)** **fvc = surf2patch(X,Y,Z,C)** **fvc = surf2patch(...,'triangles')** **[f,v,c] = surf2patch(...)**	*Convierte un objeto dado en superficie devolviendo caras, vértices y color en la estructura fvc. El objeto puede venir dado incluso como una superficie en sus diferentes formas. Se pueden crear también caras triangulares (argumento triangles)*
[Nx,Ny,Nz,Nv] = subvolume(X,Y,Z,V,limits) **[Nx,Ny,Nz,Nv] = subvolume(V,limits)** **Nv = subvolume(...)**	*Extrae un subconjunto del volumen V (X,Y,Z) usando los límites dados (limits = [xmin,xmax,ymin, ymax,zmin,zmax])*
lims = volumebounds(X,Y,Z,V) **lims = volumebounds(X,Y,Z,U,V,W)** **lims = volumebounds(V)** **lims = volumebounds(U,V,W)**	*Da coordenadas y límites de colores para el volumen V (U,V,W) cuyas coordenadas son los arrays (X,Y,Z)*
v = flow, v = flow(n) o v = flow(x,y,z)	*Volumen tipo (perfil de velocidad)*

Como primer ejemplo se grafican vectores velocidad como conos en un campo vectorial (Figura 7-68) a partir del conjunto de datos predefinido en MATLAB *wind.mat.*

```
>> load wind
xmin = min(x(:)); xmax = max(x(:)); ymin = min(y(:));
ymax = max(y(:)); zmin = min(z(:));
daspect([2,2,1])
xrange = linspace(xmin,xmax,8);
yrange = linspace(ymin,ymax,8);
zrange = 3:4:15;
[cx cy cz] = meshgrid(xrange,yrange,zrange);
hcones = coneplot(x,y,z,u,v,w,cx,cy,cz,5);
set(hcones,'FaceColor','red','EdgeColor','none')
```

A continuación creamos planos de sección a lo largo de los ejes (Figura 7-69).

```
>> hold on
wind_speed = sqrt(u.^2 + v.^2 + w.^2);
hsurfaces = slice(x,y,z,wind_speed,[xmin,xmax],ymax,zmin);
set(hsurfaces,'FaceColor','interp','EdgeColor','none')
hold off
```

Por último se define un punto de vista apropiado (Figura 7-70).

```
>> axis tight; view(30,40); axis off
```

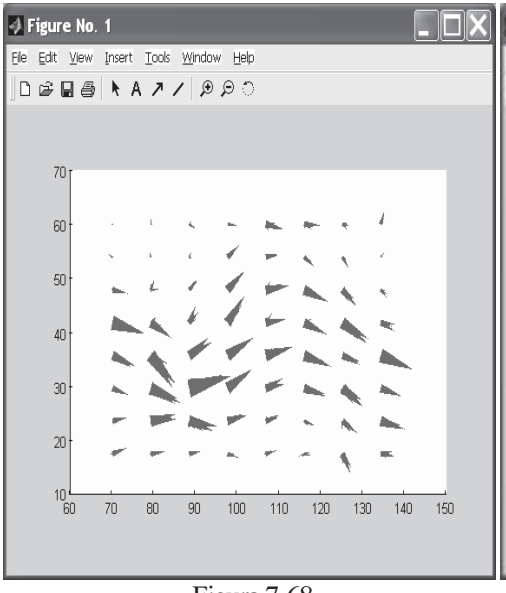


Figura 7-68

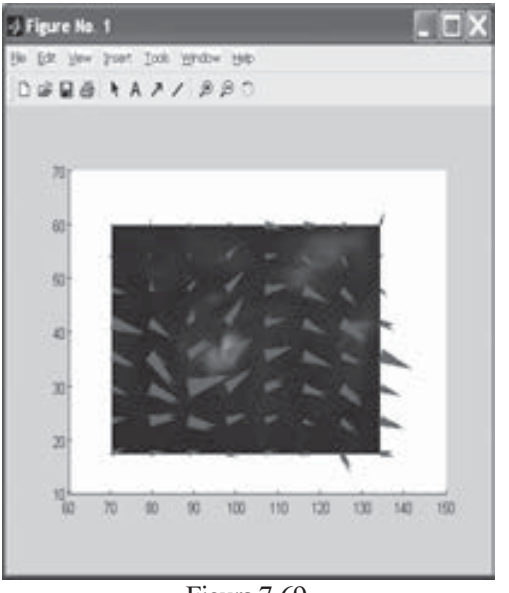


Figura 7-69

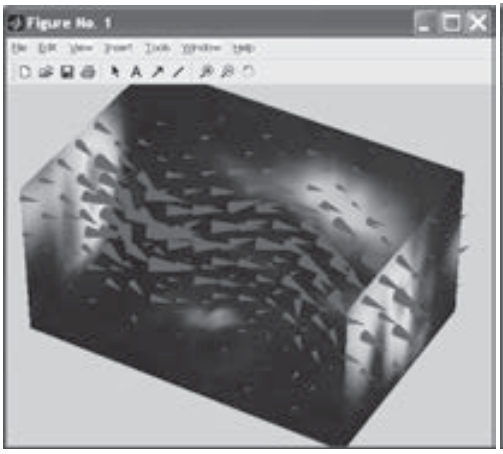


Figura 7-70

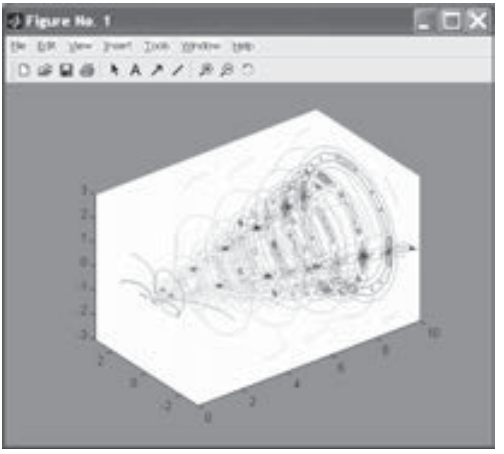


Figura 7-71

A continuación se dibujan contornos en cortes planos de volúmenes sobre el conjunto de datos *wind.mat* con un punto de vista adecuado (Figura 7-71).

```
>> [x y z v] = flow;
h = contourslice(x,y,z,v,[1:9],[],[0],linspace(-8,2,10));
axis([0,10,-3,3,-3,3]); daspect([1,1,1])
set(gcf,'Color',[.5,.5,.5],'Renderer','zbuffer')
```

En el ejemplo siguiente se representa el rotacional del campo vectorial dado por *wind.mat* usando secciones con planos coloreados (Figura 7-72).

```
>> load wind
cav = curl(x,y,z,u,v,w);
slice(x,y,z,cav,[90 134],[59],[0]);
shading interp
daspect([1 1 1]); axis tight
colormap hot(16)
camligth
```

Después se realiza la representación anterior sobre un plano (Figura 7-73).

```
>> load wind
k = 4;
x = x(:,:,k); y = y(:,:,k); u = u(:,:,k); v = v(:,:,k);
cav = curl(x,y,u,v);
pcolor(x,y,cav); shading interp
hold on;
quiver(x,y,u,v,'y')
hold off
colormap copper
```

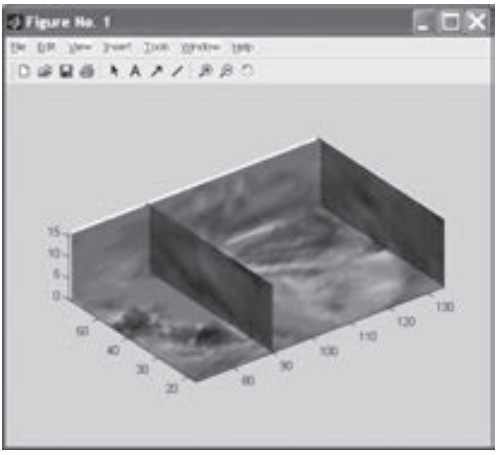


Figura 7-72

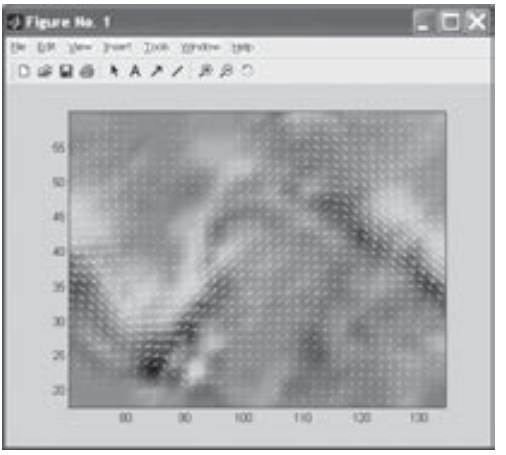


Figura 7-73

En el ejemplo siguiente se representa la divergencia del campo vectorial dado por *wind.mat* usando secciones con planos coloreados (Figura 7-74).

```
>> load wind
div = divergence(x,y,z,u,v,w);
slice(x,y,z,div,[90 134],[59],[0]);
shading interp
daspect([1 1 1])
camlight
```

A continuación se presenta un ejemplo relativo a colores y normales para una isosuperficie dada (Figura 7-75).

```
>> [x y z] = meshgrid(1:20,1:20,1:20);
data = sqrt(x.^2 + y.^2 + z.^2);
cdata = smooth3(rand(size(data)),'box',7);
p = patch(isosurface(x,y,z,data,10));
isonormals(x,y,z,data,p);
isocolors(x,y,z,cdata,p);
set(p,'FaceColor','interp','EdgeColor','none')
view(150,30); daspect([1 1 1]);axis tight
camlight; lighting phong;
```

Posteriormente se representa una isosuperficie (Figura 7-76) basada en el conjunto de datos *wind.mat*.

```
>> [x,y,z,v] = flow;
p = patch(isosurface(x,y,z,v,-3));
isonormals(x,y,z,v,p)
set(p,'FaceColor','red','EdgeColor','none');
daspect([1 1 1])
view(3); axis tight
```

En el ejemplo siguiente (Figura 7-77) reducimos en un 15% las caras de la superficie anterior.

```
>> [x,y,z,v] = flow;
p = patch(isosurface(x,y,z,v,-3));
set(p,'facecolor','w','EdgeColor','b');
daspect([1,1,1])
view(3)
figure;
h = axes;
p2 = copyobj(p,h);
reducepatch(p2,0.15)
daspect([1,1,1])
view(3)
```

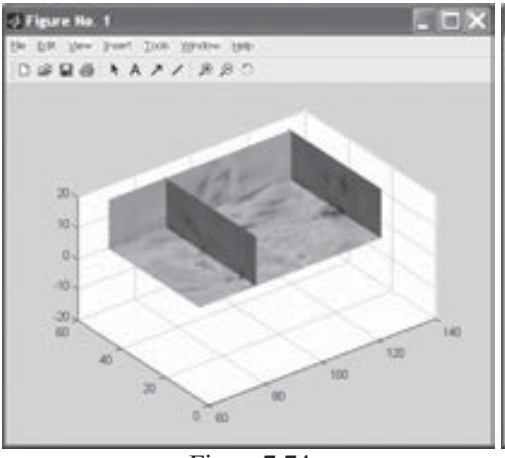


Figura 7-74

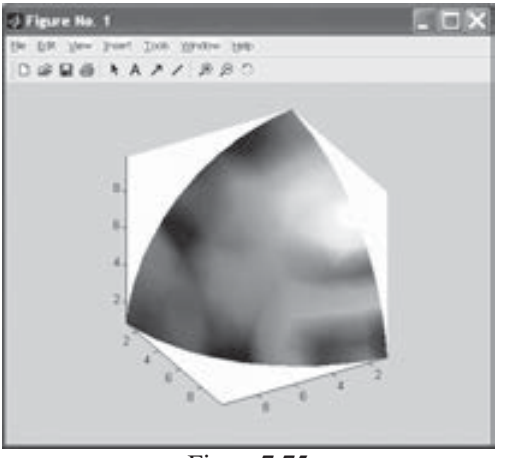


Figura 7-75

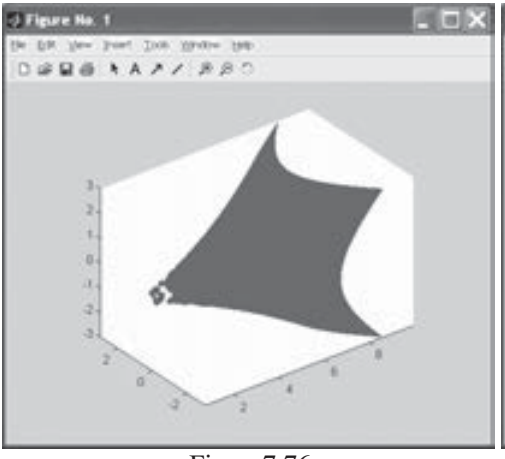


Figura 7-76

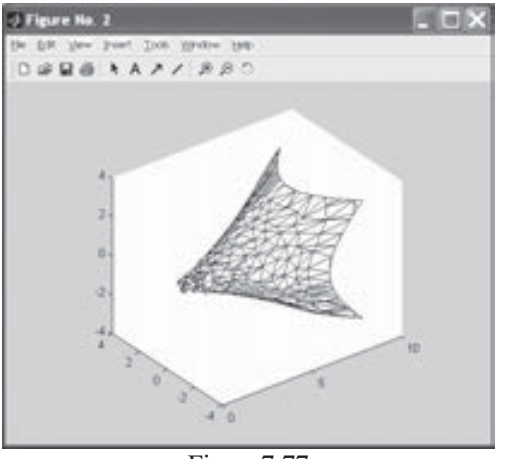


Figura 7-77

En el ejemplo siguiente se reduce el volumen de la isosuperficie anterior (Figura 7-78) y posteriormente se contrae el tamaño de sus caras (Figura 7-79).

```
>> [x,y,z,v] = flow;
[x,y,z,v] = reducevolume(x,y,z,v,2);
fv = isosurface(x,y,z,v,-3);
p1 = patch(fv);
set(p1,'FaceColor','red','EdgeColor',[.5,.5,.5]);
daspect([1 1 1]); view(3); axis tight
title('Original')

>> figure
p2 = patch(shrinkfaces(fv,.3));
set(p2,'FaceColor','red','EdgeColor',[.5,.5,.5]);
daspect([1 1 1]); view(3); axis tight
title('Despues de la contraccion de caras')
```

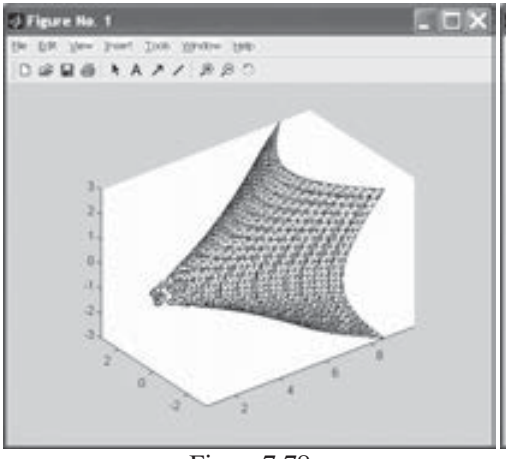


Figura 7-78

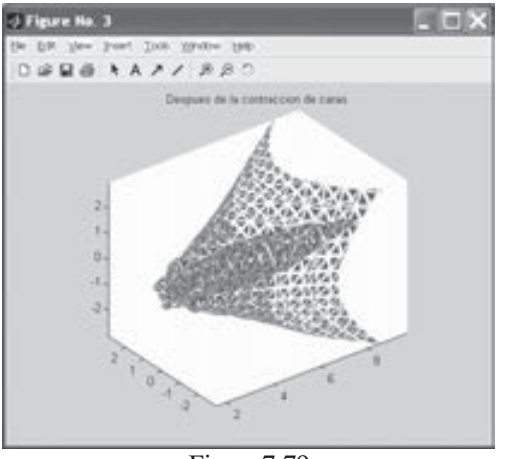


Figura 7-79

Después se crea un gráfico con líneas de corriente bidimensional (Figura 7-80) y otro tridimensional (Figura 7-81).

```
>> load wind
[sx,sy] = meshgrid(80,20:10:50);
streamline(stream2(x(:,:,5),y(:,:,5),u(:,:,5),v(:,:,5),sx,sy));

>> load wind
[sx sy sz] = meshgrid(80,20:10:50,0:5:15);
streamline(stream3(x,y,z,u,v,w,sx,sy,sz))
view(3)
```

A continuación se representa un gráfico de cintas (Figura 7-82) y uno de tubos (Figura 7-83).

```
>> load wind
[sx sy sz] = meshgrid(80,20:10:50,0:5:15);
daspect([1 1 1])
verts = stream3(x,y,z,u,v,w,sx,sy,sz);
cav = curl(x,y,z,u,v,w);
spd = sqrt(u.^2 + v.^2 + w.^2).*.1;
streamribbon(verts,x,y,z,cav,spd);

>> load wind
[sx sy sz] = meshgrid(80,20:10:50,0:5:15);
daspect([1 1 1])
streamtube(x,y,z,u,v,w,sx,sy,sz);
```

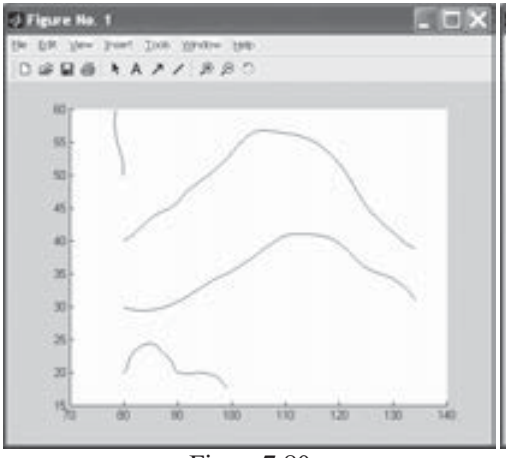


Figura 7-80

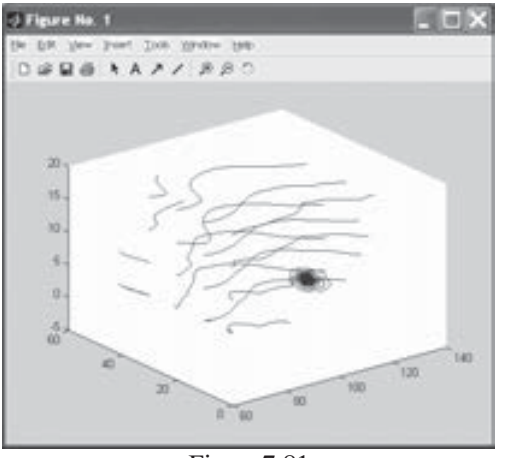


Figura 7-81

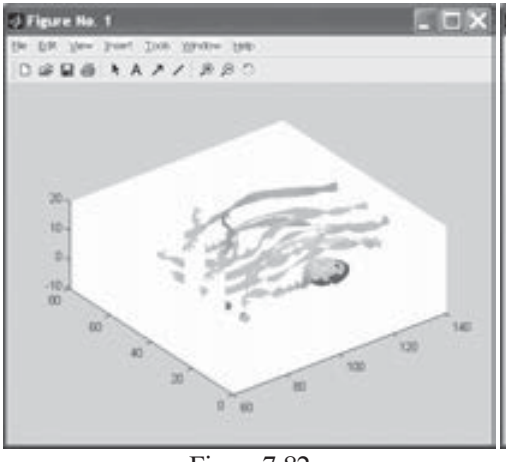


Figura 7-82

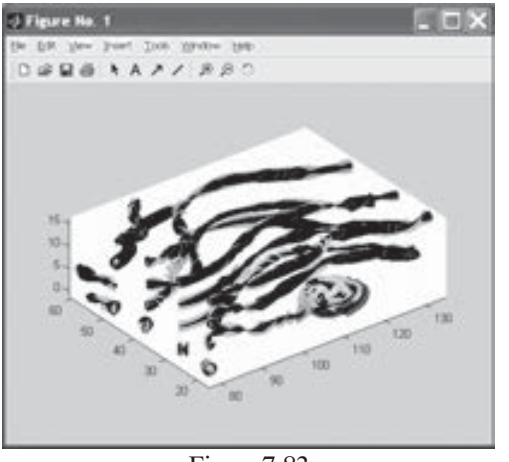


Figura 7-83

7.10 Gráficos especializados

Hay varios tipos de gráficos especializados que pueden realizarse con MATLAB (áreas, cajas, sectores tridimensionales, gráficos de Pareto, etc.) mediante los comandos que se presentan en la tabla siguiente.

area(Y)	*Realiza el gráfico de áreas relativo al vector de frecuencias Y*
area(X,Y)	*Realiza el gráfico de áreas relativo al vector de frecuencias Y cuyos elementos son dados a través del vector X*
area(...,ymin)	*Especifica el límite inferior en la dirección de y del área de relleno*
box on, box off	*Habilita y deshabilita cajas en los ejes para gráficos 2-D y3-D*
comet(y)	*Realiza el gráfico de cometa relativo al vector de frecuencias Y*
comet(x,y)	*Realiza el gráfico de cometa relativo al vector de frecuencias Y cuyos elementos son dados a través del vector X*
comet(x,y,p)	*Gráfico de cometa con cuerpo de longitud p*length(y)*
ezcontour(f)	*Gráfico de contorno de f(x,y) en [-2π, 2π]x[-2π, 2π]*
ezcontour(f,dominio)	*Gráfico de contorno de f(x,y) en el dominio dado*
ezcontour(...,n)	*Gráfico de contorno de f(x,y) en la malla nxn*
ezcontourf(f)	*Gráfico de contorno de f(x,y) relleno en [-2π, 2π]x[-2π, 2π]*
ezcontourf(f,dominio)	*Gráfico de contorno de f(x,y) relleno en el dominio dado*
ezcontourf(...,n)	*Gráfico de contorno de f(x,y) relleno en la malla nxn*
ezmesh(f)	*Gráfico de malla de f(x,y) relleno en [-2π, 2π]x[-2π, 2π]*
ezmesh(f,dominio)	*Gráfico de malla de f(x,y) relleno en el dominio dado*
ezmesh(...,n)	*Gráfico de malla de f(x,y) relleno en la malla nxn*
ezmesh(x,y,z)	*Gráfico de malla para x=x(t,u), y=y(t,u), z=z(t,u) t,u∈[-2π, 2π]*
ezmesh(x,y,z, dominio)	*Gráfico de malla para x=x(t,u), y=y(t,u), z=z(t,u) t,u∈ dominio*
ezmesh(..., 'circ')	*Gráfico de malla sobre un disco centrado en el dominio*
ezmeshc(f) **ezmeshc(f,dominio)** **ezmeshc(...,n)** **ezmeshc(x,y,z)** **ezmeshc(x,y,z, dominio)** **ezmeshc(..., 'circ')**	*Realiza una combinación de gráfico de malla y contorno*
ezsurf(f) **ezsurf(f,dominio)** **ezsurf(...,n)** **ezsurf(x,y,z)** **ezsurf(x,y,z, dominio)** **ezsurf(..., 'circ')**	*Realiza un gráfico de superficie coloreado*
ezsurfc(f) **ezsurfc(f,dominio)**	*Realiza una combinación de gráfico de superficie y de contorno*

ezsurfc(...,n) **ezsurfc(x,y,z)** **ezsurfc(x,y,z, dominio)** **ezsurfc(..., 'circ')**	
ezplot3(x,y,z) **ezplot3(x,y,z,dominio)** **ezplot3(…, 'animate')**	*Curva paramétrica 3D x=x(t), y=y(t), z=z(t) t∈[-2π, 2π]* *Curva paramétrica 3D x=x(t), y=y(t), z=z(t) t∈dominio* *Curva paramétrica 3D con animación*
ezpolar(f) **ezpolar(f, [a,b])**	*Grafica la curva polar r=f(c) con c∈[0, 2π]* *Grafica la curva polar r=f(c) con c∈[a, b]*
pareto(Y) **pareto(X,Y)**	*Realiza el gráfico de Pareto relativo al vector de frecuencias Y* *Realiza el gráfico de Pareto relativo al vector de frecuencias Y cuyos elementos son dados a través del vector X*
pie3(X) **pie3(X,explode)**	*Gráfico de sectores tridimensional para las frecuencias X* *Gráfico de sectores tridimensional desgajado*
plotmatrix(X,Y)	*Gráfico de dispersión de las columnas de X contra las de Y*
quiver(U,V) **quiver(X,Y,U,V)** **quiver(...,scale)** **quiver(...,LineSpec)** **quiver(...,LineSpec,'filled')**	*Gráfico de velocidad de los vectores con componentes (u,v) en los puntos (x,y). Se puede definir una escala, especificaciones de línea y relleno*
ribbon(Y) **ribbon(X,Y)** **ribbon(X,Y,width)**	*Grafica las columnas de Y como cintas tridimensionales* *Grafica X contra las columnas de Y* *Se especifica la anchura de las cintas*
stairs(Y) **stairs(X,Y)** **stairs(...,LineSpec)**	*Gráfico en escalera con los elementos de Y* *Gráfico en escalera de los elementos de Y correspondientes con los de X* *Especificaciones de línea para el gráfico en escalera*
scatter(X,Y,S,C) **scatter(X,Y)** **scatter(X,Y,S)** **scatter(...,marcador)** **scatter(...,'filled')**	*Gráfico de dispersión para los vectores (X,Y) según los colores C y el área de cada marcador S. También se puede obtener el gráfico relleno (opción fill) y utilizar distintos tipos de marcadores*
scatter3(X,Y,Z,S,C) **scatter3(X,Y,Z)** **scatter3(X,Y,Z,S)** **scatter3(...,marcador)** **scatter(...,'filled')**	*Gráfico de dispersión tridimensional para los vectores (X,Y,Z) según los colores C y el área de cada marcador S. También se puede obtener el gráfico relleno (opción fill) y utilizar distintos tipos de marcadores*
TRI = delaunay(x,y)	*Triangulación de Delaunay*
K = dsearch(x,y,TRI,xi,yi)	*Triangulación de Delaunay por el punto más cercano*
IN = inpolygon(X,Y,xv,yv)	*Detecta puntos en el interior de una región poligonal*

polyarea(X,Y)	*Área del polígono especificado por los vectores X e Y*
tsearch	*Triangulación de Delaunay*
voronoi(x,y) **voronoi(x,y,TRI)**	*Diagramas de Voronoi*

Como primer ejemplo realizamos un gráfico de área apilado (Figura 7-84)

```
>> Y = [   1, 5, 3;
         3, 2, 7;
         1, 5, 3;
         2, 6, 1];
area(Y)
grid on
colormap summer
```

Después creamos un gráfico bidimensional de cometa (Figura 7-85).

```
>> t = 0:.01:2*pi;
x = cos(2*t).*(cos(t).^2);
y = sin(2*t).*(sin(t).^2);
comet(x,y);
```

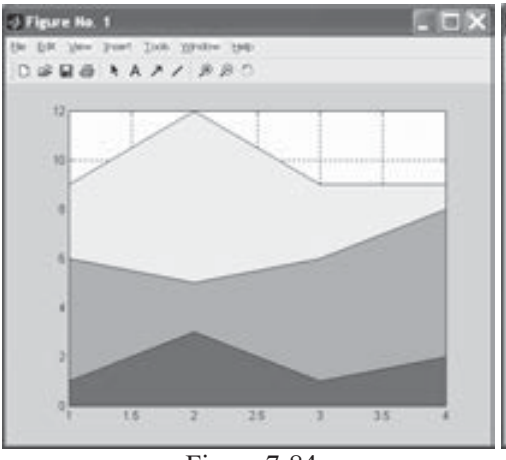


Figura 7-84

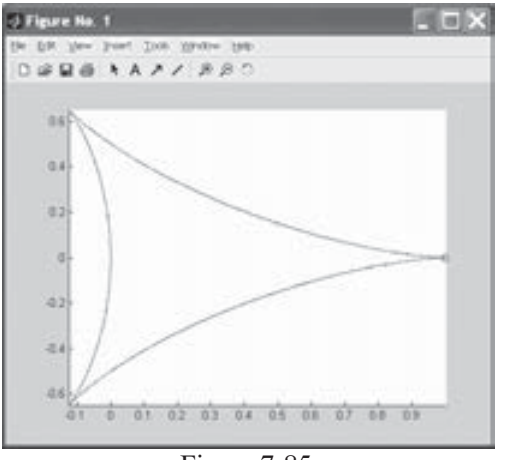


Figura 7-85

A continuación se realiza el gráfico de contorno (Figura 7-86) para la función:

$$f(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

```
>> f = ['3*(1-x)^2*exp(-(x^2)-(y+1)^2)',...
'- 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2)', '- 1/3*exp(-(x+1)^2 - y^2)'];
>> ezcontour(f,[-3,3],49)
```

Seguidamente rellenamos de color el contorno anterior (Figura 7-87).

```
>> ezcontourf(f,[-3,3],49)
```

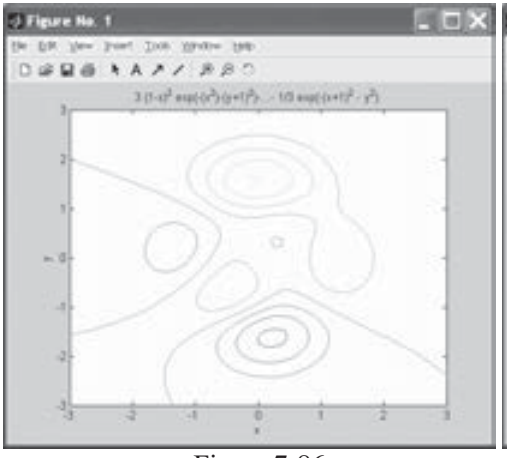


Figura 7-86

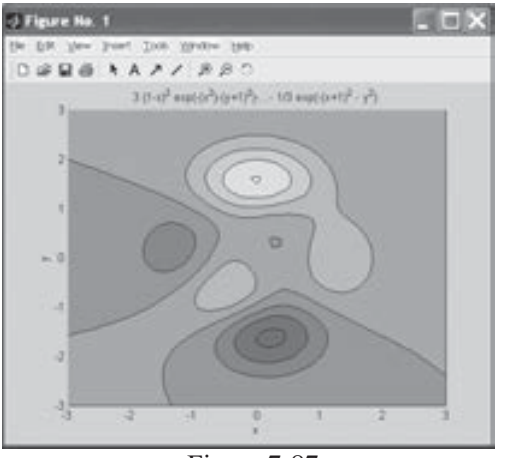


Figura 7-87

En el ejemplo siguiente realizamos un gráfico mixto de malla y contorno (Figura 7-88) para la función:

$$f(x,y) = \frac{y}{1+x^2+y^2}$$

```
>> ezmeshc('y/(1 + x^2 + y^2)',[-5,5,-2*pi,2*pi])
```

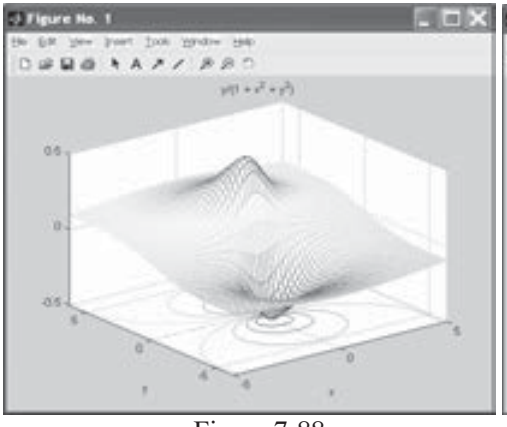


Figura 7-88

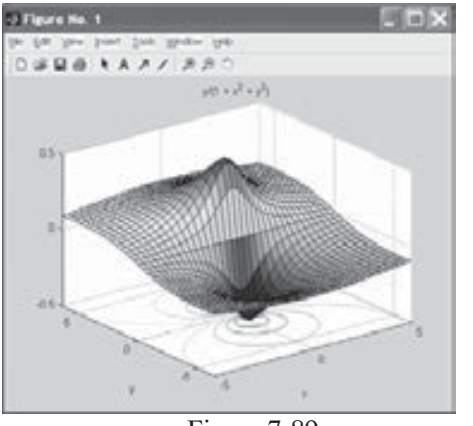


Figura 7-89

Más tarde se realiza un gráfico de superficie y contorno (Figura 7-89).

```
>> ezsurfc('y/(1 + x^2 + y^2)',[-5,5,-2*pi,2*pi],35)
```

A continuación se grafica una curva en paramétricas en el espacio (Figura 7-90).

```
>> ezplot3('sin(t)','cos(t)','t',[0,6*pi])
```

En el ejemplo siguiente se construye un gráfico se sectores tridimensional (Figura 7-91).

```
>> x = [1 3 0.5 2.5 2]
explode = [0 1 0 0 0]
pie3(x,explode)
colormap hsv
```

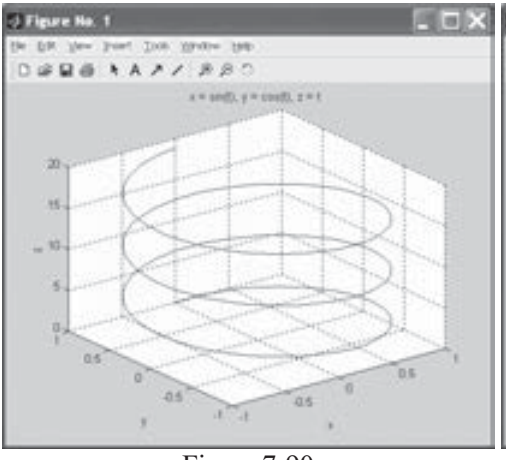


Figura 7-90

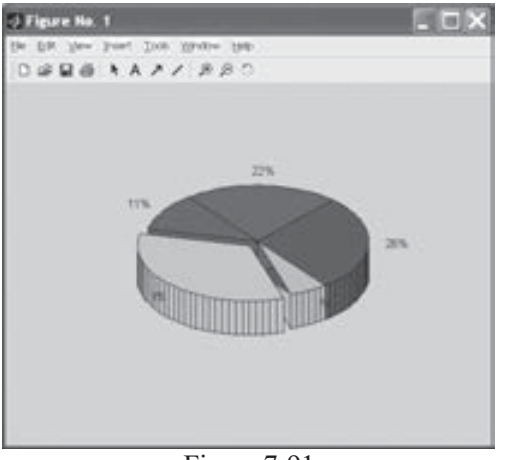


Figura 7-91

Seguidamente se crea un gráfico de cintas para la superficie *peaks (*Figura 7-92).

```
>> [x,y] = meshgrid(-3:.5:3,-3:.1:3);
z = peaks(x,y);
ribbon(y,z)
colormap hsv
```

En el ejemplo siguiente se crea un gráfico plano escalonado (Figura 7-93).

```
>> x = 0:.25:10;
stairs(x,sin(x))
```

En el código siguiente se crea un gráfico de dispersión tridimensional (Figura 7-94).

```
>> [x,y,z] = sphere(16);
X = [x(:)*.5 x(:)*.75 x(:)];
Y = [y(:)*.5 y(:)*.75 y(:)];
Z = [z(:)*.5 z(:)*.75 z(:)];
S = repmat([1 .75 .5]*10,prod(size(x)),1);
C = repmat([1 2 3],prod(size(x)),1);
scatter3(X(:),Y(:),Z(:),S(:),C(:),'filled'), view(-60,60)
```

A continuación se representa un área poligonal (Figura 7-95).

```
>> L = linspace(0,2.*pi,6); xv = cos(L)';yv = sin(L)';
xv = [xv ; xv(1)]; yv = [yv ; yv(1)];
A = polyarea(xv,yv);
plot(xv,yv); title(['Area = ' num2str(A)]); axis image
```

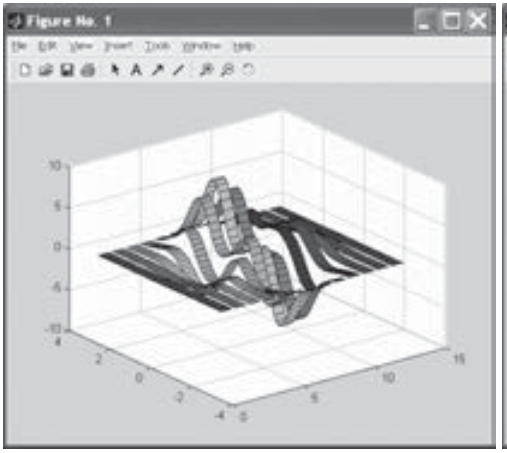


Figura 7-92

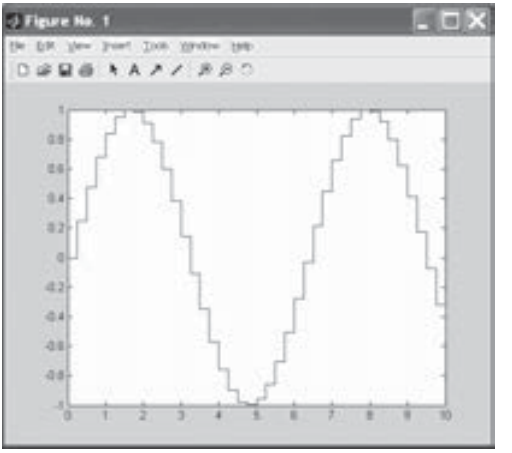


Figura 7-93

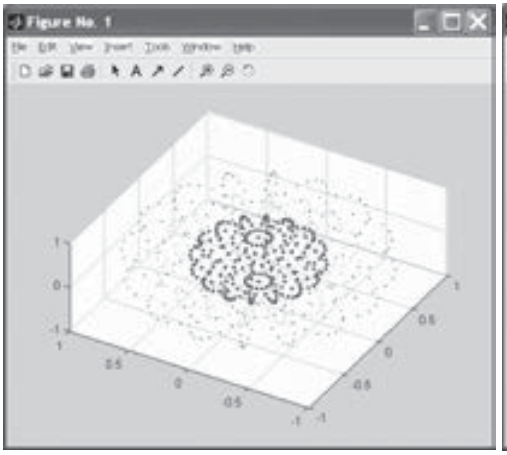


Figura 7-94

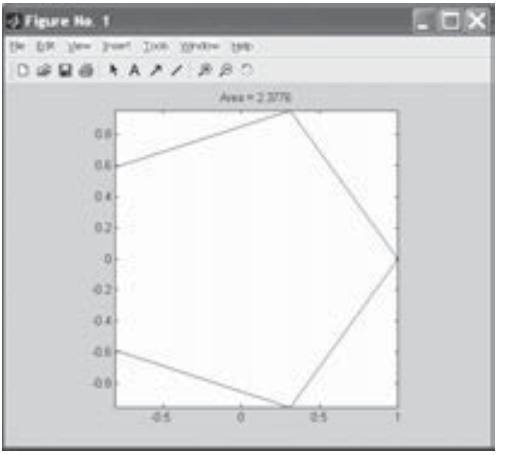


Figura 7-95

7.11 Impresión, exportación y otras tareas con gráficos

En el cuadro siguiente se presentan varios comandos de MATLAB que permiten imprimir gráficos, situar opciones de impresora y guardar gráficos a fichero.

orient **orient landscape** **orient portrait** **orient tall**	*Sitúa la orientación del papel para imprimir en su valor por defecto* *Sitúa la orientación del papel de impresión como horizontal* *Sitúa la orientación del papel de impresión como vertical* *Imprime en toda la página con orientación vertical*
pagesetupdlg	*Crea una caja de diálogo para manejar la posición de la figura*
print **print** ***-device -options*** **file**	*Imprime el gráfico mediante hardcopy* *Imprime el gráfico a fichero con opciones de device dadas*
printdlg **printdlg(fig)**	*Imprime la figura corriente* *Crea una caja de diálogo para impresión identificada por fig*
saveas(h,'file.ext') **saveas(h,'file','format')**	*Guarda la figura h en el fichero gráfico file.ext* *Guarda la figura h en el fichero file con el formato gráfico especificado*

Los formatos de fichero gráfico según su extensión son los siguientes:

Extensión	***Formato***
`ai`	*Adobe Illustrator `88*
`bmp`	*Windows bitmap*
`emf`	*Enhanced metafile*
`eps`	*EPS Level 1*
`fig`	*figura MATLAB*
`jpg`	*imagen JPEG*
`m`	*MATLAB M-file*
`pbm`	*Portable bitmap*
`pcx`	*Paintbrush 24-bit*
`pgm`	*Portable Graymap*
`png`	*Portable Network Graphics*
`ppm`	*Portable Pixmap*
`tif`	*Imagen TIFF image comprimida*

Ejercicio 7-1. Generar los cilindros definidos por los perfiles de las funciones 2+ cos(t) y 2+sen(t).

La sintaxis siguiente genera el primer cilindro (Figura 7- 96).

```
>> t = 0:pi/10:2*pi;
>> cylinder(2+cos(t));
>> axis square
```

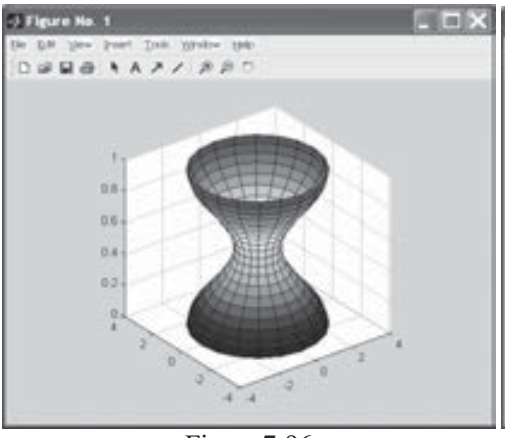


Figura 7-96

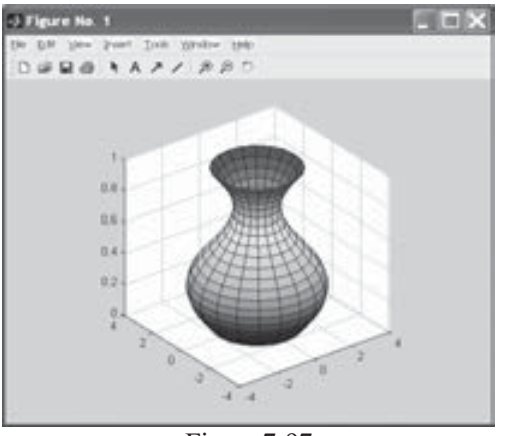


Figura 7-97

La sintaxis siguiente genera el segundo cilindro (Figura 7- 97).

```
>> t = 0:pi/10:2*pi;
>> cylinder(2+sin(t));
>> axis square
```

Ejercicio 7-2. Graficar las superficies normales de la función:

$$z = xe^{-x^2-y^2}$$

La sintaxis siguiente genera la Figura 7-98 .

```
>> [X,Y] = meshgrid(-2:0.25:2,-1:0.2:1);
Z = X.* exp(-X.^2 - Y.^2);
[U,V,W] = surfnorm(X,Y,Z);
quiver3(X,Y,Z,U,V,W,0.5);
hold on
surf(X,Y,Z);
colormap hsv
view(-35,45)
axis ([-2 2 -1 1 -.6 .6])
hold off
```

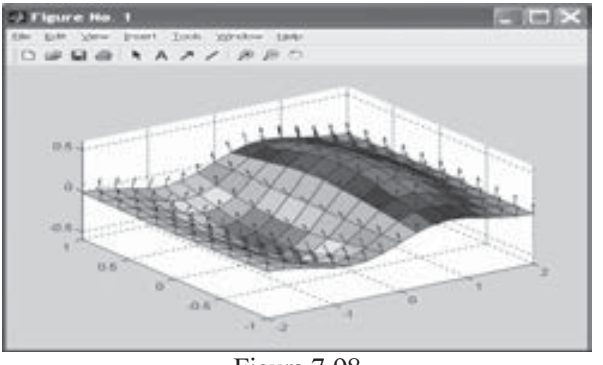


Figura 7-98

Ejercicio 7-3. Realizar un gráfico de cascada para la función peaks

La sintaxis siguiente genera la Figura 7- 99.

```
>> [X,Y,Z] = peaks(30);
waterfall(X,Y,Z)
```

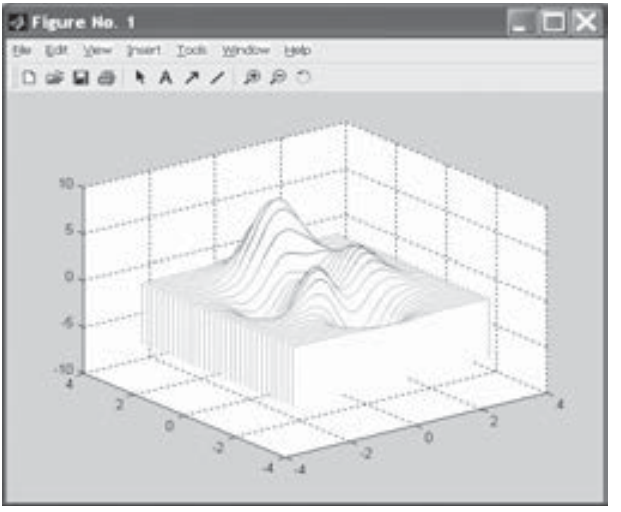


Figura 7-99

Ejercicio 7-4. Visualizar sobre el rango [-2,2]x[-2,2]x[-2,2] la función

$$v = xe^{-x^2-y^2-z^2}$$

La sintaxis siguiente genera la Figura 7- 100.

```
>> [x,y,z] = meshgrid(-2:.2:2,-2:.25:2,-2:.16:2);
v = x.*exp(-x.^2-y.^2-z.^2);
xslice = [-1.2,.8,2]; yslice = 2; zslice = [-2,0];
slice(x,y,z,v,xslice,yslice,zslice)
```

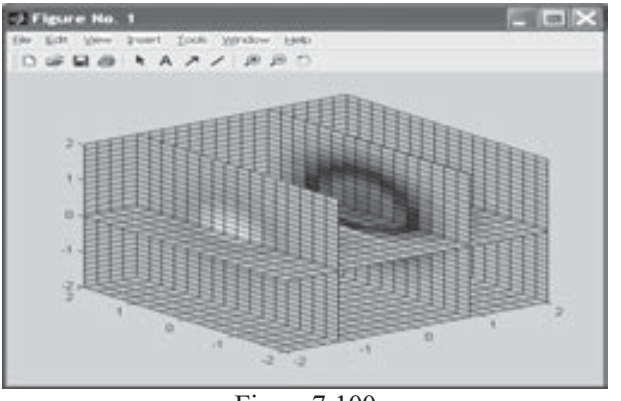


Figura 7-100

Ejercicio 7-5. Representar en el intervalo [-8,8] la función

$$f(x) = \frac{x^3}{x^2 - 4}$$

La Figura 7-101 representa la función pedida mediante la sintaxis siguiente:

```
>> ezplot('x^3/(x^2-4)',[-8,8])
```

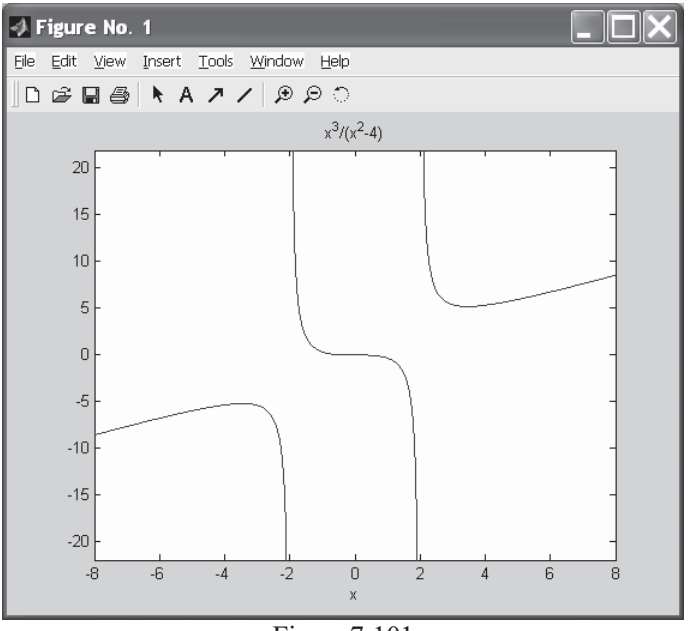


Figura 7-101

Ejercicio 7-6. Graficar sobre los mismos ejes las funciones bessel(1,x), bessel(2,x) y bessel(3,x) para valores de x entre 0 y 12, separados uniformemente entre sí dos décimas. Colocar tres leyendas y tres tipos de trazo diferentes (normal, asteriscos y círculos, respectivamente) para las tres funciones.

La Figura 7-102 representa las funciones pedidas mediante la sintaxis siguiente:

```
>> x  = 0:.2:12;
plot(x,bessel(1,x),x,bessel(2,x),'*',x,bessel(3,x),'o');
legend('Bessel(1,x)','Bessel(2,x)','Bessel(3,x)');
```

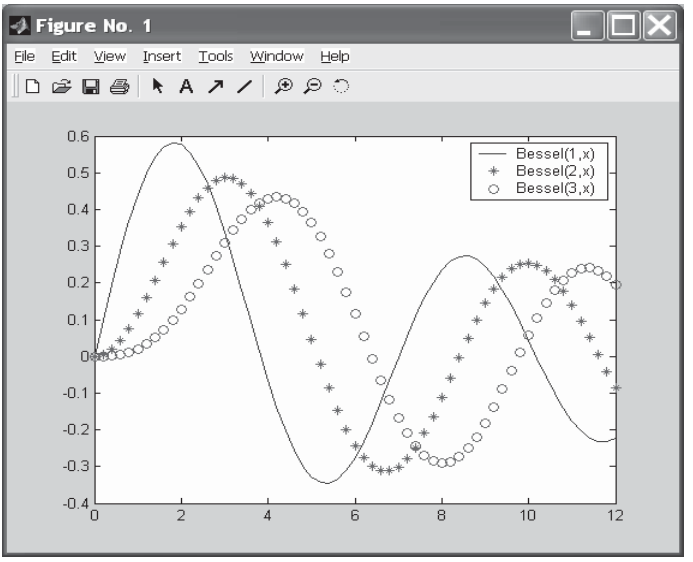


Figura 7-102

Ejercicio 7-7. Representar la curva en polares r = 4(1+Cos(a)) para a entre 0 y 2π (cardiode). Representar también la curva en polares r=3a para a entre -4π y 4π (espiral).

La primera curva (Figura 7-103) se representa mediante la sintaxis siguiente:

```
>> a=0:.01:2*pi;
r=4*(1+cos(a));
polar(a,r)
title('CARDIOIDE')
```

La segunda curva (Figura 7-104) puede representarse mediante la sintaxis:

```
>> ezpolar('3*a',[-4*pi,4*pi])
```

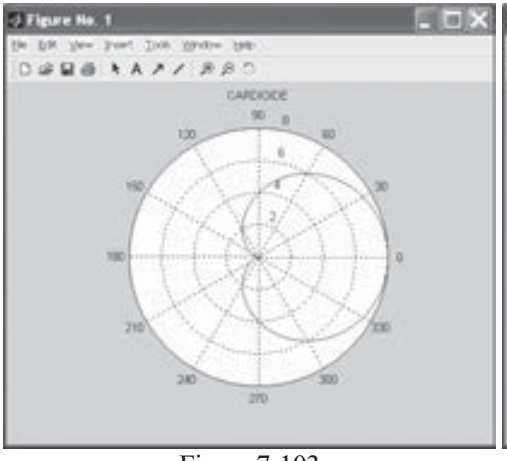


Figura 7-103

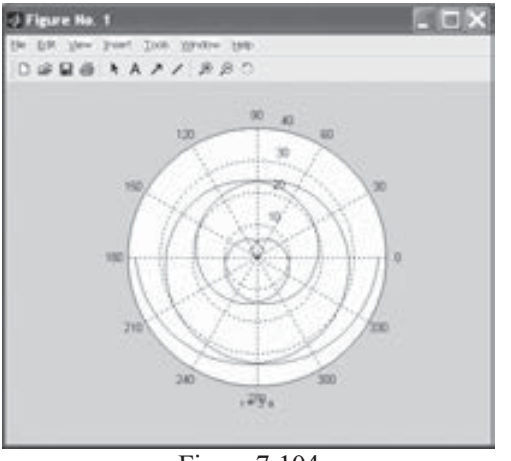


Figura 7-104

Ejercicio 7-8. Representar la curva alabeada de coordenadas paramétricas $x=Cos^2(t)$, $y=Sen(t)Cos(t)$, $z=Sen(t)$ para t variando entre -4π y 4π.

La Figura 7-105 representa las funciones pedidas mediante la sintaxis siguiente:

```
>> t=-4*pi:0.01:4*pi;
x=cos(t).^2;
y=sin(t).*cos(t);
z=sin(t);
plot3(x,y,z)
```

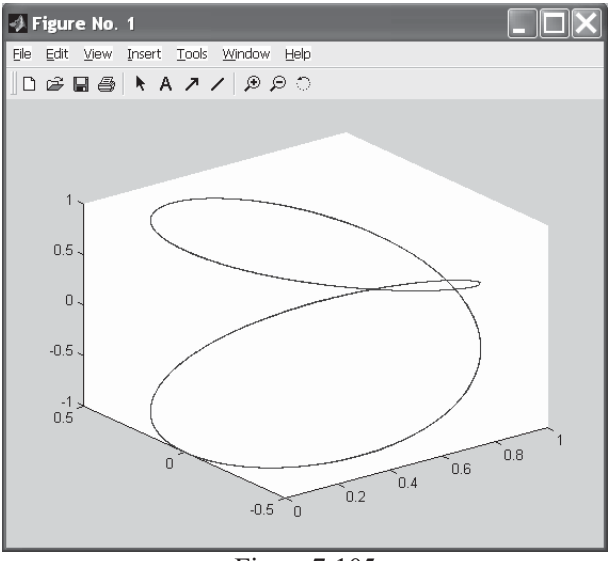


Figura 7-105

Ejercicio 7-9. Representar la superficie, su gráfico de malla y su gráfico de contorno cuya ecuación es la siguiente:

$$z = xe^{-x^2-y^2} \quad -2 < x, y < 2$$

El gráfico de la superficie (Figura 7-106) puede representarse como sigue:

```
>> ezsurf('x*exp(-x^2-y^2)',[-2,2],[-2,2])
```

El gráfico de malla (Figura 7-107) puede representarse como sigue:

```
>> ezmesh('x*exp(-x^2-y^2)',[-2,2],[-2,2])
```

El gráfico de contorno (Figura 7-108) puede representarse como sigue:

```
>> ezcontour('x*exp(-x^2-y^2)',[-2,2],[-2,2])
```

Podemos realizar el gráfico de superficie y contorno simultáneamente (Figura 7-109) como sigue:

```
>> ezsurfc('x*exp(-x^2-y^2)',[-2,2],[-2,2])
```

Podemos realizar el gráfico de malla y contorno simultáneamente (Figura 7-110) como sigue:

```
>> ezmeshc('x*exp(-x^2-y^2)',[-2,2],[-2,2])
```

También podemos representar el gráfico de malla con la opción de cortina o telón inferior (Figura 7-111) como sigue:

```
>> [X,Y]=meshgrid(-2:.1:2,-2:.1:2);
Z=X.*exp(-X.^2 - Y.^2);
meshz(X,Y,Z)
```

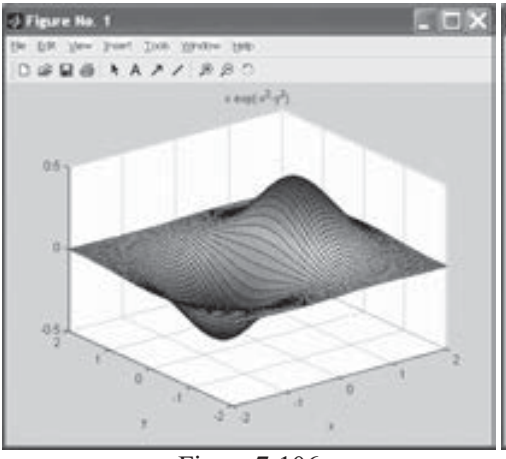


Figura 7-106

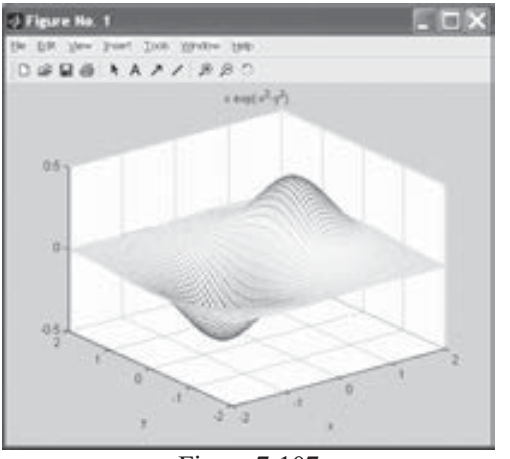


Figura 7-107

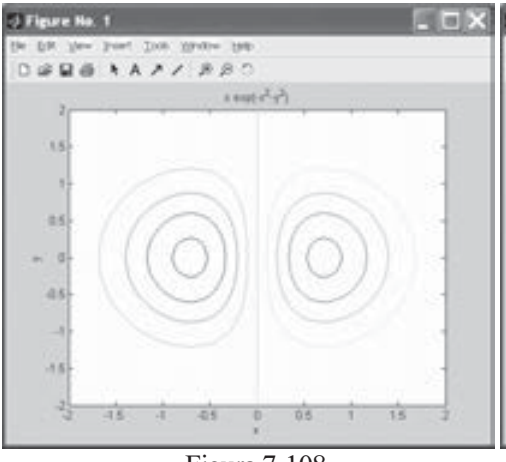


Figura 7-108

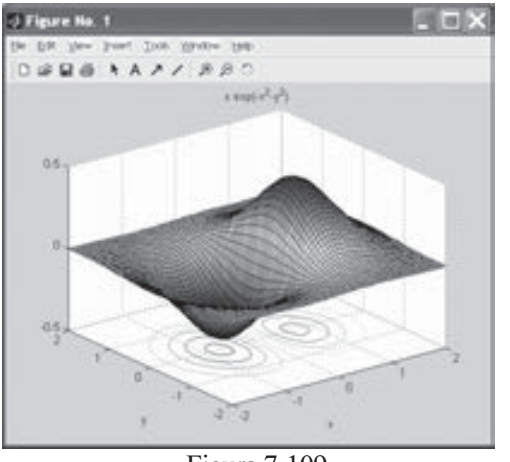


Figura 7-109

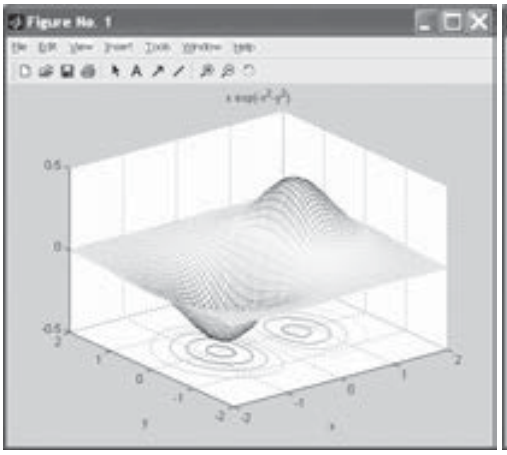


Figura 7-110

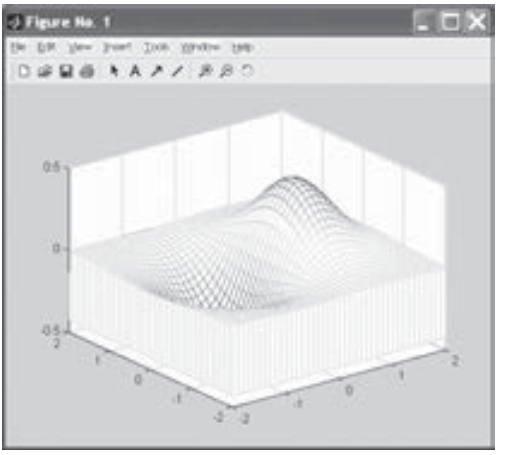


Figura 7-111

Ejercicio 7-10. Representar un gráfico de curvas de nivel con 20 líneas y un gráfico de densidad para la superficie de ecuación z = Sen(x)Sen(y) con -2<x,y<2.

El gráfico de curvas de nivel (Figura 7-112) se realiza mediante la sintaxis:

```
>> [X,Y]=meshgrid(-2:0.1:2);
Z=sin(X).*sin(Y);
contour(Z,20)
```

El gráfico de densidad (Figura 7-113) se realiza mediante la siguiente sintaxis:

```
>> [X,Y]=meshgrid(-2:0.1:2);
Z=sin(X).*sin(Y);
pcolor(X,Y,Z)
```

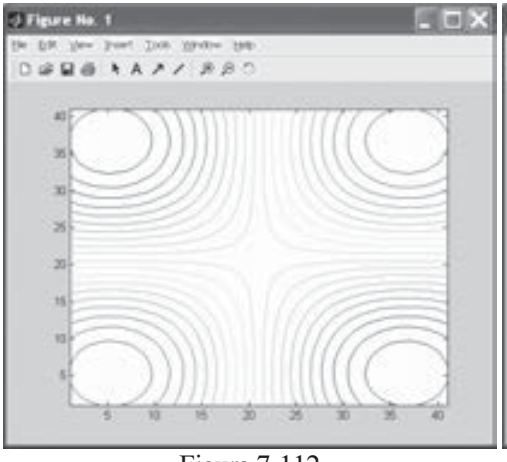


Figura 7-112

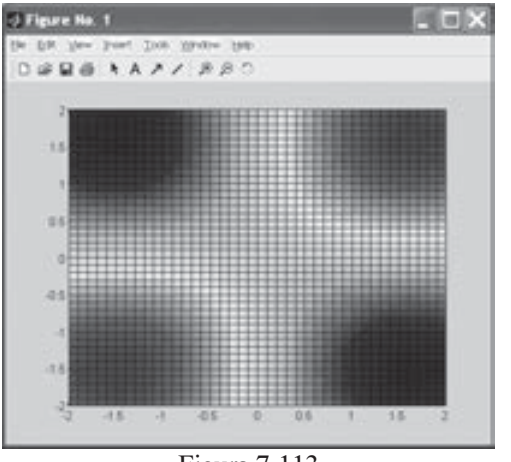


Figura 7-113

Ejercicio 7-11. Representar la superficie de coordenadas paramétricas:

$$x= 4Cos(r)Sec(t) \quad y= 2Sen(r)Sec(t) \quad z= tan(t) \quad -2\pi<r<2\pi, \ -\pi<r<\pi$$

El gráfico de la superficie paramétrica (Figura 7-114) se realiza mediante la sintaxis:

```
>> r=(-2*pi:0.1:2*pi)';
t=(-pi:0.1:pi);
X=4*cos(r)*sec(t);
Y=2*sin(r)*sec(t);
Z=ones(1,size(r))'*tan(t);
surf(X,Y,Z)
```

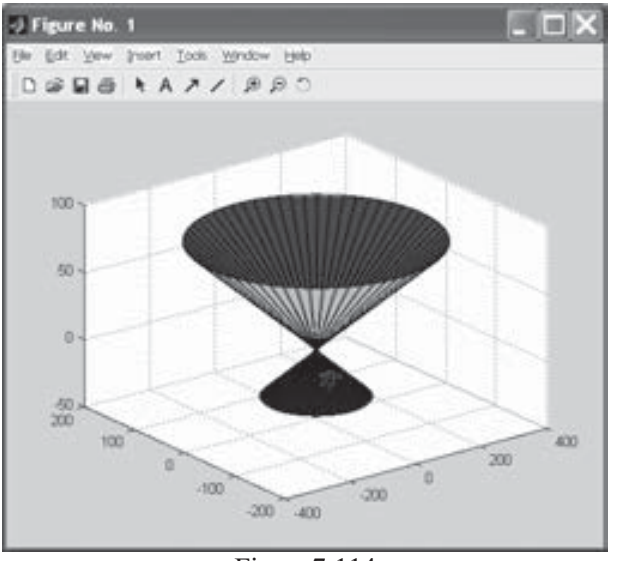


Figura 7-114

Ejercicio 7-12. Construir la gráfica de la superficie de revolución que resulta al girar la función Sen(x) alrededor del eje Z. Obtener también la gráfica de la superficie de revolución al hacer girar la función e^x alrededor del eje Y.

Para obtener la ecuación de la superficie, en el supuesto de que el giro sea alrededor del eje *Z*, hay que considerar la gráfica de la curva generatriz $y = r(z)$ en el plano *YZ*. Al girar esta gráfica alrededor del eje *Z* se forma una superficie de revolución. Las secciones por planos $z = z0$ son circunferencias cuyo radio es $r(z_0)$ y de ecuación $x^2 + y^2 = [r(z_0)]^2$. Eso significa que la ecuación $x^2 + y^2 = [r(z)]^2$ describe los puntos de la superficie de revolución. En el caso de nuestro problema, se tiene $r(z)=sen(z)$, y la curva a representar es $x^2+ y^2= Sen[z]^2$, que se grafica en paramétricas (Figura 7-115) mediante la sintaxis:

```
>> r=(0:0.1:2*pi)';
t=(-pi:0.1:2*pi);
X=cos(r)*sin(t);
Y=sin(r)*sin(t);
Z=ones(1,size(r))'*t;
surf(X,Y,Z)
```

Para obtener la gráfica anterior, pero girando la función exponencial alrededor del eje *Y* (la curva generatriz es ahora la función e^x), se utiliza la sintaxis (Figura 7-116):

```
>> r=(0:0.1:2*pi)';
t=(-2:0.1:2);
X=cos(r)*exp(t);
Y=ones(1,size(r))'*t;
Z=sin(r)*exp(t);
surf(X,Y,Z)
```

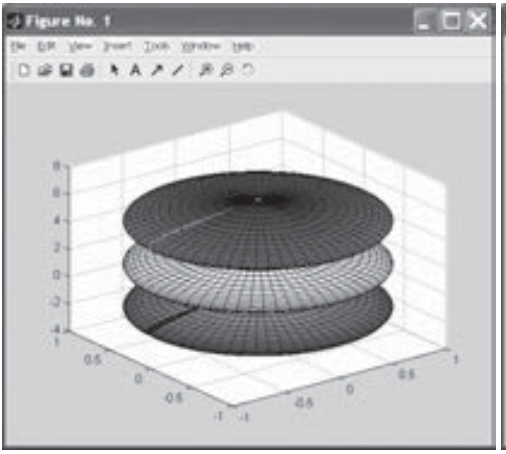


Figura 7-115

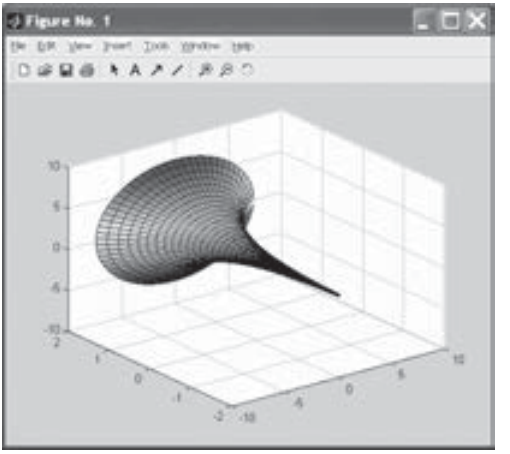


Figura 7-116

Ejercicio 7-13. Representar el toro de revolución de ecuaciones paramétricas {Cos[t](3+Cos[u]), Sin[t](3+Cos[u]),Sin[u]}, con {t,0,2 Pi} y {u,0,2Pi}. Representar también el cilindro generado por la curva 4Cos(t) con {t,0,2 Pi}.

El toro de revolución (Figura 7-117) se representa mediante la sintaxis:

```
>> r=(0:0.1:2*pi)';
t=(0:0.1:2*pi);
X=(3+cos(r))*cos(t);
Y=(3+cos(r))*sin(t);
Z=sin(r)*ones(size(t));
surf(X,Y,Z)
```

El cilindro (Figura 7-118) se representa mediante la siguiente sintaxis:

```
>> t=0:pi/10:2*pi;
>> cylinder(4*cos(t));
```

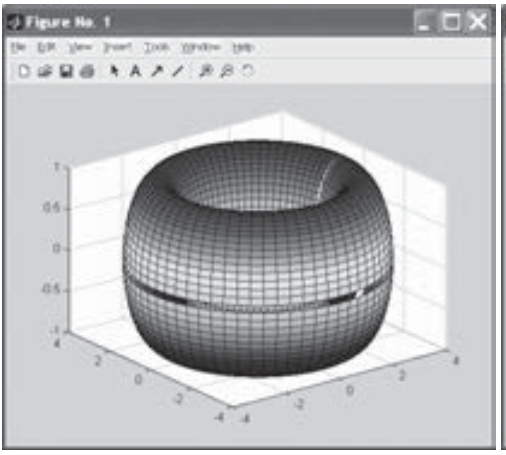


Figura 7-117

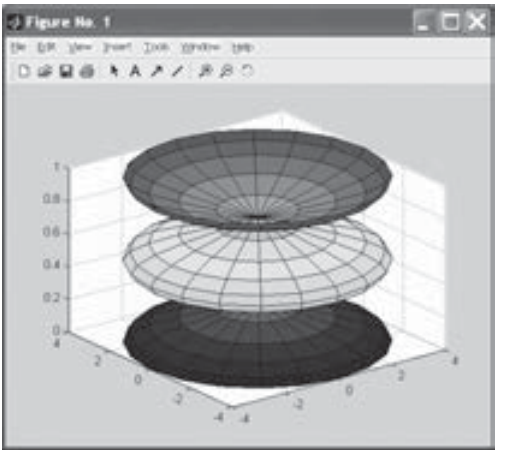


Figura 7-118

Ejercicio 7-14. Representar el paraboloide $x^2 + y^2$ seccionado por el plano z=2.

Para realizar esta representación gráfica (Figura 7-119) se usa la siguiente sintaxis:

```
>> t=0:pi/10:2*pi;
>> cylinder(4*cos(t));
>> [x,y]=meshgrid(-3:.1:3);
>> z=(1/2)*(x.^2+y.^2);
>> mesh(x,y,z)
>> hold on;
>> z=2*ones(size(z));
>> mesh(x,y,z)
>> view(-10,10)
```

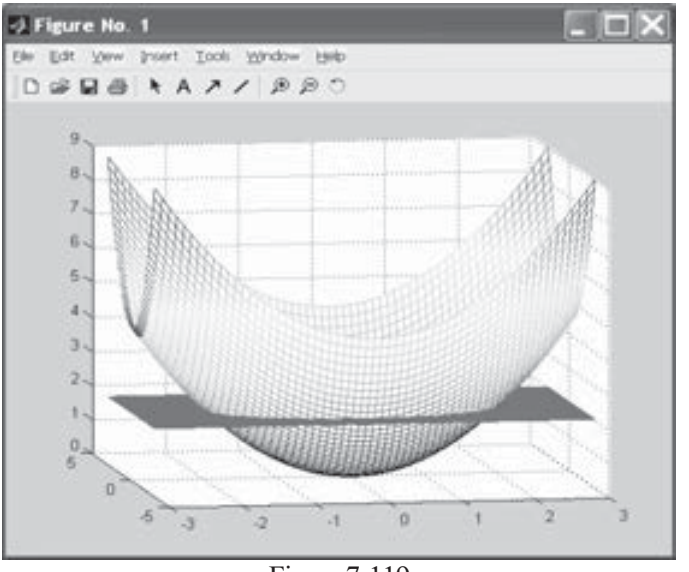


Figura 7-119

8

Programación y métodos de análisis numérico

8.1 MATLAB y la programación

MATLAB puede utilizarse como un lenguaje de programación de alto nivel que incluye estructuras de datos, funciones, instrucciones de control de flujo, manejo de entradas/salidas e incluso programación orientada a objetos.

Los programas de MATLAB suelen escribirse en ficheros denominados M-ficheros. Un M-fichero no es más que código MATLAB (*scripts*) que simplemente ejecuta una serie de comandos o funciones que aceptan argumentos y producen una salida. Los M-ficheros se crean utilizando el editor de texto, tal y como ya se vio en el Capítulo 2.

Editor de texto

Para crear un nuevo M-fichero se utiliza el *Editor/Debugger*, que se activa haciendo clic en el botón de la barra de herramientas de MATLAB o mediante *File→ New→ M-file* en el escritorio de MATLAB (Figura 8-1) o en la ventana de comandos (Figura 8-2). El *Editor/Debugger* se abre con un fichero en blanco en el cual crearemos el M-fichero, es decir, un fichero con código de programación MATLAB (Figura 8-3). Cuando se trate de abrir un M-fichero ya existente se utiliza *File→ Open* en el escritorio de MATLAB (Figura 8-1) o, alternativamente, se puede utilizar el comando *Open* en la ventana de comandos (Figura 8-2). También se puede abrir el *Editor/Debugger* haciendo clic con el botón derecho del ratón en el interior de la ventana *Current Directory* y eligiendo *New → M-file* en el menú emergente resultante (Figura 8-4). La opción *Open* de este menú abre un M-fichero existente. Se pueden abrir varios M-ficheros simultáneamente, en cuyo caso aparecerán en ventanas diferentes.

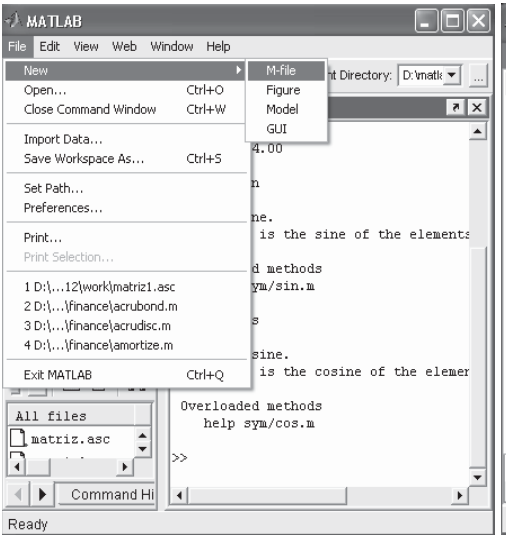


Figura 8-1

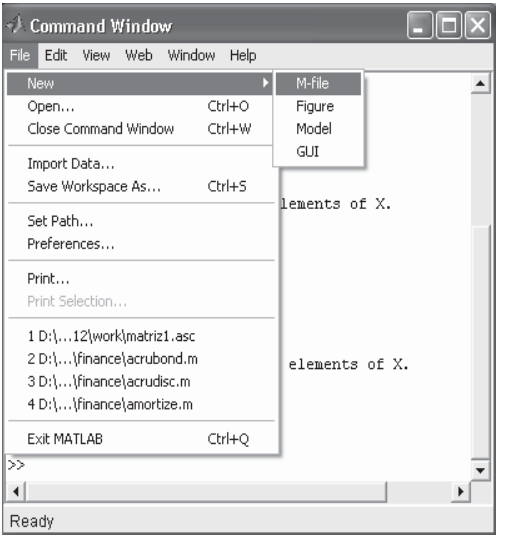


Figura 8-2

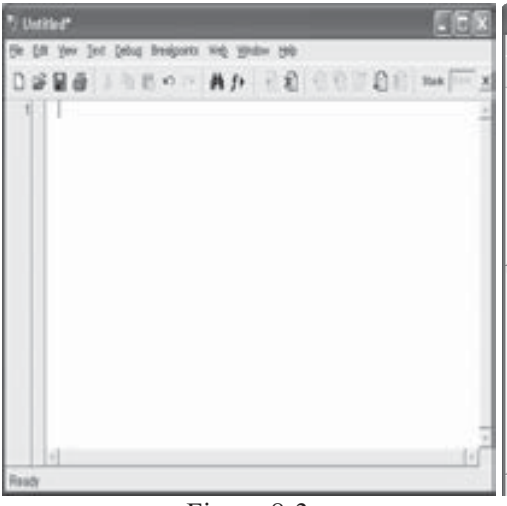


Figura 8-3

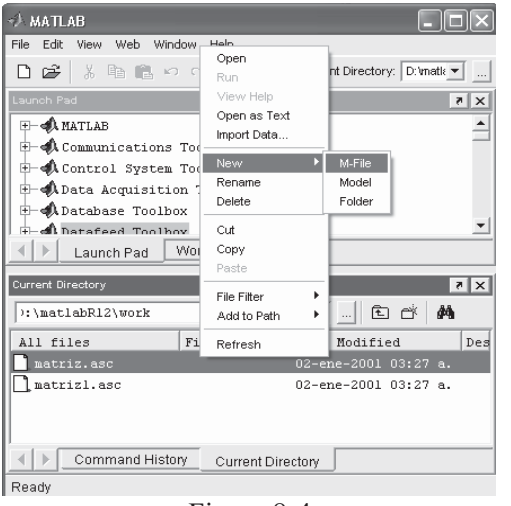


Figura 8-4

La Figura 8-5 muestra las funciones de los iconos del *Editor/Debugger*.

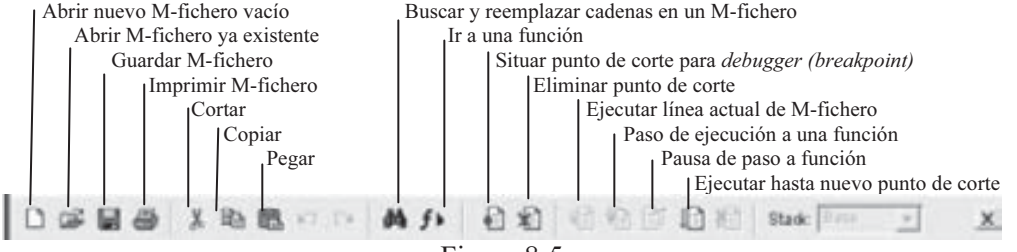


Figura 8-5

Scripts

Los scripts son el tipo de M-fichero más sencillo posible. Un script no tiene argumentos de entrada ni de salida. Sencillamente está formado por instrucciones MATLAB que se ejecutan secuencialmente y que podrían submitirse igualmente en serie en la ventana de comandos. Los scripts operan con datos existentes en el espacio de trabajo o con nuevos datos creados por el propio script. Cualquier variable que se cree mediante un script permanecerá en el espacio de trabajo y podrá utilizarse en cálculos posteriores después de finalizar el script.

A continuación se presenta un ejemplo de script que genera varias curvas en polares representando pétalos de flores. Una vez escrita la sintaxis del script en el editor (Figura 8-6), se guarda en la librería de trabajo (*work*) y simultáneamente se ejecuta, haciendo clic en el botón  o utilizando la opción *Save and run* del menú *Debug* (o presionando F5). Para pasar de un gráfico al siguiente basta con pulsar ENTER.

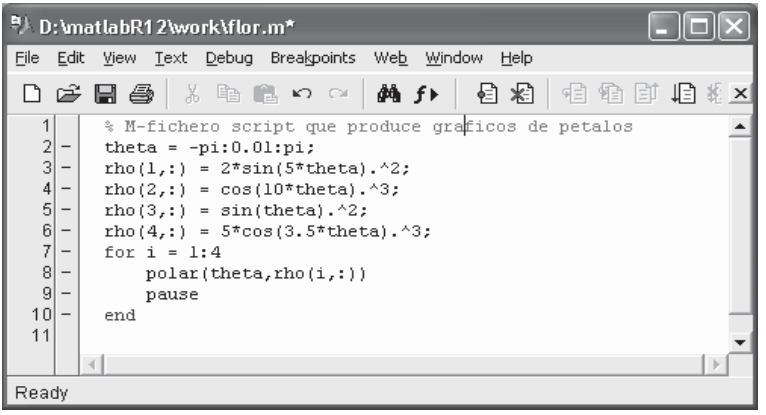


Figura 8-6

Figura 8-7

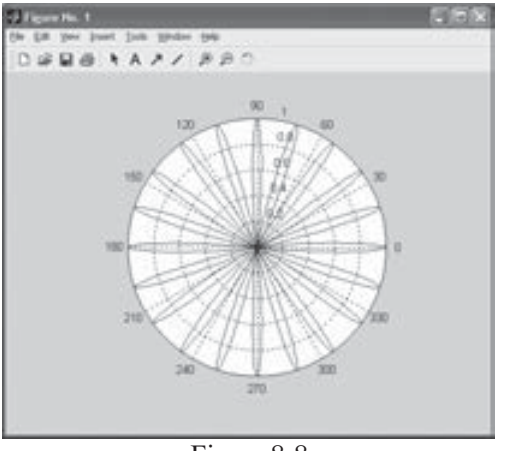


Figura 8-8

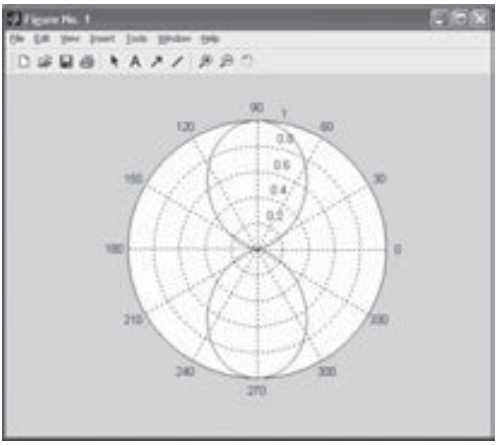


Figura 8-9

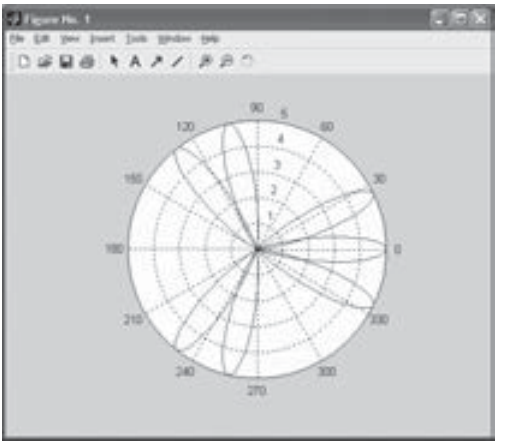


Figura 8-10

Funciones y M-ficheros. *Function, eval y feval*

Ya sabemos que MATLAB dispone de gran variedad de funciones para usar en el trabajo cotidiano con el programa. Pero, además, el programa ofrece la posibilidad de definir funciones a medida. El camino más usual para definir una función a medida es escribir su definición en un fichero texto, denominado M-fichero, que será permanente y que, por lo tanto, permitirá el uso posterior de la función siempre que se requiera.

MATLAB es habitualmente utilizado en *modo comando* (o *interactivo*), en cuyo caso se submite un comando que se escribe en una única línea sobre la ventana de comandos y se procesa de inmediato. Pero MATLAB también permite la ejecución de conjuntos de comandos en modo *batch*, en cuyo caso se submiten secuencialmente un conjunto de comandos escritos previamente en un fichero. Este fichero (M-fichero) ha de ser almacenado en disco con la extensión "*.m*" en el camino de subdirectorios de MATLAB, utilizando cualquier editor ASCII o la subopción *M-file* de la opción *New* del menú *File* de la barra superior de menús, la cual nos lleva a un editor de texto que permitirá escribir las líneas de comandos y guardar el fichero con un determinado nombre. La opción *Open M-File* del menú *File* de la barra superior de menús permite editar cualquier M-fichero preexistente.

Para ejecutar un M-fichero basta con teclear su nombre (sin extensión) en modo interactivo sobre la ventana de comandos y pulsar *Enter*. MATLAB interpreta secuencialmente todos los comandos o sentencias incluidos en las diferentes líneas del M-fichero y los ejecuta. Normalmente no aparecen en pantalla los literales de los comandos que MATLAB va interpretando, salvo que se active el comando *echo on*, y sólo se van viendo los resultados de las ejecuciones sucesivas de los comandos interpretados. Normalmente, el trabajo en modo batch es útil cuando se procesan conjuntos muy largos de comandos de escritura tediosa y con propensión a cometer errores, pero la mayor utilidad se presenta en la automatización de procesos. Además, en las líneas de un M-fichero se pueden introducir textos explicativos y comentarios, empezando cada línea al efecto por el símbolo %. Con el comando *Help* se accede al texto explicativo de un M-fichero.

El comando *function* permite la definición de funciones a medida en MATLAB, constituyendo una de la aplicaciones más útiles de los M-ficheros. La sintaxis de este comando es la siguiente:

function parámetros_salida = nombre_función(parámetros_entrada)
cuerpo de la función

Una vez que la función ha sido definida, se guarda en un M-fichero para su uso posterior. Es útil también introducir algún texto explicativo en la sintaxis de la función (entre %), al cual se accederá con el comando de ayuda *Help*.

Cuando los parámetros de salida son más de uno, se sitúan entre corchetes y separados por comas. Si los parámetros de entrada son más de uno, se separan por comas. El cuerpo de la función es la sintaxis que la define, y debe incluir comandos o instrucciones que asignen valores a los parámetros de salida. Cada comando o instrucción del cuerpo suele ir en una línea que finaliza con una coma o con un punto y coma en caso de que se definan variables (para evitar repeticiones en las salidas al ejecutar la función). La función se guarda en el M-fichero de nombre *nombre_función.m.*

A continuación vamos a definir la función *fun1(x)=x^3-2x+cosx*, creando el correspondiente M-fichero de nombre *fun1.m*. Para introducir dicha sintaxis en MATLAB se selecciona la subopción *M-file* de la opción *New* del menú *File* de la barra superior de menús (o se hace clic en el botón de la barra de herramientas de MATLAB), la cual nos lleva al editor de texto *MATLAB Editor/Debugger* que permitirá escribir las líneas de comandos relativas a la sintaxis de la función, tal y como se indica en la Figura 8-11.

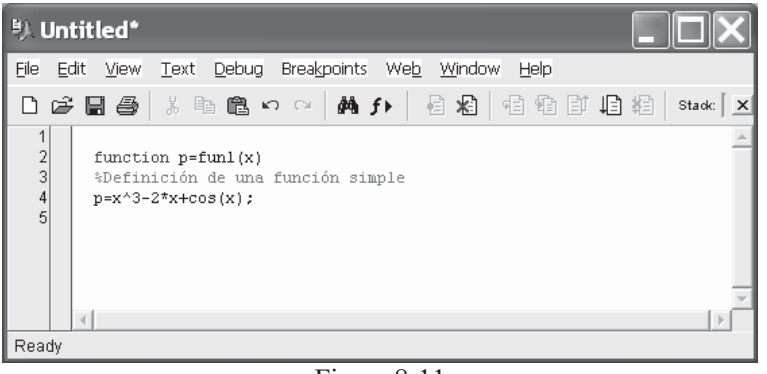


Figura 8-11

Para guardar de forma definitiva dicha sintaxis en MATLAB se selecciona la opción *Save* del menú *File* de la barra superior de menús del *MATLAB Editor/Debugger*, la cual nos lleva a la caja de diálogo *Guardar* de la Figura 8-12, mediante la cual podemos guardar nuestra función con el nombre deseado y en el subdirectorio que se indique como ruta en el campo *Nombre de archivo*. También puede hacerse clic en el botón o utilizar la opción *Save and run* del menú *Debug*.

Se recomienda guardar las funciones como ficheros de nombre igual al nombre de la función y en el subdirectorio de trabajo por defecto de MATLAB *C :\MATLAB6p1\work*. Precisamente es la situación que presenta el programa por defecto al intentar guardar cualquier función a medida y al utilizar el icono  o la opción *Save and run* del menú *Debug* para ejecutar y guardar simultáneamente el M-fichero actual del editor.

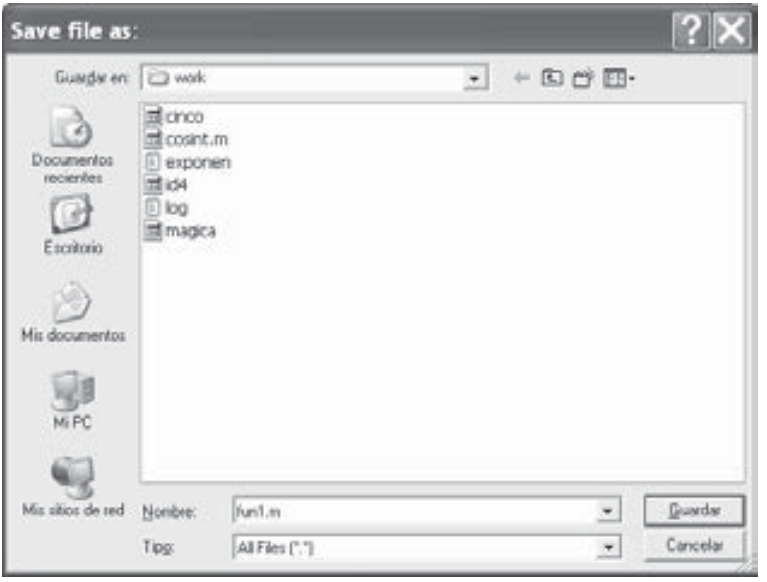


Figura 8-12

Una vez definida y guardada la función anterior en un M-fichero, se puede utilizar desde la ventana de comandos. Por ejemplo, para hallar el valor de la función en $3\pi/2$ escribimos sobre la ventana de comandos usual de MATLAB:

```
>> fun1(3*pi/2)

ans =

   95.2214
```

Para pedir ayuda sobre la función anterior (suponiendo que fue previamente introducida como comentario al elaborar el M-fichero que la define) se utiliza el comando *help*, como sigue:

```
>> help fun1

 Definición de una función simple
```

La evaluación de una función en sus argumentos (o parámetros de entrada) también puede realizarse a través del comando *feval*, cuya sintaxis es la siguiente:

feval('F',arg1,arg1,..,argn) *Evalúa la función F (M-fichero F.m) en los argumentos especificados arg1, arg2, ...,argn*

Como ejemplo construimos un M-fichero de nombre *ecuacion2.m* que contiene la función ecuacion2, cuyos argumentos son los tres coeficientes de la ecuación de segundo grado $ax^2+bx+c = 0$ y cuyas salidas son sus dos soluciones (Figura 8-13).

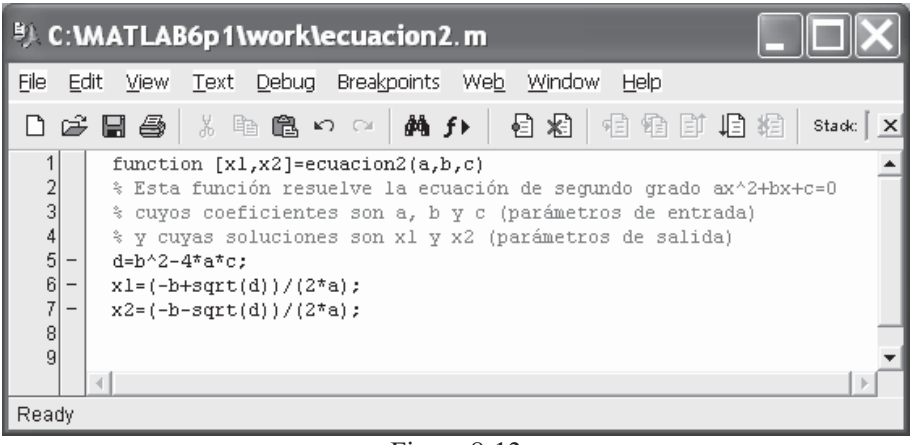


Figura 8-13

Si ahora queremos resolver la ecuación $x^2+2x+3 = 0$ utilizando *feval*, escribimos en la ventana de comandos lo siguiente:

```
>> [x1,x2]=feval('ecuacion2',1,2,3)

x1 =

  -1.0000 + 1.4142i

x2 =

  -1.0000 - 1.4142i
```

También se puede resolver la ecuación de segundo grado como sigue:

```
>> [x1,x2]=ecuacion2(1,2,3)

x1 =
  -1.0000 + 1.4142i

x2 =
  -1.0000 - 1.4142i
```

Si pedimos ayuda sobre la función ecuación2 tenemos lo siguiente:

```
>> help ecuacion2

  Esta función resuelve la ecuación de segundo grado ax^2+bx+c=0
  cuyos coeficientes son a, b y c (parámetros de entrada)
  y cuyas soluciones son x1 y x2 (parámetros de salida)
```

La evaluación de una función en sus argumentos (o parámetros de entrada) cuando son cadenas se realiza a través del comando *eval*, cuya sintaxis es la siguiente:

eval(expresión) *Ejecuta la expresión cuando es una cadena*

Como ejemplo evaluamos una expresión carácter que define la matriz mágica de orden 4.

```
>> n=4;
>> eval(['M' num2str(n) ' = magic(n)'])

M4 =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Variables locales y globales

Normalmente, cada función de MATLAB definida como un M-fichero contiene sus variables como variables locales, es decir, como variables que tienen su efecto en el interior del M-fichero, separadamente de otros M-ficheros y del espacio de trabajo base. No obstante, es posible definir variables en el interior de M-ficheros de modo que tengan efecto simultáneamente en el interior de otros M-ficheros y en el propio espacio de trabajo base. Para ello es preciso definir las variables como globales con el comando GLOBAL, cuya sintaxis es la siguiente:

GLOBAL x y z… *Define las variables x, y, z… como globales*

Cualquier variable definida como global en el interior de una función es accesible separadamente para el resto de las funciones y para el espacio de trabajo base línea de comandos. Si la variable global no existe, la primera vez que se define en la sentencia GLOBAL, se inicializará como la matriz vacía. Si ya existe una variable con el mismo nombre que la que se está definiendo como global, MATLAB emite un mensaje de peligro y cambia el valor de la variable a enlazar como global. Es conveniente declarar una variable como global en cada función que necesite acceso a ella, y también en la línea de comandos, para tener acceso a ella desde el espacio de trabajo base. En el interior de las funciones el comando GLOBAL se sitúa al principio (antes de cualquier ocurrencia de la variable).

Como ejemplo, supongamos que se quiere estudiar el efecto de los coeficientes de interacción α y β en el modelo de Lotka-Volterra:

$$\dot{y}_1 = y_1 - \alpha y_1 y_2$$
$$\dot{y}_2 = -y_2 + \beta y_1 y_2$$

Para ello creamos la función *lotka* en el M-fichero *lotka.m* de la Figura 8-14.

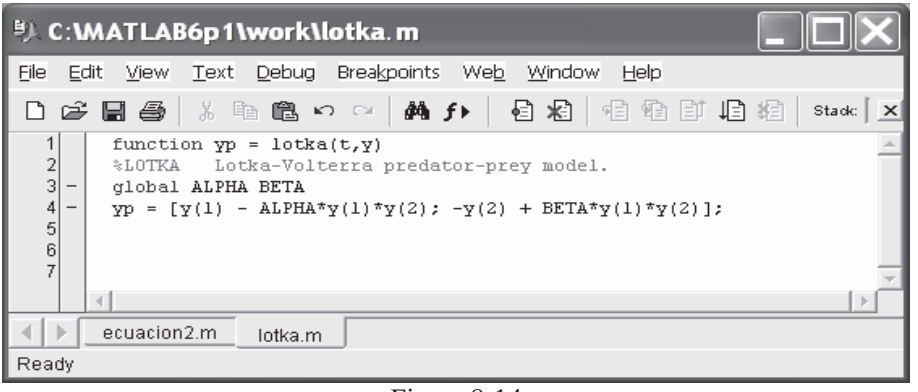


Figura 8-14

Si posteriormente sobre la línea de comandos escribimos lo siguiente:

```
>> global ALPHA BETA
ALPHA = 0.01
BETA = 0.02
```

ya se podrán utilizar estos valores globales para α y β en el interior del M-fichero *lotka.m* (sin tener que volver a especificarlos). Por ejemplo, podrá realizarse una representación (Figura 8-15) con la sintaxis siguiente:

```
>> [t,y] = ode23('lotka',0,10,[1; 1]); plot(t,y)
```

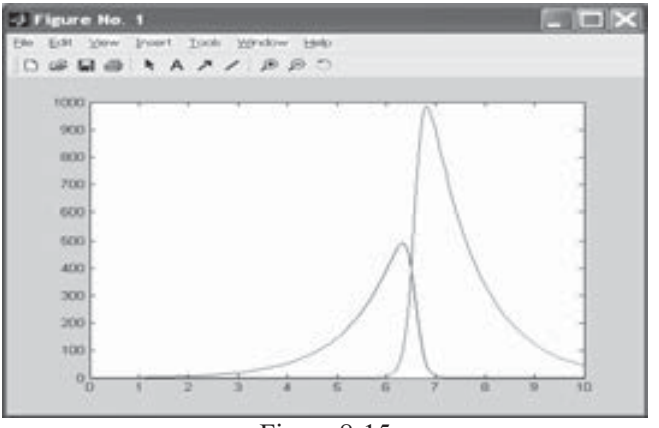


Figura 8-15

Tipos de datos

En MATLAB hay 14 tipos de datos diferentes que se representan en la Figura 8-16.

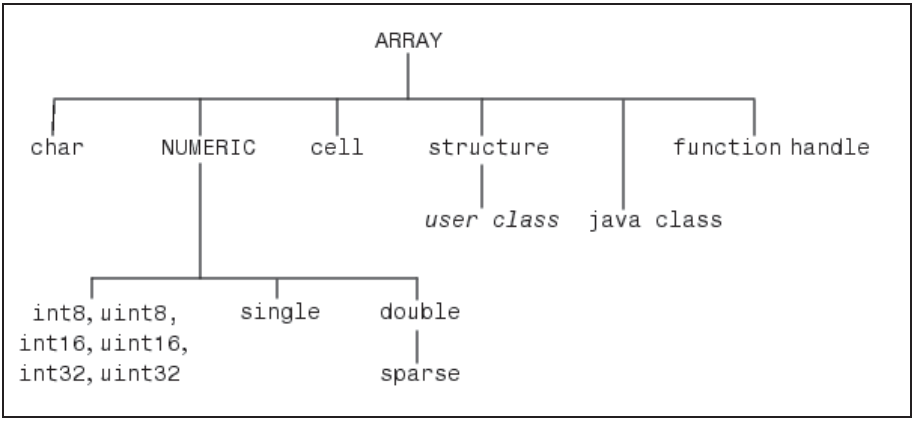


Figura 8-16

A continuación se detallan los distintos tipos de datos:

Tipo de dato	*Ejemplo*	*Descripción*
`single`	`3*10^38`	*Precisión numérica simple. Requiere menor almacenamiento que la doble precisión, pero ésta es menor. Este tipo de datos no debe usarse en operaciones matemáticas.*
`double`	`3*10^300` `5+6i`	*Doble precisión numérica. Es el tipo de variable más común en MATLAB*
`sparse`	`speye(5)`	*Matriz dispersa con doble precisión.*
`int8, uint8, int16, uint16, int32, uint32`	`uint8(magic(3))`	*Enteros con y sin signo y con 8, 16, y 32 bits de longitud. Posibilitan usar cantidades enteras con manejo eficiente de memoria. Este tipo de datos no debe usarse en operaciones matemáticas.*
`char`	`'Hello'`	*Caracteres (cada carácter tiene una longitud de 16 bits).*
`cell`	`{17 'hello' eye(2)}`	*Celda (contiene datos de similar tamaño)*

`structure`	`a.day = 12;`
	`a.color = 'Red';`
	`a.mat = magic(3);`
		Estructura (contiene celdas de similar tamaño)

`user class`	`inline('sin(x)')`	*Clase MATLAB (creada con funciones)*
`java class`	`java.awt.Frame`	*Clase Java (definida en API o propia con Java)*
`function handle`	`@humps`	*Maneja funciones MATLAB. Puede ser pasada en una lista de argumentos y evaluada con* `feval`.

Control de flujo: bucles FOR, WHILE e IF ELSEIF

El uso de funciones recursivas, condicionales y definidas a trozos es muy habitual en matemáticas. Para la definición de este tipo de funciones es necesario el manejo de bucles. Como es natural, la definición de las funciones se hará a través de M-ficheros.

El bucle FOR

MATLAB dispone de su propia versión de la sentencia DO (definida en la sintaxis de la mayoría de los lenguajes de programación). Esta sentencia permite ejecutar de forma repetitiva un comando o grupo de comandos varias veces. Por ejemplo:

```
» for i=1:3, x(i)=0, end

x =
     0
x =
     0     0
x =
     0     0     0
```

La forma general de un bucle FOR es la siguiente:

for variable = expresión
comandos
end

El bucle siempre empieza con la cláusula *for* y termina con la cláusula *end*, e incluye en su interior todo un conjunto de comandos que se separan por comas. Si algún comando define una variable, se finaliza con punto y coma para evitar repeticiones en la salida. Normalmente, los bucles se utilizan en la sintaxis de M-ficheros. Veamos un ejemplo (Figura 8-17):

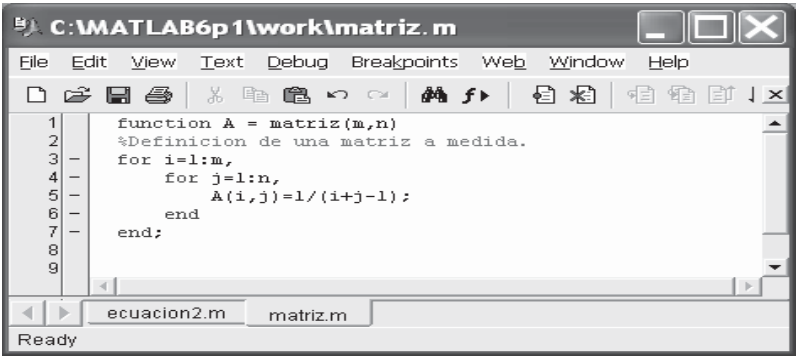


Figura 8-17

En este bucle hemos definido la matriz de Hilbert de orden *(m,n)*. Si guardamos su contenido como un M-fichero de nombre *matriz.m*, podremos construir cualquier matriz de Hilbert posteriormente ejecutando el M-fichero y especificando los valores para las variables *m* y *n* (dimensiones de la matriz) como se indica a continuación:

```
>> M=matriz(4,5)

M =

    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
```

El bucle WHILE

MATLAB dispone de su propia versión de la sentencia WHILE definida en la sintaxis de la mayoría de los lenguajes de programación. Esta sentencia permite ejecutar de forma repetitiva un comando o grupo de comandos un número determinado de veces mientras se cumple una condición lógica especificada. La sintaxis general de este bucle es la siguiente:

while condición
    comandos
end

El bucle siempre empieza con la cláusula *while* seguida de una condición, y termina con la cláusula *end*, e incluye en su interior todo un conjunto de comandos que se separan por comas y que se ejecutan mientras se cumple la condición. Si algún comando define una variable, se finaliza con punto y coma para evitar repeticiones en la salida. Como ejemplo escribimos un M-fichero (Figura 8-18) que se guarda con el nombre *while1.m*, cuya ejecución permite calcular el mayor número cuyo factorial no excede a 10^{100}.

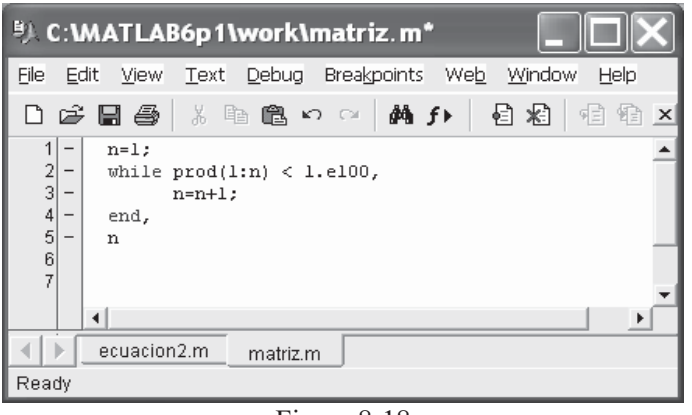


Figura 8-18

Ahora ejecutamos el M-fichero.

```
>> while1

n =

    70
```

El bucle *IF ELSEIF ELSE END*

MATLAB, al igual que la mayoría de los lenguajes de programación estructurada, también incorpora la estructura IF-ELSEIF-ELSE-END. Mediante esta estructura, se pueden ejecutar secuencias de comandos si se cumplen determinadas condiciones. La sintaxis del bucle es la siguiente:

```
if condición
  comandos
end
```

En este caso se ejecutan los comandos si la condición es cierta. Pero la sintaxis de este bucle puede ser más general.

```
if condición
  comandos1
else
  comandos2
end
```

En este caso se ejecutan los *comandos1* si la condición es cierta, y se ejecutan los *comandos2* si la condición es falsa.

Las sentencias IF, al igual que las sentencias FOR, pueden ser anidadas. Cuando se anidan varias sentencias IF se utiliza la sentencia ELSEIF, cuya sintaxis general es la siguiente:

```
if condición1
  comandos1
elseif condición2
  comandos2
elseif condición3
  comandos3
.
.
else
end
```

En este caso se ejecutan los *comandos1* si *condición1* es cierta, se ejecutan los *comandos2* si *condición1* es falsa y *condición2* es cierta, se ejecutan los *comandos3* si *condición1* y *condición2* son falsas y *condición3* es cierta, y así sucesivamente.

La sintaxis anidada anterior es equivalente, pero más rápida de ejecución, a la sintaxis sin anidar siguiente:

```
if condición1
  comandos1
else
    if condición2
      comandos2
   else
       if condición3
         comandos3
       else
.
.
      end
   end
end
```

Veamos como ejemplo (Figura 8-19) el M-fichero de nombre *else1.m* siguiente:

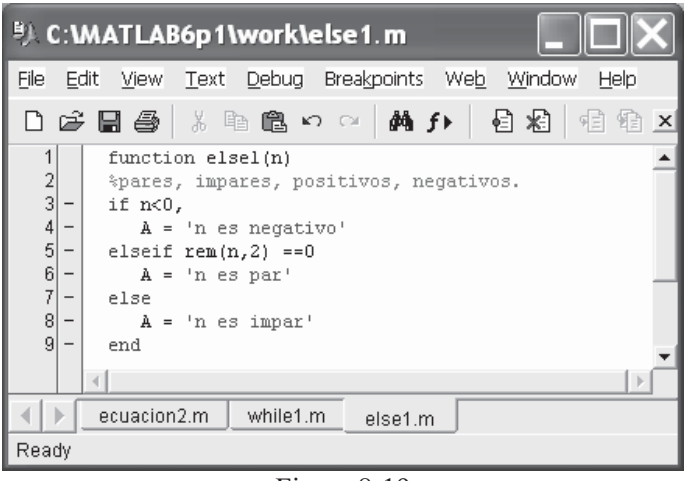


Figura 8-19

Al ejecutarlo obtendremos el tipo de número (negativo, par o impar) para un valor de *n* especificado:

```
>> else1(8), else1(5), else1(-10)
```

```
A =
n es par
A =
n es impar
A =
n es negativo
```

SWITCH y *CASE*

La instrucción *switch* ejecuta ciertas sentencias basadas en el valor de una variable o expresión. Su sintaxis básica es la siguiente:

```
switch expresión (escalar o cadena)
   case valor1
      sentencias %  Ejecuta si expresión es valor1
   case valor2
      sentencias %  Ejecuta si expresión es valor2
   .
   .
   .
   otherwise
      sentenciass %  Ejecuta si expression no cumple
                  %  ningún caso
end
```

A continuación se presenta un ejemplo de una función que devuelve -1, 0, 1 u otro número según sea la entrada (Figura 8-20).

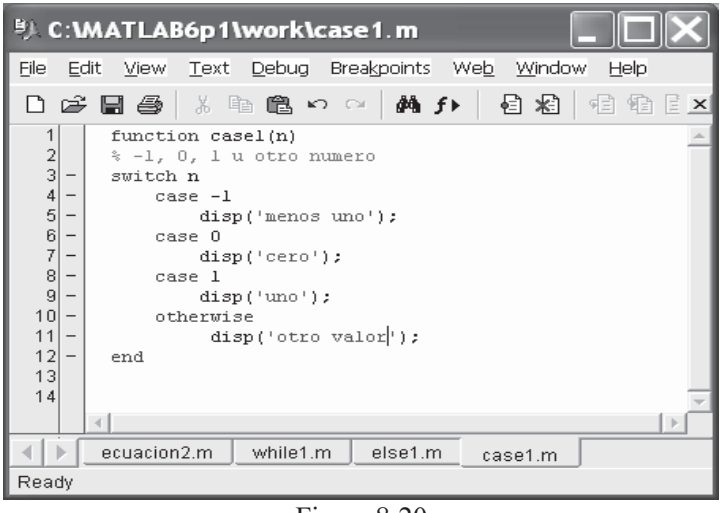


Figura 8-20

Después ejecutamos el ejemplo anterior.

```
>> case1(25)
otro valor

>> case1(-1)
menos uno
```

CONTINUE

La instrucción *continue* pasa el control a la iteración siguiente en un bucle *for* o *while* en el cual aparece ignorando las restantes instrucciones en el cuerpo del bucle. A continuación se muestra el M-fichero *continue.m* (Figura 8-21) que cuenta las líneas de código en el fichero *magig.m* ignorando las líneas blancas y los comentarios.

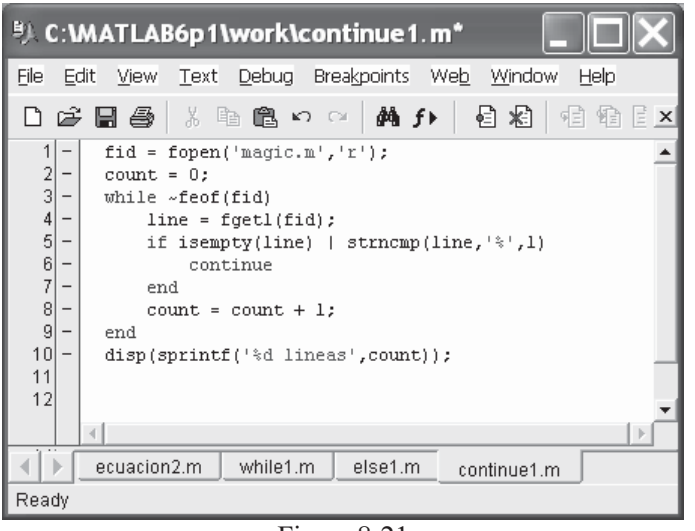


Figura 8-21

Seguidamente ejecutamos el M-fichero.

```
>> continue1
25 lineas
```

BREAK

La instrucción *break* finaliza la ejecución de un bucle *for* o *while* en el cual aparece continuando la ejecución en la siguiente instrucción fuera del bucle. A continuación se muestra el M-fichero *break1.m* (Figura 8-22) que lee las líneas de código en el fichero *fft.m* saliendo del bucle cuando se encuentre la primera línea vacía.

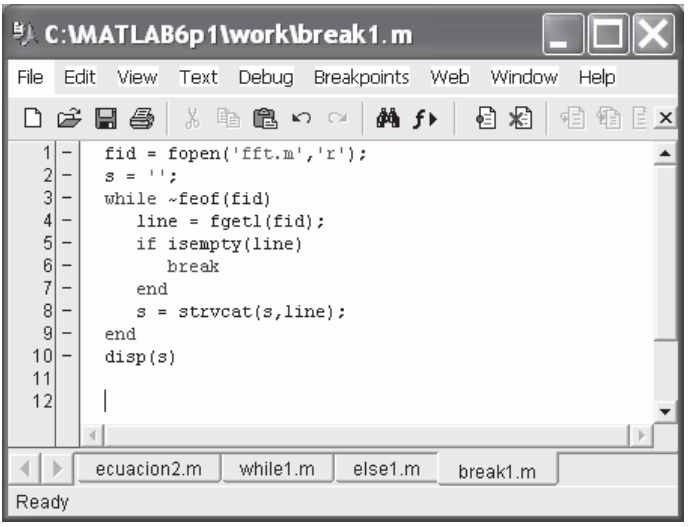


Figura 8-22

Después ejecutamos el M-fichero.

```
>> break1

%FFT Discrete Fourier transform.
%    FFT(X) is the discrete Fourier transform (DFT) of vector X.  For
%    matrices, the FFT operation is applied to each column. For N-D
%    arrays, the FFT operation operates on the first non-singleton
%    dimension.
%
%    FFT(X,N) is the N-point FFT, padded with zeros if X has less
%    than N points and truncated if it has more.
%
%    FFT(X,[],DIM) or FFT(X,N,DIM) applies the FFT operation across the
%    dimension DIM.
%
%    For length N input vector x, the DFT is a length N vector X,
%    with elements
%                     N
%       X(k) =       sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
%                    n=1
%    The inverse DFT (computed by IFFT) is given by
%                     N
%       x(n) = (1/N) sum  X(k)*exp( j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.
%                    k=1
%
%    See also IFFT, FFT2, IFFT2, FFTSHIFT.
```

TRY ... CATCH

Las instrucciones entre *try* y *catch* se ejecutan mientras no ocurra un error. La instrucción *lasterr* se utiliza para ver loa causa del error. La sintaxis general de la instrucción es la siguiente:

try,
  instrucción,
  ...,
  instrucción,
catch,
  instrucción,
  ...,
  instrucción,
end

RETURN

La instrucción *return* finaliza la secuencia actual de comandos y devuelve el control a la función invocada o al teclado. A continuación se presenta un ejemplo (Figura 8-23) que calcula el determinante de una matriz que no es vacía. Si la matriz es vacía se obtiene el valor 1.

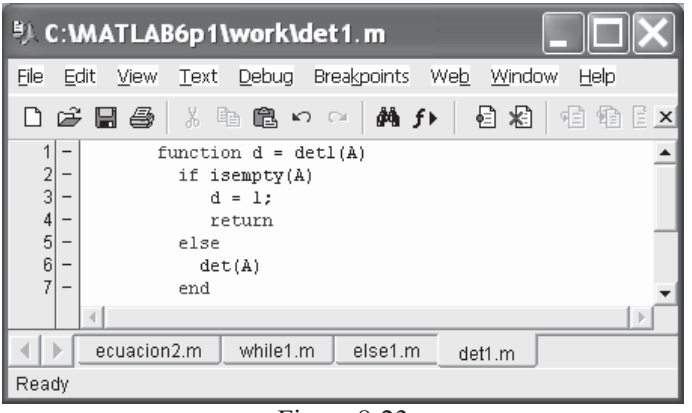


Figura 8-23

Posteriormente ejecutamos la función para una matriz no vacía.

```
>> A=[-1,-1,1;1,0,1;1,1,1]

A =
```

```
    -1    -1     1
     1     0     1
     1     1     1
>> det1(A)
ans =
     2
```

Ahora aplicamos la función a una matriz vacía.

```
>> B=[]
B =
     []
>> det1(B)
ans =
     1
```

Subfunciones

Las funciones definidas mediante M-ficheros pueden contener código para más de una función. La función principal en el M-fichero se denomina *función primaria*, que es precisamente la función que invoca el M-fichero. Pero adicionalmente puede haber subfunciones colgando de la función primaria y que sólo son visibles para dicha función primaria o para otra subfunción dentro del mismo M-fichero. Cada subfunción comienza con su propia línea de definición de función. En la Figura 8-24 se presenta un ejemplo.

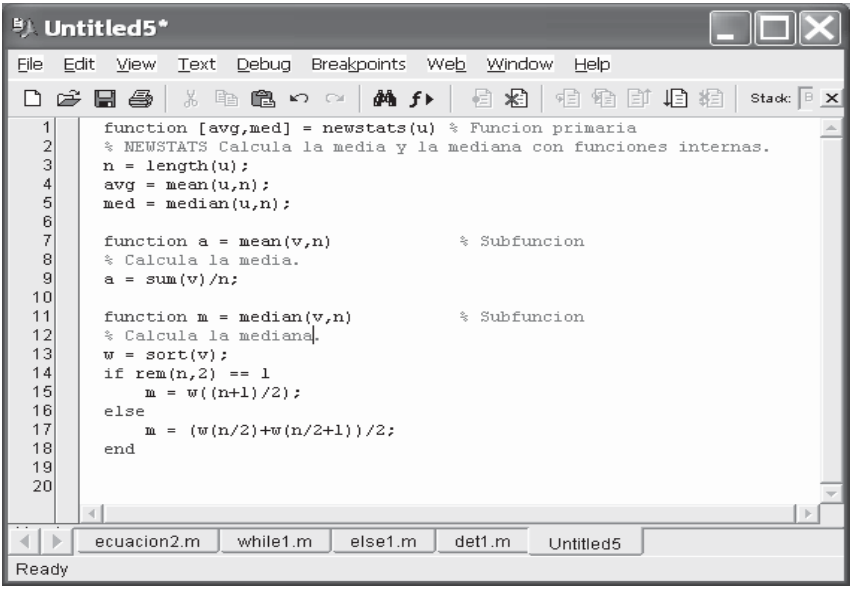


Figura 8-24

Las subfunciones *mean* y *median* calculan la media y la mediana de la lista de entrada. La función primaria *newstats* determina la longitud de la lista y llama a las subfunciones pasándoles la lista de longitud *n*. A la hora de ejecutar la función principal, basta darle su entrada correspondiente (una lista de valores para los que se calculará su media y su mediana) y las llamadas a las subfunciones se realizan automáticamente tal y como se ve a continuación.

```
>> [media, mediana] = newstats([10,20,3,4,5,6])

media =

     8


mediana =

    5.5000
```

Comandos en M-ficheros

MATLAB ofrece ciertos comandos de procedimiento que se utilizan muy a menudo en la escritura de M-ficheros. Entre ellos tenemos los siguientes:

echo on	*Permite ver en pantalla los literales de los comandos de las líneas de un M-fichero mientras se ejecuta*
echo off	*No presenta, en pantalla los literales de los comandos de las líneas de un M-fichero (opción por defecto)*
pause	*Causa la interrupción de la ejecución de un M-fichero hasta que el usuario pulse una tecla para continuar*
pause(n)	*Causa la interrupción de la ejecución de un M-fichero durante n segundos*
pause off	*Deshabilita pause y pause(n)*
pause on	*Habilita pause y pause(n)*
keyboard	*Causa la interrupción de la ejecución de un M-fichero y pasa el control al teclado para que el usuario realice otras tareas. Se vuelve a la ejecución del M-fichero tecleando sobre la ventana de comandos el comando return y pulsando Enter.*
return	*Nos devuelve a la ejecución de un M-fichero después de una interrupción*
break	*Causa la interrupción de un bucle prematuramente*
clc	*Limpia la ventana de comandos*
home	*Oculta el cursor*
more on	*Habilita la paginación de la salida de MATLAB en la ventana de comandos*
more off	*Desactiva la paginación de la salida de MATLAB en la ventana de comandos*
more(N)	*Sitúa el tamaño de página en N líneas*
menu	*Permite elegir entre varios tipos de menú para el input del usuario*

Funciones relativas a arrays de celdas

Un array es una colección de elementos individuales bien ordenada. Se trata sencillamente de una lista de elementos cada uno de los cuales está asociado a un entero positivo llamado índice del elemento, que representa el lugar (posición) que ocupa dentro de la lista. Lo esencial es que cada elemento está asociado a un índice, que también puede ser cero o negativo, que le identifica plenamente, de tal forma que para realizar cambios en algún elemento del array basta con referirse a sus índices. Los arrays pueden ser de una o varias dimensiones, según tengan uno o varios conjuntos de índices que identifiquen a sus elementos. Los comandos o funciones más importantes que habilita MATLAB para el trabajo con arrays de celdas son los siguientes:

c = cell(n)	*Crea un array nxn cuyas celdas son matrices vacías*
c = cell(m,n)	*Crea un array mxn cuyas celdas son matrices vacías*
c = cell([m n])	*Crea un array mxn cuyas celdas son matrices vacías*
c = cell(m,n,p,...)	*Crea un array mxnxpx… de matrices vacías*
c = cell([m n p ...])	*Crea un array mxnxpx… de matrices vacías*
c = cell(size(A))	*Crea un array de matrices vacías del mismo tamaño que A*
D = cellfun('f',C)	*Aplica la function f (isempty, islogical, isreal, length, ndims o prodofsize) a cada elemento del array C*
D = cellfun('size',C,k)	*Da el tamaño de cada elemento de C en la dimensión k*
D = cellfun('isclass',C,clas)	*Da true para cada elemento de C que se corresponde con clas*
c=cellstr(S)	*Sitúa cada fila del array carácter S en celdas separadas de c*
s = cell2struct(c,fields,dim)	*Convierte el array c a la estructura S incorporando la dimensión dim de c a los campos de s*
celldisp(C)	*Muestra el contenido del array C*
celldisp(C,nombre)	*Muestra el contenido del array C en la variable nombre*
cellplot(c)	*Grafica la estructura del array c*
cellplot(c,'leyenda')	*Grafica la estructura del array c e incorpora una leyenda*
c = num2cell(A)	*Convierte el array numérica A en el array de celdas c*
c = num2cell(A,dims)	*Convierte el array numérica A en el array de celdas c situando las dimensiones dadas en celdas separadas*

Como primer ejemplo creamos un array de celdas del mismo tamaño que la matriz de unos cuadrada de orden dos.

```
>> A = ones(2,2)

A =

     1     1
     1     1
```

```
>> c = cell(size(A))

c =

     []     []
     []     []
```

A continuación definimos un array de celdas 2x3 elemento a elemento, presentamos su contenido y aplicamos a sus celdas distintas funciones.

```
>> C{1,1} = [1 2; 4 5];
C{1,2} = 'Name';
C{1,3} = pi;
C{2,1} = 2 + 4i;
C{2,2} = 7;
C{2,3} = magic(3);

>> C

C =

          [2x2 double]    'Name'    [    3.1416]
    [2.0000+ 4.0000i]    [   7]    [3x3 double]

>> D = cellfun('isreal',C)

D =

     1     1     1
     0     1     1

>> len = cellfun('length',C)

len =

     2     4     1
     1     1     3

>> isdbl = cellfun('isclass',C,'double')

isdbl =

     1     0     1
     1     1     1
```

Después se presenta el contenido del array de celdas C anterior celda a celda.

```
>> celldisp(C)
 C{1,1} =
```

```
     1     2
     4     5

C{2,1} =

   2.0000 + 4.0000i

C{1,2} =

Name

C{2,2} =

     7

C{1,3} =

    3.1416

C{2,3} =

     8     1     6
     3     5     7
     4     9     2
```

En el ejemplo siguiente graficamos el array C (Figura 8-25).

```
>> cellplot(C)
```

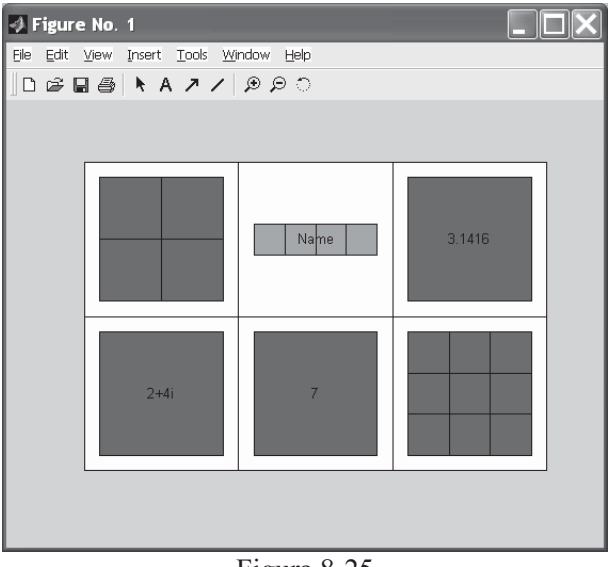


Figura 8-25

Funciones de arrays multidimensionales

Existe un grupo de funciones en MATLAB para trabajar con arrays multidimensionales. Son las siguientes:

C = cat(dim,A,B)	*Concatena los arrays A y B según la dimensión dim*
C = cat(dim,A1,A2,A3,A4...)	*Concatena los arrays A1, A2,... según la dimensión dim*
B = flipdim(A,dim)	*Da vuelta al array A a lo largo de la dimension dim*
[I,J] = ind2sub(*siz*,IND)	*Da los arrays I y J con subíndices de fila y columna equivalentes a los índices de la matriz IND de tamaño siz*
[I1,I2,I3,...,In] = ind2sub(*siz*,IND)	*Da los arrays I1, I2, ... con subíndices de fila y columna equivalentes a los índices de la matriz IND de tamaño siz*
A = ipermute(B,*order*)	*Invierte las dimensiones del array multidimensional D según los valores del vector order*
[X1,X2,X3,...] = ndgrid(x1,x2,x3,...)	*Transforma el dominio especificado por los vectores x1, x2, ... en los arrays X1, X2, ... que pueden ser usados para evaluación de funciones de varias variables y para interpolación.*
[X1,X2,...] = ndgrid(x)	*Equivale a ndgrid(x,x,x,...)*
n = ndims(A)	*Da el número de dimensiones del array A*
B = permute(A,*orden*)	*Permuta las dimensiones del array A según el vector orden*
B = reshape(A,m,n)	*Da la matriz B mxn cuyos elementos son las columnas de A*
B = reshape(A,m,n,p,...)	*Da el array B con los elementos del array A reestructurados según las dimensiones mxnxpx...*
B = reshape(A,[m n p...])	*Equivale a B = reshape(A,m,n,p,...)*
B = reshape(A,siz)	*Da el array B con los elementos del array A reestructurados según las dimensiones del vector siz*
B = shiftdim(X,n)	*Cambia las dimensiones de X por n*
[B,nshifts] = shiftdim(X)	*Da el array B con el mismo número de elementos que X pero con nshifts dimensión cambiadas*
B=squeeze(A)	*Da el array B con el mismo número de elementos que A pero con todas las dimensiones cambiadas*
IND = sub2ind(siz,I,J)	*Da el índice lineal equivalente a la fila y columna I, J de la matriz de tamaño siz*
IND = sub2ind(siz,I1,I2,...,In)	*Da el índice lineal equivalente a los n subíndices I1, I2, ..., In de la matriz de tamaño siz*

Como primer ejemplo concatenamos las matrices mágica y de Pascal de orden 3.

```
>> A = magic(3); B = pascal(3);
>> C = cat(4,A,B)
```

```
C(:,:,1,1) =

     8     1     6
     3     5     7
     4     9     2


C(:,:,1,2) =

     1     1     1
     1     2     3
     1     3     6
```

En el ejemplo siguiente se da vuelta a la matriz de Rosser.

```
>> R=rosser

R =

   611   196  -192   407    -8   -52   -49    29
   196   899   113  -192   -71   -43    -8   -44
  -192   113   899   196    61    49     8    52
   407  -192   196   611     8    44    59   -23
    -8   -71    61     8   411  -599   208   208
   -52   -43    49    44  -599   411   208   208
   -49    -8     8    59   208   208    99  -911
    29   -44    52   -23   208   208  -911    99

>> flipdim(R,1)

ans =

    29   -44    52   -23   208   208  -911    99
   -49    -8     8    59   208   208    99  -911
   -52   -43    49    44  -599   411   208   208
    -8   -71    61     8   411  -599   208   208
   407  -192   196   611     8    44    59   -23
  -192   113   899   196    61    49     8    52
   196   899   113  -192   -71   -43    -8   -44
   611   196  -192   407    -8   -52   -49    29
```

A continuación definimos un array por concatenación y permutamos e invertimos sus elementos.

```
>> a = cat(3,eye(2),2*eye(2),3*eye(2))
```

```
a(:,:,1) =

     1     0
     0     1


a(:,:,2) =

     2     0
     0     2


a(:,:,3) =

     3     0
     0     3
```

```
>> B = permute(a,[3 2 1])

B(:,:,1) =

     1     0
     2     0
     3     0

B(:,:,2) =

     0     1
     0     2
     0     3
```

```
>> C = ipermute(B,[3 2 1])

C(:,:,1) =

     1     0
     0     1

C(:,:,2) =

     2     0
     0     2

C(:,:,3) =

     3     0
     0     3
```

En el ejemplo siguiente se evalúa la función $f(x_1, x_2) = x_1 e^{-x_1^2 - x_2^2}$ en el cuadrado [-2,2]x[-2,2] y se representa gráficamente (Figura 8-26).

```
>> [X1,X2] = ndgrid(-2:.2:2, -2:.2:2);
Z = X1 .* exp(-X1.^2 - X2.^2);
mesh(Z)
```

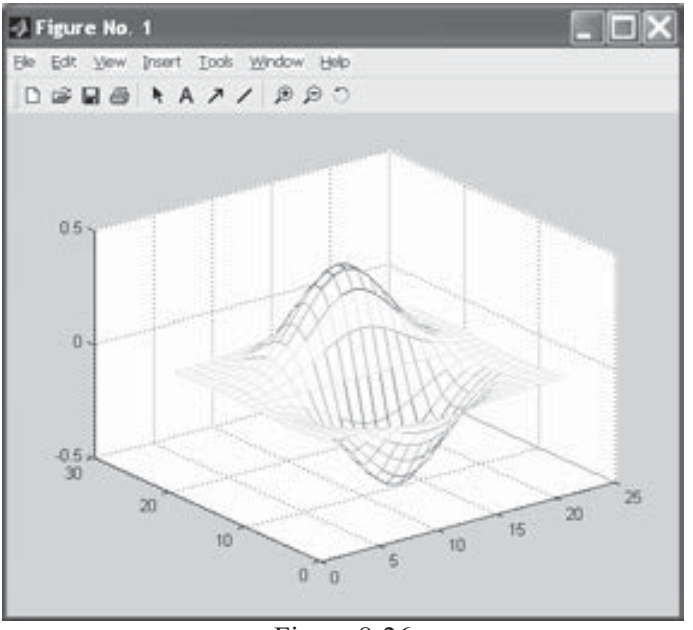


Figura 8-26

En el ejemplo siguiente redimensionamos una matriz aleatoria 3x4 a 2x6.

```
>> A=rand(3,4)

A =

    0.9501    0.4860    0.4565    0.4447
    0.2311    0.8913    0.0185    0.6154
    0.6068    0.7621    0.8214    0.7919

>> B = reshape(A,2,6)

B =

    0.9501    0.6068    0.8913    0.4565    0.8214    0.6154
    0.2311    0.4860    0.7621    0.0185    0.4447    0.7919
```

8.2 Métodos de análisis numérico en MATLAB

Las técnicas de programación en MATLAB permiten implementar la mayoría de los algoritmos relativos a métodos de cálculo numérico. Es posible diseñar programas para realizar integración y diferenciación numérica, resolución de ecuaciones diferenciales, optimización de funciones no lineales, etc. No obstante, el módulo básico de MATLAB dispone de funciones a medida para varios de estos algoritmos. En los párrafos siguientes se exponen estas funciones.

Optimización y ceros de funciones

Los comandos (funciones) que habilita el módulo básico de MATLAB para optimizar funciones y hallar sus ceros son los siguientes:

x = fminbnd(fun,x1,x2)	*Minimiza la función en el intervalo (x1x2)*
x = fminbnd(fun,x1,x2,options)	*Minimiza la función en el intervalo (x1x2) según la opción dada por optimset(…). Este último comando se explica más adelante.*
x = fminbnd(fun,x1,x2,options,P1,P2,...)	*Determina parámetros adicionales P1, P2, … a pasar a la función objetivo fun(x,P1,P2, ...)*
[x,fval] = fminbnd(...)	*Da además el valor de la función objetivo en x*
[x,fval,f] = fminbnd(...)	*Da además un indicador de convergencia f (f>0 indica convergencia a la solución, f>0 no convergencia y f=0 n.º de pasos excedido)*
[x,fval,f,output] = fminbnd(...)	*Da además información sobre la optimización (*`output.algorithm` *da el algoritmo usado,* `output.funcCount` *da el número de evaluaciones de fun y* `output.iterations` *da el número de iteraciones)*
x = fminsearch(fun,x0) **x = fminsearch(fun,x0,options)** **x = fminsearch(fun,x0,options,P1,P2,...)** **[x,fval] = fminsearch(...)** **[x,fval,f] = fminsearch(...)** **[x,fval,f,output] = fminsearch(...)**	*Comandos idénticos a los anteriores para minimizar funciones de varias variables para valores iniciales dados por x0. El valor xo puede ser un intervalo [a,b] en el que se busca la solución. Luego, para minimizar fun en [a,b] se usa x = fminsearch(fun,[a,b])*
x = fzero(fun,x0) **x = fzero(fun,x0,options)** **x = fzero(fun,x0,options,P1,P2,...)** **[x,fval] = fzero(...)** **[x,fval,exitflag] = fzero(...)** **[x,fval,exitflag,output] = fzero(...)**	*Comandos idénticos a los anteriores para encontrar ceros de funciones. El valor xo puede ser un intervalo [a,b] en el que se busca la solución. Luego, para hallar un cero de fun en [a, b] se puede usar x = fzero(fun,[a,b]), donde fun tiene signo contrario en a y b.*

options = optimset('p1',v1,'p2',v2,...)	*Crea opciones de optimización para los parámetros p1, p2, ... con valores v1, v2, ... Los parámetros posibles son* `Display` *(con valores posibles* `'off'` | `'iter'` | `'final'` | `'notify'` *para no ver la salida, ver la salida de cada iteración, ver sólo la salida final y obtener mensaje si no hay convergencia);* `MaxFunEvals`*, cuyo valor es un entero que indica el máximo número de evaluaciones;* `MaxIter` *cuyo valor es un entero que indica el máximo número de iteraciones;* `TolFun`*, cuyo valor es un entero que indica la tolerancia en el valor de la función, y* `TolX`*, cuyo valor es un entero que indica la tolerancia en el valor de x*
val = optimget(options,'param')	*Devuelve el valor del parámetro especificado en la estructura de optimización options*
g = inline(*expr*)	*Transforma en función la cadena expr*
g = inline(*expr,arg1,arg2,* ...)	*Transforma en función la cadena expr con argumentos de entrada dados*
g = inline(*expr,n*)	*Transforma en función la cadena expr con n argumentos de entrada*
f = @función	*Habilita la función para ser evaluada*

Como primer ejemplo minimizamos la función *Cos*(*x*) en el intervalo (3,4).

```
>> x = fminbnd(@cos,3,4)

x =
    3.1416
```

También podría haberse utilizado la siguiente sintaxis:

```
>> x = fminbnd(inline('cos(x)'),3,4)

x =
    3.1416
```

En el ejemplo siguiente realizamos la minimización anterior con una tolerancia de 8 decimales y hallamos tanto el valor de *x* que minimiza el coseno en el intervalo dado como el valor mínimo de la función coseno en ese intervalo presentando información relativa a todas las iteraciones del proceso.

```
>>   [x,fval,f]   =   fminbnd(@cos,3,4,optimset('TolX',1e-8,...
     'Display','iter'));


 Func-count      x          f(x)          Procedure
    1        3.38197    -0.971249        initial
    2        3.61803    -0.888633        golden
    3        3.23607    -0.995541        golden
    4        3.13571    -0.999983        parabolic
    5         3.1413           -1        parabolic
    6        3.14159           -1        parabolic
    7        3.14159           -1        parabolic
    8        3.14159           -1        parabolic
    9        3.14159           -1        parabolic

Optimization terminated successfully:
the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-008
```

En el ejemplo siguiente, tomando como valores iniciales (-1,2;1), minimizamos y hallamos el valor objetivo de la función de dos variables:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

```
>> [x,fval] = fminsearch(inline('100*(x(2)-x(1)^2)^2+...
   (1-x(1))^2'),[-1.2, 1])

x =

    1.0000    1.0000


fval =

  8.1777e-010
```

En el ejemplo siguiente se calcula un cero de la función seno cerca de 3 y un cero de la función coseno entre 1 y 2.

```
>> x = fzero(@sin,3)

x =

    3.1416
```

```
>> x = fzero(@cos,[1 2])

x =

    1.5708
```

Integración numérica

MATLAB contiene funciones que permiten realizar integración numérica mediante los métodos de Simpson y Lobato. A continuación se presenta la sintaxis de estas funciones:

q = quad(f,a,b)	*Hallar la integral de f entre a y b por el método de Simpson con un error de 10^{-6}*
q = quad(f,a,b,tol)	*Hallar la integral de f entre a y b por el método de Simpson con la toleracia tol en vez de con un error de 10^{-6}*
q = quad(f,a,b,tol,trace)	*Hallar la integral de f entre a y b por el método de Simpson con la tolerancia tol y presentando la traza de las iteraciones*
q = quad(f,a,b,tol,trace,p1,p2,...)	*Pasa los argumentos adicionales p1, p2, ... a la función f, f(x,p1,p2, ...)*
[q,fcnt] = quadl(f,a,b,...)	*Devuelve adicionalmente el número de evaluaciones de f*
q = quadl(f,a,b)	*Hallar la integral de f entre a y b por el método de la cuadratura de Lobato con un error de 10^{-6}*
q = quadl(f,a,b,tol)	*Hallar la integral de f entre a y b por el método de la cuadratura de Lobato con la tolerancia tol en vez de con un error de 10^{-6}*
q = quadl(f,a,b,tol,trace)	*Hallar la integral de f entre a y b por el método de la cuadratura de Lobato con la tolerancia tol y presentando la traza de las iteraciones*
q = quad(f,a,b,tol,trace,p1,p2,...)	*Pasa los argumentos adicionales p1, p2, ... a la función f, f(x,p1,p2, ...)*
[q,fcnt] = quadl(f,a,b,...)	*Devuelve adicionalmente el número de evaluaciones de f*
q = dblquad(f,xmin,xmax, ymin,ymax)	*Halla la integral doble de f(x,y) en el campo de variación de (x,y) especificado con un error de 10^{-6}*
q = dblquad(f,xmin,xmax, ymin,ymax,tol)	*Halla la integral doble de f(x,y) en el campo de variación de (x,y) especificado con la tolerancia tol*
q = dblquad(f,xmin,xmax, ymin,ymax,tol,@quadl)	*Halla la integral doble de f(x,y) en el campo de variación de (x,y) especificado con la tolerancia tol y el método quadl*
q = dblquad(f,xmin,xmax, ymin,ymax,tol,method,p1,p2,...)	*Pasa los argumentos adicionales p1, p2, ... a la función f*

Como primer ejemplo calculamos $\int_0^2 \frac{1}{x^3-2x-5}dx$ por el método de Simpson.

```
>> F = inline('1./(x.^3-2*x-5)');
>> Q = quad(F,0,2)

Q =

    -0.4605
```

A continuación se observa que se mantiene el valor de la integral aunque aumentemos la tolerancia a 10^{-18}.

```
>> Q = quad(F,0,2,1.0e-18)

Q =

    -0.4605
```

En el ejemplo siguiente se evalúa la integral mediante el método de Lobato.

```
>> Q = quadl(F,0,2)

Q =

    -0.4605
```

Después evaluamos la integral doble $\int_{\pi}^{2\pi}\int_{0}^{\pi}(ysen(x)+xcos(y))\,dy\,dx$.

```
>> Q = dblquad(inline('y*sin(x)+x*cos(y)'), pi, 2*pi, 0, pi)

Q =

    -9.8696
```

Derivación numérica

De forma muy simple puede definirse la derivada $f'(x)$ de una función $f(x)$ como la tasa de cambio de $f(x)$ con respecto a x. La derivada puede expresarse como un ratio entre el cambio en $f(x)$ indicado por $df(x)$ y el cambio en x indicado por dx. La derivada de una función f en el punto x_k puede estimarse mediante la expresión:

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Los valores x_k y x_{k-1} son lo suficientemente próximos. De forma similar puede estimarse la segunda derivada $f''(x)$ de la función $f(x)$ como la primera derivada de $f'(x)$, es decir:

$$f''(x_k) = \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}$$

MATLAB incluye en su módulo básico la función *diff*, que permite aproximar la derivada y cuya sintaxis es la siguiente:

Y = diff(X)	*Calcula las diferencias entre elementos adyacentes del vector X: [X(2)-X(1), X(3)-X(2), ..., X(n)-X(n-1)]. Si X es una matriz (m,n), diff(X) devuelve la matriz de diferencias por filas: [X(2:m,:)-X(1:m-1,:)]*
Y = diff(X,*n*)	*Hallar diferencias de orden n, por ejemplo: diff(X,2)=diff(diff(X))*

Como ejemplo consideramos la función $f(x)= x^5-3x4-11x^3+27x^2+10x-24$ y representamos su función derivada en el intervalo [-4,5]. Véase la Figura 8-27.

```
>> x=-4:0.1:5;
>> f=x.^5-3*x.^4-11*x.^3+27*x.^2+10*x-24;
>> df=diff(f)./diff(x);
>> plot(x,f)
```

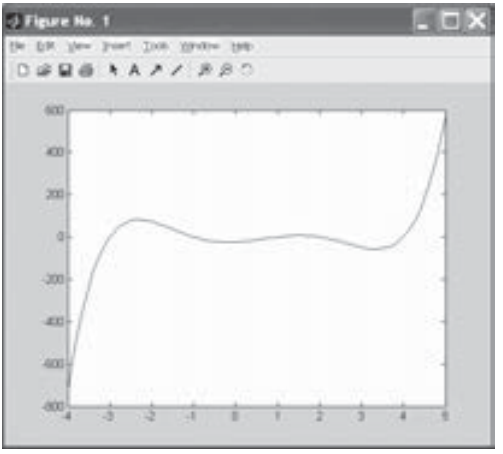


Figura 8-27

Solución aproximada de ecuaciones diferenciales

MATLAB dispone de comandos en su módulo básico que permiten resolver numéricamente ecuaciones diferenciales ordinarias (ODEs), ecuaciones diferenciales algebraicas (DAEs) y resolución de problemas con valores en la frontera ODEs. También es posible la resolución de sistemas de ecuaciones diferenciales con valores en la frontera y ecuaciones diferenciales en derivadas parciales parabólicas y elípticas.

Ecuaciones diferenciales ordinarias con valores iniciales

Una ecuación diferencial ordinaria contiene una o más derivadas de la variable dependiente y respecto a la variable independiente *t*. Una ecuación diferencial ordinaria de primer orden con valores iniciales para la variable independiente puede representarse como:

$$y' = f(t, y)$$
$$y(t_0) = y_0$$

Se puede generalizar el problema anterior para el caso de que *y* sea un vector $y = (y_1, y_2, \ldots, y_n)$.

Los comandos del módulo básico de MATLAB relativos a ecuaciones diferenciales ordinarias y algebraicas con valores iniciales se presentan en la tabla siguiente:

Comando	*Clase de problema que resuelve, método numérico y sintaxis*
ode45	*Ecuaciones diferenciales comunes por el método de Runge-Kutta*
ode23	*Ecuaciones diferenciales comunes por el método de Runge-Kutta*
ode113	*Ecuaciones diferenciales comunes por el método de Adams*
ode15s	*Ecuaciones diferenciales ordinarias y algebraicas mediante NDFs (BDFs)*
ode23s	*Ecuaciones diferenciales ordinarias por el método de Rosenbrock*
ode23t	*Ecuaciones diferenciales ordinarias y algebraicas por la regla trapezoidal*
ode23tb	*Ecuaciones diferenciales ordinarias mediante TR-BDF2*

La sintaxis común para los 7 comandos anteriores es la siguiente:

```
[T,Y] = solver(odefun,tspan,y0)
[T,Y] = solver(odefun,tspan,y0,opciones)
[T,Y] = solver(odefun,tspan,y0,opciones,p1,p2...)
[T,Y,TE,YE,IE] = solver(odefun,tspan,y0,opciones)
```

En la sintaxis anterior *solver* puede ser cualquiera de los comandos *ode45, ode23, ode113, ode15s, ode23s, ode23t* u *ode23tb*.

El argumento *odefun* evalúa el lado derecho de la ecuación diferencial o sistema escrita en la forma $y'=f(t,y)$ o $M(t,y)y'=f(t,y)$, donde $M(t,y)$ se denomina matriz de masa. El comando *ode23s* sólo puede resolver ecuaciones con matriz de masa constante. Los comandos *ode15s* y *ode23t* pueden resolver ecuaciones con matriz de masas singular y ecuaciones diferenciales algebraicas. El argumento *tspan* es un vector que especifica el intervalo $[t_0, t_f]$ de integración (para obtener soluciones en valores específicos de *t,* todos crecientes o decrecientes, se usa *tspan* = $[t_0, t_1, \ldots, t_f]$). El argumento y_0 especifica un vector de condiciones iniciales. Los argumentos $p1, p2, \ldots$ son parámetros opcionales que se pasan a *odefun*. El argumento *opciones* especifica opciones adicionales de integración mediante el comando *odeset* que pueden encontrarse en el manual del programa. Los vectores *T* e *Y* presentan los valores numéricos de las variables independiente y dependiente encontrados para las soluciones.

Como primer ejemplo hallamos las soluciones en el intervalo [0,12] del sistema de ecuaciones diferenciales ordinarias siguiente:

$$\begin{aligned} y'_1 &= y_2 y_3 & y_1(0) &= 0 \\ y'_2 &= -y_1 y_3 & y_2(0) &= 1 \\ y'_3 &= -0.51 y_1 y_2 & y_3(0) &= 1 \end{aligned}$$

Para ello definimos una función de nombre *sistema1* en un M-fichero, con la finalidad de almacenar en ella las ecuaciones del sistema. La forma de definir esta función es partir de un vector columna con tres filas vacío, para asignarle posteriormente tres componentes que constituyen la sintaxis de las tres ecuaciones (Figura 8-28).

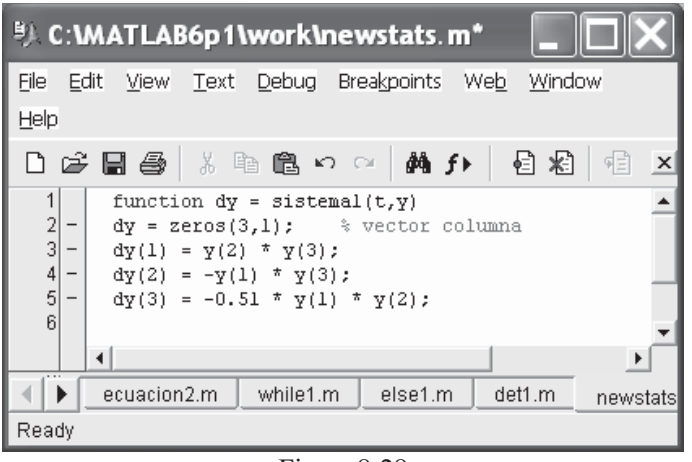


Figura 8-28

A continuación resolvemos el sistema tecleando en la ventana de comandos lo siguiente:

```
>> [T,Y] = ode45(@sistema1,[0 12],[0 1 1])

T =

         0
    0.0001
    0.0001
    0.0002
    0.0002
    0.0005
     .
     .
   11.6136
   11.7424
   11.8712
   12.0000

Y =
         0    1.0000    1.0000
    0.0001    1.0000    1.0000
    0.0001    1.0000    1.0000
    0.0002    1.0000    1.0000
    0.0002    1.0000    1.0000
    0.0005    1.0000    1.0000
    0.0007    1.0000    1.0000
    0.0010    1.0000    1.0000
    0.0012    1.0000    1.0000
    0.0025    1.0000    1.0000
    0.0037    1.0000    1.0000
    0.0050    1.0000    1.0000
    0.0062    1.0000    1.0000
    0.0125    0.9999    1.0000
    0.0188    0.9998    0.9999
    0.0251    0.9997    0.9998
    0.0313    0.9995    0.9997
    0.0627    0.9980    0.9990
     .
     .
    0.8594   -0.5105    0.7894
    0.7257   -0.6876    0.8552
    0.5228   -0.8524    0.9281
    0.2695   -0.9631    0.9815
   -0.0118   -0.9990    0.9992
   -0.2936   -0.9540    0.9763
   -0.4098   -0.9102    0.9548
   -0.5169   -0.8539    0.9279
   -0.6135   -0.7874    0.8974
   -0.6987   -0.7128    0.8650
```

Para interpretar mejor los resultados, la solución numérica anterior puede graficarse (Figura 8-29) mediante la siguiente sintaxis:

```
>> plot(T,Y(:,1),'-',T,Y(:,2),'-.',T,Y(:,3),'.')
```

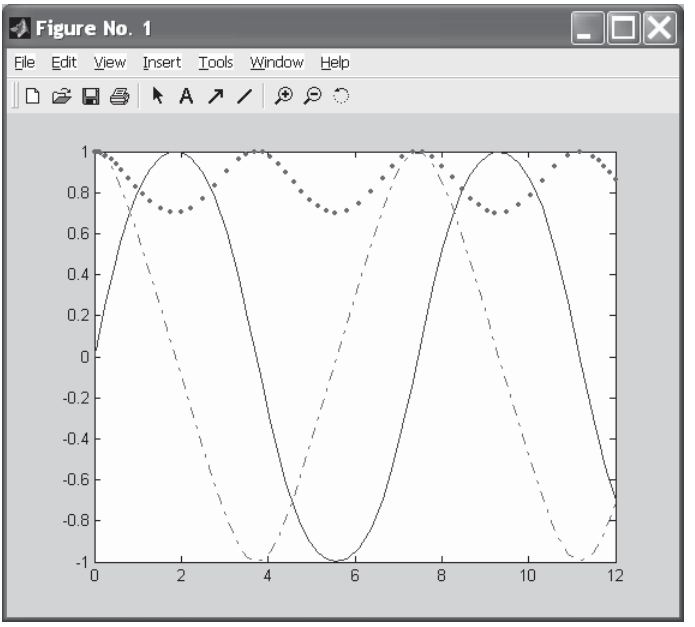


Figura 8-29

Ecuaciones diferenciales ordinarias con valores en la frontera

MATLAB también permite resolver ecuaciones diferenciales ordinarias con valores en la frontera. Los valores en la frontera no son más que condiciones que especifican una relación entre los valores de la solución en los puntos inicial y final de la variable independiente. El problema más sencillo de este tipo es el sistema de ecuaciones

$$y' = f(x, y)$$

donde x es la variable independiente, y es la variable dependiente e y' es la derivada de y respecto a x ($y'=dy/dx$). Además la solución en el intervalo $[a, b]$ ha de satisfacer:

$$g(y(a), y(b)) = 0$$

Una forma más general de este tipo de ecuaciones diferenciales puede expresarse como sigue:

$$y' = f(x, y, p)$$
$$g(y(a), y(b), p) = 0$$

donde el vector p está formado por parámetros que han de ser determinados simultáneamente con la solución a través de las condiciones de valores en la frontera.

El comando que resuelve estos problemas es *bvp4c*, cuya sintaxis es la siguiente:

```
sol = bvp4c(odefun,bcfun,solinit)
sol = bvp4c(odefun,bcfun,solinit,options)
sol = bvp4c(odefun,bcfun,solinit,options,p1,p2...)
```

En la sintaxis anterior *odefun* es una función que evalúa las ecuaciones diferenciales *f(x,y)* y que puede tener la forma siguiente:

```
dydx = odefun(x,y)
dydx = odefun(x,y,p1,p2,...)
dydx = odefun(x,y,parametros)
dydx = odefun(x,y,parametros,p1,p2,...)
```

En la sintaxis de *bvp4c* el argumento *bcfun* es una función que computa el residuo en las condiciones en la frontera. Su forma es la siguiente:

```
res = bcfun(ya,yb)
res = bcfun(ya,yb,p1,p2,...)
res = bcfun(ya,yb,parametros)
res = bcfun(ya,yb,parametros,p1,p2,...)
```

El argumento *solinit* es una estructura con los campos *x* (nodos ordenados de la malla inicial de modo que las condiciones en la frontera son impuestas mediante a = `solinit.x(1)` y b = `solinit.x(end)` e *y* (suposiciones iniciales para la solución de modo que a = `solinit.x(1)` y b = `solinit.x(end)` es una suposición para la solución en el nodo `solinit.x(i)`. El comando *vpinit* fija *solinit* con la sintaxis *solinit = vpinit(x,y)*.

Como ejemplo resolvemos la ecuación diferencial de segundo orden:

$$y'' + |y| = 0$$

cuyas soluciones han de satisfacer las condiciones en la frontera:

$$y(0) = 0$$
$$y(4) = -2$$

El problema anterior es equivalente al siguiente:

$$y_1' = y_2$$
$$y_2' = -|y_1|$$

Consideramos como suposición inicial una malla de 5 puntos igualmente espaciados en [0,4] y valores constantes $y_1(x)=0$ e $y_2(x)=0$. La sintaxis siguiente recoge estas suposiciones:

```
>> solinit = bvpinit(linspace(0,4,5),[1 0]);
```

Codificaremos en MATLAB la función y las condiciones en la frontera mediante los M-ficheros de las Figuras 8-30 y 8-31.

Figura 8-30

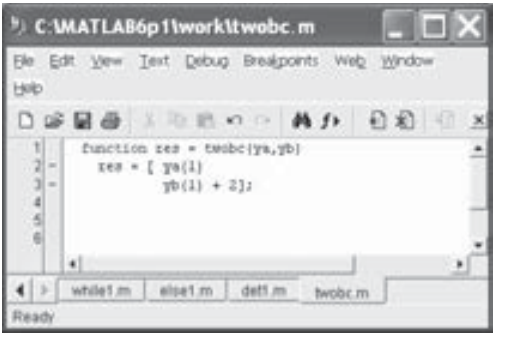


Figura 8-31

La solución de la ecuación la ofrece la sintaxis siguiente:

```
>> sol = bvp4c(@twoode,@twobc,solinit);
```

que puede graficarse (Figura 8-32) usando el comando *bvpval* como sigue:

```
>>y = bvpval(sol,linspace(0,4));
>>plot(x,y(1,:));
```

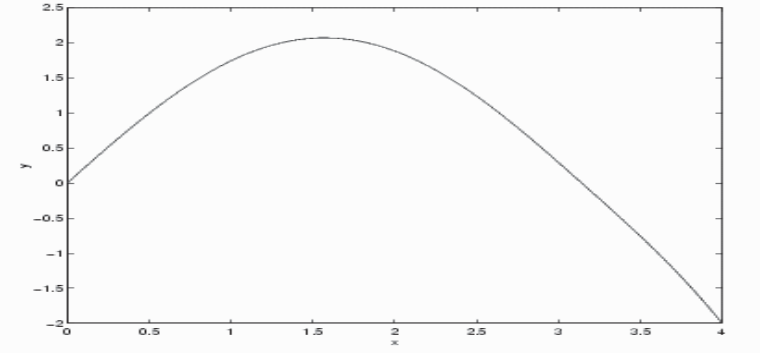


Figura 8-32

Ecuaciones diferenciales en derivadas parciales

El módulo básico de MATLAB dispone de funciones que permiten resolver ecuaciones y sistemas de ecuaciones diferenciales en derivadas parciales con valores iniciales en la frontera. La función básica para el cálculo de las soluciones es *pedepe*, y la función básica para evaluar dichas soluciones es *pdeval*.

La sintaxis de la función *pedepe* es la siguiente:

```
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan)
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan,options)
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan,options,p1,p2...)
```

El parámetro *m* vale 0, 1 o 2 según la naturaleza de la simetría (bloque, cilíndrica o esférica, respectivamente). El argumento *pdefun* es la función que define las componentes de la ecuación diferencial, *icfun* es la función que define las condiciones iniciales, *bcfun* es la función que define las condiciones frontera, *xmesh* y *tspan* son los vectores $[x_0, x_1,\ldots,x_n]$ y $[t_0, t_1,\ldots,t_f]$ que especifican los puntos en los cuales se requiere la solución ($n,f\geq 3$), *options* especifica las opciones de cálculo de las soluciones (`RelTol`, `AbsTol`, `NormControl`, `InitialStep`, y `MaxStep` para especificar tolerancia relativa, tolerancia absoluta, tolerancia en norma, paso inicial y paso máximo, respectivamente) y $p1$, $p2$,… son parámetros a pasar a las funciones *pdefun, icfun* y *bcfun*.

La forma de la ecuación diferencial en derivadas parciales es:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

donde $a\leq x\leq b$ y $t_0\leq t\leq t_f$. Además, para $t = t_0$ y para todo x las componentes solución satisfacen las condiciones iniciales:

$$u(x,t_0) = u_0(x)$$

y para todo t y cada $x=a$ o $x=b$, las componentes solución satisfacen ecuaciones frontera de la forma:

$$p(x,t,u) + q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right) = 0$$

Además, se tiene que `xmesh(1)=a, xmesh(end)=b, tspan(1)=` t_0 y `tspan(end)=` t_f. Por otra parte *pdfun* halla los términos *c*, *f* y *s* de la ecuación diferencial en derivadas parciales, de modo que:

```
[c,f,s] = pdefun(x,t,u,dudx)
```

De modo similiar *icfun* evalúa las condiciones iniciales mediante

```
u = icfun(x)
```

Por último, *bcfun* evalúa los valores *b* y *c* de las condiciones en la frontera mediante:

```
[pl,ql,pr,qr] = bcfun(xl,ul,xr,ur,t)
```

Como primer ejemplo resolveremos la ecuación en derivadas parciales ($x \in [0,1]$ y $t \geq 0$) siguiente:

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right)$$

satisfaciendo la condición inicial :

$$u(x, 0) = \sin \pi x$$

y las condiciones en la frontera :

$$u(0, t) \equiv 0$$
$$\pi e^{-t} + \frac{\partial u}{\partial x}(1, t) = 0$$

Comenzamos definiendo las funciones en M-ficheros de las figura 8-33 a 8-35.

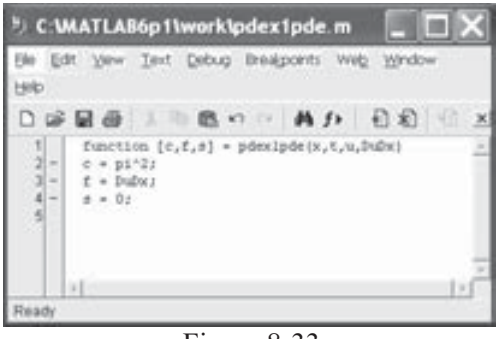


Figura 8-33

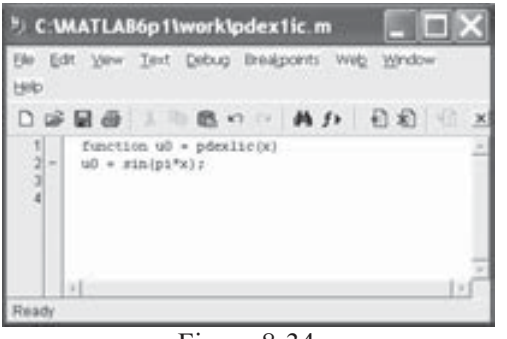


Figura 8-34

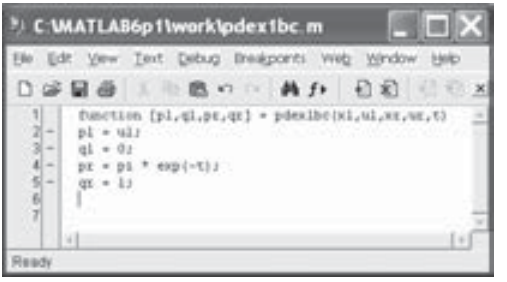


Figura 8-35

Una vez definidas las funciones de apoyo, se define la función que resuelve el problema mediante el M-fichero de la Figura 8-36.

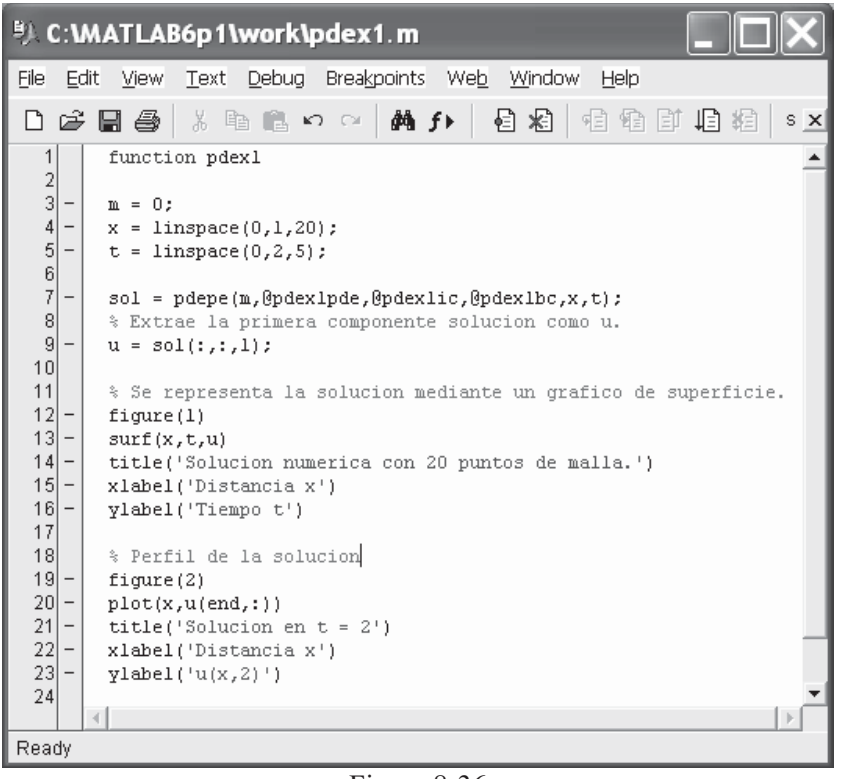


Figura 8-36

Para ejecutar la solución (Figuras 8-37 y 8-38), en la ventana de comandos de MATLAB se utiliza la sintaxis:

```
>> pdex1
```

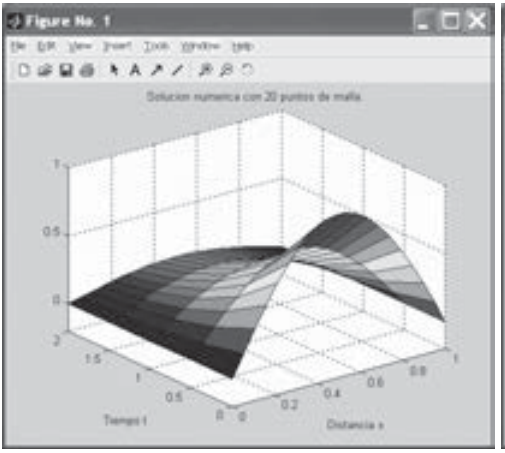


Figura 8-37

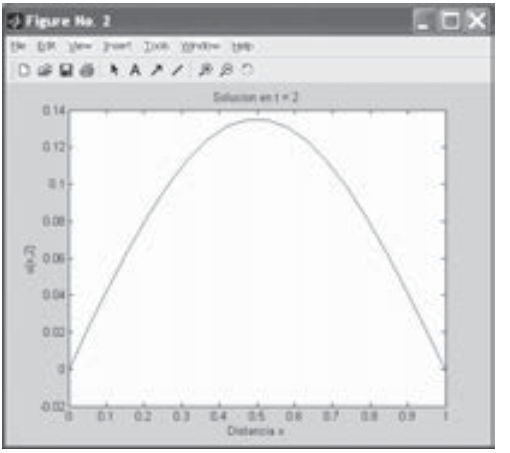


Figura 8-38

Como segundo ejemplo resolvemos el sistema de ecuaciones diferenciales en derivadas parciales ($x \in [0,1]$ y $t \geq 0$) siguiente:

$$\frac{\partial u_1}{\partial t} = 0.024 \frac{\partial^2 u_1}{\partial x^2} - F(u_1 - u_2)$$

$$\frac{\partial u_2}{\partial t} = 0.170 \frac{\partial^2 u_2}{\partial x^2} + F(u_1 - u_2)$$

$$F(y) = \exp(5.73y) - \exp(-11.46y)$$

satisfaciendo las condiciones iniciales:

$$u_1(x, 0) \equiv 1$$
$$u_2(x, 0) \equiv 0$$

y las condiciones en la frontera:

$$\frac{\partial u_1}{\partial x}(0, t) \equiv 0$$
$$u_2(0, t) \equiv 0$$
$$u_1(1, t) \equiv 1$$
$$\frac{\partial u_2}{\partial x}(1, t) \equiv 0$$

Para utilizar convenientemente la función *pdepe*, el sistema puede escribirse como:

$$\begin{bmatrix}1\\1\end{bmatrix} .* \frac{\partial}{\partial t}\begin{bmatrix}u_1\\u_2\end{bmatrix} = \frac{\partial}{\partial x}\begin{bmatrix}0.024(\partial u_1/\partial x)\\0.170(\partial u_2/\partial x)\end{bmatrix} + \begin{bmatrix}-F(u_1-u_2)\\F(u_1-u_2)\end{bmatrix}$$

La condición izquierda en la frontera puede escribirse como:

$$\begin{bmatrix}0\\u_2\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix} .* \begin{bmatrix}0.024(\partial u_1/\partial x)\\0.170(\partial u_2/\partial x)\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

y la condición derecha en la frontera puede escribirse como:

$$\begin{bmatrix}u_1-1\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} .* \begin{bmatrix}0.024(\partial u_1/\partial x)\\0.170(\partial u_2/\partial x)\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

Comenzamos definiendo las funciones en M-ficheros de las Figuras 8-39 a 8-41.

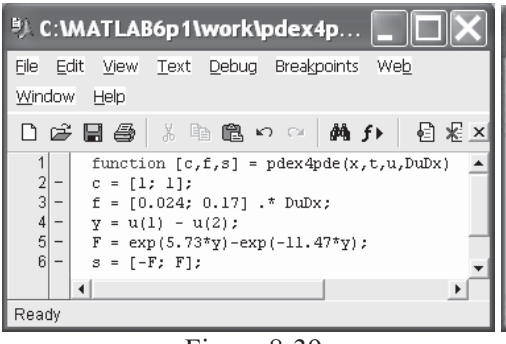


Figura 8-39

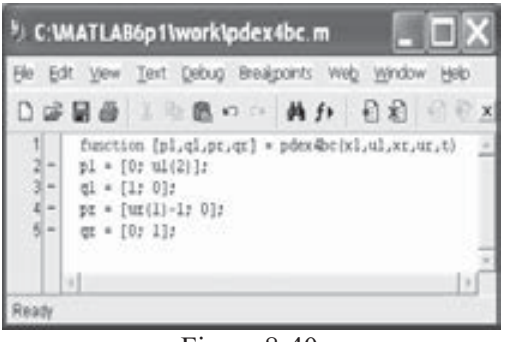


Figura 8-40

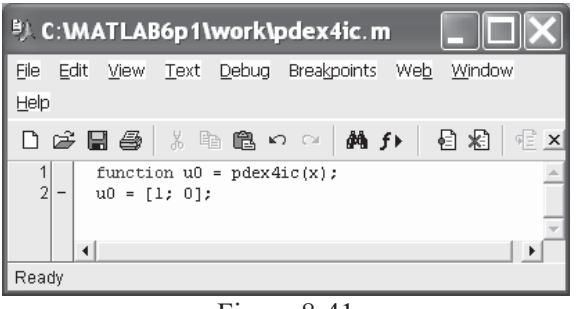


Figura 8-41

Una vez definidas las funciones de apoyo, se define la función que resuelve el problema mediante el M-fichero de la Figura 8-42.

C:\MATLAB6p1\work\pdex4.m

File Edit View Text Debug Breakpoints Web Window Help

```
function pdex4
m = 0;
x = [0 0.005 0.01 0.05 0.1 0.2 0.5 0.7 0.9 0.95 0.99 0.995 1];
t = [0 0.005 0.01 0.05 0.1 0.5 1 1.5 2];

sol = pdepe(m,@pdex4pde,@pdex4ic,@pdex4bc,x,t);
u1 = sol(:,:,1);
u2 = sol(:,:,2);

figure
surf(x,t,u1)
title('u1(x,t)')
xlabel('Distance x')
ylabel('Time t')

figure
surf(x,t,u2)
title('u2(x,t)')
xlabel('Distancia x')
ylabel('Tiempo t')
```

Ready

Figura 8-42

Para ejecutar la solución (Figuras 8-43 y 8-44), en la ventana de comandos de MATLAB se utiliza la sintaxis:

```
>> pdex4
```

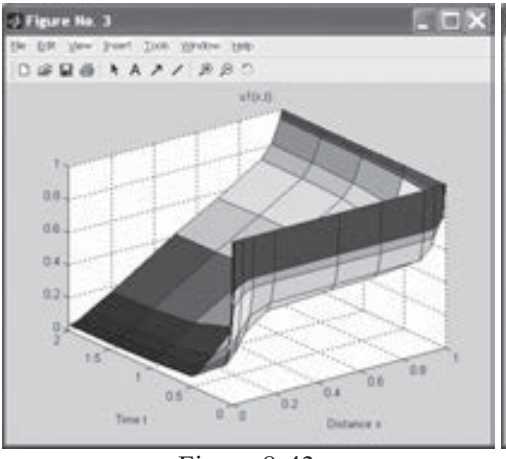


Figura 8-43

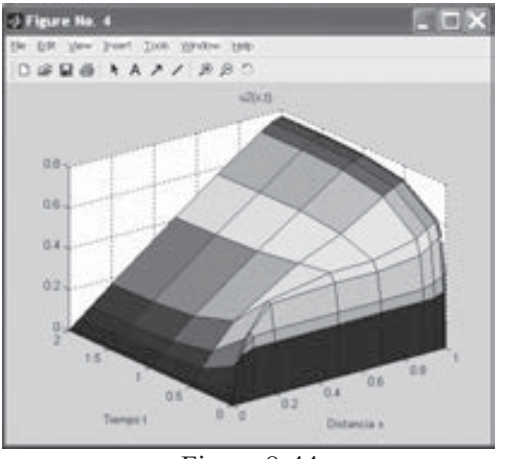


Figura 8-44

Ejercicio 8-1. Minimizar la función x^3-2x-5 en el intervalo (0,2) y calcular el valor mínimo que toma la función es ese intervalo presentando información sobre todas las iteraciones del proceso de optimización.

```
>> f = inline('x.^3-2*x-5');
>> [x,fval] = fminbnd(f, 0, 2,optimset('Display','iter'))

 Func-count     x          f(x)          Procedure
    1       0.763932     -6.08204        initial
    2        1.23607     -5.58359        golden
    3       0.472136     -5.83903        golden
    4       0.786475     -6.08648        parabolic
    5       0.823917     -6.08853        parabolic
    6         0.8167     -6.08866        parabolic
    7        0.81645     -6.08866        parabolic
    8       0.816497     -6.08866        parabolic
    9        0.81653     -6.08866        parabolic

Optimization terminated successfully:
the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004


x =

    0.8165


fval =

   -6.0887
```

Ejercicio 8-2. Hallar en un entorno de x=1,3 un cero de la función:

$$f(x)=\frac{1}{(x-0{,}3)^2+0{,}01}+\frac{1}{(x-0{,}9)^2+0{,}04}-6$$

Minimizar también dicha función en el intervalo (0,2).

En primer lugar hallamos un cero de la función dada en un entorno de x=1,3 presentando información sobre las iteraciones realizadas y comprobando que se trata efectivamente de un cero.

```
>> [x,feval]=fzero(inline('1/((x-0.3)^2+0.01)+...
   1/((x-0.9)^2+0.04)-6'),1.3,optimset('Display','iter'))
```

```
 Func-count      x          f(x)          Procedure
    1             1.3   -0.00990099        initial
    2          1.26323      0.882416        search

   Looking for a zero in the interval [1.2632, 1.3]

    3          1.29959   -0.00093168        interpolation
    4          1.29955  1.23235e-007        interpolation
    5          1.29955 -1.37597e-011        interpolation
    6          1.29955             0        interpolation
Zero found in the interval: [1.2632, 1.3].

x =

    1.2995


feval =

     0
```

En segundo lugar minimizamos la función especificada en el intervalo [0,2] y presentamos también información sobre todo el proceso iterativo hasta encontrar el valor de *x* que hace mínima la función. Asimismo, se calcula el valor objetivo de la función minimizada en el intervalo dado.

```
>> [x,feval]=fminbnd(inline('1/((x-0.3)^2+0.01)+...
   1/((x-0.9)^2+0.04)-6'),0,2,optimset('Display','iter'))

 Func-count     x          f(x)          Procedure
    1       0.763932      15.5296        initial
    2        1.23607      1.66682        golden
    3        1.52786     -3.03807        golden
    4         1.8472     -4.51698        parabolic
    5        1.81067     -4.41339        parabolic
    6        1.90557     -4.66225        golden
    7        1.94164     -4.74143        golden
    8        1.96393     -4.78683        golden
    9        1.97771     -4.81365        golden
   10        1.98622     -4.82978        golden
   11        1.99148     -4.83958        golden
   12        1.99474     -4.84557        golden
   13        1.99675     -4.84925        golden
   14        1.99799     -4.85152        golden
   15        1.99876     -4.85292        golden
   16        1.99923     -4.85378        golden
   17        1.99953     -4.85431        golden
```

```
   18        1.99971     -4.85464        golden
   19        1.99982     -4.85484        golden
   20        1.99989     -4.85497        golden
   21        1.99993     -4.85505        golden
   22        1.99996     -4.85511        golden

Optimization terminated successfully:
 the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004


x =

    2.0000


feval =

   -4.8551
```

Ejercicio 8-3. El teorema del valor intermedio dice que si f es una función continua en el intervalo [a,b] y L es un número entre f(a) y f(b), entonces existe un valor c intermedio entre a y b (a<c<b) tal que f(c)=L. Dada la función f(x) = Cos(x-1), se trata de calcular el valor de c para el intervalo [1;2,5] correspondiente al valor L=0,8.

Se trata de resolver la ecuación $Cos(x\text{-}1)\text{-}8 = 0$ en el intervalo [1;2,5].

```
>> c = fzero(inline('cos(x-1)-0.8'),[1 2.5])

c =

    1.6435
```

Ejercicio 8-4. Calcular mediante los métodos aproximados de Simpson y Lobato la integral siguiente:

$$\int_{1}^{6} \left(2 + sen(2\sqrt{x})\right) dx$$

Para la solución mediante el método de Simpson tenemos:

```
>> quad(inline('2+sin(2*sqrt(x))'),1,6)

ans =

    8.1835
```

Para la solución mediante el método de Lobato tenemos:

```
>> quadl(inline('2+sin(2*sqrt(x))'),1,6)

ans =

    8.1835
```

Ejercicio 8-5. Calcular el área comprendida bajo la curva normal (0,1) entre los límites -1,96 y 1,96.

Se trata de calcular el valor de la integral $\int_{-1,96}^{1,96} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} dx$

El cálculo se realiza mediante MATLAB utilizando el método aproximado de Lobato como sigue:

```
>> quadl(inline('exp(-x.^2/2)/sqrt(2*pi)'), -1.96,1.96)

ans =

    0.9500
```

Ejercicio 8-6. Calcular el volumen del hemisferio definido en [-1,1] x [-1,1] por la función $f(x,y) = \sqrt{1-(x^2+y^2)}$

```
>> dblquad(inline('sqrt(max(1-(x.^2+y.^2),0))'),-1,1,-1,1)

ans =

    2.0944
```

El cálculo también podría haberse hecho de la siguiente forma:

```
>> dblquad(inline('sqrt(1-(x.^2+y.^2)).*(x.^2+y.^2<=1)'),-1,1,-1,1)

ans =

    2.0944
```

Ejercicio 8-7. Evaluar la integral doble siguiente:

$$\int_3^4 \int_1^2 \frac{1}{(x+y)^2} dxdy$$

```
>> dblquad(inline('1./(x+y).^2'),3,4,1,2)

ans =

    0.0408
```

Ejercicio 8-8. Resolver el sistema de ecuaciones diferenciales de Van der Pol siguiente:

$$y_1' = y_2 \qquad y_1(0) = 0$$
$$y_2' = 1000(1 - y_1^2)y_2 - y_1 \qquad y_2(0) = 1$$

Comenzamos definiendo una función de nombre *vdp100* en un M-fichero, con la finalidad de almacenar en ella las ecuaciones del sistema. La forma de definir esta función es partir de un vector columna con dos filas vacío, para asignarle posteriormente tres componentes que constituyen la sintaxis de las tres ecuaciones (Figura 8-45).

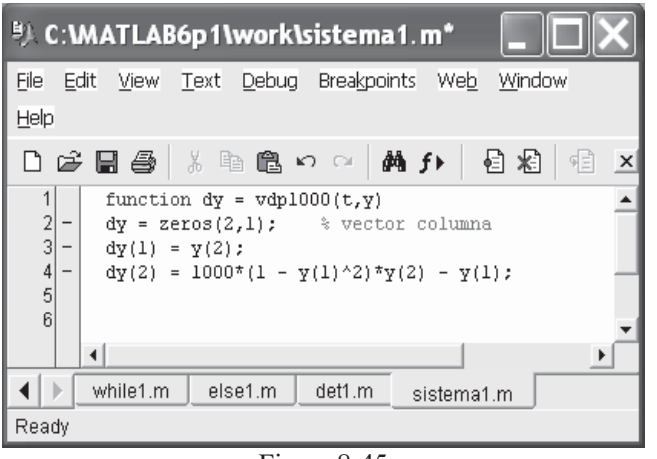


Figura 8-45

A continuación resolvemos el sistema y graficamos la solución $y_1 = y_1(t)$ relativa a la primera columna (Figura 8-46) de la salida tecleando lo siguiente en la ventana de comandos:

```
>> [T,Y] = ode15s(@vdp1000,[0 3000],[2 0]);
>> plot(T,Y(:,1),'-')
```

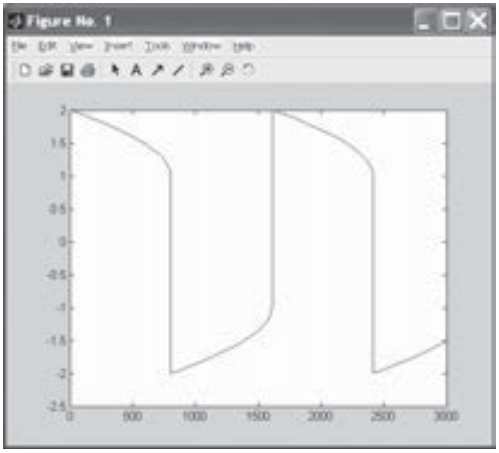


Figura 8-46

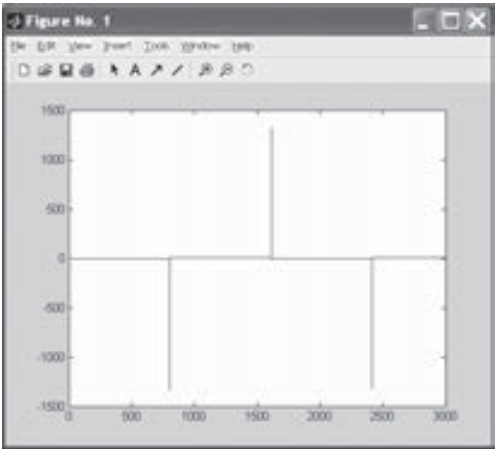


Figura 8-47

De forma similar graficamos la solución $y_2 = y_2(t)$ (Figura 8-47) mediante la sintaxis:

```
>> plot(T,Y(:,2),'-')
```

Ejercicio 8-9. Dada la ecuación diferencial siguiente:

$$y''+(\lambda-2q\cos(2x))y=0$$

sujeta a las condiciones frontera y(0)=1, y'(0)=0, y'(π)=0, encontrar una solución para q=5 y λ=15 basándose en una solución inicial para 10 puntos de x igualmente espaciados en el intervalo [0, π] y graficar su primera componente en 100 puntos igualmente espaciados en el intervalo [0, π].

La ecuación dada es equivalente al sistema de ecuaciones diferenciales de primer orden siguiente:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -(\lambda - 2\,q\cos 2x)y_1 \end{aligned}$$

con las siguientes condiciones en la frontera:

$$\begin{aligned} y_1(0)-1 &= 0 \\ y_2(0) &= 0 \\ y_2(\pi) &= 0 \end{aligned}$$

Para representar el sistema usamos la función del M-ficheros de la Figura 8-48, para representar las condiciones en la frontera usamos la función del M-fichero de la Figura 8-49, y mediante la función del M-fichero de la Figura 8-50 realizamos suposiciones para la solución inicial.

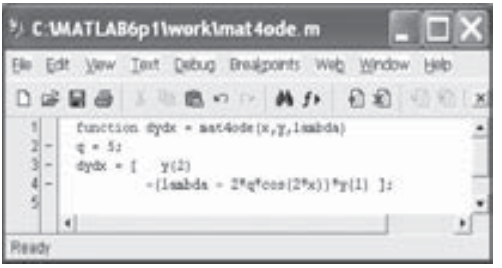


Figura 8-48

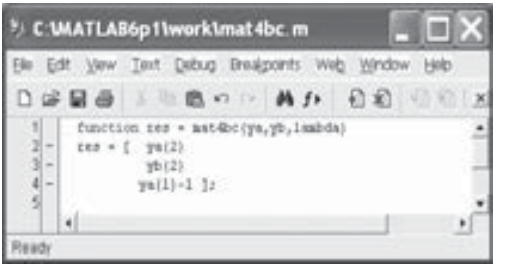


Figura 8-49

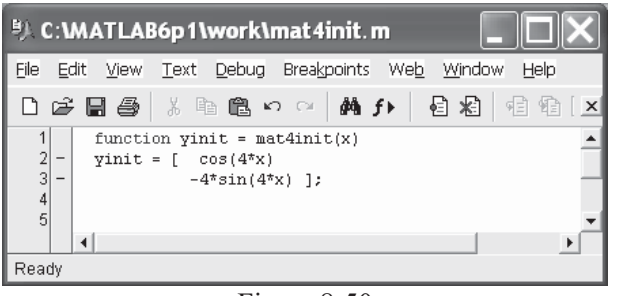


Figura 8-50

La solución inicial para λ=15 y 10 puntos igualmente espaciados en [0, π] se calcula mediante la sintaxis MATLAB siguiente:

```
>> lambda = 15;
solinit = bvpinit(linspace(0,pi,10),@mat4init,lambda);
```

La solución numérica del sistema se calcula mediante la sintaxis siguiente:

```
>> sol = bvp4c(@mat4ode,@mat4bc,solinit);
```

Para graficar su primera componente en 100 puntos igualmente espaciados en el intervalo [0, π] se utiliza la siguiente sintaxis:

```
>> xint = linspace(0,pi);
Sxint = bvpval(sol,xint);
plot(xint,Sxint(1,:))
axis([0 pi -1 1.1])
xlabel('x')
ylabel('solucion y')
```

El resultado se presenta en la Figura 8-51.

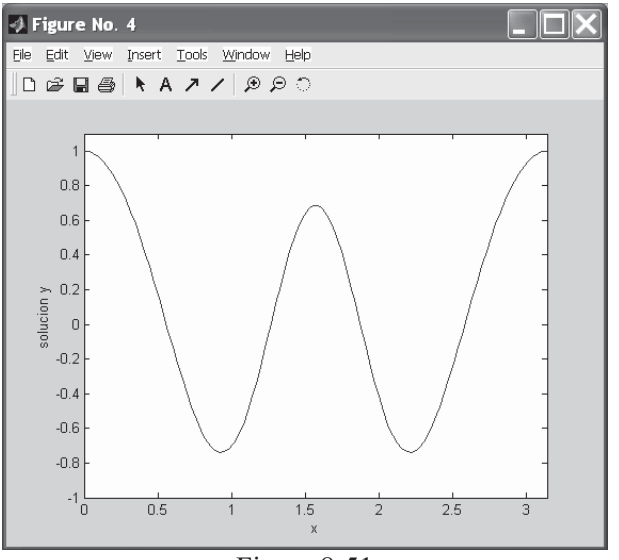


Figura 8-51

Ejercicio 8-10. Resolver la ecuación diferencial siguiente:

$$y''+(1-y^2)y'+y=0$$

para valores iniciales y=2, y'=0 en [0,20] y generalizar la solución para la ecuación:

$$y''+\mu(1-y^2)y'+y=0 \quad \mu>0$$

La ecuación general anterior es equivalente al sistema de ecuaciones lineales de primer orden siguiente:

$$y'_1 = y_2$$
$$y'_2 = \mu(1-y_1^{\,2})y_2-y_1$$

cuyas ecuaciones pueden definirse para μ=1 mediante el M-fichero de la Figura 8-52.

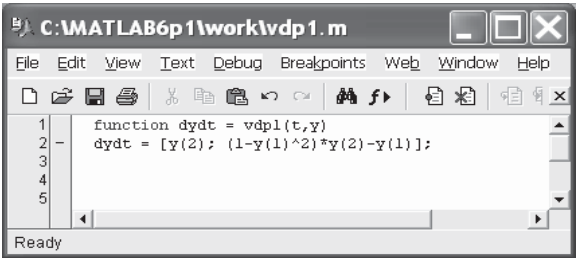


Figura 8-52

Si consideramos valores iniciales y_1=2 e y_2=0 en el intervalo [0,20], se puede resolver el sistema mediante la sintaxis MATLAB siguiente:

```
>> [t,y] = ode45(@vdp1,[0 20],[2; 0])

t =

         0
    0.0000
    0.0001
    0.0001
    0.0001
    0.0002
    0.0004
    0.0005
    0.0006
    0.0012
      .
      .
   19.9559
   19.9780
   20.0000

y =

    2.0000         0
    2.0000   -0.0001
    2.0000   -0.0001
    2.0000   -0.0002
    2.0000   -0.0002
    2.0000   -0.0005
      .
      .
    1.8729    1.0366
    1.9358    0.7357
    1.9787    0.4746
    2.0046    0.2562
    2.0096    0.1969
    2.0133    0.1413
    2.0158    0.0892
    2.0172    0.0404
```

Podemos graficar las soluciones (Figura 8-53) mediante la sintaxis:

```
>> plot(t,y(:,1),'-',t,y(:,2),'--')
>> xlabel('tiempo t')
>> ylabel('solucion y')
>> legend('y_1','y_2')
```

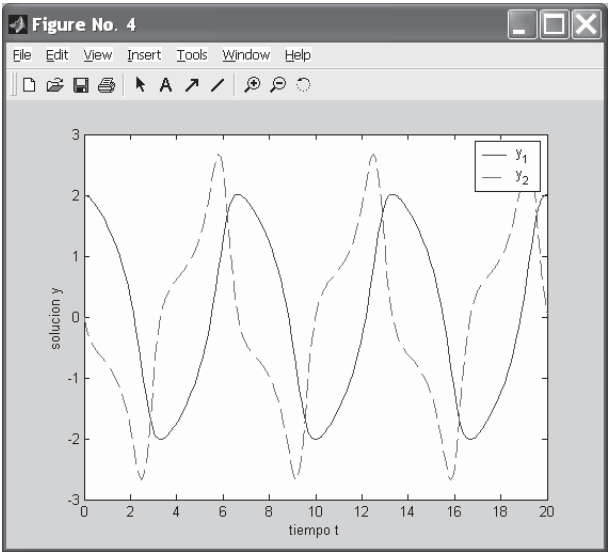


Figura 8-53

Para resolver el sistema general con el parámetro μ se define el sistema mediante el M-fichero de la Figura 8-54.

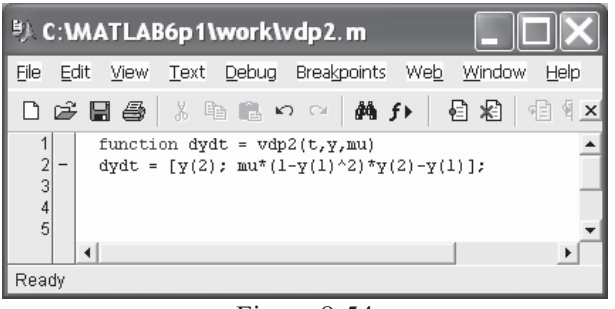


Figura 8-54

Ahora podemos graficar la primera solución (Figura 8-55) correspondiente a μ=1000 en el intervalo [0,1500] para y_1=2 e y_2=0 mediante la sintaxis siguiente:

```
>> [t,y] = ode15s(@vdp2,[0 1500],[2; 0],[],1000);
>> xlabel('tiempo t')
>> ylabel('solucion y_1')
```

Si queremos graficar la primera solución para otro valor del parámetro, por ejemplo μ=100, en el intervalo [0,1500] para y_1=2 e y_2=0 (Figura 8-56), utilizaremos la sintaxis:

```
>> [t,y] = ode15s(@vdp2,[0 1500],[2; 0],[],100);
>> plot(t,y(:,1),'-');
```

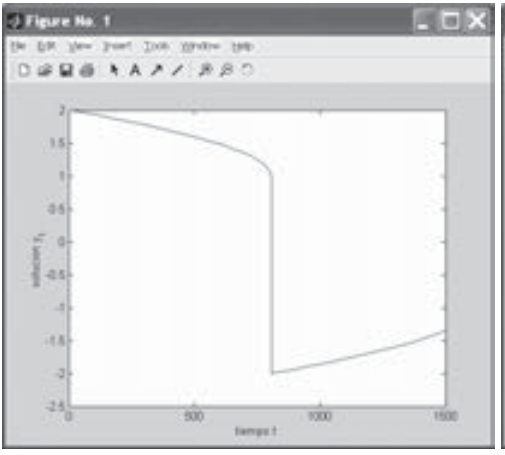


Figura 8-55

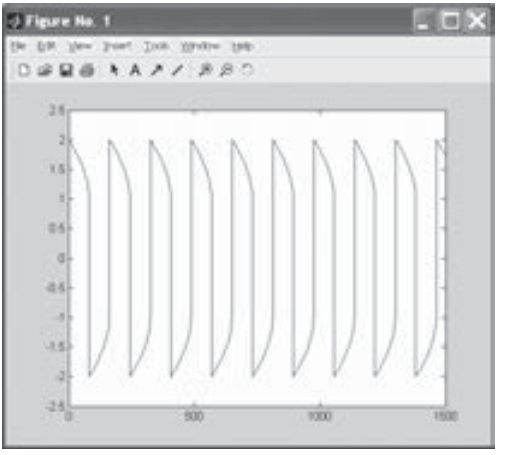


Figura 8-56

Ejercicio 8-11. La sucesión {an} de Fibonacci se define mediante la ley de recurrencia dada por $a_1 = 1$, $a_2 = 1$, $a_n = a_{n-1} + a_{n-2}$. Representar esta sucesión por una función recurrente y calcular a_2 , a_5 y a_{20}.

Para representar la sucesión de Fibonacci mediante una función recurrente definiremos la función mediante el *M-fichero fibo.m* de la Figura 8-57.

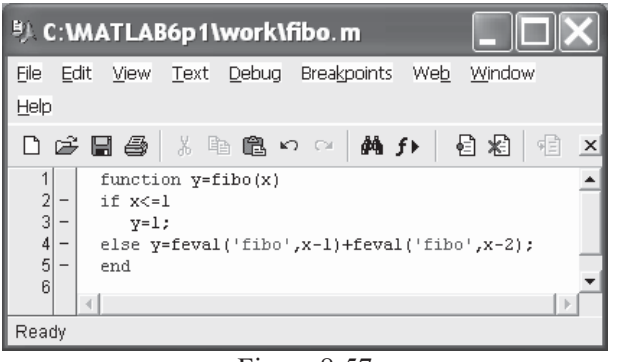


Figura 8-57

Los términos 2, 5 y 20 de la sucesión se calculan ahora mediante la sintaxis:

```
>> [fibo(2), fibo(5), fibo(20)]

ans =

           2           8       10946
```

Ejercicio 8-12. Definir la función delta de Kronecker, que vale 1 si x=0, y vale =0 si x es distinto de cero. Definir también la función delta de Kronecker modificada, que vale 0 si x=0, 1 si x>0 y -1 si x<0 y representarla gráficamente. Definir también la función a trozos que vale 0 si x≤-3, vale x3 si -3<x<-2, vale x2 si -2≤x≤2, vale x si 2<x<3 y vale 0 si 3≤x y representarla gráficamente.

Para definir la función delta de Kronecker *delta(x)* creamos el M-fichero *delta.m* de la Figura 8-58. Para definir la función delta de Kronecker modificada *delta1(x)* creamos el M-fichero *delta.m* de la Figura 8-59. Para definir la función a tyrozos *trozos1(x)* creamos el M-fichero *trozos1.m* de la Figura 8-60.

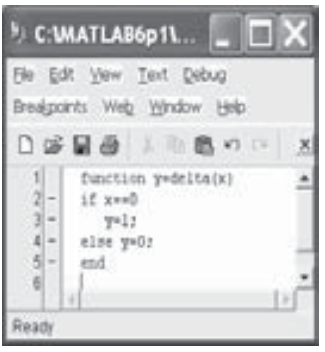


Figura 8-58

Figura 8-59

Figura 8-60

Para representar gráficamente la función delta de Kronecker modificada en [-10,10]*x*[-2,2] (Figura 8-61) se utiliza la siguiente sintaxis:

```
>> fplot('delta1(x)', [-10 10 -2 2])
>> title 'Delta de Kronecker modificada'
```

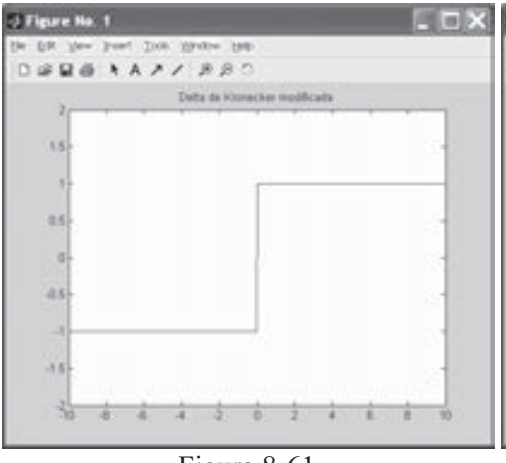


Figura 8-61

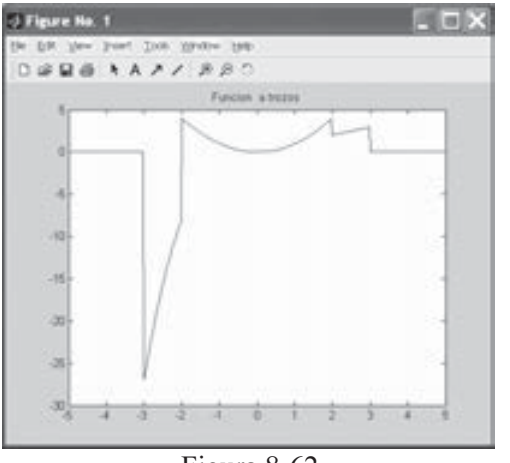


Figura 8-62

Para representar gráficamente la función a trozos en [-5,5] (Figura 8-62) se utiliza la siguiente sintaxis:

```
>> fplot('trozos1(x)', [-5 5]);
>> title 'Funcion  a trozos'
```

Ejercicio 8-13. Definir una función descriptiva(v) que devuelva la varianza y el coeficiente de variación de los elementos de un vector dado v. Como aplicación, hallar la varianza y el coeficiente de variación de los números 1, 5 , 6, 7 y 9.

En la Figura 8-63 se define el M-fichero *descriptiva* adecuado.

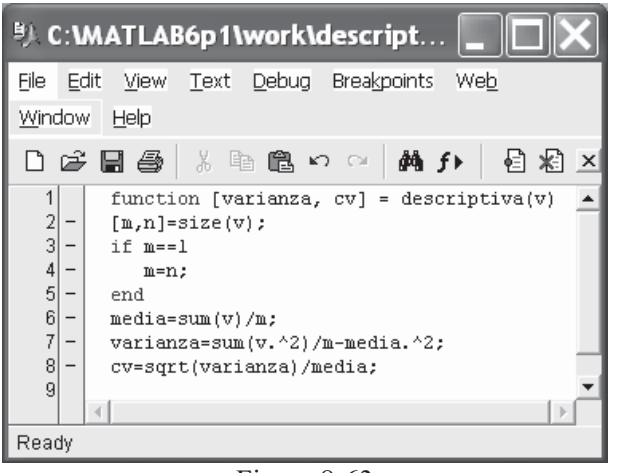


Figura 8-63

Para hallar la varianza y el coeficiente de variación del vector dado se utiliza la siguiente sintaxis:

```
>> [varianza, cv]=descriptiva([1 5 6 7 9])

varianza =

    7.0400

cv =

    0.4738
```

9

Algoritmos de cálculo numérico: ecuaciones, derivadas e integrales

9.1 Resolución de ecuaciones no lineales

Los métodos de programación de MATLAB permiten abordar las técnicas de cálculo numérico mediante la implementación sencilla de los algoritmos teóricos que resuelven una casuística importante de aproximación a soluciones de problemas. Entre estos problemas juega un papel primordial la resolución de ecuaciones no lineales.

Es sencillo construir programas en el lenguaje de MATLAB que desarrollen los algoritmos y que se guardan en M-ficheros. Ya sabemos de capítulos anteriores que un M-fichero no es mas que código MATLAB que simplemente ejecuta una serie de comandos o funciones que aceptan argumentos y producen una salida. Los M-ficheros se crean utilizando el editor de texto, tal y como ya se vio en el Capítulo 2.

Método del punto fijo para resolver *x=g(x)*

El método iterativo del punto fijo trata de resolver la ecuación $x = g(x)$ bajo determinadas condiciones para la función g, mediante un método iterativo que parte de un valor inicial p_0 (aproximación a la solución) y que se define como $p_{k+1} = g(p_k)$. El teorema del punto fijo asegura que esta sucesión así definida converge hacia una solución de la ecuación $x = g(x)$. En la práctica el proceso iterativo hay que detenerlo cuando el error absoluto o relativo correspondiente a dos iteraciones consecutivas es inferior a una cota prefijada (*tolerancia*). Cuanto menor sea esta cota, mejor será la aproximación a la solución de la ecuación que ofrece el método iterativo.

Puede definirse un M-fichero mediante la sintaxis de la Figura 9-1 para implementar este método iterativo.

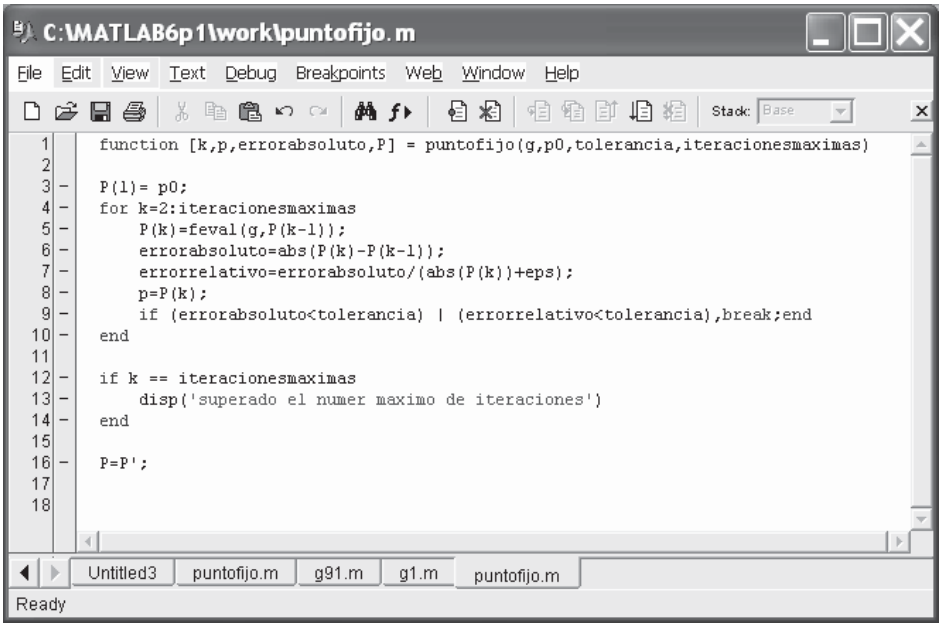


Figura 9-1

Como ejemplo de aplicación resolvemos la ecuación no lineal siguiente:

$$x^2 - x - Sen(x + 0{,}15) = 0$$

Para poder aplicar el algoritmo del punto fijo escribimos la ecuación en la forma $x = g(x)$ como sigue:

$$x = x^2 - Sen(x + 0{,}15) = g(x)$$

Comenzaremos intuyendo una solución aproximada para elegir adecuadamente p_o. Para ello representamos la curva cuya ecuación es la dada y el eje x sobre el mismo gráfico (Figura 9-2) mediante la siguiente sintaxis:

```
>> fplot('[x^2-x-sin(x+0.15),0]',[-2,2])
```

El gráfico nos muestra que existe una solución en un entorno de x = 1,5 y otra solución en un entorno de cero. Podemos tomar cualquiera de estos dos valores como aproximación inicial a la solución. Elegimos por ejemplo p_o = 1,5. Si consideramos una tolerancia de una diezmilésima para un número máximo de 100 iteraciones, podremos resolver el problema definiendo previamente la función $g(x)$ como el M-fichero *g1.m* de la Figura 9-3.

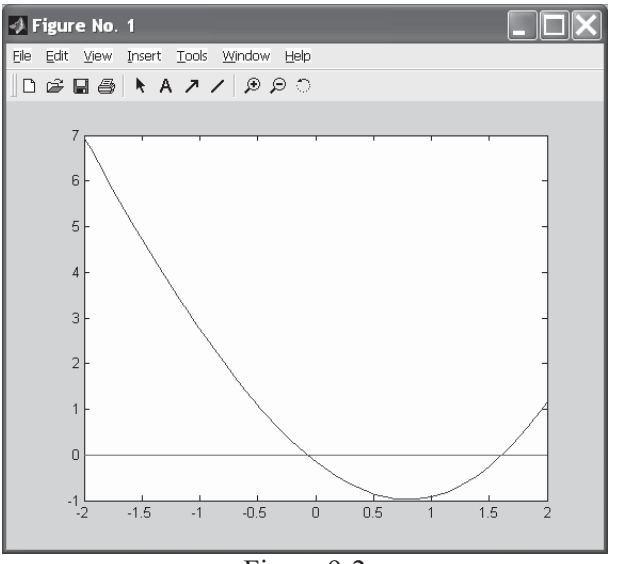


Figura 9-2

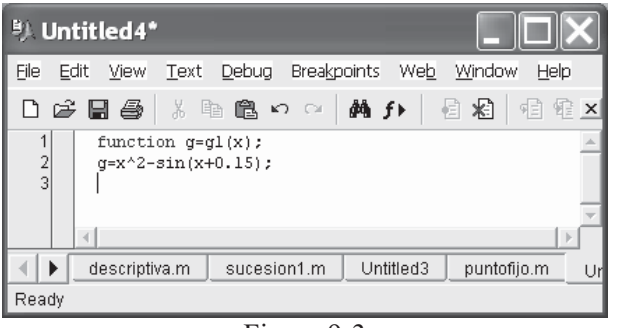


Figura 9-3

Podemos ya resolver la ecuación mediante la sintaxis MATLAB:

```
>> [k,p]=puntofijo('g1',1.5,0.1,1000)
superado el numer maximo de iteraciones
k =
        1000
p =
   -0.3513
```

Obtenemos como solución x = -0,3513 en 100 iteraciones. Para comprobar si la solución es aproximadamente correcta, hemos de ver que g1(-0,3513) = 0,3513.

```
>> g1(0.3513)

ans =

   -0.3572
```

Observamos que la solución puede ser aceptable.

Método de Newton para resolver la ecuación $f(x)=0$

El método de Newton resuelve la ecuación $f(x) = 0$, bajo determinadas condiciones exigidas a f, mediante la iteración $x_{r+1} = x_r - f(x_r)/f'(x_r)$ para un valor inicial dado x_0 lo suficientemente próximo a una solución.

Podemos elaborar el M-fichero de la Figura 9-4 para construir el programa que resuelva ecuaciones por el método de Newton con una determinada precisión.

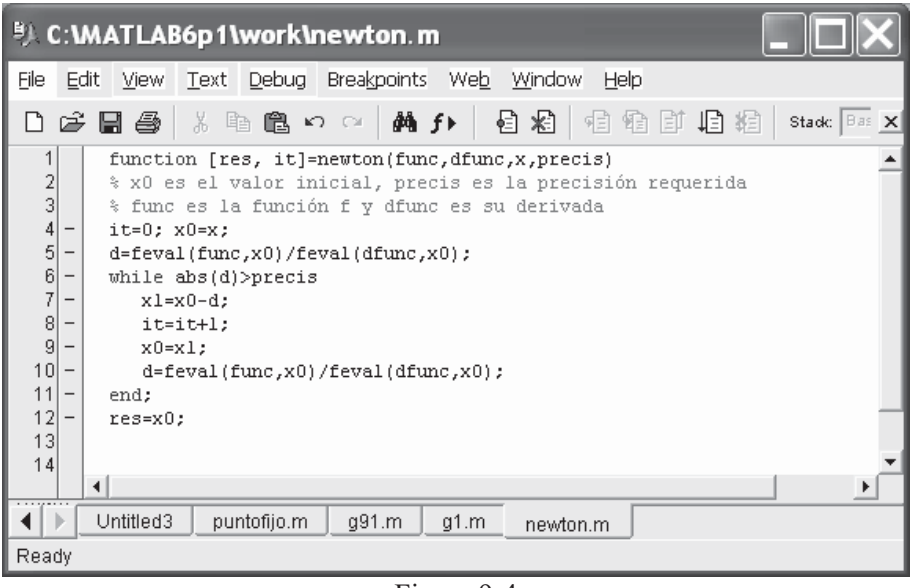


Figura 9-4

Como ejemplo resolvemos por el método de Newton la ecuación del apartado anterior:

$$x^2 - x - Sen(x + 0{,}15) = 0$$

Definimos previamente la función $f(x)$ como el M-fichero *f1.m* de la Figura 9-5, y su función derivada $f'(x)$ como el M-fichero *derf1.m* de la Figura 9-6.

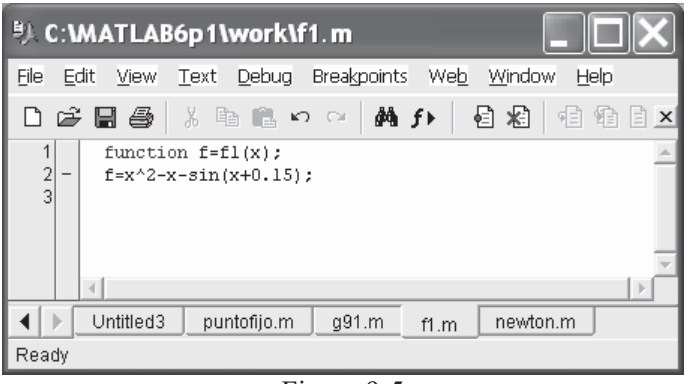


Figura 9-5

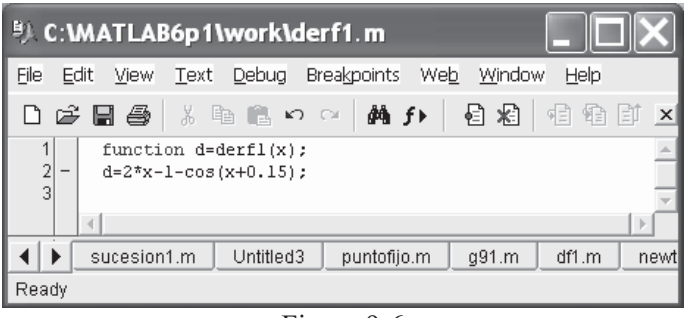


Figura 9-6

Podemos ya resolver la ecuación con precisiones de una diezmilésima y una millonésima mediante la sintaxis MATLAB siguiente:

```
>> [x,it]=newton('f1','derf1',1.5,0.0001)

x =

    1.6101


it =

     2

>> [x,it]=newton('f1','derf1',1.5,0.000001)
```

```
x =

    1.6100

it =

     3
```

Se ha obtenido como solución x=1,61 en 2 iteraciones para la precisión de una diezmilésima y en 3 iteraciones para la precisión de una millonésima.

Observamos que el método de Newton nos da la solución más cercana a x = 1,5, mientras que el método del punto fijo nos dio la solución más cercana a 0 (ver la Figura 9-2), aunque en ambos casos partimos de la misma solución inicial x=1.5.

Método de Schroder's para resolver la ecuación $f(x)=0$

El método de Schroder's, que es semejante al de Newton, resuelve la ecuación $f(x)=0$ bajo determinadas condiciones exigidas a f, mediante la iteración $x_{r+1} = x_r - m\, f(x_r)/f'(x_r)$ para un valor inicial dado x_0 lo suficientemente próximo a una solución, siendo m el orden de multiplicidad de la raíz buscada.

Podemos elaborar el M-fichero de la Figura 9-7 para construir el programa que resuelva ecuaciones por el método de Newton con una determinada precisión.

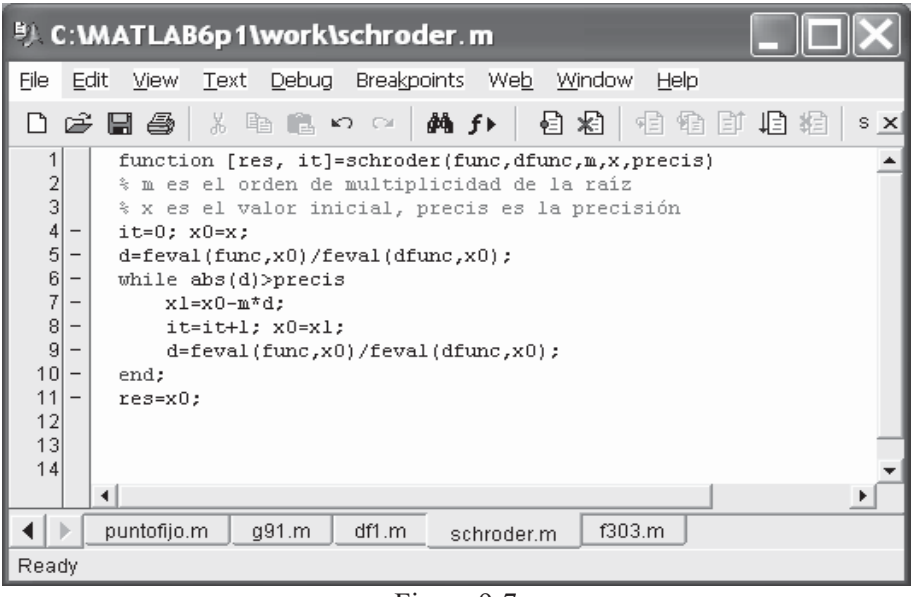


Figura 9-7

9.2 Resolución de sistemas de ecuaciones no lineales

Al igual que para el caso de las ecuaciones, es posible implementar con MATLAB algoritmos que resuelven sistemas de ecuaciones no lineales mediante métodos iterativos clásicos en el cálculo numérico.

Entre la diversidad de métodos existentes consideraremos el método de Seidel y el de Newton Raphson.

Método de Seidel

El método de Seidel para la resolución de sistemas es una generalización de la iteración del punto fijo para ecuaciones.

Para el caso del sistema de dos ecuaciones $x = g_1(x, y)$ e $y = g_2(x, y)$ las funciones del método de iteración del punto fijo se definen como:

$$p_{k+1} = g_1(p_k, q_k) \text{ y } q_{k+1} = g_2(p_k, q_k)$$

Análogamente, para el caso del sistema de tres ecuaciones $x = g_1(x, y, z)$, $y = g_2(x, y, z)$ y $z = g_3(x, y, z)$ las funciones del método de iteración del punto fijo se definen como:

$$p_{k+1} = g_1(p_k, q_k, r_k) \text{ , } q_{k+1} = g_2(p_k, q_k, r_4) \text{ y } r_{k+1} = g_3(p_k, q_k, r_4)$$

Siguiendo el camino del algoritmo del punto fijo generalizado para varias ecuaciones, podemos construir el M-fichero de la Figura 9-8.

Método de Newton-Raphson

El método de Newton-Raphson para la resolución de sistemas es una generalización del mismo método para ecuaciones simples.

El camino de trabajo para construir el algoritmo es el habitual. La solución del sistema de ecuaciones no lineales $F(X) = 0$ se obtiene generando a partir de una aproximación inicial P_0 una sucesión P_k que converge a la solución. De esta forma podemos construir el algoritmo generalizado de Newton-Raphson a partir del M-fichero de la Figura 9-9.

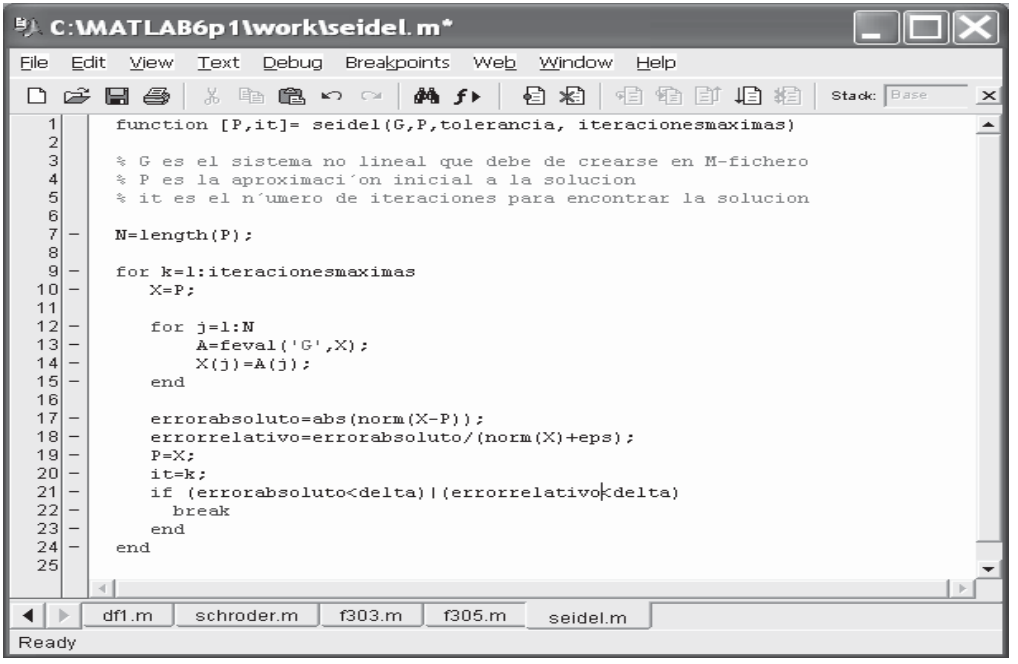


Figura 9-8

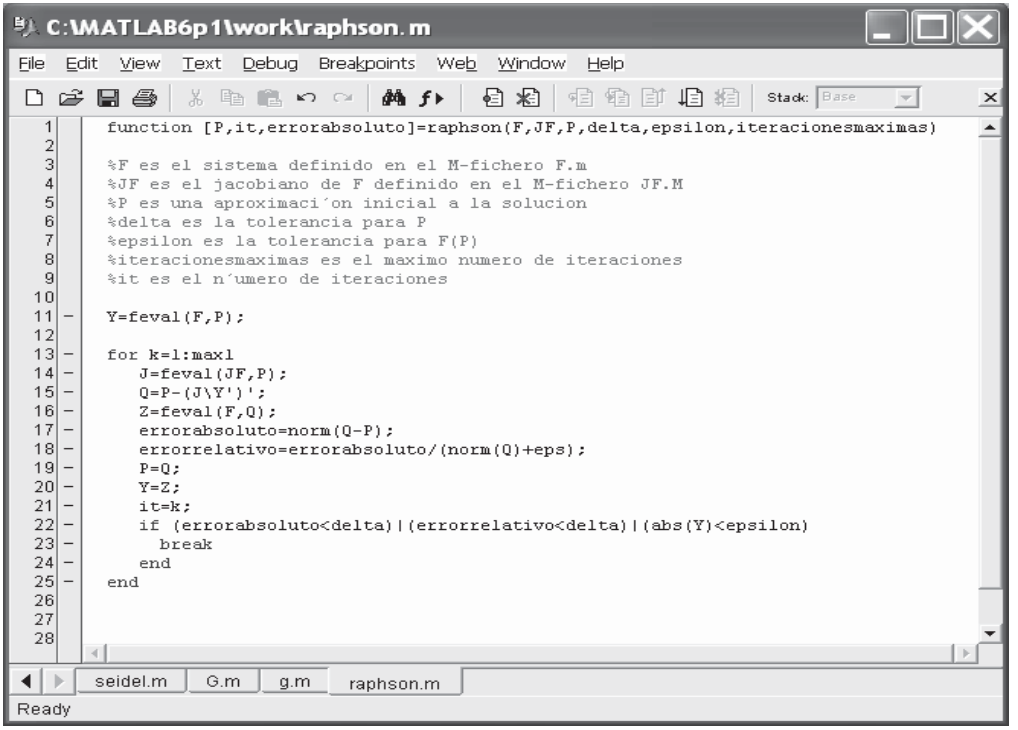


Figura 9-9

Como ejemplo resolvemos por el método de Newton-Raphson el sistema siguiente:

$$x^2 - 2x - y = -0.5$$
$$x^2 + 4y^2 - 4 = 0$$

partiendo de una aproximación inicial a la solución P = [2 3]

Comenzamos definiendo el sistema $F(X) = 0$ y su matriz jacobiana JF según los M-ficheros *F.m* y *JF.m* de las Figuras 9-10 y 9-11.

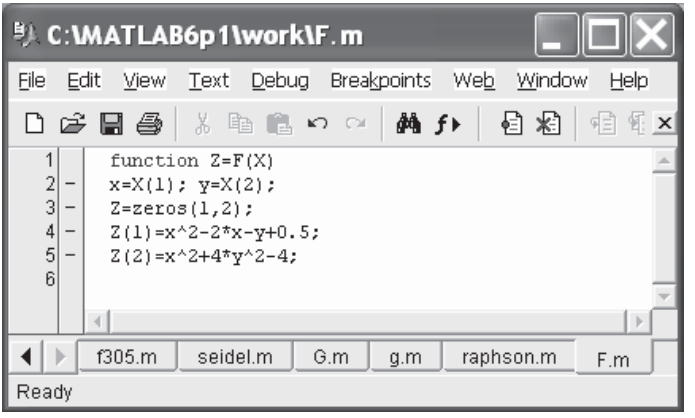


Figura 9-10

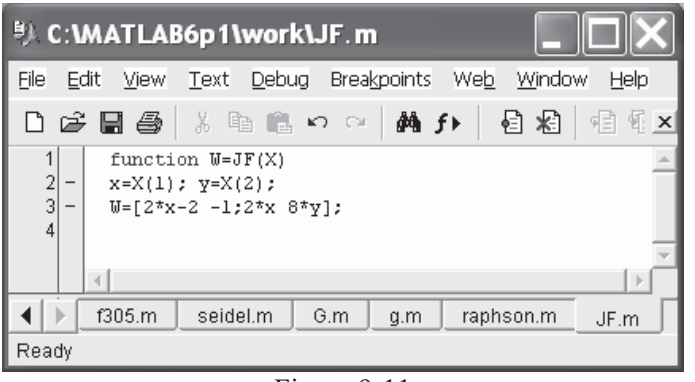


Figura 9-11

A continuación se resuelve el sistema, con tolerancias de 0,00001 y 100 iteraciones máximas, utilizando la sintaxis MATLAB siguiente:

```
>> [P,it,errorabsoluto]=raphson('F','JF',[2 3],0.00001,0.00001,100)
```

```
P =

    1.9007    0.3112

it =

     6

errorabsoluto =

  8.8751e-006
```

La solución obtenida en 6 iteraciones es x = 1,9007, y = 0,3112 con un error absoluto de valor 8.8751*e*-006.

9.3 Métodos de interpolación

Existen diferentes métodos para encontrar un polinomio interpolador que ajuste de la mejor forma posible una nube de puntos dada.

Entre los métodos más comunes de interpolación tenemos el polinomio interpolador de Lagrange, el polinomio interpolador de Newton y la aproximación de Chebyshev.

Polinomio interpolador de Lagrange

El polinomio interpolador de Lagrange que pasa por los N+1 puntos (x_k y_k) k=0,1, …, N se expresa como sigue:

$$P(x) = \sum_{k=0}^{N} y_k L_{N,k}(x)$$

donde:

$$L_{N,k}(x) = \frac{\prod_{\substack{j=0 \\ j\neq k}}^{N} (x - x_j)}{\prod_{\substack{j=0 \\ j\neq k}}^{N} (x_k - x_j)}$$

El algoritmo de obtención de P y L puede implementarse fácilmente mediante el M-fichero de la Figura 9-12.

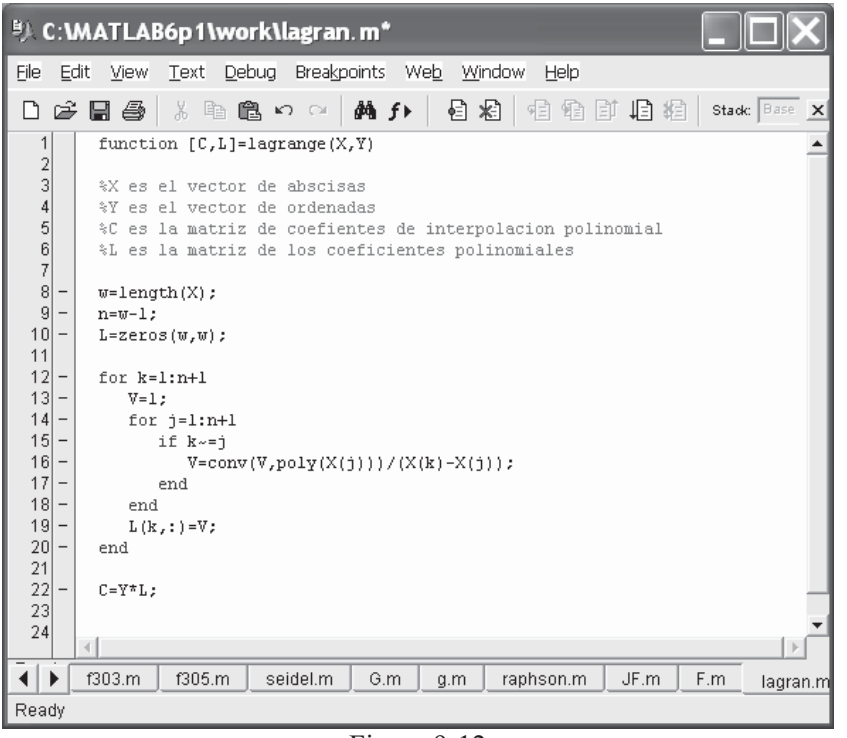


Figura 9-12

Como ejemplo podemos hallar el polinomio interpolador de Lagrange que pasa por los puntos (2,3), (4,5), (6,5), (7,6), (8,8) y (9,7).

Sencillamente utilizaremos la siguiente sintaxis MATLAB:

```
>> [C,L]=lagrange([2 4 6 7 8 9],[3 5 5 6 8 7 ])

C =

   -0.0185    0.4857   -4.8125   22.2143  -46.6690   38.8000

L =

   -0.0006    0.0202   -0.2708    1.7798   -5.7286    7.2000
    0.0042   -0.1333    1.6458   -9.6667   26.3500  -25.2000
   -0.0208    0.6250   -7.1458   38.3750  -94.8333   84.0000
    0.0333   -0.9667   10.6667  -55.3333  132.8000 -115.2000
   -0.0208    0.5833   -6.2292   31.4167  -73.7500   63.0000
    0.0048   -0.1286    1.3333   -6.5714   15.1619  -12.8000
```

Podemos obtener la forma simbólica del polinomio cuyos coeficientes son el vector *C* mediante la siguiente sintaxis MATLAB:

```
>> pretty(poly2sym(C))

        31   5   1093731338075689  4   77  3   311  2   19601
    - ---- x  + ---------------- x  - -- x  + --- x  - ----- x + 194/5
      1680      2251799813685248      16      14        420
```

Polinomio interpolador de Newton

El polinomio interpolador de Newton que pasa por los N+1 puntos definidos como $(x_k\, y_k) = (x_k, f(x_k))$ k=0,1,…, N se expresa como sigue:

$$P(x) = d_{0,0} + d_{1,1}(x - x_0) + d_{2,2}(x - x_0)(x - x_1) + \cdots + d_{N,N}(x - x_0)(x - x_1)\cdots(x - x_{N-1})$$

donde:

$$d_{k,j} = y_k \qquad d_{k,j} = \frac{d_{k,j-1} - d_{k-1,j-1}}{x_k - x_{k-j}}$$

El algoritmo de obtención de los coeficientes del polinomio interpolador C y la tabla de diferencias divididas D puede implementarse fácilmente mediante el M-fichero de la Figura 9-13.

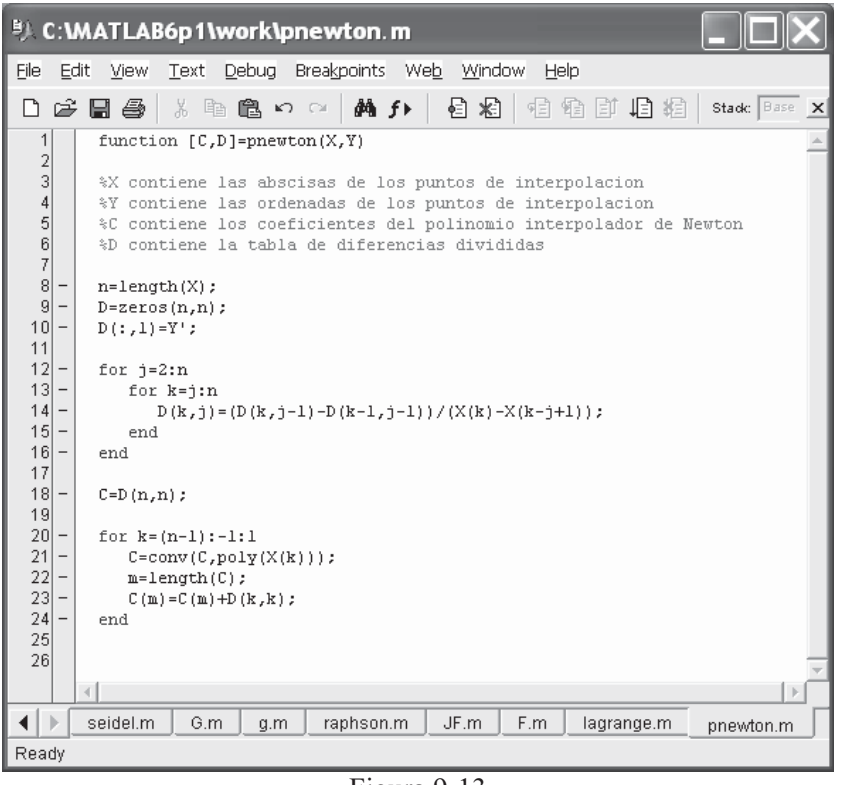


Figura 9-13

Como ejemplo podemos realizar por el método de Newton la misma interpolación realizada por el método de Lagranje en el apartado anterior. Utilizaremos la siguiente sintaxis MATLAB:

```
>> [C,D]=pnewton([2 4 6 7 8 9],[3 5 5 6 8 7 ])

C =

   -0.0185    0.4857   -4.8125   22.2143  -46.6690   38.8000

D =

    3.0000         0         0         0         0         0
    5.0000    1.0000         0         0         0         0
    5.0000         0   -0.2500         0         0         0
    6.0000    1.0000    0.3333    0.1167         0         0
    8.0000    2.0000    0.5000    0.0417   -0.0125         0
    7.0000   -1.0000   -1.5000   -0.6667   -0.1417   -0.0185
```

El polinomio interpolador en forma simbólica se calcula como sigue:

```
>> pretty(poly2sym(C))

                31   5   17  4   77  3   311  2   19601
              - ---- x  + -- x  - -- x  + --- x  - ----- x + 194/5
                1680      35      16      14        420
```

Se observa que el resultado obtenido por ambos métodos de interpolación es similar.

9.4 Métodos de derivación numérica

Existen diferentes métodos para obtener fórmulas de derivación numérica. Estas fórmulas tienen mucha importancia en el desarrollo de algoritmos para resolver problemas de contorno de ecuaciones diferenciales ordinarias y ecuaciones en derivadas parciales.

Entre los métodos más comunes de derivación numérica tenemos la derivación mediante límites, la derivación usando extrapolación y la derivación badas en *N*-1 nodos.

Derivación numérica mediante límites

Este método consiste en construir las aproximaciones numéricas a $f'(x)$ mediante la generación de la sucesión:

$$f'(x) \approx D_k = \frac{f(x+10^{-k}h) - f(x-10^{-k}h)}{2(10^{-k}h)} \quad k = 0, \cdots, n$$

Las iteraciones continúan hasta que $|D_{n+1} - D_n| \geq |D_n - D_{n-1}|$ o bien hasta que $|D_n - D_{n-1}| <$ tolerancia. Este criterio aproxima $f'(x)$ por D_n.

El algoritmo de obtención de la derivada D puede implementarse fácilmente mediante el M-fichero de la Figura 9-14.

C:\MATLAB6p1\work\derivadalim.m*

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
function  [L,n]=derivadalim(f,x,tolerancia)

% f es la funcion a derivar
% x es el punto de derivacion
% L=[H' D' E'] donde H es el vector de tamaños de paso,
% D es el vector de derivadas aproximadas y E es el vector de cotas de error
% n es la coordenada de la mejor aproximacion

iteracionesmaximas=15;
h=1;
H(1)=h;
D(1)=(feval(f,x+h)-feval(f,x-h))/(2*h);
E(1)=0;
R(1)=0;

for n=1:2
   h=h/10;
   H(n+1)=h;
   D(n+1)=(feval(f,x+h)-feval(f,x-h))/(2*h);
   E(n+1)=abs(D(n+1)-D(n));
   R(n+1)=2*E(n+1)*(abs(D(n+1))+abs(D(n))+eps);
end

n=2;

while((E(n)>E(n+1))&(R(n)>tolerancia))&n<iteracionesmaximas
   h=h/10;
   H(n+2)=h;
   D(n+2)=(feval(f,x+h)-feval(f,x-h))/(2*h);
   E(n+2)=abs(D(n+2)-D(n+1));
   R(n+2)=2*E(n+2)*(abs(D(n+2))+abs(D(n+1))+eps);
   n=n+1;
end

n=length(D)-1;
L=[H' D' E'];
```

g.m | raphson.m | JF.m | F.m | lagrange.m | pnewton.m | derivadalim.m

Ready

Figura 9-14

Como ejemplo aproximamos la derivada de la función:

$$f(x) = Sen\left(Cos\left(\frac{1}{x}\right)\right)$$

en el punto $\frac{1-\sqrt{5}}{2}$.

Para comenzar definimos la función *f* en el M-fichero de nombre funcion de la Figura 9-15.

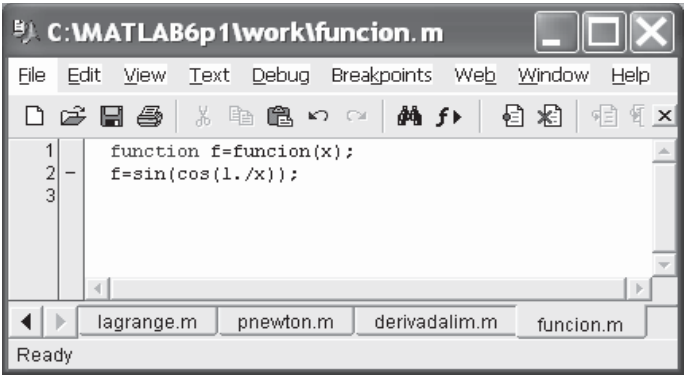


Figura 9-15

La derivada se obtiene mediante la siguiente sintaxis de MATLAB:

```
>> [L,n]=derivadalim('funcion',(1-sqrt(5))/2,0.01)

L =
    1.0000   -0.7448         0
    0.1000   -2.6045    1.8598
    0.0100   -2.6122    0.0077
    0.0010   -2.6122    0.0000
    0.0001   -2.6122    0.0000

n =
     4
```

Hemos obtenido que la derivada aproximada es -2,6122, resultado que puede comprobarse como sigue:

```
>> f=diff('sin(cos(1/x))')

f =
cos(cos(1/x))*sin(1/x)/x^2

>> subs(f,(1-sqrt(5))/2)
ans =
   -2.6122
```

Método de extrapolación de Richardson

Este método consiste en construir las aproximaciones numéricas a $f'(x)$ mediante la construcción de la tabla $D(j,k)$ con $k \leq j$ en la que se obtienen como solución final para la derivada $f'(x) = D(n,n)$. Los valores $D(j,k)$ forman una matriz triangular inferior cuya primera columna se define como:

$$D(j,1) = \frac{f(x+2^{-j}h) - f(x-2^{-j}h)}{2^{-j+1}h}$$

y cuya fila j-ésima para $j \geq 2$ tiene los siguientes elememntos:

$$D(j,k) = D(j,k-1) + \frac{D(j,k-1) - D(j-1,k-1)}{4^k - 1} \quad (2 \leq k \leq j)$$

El algoritmo de obtención de la derivada D puede implementarse fácilmente mediante el M-fichero de la Figura 9-16.

C:\MATLAB6p1\work\richardson.m

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
function [D,errorabsoluto,errorrelativo,n]=richardson(f,x,delta,tolerancia)

%f es la funci´on a derivar, x es el punto en el que se halla la derivada,
%delta es la tolernacia del error, tolerancia es la tolerancia del error relativo
%y D es la matriz de aproximaci´on de las derivadas

errorabsoluto=1;
errorrelativo=1;
h=1;
j=1;
D(1,1)=(feval(f,x+h)-feval(f,x-h))/(2*h);

while errorrelativo > tolerancia & errorabsoluto > delta &j <12
   h=h/2;
   D(j+1,1)=(feval(f,x+h)-feval(f,x-h))/(2*h);
   for k=1:j
      D(j+1,k+1)=D(j+1,k)+(D(j+1,k)-D(j,k))/((4^k)-1);
   end
   errorabsoluto=abs(D(j+1,j+1)-D(j,j));
   errorrelativo=2*errorabsoluto/(abs(D(j+1,j+1))+abs(D(j,j))+eps);
   j=j+1;
end

[n,n]=size(D);
```

pnewton.m | derivadalim.m | funcion.m | richardson.m

Ready

Figura 9-16

Como ejemplo aproximamos la derivada de la función:

$$f(x) = Sen\left(Cos\left(\frac{1}{x}\right)\right)$$

en el punto $\frac{1-\sqrt{5}}{2}$.

Como el M-fichero que define la función *f* ya está definido en el apartado anterior, calcularemos ya la derivada aproximada mediante la sintaxis MATLAB:

```
>> [D,errorrelativo,errorabsoluto,n]=richardson('funcion',
   (1-sqrt(5))/2,0.001,0.001)

D =

   -0.7448         0         0         0         0         0
   -1.1335   -1.2631         0         0         0         0
   -2.3716   -2.7843   -2.8857         0         0         0
   -2.5947   -2.6691   -2.6614   -2.6578         0         0
   -2.6107   -2.6160   -2.6125   -2.6117   -2.6115         0
   -2.6120   -2.6124   -2.6122   -2.6122   -2.6122   -2.6122

errorrelativo =

  6.9003e-004

errorabsoluto =

  2.6419e-004

n =

     6
```

Se observa que se obtiene el mismo resultado que para el método de la derivación numérica mediante límite.

Derivación mediante interpolación (*N*+1 nodos)

Este método consiste en construir el polinomio interpolador de Newton de grado *N*:

$$P(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_N(x-x_0)(x-x_1)\cdots(x-x_{N-1})$$

y aproximar numéricamente a $f'(x_0)$ mediante $P'(x_0)$.

El algoritmo de obtención de la derivada *D* puede implementarse fácilmente mediante el M-fichero de la Figura 9-17.

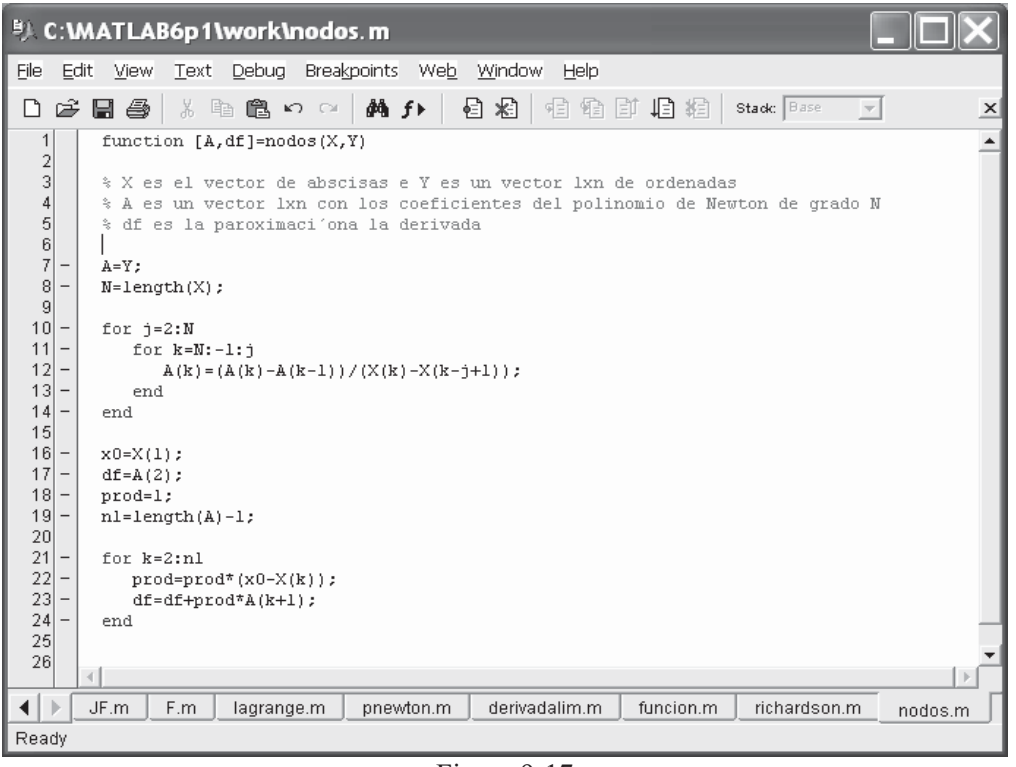


Figura 9-17

Como ejemplo aproximamos la derivada de la función:

$$f(x) = Sen\left(Cos\left(\frac{1}{x}\right)\right)$$

en el punto $\dfrac{1-\sqrt{5}}{2}$.

Como el M-fichero que define la función *f* ya está definido en el apartado anterior, calcularemos ya la derivada aproximada mediante la sintaxis MATLAB:

```
>> [A,df]=nodos([2 4 6 7 8 9],[3 5 5 6 8 7 ])

A =
    3.0000    1.0000   -0.2500    0.1167   -0.0125   -0.0185
df =
   -1.4952
```

9.5 Métodos de integración numérica

Dada la dificultad de obtener primitivas exactas para muchas funciones, los métodos e integración numérica adquieren importancia relevante. Existen diferentes métodos para obtener resultados numéricos aproximados para integrales definidas, entre los que destacan el método del trapecio, el método de Simpson y el método de Romberg (implementado en el módulo básico de MATLAB).

Método del trapecio

El método del trapecio para integración numérica tiene dos variantes: la regla compuesta del trapecio y la regla recursiva del trapecio.

La *regla compuesta del trapecio* aproxima la integral definida de la función $f(x)$ entre los puntos a y b de la siguiente forma:

$$\int_a^b f(x)dx \approx \frac{h}{2}(f(a)+f(b))+h\sum_{k=1}^{M-1} f(x_k)$$

calculándose $f(x)$ en los puntos equidistantes $x_k = a+kh$ $k=0,1, ..., M$ con $x_o=a$ y $x_M=b$.

La regla compuesta del trapecio puede implementarse mediante el M-fichero de la Figura 9-18.

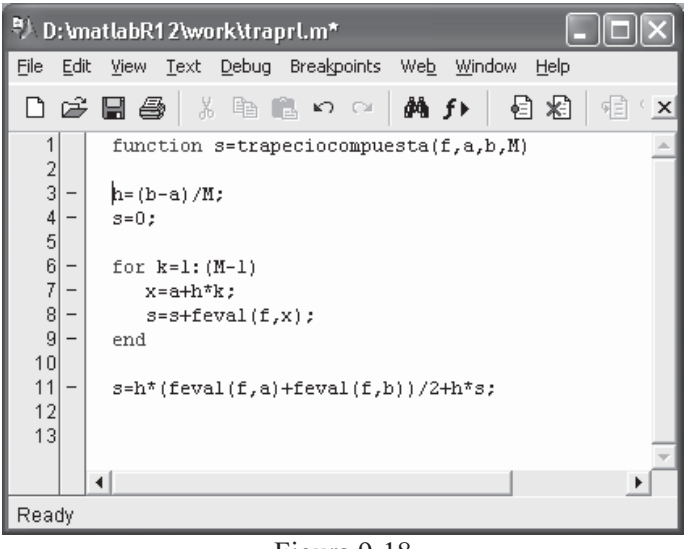


Figura 9-18

La *regla recursiva del trapecio* considera que los puntos $x_k=a+kh$ $k=0,1, ..., M$ con $x_o=a$ y $x_M=b$ dividen el intervalo $[a,b]$ en $2^J=2M$ subintervalos del mismo tamaño $h=(b-a)/2^J$. En esta situación se considera la siguiente fórmula recursiva:

$$T(0)=\frac{h}{2}(f(a)+f(b))$$

$$T(J)=\frac{T(J-1)}{2}+h\sum_{k=1}^{M}f(x_{2k-1}) \quad J=1,2,\cdots$$

y la integral definida de la función $f(x)$ entre los puntos a y b puede calcularse de la siguiente forma:

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{k=1}^{2^J}(f(x_k)+f(x_{k-1}))$$

utilizando la recursión del trapecio según se aumenta el número de subintervalos de $[a,b]$ y tomando en la iteración J un conjunto de 2^J+1 puntos equidistantes.

La regla recursiva del trapecio puede implementarse mediante el M-fichero de la Figura 9-19.

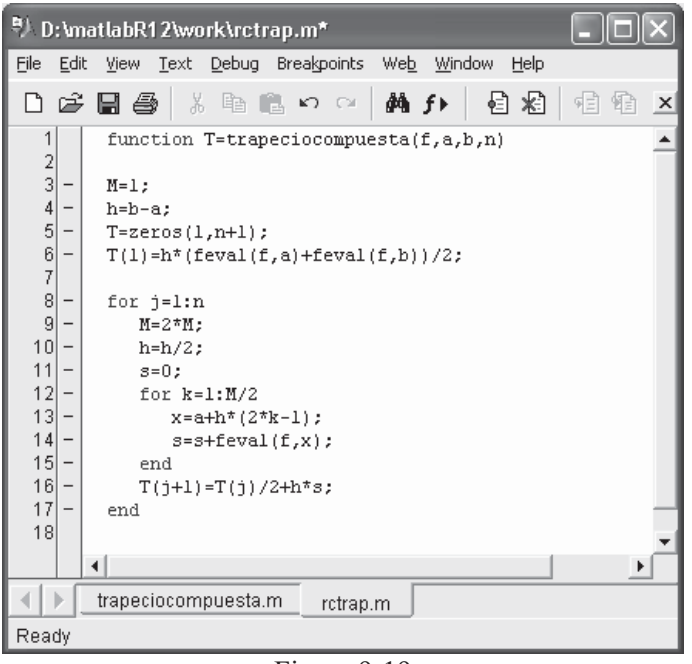


Figura 9-19

Como ejemplo calculamos en 100 iteraciones mediante la regla recursiva del trapecio la integral:

$$\int_0^2 \frac{1}{x^2+\frac{1}{10}}\,dx$$

Comenzamos definiendo la función del integrando mediante el M-fichero *integrando1.m* de la Figura 9-20.

Figura 9-20

A continuación calculamos la integral pedida como sigue:

```
>> trapeciorecursiva('integrando1',0,2,14)

ans =

  Columns 1 through 4

10.24390243902439       6.03104212860310       4.65685845031979
4.47367657743630

  Columns 5 through 8

4.47109102437123       4.47132194954670       4.47138003053334
4.47139455324593

  Columns 9 through 12

4.47139818407829       4.47139909179602       4.47139931872606
4.47139937545860

  Columns 13 through 15

   4.47139938964175   4.47139939318754   4.47139939407398
```

Se observa que el valor más preciso encontrado para la integral después de 14 iteraciones resulta ser 4,47139939407398.

A continuación resolvemos la misma integral por la regla compuesta del trapecio, utilizando M=14 subintervalos mediante la siguiente sintaxis MATLAB:

```
>> trapeciocompuesta('integrando1',0,2,14)

ans =
   4.47100414648074
```

Ahora el resultado será 4.47100414648074.

Método de Simpson

En el método de Simpson para integración numérica suelen considerarse dos variantes: la regla simple de Simpson y la regla compuesta de Simpson.

La *regla simple de Simpson* aproxima la integral definida de la función $f(x)$ entre los puntos a y b de la siguiente forma:

$$\int_a^b f(x)dx \approx \frac{h}{3}(f(a)+f(b)+4f(c)) \quad c=\frac{a+b}{2}$$

La regla simple de Simpson puede implementarse mediante el M-fichero de la Figura 9-21.

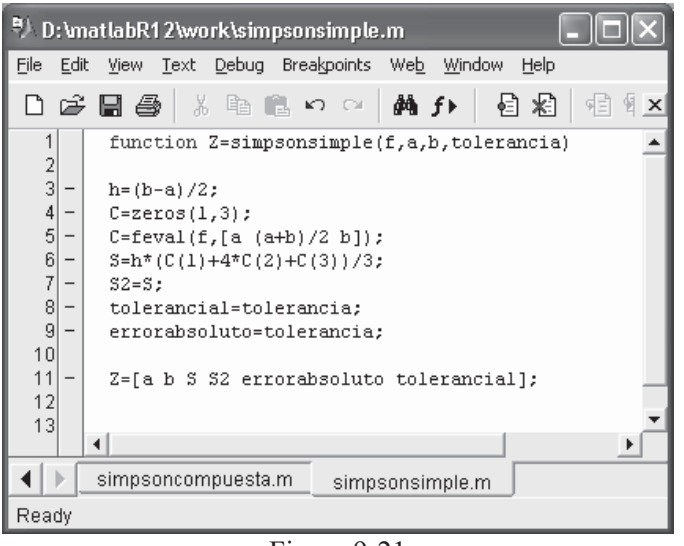


Figura 9-21

La *regla compuesta de Simpson* aproxima la integral definida de la función $f(x)$ entre los puntos a y b de la siguiente forma:

$$\int_a^b f(x)dx \approx \frac{h}{3}(f(a)+f(b))+\frac{2h}{3}\sum_{k=1}^{M-1} f(x_{2k})+\frac{4h}{3}\sum_{k=1}^{M} f(x_{2k-1})$$

calculándose $f(x)$ en los puntos equidistantes $x_k=a+kh \quad k=0,1,\ldots,2M$ con $x_o=a$ y $x_{2M}=b$.

La regla compuesta de Simpson puede implementarse mediante el M-fichero de la Figura 9-22.

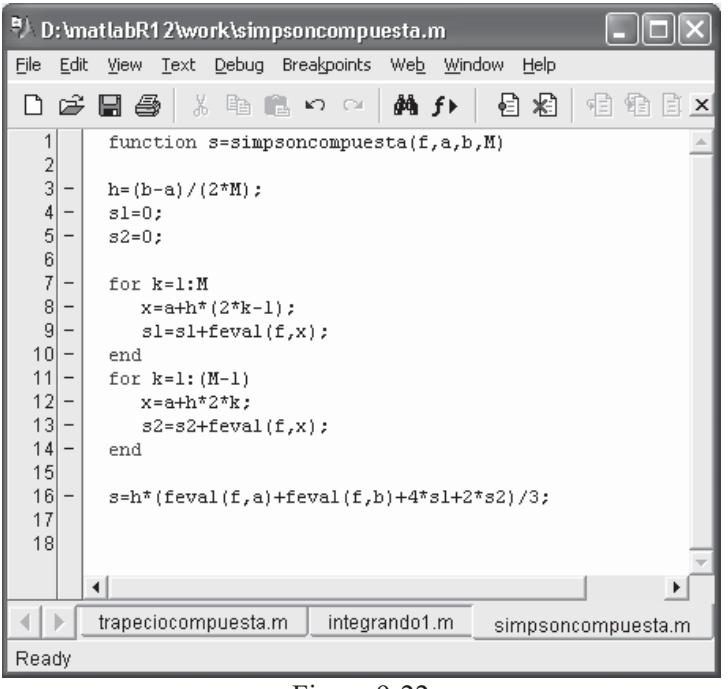


Figura 9-22

Como ejemplo calculamos mediante la regla compuesta de Simpson utilizando 2M=28 subintervalos la integral:

$$\int_0^2 \frac{1}{x^2+\frac{1}{10}}dx$$

Utilizaremos la siguiente sintaxis:

```
>> simpsoncompuesta('integrando1',0,2,14)

ans =

  4.47139628125498
```

A continuación calculamos la misma integral mediante la regla de Simpson simple usando la siguiente sintaxis:

```
>> Z=simpsonsimple('integrando2',0,2,0.0001)

Z =

  Columns 1 through 4

   0   2.00000000000000   4.62675535846268   4.62675535846268

  Columns 5 through 6

   0.00010000000000   0.00010000000000
```

Como vemos, la regla de Simpson simple es menos precisa que la regla compuesta.

En este caso hemos definido previamente el integrando como el M-fichero de nombre *integrando2.m* de la Figura 9-23

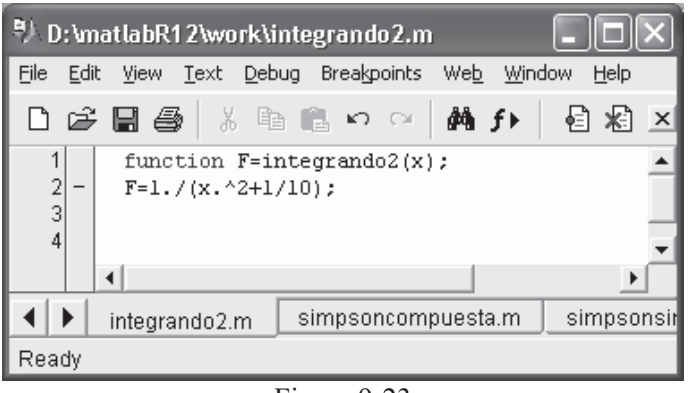


Figura 9-23

9.6 Ecuaciones diferenciales ordinarias

Obtener soluciones exactas de ecuaciones diferenciales ordinarias no es una tarea sencilla. Existen diferentes métodos para obtener resultados numéricos aproximados para las soluciones de ecuaciones diferenciales ordinarias, entre los que destacan el método de Euler, el método de Heun, el método de las series de Taylor, los métodos de Runge-Kutta (implementados en el módulo básico de MATLAB), el método de Adams-Bashforth-Moulton, el método de Milne y el método de Hamming.

Método de Euler

Sea $[a,b]$ el intervalo en el que queremos resolver la ecuación diferencial $y'=f(t,y)$ con $y(a)=y_0$. Dividimos el intervalo $[a,b]$ en M subintervalos del mismo tamaño usando la partición dada por los puntos $t_k=a+kh$ $k=0,1, ..., M$ con $h=(b-a)/M$. El método de Euler obtiene las aproximaciones a la solución de la ecuación diferencial mediante la iteración $y_{k+1} = y_k+hf(t_k, y_k)$ $k=0,1, ..., M-1$.

El método de Euler puede implementarse mediante el M-fichero de la Figura 9-24.

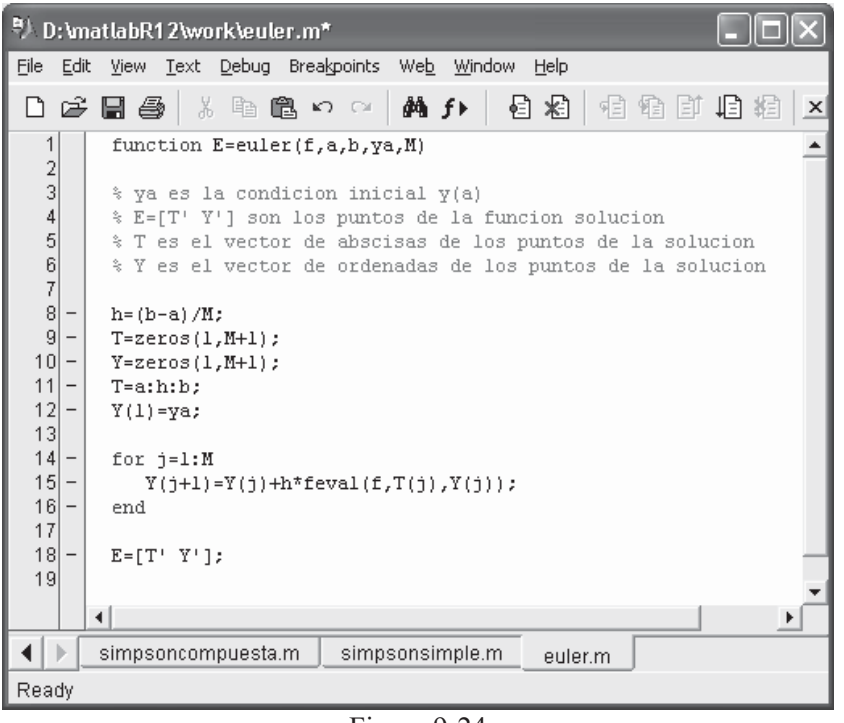


Figura 9-24

Método de Heun

Sea $[a,b]$ el intervalo en el que queremos resolver la ecuación diferencial $y'=f(t,y)$ con $y(a)=y_0$. Dividimos el intervalo $[a,b]$ en M subintervalos del mismo tamaño usando la partición dada por los puntos $t_k=a+kh$ $k=0,1,...,M$ con $h=(b-a)/M$. El método de Heun obtiene las aproximaciones a la solución de la ecuación diferencial mediante la iteración $y_{k+1}=y_k+h(f(t_k,y_k)+f(t_{k+1},y_k+f(t_k,y_k)))/2$ $k=0,1,...,M-1$.

El método de Heun puede implementarse mediante el M-fichero de la Figura 9-25.

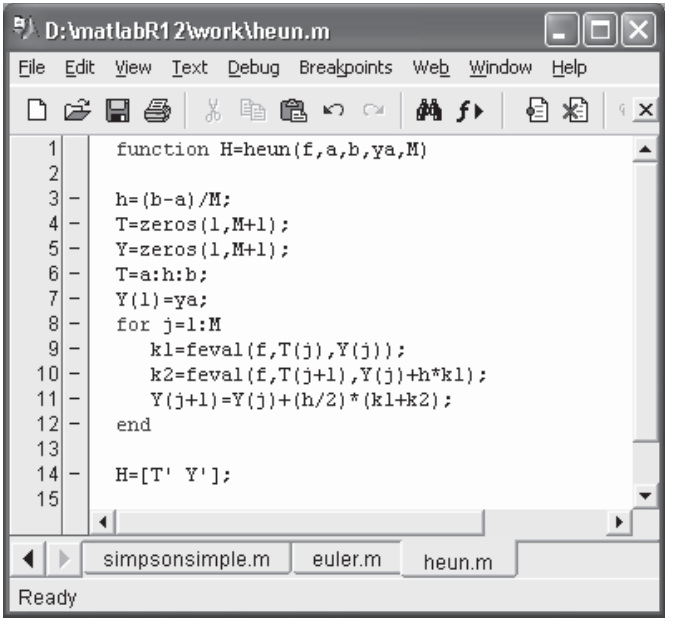


Figura 9-25

Método de las series de Taylor

Sea $[a,b]$ el intervalo en el que queremos resolver la ecuación diferencial $y'=f(t,y)$ con $y(a)=y_0$. Dividimos el intervalo $[a,b]$ en M subintervalos del mismo tamaño usando la partición dada por los puntos $t_k=a+kh$ $k=0,1,...,M$ con $h=(b-a)/M$. El método de las series de Taylor (vamos a considerarlo de orden 4) obtiene las aproximaciones a la solución de la ecuación diferencial evaluando y', y'', y''' e y'''' mediante sus respectivos desarrollos en series de Taylor de orden 4.

El método de las series de Taylor puede implementarse mediante el M-fichero de la Figura 9-26.

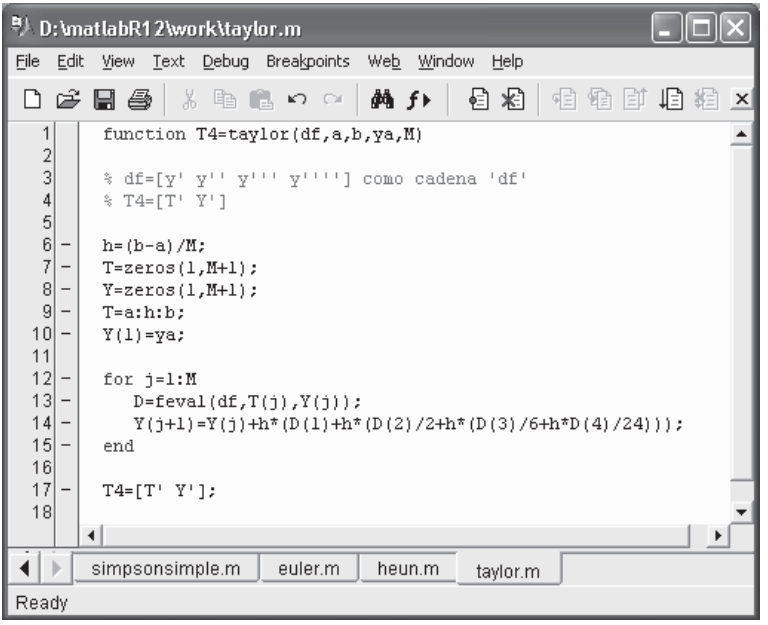


Figura 9-26

Como ejemplo resolvemos la ecuación diferencial $y'(t) = (t-y)/2$ en [0,3] con $y(0)=1$ mediante los métodos de Euler, Heun y series de Taylor.

Comenzaremos definiendo la función $f(t,y)$ mediante el M-fichero de la Figura 9-27.

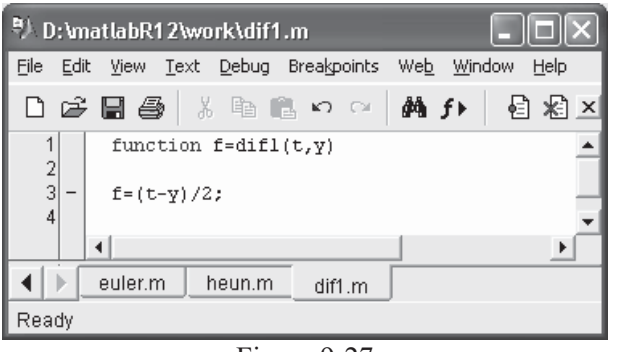


Figura 9-27

La solución de la ecuación por el método de Euler en 100 pasos se calculará como sigue:

```
>> E=euler('dif1',0,3,1,100)
```

```
E =

                  0   1.00000000000000
   0.03000000000000   0.98500000000000
   0.06000000000000   0.97067500000000
   0.09000000000000   0.95701487500000
   0.12000000000000   0.94400965187500
   0.15000000000000   0.93164950709688
   0.18000000000000   0.91992476449042
      .
      .
      .
   2.85000000000000   1.56377799005910
   2.88000000000000   1.58307132020821
   2.91000000000000   1.60252525040509
   2.94000000000000   1.62213737164901
   2.97000000000000   1.64190531107428
   3.00000000000000   1.66182673140816
```

Esta solución puede graficarse (Figura 9-28) como sigue:

```
>> plot(E(:,2))
```

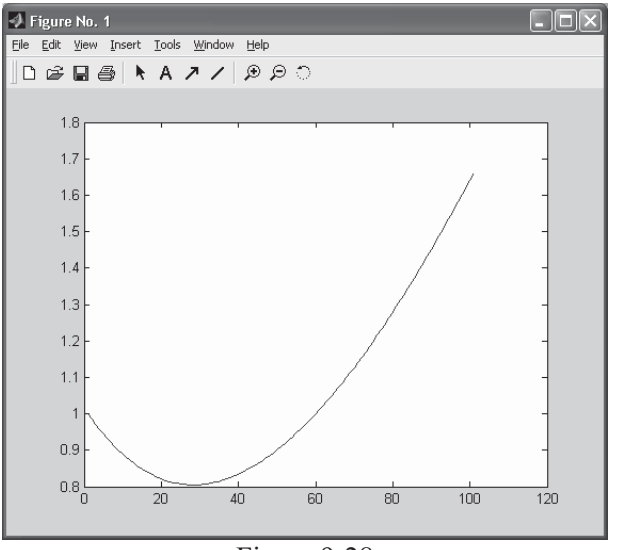


Figura 9-28

La solución de la ecuación por el método de Heun en 100 pasos se calculará como sigue:

```
>> H=heun('dif1',0,3,1,100)
```

```
H =
                   0   1.00000000000000
   0.03000000000000   0.98533750000000
   0.06000000000000   0.97133991296875
   0.09000000000000   0.95799734001443
   0.12000000000000   0.94530002961496
       .
       .
       .
   2.88000000000000   1.59082209379464
   2.91000000000000   1.61023972987327
   2.94000000000000   1.62981491089478
   2.97000000000000   1.64954529140884
   3.00000000000000   1.66942856088299
```

La solución mediante el método de las series de Taylor exige definir previamente la función *df* = [*y' y'' y''' y''''*] mediante el M-fichero de la Figura 9-29.

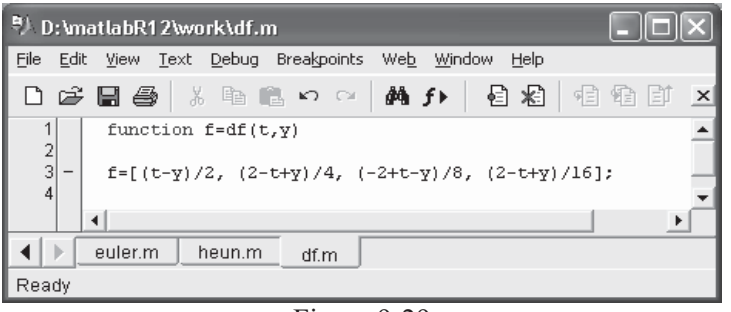


Figura 9-29

La ecuación diferencial se resolverá por Taylor mediante la sintaxis:

```
>> T=taylor('df',0,3,1,100)

T =
                   0   1.00000000000000
   0.03000000000000   0.98533581882813
   0.06000000000000   0.97133660068283
   0.09000000000000   0.95799244555443
   0.12000000000000   0.94529360082516
       .
       .
       .
   2.88000000000000   1.59078327648360
   2.91000000000000   1.61020109213866
   2.94000000000000   1.62977645599332
   2.97000000000000   1.64950702246046
   3.00000000000000   1.66939048087422
```

Ejercicio 9-1. Resolver mediante el método iterativo del punto fijo la ecuación no lineal siguiente:

$$x = Cos(Sen(x))$$

Comenzaremos intuyendo una solución aproximada para elegir p_o. Para ello representamos la curva x- $Cos(Sen(x))$ y el eje x sobre el mismo gráfico (Figura 9-30) mediante la siguiente sintaxis:

```
>> fplot('[x-cos(sin(x)),0]',[-2,2])
```

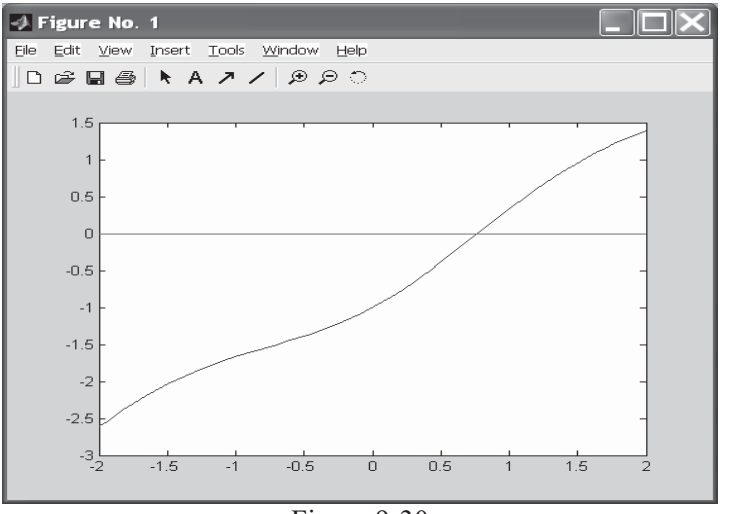


Figura 9-30

El gráfico nos muestra que existe una solución en un entorno de $x = 1$, valor que podemos tomar como aproximación inicial a la solución, es decir $p_o = 1$. Si consideramos una tolerancia de una diezmilésima para un número máximo de 100 iteraciones, podremos resolver el problema definiendo previamente la función $g(x) = Cos(Sen(x))$ como el M-fichero *g91.m* de la Figura 9-31.

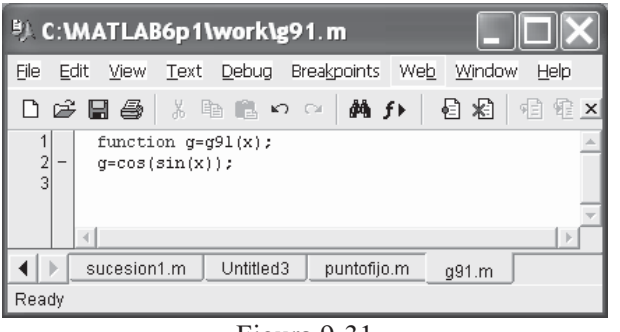


Figura 9-31

Podemos ya resolver la ecuación mediante la sintaxis MATLAB:

```
>> [k,p,errorabsoluto,P] = puntofijo('g91',1,0.0001,1000)
k =
    13
p =
    0.7682
errorabsoluto =
  6.3361e-005
P =
    1.0000
    0.6664
    0.8150
    0.7467
    0.7781
    0.7636
    0.7703
    0.7672
    0.7686
    0.7680
    0.7683
    0.7681
    0.7682
```

Se observa que la solución es $x = 0{,}7682$, solución que se ha encontrado en 13 iteraciones cometiendo un error absoluto de 6.3361*e*-005. Por lo tanto, la convergencia a la solución ha sido rápida y eficiente.

Ejercicio 9-2. Resolver por el método de Newton con una determinada precisión y aplicarlo al cálculo de una raíz de la ecuación $x^3 - 10x^2 + 29x - 20 = 0$ cercana al punto x=7 con una precisión de 0.00005. Utilizar también la precisión 0.0005.

Definimos las funciones $f(x) = x^3 - 10x^2 + 29x - 20$ y su derivada a través de los M-ficheros de nombres *f302.m* y *f303.m* según las Figuras 9-32 y 9-33

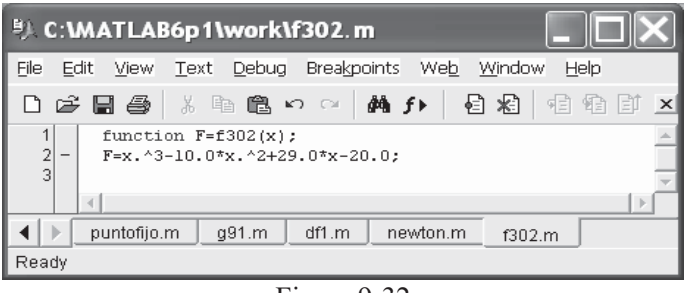


Figura 9-32

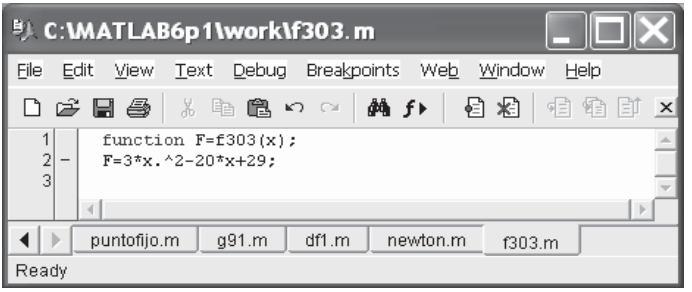


Figura 9-33

Para ejecutar el programa que resuelve la ecuación pedida tecleamos:

```
>> [x, it]=newton('f302','f303',7,.00005)
x =
    5.0000
it =
     6
```

Mediante un proceso de 6 iteraciones y precisión 0,00005 se ha obtenido la solución $x = 5$. Para 5 iteraciones y precisión 0,0005 se obtiene $x = 5{,}0002$ mediante:

```
>> [x, it]=newton('f302','f303',7,.0005)
x =
    5.0002
it =
     5
```

Ejercicio 9-3. Elaborar un programa que calcule una raíz con multiplicidad 2 de la ecuación $(e^{-x} - x)^2 = 0$ cercana al punto $x = -2$ con una precisión de 0.00005.

Definimos las funciones $f(x)= (e^{-x} - x)^2$ y su derivada en los M-ficheros de las Figuras 9-34 y 9-35 y las guardamos en los ficheros de nombres *f304.m* y *f305.m*:

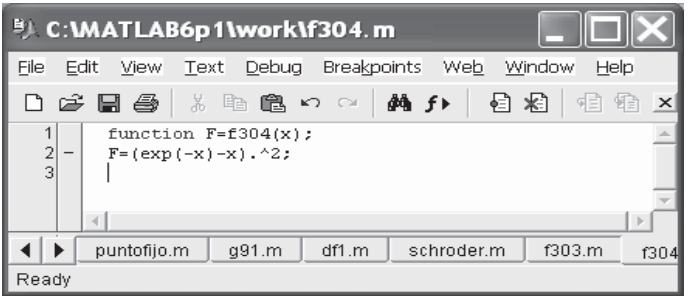


Figura 9-34

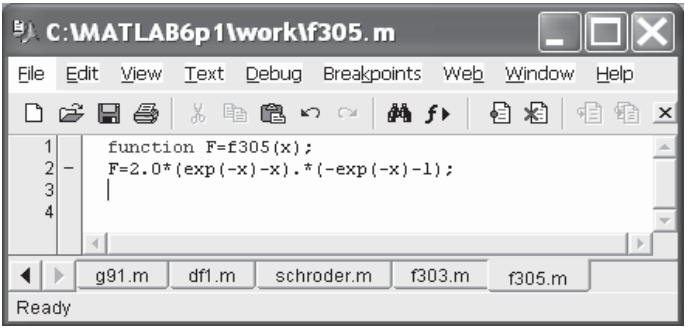


Figura 9-35

Para ejecutar el programa que resuelve la ecuación pedida tecleamos:

```
>> [x,it]=schroder('f304','f305',2,-2,.00005)
x =
    0.5671
it =
     5
```

Mediante un proceso de 5 iteraciones se ha obtenido la solución $x = 0.56715$.

Ejercicio 9-4. Aproximar la derivada de la función:

$$f(x) = Tan\left(Cos\left(\frac{\sqrt{5}+Sen(x)}{1+x^2}\right)\right)$$

en el punto $\dfrac{1+\sqrt{5}}{3}$

Para comenzar definimos la función *f* en el M-fichero de nombre *funcion1.m* de la Figura 9-36.

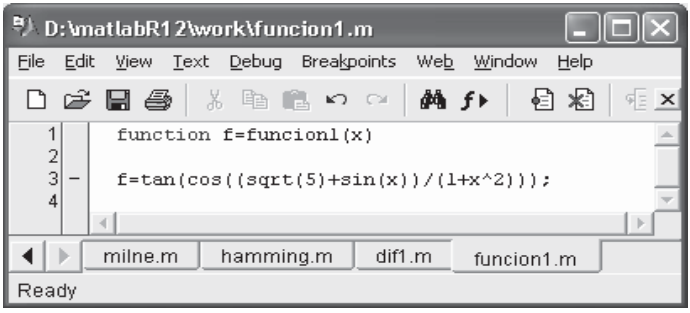


Figura 9-36

La derivada puede obtenerse por el método de derivación numérica mediante límites con precisión 0,0001 a través de la siguiente sintaxis de MATLAB:

```
>> [L,n]=derivadalim('funcion1',(1+sqrt(5))/3,0.0001)

L =

   1.00000000000000   0.94450896913313                   0
   0.10000000000000   1.22912035588668   0.28461138675355
   0.01000000000000   1.22860294102802   0.00051741485866
   0.00100000000000   1.22859747858110   0.00000546244691
   0.00010000000000   1.22859742392997   0.00000005465113

n =

     4
```

Se observa que el valor de la derivada se ha aproximado por 1,22859742392997.

Si utilizamos el método de Richardson, la derivada se calcula como sigue:

```
>> [D,errorabsoluto,errorrelativo,n]=richardson('funcion1',
   (1+sqrt(5))/3,0.0001,0.0001)

D =

  Columns 1 through 4

   0.94450896913313                  0                   0                   0
   1.22047776163545   1.31246735913623                   0                   0
   1.23085024935646   1.23430774526347   1.22909710433862                   0
   1.22938849854454   1.22890124827389   1.22854081514126   1.22853198515400
   1.22880865382036   1.22861537224563   1.22859631384374   1.22859719477553

  Column 5

                  0
                  0
                  0
                  0
   1.22859745049954

errorabsoluto =

    6.546534553897310e-005

errorrelativo =

    5.328603742973844e-005

n =

     5
```

Ejercicio 9-5. Aproximar la integral siguiente:

$$\int_{1}^{\frac{2\pi}{3}} Tan\left(Cos\left(\frac{\sqrt{5}+Sen(x)}{1+x^2}\right)\right)dx$$

Podemos utilizar la regla de Simpson compuesta con 100 subintervalos mediante la sintaxis siguiente:

```
>> s=simpsoncompuesta('funcion1',1,2*pi/3,100)

s =

   0.68600990924332
```

Si utilizamos la regla compuesta del trapecio se obtiene el resultado siguiente:

```
>> s=trapeciocompuesta('funcion1',1,2*pi/3,100)

s =

   0.68600381840334
```

Ejercicio 9-6. Aproximar en el intervalo [0, 0,8] la ecuación diferencial:

$$y' = t^2 + y^2 \quad y(0) = 1$$

Comenzamos definiendo la función *f*(*t*,*y*) mediante el M-fichero de la Figura 9-37.

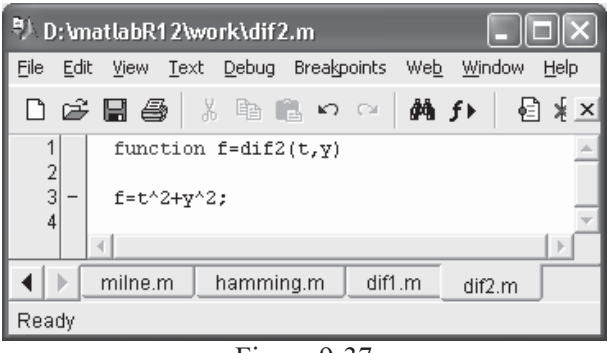


Figura 9-37

A continuación resolvemos la ecuación diferencial por el método de Euler en 20 pasos mediante la sintaxis siguiente:

```
>> E=euler('dif2',0,0.8,1,20)
```

```
E =
                   0   1.00000000000000
   0.04000000000000   1.04000000000000
   0.08000000000000   1.08332800000000
   0.12000000000000   1.13052798222336
   0.16000000000000   1.18222772296696
   0.20000000000000   1.23915821852503
   0.24000000000000   1.30217874214655
   0.28000000000000   1.37230952120649
   0.32000000000000   1.45077485808625
   0.36000000000000   1.53906076564045
   0.40000000000000   1.63899308725380
   0.44000000000000   1.75284502085643
   0.48000000000000   1.88348764754208
   0.52000000000000   2.03460467627982
   0.56000000000000   2.21100532382941
   0.60000000000000   2.41909110550949
   0.64000000000000   2.66757117657970
   0.68000000000000   2.96859261586445
   0.72000000000000   3.33959030062305
   0.76000000000000   3.80644083566367
   0.80000000000000   4.40910450907999
```

La solución puede graficarse (Figura 9-38) como sigue:

```
>> plot(E(:,2))
```

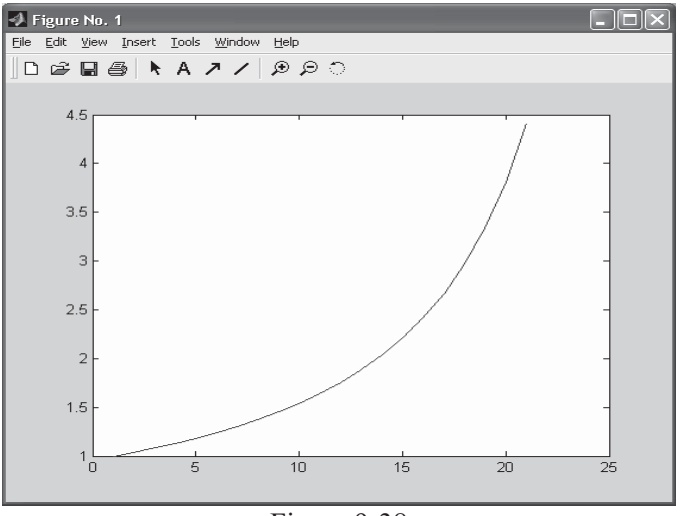


Figura 9-38

10

Cálculo simbólico: análisis matemático y álgebra

10.1 Cálculo simbólico con MATLAB. Variables simbólicas

MATLAB dispone del módulo *Symbolic Math Toolbox*, que permite manejar perfectamente el cálculo matemático simbólico, manipular con facilidad y rapidez las fórmulas y expresiones algebraicas y realizar la mayoría de operaciones con las mismas. Es posible expandir, factorizar y simplificar polinomios y expresiones racionales y trigonométricas; encontrar soluciones algebraicas de ecuaciones polinómicas y sistemas de ecuaciones; evaluar derivadas e integrales simbólicamente y encontrar funciones solución de ecuaciones diferenciales; manipular series de potencias, límites y muchas otras facetas de la matemática algebraica. Para realizar esta tarea, MATLAB requiere que todas las variables (o expresiones algebraicas) sean declaradas como simbólicas previamente con el comando *syms* (o con *sym*). Por ejemplo, si queremos tratar como simbólica la expresión 6**a***b* + 3**a*^{2} + 2**a***b* para simplificarla, tenemos que declarar sus dos variables *a* y *b* como simbólicas como sigue:

```
>> syms a b
>> simplify(6*a*b + 3*a^2 + 2*a*b)
ans =
8*a*b+3*a^2
```

También hubiera sido válida la siguiente expresión:

```
>> simplify(sym('6*a*b + 3*a^2 + 2*a*b'))
ans =
8*a*b+3*a^2
```

El *toolbox* de matemática simbólica de MATLAB aporta varios comandos para definición y conversión de variables simbólicas que se explican a continuación:

syms x y z...t	*Convierte las variables x, y, z, ..., t en simbólicas*
syms x y z...t real	*Convierte las variables x, y, z, ..., t en simbólicas con valores reales*
syms x y z...t unreal	*Convierte las variables x, y, z, ..., t en simbólicas con valores no reales*
syms	*Lista las variables simbólicas en el espacio de trabajo*
x=sym('x')	*Convierte la variable x en simbólica (equivale a syms x)*
x=sym('x',real)	*Convierte a x en una variable simbólica real*
x=sym('x',unreal)	*Convierte a x en una variable simbólica no real*
S=sym(A)	*Crea un objeto simbólico a partir de A, donde A puede ser una cadena, una escalar, una matriz, una expresión numérica, etc.*
S = sym(A,'opcion')	*Convierte la matriz, escalar o expresión numérica A a simbólicos según la opción especificada. La opción puede ser 'f' para punto flotante, 'r' para racional, 'e' para formato de error y 'd' para decimal*
numeric(x) ó double(x)	*Convierte la variable o expresión x a numérica de doble precisión*
sym2poly(poli)	*Convierte el polinomio simbólico poli en un vector cuyas componentes son sus coeficientes*
poly2sym(vector)	*Convierte el vector en un polinomio simbólico cuyos coeficientes son las componentes del vector*
poly2sym(vector,'v')	*Convierte el vector en un polinomio simbólico en la variable v cuyos coeficientes son las componentes del vector*
char(X)	*Convierte el array X de enteros ASCII a su valor como cadena*
latex(S)	*Convierte a código Latex la expresión simbólica S*
ccode(S)	*Convierte a código C la expresión simbólica S*
fortran(S)	*Convierte a código Fortran la expresión simbólica S*
pretty(expr)	*Convierte la expresión simbólica a escritura matemática*
digits(d)	*Sitúa la precisión de las variables simbólicas en d dígitos decimales exactos*
digits	*Da la precisión actual para variables simbólicas*
vpa(expr)	*Resultado numérico de la expresión con los dígitos decimales de precisión situados en digits*
vpa(expr, n)	*Resultado numérico de la expresión con n dígitos decimales*
vpa('expr', n)	*Resultado numérico de la expresión con n dígitos decimales*
findsym(S)	*Devuelve todas las variables simbólicas en la expresión simbólica o matriz simbólica S*
isvarname(S)	*Devuelve TRUE si S es una variable simbólica válida*
vectorize(S)	*Inserta un punto en la cadena S antes de cualquier símbolo ^,*o / con la finalidad de operar vectorialmente de forma correcta*

Como primer ejemplo consideramos $H = 3\,a^2 - 2\,a + 7$, $F = 6\,a^3 - 5\,a + 2$ y $G = 5\,a^2 + 4\,a - 3$, y calculamos: $H + F + G$, $H - F + G$ y $H - F - G$.

```
>> syms a
>> H = 3*a^2 - 2*a + 7; F = 6*a^3 - 5*a + 2; G = 5*a^2 + 4*a - 3;
>> pretty(H+F+G)
                                    2                3
                                 8 a  - 3 a + 6 + 6 a
>> pretty(H-F+G)

                                    2                3
                                 8 a  + 7 a + 2 - 6 a
>> pretty(H-F-G)

                                    2                3
                                 -2 a  - a + 8 - 6 a
```

En el ejemplo siguiente realizamos las operaciones racionales simbólicas siguientes:

$$\frac{1}{2a}+\frac{1}{3b}+\frac{1}{4a}+\frac{1}{5b}+\frac{1}{6c},\ \frac{1-a^9}{1-a^3}\ \text{y}\ \frac{1}{1+a}+\frac{1}{(1+a)^2}+\frac{1}{(1+a)^3}$$

Comenzaremos definiendo las variables *a*, *b* y *c* como simbólicas y posteriormente realizamos las operaciones especificadas.

```
>> syms a b c
>> pretty(1/(2*a)+1/(3*b)+1/(4*a)+1/(5*b)+1/(6*c))

                          3/4 1/a + 8/15 1/b + 1/6 1/c

>> pretty((1-a)^9/(1-a)^3)

                                        6
                                  (1 - a)

>> pretty(1/(1+a)+1/(1+a)^2+1/(1+a)^3)

                            1         1          1
                          ----- + -------- + --------
                          1 + a          2          3
                                  (1 + a)    (1 + a)

>> pretty(simplify(1/(1+a)+1/(1+a)^2+1/(1+a)^3))
```

```
                    2
  3 + 3 a + a
  -----------
             3
    (1 + a)
```

A continuación calculamos $\sqrt{2}+3\sqrt{2}-\sqrt{2}/2$ y $\frac{1}{1+\sqrt{2}}+\frac{1}{1-\sqrt{2}}$ de forma simbólica.

```
>> pretty(sym(sqrt(2)+3*sqrt(2)-sqrt(2)/2))

                                      1/2
                                7/2 2
```

```
>> pretty(sym(1/(1+sqrt(2)) + 1/(1-sqrt(2))))

                                   -2
```

En el ejemplo siguiente calculamos $\frac{1}{2+\sqrt{5}}$ de forma simbólica y pasamos el resultado a forma numérica.

```
>> r=sym(1/(2+sqrt(5)))

r =

8505245244017276*2^(-55)

>> numeric(r)

ans =

          0.24
```

A continuación resolvemos la ecuación $x^4+1 = 0$ y presentamos el resultado en escritura matemática habitual.

```
>> solve('x^4+1=0')

ans =

[  1/2*2^(1/2)+1/2*i*2^(1/2)]
[ -1/2*2^(1/2)-1/2*i*2^(1/2)]
[  1/2*2^(1/2)-1/2*i*2^(1/2)]
[ -1/2*2^(1/2)+1/2*i*2^(1/2)]
```

```
>> pretty(solve('x^4+1=0'))
```

```
[       1/2           1/2 ]
[  1/2 2    + 1/2 i 2     ]
[                        ]
[        1/2           1/2]
[- 1/2 2    - 1/2 i 2    ]
[                        ]
[       1/2           1/2 ]
[  1/2 2    - 1/2 i 2     ]
[                        ]
[        1/2           1/2]
[- 1/2 2    + 1/2 i 2    ]
```

A continuación transformamos un vector a polinomio y viceversa.

```
>> pretty(poly2sym([1 0 9 6 2]))
```

```
   4      2
  x  + 9 x  + 6 x + 2
```

```
>> sym2poly(x^4+9*x^2+6*x+2)

ans =

   1.00            0         9.00         6.00         2.00
```

A continuación presentamos la matriz de Hilbert de orden 2 con cinco dígitos decimales.

```
>> vpa(hilb(2),5)

ans =

[     1., .50000]
[ .50000, .33333]
```

En el ejemplo siguiente definimos una matriz simbólica y calculamos su determinante.

```
>> syms a x
A = [cos(a*x),sin(a*x);-sin(a*x),cos(a*x)]

 A =

[  cos(a*x),  sin(a*x)]
[ -sin(a*x),  cos(a*x)]
```

```
 >> det(A)

ans =

cos(a*x)^2+sin(a*x)^2
```

A continuación definimos la matriz simbólica anterior de una forma alternativa y calculamos su inversa.

```
>> A=sym([cos(a*x),sin(a*x);-sin(a*x),cos(a*x)])

A =

[  cos(a*x),  sin(a*x)]
[ -sin(a*x),  cos(a*x)]

>> pretty(inv(A))
```

```
                          [cos(a x)       sin(a x)]
                          [--------     - --------]
                          [   %1             %1   ]
                          [                       ]
                          [sin(a x)      cos(a x) ]
                          [--------      -------- ]
                          [   %1            %1    ]

                                        2           2
                           %1 := cos(a x)  + sin(a x)
```

En el ejemplo siguiente calculamos 1/2+1/3 de forma simbólica y posteriormente situamos la precisión a 25 dígitos y calculamos el valor numérico de la operación anterior, para finalizar comprobando el nivel de precisión numérica actual.

```
>> sym(1/2+1/3)

ans =

5/6

>> digits(25)
vpa('1/2+1/3')

ans =

.83333333333333333333333

>> digits

Digits = 25
```

En el ejemplo siguiente se obtienen los caracteres ASCII cuyos códigos numéricos son 93, 126 y 115

```
>> char(93,126,115)

ans =

]
~
s
```

En el ejemplo siguiente se transforma a código Latex, a código C y a código Fortan el desarrollo en serie de la función *log*(1+*x*).

```
>> syms x
>> f = taylor(log(1+x));
>> latex(f)

ans =

x-1/2\,{x}^{2}+1/3\,{x}^{3}-1/4\,{x}^{4}+1/5\,{x}^{5}

>> ccode(f)

ans =

      t0 = x-x*x/2.0+x*x*x/3.0-x*x*x*x/4.0+x*x*x*x*x/5.0;

>> fortran(f)

ans =

      t0 = x-x**2/2+x**3/3-x**4/4+x**5/5
```

10.2 Funciones simbólicas. Sustitución y operaciones funcionales

El módulo de matemática simbólica de MATLAB permite definir funciones simbólicas directamente mediante la sintaxis *f='función'* (o *f=función* si las variables se han definido previamente como simbólicas con *syms*).

Una vez definida explícitamente una función simbólica, es posible sustituir valores de las variables en la función, es decir, calcular el valor de la función en un punto dado, mediante los comandos que se indican a continuación:

subs(f,a)	*Aplica la función f en el punto a*
subs(f,a,b)	*Sustituye en f el valor de a por el valor de b*
subs(f, variable, valor)	*Sustituye en la ecuación de f la variable por el valor*
subs(f, {x,y,…}, {a,b,…})	*Sustituye en la ecuación de f las variables {x,y,...} por los valores {a,b,...}*

Como primer ejemplo definimos la función $f(x) = x^3$ y calculamos $f(2)$ y $f(b+2)$.

```
>> f='x^3'

f =

x^3

>> A=subs(f,2)

A =

     8

>> syms b
>> B=subs(f,b+2)

B =

(b+2)^3
```

En el ejemplo siguiente consideramos la función de dos variables $f(a,b) = a+b$ y sustituimos primeramente a por 4, y posteriormente a por 3 y b por 5 ($f(3,5)$).

```
>> syms a b
>> subs(a+b,a,4)

ans =

4+b

>> subs(a+b,{a,b},{3,5})

ans =

     8
```

A continuación se presentan tres ejemplos adicionales sobre sustituciones.

```
>> subs(cos(a)+sin(b),{a,b},{sym('alpha'),2})
```

```
ans =

cos(alpha)+sin(2)

>> subs('exp(a*t)','a',-magic(2))

ans =

[   exp(-t), exp(-3*t)]
[ exp(-4*t), exp(-2*t)]

>> syms x y
>> subs(x*y,{x,y},{[0 1;-1 0],[1 -1;-2 1]})

ans =

     0    -1
     2     0
```

Además de la sustitución, MATLAB aporta comandos que permiten las operaciones funcionales, como la suma, resta, multiplicación y división de funciones, además de la composición y la inversión de funciones. A continuación se presenta la sintaxis de estos comandos:

symadd(f,g)	*Suma las funciones f y g (f+g)*
symop(f, '+',g, '+',h, '+',.....)	*Realiza la suma f+g+h+....*
symsub(f,g)	*Realiza la diferencia de f y g (f-g)*
symop(f, '-',g, '-',h,'-',.....)	*Realiza la diferencia f-g-h-...*
symmul(f,g)	*Realiza el producto de f y g (f*g)*
symop(f, ' * ',g, ' * ',h,' * ',.....)	*Realiza el producto f*g*h*...*
symdiv(f,g)	*Realiza el cociente entre f y g (f/g)*
symop(f, '/',g, '/ ',h, '/ ',.....)	*Realiza el cociente f/g/h/...*
sympow(f,k)	*Eleva f a la potencia k (k es un escalar)*
symop(f, '^',g)	*Eleva una función a otra función (f^g)*
compose(f,g)	*Función compuesta de f y g ($f°g(x) = f(g(x))$)*
compose(f,g,u)	*Función compuesta de f y g, tomando la expresión u como dominio de f y g*
g = finverse(f)	*Función inversa de f*
g = finverse(f,v)	*Función inversa de f usando la variable simbólica v como variable independiente*

En el ejemplo siguiente, dadas las funciones $f(x)=x^2+x$, $g(x)=x^3+1$ y $h(x) = Sen(x)+Cos(x)$, calcularemos $(f+g)(x)$, $(f-g+h)(x)$, $(f/g)(x)$, $f(g(x))$, $f(h(\pi/3))$ y $f(g(h(Sen(x))))$.

```
>> syms x
>> f=x^2;g=x^3+1;h=sin(x)+cos(x);
>> suma=symadd(f,g)

suma =

x^2+x^3+1

>> combinada=symadd(symsub(f,g),h)

combinada =

x^2-x^3-1+sin(x)+cos(x)

>> combinada=symop(f,'-',g,'+',h)

combinada =

x^2-x^3-1+sin(x)+cos(x)

>> cociente=symdiv(f,g)

cociente =

x^2/(x^3+1)

>> compuesta=subs(compose(g,f),x-1)

compuesta =

(x-1)^6+1

>> compuesta1=subs(compose(f,h),pi/3)

compuesta1 =

    1.8660

>> compuesta1=subs(compose(f,h),'pi/3')

compuesta1 =

(sin((pi/3))+cos((pi/3)))^2

>> compuesta2=subs(compose(f,compose(g,h)),'sin(x)')

compuesta2 =

((sin(sin(x))+cos(sin(x)))^3+1)^2
```

En el ejemplo siguiente se halla la función inversa de $f(x) = Sen(x^2)$ y se comprueba que el resultado es correcto.

```
>> syms x
>> f=sin(x^2)

f =

sin(x^2)

>> g=finverse(f)

g =

asin(x)^(1/2)

>> compose(f,g)

ans =

x
```

MATLAB también aporta un grupo de funciones simbólicas predefinidas especiales, cuya sintaxis se presenta en el cuadro siguiente:

cosint(*x*)	*Coseno integral,* $Ci(x) = \gamma + Ln(x) + \int_0^x \frac{Cos(t)-1}{t} dt \quad \gamma=0.57...$
sinint(*x*)	*Seno integral,* $Si(x) = \int_0^x \frac{Sen(t)}{t} dt$
hypergeom(*n,d,z*)	*Función hipergeométrica generalizada*
lambertw(*x*)	*Resuelve la ecuación* $\lambda(x)e^{\lambda(x)} = x$
zeta(*x*)	*Z de Riemann, definida como* $Z(x) = \sum_{k=1}^{\infty} \frac{1}{k^x}$
zeta(*n,x*)	*Derivada enésima de zeta(x)*

Como primer ejemplo sumamos la serie $\sum_{k=1}^{\infty} \frac{1}{k^4}$, cuyo valor será la $Z(4)$.

```
>> zeta(4)

ans =

    1.0823
```

A continuación resolvemos la integral $\int_0^2 \frac{Sen(t)}{t} dt$. Para ello utilizamos la función seno integral.

```
>> sinint(2)
ans =
    1.6054
```

10.3 Funciones de análisis matemático. Límites, continuidad y series

MATLAB ofrece funciones en el módulo de matemática simbólica que permite trabajar en análisis matemático de forma fluida. Es posible calcular límites, obtener derivadas, sumar series, desarrollar funciones en series de Taylor, calcular integrales y trabajar con ecuaciones diferenciales.

En cuanto al cálculo de límites y al trabajo con series numéricas, las mismas funciones se aplican para calcular límites de sucesiones, para calcular límites de funciones y para calcular límites de sucesiones de funciones, y por supuesto, para analizar la continuidad de funciones y la convergencia de series numéricas y series de potencias. El análisis para una y varias variables es semejante. Se dispone de las funciones siguientes.

limit(sucesión, n, inf)	*Calcula el límite de la sucesión, indicada por su término general, cuando n tiende a infinito*
limit(sucesión, inf)	*Calcula el límite de la sucesión, indicada por su término general, cuando n tiende a infinito*
limit(función, x, a)	*Calcula el límite de la función de variable x, indicada por su expresión analítica, cuando la variable x tiende hacia el valor a*
limit(función, a)	*Calcula el límite de la función de variable x, indicada por su expresión analítica, cuando la variable x tiende hacia el valor a*
limit(función,x,a,'right')	*Calcula el límite de la función de variable x, indicada por su expresión analítica, cuando la variable x tiende al valor a por la derecha*
limit(función,x,a,'left')	*Calcula el límite de la función de variable x, indicada por su expresión analítica, cuando la variable x tiende al valor a por la izquierda*
symsum(S,v,a,b)	*Suma la serie S para la variable v variando entre a y b*
symsum(S,v)	*Suma la serie S en la variable v variando de 0 a v-1*
r = symsum(S)	*Suma la serie S respecto de su variable simbólica k (determinada por findsym) desde 0 hasta k-1*
symsum(S, a,b)	*Suma la serie S respecto de su variable simbólica k (determinada por findsym) variando entre a y b*

Como primer ejemplo calculamos los límites de las sucesiones siguientes:

$$\lim_{n\to\infty}\left(\frac{-3+2n}{-7+3n}\right)^4, \quad \lim_{n\to\infty}\frac{1+7n^2+3n^3}{5-8n+4n^3}, \quad \lim_{n\to\infty}\left[\left(\frac{1+n}{2}\right)^4\frac{1+n}{n^5}\right], \quad \lim_{n\to\infty}\sqrt[n]{\frac{1+n}{n^2}}$$

Tenemos :

```
>> syms n
>> limit(((2*n-3)/(3*n-7))^4, inf)

ans =

16/81

>> limit((3*n^3+7*n^2+1)/(4*n^3-8*n+5),n,inf)

ans =

3/4

>> limit(((n+1)/2)*((n^4+1)/n^5), inf)

ans =

1/2

>> limit(((n+1)/n^2)^(1/n), inf)

ans =

1
```

A continuación calculamos los límites de las funciones siguientes:

$$\lim_{x\to 1}\frac{-1+x}{-1+\sqrt{x}}, \quad \lim_{x\to 2}\frac{x-\sqrt{2+x}}{-3+\sqrt{1+4x}}, \quad \lim_{x\to 0}\sqrt[x]{1+x}, \quad \lim_{x\to 0}\frac{Sen\left[(ax)^2\right]}{x^2}$$

Tenemos:

```
>> syms x a
>> limit((x-1)/(x^(1/2)-1),x,1)

ans =

2
```

```
 >> limit((x-(x+2)^(1/2))/((4*x+1)^(1/2)-3),2)
ans =
9/8
>> limit((1+x)^(1/x))
ans =
exp(1)
>> limit(sin(a*x)^2/x^2,x,0)
ans =
a^2
```

En el ejemplo siguiente calculamos la función límite de la sucesión de funciones $g_n(x) = (x^2+nx)/n$ con $x \in R$.

```
>> limit((x^2+n*x)/n,n,inf)
ans =
x
```

Hemos obtenido que la función límite es la diagonal del primer, tercer cuadrante, resultado que puede comprobarse gráficamente (Figura 10-1) como sigue:

```
>> fplot('[(x^2+x),(x^2+2*x)/2,(x^2+3*x)/3,(x^2+4*x)/4,
(x^2+5*x)/5,(x^2+5*x)/5,(x^2+6*x)/6,(x^2+7*x)/7,(x^2+8*x)/8,
(x^2+9*x)/9]',[-2,2,-2,2])
```

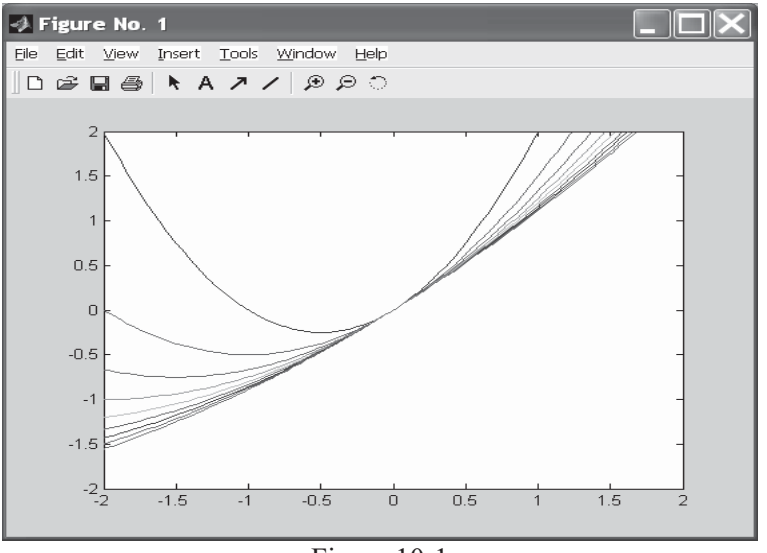


Figura 10-1

En el ejemplo siguiente se comprueba la continuidad en R-{0} de la función $f(x) = Sen(x)/x$. Para ello comprobaremos que $\lim_{x\to a} f(x) = f(a)$.

```
>> syms x a
>> limit(sin(x)/x,x,a)

ans =

sin(a)/a
```

A continuación se comprueba que la función $f(x) = \sqrt[x]{e}$ no es continua en el punto $x = 0$ ya que los límites laterales no coinciden (uno es cero y el otro infinito).

```
>> syms x
>> limit((exp(1/x)),x,0,'right')

ans =

inf

>> limit((exp(1/x)),x,0, 'left')

ans =

0
```

En el ejemplo siguiente comprobamos si la serie numérica $\sum_{n=1}^{\infty} \frac{n}{2^n}$ es convergente aplicando el criterio del cociente $(\lim_{n\to\infty} \frac{a(n+1)}{a(n)} < 1)$ y en caso afirmativo calculamos su suma.

```
>> syms n
>> f=n/2^n

f =

n/(2^n)

>> limit(subs(f,n,n+1)/f,n,inf)

ans =

1/2
```

Se observa que el límite es menor que la unidad, con lo que la serie converge y es sumable. Calculamos su suma de la siguiente forma:

```
>> symsum(f,n,1,inf)

ans =

2
```

10.4 Derivadas, integrales y ecuaciones diferenciales

Siguiendo con las funciones de MATLAB para el trabajo en análisis matemático, a continuación se presentan las relativas a derivadas, integrales y ecuaciones diferenciales. Comenzaremos con las funciones relacionadas con la derivación.

diff('f', 'x')	*Halla la función derivada de f respecto a x*
syms x, diff(f,x)	*Halla la función derivada de f respecto a x*
diff('f', 'x', n)	*Halla la función derivada enésima de f respecto a x*
syms x,diff(f, x, n)	*Halla la función derivada enésima de f respecto a x*
r = taylor(f,n,v)	*Halla el desarrollo de MacLaurin de orden n-1 para la función f en la variable v*
r = taylor(f)	*Halla el desarrollo de MacLaurin de orden 5 para la función f en la variable por defecto*
r = taylor(f,n,v,a)	*Halla el desarrollo de Taylor de orden n-1 para la función f en la variable v en un entorno del punto a*
R = jacobian(w,v)	*Halla la matriz jacobiana de w respecto de v*

A continuación se presentan las funciones relacionadas con la integración:

syms x, int(f(x), x) o int('f(x)', 'x')	*Calcula la integral indefinida* $\int f(x)dx$
int(int('f(x,y)', 'x'), 'y'))	*Calcula la integral doble* $\int\int f(x,y)dxdy$
syms x y, int(int(f(x,y), x), y))	*Calcula la integral doble* $\int\int f(x,y)dxdy$
int(int(int(...int('f(x,y...z)', 'x'), 'y')...), 'z')	*calcula* $\int\int\cdots\int f(x,y,\cdots,z)\,dxdy\cdots dz$
syms x y z, int(int(int(...int(f(x,y...z), x), y)...), z)	*Calcula* $\int\int\cdots\int f(x,y,\cdots,z)\,dxdy\cdots dz$

syms x a b, int(f(x), x, a, b)	*Calcula la integral definida* $\int_a^b f(x)dx$
int('f(x)', 'x', 'a', 'b')	*Calcula la integral definida* $\int_a^b f(x)dx$
int(int('f(x,y)', 'x', 'a', 'b'), 'y', 'c', 'd'))	*Calcula la integral* $\int_a^b \int_c^d f(x,y)dxdy$
syms x y a b c d, int(int(f(x,y), x, a, b), y, c, d))	*Calcula* $\int_a^b \int_c^d f(x,y)dxdy$
int(int(int(....int('f(x,y,...,z)', 'x', 'a', 'b'), 'y', 'c', 'd'),...), 'z', 'e', 'f')	*Halla* $\int_a^b \int_c^d \cdots \int_e^f f(x,y,\cdots,z)\,dxdy\cdots dz$
syms x y z a b c d e f, int(int(int(....int(f(x,y,...,z), x, a, b), y, c, d),...), z, e, f)	*Halla* $\int_a^b \int_c^d \cdots \int_e^f f(x,y,\cdots,z)\,dxdy\cdots dz$

El cuadro siguiente resume las funciones relacionadas con las ecuaciones diferenciales:

dsolve('e', 'v')	*Resuelve la ecuación diferencial e siendo v la variable independiente (si no se especifica 'v', la variable independiente es x). Sólo devuelve soluciones explícitas*
dsolve('e', 'c',... , 'v')	*Resuelve la ecuación diferencial e sujeta a la condición inicial especificada c*
dsolve('e','c1','c2',...,'cn','v')	*Resuelve la ecuación diferencial e sujeta a las condiciones iniciales especificadas* c_i
dsolve('e','c1,c2,...,cn','v')	*Resuelve la ecuación diferencial sujeta a las condiciones iniciales especificadas*
dsolve('e1', 'e2',..., 'en', 'c1', 'c2',..., 'cn' , 'v')	*Resuelve el sistema diferencial sujeto a las condiciones iniciales especificadas (explícitas)*
dsolve('e1, e2,..., en', 'c1, c2,..., cn' , 'v')	*Resuelve el sistema diferencial sujeto a las condiciones iniciales especificadas*

Como primer ejemplo calculamos la derivada de la función *Log*(*Sen*(2*x*)).

```
>> pretty(diff('log(sin(2*x))','x'))
```

```
    cos(2 x)
2   ---------
    sin(2 x)
```

A continuación se obtiene la derivada totalmente simplificada:

```
>> pretty(simple(diff('log(sin(2*x))','x')))

     2
--------
tan(2 x)
```

En el ejemplo siguiente calculamos las cuatro primeras derivadas de la función $f(x) = 1/x$

```
>> f='1/x';
 [diff(f),diff(f,2),diff(f,3),diff(f,4),diff(f,5)]

ans =

[   -1/x^2,    2/x^3,   -6/x^4,   24/x^5, -120/x^6]
```

A continuación, dada la función $f(x,y) = Sen(xy)+Cos(xy^2)$, se calcula:

$\partial f/\partial x$, $\partial f/\partial y$, $\partial^2 f/\partial x^2$, $\partial^2 f/\partial y^2$, $\partial^2 f/\partial x\partial y$, $\partial^2 f/\partial y\partial x$ y $\partial^4 f/\partial^2 x\partial^2 y$

```
>> syms x y
>> f=sin(x*y)+cos(x*y^2)

f =

sin(x*y)+cos(x*y^2)

>> diff(f,x)

ans =

cos(x*y)*y-sin(x*y^2)*y^2

>> diff(f,y)

ans =

cos(x*y)*x-2*sin(x*y^2)*x*y

>> diff(diff(f,x),x)

ans =

-sin(x*y)*y^2-cos(x*y^2)*y^4

>> diff(diff(f,y),y)
```

```
ans =

-sin(x*y)*x^2-4*cos(x*y^2)*x^2*y^2-2*sin(x*y^2)*x

>> diff(diff(f,x),y)

ans =

-sin(x*y)*x*y+cos(x*y)-2*cos(x*y^2)*x*y^3-2*sin(x*y^2)*y

>> diff(diff(f,y),x)

ans =

-sin(x*y)*x*y+cos(x*y)-2*cos(x*y^2)*x*y^3-2*sin(x*y^2)*y

>> diff(diff(diff(diff(f,x),x),y,y))

ans =

sin(x*y)*y^3*x-
3*cos(x*y)*y^2+2*cos(x*y^2)*y^7*x+6*sin(x*y^2)*y^5
```

A continuación se realiza el desarrollo en serie de Taylor de grado 10 para la función $1/(2-x)$ en un entorno del punto $x=1$

```
>> syms x
>> f=1/(2-x)

f =
1/(2-x)

>> pretty(taylor(f,11,x,1))

             2           3           4           5           6           7
  x + (x - 1)  + (x - 1)  + (x - 1)  + (x - 1)  + (x - 1)  + (x - 1)

                   8           9           10
         + (x - 1)  + (x - 1)  + (x - 1)
```

En el ejemplo siguiente se calcula la integral $\int \frac{1}{x^2-1} dx$.

```
>> int('1/(x^2-1)','x')
ans =
-atanh(x)
```

En el ejemplo siguiente se calcula la integral $\int aLn(1-bx)dx$, siendo a y b parámetros.

```
>> syms x a b, pretty(simple(int(a*log(1+b*x),x)))

                a (log(1 + b x) - 1) (1 + b x)
                ------------------------------
                              b
```

En el ejemplo siguiente se calcula la integral doble $\iint aLn(1-bx)dxdb$, siendo a un parámetro.

```
>> syms x a b, pretty(simple(int(int(a*log(1+b*x),x),b)))

a (-dilog(1 + b x) + log(1 + b x) + log(1 + b x) x b - 1 - 2 b x - log(b))
```

En el ejemplo siguiente se calcula la integral triple $\iiint aLn(1-bx)dxdbda$.

```
>> syms x a b, pretty(simple(int(int(int(a*log(1+b*x),x),b),a)))

    2
1/2a(-dilog(1 + b x) + log(1 + b x) + log(1 + b x) x b - 1 - 2 b x - log(b))
```

A continuación calculamos $\int_0^1 aLn(1-bx)dx$.

```
>> syms x a b, pretty(simple(int(a*log(1+b*x),x,0,1)))

                a log(1 + b)
                ------------ - a + a log(1 + b)
                     b
```

En el ejemplo siguiente se calcula $\int_0^1 \int_2^3 aLn(1-bx)dxdb$.

```
>> syms x a b, pretty(simple(int(int(a*log(1+b*x),x,0,1),b,2,3)))

          (-2 + 8 log(2) - dilog(4) - 3 log(3) + dilog(3)) a
```

En el ejemplo siguiente se resuelve la ecuación diferencial de primer orden y primer grado $y'(t) = ay(t)$ con a = parámetro.

```
>> pretty(dsolve('Dy = a*y'))

                                C1 exp(a t)
```

La familia de soluciones resulta ser $y(t) = c_1 e^{at}$.

A continuación resolvemos la ecuación diferencial anterior con la condición inicial $y(0) = b$.

```
>> pretty(dsolve('Dy = a*y', 'y(0) = b'))

                                b exp(a t)
```

A continuación resolvemos la ecuación diferencial de segundo grado y primer orden $y'^2(s) + y^2(s) = 1$ con la condición inicial $y(0) = 0$.

```
>> y = dsolve('(Dy)^2 + y^2 = 1', 'y(0) = 0', 's')

y =

[ -sin(s)]
[  sin(s)]
```

A continuación resolvemos la ecuación diferencial de segundo orden y primer grado $y''(t) = -a^2 y'(t)$ con las condiciones iniciales $y(0) = 1$ e $y'(\pi/a) = 0$.

```
>> pretty(dsolve('D2y = -a^2*y', 'y(0) = 1, Dy(pi/a) = 0'))

                                cos(a t)
```

Por lo tanto, la solución es la función $y(t) = Cos(at)$.

En el ejemplo siguiente se resuelve el sistema: $x'(t) = y(t)$, $y'(t) = -x(t)$.

```
>> [x,y]=dsolve('Dx = y', 'Dy = -x')

x =

cos(t)*C1+sin(t)*C2


y =

-sin(t)*C1+cos(t)*C2
```

A continuación calculamos la solución del sistema de ecuaciones diferenciales anterior para las condiciones iniciales $x(0) = 0$ e $y(0) = 1$.

```
>> [x,y]=dsolve('Dx = y, Dy = -x', 'x(0)=0, y(0)=1')

x =

sin(t)


y =

cos(t)
```

10.5 Algebra lineal: simplificación y resolución de ecuaciones

Los cálculos con expresiones algebraicas simples, racionales y complejas son tratados especialmente en MATLAB. Las funciones del toolbox *Symbolic Match* ejecutan rápida y eficientemente todas las operaciones de simplificación, factorización, agrupación y expansión de expresiones algebraicas complicadas, incluyendo expresiones trigonométricas y expresiones en variable compleja. A continuación se presenta la sintaxis de estas funciones.

R = collect(S) **R = collect(S,v)**	*Para cada polinomio en el array de polinomios S se agrupan términos en la variable v (o en x si v no se especifica)*
R = expand(S)	*Expande los polinomios o funciones trigonométricas, exponenciales y logarítmicas contenidas en S.*
factor(X)	*Factoriza cada elemento (simbólico o numérico) de X*
R = horner(P)	*Representación polinomial anidada de Horner para el polinomio P*
[N,D] = numden(A)	*Convierte cada elemento de la matriz simbólica o numérica A a forma racional totalmente simplificada*
r = simple(S) **[r,how] = simple(S)**	*Simplifica la expresión simbólica S buscando el resultado más corto posible. La segunda opción presenta sólo el resultado final y una cadena describiendo la simplificación particular*
R = simplify(S)	*Simplifica cada elemento de la matriz simbólica S*
[Y,SIGMA] = subexpr(X,SIGMA) **[Y,SIGMA] = subexpr(X,'SIGMA')**	*Reescribe la expression simbólica X en términos de subexpresiones comunes.*

Como primer ejemplo agrupamos en *sin*(*x*) la expresión *y*(*sin*(*x*)+1)+*sin*(*x*).

```
>> syms x y
>> pretty(collect(y*(sin(x)+1)+sin(x),sin(x)))

                        (y + 1) sin(x) + y
```

A continuación agrupamos, primeramente en *x,* y luego en *Ln*(*x*), la función *f*(*x*) = *aLn*(*x*)-*xLn*(*x*)-*x*.

```
>> syms a x
>> f=a*log(x)-log(x)*x-x

f =

a*log(x)-log(x)*x-x

>> pretty(collect(f,x))

                    (-log(x) - 1) x + a log(x)

>> pretty(collect(f,log(x)))

                         (a - x) log(x) - x
```

En el ejemplo siguiente se expanden varias expresiones algebraicas.

```
>> syms a b x y t
>> expand([sin(2*t), cos(2*t)])

ans =

[ 2*sin(t)*cos(t),    2*cos(t)^2-1]

>> expand(exp((a+b)^2))

ans =

exp(a^2)*exp(a*b)^2*exp(b^2)

>> expand(cos(x+y))

ans =

cos(x)*cos(y)-sin(x)*sin(y)
```

```
>> expand((x-2)*(x-4))

ans =

x^2-6*x+8
```

A continuación se factorizan varias expresiones.

```
>> factor(x^3-y^3)

ans =

(x-y)*(x^2+x*y+y^2)

>> factor([a^2-b^2, a^3+b^3])

ans =

[          (a-b)*(a+b), (a+b)*(a^2-a*b+b^2)]

>> factor(sym('123456789012345678901'))

ans =

(2)*(3)^2*(5)*(101)*(3803)*(3607)*(27961)*(3541)
```

A continuación se simplifican varias expresiones.

```
>> syms x y z a b c
>> simplify(exp(c*log(sqrt(a+b))))

ans =

(a+b)^(1/2*c)

>> simplify(sin(x)^2 + cos(x)^2)

ans =

1

>> S = [(x^2+5*x+6)/(x+2),sqrt(16)];
R = simplify(S)

 R =

[ x+3,   4]
```

En cuanto a la resolución de ecuaciones y sistemas simbólicos, pueden utilizarse las funciones siguientes:

solve('ecuación', 'x')	*Resuelve la ecuación en la variable x*
syms x; solve(ecuación,x)	*Resuelve la ecuación en la variable x.*
sove('e1,e2,...,en', 'x1,x2,...,xn')	*Resuelve el sistema de ecuaciones e1,...,en siendo las variables x1,...,xn*
syms x1 x2...xn; solve(e1,e2,...,en, x1,x2,...,xn)	*Resuelve el sistema de ecuaciones e1,...,en siendo las variables x1,...,xn*

Como primer ejemplo resolvemos la ecuación en x $3ax-7x^2+x^3=0$ siendo a un parámetro.

```
>> solve('3*a*x-7*x^2+x^3=0','x')

ans =

[                          0]
[ 7/2+1/2*(49-12*a)^(1/2)]
[ 7/2-1/2*(49-12*a)^(1/2)]
```

A continuación resolvemos la ecuación anterior siendo a la variable y x el parámetro.

```
>> pretty(solve('3*a*x-7*x^2+x^3=0','a'))

                           - 1/3 x (-7 + x)
```

En el ejemplo siguiente calculamos las raíces cuartas de -1 y de 1.

```
>> S=solve('x^4+1=0')

S =

[  1/2*2^(1/2)+1/2*i*2^(1/2)]
[ -1/2*2^(1/2)-1/2*i*2^(1/2)]
[  1/2*2^(1/2)-1/2*i*2^(1/2)]
[ -1/2*2^(1/2)+1/2*i*2^(1/2)]

>> numeric(S)

ans =

  0.70710678118655 + 0.70710678118655i
 -0.70710678118655 - 0.70710678118655i
  0.70710678118655 - 0.70710678118655i
 -0.70710678118655 + 0.70710678118655i
```

```
>> S1=solve('x^4-1=0')

S1 =

[  1]
[ -1]
[  i]
[ -i]
```

A continuación calculamos las raíces quintas del número complejo 2+2*i*.

```
>> numeric(solve('x^5-(2+2*i)=0'))

ans =

  1.21598698264961 + 0.19259341768888i
  0.19259341768888 + 1.21598698264961i
 -1.09695770450838 + 0.55892786746601i
 -0.87055056329612 - 0.87055056329612i
  0.55892786746601 - 1.09695770450838i
```

A continuación resolvemos en la variable *x* la ecuación:

```
>> simple(solve('sin(x)*cos(x)=a','x'))

ans =

[   atan(1/2*(1+2*a)^(1/2)-1/2*(1-2*a)^(1/2),1/2*(1+2*a)^(1/2)+1/2*(1-2*a)^(1/2))]
[   atan(1/2*(1+2*a)^(1/2)+1/2*(1-2*a)^(1/2),1/2*(1+2*a)^(1/2)-1/2*(1-2*a)^(1/2))]
[ atan(-1/2*(1+2*a)^(1/2)-1/2*(1-2*a)^(1/2),-1/2*(1+2*a)^(1/2)+1/2*(1-2*a)^(1/2))]
[ atan(-1/2*(1+2*a)^(1/2)+1/2*(1-2*a)^(1/2),-1/2*(1+2*a)^(1/2)-1/2*(1-2*a)^(1/2))]
```

```
>> pretty(ans)

       [                  1/2                1/2
       [atan(1/2 (1 + 2 a)    - 1/2 (1 - 2 a)   ,

                   1/2                1/2 ]
              1/2 (1 + 2 a)    + 1/2 (1 - 2 a)   )]

       [                  1/2                1/2
       [atan(1/2 (1 + 2 a)    + 1/2 (1 - 2 a)   ,

                   1/2                1/2 ]
              1/2 (1 + 2 a)    - 1/2 (1 - 2 a)   )]

       [                    1/2                1/2
       [atan(- 1/2 (1 + 2 a)    - 1/2 (1 - 2 a)   ,

                     1/2                1/2 ]
       - 1/2 (1 + 2 a)    + 1/2 (1 - 2 a)   )]

       [                    1/2                1/2
       [atan(- 1/2 (1 + 2 a)    + 1/2 (1 - 2 a)   ,

                     1/2                1/2 ]
       - 1/2 (1 + 2 a)    - 1/2 (1 - 2 a)   )]
```

Si resolvemos la ecuación anterior para el caso particular a=0 tenemos:

```
>> solve('sin(x)*cos(x)=0','x')

ans =

[       0]
[  1/2*pi]
[ -1/2*pi]
```

En el ejemplo siguiente se trata de resolver el sistema $u+v+w=a$, $3u+v=b$, $u-2v-w=0$, siendo u, v y w las variables y a, b y c los parámetros.

```
>> syms u v w a b c
>> [u,v,w]=solve('u+v+w=a,3*u+v=b,u-2*v-w=c',u,v,w)

u =

1/5*b+1/5*a+1/5*c


v =

2/5*b-3/5*a-3/5*c


w =

-3/5*b+7/5*a+2/5*c
```

Ejercicio 10-1. Consideramos la matriz A simbólica siguiente:

$$\begin{bmatrix} a & b & c \\ 3c & a-3c & b \\ 3b & -3b+3c & a-3c \end{bmatrix}$$

Calcular A', A^{-1}, determinante(A), traza(A), rango(A) y A^2.

Comenzamos definiendo la matriz simbólica de nuestro problema como sigue:

```
>> A=sym('[a,b,c; 3*c,a-3*c,b; 3*b,-3*b+3*c,a-3*c]')

A =

[   a,       b,    c]
[ 3*c,   a-3*c,    b]
[ 3*b,-3*b+3*c,a-3*c]
```

Alternativamente, la misma matriz simbólica puede definirse declarando previamente todas sus variables como simbólicas de la forma siguiente:

```
>> syms a b c
>> A=sym([a,b,c; 3*c,a-3*c,b; 3*b,-3*b+3*c,a-3*c])

A =

[        a,         b,         c]
[      3*c,     a-3*c,         b]
[      3*b, -3*b+3*c,     a-3*c]

>> transpose(A)

ans =

[a,   3*c,       3*b]
[b, a-3*c, -3*b+3*c]
[c,     b,     a-3*c]

>> pretty(inv(A))
```

$$\begin{bmatrix} \dfrac{a^2 - 6ac + 9c^2 + 3b^2 - 3bc}{\%1} & \dfrac{ab - 3c^2}{\%1} & \dfrac{-b^2 + ac - 3c^2}{\%1} \\ \\ -3\,\dfrac{-b^2 + ac - 3c^2}{\%1} & \dfrac{a^2 - 3ac - 3bc}{\%1} & \dfrac{ab - 3c^2}{\%1} \\ \\ \dfrac{ab - 3c^2}{\%1} & \dfrac{ab - ac + b^2}{\%1} & \dfrac{a^2 - 3ac - 3bc}{\%1} \end{bmatrix}$$

$$\%1 := a^3 - 6ca^2 + 9c^2a + 3ab^2 - 9abc + 9c^3 + 3b^3 + 9bc^2$$

```
>> pretty(det(A))
```

$$a^3 - 6ca^2 + 9c^2a + 3ab^2 - 9abc + 9c^3 + 3b^3 + 9bc^2$$

```
>> pretty(trace (A))
```

$$3a - 6c$$

```
>> rank(A)

ans =

     3
```

```
>> A^2

ans =

[                    a^2+6*b*c,        a*b+b*(a-3*c)+c*(-3*b+3*c),                 a*c+b^2+c*(a-3*c)]
[ 3*a*c+3*c*(a-3*c)+3*b^2,          3*b*c+(a-3*c)^2+b*(-3*b+3*c),                 3*c^2+2*b*(a-3*c)]
[ 3*a*b+3*c*(-3*b+3*c)+3*b*(a-3*c),  3*b^2+2*(-3*b+3*c)*(a-3*c),     3*b*c+(a-3*c)^2+b*(-3*b+3*c)]
```

Ejercicio 10-2. Hallar la intersección de las hipérbolas de ecuaciones $x^2-y^2=1$ y $a^2x^2-b^2y^2=16$ con la parábola $z^2 = 2x$.

Se trata de resolver el sistema que forma las tres ecuaciones como sigue:

```
>> [x,y,z]=solve('a^2*x^2-b^2*y^2=16','x^2-y^2=1','z^2=2*x', 'x,y,z')

x =

[  1/2*(((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4))^2]
[  1/2*(((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4))^2]
[ 1/2*(-((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4))^2]
[ 1/2*(-((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4))^2]
[  1/2*(((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4))^2]
[  1/2*(((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4))^2]
[ 1/2*(-((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4))^2]
[ 1/2*(-((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4))^2]

y =

[  1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]
[ -1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]
[  1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]
[ -1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]
[  1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]
[ -1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]
[  1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]
[ -1/(a^2-b^2)*(-(a^2-b^2)*(a^2-16))^(1/2)]

z =

[  ((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4)]
[  ((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4)]
[ -((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4)]
[ -((b^2-16)/(a^2-b^2))^(1/4)+i*((b^2-16)/(a^2-b^2))^(1/4)]
[  ((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4)]
[  ((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4)]
[ -((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4)]
[ -((b^2-16)/(a^2-b^2))^(1/4)-i*((b^2-16)/(a^2-b^2))^(1/4)]
```

Ejercicio 10-3. Resolver las siguientes integrales siguientes:

$$\int_{-3}^{3} \frac{1}{3}\frac{Sen(2t)}{t}dt\,, \quad \int_{0}^{5} \frac{Cosh(x)-1}{x}dx$$

Para la primera integral la función integrando es par, luego, nuestra integral entre -3 y 3 será el doble de la integral entre 0 y 3. A continuación, hacemos el cambio de variable $2t=v$, y llegamos a la integral:

$$\int_{-3}^{3} \frac{1}{3}\frac{Sen(2t)}{t}dt = 2\int_{0}^{3} \frac{1}{3}\frac{Sen(2t)}{t}dt = \frac{2}{3}\int_{0}^{6} \frac{Sen(v)}{v}dv$$

cuya solución por MATLAB se halla como sigue:

```
>> (2/3)*(sinint(6))

ans =

    0.9498
```

Para calcular la segunda integral tenemos en cuanta lo siguiente:

$$Ci(x) = \gamma + Ln(x) + \int_{0}^{x} \frac{Cos(t)-1}{t}dt \Rightarrow \int_{0}^{5} \frac{Cos(x)-1}{x}dx = Ci(5) - \gamma + Ln(5)$$

que puede calcularse en MATLAB como sigue:

```
>> cosint(5)-0.577215664-log(5)

ans =

   -2.3767
```

Ejercicio 10-4. Dada la función h definida por: h(x,y)= (cos(x^2-y^2), sin(x^2-y^2)), calcular h(1,2), h(-Pi,Pi) y h(cos(a^2), cos(1-a^2)).

Creamos la función vectorial de dimensión 2 como sigue:

```
>> syms x y a
>> h=[cos(x^2-y^2),sin(x^2-y^2)]

h =

[ cos(x^2-y^2), sin(x^2-y^2)]
```

Ahora calculamos los valores pedidos:

```
>> subs(h,{x,y},{1,2})

ans =

   -0.9900   -0.1411

>> subs(h,{x,y},{-pi,pi})

ans =

     1     0

>> subs(h,{x,y},{cos(a^2),cos(1-a^2)})

ans =

[ cos(cos(a^2)^2-cos(-1+a^2)^2), sin(cos(a^2)^2-cos(-1+a^2)^2)]
```

Ejercicio 10-5. Dada la función f definida por:

$$f(x,y)= 3(1-x)^2\, e^{-(y+1)^2-x^2} -10(x/5-x^3-y/5)e^{-x^2-y^2}-1/3e^{-(x+1)^2-y^2}$$

hallar f(0,0) y representarla gráficamente.

En este caso, dado que es necesario representar la función, lo más conveniente es definirla mediante el M-fichero de la Figura 10-2.

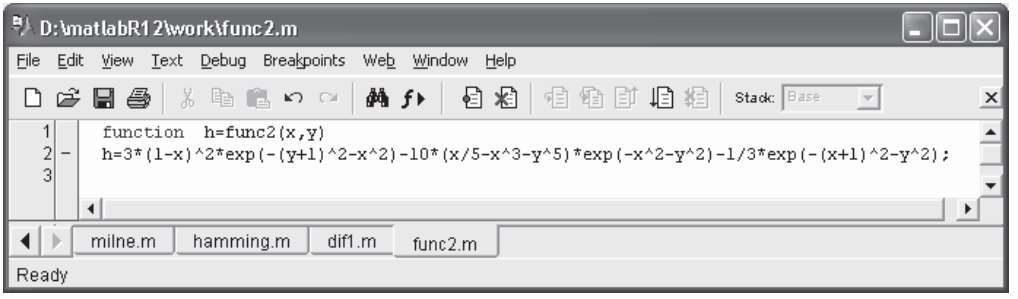


Figura 10-2

Ahora, calculamos el valor de *f* en *(0,0)*:

```
>> func2(0,0)
ans =
    0.9810
```

Para construir la gráfica de la función, utilizamos el comando *meshgrid* para definir la malla de representación (en un entorno del origen), y el comando *surf* para realizar el gráfico de superficie:

```
>> [x,y]=meshgrid(-0.5:.05:0.5,-0.5:.05:0.5);
>> z=func2(x,y);
>> surf(x,y,z)
```

Y se obtiene la gráfica de la Figura 10-3:

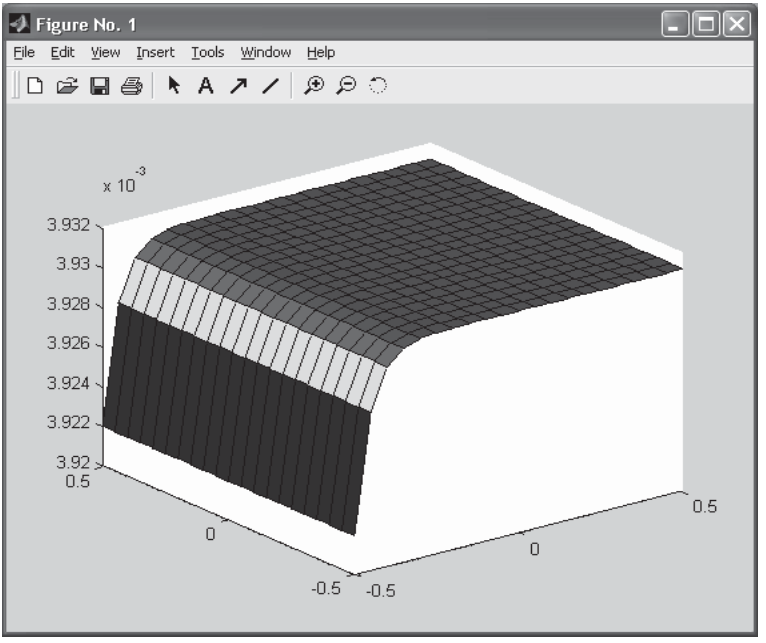


Figura 10-3

Ejercicio 10-6. Dadas las funciones f(x)=sin(cos(x$^{1/2}$) y de g(x)=sqrt(tan(x^2)) calcular la compuesta de f y g y la compuesta de g y f. Calcular también las funciones inversas de f y g.

```
>> syms x, f= sin(cos(x^(1/2)));
>> g=sqrt(tan(x^2));
>> simple(compose(f,g))

ans =

sin(cos(tan(x^2)^(1/4)))

>> simple(compose(g,f))
```

```
ans =

tan(sin(cos(x^(1/2)))^2)^(1/2)

>> F=finverse(f)

F =

acos(asin(x))^2

>> G=finverse(g)

G =

atan(x^2)^(1/2)
```

Ejercicio 10-7. Dada la función definida como:

$$f(x)=\frac{1}{1+\sqrt[x]{e}} \quad si \quad x\neq 0 \quad y \quad f(x)=1 \quad si \quad x=0$$

estudiar su continuidad en la recta real.

Salvo en el punto $x=0$ la continuidad es clara. Para analizar el punto $x=0$ calculamos los límites laterales cuando $x\to 0$:

```
>> syms x
limit(1/(1+exp(1/x)),x,0,'right')

ans =

0

>> limit(1/(1+exp(1/x)),x,0,'left')

ans =

1
```

El límite de la función en $x\to 0$ no existe porque los límites laterales son distintos. Pero como los límites laterales son finitos, la discontinuidad de primera especie en $x=0$ es de salto finito. Se ilustra este resultado con la representación de la Figura 10-4.

```
>> fplot('1/(1+exp(1/x))',[-5,5])
```

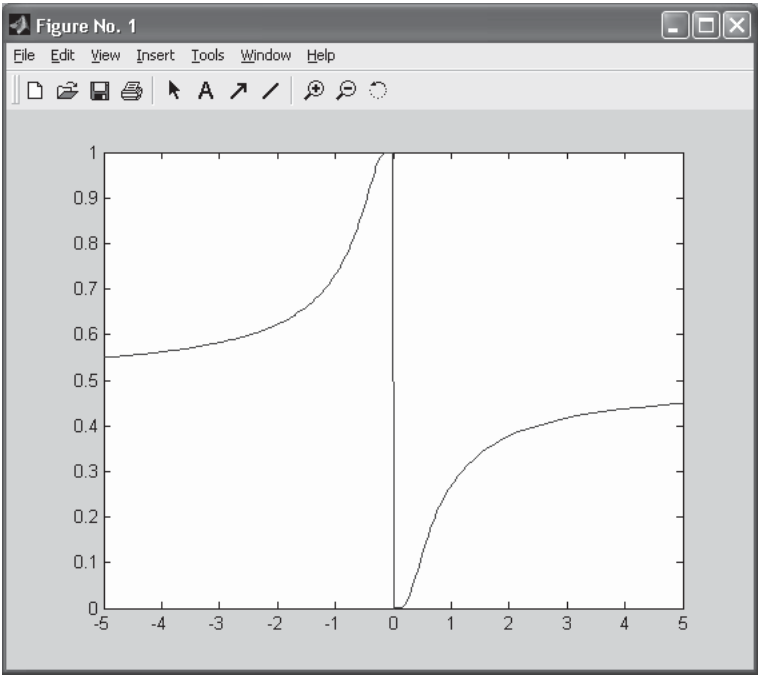


Figura 10-4

Ejercicio 10-8. Estudiar la continuidad de la función f: $R^2 \rightarrow R$ definida por:

$$f(x,y)=\frac{(x-1)^2 y^2}{(x-1)^2+y^2} \quad \textit{si (x,y)}\neq\textit{(1,0) y f(1,0)=0}$$

El único punto problemático es (1,0). Para que exista continuidad será necesario comprobar que

$$\lim_{(x,y)\to(1,0)} f(x,y)=0$$

```
» syms x y m a r
» limit(limit(y^2*(x-1)^2/(y^2+(x-1)^2),x,0),y,0)

ans =

0

» limit(limit(y^2*(x-1)^2/(y^2+(x-1)^2),y,0),x,0)

ans =

0
```

```
» limit((m*x)^2*(x-1)^2/((m*x)^2+(x-1)^2),x,0)
ans =

0

» limit((m*x)*(x-1)^2/((m*x)+(x-1)^2),x,0)

ans =

0
```

Resulta que los límites iterados y direccionales (según rectas $y=mx$) coinciden, lo que nos lleva a pensar en la existencia del límite y en que su valor es cero. Para corroborarlo calculamos a continuación el límite en coordenadas polares:

```
» limit(limit((r^2*sin(a)^2)*(r*cos(a)-1)^2/((r^2*sin(a)^2)+
   +(r*cos(a)-1)^2),r,1), a,0)

ans =

0
```

Como conclusión se obtiene que el límite vale cero en el punto (1,0), lo que asegura la continuidad de la función. En la Figura 10-5 se grafica la superficie, y se observa la continuidad y la tendencia a 0 en un entorno del punto (1,0)

```
» [x,y]=meshgrid(0:0.05:2,-2:0.05:2);
z=y.^2.*(x-1).^2./(y.^2+(x-1).^2);
mesh(x,y,z),view([-23,30])
```

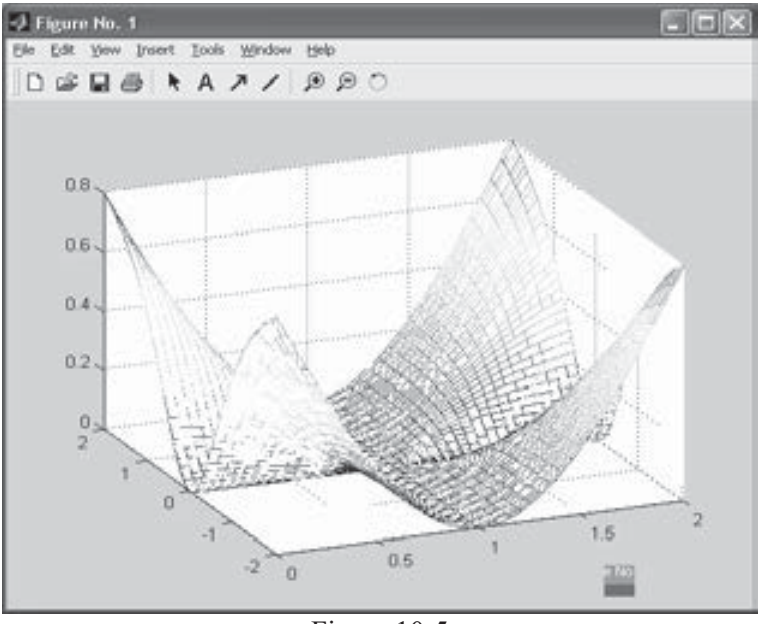


Figura 10-5

Ejercicio 10-9. Sumar las series siguientes:

$$\sum_{n=2}^{\infty} \frac{3+2n}{7^n n(1+n)} \qquad \sum_{n=1}^{\infty} \frac{n}{p^n} \qquad \textbf{\textit{p=parámetro}}$$

Antes de realizar las sumas será necesario estudiar la convergencia de las series. Aplicamos el criterio del cociente para la primera serie:

```
>> syms n
>> f=(3+2*n)/((1-n)*n*7^n);
>> pretty(f)
```

```
                              3 + 2 n
                            ------------
                                       n
                            (1 - n) n 7
```

```
>> limit(subs(f,n,n+1)/f,n,inf)

ans =

1/7
```

Como el límite es menor que 1, la serie es convergente y, por tanto, sumable. Vamos a calcular su suma. MATLAB intenta devolver el resultado, por complicado que éste sea. En muchas ocasiones, el resultado se ofrece dependiendo de determinadas funciones del programa. Veamos:

```
>> S1=symsum(f,n,2,inf)

S1 =

-6*log(6/7)-22/21+13/343*hypergeom([2, 2],[3],1/7)
```

Ahora aplicamos el criterio del cociente para la segunda serie.

```
>> syms n p
>> g=n/p^n;
>> pretty(g)
```

```
                                  n
                                 ----
                                  n
                                 p
```

```
>> limit(subs(g,n,n+1)/g,n,inf)

ans =

1/p
```

Luego, si $p > 1$, la serie converge, si $p < 1$, la serie diverge y si $p=1$, obtenemos la serie de término general n, que diverge. Cuando p es mayor que 1, hallamos la suma de la serie:

```
>> S2=symsum(g,n,2,inf)

S2 =

2/p^2*(1/2/(-1+p)^3*p^4*(-1/p+1)-1/2*p)

>> pretty(simple(S2))

                                  -1 + 2 p
                                 -----------
                                           2
                                p (-1 + p)
```

Ejercicio 10-10. Hallar el desarrollo de MacLaurin de grado 13 para la función sh(x). Hallar también el desarrollo de Taylor de orden 6 de la función 1/(1+x) en un entorno del punto x=1.

```
>> pretty(taylor(sinh(x),13))

             3          5           7           9              11
      x + 1/6 x  + 1/120 x  + 1/5040 x  + 1/362880 x  + 1/39916800 x

>> pretty(taylor(1/(1+x),6,1))

                          2              3              4              5
  3/4 - 1/4 x + 1/8 (x - 1)  - 1/16 (x - 1)  + 1/32 (x - 1)  - 1/64 (x - 1)
```

Ejercicio 10-11. Realizar un estudio completo de la función:

$$f(x) = \frac{x^3}{x^2 - 1}$$

calculando las asíntotas, máximos, mínimos, puntos de inflexón, intervalos de crecimiento y decrecimiento e intervalos de concavidad y convexidad.

```
>> f='x^3/(x^2-1)'

f =

x^3/(x^2-1)

>> syms x, limit(x^3/(x^2-1),x,inf)
```

```
ans =

NaN
```

Por lo tanto, no hay asíntotas horizontales. Para ver si las hay verticales, analicemos los valores de x que hacen y infinito:

```
>> solve('x^2-1')

ans =

[ 1]
[-1]
```

Las asíntotas verticales serán las rectas $x=1$ y $x=-1$. Veamos ahora si existen asíntotas oblicuas:

```
>> limit(x^3/(x^2-1)/x,x,inf)

ans =

1

>> limit(x^3/(x^2-1)-x,x,inf)

ans =

0
```

La recta $y = x$ es la asíntota oblicua. Ahora, se analizarán los máximos y mínimos, puntos de inflexión e intervalos de crecimiento y concavidad:

```
>> solve(diff(f))

ans =

[        0]
[        0]
[ 3^(1/2)]
[-3^(1/2)]
```

La primera derivada se anula en los puntos de abscisa $x=0$, $x=\sqrt{3}$ y $x=-\sqrt{3}$. Estos puntos son los candidatos a máximos y mínimos. Para corroborar si son máximos o mínimos, hallamos el valor de la segunda derivada en esos puntos:

```
>> [numeric(subs(diff(f,2),0)),numeric(subs(diff(f,2),sqrt(3))),
   numeric(subs(diff(f,2),-sqrt(3)))]

ans =

         0    2.5981   -2.5981
```

Por lo tanto, en el punto de abscisa $x= -\sqrt{3}$ hay un máximo y en el punto de abscisa y $x=\sqrt{3}$ hay un mínimo. En *x=0* no sabemos nada:

```
>> [numeric(subs(f,sqrt(3))),numeric(subs(f,-sqrt(3)))]

ans =

    2.5981   -2.5981
```

Por lo tanto, el punto máximo es $(-\sqrt{3}, -2.5981)$ y el punto mínimo es $(\sqrt{3}, 2.5981)$.

Vamos a analizar ahora los puntos de inflexión:

```
>> solve(diff(f,2))

ans =

[          0]
[ i*3^(1/2)]
[-i*3^(1/2)]
```

El único punto de inflexión posible se presenta en *x=0*, y como *f(0)=0*, el punto de inflexión posible es (0,0):

```
>> subs(diff(f,3),0)

ans =

-6
```

Como la tercera derivada en *x=0* no se anula, el origen es realmente un punto de inflexión:

```
>> pretty(simple(diff(f)))
```

$$\frac{x^2 (x^2 - 3)^2}{(x^2 - 1)^2}$$

La curva será creciente cuando *y'>0*, es decir, en los intervalos definidos por $(-\infty,-\sqrt{3})$ y $(\sqrt{3},\infty)$.

La curva será decreciente cuando *y'<0*, es decir, en los intervalos definidos por $(-\sqrt{3},-1)$, $(-1,0)$, $(0,1)$ y $(1,\sqrt{3})$.

```
>> pretty(simple(diff(f,2)))
```

$$2\,\frac{x\,(x^2+3)}{(x^2-1)^3}$$

La curva será cóncava cuando $y''>0$, es decir, en los intervalos $(-1,0)$ y $(1,\infty)$.

La curva será convexa cuando $y''<0$, es decir, en los intervalos $(0,1)$ y $(-\infty,-1)$.

La curva tiene tangentes horizontales en los tres puntos que anulan la primera derivada. Las ecuaciones de las tangentes horizontales son $y=0$, $y=2.5981$ e $y=-2.5981$.

La curva presenta tangentes verticales en los puntos que hacen infinita la primera derivada. Dichos puntos son $x=1$ y $x=-1$. Por lo tanto, las tangentes verticales coinciden con las dos asíntotas verticales.

A continuación, representamos la curva junto con sus asíntotas (obsérvese la Figura 10-6):

```
>> fplot('[x^3/(x^2-1),x]',[-5,5,-5,5])
```

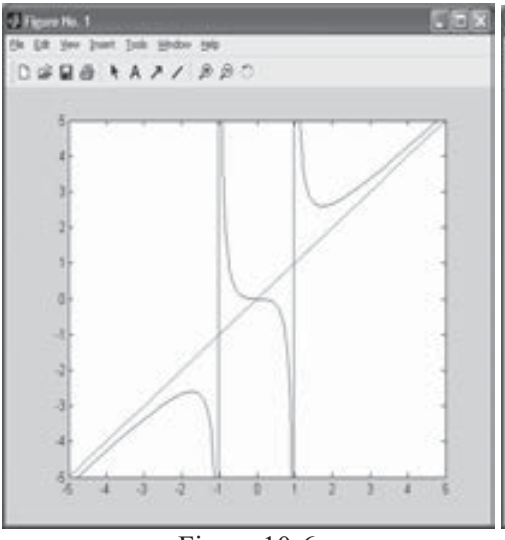


Figura 10-6

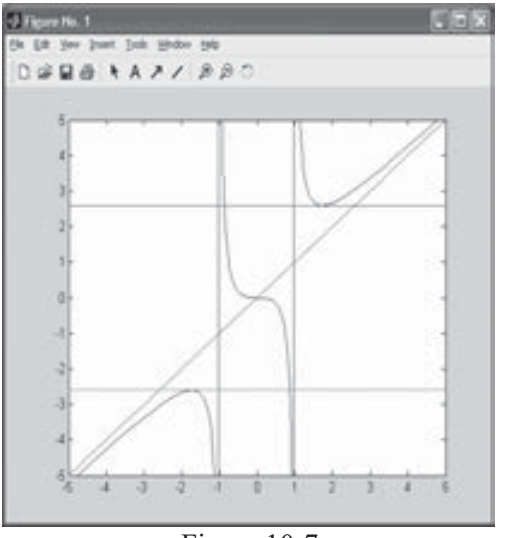


Figura 10-7

También podemos representar la curva, sus asíntotas y sus tangentes horizontales y verticales en el mismo gráfico (Figura 10-7):

```
>> fplot('[x^3/(x^2-1),x,2.5981,-2.5981]',[-5,5,-5,5])
```

Ejercicio 10-12. Dada la función vectorial (u(x,y), v(x,y)), donde:

$$u(x,y)=\frac{x^4+y^4}{x}, \qquad v(x,y)=Sen(x)+Cos(y)$$

hallar las condiciones para que exista la función vectorial inversa (x(u,v),y(u,v)) con x=x(u,v) e y=y(u,v) y hallar la derivada y el jacobiano de la transformación inversa. Hallar su valor en el punto (π/4, -π/4).

Las condiciones que tiene que cumplir serán las hipótesis del teorema de la función inversa. Las funciones son derivables con función derivada continua, salvo acaso en x=0. Ahora vamos a plantear el jacobiano de la transformación directa $\partial(u(x,y), v(x,y))/\partial(x, y)$:

```
>> syms x y
>> J=simple((jacobian([(x^4+y^4)/x,sin(x)+cos(y)],[x,y])))

J =

[ 3*x^2-1/x^2*y^4,          4*y^3/x]
[          cos(x),          -sin(y)]

>> pretty(det(J))

                          4            4      3
                  3 sin(y) x  - sin(y) y  + 4 y  cos(x) x
                - ----------------------------------------
                                     2
                                    x
```

Por lo tanto, en los puntos donde esta expresión no se anule, se puede resolver para x e y en términos de u y v. Además, también ha de cumplirse que $x\neq0$.

Calculamos la derivada de la función inversa. Su valor será la matriz inversa de la matriz jacobiana inicial y el determinante de su jacobiano será el recíproco del determinante del jacobiano inicial:

```
>> I=simple(inv(J));
>> pretty(simple(det(I)))

                                    2
                                   x
                - ----------------------------------------
                          4            4      3
                  3 sin(y) x  - sin(y) y  + 4 y  cos(x) x
```

Se observa que el determinante del jacobiano de la función vectorial inversa es el recíproco del determinante del jacobiano de la función directa.

Vamos a hallar los valores para el punto *(π/4, -π/4)*:

```
>> numeric(subs(subs(determ(I),pi/4,'x'),-pi/4,'y'))

ans =

   0.38210611216717

>> numeric(subs(subs(symdiv(1,determ(J)),pi/4,'x'),-pi/4,'y'))

ans =

   0.38210611216717
```

Estos resultados corroboran que el determinante del jacobiano de la función inversa es el recíproco del determinante del jacobiano de la función.

Ejercicio 10-13. Dada la función $f(x,y) = e^{-(x+y)}$ y la transformación u=u(x,y)=x+y, v=v(x,y)=x, hallar f(u,v).

Calculamos la transformación inversa y su jacobiano para aplicar el teorema del cambio de variable:

```
>> syms x y u v
>> [x,y]=solve('u=x+y,v=x','x','y')

x =

v

y =

u-v

>> jacobian([v,u-v],[u,v])

ans =

[  0,  1]
[  1, -1]

>> f=exp(x-y);
>> pretty(simple(subs(f,{x,y},{v,u-v})* abs(det(jacobian(
   [v,u-v],[u,v])))))

                                    exp(2 v - u)
```

La función pedida es $f(u,v) = e^{2v-u}$.

Ejercicio 10-14. Resolver las integrales siguientes:

$$\int \frac{dx}{x^3\sqrt{x^2+3x-1}}, \quad \int \frac{\sqrt{9-4x^2}}{x}\,dx, \quad \int x^8\left(3+5x^3\right)^{\frac{1}{4}}dx$$

```
>> syms x
>> pretty(simple(int(x^(-3)*(x^2+3*x-1)^(-1/2),x)))

        2            1/2          2            1/2
      (x  + 3 x - 1)           (x  + 3 x - 1)
  1/2 ----------------- + 9/4 -----------------
                2                       x
               x

                              -2 + 3 x
          + 31/8 atan(1/2 -----------------)
                            2           1/2
                          (x  + 3 x - 1)

>> pretty(simple(int(x^(-1)*(9-4*x^2)^(1/2), x)))

                             2 1/2                   3
                    (9 - 4 x )     - 3 atanh(-------------)
                                                2 1/2
                                           (9 - 4 x )

>> pretty(simple(int(x^8*(3+5*x^3)^(1/4),x)))

                                3        6         9          3 1/4
          4/73125 (288 - 120 x  + 125 x  + 1875 x ) (3 + 5 x )
```

Ejercicio 10-15. Consideramos la curva en ecuaciones polares r=3-3cos(a). Calcular la longitud del arco correspondiente a una vuelta completa ($0 \le a \le 2\pi$).

```
>> r='3-3*cos(a)';
>> diff(r,'a')

ans =

3*sin(a)

>> R=simple(int('((3-3*cos(a))^2+(3*sin(a))^2)^(1/2)','a','0','2*pi'))

R =

24
```

Ejercicio 10-16. Calcular el valor de la integral siguiente:

$$\int_{-1,96}^{1,96} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}\,dx$$

que representa el área bajo la curva normal entre los límites especificados

```
>> numeric(int('exp(-x^2/2)/(2*pi)^(1/2)','x',-1.96,1.96))

ans =

   0.95000420970356
```

Ejercicio 10-17. Se interceptan el paraboloide $ax^2+y^2=z$ y el cilindro $z=a^2-y^2$ y se trata de calcular el volumen que encierra la intersección. Hallar también el volumen de la intersección de los cilindros $z=x^2$ y $4-y^2=z$.

El primer volumen se calcula mediante la integral:

```
>> pretty(simple(int(int(int('1','z','a*x^2+y^2',
   'a^2-y^2'),'y',0,'sqrt((a^2-a*x^2)/2)'),'x',0,'sqrt(a)')))
```

```
      /
      |                2       2        2 1/2
 1/24 |    lim       3 a  x (2 a  - 2 a x )
      |       1/2
      \x -> (a   )-

                              1/2  1/2                              \
           7/2  1/2          2    a    x              2         2 3/2|
  + 3 a    2    atan(------------------) + x (2 a  - 2 a x )    |
                              2         2 1/2                     |
                         (2 a  - 2 a x )                         /
```

Para calcular el segundo volumen se realiza una gráfica de la intersección pedida, con el objeto de clarificar los límites de integración, mediante la sintaxis siguiente:

```
>> [x,y]=meshgrid(-2:.1:2);
z=x.^2;
mesh(x,y,z)
hold on;
z=4-y.^2;
mesh(x,y,z)
```

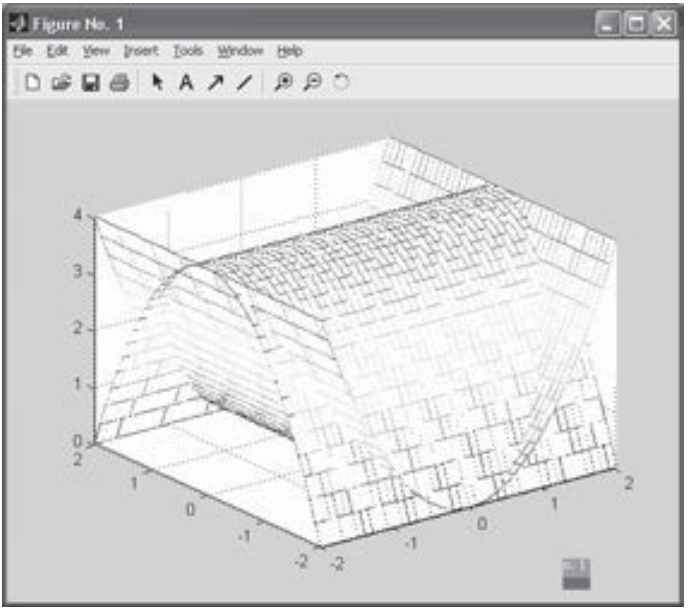


Figura 10-8

Ahora ya se puede calcular el volumen pedido mediante la integral siguiente:

```
>> pretty(simple(int(int(int('1','z','x^2','4-y^2'),
   'y',0,'sqrt(4-x^2)'),'x',0,2)))

                                    2 pi
```

Ejercicio 10-18. Resolver la ecuaciones diferenciales siguientes:

$$\frac{dy}{dx} = \frac{xy}{y^2 - x^2}$$

```
>> pretty(simple(dsolve('Dy=(x*y)/(y^2-x^2)')))

                                  t + C1
                              exp(-2 ------)
                                     x
      - 1/2 lambertw(- ---------------) x - t - C1
                                2
                               x
exp(-------------------------------------------)
                              x
```

Ejercicio 10-19. Resolver las siguientes ecuaciones diferenciales:

$$9y'''' - 6y''' + 46y'' - 6y' + 37y = 0$$
$$3y'' + 2y' - 5y = 0$$
$$2y'' + 5y' + 5y = 0 \qquad y(0) = 0 \;\; y'(0) = ½$$

```
>> pretty(simple(dsolve('9*D4y-6*D3y+46*D2y-6*Dy+37*y=0')))

C1 sin(t) + C2 cos(t) + C3 exp(1/3 t) sin(2 t) + C4 exp(1/3 t) cos(2 t)

>> pretty(dsolve('3*D2y+2*Dy-5*y=0'))

                        C1 exp(t) + C2 exp(- 5/3 t)

>> pretty(dsolve('2*D2y+5*Dy+5*y=0','y(0)=0,Dy(0)=1/2'))

                  1/2                          1/2
              2/15 15    exp(- 5/4 t) sin(1/4 15    t)
```

Ejercicio 10-20. Para las condiciones iniciales x(0)=1 e y(0)=2, resolver el siguiente sistema de ecuaciones diferenciales:

$$x' - y = e^{-t}$$
$$y' + 5x + 2y = Sin(3t)$$

```
>> [x,y]=dsolve('Dx-Dy=exp(-t),Dy+5*x+2*y=sin(3+t)','x(0)=1,y(0)=2','t')

x =

(-7/50*sin(3)+1/50*cos(3)+7/6)*exp(-7*t)+7/50*sin(3+t)-1/50*cos(3+t)-
1/6*exp(-t)


y =

(-7/50*sin(3)+1/50*cos(3)+7/6)*exp(-7*t)+5/6*exp(-t)+7/50*sin(3+t)-
1/50*cos(3+t)
```

11

Estadística, control de calidad y diseño de experimentos

11.1 Statistics Toolbox

Statistics Toolbox amplía el entorno de cálculo de MATLAB para proporcionar funciones y herramientas interactivas para análisis de datos históricos, obtención de modelos de datos, simulación de sistemas, desarrollo de algoritmos estadísticos y aprendizaje y enseñanza de estadística. Este toolbox proporciona así a ingenieros, científicos, investigadores, analistas financieros y técnicos de estadística el software estadístico que necesitan para evaluar y entender sus datos. Construido sobre las potentes y sólidas funciones numéricas de MATLAB, este toolbox contiene más de 250 funciones, así como interfaces gráficas de usuario. Estas funciones abarcan temas desde estadística descriptiva básica a desarrollo y visualización de modelos no lineales multidimensionales. Todas las funciones están ejecutadas en el lenguaje abierto de MATLAB, siendo posible explorar y personalizar los algoritmos de toolbox existentes o desarrollar los propios. Añadiendo las potentes funciones gráficas 2-D y 3-D de MATLAB, el toolbox ofrece gran variedad de tipos de plot estadístico y gráficos interactivos, tales como ajuste polinomial y modelización de superficie de respuesta.

Sus características generales más importantes son las siguientes:

- Modelización lineal y no lineal.
- Estadística multivariante.
- Estadística descriptiva.

- Cálculo y ajuste de distribuciones de probabilidad.
- Análisis de varianza (ANOVA).
- Contrastes de hipótesis.
- Estadística industrial (control de procesos estadísticos, diseño de experimentos).
- Representación gráfica estadística y gráficos interactivos.

11.2 Estadística descriptiva

La estadística descriptiva es una materia fundamental para poder entender y describir rápidamente series de datos potencialmente grandes. Para ayudarle a resumir los valores en una muestra de datos con unos pocos números muy relevantes, Statistics Toolbox incluye funciones para calcular mediciones de tendencias (posición), incluyendo promedio, mediana y varias medias; mediciones de dispersión incluyendo alcance, varianza, desviación estándar y desviación absoluta media, resultados basados en datos con valores que faltan (NaNs), estimaciones de percentiles y estadística de puntos de control entre otras materias. A continuación se presentan los comandos más relevantes.

bootstat = bootstrp(nboot,'bootfun',d1,d2,...) **[bootstat,bootsam] = bootstrp(...)**	*Extrae muestras boostrap de cada uno de los conjuntos de datos d1, d2, ... y aplica la función bootfun para obtener estadísticos de las muestras. Se puede obtener una matriz de índices boostrap mediante bootsam*
R = corrcoef(X)	*Matriz de correlaciones de X*
C = cov(X) **C = cov(x,y)**	*Matriz de covarianzas de X* *Covarianza de las variables x e y (columnas)*
table = crosstab(col1,col2) **table = crosstab(col1,col2,col3,...)** **[table,chi2,p] = crosstab(col1,col2)** **[table,chi2,p,label] = crosstab(col1,col2)**	*Tabulación cruzada de dos vectores* *Tabulación cruzada de varios vectores* *Da el estadístico chi2 para la independencia de las columnas de la tabla y el nivel p del test (valores de p cercanos a cero favorecen la independencia)* *Da también un array de celdas con una columna por cada argumento de entrada donde (i,j) es el valor de la columna j que define el grupo i en la j-ésima dimensión*
m = geomean(X)	*Media geométrica de X (si X es una matriz se obtiene un vector fila con las medias geométricas de de todas las columnas de X)*

means = grpstats(X,group) **[means,sem,counts,name] = grpstats(X,group)** **grpstats(x,group,alpha)**	*Media de cada columna de X dada por group* *Media, error estándar, número de elementos y nombre de cada columna de X dada por group* *Grafica intervalos de confianza alrededor de cada media al 100(1-alpha)%*
m = harmmean(X)	*Media armónica de X*
y = iqr(X)	*Rango intercuartílico de X (Q75-Q25)*
k = kurtosis(X) **k = kurtosis(X,flag)**	*Kurtosis de X* *Kurtosis de X corregida de sesgo sistemático si flag = 0 (Si flag = 1 vale kurtosis(X))*
y = mad(X)	*Desviación media absoluta de X*
m = mean(X)	*Media de X (Si X es matriz da un vector fila con las medias de cada columna de X)*
m = median(X)	*Mediana de X*
m = moment(X,n)	*Momento de orden n de X centrado en el origen*
m = nanmax(a) **[m,ndx] = nanmax(a)** **m = nanmax(a,b)**	*Máximo del vector a ignorando datos missing (para matrices da los máximos de las columnas)* *El vector ndx devuelve los índices de los valores máximos en las columnas de a* *Máximo de los valores correspondientes de a y b*
y = nanmean(X)	*Media de X ignorando valores missing*
y = nanmedian(X)	*Mediana de X ignorando valores missing*
m = nanmin(a) **[m,ndx] = nanmin(a)** **m = nanmin(a,b)**	*Mínimo del vector a ignorando datos missing (para matrices da los mínimos de las columnas)* *El vector ndx devuelve los índices de los valores mínimos en las columnas de a* *Máximo de los valores correspondientes de a y b*
y = nanstd(X)	*Desviación típica ignorando valores missing*
y = nansum(X)	*Suma ignorando valores missing*
Y = prctile(X,p)	*Percentil empírico p de la muestra X*
y = range(X)	*Rango de X*
y = skewness(X) **y = skewness(X,flag)**	*Coeficiente de asimetría de X* *Asimetría de X corregida de sesgo sistemático si flag = 0 (Si flag = 1 vale skewness(X))*
y = std(X)	*Desviación típica de X*
table = tabulate(x) **tabulate(x)**	*Tabla de frecuencias del vector x (valores y frecuencias absolutas)*
m = trimmean(X,percent)	*Media de X sin un porcentaje de valores extremos*
y = var(X) **y = var(X,1)** **y = var(X,w)**	*Cuasivarianza de X* *Varianza de X* *Varianza de X con el vector de pesos w*

Como primer ejemplo se halla cuasivarianza, varianza y varianza ponderada para los elementos de un vector *v* con vector de ponderaciones *w*.

```
>> x = [-1 1];
w = [1 3];
v1 = var(x)

v1 =

     2

>> v2 = var(x,1)

v2 =

     1

>> v3 = var(x,w)

v3 =

   0.75000000000000
```

En el ejemplo siguiente se hallan media aritmética, armónica, geométrica, mediana, coeficiente de asimetría, coeficiente de curtosis y desviación típica de los 100 primeros números naturales.

```
>> x=1:1:100;
>> mean(x)

ans =

  50.50000000000000

>> geomean(x)

ans =

  37.99268934483433

>> harmmean(x)

ans =

  19.27756359739600

>> median(x)

ans =

  50.50000000000000
```

```
>> skewness(x)

ans =

     0

>> kurtosis(x)

ans =

   1.79975997599760

>> std(x)

ans =

  29.01149197588202
```

En el ejemplo siguiente se construye la tabla de frecuencias para las puntuaciones 1 2 4 4 3 4.

```
>> tabulate([1 2 4 4 3 4])

  Value    Count    Percent
      1        1     16.67%
      2        1     16.67%
      3        1     16.67%
      4        3     50.00%
```

11.3 Distribuciones de probabilidad

Statistics Toolbox incluye una GUI interactiva que permite experimentar, describir o ajustar sus datos a una variedad de diferentes probabilidades. Por ejemplo, puede usar la GUI para representar gráficamente una función de densidad de probabilidad o una función de distribución acumulativa para investigar cómo los parámetros de distribución afectan a su posición y forma. Además, puede usar el generador de números aleatorios para simular el comportamiento asociado a distribuciones particulares. Se pueden utilizar entonces estos datos aleatorios para comprobar hipótesis u obtener modelos bajo diferentes condiciones.

El Statistics Toolbox aporta 20 distribuciones de probabilidad diferentes, incluyendo Student descentrada, Fisher y chi-cuadrado. Las funciones soportadas para cada distribución incluyen:

- Función de densidad de probabilidad (*pdf*).
- Función de distribución acumulativa (*cdf*).

- Inversa de la función de distribución acumulativa.
- Media y varianza.
- Generador de números aleatorios (por ejemplo, simulación de ruido).

Hay disponibles funciones adicionales para calcular estimaciones de parámetros e intervalos de confianza para las distribuciones guiadas por los datos, por ejemplo beta, binominal, exponencial, gamma, normal, Poisson, uniforme y Weibull.

A continuación se presentan las funciones más interesantes de MATLAB relativas a distribuciones de probabilidad.

Funciones de densidad, distribución e inversas

A continuación se presentan los comandos correspondientes al cálculo de la función de densidad en X, distribución y su inversa para las distribuciones estadísticas más comunes.

Valor en X de la función de distribución	Distribución estadística	Valor en X de la función de densidad	Valor en X de la inversa de la función de distribución
p = betacdf(X,A,B)	Beta(a,b)	p = betapdf(X,A,B)	p = betainv(X,A,B)
Y = binocdf(X,N,P)	Binomial(n,p)	Y = binopdf(X,N,P)	Y = binoinv(X,N,P)
P = cdf('*d*',X,A1,A2,A3)	Distribución d	P = pdf('*d*',X,A1,A2,A3)	P = inv('*d*',X,A1,A2,A3)
P = chi2cdf(X,V)	$\chi 2$ (v)	P = chi2pdf(X,V)	P = chi2inv(X,V)
P = expcdf(X,M)	Exponencial(m)	P = exppdf(X,M)	P = expinv(X,M)
P = fcdf(X,V1,V2)	F(v1,v2) de Fisher	P = fpdf(X,V1,V2)	P = finv(X,V1,V2)
P = gamcdf(X,A,B)	Gamma(a,b)	P = gampdf(X,A,B)	P = gaminv(X,A,B)
Y = geocdf(X,P)	Geométrica(p)	Y = geopdf(X,P)	Y = geoinv(X,P)
P = hygecdf(X,M,K,N)	Hyperg.(m,k,n)	P = hygepdf(X,M,K,N)	P = hygeinv(X,M,K,N)
P = logncdf(X,M,S)	Lognormal(m,s)	P = lognpdf(X,M,S)	P = logninv(X,M,S)
Y = nbincdf(X,R,P)	Binomial negativa(r,p)	Y = nbinpdf(X,R,P)	Y = nbininv(X,R,P)
P = ncfcdf(X,N1,N2,D)	F descentrada(n1,n2,d)	P = ncfpdf(X,N1,N2,D)	P = ncfinv(X,N1,N2,D)
P = nctcdf(X,NU,D)	T descentrada(η,d)	P = nctpdf(X,NU,D)	P = nctinv(X,NU,D)
P = ncx2cdf(X,V,D)	$\chi 2$ (v, d) descentrada	P = ncx2pdf(X,V,D)	P = ncx2inv(X,V,D)
P = normcdf(X,M,S)	Normal(m,s)	P = normpdf(X,M,S)	P = norminv(X,M,S)
P = poisscdf(X,L)	Poisson(l)	P = poisspdf(X,L)	P = poissinv(X,L)
P = raylcdf(X,B)	Rayleigh(b)	P = raylpdf(X,B)	P = raylinv(X,B)
P = tcdf(X,V)	T(v) de Student	P = tpdf(X,V)	P = tinv(X,V)
P = unidcdf(X,N)	Unif. discreta(n)	P = unidpdf(X,N)	P = unidinv(X,N)
P = unifcdf(X,A,B)	Unif. continua(a,b)	P = unifpdf(X,A,B)	P = unifinv(X,A,B)
P = weibcdf(X,A,B)	Weibull(a,b)	P = weibpdf(X,A,B)	P = weibinv(X,A,B)

Como primer ejemplo calculamos el valor de la función de distribución de una beta (2,2) en los números impares entre 0,1 y 0,9.

```
>> x = 0.1:0.2:0.9;
a = 2;
b = 2;
p = betacdf(x,a,b)

p =

    0.0280    0.2160    0.5000    0.7840    0.9720
```

En el ejemplo siguiente se calculan los valores de la función de distribución en los punto enteros entre -2 y 2 para un normal (0,1).

```
>> p = cdf('Normal',-2:2,0,1)

p =

    0.0228    0.1587    0.5000    0.8413    0.9772
```

En el ejemplo siguiente se considera una distribución binomial (200, 0,02) y se calcula el valor de su ley de probabilidad en los números enteros entre 0 y 5.

```
>> binopdf([0:5],200,0.02)

ans =

    0.0176    0.0718    0.1458    0.1963    0.1973    0.1579
```

En el ejemplo siguiente se contesta a la pregunta anterior, pero para una hipergeométrica (100,20,10).

```
>> p = hygepdf(0:5,100,20,10)

p =

    0.0951    0.2679    0.3182    0.2092    0.0841    0.0215
```

En el ejemplo siguiente se calcula el intervalo que contiene el 95% de los valores de una distribución normal (0,1). La solución consiste en calcular los valores de la inversa de un anormal (0,1) en los puntos 0,025 y 0,975 como sigue:

```
>> x = norminv([0.025 0.975],0,1)

x =

   -1.9600    1.9600
```

Momentos y generación de números aleatorios

A continuación se presentan los comandos correspondientes al cálculo de los momentos (media y varianza) para las distribuciones estadísticas más comunes y de generación de números aleatorios según estas distribuciones.

Media M y varianza V de la distribución	Distribución estadística	Matriz R(m,n) de números aleatorios según la distribución (*m* y *n* son opcionales)
[M,V] = betastat(A,B)	Beta(a,b)	R = betarnd(A,B,m,n)
[M,V] = binostat(N,P)	Binomial(n,p)	R = binornd(N,P,m,n)
[M,V] = stat('*d*', A1,A2,A3)	Distribución d	
[M,V] = chi2stat(V)	$\chi 2$ (v)	R = chi2rnd(V,m,n)
[M,V] = expstat(M)	Exponencial(m)	R = exprnd(M,m,n)
[M,V] = fstat(V1,V2)	F(v1,v2) de Fisher	R = frnd(V1,V2,m,n)
[M,V] = gamstat(A,B)	Gamma(a,b)	R = gamrnd(A,B,m,n)
[M,V] = geostat(P)	Geométrica(p)	R = geornd(P,m,n)
[M,V] = hygestat(M,K,N)	Hyperg.(m,k,n)	R = hygernd(M,K,N,m,n)
[M,V] = lognstatM,S)	Lognormal(m,s)	R = lognrnd(M,S,m,n)
[M,V] = nbinstat(R,P)	Binomial negativa(r,p)	R = nbinrnd(R,P,m,n)
[M,V] = ncfstat(N1,N2,D)	F descentrada(n1,n2,d)	R = ncfrnd(N1,N2,D,m,n)
[M,V] = nctstat(NU,D)	T descentrada(η,d)	R = nctrnd(NU,D,m,n)
[M,V] = ncx2stat(V,D)	$\chi 2$ (v, d) descentrada	R = ncx2rnd(V,D,m,n)
[M,V] = normstat(M,S)	Normal(m,s)	R = normrnd(M,S,m,n)
[M,V] = poisstat(L)	Poisson(l)	R = poissrnd(L,m,n)
[M,V] = raylstat(B)	Rayleigh(b)	R = raylrnd(B,m,n)
[M,V] = tstat(V)	T(v) de Student	R= trnd(V,m,n)
[M,V] = unidstat(N)	Unif. discreta(n)	R= unidrnd(N,m,n)
[M,V] = unifstat(A,B)	Unif. continua(a,b)	P = unifrnd(A,B,m,n)
[M,V] = weibstat(A,B)	Weibull(a,b)	P = weibrnd(A,B,m,n)

En el ejemplo siguiente hallamos primeramente un número aleatorio según una normal (0,1) y posteriormente hallamos una matriz 2x2 de valores aleatorios normales 0,1).

```
>> R = normrnd(0,1)

R =

   -0.4326

>> R = normrnd(0,1,2,2)

R =

   -1.6656    0.2877
    0.1253   -1.1465
```

En el ejemplo siguiente calculamos una matriz 2x3 de números aleatorios exponenciales de parámetro 5.

```
>> n3 = exprnd(5,2,3)

n3 =

    0.2558    2.4974    0.5754
    7.3237    3.6079    1.3584
```

En el ejemplo siguiente calculamos la media y la varianza de las distribuciones gamma de parámetros (a,a) a=1,...,5,

```
>> [m,v] = gamstat(1:5,1:5)

m =

     1     4     9    16    25


v =

     1     8    27    64   125
```

11.4 Gráficos estadísticos

MATLAB permite realizar los gráficos estadísticos más comunes a través de los comandos cuya sintaxis se presenta a continuación:

boxplot(X)	*Gráfico de caja y bigotes para X*
boxplot(X,notch)	*Gráfico con muescas si notch=1*
boxplot(X,notch,'*sym*')	*Gráfico con el símbolo dado para los outliers*
boxplot(X,notch,'*sym*',vert)	*Gráfico de caja y bigotes vertical*
boxplot(X,notch,'*sym*',vert,whis)	*Gráfico con longitud de los bigotes whis dada*
cdfplot(X)	*Gráfico de la función de distribución empírica de X*
h = cdfplot(X)	*Crea un manejador para el gráfico*
[h,stats] = cdfplot(X)	*Devuelve el manejador y estadísticos (mínimo, máximo, media, mediana y desviación típica)*
errorbar(X,Y,L,U,symbol)	*Gráfico de X contra Y con barras de error dadas por L y U y con el símbolo especificado*
errorbar(X,Y,L)	*Gráfico de X contra Y con barras de error dadas por L simétricas respecto de Y*
errorbar(Y,L)	*Grafica Y con barras de error [Y-L, Y+L]*

fsurfht('fun',xlims,ylims) **fsurfht('fun',xlims,ylims,** **p1,p2,p3,p4,p5)**	*Gráfico de contorno interactivo para una función* *Son posibles 5 parámetros opcionales*
gline(fig) **h = gline(fig)** **gline**	*Gráfico de línea interactivo*
gname('cases') **gname**	*Etiquetado de puntos interactivo*
gplotmatrix(x,y,g) **gplotmatrix(x,y,g,'clr','sym',siz)** **gplotmatrix(x,y,g,'clr','sym',siz,** **'doleg')** **gplotmatrix(x,y,g,'clr','sym',siz,** **'doleg','dispopt')** **gplotmatrix(x,y,g,'clr','sym',siz,** **'doleg','dispopt','xnam','ynam')** **[h,ax,bigax] = gplotmatrix(...)**	*Crea una matriz de gráficos de dispersión de x contra y agrupados por una variable común g. Se puede especificar color (clr), tipo de marcador (sym) y tamaño de cada grupo (siz). Se puede controlar la leyenda en el gráfico (doleg=on) y la presencia de la diagonal (x,x) mediante dispopt y los nombres de las columnas de x e y mediante xnam e ynam*
gscatter(x,y,g) **gscatter(x,y,g,'*clr*','*sym*',siz)** **gscatter(x,y,g,'*clr*','*sym*',siz, '*doleg*')** **gscatter(x,y,g,'*clr*','*sym*',siz,** **'*doleg*','xnam','ynam')** **h = gscatter(...)**	*Crea un gráfico de dispersión de x contra Y agrupado según g*
lsline	*Añade a un gráfico la línea de ajuste por mínimos cuadrados*
normplot(X)	*Realiza el gráfico de probabilidad normal*
pareto(y)	*Realiza el gráfico de Pareto de Y*
qqplot(X) **qqplot(X,Y)** **qqplot(X,Y,pvec)**	*Gráfico normal de cuantiles para X* *Gráfico de cuantiles para dos muestras X e Y* *Especifica los cuantiles en el vector pvec*
rcoplot(r,rint)	*Gráfico de barras de error para los intervalos de confianza de los residuos de una regresión cuya salida es (r,rint)*
h = refcurve(p)	*Añade una curva polinómica al gráfico actual*
refline(slope,intercept) **refline(slope)** **h = refline(slope,intercept)** **refline**	*Añade a los ejes corrientes una línea de referencia de pendiente slope y ordenada en el origen intercept*
surfht(Z) **surfht(x,y,Z)**	*Gráfico de contorno interactivo para la matriz Z. Los vectores x, y especifican los ejes X e Y en el contorno*
weibplot(X) **h = weibplot(X)**	*Gráfico de probabilidad de Weibull para X*

Como primer ejemplo consideramos dos variables *x*1 y *x*2 que contienen 100 números aleatorios según distribuciones normales (5,1) y (6,1), respectivamente. Se trata de realizar un gráfico de caja y bigotes para las dos variables en el mismo gráfico (Figura 11-1).

```
>> x1 = normrnd(5,1,100,1);
x2 = normrnd(6,1,100,1);
x = [x1 x2];
boxplot(x,1)
```

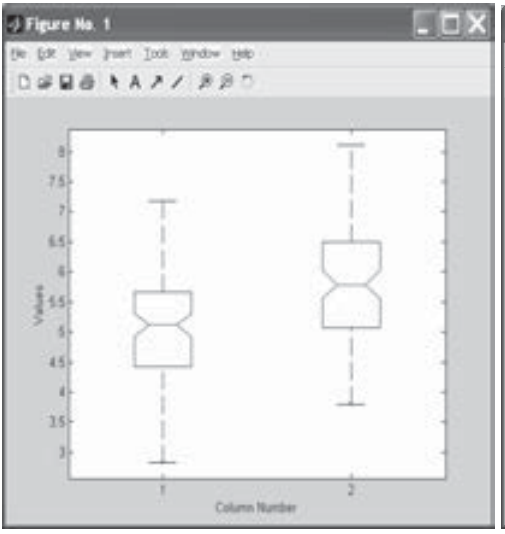


Figura 11-1

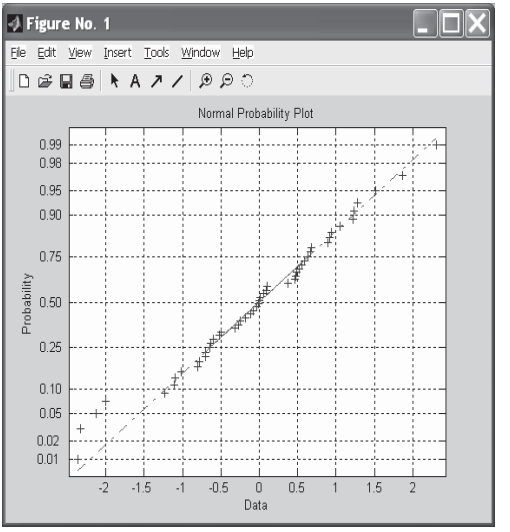


Figura 11-2

En el ejemplo siguiente realizamos un gráfico de normalidad para 50 puntos aleatorios de una normal (0,1). Evidentemente el gráfico se ajusta a la diagonal porque los datos ya son en sí normales (Figura 11-2).

```
>> x = normrnd(0,1,50,1);
h = normplot(x);
```

En el ejemplo siguiente se realiza un gráfico de Pareto para controlar 4 defectos en un producto elaborado con frecuencias dadas (Figura 11-3).

```
>> defects = ['d1';'d2';'d3';'d4'];
>> quantity = [5 3 19 25];
pareto(quantity,defects)
```

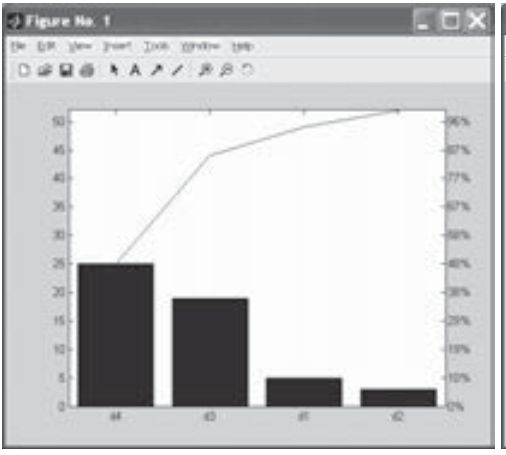


Figura 11-3

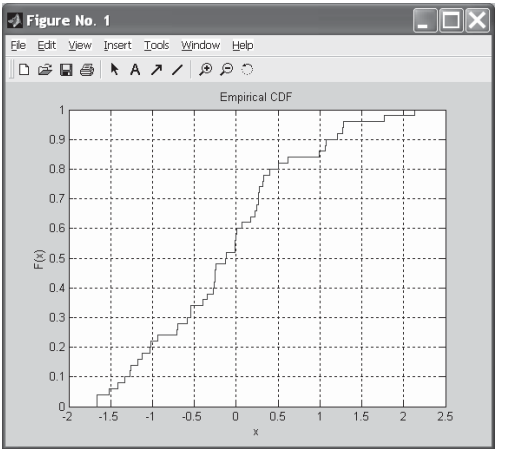


Figura 11-4

A continuación se realiza un gráfico de función de distribución empírica (Figura 11-4) para una muestra de 50 números aleatorios normales (0,1).

```
x = normrnd(0,1,50,1);
cdfplot(x)
```

En el ejemplo consideramos dos variables $x1$ y $x2$ que contienen 100 números aleatorios según distribuciones normales (5,1) y (6,1), respectivamente. Se trata de realizar un gráfico de cuantiles para las dos variables (Figura 11-5).

```
>> x = normrnd(0,1,100,1);
y = normrnd(0.5,2,50,1);
qqplot(x,y);
```

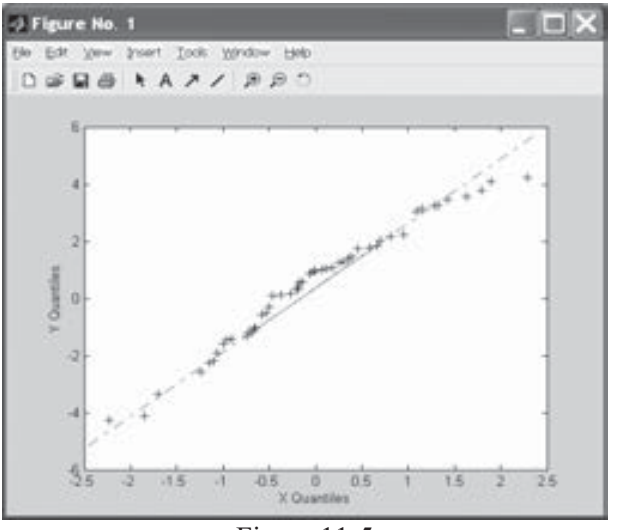


Figura 11-5

11.5 Modelos lineales y no lineales

Los modelos lineales y no lineales proporcionados en el Statistics Toolbox son útiles para obtener el modelo de una variable de respuesta como función de una o más variables predictoras. Estos modelos hacen previsiones, establecen relaciones entre variables o simplifican un problema. Por ejemplo, los modelos de regresión lineal y no lineal pueden ayudar a establecer qué variables tienen más impacto sobre una respuesta. Métodos de regresión sólidos pueden ayudarle a encontrar resultados aislados y reducir su efecto sobre el modelo ajustado. El toolbox soporta algoritmos lineales para ANOVA de una, dos y multivía; regresión polinonial, progresiva, de crestas, robusta y lineal múltiple; modelos lineales generalizados MLG y ajuste de la superficie de respuesta. Las funciones de ajuste no lineal proporcionan estimación de parámetros, visualización interactiva y predicción de ajuste no lineal multidimensional e intervalos de confianza para parámetros y valores previstos. A continuación se presentan las funciones adecuadas:

p = anova1(X) **p = anova1(X,group)** **p = anova1(X,group,'*displayopt*')** **[p,table] = anova1(...)** **[p,table,stats] = anova1(...)**	*One-way ANOVA balanceado comparando las medias de las columnas de la matriz X. Da el p-valor p para contrastar la hipótesis nula de que las muestras (columnas) provienen de la misma población. Lops p-valores entre 0,05 y 0,01 indican muestras significativamente diferentes. Se puede obtener la tabla ANOVA y el gráfico de caja para cada columna de X que se pueden etiquetar según los valores de la variable group. También se pueden obtener estadísticos de estructura de tests de comparación múltiple (stats)*
p = anova2(X,reps) **p = anova2(X,reps,'displayopt')** **[p,table] = anova2(...)** **[p,table,stats] = anova2(...)**	*Two-way NOVA balanceado para comparar las medias de dos o más columnas y dos o más filas de observaciones en X. El valor reps indica el número de replicaciones en cada celda*
p = anovan(X,group) **p = anovan(X,group,'*model*')** **p = anovan(X,group,'*model*', sstype)** **p = anovan(X,group,'*model*', sstype,gnames)** **p = anovan(X,group,'*model*',sstype, gnames,'*displayopt*')** **[p,table] = anovan(...)** **[p,table,stats] = anovan(...)** **[p,table,stats,terms] = anovan(...)**	*N-way ANOVA balanceado o no balanceado para comparar las medias de las observaciones del vector X respecto a N diferentes factores. Los factores y los niveles de las observaciones en X se asignan mediante group. Se pueden utilizar los modelos (model) 'linear', 'interaction' y 'full' con los tipos de sumas de cuadrados 1, 2 o 3 (sstype). Se pueden mostrar etiquetas para los tipos de factores según los valores de gnames y mostrar la tabla ANOVA si displayopt = on. También se pueden mostrar estadísticos para tests de comparación múltiple (stats).*

aoctool(x,y,g) **aoctool(x,y,g,alpha)** **aoctool(x,y,g,alpha,xname, yname,gname)** **aoctool(x,y,g,alpha,xname, yname,gname,'*displayopt*')** **aoctool(x,y,g,alpha,xname, yname,gname,'*displayopt*','*model*')** **h = aoctool(...)** **[h,atab,ctab] = aoctool(...)** **[h,atab,ctab,stats] = aoctool(...)**	*Herramienta interactiva para el análisis de la covarianza. Ajusta separadamente los vectores columna x e y para cada valor de g. Estos modelos se conocen como ANCOVA y devuelven un gráfico interactivo de dato y curvas de predicción, una tabla ANOVA y una tabla de parámetros estimados. El valor alpha determina el nivel de confianza de los intervalos de predicción y xname, yname y gname indican los nombres a utilizar para x, y y g en las tablas, y model puede tomar los valores same mean, separate means, same line, parallel lines y separate lines, según se quiera ajuste de medias sin considerar grupos, considerando grupos, línea simple o líneas separadas por grupos con y sin restricciones. Se obtienen tablas ANOVA y coeficientes estimados mediante atabs y ctabs.*
D = dummyvar(group)	*Codificación 0-1 de variables dummy*
p = friedman(X,reps) **p = friedman(X,reps,'*displayopt'*)** **[p,table] = friedman(...)** **[p,table,stats] = friedman(...)**	*Test no paramétrico two-way ANOVA de Friedman para comparar las medias de las columnas de X para un número dado de replicaciones reps.*
b = glmfit(X,Y,'distr') **b = glmfit(X,Y,'distr','link', 'estdisp',offset,pwts,'const')** **[b,dev,stats] = glmfit(...)**	*Ajusta un modelo lineal generalizado GLM con respuesta Y, predicor X y distribución distr ('binomial', 'gamma', 'inverse gaussian', 'lognormal', 'normal' y 'poisson'). Link indica el tipo ajuste: identity (u=xb), log (log(u)=xb), logit (log(u/(1-u))=xb), probit (norminv(μ) = xb), reciprocal (1/u=xb) y p (u^p=xb).*
p = kruskalwallis(X) **p = kruskalwallis(X,group)** **p = kruskalwallis(X,group,'*displayopt*')** **[p,table] = kruskalwallis(...)** **[p,table,stats] = kruskalwallis(...)**	*Test no paramétrico one-way anova de Kruskal-Wallis test para comparar las medias de las columnas de X. Se puede obtener la tabla ANOVA y el gráfico de caja para cada columna de X que se pueden etiquetar según los valores de la variable group. También se pueden obtener estadísticos de estructura de tests de comparación múltiple (stats)*
h = leverage(data) **h = leverage(data,'*model*')**	*Contraste de puntos influyentes en un ajuste por regresión, incluso usando un modelo dado (interaction, quadratic o purequadratic)*
x = lscov(A,b,V) **[x,dx] = lscov(A,b,V)**	*Ajusta por mínimos A*x = b + e con matriz de covarianzas dada y normalidad de residuos. El valor dx devuelve el error estándar de x*

d = manova1(X,group) **d = manova1(X,group,alpha)** **[d,p] = manova1(...)** **[d,p,stats] = anova1(...)**	*One-way ANOVA multivariable para comparar medias multiples de las columnas de X agrupadas por group. Se puede especificar el nivel de significación alpha y obtener el vector p de p-valores y estadísticos de ajuste (stats)*
manovacluster(stats) **manovacluster(stats,'method')** **H = manovacluster(stats)**	*Dibuja cluster para los grupos de medias en manova1 mediante un dendograma para la salida stats de manova1. El método de enlace (method) puede ser single, complete, average, centroid y ward.*
c = multcompare(stats) **c = multcompare(stats,alpha)** **c = multcompare(stats,alpha, '*displayopt*')** **c = multcompare(stats,alpha, '*displayopt*','*ctype*')** **c = multcompare(stats,alpha, '*displayopt*','*ctype*','*estimate*')** **c = multcompare(stats,alpha, '*displayopt*','*ctype*','*estimate*',dim)** **[c,m] = multcompare(...)** **[c,m,h] = multcompare(...)**	*Comparaciones múltiples de medias y otros estimadores. El argumento ctype especifica el método de comparación múltiple (hsd, lsd, bonferroni, scheffe y dunn-sidak). El argumento estimate indica la estimación en la comparación (anova1, anova2, anovan, aoctool, Friedman o kruskalwallis). La población marginal a comparar se especifica por dim y se puede obtener la matriz m cuya primera columna estima las medias de los grupos y cuya segunda columna estima las desviaciones típicas. El valor h es el manejador de un gráfico de comparación*
[Y,D] = polyconf(p,X,S) **[Y,D] = polyconf(p,X,S,alpha)**	*Predicción polinómica con intervalos de confianza Y±D de la salida S dada por polyfit con nivel de confianza alpha (o 95%)*
[p,S] = polyfit(x,y,n)	*Da los coeficientes del polinomio p de grado n que ajusta los puntos (x,y) por mínimos cuadrados, con errores estimados S*
Y = polyval(p,X) **[Y,D] = polyval(p,X,S)**	*Predice el valor de p en los valores X* *Usa la salida S de polyfit para generar los errores Y±D*
rcoplot(r,rint)	*Muestra en gráfico de barras de error de los intervalos de confianza residuales donde r y rint son las salidas de regress*
b = regress(y,X) **[b,bint,r,rint,stats] = regress(y,X)** **[b,bint,r,rint,stats] = regress(y,X,alpha)**	*Regresión lineal múltiple $y = X\beta + e$* *Da el estimador de β (b), un intervalo de confianza para β (bint), los residuos (r), un intervalo de confianza para los residuos (rint) y los estadísticos R2, F y los p-valores de los parámetros (stats). El nivel de confianza es el 95%, pero se puede dar otro*

regstats(responses,DATA) **regstats(responses,DATA,'*model*')**	*Genera diagnósticos para la regresión lineal con término constante. Se puede tomar model como interaction, quadratic y purequadratic*
b = ridge(y,X,k)	*Regresión en cadena $y = X\beta + e$ con constante k (si k = 0 es mínimos cuadrados)*
rstool(x,y) **rstool(x,y,'model')**	*Genera superficies de respuesta para una regresión múltiple de x sobre y*
b = robustfit(X,Y) **[b,stats] = robustfit(X,Y)**	*Realiza regresión robusta de X sobre Y*
rstool(x,y) **rstool(x,y,'model')**	*Herramienta de visualización de superficies de respuesta multidimensional (RSM)*
stepwise(X,y) **stepwise(X,y,inmodel)** **stepwise(X,y,inmodel,alpha)**	*Realiza regresión paso a paso de y sobre X y Muestra una ventana interactiva controlando el paso y los términos del modelo. El vector inmodel indica los índices de las columnas de la matriz X que se incluyen en el modelo inicial y alpha es el nivel de confianza*
D = x2fx(X) **D = x2fx(X,'model')**	*Transforma una matriz X de inputs del sistema a una matriz de diseño D para modelo lineal aditivo con término constante*
Regresión no lineal	
[beta,r,J] = nlinfit(X,y,f,beta0)	*Ajuste no lineal por mínimos cuadrados del modelo $y = f(\beta X)+e$ con valores iniciales beta0 para β (método de Gauss-Newton)*
nlintool(x,y,FUN,beta0) **nlintool(x,y,FUN,beta0,alpha)**	*Gráficos de predicción para el ajuste no lineal por mínimos cuadrados de Gauss-Newton*
ci = nlparci(beta,r,J)	*Intervalos de confianza al 95% para las estimaciones beta del modelo no lineal con residuos r y jacobiana J*
ypred = **nlpredci(FUN,inputs,beta,r,J)** **[ypred,delta] =** **nlpredci(FUN,inputs,beta,r,J)**	*Intervalos de confianza para las predicciones en el modelo no lineal donde inputs es una matriz de valores de la variable independiente para generar las predicciones*
x = nnls(A,b) **x = nnls(A,b,tol)** **[x,w] = nnls(A,b)** **[x,w] = nnls(A,b,tol)**	*Mínimos cuadrados no negativos. Soluciones no negativas de $Ax = b$. Se puede dar una tolerancia y obtenerse el vector dual w del problema*

Como primer ejemplo consideramos tres tipos de aleaciones y sus respectivas resistencias medidas sobre una muestra de 20 tornillos. Se trata de contrastar si existen diferencias significativas entre los tres tipos de aleaciones en cuanto a resistencia.

```
>> resistencia = [82 86 79 83 84 85 86 87 74 82 78 75 76 77 79 79
                  77 78 82 79];

>> aleacion = {'st','st','st','st','st','st','st','st','al1','al1',
  'al1','al1','al1','al1','al1','al2','al2','al2','al2','al2','al2'};
>> p = anova1(resistencia,aleacion)

p =

  1.5264e-004
```

Se obtiene como resultado un *p*-valor muy pequeño que indica que las tres aleaciones son significativamente diferentes. Además se obtienen la tabla ANOVA de la Figura 11-6 y el gráfico de caja y bigotes de la Figura 11-7 que presenta las distribuciones de los tres tipos de aleación. Se observa claramente que esas distribuciones son diferentes.

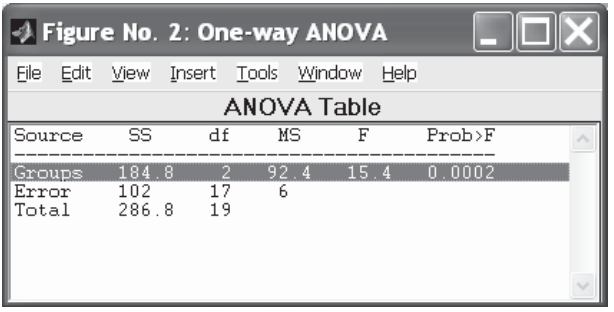


Figura 11-6

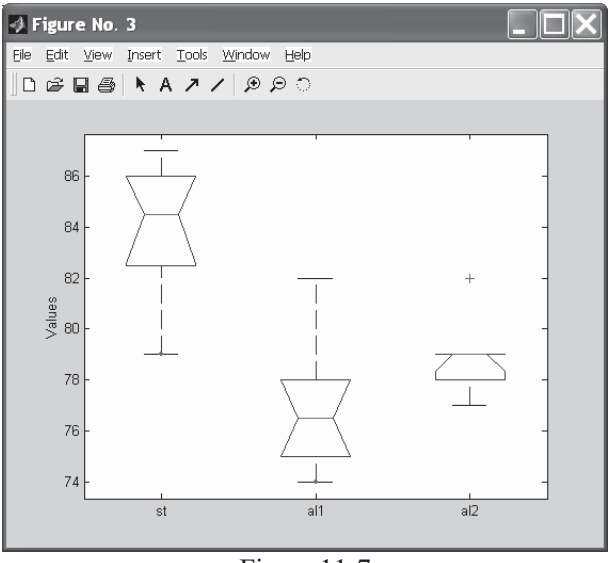


Figura 11-7

En el ejemplo siguiente consideramos automóviles con pesos entre 2.100 y 4.300. Para cada peso hay un número total de automóviles de ese peso y un número de entre ellos con distancia recorrida deficiente. Se presentan ajustes logit y probit para los datos.

```
>> [bl,dl] = glmfit(w,[deficientes total],'binomial')

bl =

  -13.3801
    0.0042

dl =

    6.4842

>> [bp,dp] = glmfit(w,[deficientes total],'binomial','probit')

bp =

   -7.3628
    0.0023

dp =

    7.5693
```

El valor de *dl* en el logit es menor que el valor de *dp* en el probit, lo que es un indicio de que puede ser mejor ajuste el modelo logiat.

En el ejemplo siguiente realizamos regresión lineal simple para los datos anteriores obteniendo el estimador de β (*b*), un intervalo de confianza para β (*bint*) al 95%, los residuos (*r*), un intervalo de confianza para los residuos (*rint*) y los estadísticos $R2$, F y los p-valores de los parámetros (*stats*).

```
>> [b,bint,r,rint,stats] = regress(deficientes,total)

b =

    0.2712

bint =

    0.0420    0.5005
```

```
r =

  -12.0199
   -9.3924
   -8.4087
   -6.2224
   -0.4087
    2.3038
    7.7613
   10.7613
   13.3038
   10.6600
   12.3888
   15.3038

rint =

  -31.4899    7.4502
  -30.3102   11.5254
  -30.6001   13.7827
  -28.5087   16.0639
  -23.3571   22.5398
  -21.2377   25.8453
  -15.0970   30.6197
  -11.5004   33.0230
   -8.3994   35.0071
  -11.9766   33.2967
   -9.7744   34.5520
   -5.7512   36.3588

stats =

    0.2958       NaN       NaN
```

A continuación se calculan varios estadísticos de la regresión mediante la sintaxis de la Figura 11-8.

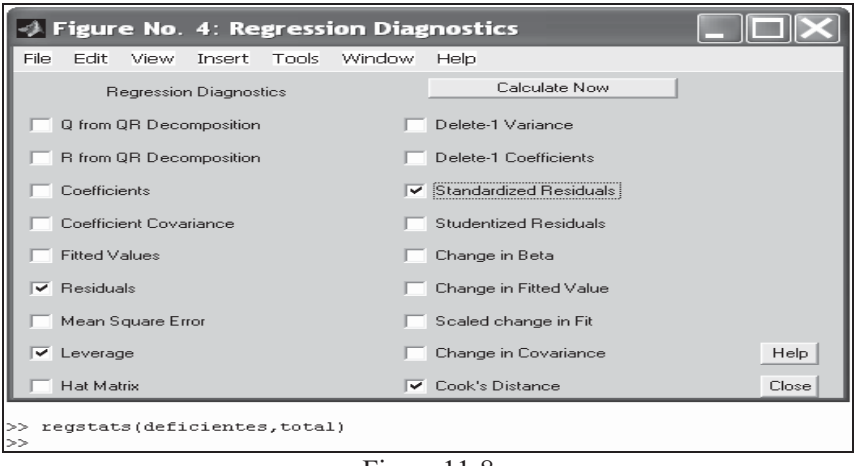


Figura 11-8

11.6 Análisis multivariante

La estadística multivariante permite analizar datos evaluando grupos de variables en conjunto. El análisis multivariante puede ayudar a segmentar datos en grupos para análisis posterior. También puede ayudar a evaluar y apreciar las diferencias entre grupos en una serie de datos. Además, los métodos multivariante pueden reducir un número importante de variables a una serie más manejable pero todavía representativa de la serie inicial. Las áreas de estadística multivariante cubiertas en el Statistics Toolbox incluyen análisis de componentes principales (PCA), análisis de agrupamientos (cluster), análisis de discriminante lineal y ANOVA multivariante (técnica ya estudiada). Las funciones más interesantes son las siguientes:

Análisis en componentes principales	
ndim = barttest(x,alpha) **[ndim,prob,chisquare] = barttest(x,alpha)**	*Test de Bartlett para dimensionalidad (número de componentes principales a retener). Se puede obtener la significación del test y la* χ^2
pc = pcacov(X) **[pc,latent,explained] = pcacov(X)**	*A partir de la matriz de covarianzas X de las variables iniciales, da las componentes principales pc, sus autovalores (latent) y el porcentaje de varianza que explica cada uno (explained)*
residuals = pcares(X,ndim)	*Residuos en el análisis de componentes principales*
PC = princomp(X) **[PC,SCORE,latent,tsquare] = princomp(X)**	*Análisis de componentes principales para matriz de datos fila, ofreciendo las componentes, sus puntuaciones, varianza explicada por cada componente y la* T^2 *de Hotelling*
Análisis discriminante	
class = classify(sample,training,group)	*Realiza análisis discriminante lineal asignando cada fila de la muestra (simple) a un grupo (group) en que se ha divido el conjunto training*
d = mahal(Y,X)	*Distancia de Mahalanobis entre X e Y*
Análisis Cluster	
T = cluster(Z,cutoff) **T = cluster(Z,cutoff,depth,flag)**	*Construye clusters para la salida del comando linkage del arbol Z*
T = clusterdata(X,cutoff)	*Construye clusters de los datos de la matriz X*
c = cophenet(Z,Y)	*Calcula los coeficientes de correlación cophenetic*
H = dendrogram(Z) **H = dendrogram(Z,p)**	*Realiza el dendograma del árbol Z* *Realiza el dendograma del árbol Z con p nodos*
Y = inconsistent(Z) **Y = inconsistent(Z,d)**	*Calcula el coeficiente de inconsistencia del árbol Z en clusters jerárquicos (pueden usarse p nodos)*
Z = linkage(Y) **Z = linkage(Y,'method')**	*Crea el árbol de clusters jerárquicos por el algoritmo Single Linkage para el vector de distancias Y generado por el comando pdist*

Y = pdist(X) **Y = pdist(X,'metric')** **Y = pdist(X,'minkowski',p)**	*Calcula la distancia euclidiana entre pares de objetos de la matriz X. Se puede usar la distancia dada en metric (Euclid, SEuclid, mahal, CityBlock y Minkowski) y p es el coeficiente de Minkowski*
S = squareform(Y)	*Reformatea la salida de pdist a matriz cuadrada*
Z = zscore(D)	*Estandariza las columnas de la matriz D*

Como primer ejemplo leemos el fichero *latent* y realizamos un análisis de componentes principales para la matriz *ingredients*.

```
>> load hald;
>> [pc,score,latent,tsquare] = princomp(ingredients)

pc =

    0.0678    0.6460   -0.5673    0.5062
    0.6785    0.0200    0.5440    0.4933
   -0.0290   -0.7553   -0.4036    0.5156
   -0.7309    0.1085    0.4684    0.4844

score =

  -36.8218    6.8709    4.5909    0.3967
  -29.6073   -4.6109    2.2476   -0.3958
   12.9818    4.2049   -0.9022   -1.1261
  -23.7147    6.6341   -1.8547   -0.3786
    0.5532    4.4617    6.0874    0.1424
   10.8125    3.6466   -0.9130   -0.1350
   32.5882   -8.9798    1.6063    0.0818
  -22.6064  -10.7259   -3.2365    0.3243
    9.2626   -8.9854    0.0169   -0.5437
    3.2840   14.1573   -7.0465    0.3405
   -9.2200  -12.3861   -3.4283    0.4352
   25.5849    2.7817    0.3867    0.4468
   26.9032    2.9310    2.4455    0.4116

latent =

  517.7969
   67.4964
   12.4054
    0.2372
tsquare =

    5.6803
    3.0758
    6.0002
    2.6198
    3.3681
    0.5668
```

```
    3.4818
    3.9794
    2.6086
    7.4818
    4.1830
    2.2327
    2.7216
```

Se obtiene la matriz con las cuatro componentes principales, sus puntuaciones, varianza explicada por cada componente y los valores de la *T*2 de Hotelling.

Podemos aplicar el test de Barlet para ver cuántas componentes principales se retendrán.

```
>> ndim = barttest(ingredients,0.05)

ndim =
     4
```

A continuación, a partir de la matriz de covarianzas de *ingredients*, calculamos las componentes principales, sus autovalores y el porcentaje de varianza explicado por cada componente.

```
>> load hald
covx = cov(ingredients);
[pc,variances,explained] = pcacov(covx)

pc =

   -0.0678    0.6460   -0.5673    0.5062
   -0.6785    0.0200    0.5440    0.4933
    0.0290   -0.7553   -0.4036    0.5156
    0.7309    0.1085    0.4684    0.4844

variances =

  517.7969
   67.4964
   12.4054
    0.2372

explained =

   86.5974
   11.2882
    2.0747
    0.0397
```

11.7 Contrastes de hipótesis

La variación aleatoria dificulta a menudo determinar si las muestras tomadas en distintas condiciones son realmente diferentes. La verificación de hipótesis es una herramienta efectiva para analizar si las diferencias entre muestras son significativas y requieren más evaluación o si son consistentes con la variación de datos aleatoria y esperada.

El Statistics Toolbox soporta los procedimientos de verificación de hipótesis paramétricos y no paramétricos más ampliamente usados, tal como test *T* de una y dos muestras, test *Z* de una muestra, pruebas no paramétricas para una muestra, pruebas no paramétricas para dos muestras independientes, tests de ajuste a una distribución (Jarque-Bera, Lilliefors, Kolmogorov-Smirnov) y tests de comparación de distribuciones (Kolmogorov-Smirnov de dos muestras). A continuación se presenta la sintaxis de las funciones adecuadas.

p = ranksum(x,y,alpha) **[p,h] = ranksum(x,y,alpha)** **[p,h,stats] = ranksum(x,y,alpha)**	*Test de Wilcoxon de suma de rangos de independencia de las muestras x e y al nivel alpha (probabilidad p de que, al nivel alpha, sean idénticas). Si h es cero, las muestras no son independientes, y si h=1, sí lo son. Se pueden obtener estadísticos del contraste (stats)*
p = signrank(x,y,alpha) **[p,h] = signrank(x,y,alpha)** **[p,h,stats] = signrank(x,y,alpha)**	*Test de Wilcoxon de signos-rangos para la igualdad de medianas de x e y*
p = signtest(x,y,alpha) **[p,h] = signtest(x,y,alpha)** **[p,h,stats] = signtest(x,y,alpha)**	*Test de los signos de igualdad de medianas para muestras pareadas*
h = ttest(x,m) **h = ttest(x,m,alpha)** **[h,sig,ci] = ttest(x,m,alpha,tail)**	*Test T para contrastar si la media de la muestra x (supuesta normal) vale m al nivel alpha. El valor de tail (0, -1 o 1) indica contraste bilateral, unilateral izquierdo y unilateral derecho, respectivamente.*
[h,significance,ci] = ttest2(x,y) **[h,significance,ci] = ttest2(x,y,alpha)** **[h,significance,ci] = ttest2(x,y,alpha,tail)**	*Contraste T para la igualdad de medias de x e y. Si h es 1 no hay igualdad, significance da el nivel de significación, ci da un intervalo de confianza para la diferencia de medias, alpha asigna el coeficiente de confianza y tail la forma del test (unilateral o bilateral)*
h = ztest(x,m,sigma) **h = ztest(x,m,sigma,alpha)** **[h,sig,ci,zval] = ztest(x,m,sigma,alpha,tail)**	*Contraste Z para la igualdad de medias de x e y con varianzas conocidas*

H = jbtest(X) **H = jbtest(X,alpha)** **[H,P,JBSTAT,CV] = jbtest(X,alpha)**	*Test de Jarque-Bera para la normalidad de X. Se puede introducir un nivel alpha a medida. Se puede obtener además el p,valor p del test, el valor del estadístico JBSTAT y el valor crítico. Si H=1 se rechaza la hipótesis de normalidad*
H = kstest(X) **H = kstest(X,cdf)** **H = kstest(X,cdf,alpha,tail)** **[H,P,KSSTAT,CV] = kstest(X,cdf,alpha,*tail*)**	*Contraste de normalidad de Kolmogorov-Smirnov para una muestra X* *Contraste de Kolmogorov-Smirnov de ajuste a la distribución cdf de la muestra X.* *Se puede dar nivel de confianza y bilateralidad. Se obtiene además de H, el p-valor p, el valor del estadístico y el valor crítico. Si H=1 se rechaza la hipótesis de ajuste a la distribución*
H = kstest2(X1,X2) **H = kstest2(X1,X2,alpha,tail)** **[H,P,KSSTAT] = kstest(X,cdf,alpha,tail)**	*Contraste de Kolmogorov-Smirnov para ver si dos muestras provienen de la misma distribución (son homogéneas), siendo posible fijar la distribución mediante cdf*
H = lillietest(X) **H = lillietest(X,alpha)** **[H,P,LSTAT,CV] = lillietest(X,alpha)**	*Contraste de Lilliefors para la normalidad de X al nivel alpha, pudiendo obtener el p-valor p, el valor del estadístico y el valor crítico. Si H=1 se rechaza la hipótesis de ajuste a la distribución*

Como primer ejemplo consideramos los números enteros entre -2 y 4 y comprobamos si pueden considerase provenientes de una distribución normal.

```
>> x = -2:1:4
x =
    -2    -1     0     1     2     3     4
>> [h,p,k,c] = kstest(x,[],0.05,0)
h =
     0
p =
    0.1363
k =
    0.4128
c =
    0.4834
```

Se observa que *H*=0 y que el valor del estadístico es menor que el valor crítico, con lo cual se acepta la hipótesis de muestra proveniente de una distribución normal.

En el ejemplo siguiente se realiza el contraste de los signos para comprobar que dos muestras aleatorias de tamaño 20 provenientes de normales (0,1) y (0,2) tienen la media media al 95% de confianza.

```
>> x = normrnd(0,1,20,1);
y = normrnd(0,2,20,1);
[p,h] = signtest(x,y,0.05)
p =
    0.2632
h =
     0
```

Como *H* es cero se acepta la igualdad de medias para ambas muestras.

11.8 Estadística industrial: control de procesos y diseño de experimentos

Statistics Toolbox proporciona una serie de funciones de uso fácil que soporta el control de procesos estadísticos (SPC). Estas funciones permiten controlar y mejorar productos o procesos ayudándole a evaluar la variabilidad del proceso. Además, las funciones de diseño de experimentos (DOE) ayudan a crear y verificar proyectos prácticos de recogida de datos para obtención de modelos estadísticos. Estos proyectos le muestran cómo manipular sus entradas de datos en tándem de modo que pueda obtener la información que necesita sobre su efecto en las salidas a mínimo coste. Statistics Toolbox soporta las siguientes áreas de estadística industrial: gráficos de control y estudios de capacidad para SPC y diseños de Hadamard y factoriales *D*-óptimos usados junto con DOE. Las funciones más importantes son las siguientes:

Control de procesos	
p = capable(data,specs) **[p,Cp,Cpk] = capable(data,specs)**	*Da la probabilidad de que una muestra quede fuera de las bandas de specs = [lower, upper]* *Calcula índices de capacidad de un proceso*
p = capaplot(data,specs)	*Estima media y varianza de data y grafica la función de densidad de la T resultante*
ewmaplot(data) **ewmaplot(data,lambda)** **ewmaplot(data,lambda,alpha)** **ewmaplot(data,lambda,alpha,specs)** **h = ewmaplot(...)**	*Realiza gráficos de control 3-sigmas de medias móviles exponenciales (EWMA) para data considerando cada una de sus filas como un subgrupo. El factor lambda introduce peso a las observaciones pasadas y alpha es el nivel de significación para límites probabilísticos*
histfit(data) **histfit(data,nbins)** **h = histfit(data,nbins)**	*Histograma con densidad normal superpuesta*
p = normspec(specs,mu,sigma) **[p,h] = normspec(specs,mu,sigma)**	*Grafica densidad normal entre límites*

schart(DATA,conf) **schart(DATA,conf,specs)** **schart(DATA,conf,specs)** **[outliers,h] =** **schart(DATA,conf,specs)**	*Gráfico de control S (3-sigmas) para los datos de la matriz DATA considerando cada una de sus filas como un subgrupo. El parámetro conf indica la confianza para límites probabilísticos (conf=0,9973 equivale a 3 sigmas)*
xbarplot(DATA) **xbarplot(DATA,conf)** **xbarplot(DATA,conf,specs,'*sigmaest*')** **[outlier,h] = xbarplot(...)**	*Realiza un gráfico X de medias (3-sigmas) para los datos de la matriz DATA considerando cada una de sus filas como un subgrupo. El parámetro conf indica la confianza para límites probabilísticos (conf=0,9973 equivale a 3 sigmas)*
Diseño de experimentos	
settings = cordexch(nfactors,nruns) **[settings,X] = cordexch(nfactors,nruns)** **[settings,X] =** **cordexch(nfactors,nruns,'model')**	*Diseño D-óptimo usando coordenadas de cambio y modelo lineal aditivo con término constante, donde la matriz de factores settings tiene nruns filas y nfactors columnas. También puede obtenerse la matriz X asociada al diseño*
settings = daugment(startdes,nruns) **[settings,X] =** **daugment(startdes,nruns,'*model*')**	*Aumenta el diseño D-óptimo inicial startdates con nruns nuevos tests*
settings = dcovary(factors,covariates) **[settings,X] =** **dcovary(factors,covariates,'*model*')**	*Diseño D-óptimo con covariantes fijados siendo factors el número de variables experimentales a controlar*
X = ff2n(n)	*Crea un diseño factorial completo con n columnas*
x = fracfact('gen') **[x,conf] = fracfact('gen')**	*Crea un diseño factorial fraccional de dos niveles para la cadena generadora gen (palabras separadas por espacios)*
design = fullfact(levels)	*Crea un diseño factorial completo mezclado. Cada elemento del vector levels indica el número de valores únicos en la columna correspondiente de design*
H = hadamard(n)	*Diseño de Hadamard de orden n*
settings = rowexch(nfactors,nruns) **[settings,X] = rowexch(nfactors,nruns)** **[settings,X] =** **rowexch(nfactors,nruns,'model')**	*Genera la matriz factorial settings para diseños D-óptimos con modelo lineal aditivo y término constante, donde settings tiene nruns filas and nfactors columnas. También puede obtenerse la matriz X asociada al diseño*

Como primer ejemplo creamos un diseño factorial completo con 2 niveles y con tres columnas en la matriz de diseño.

```
>> X = ff2n(3)

X =

     0     0     0
     0     0     1
     0     1     0
     0     1     1
     1     0     0
     1     0     1
     1     1     0
     1     1     1
```

En el ejemplo siguiente se muestra que un diseño *D*-óptimo para tres factores y 8 ejecuciones usando un modelo con interacción es un diseño factorial completo de dos niveles.

```
>> s = rowexch(3,8,'interaction')

s =

    -1     1    -1
     1    -1    -1
     1    -1     1
     1     1     1
    -1     1     1
     1     1    -1
    -1    -1     1
    -1    -1    -1
```

A continuación generamos un diseño factorial completo con 8 ejecuciones, 2 niveles en la primera columna y 4 en la segunda (cada elemento del vector de niveles indica el número de valores únicos en la columna correspondiente).

```
>> d = fullfact([2 4])

d =

     1     1
     2     1
     1     2
     2     2
     1     3
     2     3
     1     4
     2     4
```

En el ejemplo siguiente se calculan los índices de capacidad para una muestra aleatoria de tamaño 30 de una normal (1,1) y con límites de especificación [-3, 3].

```
>> data = normrnd(1,1,30,1);
[p,Cp,Cpk] = capable(data,[-3 3])

p =

    0.0158

Cp =

    1.0592

Cpk =

    0.7166
```

A continuación realizamos un gráfico de capacidad para el ejemplo anterior (Figura 11-9).

```
>> data = normrnd(1,1,30,1);
p = capaplot(data,[-3 3])

p =

    0.9978
```

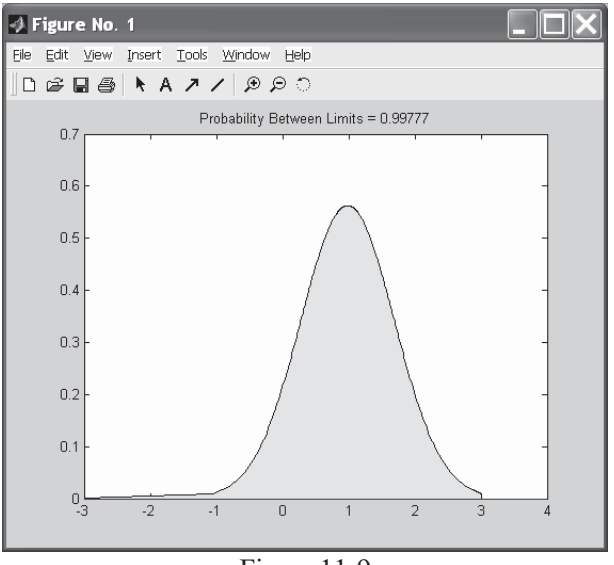


Figura 11-9

A continuación realizamos un gráfico de control EWMA para un proceso con cambio lento en la media (Figura 11-10).

```
>> t = (1:28)';
r = normrnd(10+0.02*t(:,ones(4,1)),0.5);
ewmaplot(r,0.4,0.01,[9.75 10.75])
```

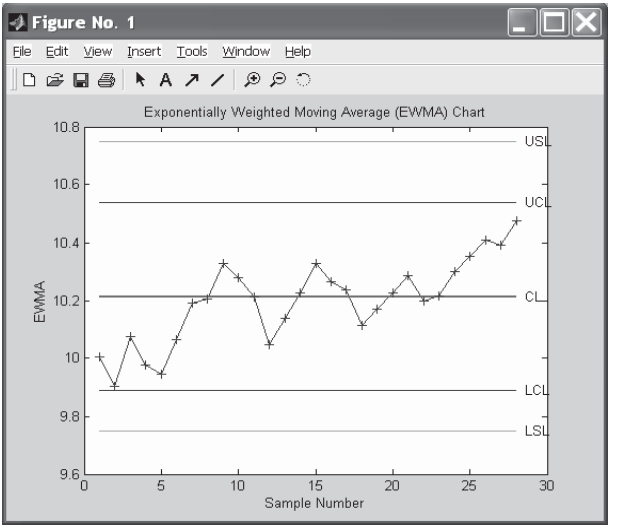


Figura 11-10

En el ejemplo siguiente se construye un histograma con campana de Gauss para una muestra de 100 valores aleatorios según una normal (10,1).

```
>> r = normrnd(10,1,100,1);
histfit(r)
```

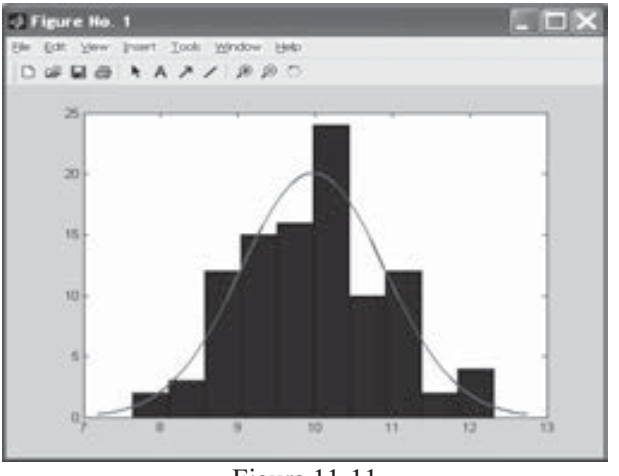


Figura 11-11

Ejercicio 11-1. Queremos ejecutar un experimento que estudie los efectos de 4 factores sobre una variable respuesta pudiendo permitirse sólo 8 ejecuciones. Una ejecución consiste en llevar a cabo el experimento para una combinación específica de valores de factor. Nuestra meta es determinar qué factores afectan a la respuesta pudiendo existir interacciones entre cada par de factores.

Si consideramos todas las posibles combinaciones de los 4 factores tendremos 16 ejecuciones.

```
>> x = fracfact('a b c d')

x =

     -1     -1     -1     -1
     -1     -1     -1      1
     -1     -1      1     -1
     -1     -1      1      1
     -1      1     -1     -1
     -1      1     -1      1
     -1      1      1     -1
     -1      1      1      1
      1     -1     -1     -1
      1     -1     -1      1
      1     -1      1     -1
      1     -1      1      1
      1      1     -1     -1
      1      1     -1      1
      1      1      1     -1
      1      1      1      1
```

Una forma de obtener sólo 8 ejecuciones es considerar que no hay interacción entre tres factores.

```
>> [x,conf] = fracfact('a b c abc')

x =

     -1     -1     -1     -1
     -1     -1      1      1
     -1      1     -1      1
     -1      1      1     -1
      1     -1     -1      1
      1     -1      1     -1
      1      1     -1     -1
      1      1      1      1
```

```
conf =

    'Term'      'Generator'     'Confounding'
    'X1'        'a'             'X1'
    'X2'        'b'             'X2'
    'X3'        'c'             'X3'
    'X4'        'abc'           'X4'
    'X1*X2'     'ab'            'X1*X2 + X3*X4'
    'X1*X3'     'ac'            'X1*X3 + X2*X4'
    'X1*X4'     'bc'            'X1*X4 + X2*X3'
    'X2*X3'     'bc'            'X1*X4 + X2*X3'
    'X2*X4'     'ac'            'X1*X3 + X2*X4'
    'X3*X4'     'ab'            'X1*X2 + X3*X4'
```

Las tres primeras columnas de la matriz *x* forman un diseño factorial completo y la cuarta columna es el producto de las otras tres. El patrón de confusión muestra que los efectos principales para los cuatro factores son estimables, pero las iteraciones de dos factores no lo son. Por ejemplo, las interacciones `X1*X2` y `X3*X4'` se confunden y es imposible estimar sus efectos separadamente.

Ejercicio 11-2. El número de pétalos de 13 flores de una determinada especie es el siguiente: 8, 10, 6, 5, 8, 11, 8, 10, 7, 10, 7, 10 y 9. Calcular la media, la varianza y el coeficiente de variación.

Definimos el vector *V* cuyos componentes son los números de pétalos y a continuación calculamos la media varianza y coeficiente de variación con las funciones adecuadas de MATLAB.

```
>> V=[8, 10, 6, 5, 8, 11, 8, 10, 7, 10, 7, 10, 9];
>> Media=mean(V)

Media =
    8.3846

>> Varianza=var(V,1)

Varianza =
    3.0059

>> Coeficiente_de_variacion=std(V,1)/mean(V)

Coeficiente_de_variacion =
    0.2068
```

El coeficiente de variación se ha calculado como cociente entre la desviación típica y la media.

Ejercicio 11-3. Sea una variable aleatoria Y cuya función de densidad está dada por f(y) = 6y (1-y) si 0 < y <1, y f(y) = 0 en otro caso. Calcular P(0,5 < Y < 8).

La variable aleatoria *Y* se ajusta a una distribución beta de parámetros *z*=2, *w*=2, ya que:

$$\frac{1}{\beta(z,w)} = 6$$

```
>> 1/beta(2,2)

ans =

    6.0000
```

. El problema nos pide:

$$P(0{,}5 < Y < 0{,}8) = P(Y < 0{,}8) - P(Y \le 0{,}5) = I_{0,8}(2,2) - I_{0,5}(2,2)$$

Este valor se puede calcular utilizando la función *betainc* de MATLAB como sigue:

```
>> betacdf(0.8,2,2)-betacdf(0.5,2,2)

ans =

    0.3960
```

Ejercicio 11-4. Con los datos de la economía española correspondientes al Producto Interior Bruto a precios de mercado en pesetas constantes de 1980 que se presentan a continuación:

1970	*10.595,1*	*1977*	*14.685,4*
1971	*11.111,5*	*1978*	*14.934,4*
1972	*11.948,6*	*1979*	*15.003,1*
1973	*12.948,0*	*1980*	*15.209,1*
1974	*13.715,1*	*1981*	*15.195,9*
1975	*13.803,4*	*1982*	*15.379,2*
1976	*14.184,7*	*1983*	*15.679,7*

ajustar un modelo lineal que explique el PIB en función del tiempo y predecir el valor del PIB en el año 2000. Realizar también un ajuste cuadrático y cúbico en función de t.

Introducimos las variables como dos vectores fila en MATLAB como sigue:

```
>> PIBpmt=[10591.1, 11111.5, 11948.6, 12948, 13715.1, 13803.4,
           14184.7, 14685.4, 14934.4, 15000.31, 15209.1, 15195.9,
           15379.2, 15679.7];

>> Tiempo=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14];
```

```
>> [p,S] = polyfit(Tiempo,PIBpmt,1)

p =

  1.0e+004 *

    0.0370    1.1108

S =
        R: [2x2 double]
       df: 12
    normr: 2.0043e+003
```

El modelo ajustado resulta ser $PIBpmt = 370\ t + 11.108$.

El valor de *PIBpmt* predicho para el año 2000 corresponde a $t = 31$ y se calcula como sigue:

```
>> polyconf(p,31,S,0.05)

ans =

  2.2586e+004
```

Los ajustes y predicciones para polinomios cuadráticos y cúbicos se hallan como sigue:

```
>> [p,S] = polyfit(Tiempo,PIBpmt,2)

p =
  1.0e+003 *

   -0.0349    0.8933    9.7129

S =
        R: [3x3 double]
       df: 11
    normr: 690.0013
```

El modelo cuadrático ajustado resulta ser $PIBpmt = -34{,}9\ t^2 + 893{,}3\ t + 9712{,}9$ y la predicción para el 2000 será:

```
>> polyconf(p,31,S,0.05)

ans =
  3.8950e+003
```

```
>> [p,S] = polyfit(Tiempo,PIBpmt,3)

p =

  1.0e+003 *

    0.0022   -0.0845    1.2018    9.2625

S =

        R: [4x4 double]
       df: 10
    normr: 552.6807
```

El modelo cúbico ajustado resulta ser $PIBpmt = 2{,}2\ t^3 - 84{,}5\ t^2 + 1201{,}8\ t + 9262{,}5$ y la predicción para el 2000 será:

```
>> polyconf(p,31,S,0.05)

ans =

  3.1041e+004
```

Se han utilizado predicciones al 95% de confianza.

Ejercicio 11-5. Un agente de seguros vende pólizas a 5 individuos, todos de la misma edad. De acuerdo con las tablas actuariales, la probabilidad de que un individuo con esa edad viva 30 años más es de 3/5. Determinar la probabilidad de que dentro de 30 años vivan:

a) al menos 3 individuos;
c) como mucho, 2.

Como quiera que dentro de 30 años la circunstancia de cada individuo será que viva o que no viva, y al menos una de las dos se ha de presentar, la situación para cada individuo se ajusta a una variable de Bernoulli con probabilidad de éxito (vivir 30 años más) igual a 3/5 = 0,6. Al considerar los 5 individuos, estamos ante una variable aleatoria X binomial con $n = 5$ y $p = 0.6$, $X = B(5, 0{,}6)$. Si llamamos $F(x)$ a la función de distribución de $X = B(5, 0{,}6)$ en el punto x, los apartados del problema se calculan como sigue:

a) Habrá que calcular $P(X \geq 3)$ o, lo que es lo mismo, $1\text{-}p(X < 3) = 1\text{-}F(2)$.
b) Habrá que calcular $P(X \leq 2)$ o, lo que es lo mismo, $F(2)$.

Estos valores se calculan en MATLAB de la siguiente forma:

```
>> [1-binocdf(2,5,0.6), binocdf(2,5,0.6)]

ans =

    0.6826    0.3174
```

Ejercicio 11-6. El número medio de automóviles que llega a una estación de suministro de gasolina es de 210 por hora. Si dicha estación puede atender a un máximo de 10 automóviles por minuto, determinar la probabilidad de que en un minuto dado lleguen a la estación de suministro más automóviles de los que puede atender.

El número aleatorio de automóviles que llegan a la estación de servicio en un minuto puede representarse por una variable X de Poisson de parámetro $m = 210/60 = 3{,}5$, ya que m es el número medio de llegadas por minuto (teníamos 210 llegadas a la hora).

La probabilidad que vamos a obtener vendrá dada por $P(X > 10)$, ya que para que lleguen a la estación más automóviles por minuto de los que se puedan atender es necesario que lleguen más de 10 por minuto. Pero $P(X > 10) = 1 - P(X \leq 10) = 1 - F(10)$, siendo F la función de distribución de una variable aleatoria de Poisson de parámetro 3,5.

Para calcular la probabilidad pedida tendremos en cuenta que:

$$1 - F(10) = 1 - \sum_{k=0}^{10} \frac{e^{-3,5}\, 3,5^k}{k!}$$

que puede calcularse en MATLAB como sigue:

```
>> 1-poisscdf(10,3.5)

ans =

    0.0010
```

Ejercicio 11-7. En un examen final de estadística, los estudiantes recibieron las siguientes calificaciones:

80, 70, 90, 75, 55, 80, 80, 65, 100, 75, 60, 60,75, 95, 80, 80, 90, 85, 70, 95, 75, 70, 85, 80, 80, 65, 65, 50, 75, 75, 85, 85, 90, 70.

Compruébese si las calificaciones fueron o no distribuidas según una ley normal a un nivel 0,05.

Vamos a utilizar el contraste de normalidad de Jarque Bera.

```
>> x=[80, 70, 90, 75, 55, 80, 80, 65, 100, 75, 60, 60,75, 95,
      80, 80, 90, 85, 70, 95, 75, 70, 85, 80, 80, 65, 65, 50,
      75, 75, 85, 85, 90, 70];
>> [H,P,JBSTAT,CV] = jbtest(x,0.05)

H =

     0

P =

    0.7780

JBSTAT =

    0.5021

CV =

    5.9915
```

Se observa que el valor del estadístico 0, 5021 resulta menor que el valor crítico 5,9915, y que por lo tanto cae fuera de la región crítica. Esto nos lleva a aceptar la hipótesis nula de que las calificaciones se distribuyen según una ley normal. Además, H=0, lo que corrobora la aceptación de la hipótesis nula de ajuste de los datos a una normal. También puede verse el resultado de forma gráfica (Figura 11-12) mediante la sintaxis siguiente:

```
>> histfit(x)
```

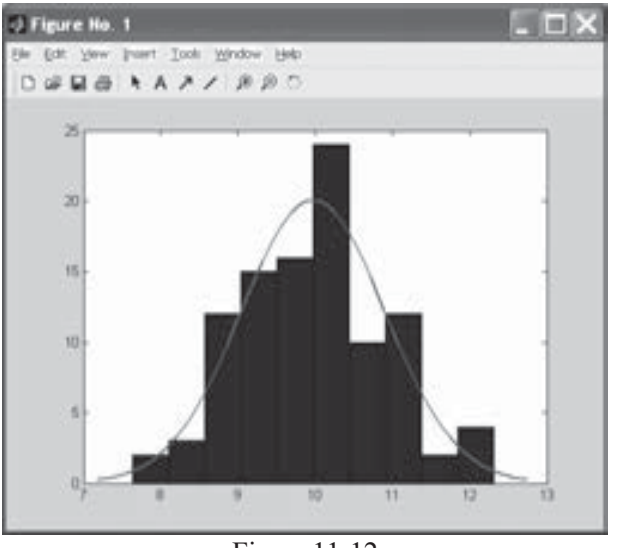


Figura 11-12

Ejercicio 11-8. Una central de productos lácteos recibe diariamente la leche de dos granjas, X e Y. Con el fin de estudiar la calidad de los productos recibidos, se extraen dos muestras, una de cada granja, y se analiza el contenido de materia grasa, obteniendo los siguientes resultados:

X→ 0.32, 0.29, 0.30, 0.28, 0.33, 0.31, 0.30, 0.29, 0.33, 0.32, 0.30, 0.29.
Y→ 0.28, 0.30, 0.32, 0.29, 0.31, 0.29, 0.33, 0.32, 0.29, 0.32, 0.31, 0.29, 0.32, 0.31, 0.32, 0.33.

Realizar el contraste de hipótesis de homogeneidad de calidades.

Realizamos un contraste de homogeneidad, es decir, se contrasta si las dos muestras provienen de la misma distribución. Se utiliza el test de sumas rangos de Wilcoxon mediante la sintaxis siguiente:

```
>> x=[0.32, 0.29, 0.30, 0.28, 0.33, 0.31, 0.30, 0.29, 0.33, 0.32,
      0.30, 0.29];
>> y=[0.28, 0.30, 0.32, 0.29, 0.31, 0.29, 0.33, 0.32, 0.29, 0.32,
      0.31, 0.29, 0.32, 0.31, 0.32, 0.33];
>> [p,h] = ranksum(x,y,0.05)

p =

    0.6528

h =

     0
```

Como $h = 0$ y p alta se acepta la homogeneidad de poblaciones, es decir, ambas muestras provienen de la misma población al 95% de confianza. Lógicamente los contrastes de igualdad de medianas y de igualdad de medias deberán aceptar ambas hipótesis.

Ejercicio 11-9. Las presiones críticas de dos grupos independientes de recipientes de distintos vidrios dan los siguientes valores:

Grupo 1º: 100, 102, 96, 106, 110, 110, 120, 112, 112, 90
Grupo 2º: 104, 88, 100, 98, 102, 92, 96, 100, 96, 96

a) Suponiendo que las dos poblaciones son normales y de varianzas iguales y desconocidas, contrastar la hipótesis de igualdad de medias al nivel 0,05.
c) Realizar el contraste para igualdad de medianas.

```
>> a=[100, 102, 96, 106, 110, 110, 120, 112, 112, 90];
>> b=[104, 88, 100, 98, 102, 92, 96, 100, 96, 96];
>> [p,h] = ttest2(a,b,0.05)

p =

     1

h =

    0.0145
```

Los resultados del test de la *T* de Student para dos muestran indican la aceptación de igualdad de medias (*p* muy alto y *h* casi 0).

```
>> [p,h] = signtest(a,b,0.05)

p =

    0.3438

h =

     0
```

El resultado del test de los signos indica la igualdad de medianas.

Ejercicio 11-10. A continuación, se presenta información de la economía española correspondiente al período 1964-1980, sobre tres variables macroeconómicas: importaciones energéticas reales (IMPEN), producto interior bruto a precios de mercado (PIB) (ambas en miles de millones de pesetas constantes de 1970) y el precio relativo de las importaciones energética (PREN):

AÑO	*IMPEN*	*PIB*	*PREN*
1964	*18,9*	*1791,8*	*136,07*
1965	*20,1*	*1905,3*	*123,43*
1966	*24,0*	*2040,0*	*109,21*
1967	*30,2*	*2127,9*	*101,41*
1968	*39,6*	*2272,0*	*106,80*
1969	*38,2*	*2475,2*	*104,17*
1970	*44,1*	*2576,2*	*100,00*
1971	*49,4*	*2703,8*	*106,76*
1972	*54,2*	*2923,9*	*99,06*
1973	*57,3*	*3153,6*	*96,96*
1974	*64,0*	*3333,9*	*230,72*
1975	*59,7*	*3370,5*	*225,48*
1976	*63,2*	*3472,0*	*259,36*
1977	*58,0*	*3586,5*	*258,11*
1978	*56,4*	*3650,9*	*234,23*
1979	*63,8*	*3657,7*	*224,79*
1980	*62,7*	*3714,2*	*367,47*
1981	*59,1*	*3730,7*	*458,62*
1982	*57,0*	*3763,5*	*457,01*
1983	*59,4*	*3842,6*	*476,54*
1984	*58,0*	*3921,2*	*455,26*
1985	*59,9*	*3999,6*	*420,17*

a) Estimar el modelo de regresión que pretende explicar el volumen de importaciones energéticas en función de las variaciones del PIB y el precio relativo de las importaciones.
b) Predecir el volumen de importaciones previsto para un PIB de valor 4.000 y un precio relativo de las importaciones de valor 300.

Comenzamos introduciendo los datos completos como una matriz en MATLAB:

```
>> X=[1964   18.9   1791.8 136.07
      1965   20.1   1905.3 123.43
      1966   24.0   2040.0 109.21
      1967   30.2   2127.9 101.41
      1968   39.6   2272.0 106.80
      1969   38.2   2475.2 104.17
      1970   44.1   2576.2 100.00
      1971   49.4   2703.8 106.76
      1972   54.2   2923.9 99.06
      1973   57.3   3153.6 96.96
      1974   64.0   3333.9 230.72
      1975   59.7   3370.5 225.48
      1976   63.2   3472.0 259.36
      1977   58.0   3586.5 258.11
      1978   56.4   3650.9 234.23
      1979   63.8   3657.7 224.79
      1980   62.7   3714.2 367.47
      1981   59.1   3730.7 458.62
      1982   57.0   3763.5 457.01
      1983   59.4   3842.6 476.54
      1984   58.0   3921.2 455.26
      1985   59.9   3999.6 420.17
      ];
```

A continuación extraemos las 3 últimas columnas como variables individuales:

```
>> IMPEN=X(:,2);
>> PIB=X(:,3);
>> PREN=X(:,4)
```

Ahora ya estamos en condiciones de efectuar la regresión múltiple obteniendo además intervalos de confianza al 95% para los parámetros estimados.

```
>> [b,bint] = regress(IMPEN,[ones(22,1),PIB,PREN],0.05)

b =

  -21.0283
    0.0267
   -0.0504

bint =

  -30.8012  -11.2554
    0.0224    0.0311
   -0.0725   -0.0282
```

Según el valor de *b*, el modelo ajustado es:

$$IMPEN = -21{,}0283 + 0{,}0267*PIB - 0{,}0504*PREN$$

Los intervalos de confianza para cada parámetro son las tres filas de *bint*.

La predicción pedida en el problema para *PIB = 4.000* y *PREN=300* será:

$$IMPEN = -21{,}0283 + 0{,}0267*4.000 - 0{,}0504*300 = 70{,}85$$

12

Sistemas de control

12.1 Introducción a los sistemas de control

MATLAB permite trabajar mediante un entorno integrado en el diseño de sistemas de control. Según se indica en la Figura 12-1, en un problema de ingeniería se parte de datos experimentales y, mediante las herramientas de modelización, análisis y visualización de datos, se llega a un modelo de comportamiento que nos conduce al diseño y análisis del sistema de control. Posteriormente, previa simulación y generación de código, se puede realizar prototipaje rápido e implementación del código para obtener el sistema.

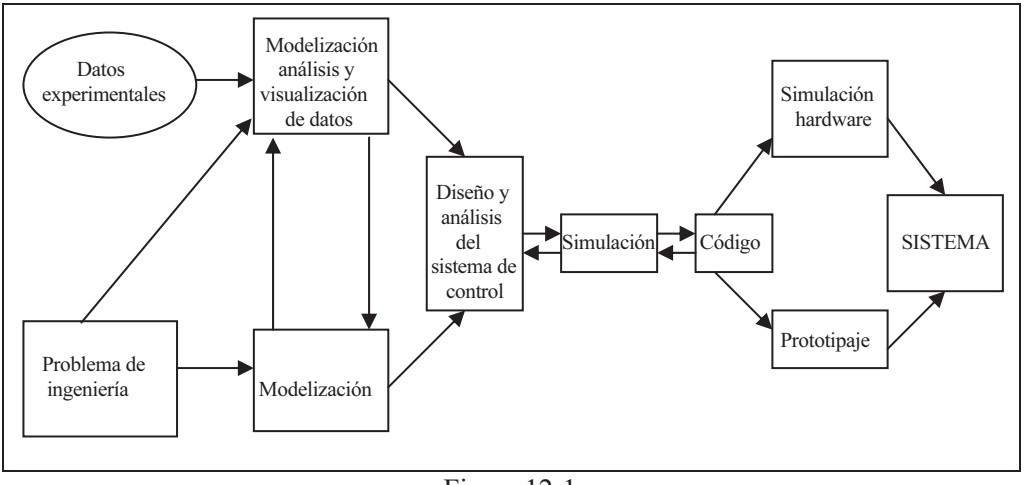


Figura 12-1

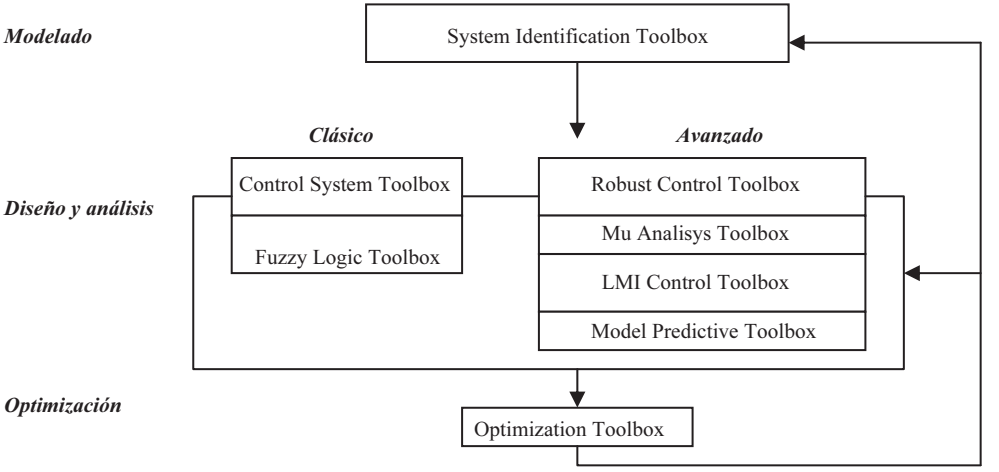
MATLAB proporciona una plataforma de cálculo técnico superior para obtención de modelos, análisis de datos y desarrollo de algoritmos. MATLAB combina una amplia funcionalidad de ingeniería y matemática incorporada con unas potentes prestaciones de visualización y animación, todo dentro de un lenguaje de programación interactivo y de alto nivel. Los toolboxes de MATLAB amplían el entorno MATLAB en una extensa gama de técnicas clásicas y modernas de diseño de controles, proporcionando algoritmos de control de vanguardia elaborados por expertos de reputación internacional.

MATLAB contiene más de 600 funciones matemáticas, estadísticas y de ingeniería, proporcionando la potencia de cálculo numérico que necesita para analizar datos, desarrollar algoritmos y optimizar el rendimiento del sistema. Con MATLAB puede ejecutar rápidamente repeticiones de diseños y comparar el rendimiento entre estrategias de control alternativas. Además, MATLAB es un lenguaje de programación de alto nivel que permite desarrollar algoritmos en una fracción del tiempo empleado en *C*, *C++* o Fortran. Ya que MATLAB es abierto y extensible, se puede ver el código fuente, modificar algoritmos e incorporar programas *C*, *C++* y Fortran existentes.

Por otra parte, las herramientas interactivas de *Control System Toolbox* facilitan el diseño y el ajuste de los sistemas de control. Por ejemplo, es posible arrastrar polos y ceros y ver inmediatamente cómo reacciona el sistema (Figura 12-2). Además, MATLAB proporciona potentes prestaciones interactivas de representación gráfica 2-D y 3-D que permiten visualizar datos, ecuaciones y resultados (Figura 12-3). Es posible usar una amplia gama de prestaciones de visualización de MATLAB o aprovechar las funciones específicas de control que se facilitan en los toolboxes de MATLAB.

Los toolboxes de MATLAB incluyen la funcionalidad específica de la aplicación escrita en lenguaje MATLAB. Los toolboxes de MATLAB relacionados con controles abarcan prácticamente cualquier técnica fundamental de diseño de controles, desde LQG y Root-locus a H y lógica difusa. Por ejemplo, es posible añadir un control de lógica difusa al diseño de un sistema usando los algoritmos incorporados del *Fuzzy Logic Toolbox* (Figura 12-4).

Los toolboxes más importantes de MATLAB relacionados con control de sistemas pueden clasificarse en tres familias: productos de modelado (*System Identification Toolbox*), productos de diseño y análisis clásico (*Control System Toolbox* y *Fuzzy Logic Toolbox*), productos de diseño y análisis avanzado (*Robust Control Toolbox, Mu Analisys toolbox, LMI Control Toolbox* y *Model Predictive Toolbox*) y productos de optimización (*Optimization Toolbox*). El esquema siguiente ilustra esta clasificación.


Modelado
System Identification Toolbox
Clásico
Avanzado
Diseño y análisis
Control System Toolbox
Fuzzy Logic Toolbox
Robust Control Toolbox
Mu Analisys Toolbox
LMI Control Toolbox
Model Predictive Toolbox
Optimización
Optimization Toolbox

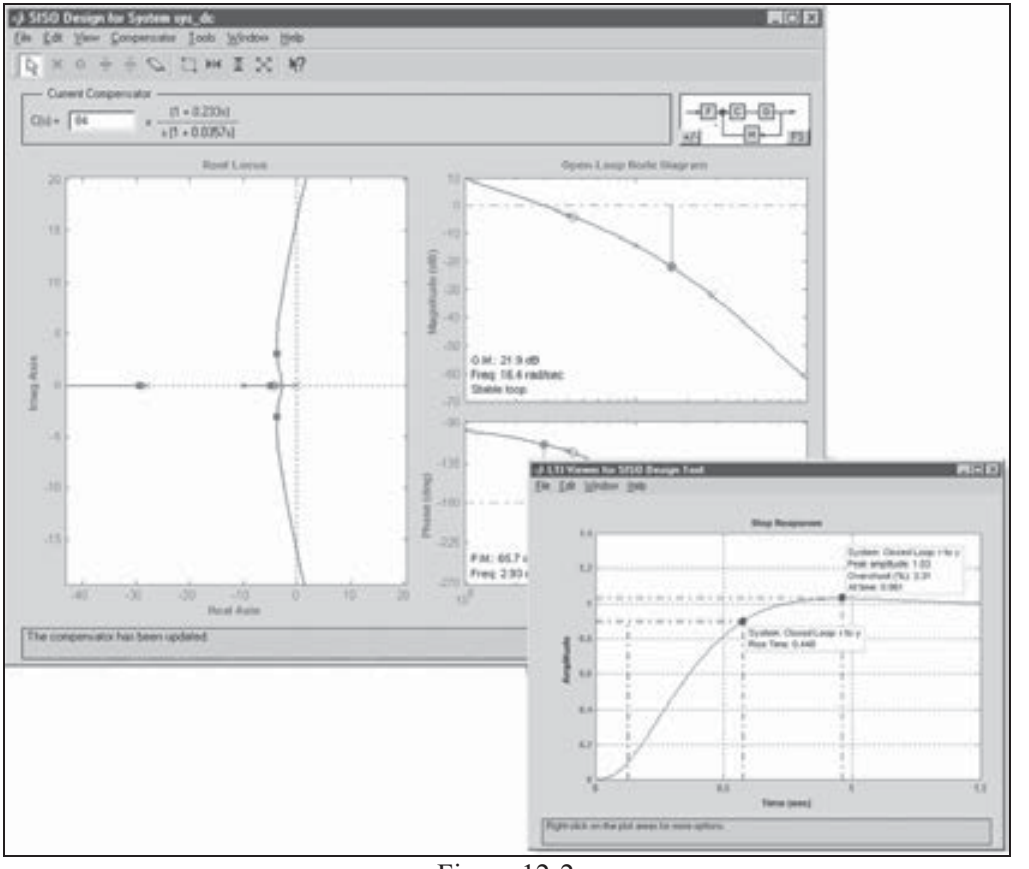


Figura 12-2

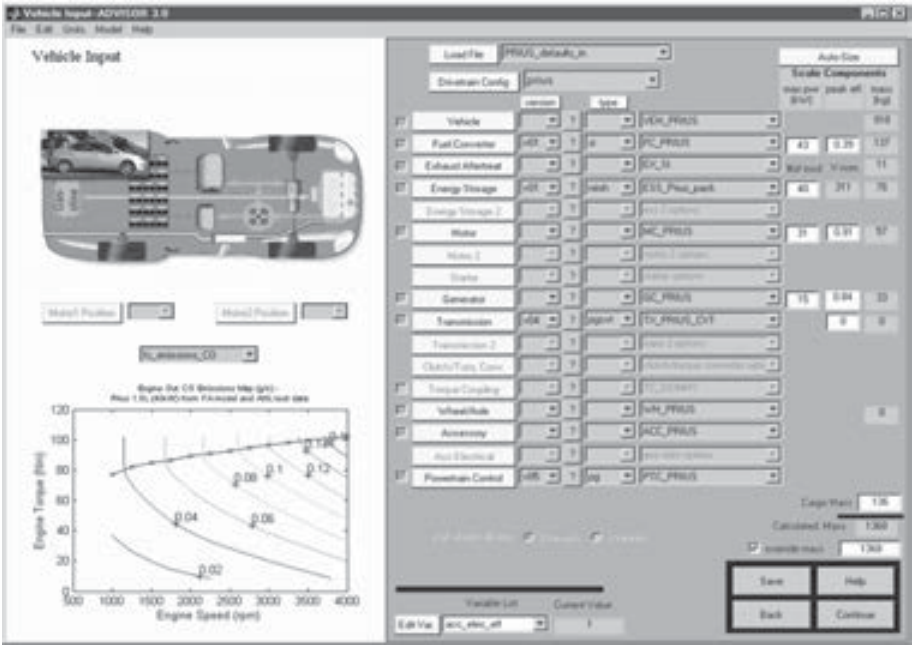


Figura 12-3

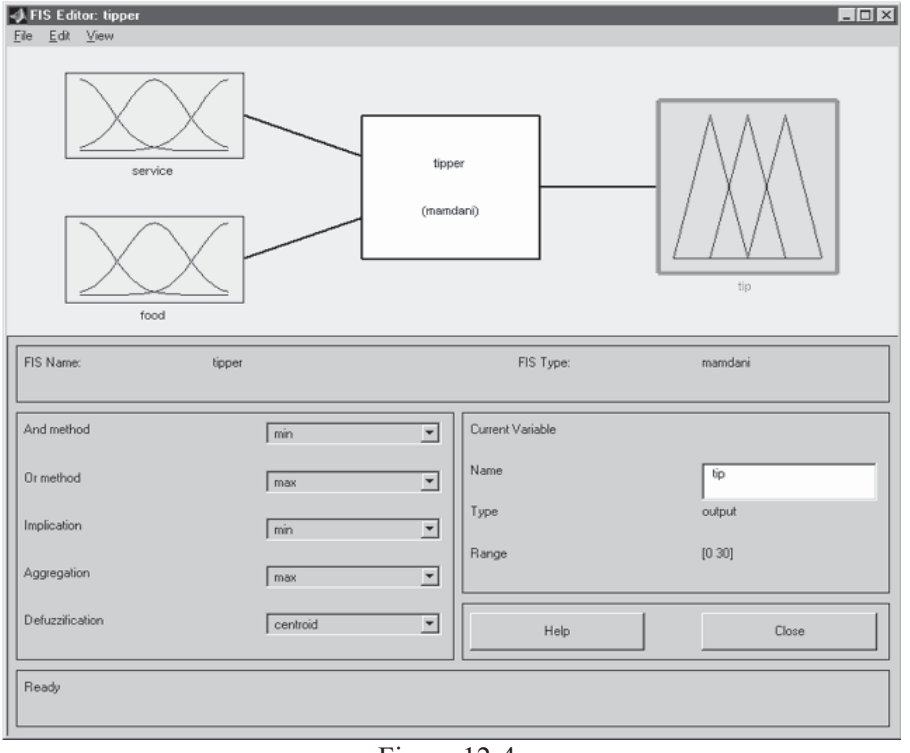


Figura 12-4

12.2 Diseño y análisis de sistemas de control: Control System Toolbox

Control System Toolbox es una colección de algoritmos que implementan técnicas comunes de diseño, análisis y obtención de modelos de sistemas de control. Su amplia gama de prestaciones comprende métodos clásicos y modernos de diseño de controles, incluido lugar geométrico de las raíces, colocación de polos y diseño de reguladores LQG. Determinadas interfaces gráficas de usuario apropiadas simplifican las tareas típicas de la ingeniería de control. Este toolbox se construye sobre los fundamentos de MATLAB para facilitar herramientas especializadas para ingeniería de sistemas de control. El toolbox es una colección de algoritmos, escritos principalmente como ficheros .M, que ejecutan técnicas comunes de diseño, análisis y creación de modelos de sistemas de control.

Con Control System Toolbox puede crear modelos de sistemas lineales invariantes en el tiempo (LTI) como función de transferencia, cero/polo/amplificación o forma de espacio de estados. Puede manipular sistemas de tiempo tanto discreto como continuo y hacer conversiones entre varias representaciones de modelos. Puede calcular y representar gráficamente respuestas de tiempo, respuestas de frecuencia y lugares geométricos de raíces. Otras funciones le permiten realizar colocación de polos, control óptimo y estimaciones. El Control System Toolbox es abierto y ampliable, permitiéndole crear ficheros .M personalizados para adecuarse a su aplicación concreta.

Son características clave de Control System Toolbox las siguientes:

- *LTI Viewer*: GUI interactiva para analizar y comparar sistemas LTI.
- *SISO Design Tool*: GUI interactiva para analizar y ajustar sistemas de control feedback de entrada única/salida única (SISO).
- *GUI Suite*: Ajuste de preferencias y propiedades, para conseguir un control completo sobre la visualización de los plots de tiempo y frecuencia.
- *Objetos LTI*: Estructuras especializadas de datos para representar de manera concisa datos de modelos en formatos de función de transferencia, espacio de estado, ceros/polos/ganancia y de respuesta en frecuencia.
- MIMO: Soporte para sistemas de entrada múltiple/salida múltiple, sistemas de datos muestreados y tiempo continuo y sistemas con retraso temporal.

- *Funciones y operadores para conectar modelos LTI*: Con diagramas de bloques complejos (conexiones en serie, paralelo y realimentación).

- Soporte para varios métodos de conversión discreto a continuo.

- Funciones para representar gráficamente las respuestas en tiempo y en frecuencia de sistemas y comparar varios sistemas con un único comando.

- Herramientas para técnicas clásicas y modernas de diseño de controles, incluido lugar geométrico de raíces, *loop shaping*, colocación de polos y regulación LQR/LQG

Construcción de modelos

El Control System Toolbox soporta representaciones de cuatro modelos lineales: modelos de espacio de estados (SS), funciones de transferencia (TF), modelos de ceros/polos/ganancias (ZPK) y modelos de datos de en frecuencia (FRD). Para cada tipo de modelo se facilitan objetos LTI. Además de los datos de modelo, los objetos LTI pueden almacenar el tiempo de muestreo de sistemas de tiempo discreto, retrasos, nombres de entradas y salidas, notas sobre el modelo y otras más. Usando objetos LTI, puede manipular modelos como entidades únicas y combinarlos usando operaciones de tipo matricial. En la Figura 12-5 se presenta un ejemplo ilustrativo del diseño de un regulador LQG simple. El extracto de código situado en la parte inferior muestra cómo está diseñado el controlador y cómo se ha creado el sistema de bucle cerrado. El plot de respuesta en la frecuencia muestra una comparación entre el sistema de bucle abierto (rojo) y el de bucle cerrado (azul).

El Control System Toolbox contiene comandos para consultar características del modelo tales como dimensiones de E/S, polos, ceros y ganancias DC. Estos comandos se aplican tanto a modelos de tiempo continuo como discreto. También existen comandos que permiten consultar características del modelo tales como dimensiones de E/S, polos, ceros y ganancias DC. Estos comandos se aplican tanto a modelos de tiempo continuo como discreto.

Análisis y diseño

Algunas tareas se prestan por sí mismas a la manipulación gráfica, mientras que otras se benefician de la flexibilidad de la línea de comandos. El Control System Toolbox está diseñado para alojar ambos enfoques, proporcionando GUI y una serie completa de funciones de línea de comando para diseño y análisis de modelos.

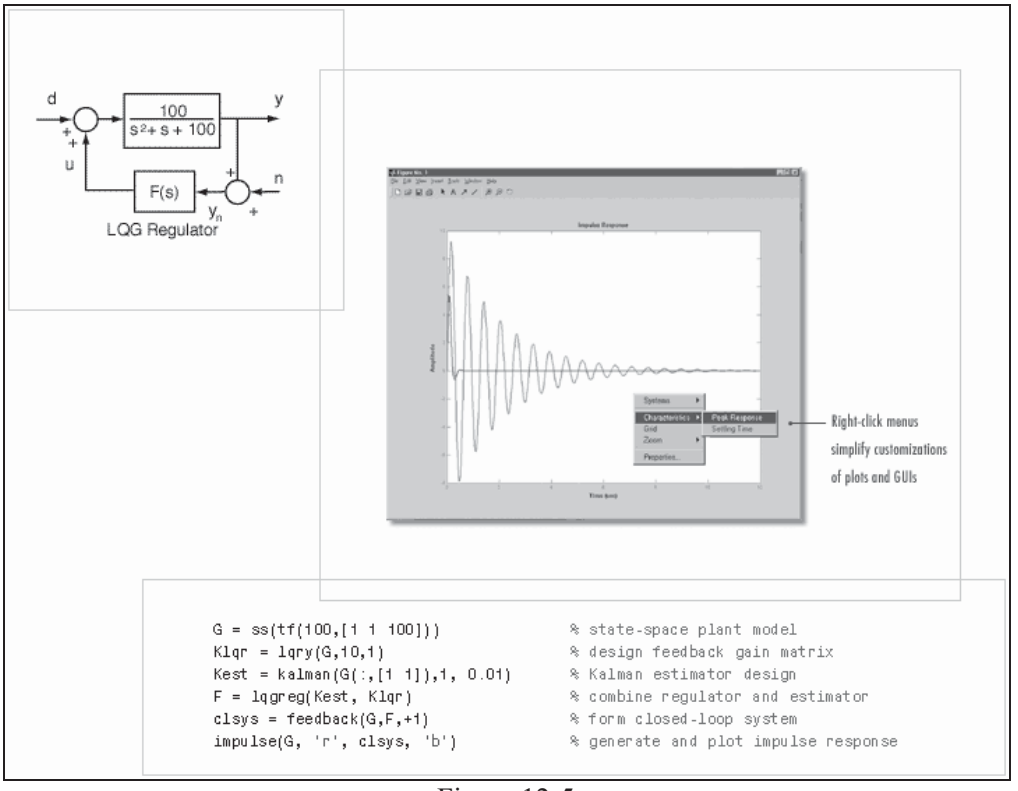


Figura 12-5

Análisis gráfico de modelos usando el LTI Viewer

El Control System Toolbox LTI Viewer es una GUI que simplifica el análisis de sistemas lineales invariantes en el tiempo (se carga escribiendo `>> ltiview` en la ventana de comandos). Se utiliza LTI Viewer para ver y comparar al mismo tiempo los plots de respuesta de varios modelos lineales. Es posible generar plots de respuesta al tiempo y la frecuencia para inspeccionar parámetros de respuesta clave tales como tiempo de subida, sobrepasamiento máximo y márgenes de estabilidad. Usando interacciones guiadas por ratón, puede seleccionar canales de entrada y salida desde sistemas MIMO. El LTI Viewer puede mostrar simultáneamente hasta seis tipos diferentes de plots incluido escalón, impulso, *Bode* (magnitud y fase o sólo magnitud), *Nyquist, Nichols*, sigma, y polo/cero. Usando las opciones de menú del botón derecho del ratón puede acceder a varios controles y opciones de LTI Viewer, incluyendo:

- *Plot Type:* Cambia el tipo de plot.

- *Systems:* Selecciona o deja de seleccionar cualquiera de los modelos cargados en LTI Viewer .

- *Characteristics:* Muestra parámetros y características de respuesta claves.
- *Zoom:* Ampliación y reducción de regiones del plot.
- *Grid:* Añade cuadrículas a sus plots.
- *Properties:* Abre el Property Editor, donde puede personalizar atributos del plot.

Además de los menús del pulsador derecho, todos los plots de respuesta incluyen marcadores de datos. Éstos le permiten escanear los datos del plot, identificar datos clave y determinar el sistema fuente para un plot dado. Con el LTI Viewer se pueden representar gráficamente con facilidad las respuestas de uno o varios sistemas todo en una ventana plots de escalón e impulso, plots de polos/ceros y todas las respuestas en el dominio de la frecuencia (*Bode, Nyquist, Nichols* y valores singulares). El LTI Viewer permite mostrar características de respuesta importantes, tales como márgenes de estabilidad en los plots usando marcadores de datos. Puede representar gráficamente con facilidad las respuestas de uno o varios sistemas (Figura 12-6).

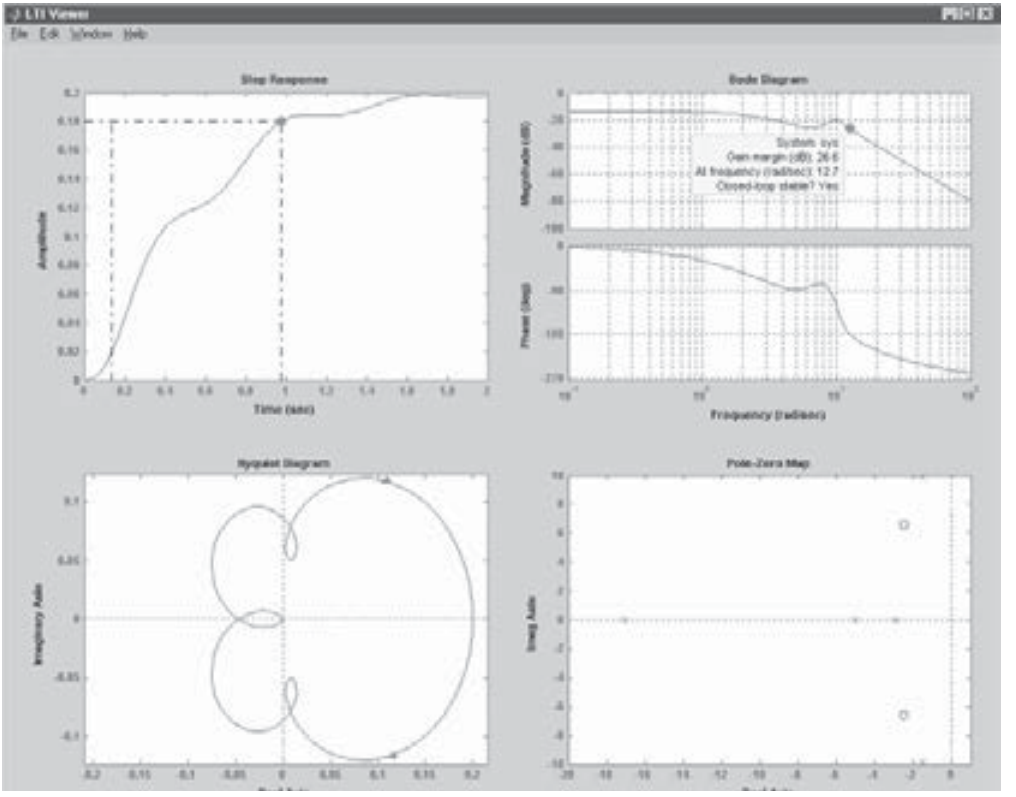


Figura 12-6

Análisis de modelos usando la línea de comandos

El LTI Viewer es adecuado para una amplia gama de aplicaciones donde se desea un entorno dirigido por GUI. Para situaciones que requieren programación, plots personalizados o la inclusión de datos no relacionados con sus modelos LTI, el Control System Toolbox proporciona funciones de línea de comando que realizan los plots básicos para el análisis de dominio de frecuencia y tiempo usados en la ingeniería de sistemas de control. Estas funciones se aplican a cualquier tipo de modelo lineal (continuo o discontinuo, SISO o MIMO) o arrays de modelos.

Diseño de compensadores usando SISO Design Tool

El Control System Toolbox SISO Design Tool es una GUI que le permite analizar y ajustar sistemas de control retroalimentados SISO (se carga escribiendo `>> sisotool` en la ventana de comandos). Usando el SISO Design Tool, puede ajustar gráficamente la dinámica y ganancia del compensador usando una mezcla de técnicas de lugar geométrico de las raíces y de *loop shaping*. Por ejemplo, puede usar la vista del lugar geométrico de las raíces para estabilizar el bucle de retroalimentación y forzar un amortiguamiento mínimo, y usar diagramas de Bode para ajustar el ancho de banda, comprobar los márgenes de fase y ganancia o añadir un filtro *notch* para rechazar perturbaciones. La SISO Design GUI puede usarse para plantas de tiempo continuo y discreto. En la Figura 12-7 se muestran los diagramas de lugar geométrico de las raíces y de Bode para una planta de tiempo discreto.

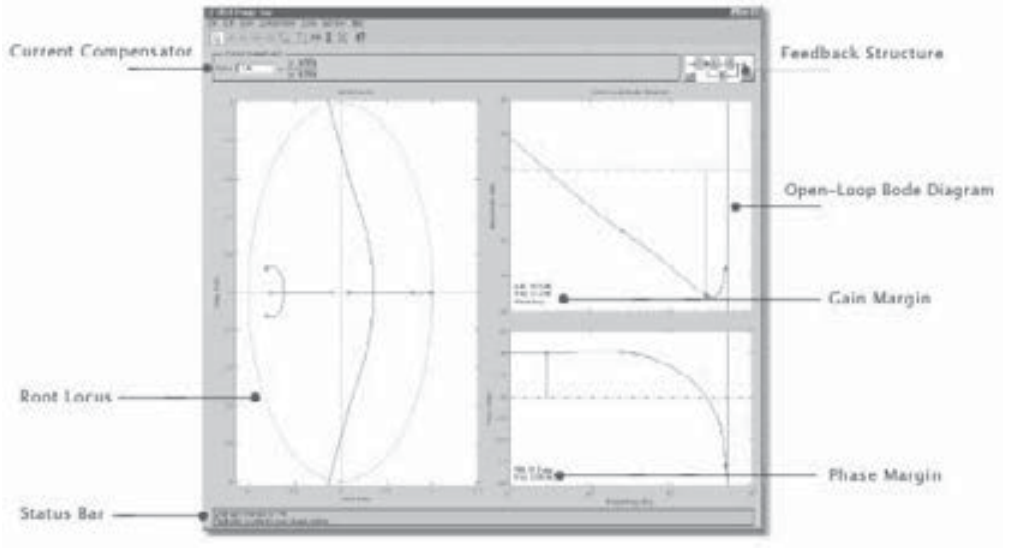


Figura 12-7

El SISO Design Tool está designado para funcionar estrechamente con el LTI Viewer, permitiéndole reiterar rápidamente en su diseño y ver inmeditamente los resultados en el LTI Viewer. Cuando hace un cambio en el compensador, el LTI Viewer asociado con su SISO Design Tool actualiza automáticamente los plots de respuesta que usted ha elegido. El SISO Design Tool integra la mayor parte de la funcionalidad del Control System Toolbox en una única GUI, enlaza dinámicamente vistas de tiempo, frecuencia y polos/ceros, ofreciendo vistas complementarias de los temas y objetivos de diseño, proporciona una visión gráfica de los cambios del diseño y ayuda a gestionar complejidad e iteraciones del diseño. Los menús desplegables y del botón derecho le dan flexibilidad para realizar tareas de diseño de controles con un clic del ratón. En particular, es posible colocar polos y ceros del compensador en las vistas del diagrama de Bode y del lugar geométrico de las raíces, añadir redes de atraso/adelanto y filtros de notch, ajustar gráficamente con el ratón los parámetros del compensador, inspeccionar respuestas de bucle cerrado (usando LTI Viewer), ajustar márgenes de fase y ganancia y convertir modelos entre tiempo discreto y continuo.

Diseño de compensadores usando la línea de comandos

Además, SISO Design Tool, Control System Toolbox aporta una serie de comandos que pueden usarse para una gama más amplia de aplicaciones de control, incluyendo funciones para diseño SISO clásico (datos de amortiguamiento, lugar geométrico de las raíces y márgenes de ganancia y fase), funciones para diseño MIMO moderno (colocación de polos, métodos LQR/LQG y filtrado de Kalman). El control Linear-Quadratic-Gaussian (LQG) es una técnica moderna de espacio de estado para diseñar reguladores dinámicos óptimos que permite equilibrar las prestaciones de la regulación y el esfuerzo en control teniendo en cuenta las perturbaciones del proceso y el ruido de medición.

12.3 Comandos de Control System Toolbox

Inicialmente podrían clasificarse los comandos del Control System Toolbox según su finalidad de la forma siguiente:

General
Ctrlpref:. Sitúa las preferencias del Control System Toolbox a través de un interfaz gráfico de usuario (Figura 12-8)
Creación de modelos lineales
tf: Crea un modelo de función de transferencia ***zpk:*** Crea un modelo zero/polo/ganancia ***ss, dss:*** Crea un modelo de espacio de los estados ***frd:*** Crea un modelo de datos de frecuencia de respuesta ***set:*** Sitúa y modifica propiedades de modelos LTI

Extracción de datos
ffdata: Extrae numerador y denominador de la función de transferencia ***zpkdata:*** Extrae datos cero/polo/ganancia ***ssdata:*** Extrae matrices del espacio de los estados ***get:*** Acceso a propiedades de modelos LTI
Conversiones
ss: Conversión al espacio de los estados ***zpk:*** Conversión a cero/polo/ganancia ***tf:*** Conversión a función de transferencia ***frd:*** Conversión a datos de frecuencia ***c2d:*** Conversión de continuo a discreto ***d2c:*** Conversión de discreto a continuo ***d2d:*** Remuestreo de un modelo discreto de tiempo
Interconexiones del sistema
append: Agrupa sistemas LTI añadiendo inputs y outputs ***parallel:*** Generaliza conexiones en paralelo ***series:*** Generaliza conexiones en serie ***feedback:*** Conexión feedback de dos sistemas ***lft:*** Interconexión feedback generalizada ***connect:*** Deriva el modelo de espacio de los estados de un diagrama de bloques
Modelos dinámicos
iopzmap: Mapea polos y ceros para pares input/output ***bandwidth:*** Amplitud o banda del sistema ***pole:*** Polos del sistema ***zero:*** Ceros del sistema (transmisión) ***pzmap:*** Mapea polo/cero ***damp:*** Frecuencia natural y damping de polos del sistema ***dcgain:*** Ganancia DC (baja frecuencia) ***norm:*** Normas de sistemas LTI ***covar:*** Covarianza de la respuesta de ruido blanco
Análisis en el dominio de tiempo
ltiview: GUI de respuesta GUI (LTI Viewer) ***step:*** Paso de respuesta ***impulse:*** Impulso de respuesta ***initial:*** Respuesta del sistema de espacio de los estados con estado inicial dado ***lsim:*** Respuesta a inputs arbitrarios
Análisis en el dominio de la frecuencia
ltiview: GUI de análisis de respuesta (LTI Viewer) ***bode:*** Diagramas de frecuencia de respuesta ***sigma:*** Grafica frecuencias de valores singulares ***nyquist:*** Nyquist plot ***nichols:*** Nichols plot ***margin:*** Márgenes de ganancia y fase ***allmargin:*** Todas las frecuencias cruzadas y márgenes ganancia/fase ***freqresp:*** Frecuencia de respuesta sobre una malla de frecuencia

Diseño clásico
sisotool: GUI de diseño SISO (técnicas *root locus* y *loop shaping*)
rlocus: Evans *root locus*
Pole placement
place: MIMO *pole placement*
estim: Estimación de ganancia
reg: Regulador de *state-feedback* y estimador de ganancia
Diseño LQR/LQG
lqr, dlqr: Regulador lineal cuadrático (LQ) de *state-feedback*
lqry: Regulador LQ con salida ponderada
lqrd: Regulador LQ discreto
kalman: Estimador de Kalman
kalmd: Estimador discreto de Kalman
Modelos de espacio de los estados
rss, drss: Modelos estables aleatorios de espacio de estados
ss2ss: Transformación de coordenadas de estado
ctrb, obsv: Matrices de control y observación
gram: Control y observación *gramians*
minreal: Realización mínima y cancelación polo/cero
ssbal: Balanceo diagonal y realizaciones del espacio de los estados
balreal: Balanceo *gramian*- input/output
modred: Modelo de reducción de estado
Desfases temporales
totaldelay: Desfase total entre cada par input/output
delay2z: Reemplaza desfases por polos en z=0 o cambio de fase FRD
pade: Aproximación de Pade de desfases temporales
Solucionadores de matrices de ecuaciones
lyap: Resuelve ecuaciones continuas de Lyapunov
dlyap: Resuelve ecuaciones discretas de Lyapunov
care: Resuelve ecuaciones continuas algebraicas de Riccati
dare: Resuelve ecuaciones continuas algebraicas de Riccati

En los apartados siguientes se presenta la sintaxis de los comandos citados anteriormente realizando grupos adecuados por categorías.

Comandos sobre Modelos LTI

Comando	***Descripción***
sys = drss(n,m,p)	*Genera modelos de espacio de los estados discretos aleatorios de orden n con m inputs y p outputs.*
sys = drss(n,p)	*Caso particular de m=1*
sys = drss(n)	*Caso particular de n=m=1*
sys = drss(n,p,m,s1,...sn)	*Genera un array de modelos de espacio de los estados*

dss(a,b,c,d,e)	*Crea el descriptor de tiempo continuo del modelo de espacio de los estados:* $E\dot{x} = Ax + Bu$ $y = Cx + Du$
dss(a,b,c,d,e,Ts)	*Crea el descriptor de tiempo simple discreto (con muestra de tiempo Ts en segundos) del modelo de espacio de los estados:* $Ex[n+1] = Ax[n] + Bu[n]$ $y[n] = Cx[n] + Du[n]$
dss(a,b,c,d,e,ltisys)	*Crea el descriptor del modelo de espacio de los estados con propiedades genéricas LTI heredadas del modelo ltisys.*
dss(a,b,c,d,e,p1,v1,p2,v2,...)	*Crea el descriptor de tiempo continuo del modelo de espacio de los estados con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
dss(a,b,c,d,e,Ts,p1,v1,p2,v2,...)	*Crea el descriptor de tiempo discreto (Ts en segundos) del modelo de espacio de los estados con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
sys = filt(num,den)	*Crea la función de transferencia en tiempo discreto en convención DSP con numerador y denominador dados y muestra de tiempo Ts unitaria*
sys = filt(num,den,Ts)	*Crea la función de transferencia en tiempo discreto en convención DSP con numerador y denominador dados y muestra de tiempo Ts en segundos*
sys = filt(M)	*Especifica un filtro estático con matriz de ganancia M*
sys = filt(num,den, p1,v1,p2,v2,...)	*Crea la función de transferencia en tiempo discreto en convención DSP con numerador y denominador dados y propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
sys = filt(num,den,Ts, p1,v1,p2,v2,...))	*Crea la función de transferencia en tiempo discreto en convención DSP con numerador y denominador dados, muestra de tiempo Ts en segundos y propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
sys = frd(r,f) **sys = frd(r,f,Ts)** **sys = frd** **sys = frd(r,f,ltisys)**	*Crea un modelo de frecuencia de respuesta de datos (FRD) almacenados en el array r y vector de frecuencias f* *Crea un modelo FRD de array r, vector de frecuencias f y muestra de tiempo Ts en segundos* *Crea un modelo FRD vacío* *Crea un modelo FRRD de array r, vector de frecuencias f y propiedades genéricas LTI del modelo ltisys*

sysfrd = frd(sys,f)	*Convierte un modelo TF, SS o ZPK a modelo FRD de frecuencia f*
sysfrd = frd(sys,f,'U',u)	*Convierte un modelo TF, SS o ZPK a modelo FRD de frecuencia f especificando las uniades para la frecuencia (*`'rad/s'` *or* `'Hz'`*)*
[r,f] = frdata(sys)	*Da los datos de respuesta y muestras de frecuencia del modelo FRD sys*
[r,f,Ts] = frdata(sys)	*Da los datos de respuesta, muestras de frecuencia y muestra de tiempo del modelo FRD sys*
[r,f] = frdata(sys,'v')	*Da los datos de respuesta y muestras de frecuencia del modelo FRD sys directamente como vectores columna*
get(sys)	*Muestra todas las propiedades y sus valores del modelo FRD sys*
get(sys, 'P')	*Muestra la propiedad de nombre P y su valor en sys*
sys = rss(n,m,p)	*Genera un modelo continuo estable aleatorio de orden n con m inputs y p outputs*
sys = rss(n,p)	*Caso particular de m=1*
sys = rss(n)	*Caso particular de n=m=1*
sys = rss(n,p,m,s1,...sn)	*Genera un array de modelos estables aleatorios*
set(sys,'*P*',V)	*Asigna el valor V a la propiedad dada del modelo LTI sys*
set(sys,'P1',V1,'P2',V2,...)	*Asiga los valores V1,...Vn a las propiedades P1,...Pn*
set(sys,'P')	*Muestra los valores admisibles para la propiedad P*
set(sys)	*Muestra todas las propiedades de sys y sus valores*
ss(a,b,c,d,e)	*Crea el modelo continuo de espacio de los estados:* $\dot{x} = Ax + Bu$ $y = Cx + Du$
ss(a,b,c,d,e,Ts)	*Crea el modelo discreto de espacio de los estados (con muestra de tiempo Ts en segundos)* $x[n+1] = Ax[n] + Bu[n]$ $y[n] = Cx[n] + Du[n]$
s(d)	*Crea el modelo de espacio de los estados con matriz de ganancia d. Equivale a* `sys = ss([],[],[],d)`
ss(a,b,c,d,e,ltisys)	*Crea el modelo de espacio de los estados con propiedades genéricas LTI heredadas del modelo ltisys.*
ss(a,b,c,d,e,p1,v1,p2,v2,...)	*Crea el modelo de espacio de los estados con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
ss(a,b,c,d,e,Ts,p1,v1,p2,v2,...)	*Crea el modelo de espacio de los estados con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi) y muestra de tiempo Ts en segundos*

sys_ss = ss(sys) **sys_ss = ss(sys,'minimal')**	*Convierte el modelo sys (TF o ZPK) al espacio de los estados* *Conversión con estados inobservables*
[a,b,c,d] = ssdata(sys) **[a,b,c,d,Ts] = ssdata(sys)**	*Extrae la matriz [A,B,C,D] del modelo de espacio de los estados sys.* *Extrae la matriz [A,B,C,D] y la muestra de tiempo Ts del modelo de espacio de los estados sys*
[a,b,c,d] = dsdata(sys) **[a,b,c,d,Ts] = dsdata(sys)**	*Extrae la matriz [A,B,C,D] del descriptor de tiempo del modelo de espacio de los estados sys.* *Extrae la matriz [A,B,C,D] del descriptor de tiempo y la muestra de tiempo Ts del modelo de espacio de los estados sys*
sys = tf(num,den)	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados y muestra de tiempo Ts unitaria*
sys = tf(num,den,Ts)	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados y muestra de tiempo Ts en segundos*
sys = tf(M)	*Crea una matriz de ganancia estática M*
sys = tf(num,den,ltisys)	*Crea la función de transferencia con propiedades genéricas heredadas del modelo LTI ltisys*
sys = tf(num,den, **p1,v1,p2,v2,...)**	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados y propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi)*
sys = tf(num,den,Ts, **p1,v1,p2,v2,...))**	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados, muestra de tiempo Ts en segundos y propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*

`s = tf('s')`
 	
`z = tf('z',Ts)`
	Especifica un modelo TF usando una función racional en la variable de Laplace s
	Especifica un modelo de tiempo TF con muestra de tiempo Ts usando una función racional en la variable discreta de tiempo z

tfsys = tf(sys) **tfsys = tf(sys,'inv')**	*Convierte el modelo sys (TF o ZPK) a forma de función de transferencia* *Conversión usando fórmulas de inversión*
[num,den] = tfdata(sys) **[num,den] = tfdata(sys,'v')** **[num,den,Ts] = tfdata(sys)**	*Da numerador y denominador para el modelo de función de transferencia sys de tipo TF, SS o ZPK* *Da numerador y denominador como vectores fila* *Da además la muestra de tiempo Ts*
td = totaldelay(sys)	*Da el desfase total combinado I/O del modelo LTI sys*
sys = zpk(z,p,k) **sys = zpk(z,p,k,Ts)**	*Crea el modelo con ceros z, polos p y ganancias k* *Crea el modelo con ceros z, polos p, ganancias k y muestra de tiempo en segundos Ts*

sys = zpk(M) **sys = zpk(z,p,k,ltisys)**	*Especifica una matriz de ganancia estática M* *Crea el modelo con ceros z, polos p y ganancias k con propiedades genéricas heredadas del modelo LTI ltisys*
sys=zpk(z,p,k,p1,v1,p2,v2,...)	*Crea el modelo con ceros z, polos p y ganancias k con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
sys=zpk(z,p,k,Ts,p1,v1,p2,v2,..)	*Crea el modelo con ceros z, polos p y ganancias k con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi) y muestra de tiempo Ts en segundos*

`sys = zpk('s')`
 	
`sys = zpk('z',Ts)`
	Especifica un modelo ZPK usando una función racional en la variable de Laplace s
	Especifica un modelo de tiempo ZPK con muestra de tiempo Ts usando una función racional en la variable discreta de tiempo z

zsys = zpk(sys) **zsys = zpk(sys,'inv')**	*Convierte el modelo sys LTI a tipo cero-polo-ganancia* *Convierte el modelo sys LTI a tipo cero-polo-ganancia usando fórmulas de inversión*
[z,p,k] = zpkdata(sys) **[z,p,k] = zpkdata(sys,'v')** **[z,p,k,Ts,Td] = zpkdata(sys)**	*Devuelve ceros z, polos p y ganancias k del modelo sys* *Devuelve ceros z, polos p y ganancias k del modelo sys como vectores columna* *Devuelve además la muestra de tiempo Ts y el input de desfase de datos Td.*

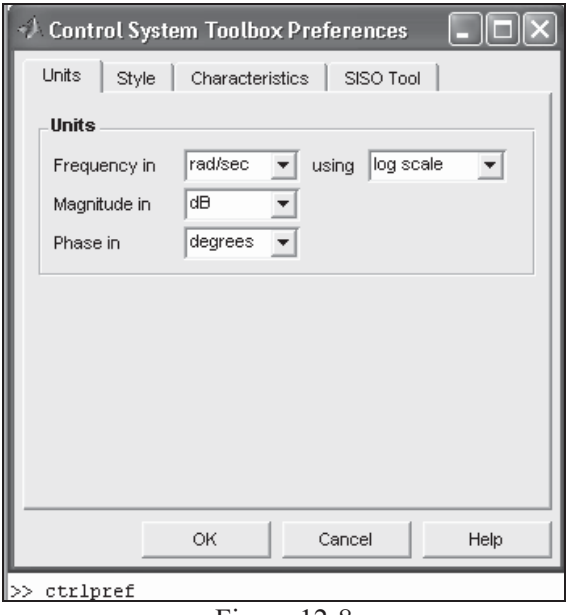


Figura 12-8

Como primer ejemplo generamos un sistema LTI discreto aleatorio con tres estados, dos inputs y dos outputs.

```
>> sys = drss(3,2,2)

a =
                    x1           x2           x3
           x1   -0.048856      0.40398      0.23064
           x2    0.068186      0.35404     -0.40811
           x3    -0.46016    -0.089457    -0.036824


b =
                    u1           u2
           x1    -0.43256      0.28768
           x2           0      -1.1465
           x3     0.12533       1.1909


c =
                    x1           x2           x3
           y1      1.1892      0.32729     -0.18671
           y2   -0.037633      0.17464      0.72579


d =
                    u1           u2
           y1           0      -0.1364
           y2      2.1832            0

Sampling time: unspecified
Discrete-time model.
>>
```

En el ejemplo siguiente creamos el modelo

$$5\dot{x} = x + 2u$$
$$y = 3x + 4u$$

con un desfase de entrada de 0,1 segundos y etiquetado como '*voltage*'.

```
>> sys = dss(1,2,3,4,5,0.1,'inputname','voltage')

a =
                    x1
           x1           1
```

```
b =
                    voltage
           x1             2

c =
                         x1
           y1             3

d =
                    voltage
           y1             4

e =
                         x1
           x1             5

Sampling time: 0.1
Discrete-time model.
```

En el ejemplo siguiente se crea el filtro digital de dos entradas siguiente:

$$H(z^{-1}) = \left[\frac{1}{1+z^{-1}+2z^{-2}} \quad \frac{1+0.3z^{-1}}{5+2z^{-1}}\right]$$

especificando tiempo de muestra y nombres de entradas *'channel1' y 'channel2'*

```
>> num = {1 , [1 0.3]}
den = {[1 1 2] ,[5 2]}
H = filt(num,den,'inputname',{'channel1' 'channel2'})

num =

    [1.00]    [1x2 double]

den =

    [1x3 double]    [1x2 double]

Transfer function from input "channel1" to output:

        1
-----------------
1 + z^-1 + 2 z^-2

Transfer function from input "channel2" to output:

1 + 0.3 z^-1
------------
 5 + 2 z^-1

Sampling time: unspecified
```

A continuación se crea un modelo SISO FRD.

```
>> freq = logspace(1,2);
resp = .05*(freq).*exp(i*2*freq);
sys = frd(resp,freq)

From input 1 to:

  Frequency(rad/s)        output 1
  ----------------        --------
       10.000000     0.204041+0.456473i
       10.481131    -0.270295+0.448972i
       10.985411    -0.549157+0.011164i
       11.513954    -0.293037-0.495537i
       12.067926     0.327595-0.506724i
       12.648552     0.623904+0.103480i
       13.257114     0.124737+0.651013i
       13.894955    -0.614812+0.323543i
       14.563485    -0.479139-0.548328i
       15.264180     0.481814-0.591898i
       15.998587     0.668563+0.439215i
       16.768329    -0.438184+0.714799i
       17.575106    -0.728874-0.490870i
       18.420700     0.602513-0.696623i
       19.306977     0.588781+0.765007i
             .
             .
             .
       86.851137    -2.649156-3.440897i
       91.029818     4.498503-0.692487i
       95.409548    -3.261293+3.481583i
      100.000000     2.435938-4.366486i

 Continuous-time frequency response data model.
```

A continuación se define un modelo FRD y se devuelven sus datos.

```
>> freq = logspace(1,2,2);
resp = .05*(freq).*exp(i*2*freq);
sys = frd(resp,freq);
[resp,freq] = frdata(sys,'v')

resp =
          0.20
          2.44
freq =
         10.00
        100.00
```

En el ejemplo siguiente se crea la función de transferencia 2-output/1-input siguiente:

$$H(p) = \begin{bmatrix} \dfrac{p+1}{p^2+2p+2} \\ \dfrac{1}{p} \end{bmatrix}$$

```
>> num = {[1 1] ; 1}
den = {[1 2 2] ; [1 0]}
H = tf(num,den)

num =

    [1x2 double]
    [      1.00]

den =

    [1x3 double]
    [1x2 double]

Transfer function from input to output...
          s + 1
 #1:  -------------
      s^2 + 2 s + 2

      1
 #2:  -
      s
```

En el ejemplo siguiente se computa la función de transferencia para el modelo de espacio de los estados siguiente:

$$A = \begin{bmatrix} -2 & -1 \\ 1 & -2 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad D = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

```
>> sys = ss([-2 -1;1 -2],[1 1;2 -1],[1 0],[0 1])
tf(sys)

a =
                      x1           x2
           x1         -2           -1
           x2          1           -2
```

```
b =
                       u1           u2
           x1           1            1
           x2           2           -1

c =
                       x1           x2
           y1           1            0

d =
                       u1           u2
           y1           0            1

Continuous-time model.

Transfer function from input 1 to output:
s - 2.963e-016
--------------
s^2 + 4 s + 5

Transfer function from input 2 to output:
s^2 + 5 s + 8
-------------
s^2 + 4 s + 5
```

En el ejemplo siguiente se especifican las funciones de transferencia discretas:

$$g(z) = \frac{z+1}{z^2+2z+3} \qquad h(z^{-1}) = \frac{1+z^{-1}}{1+2z^{-1}+3z^{-2}} = zg(z)$$

```
>> g = tf([1 1],[1 2 3],0.1)

 Transfer function:
    z + 1
-------------
z^2 + 2 z + 3

Sampling time: 0.1

>> h = tf([1 1],[1 2 3],0.1,'variable','z^-1')

Transfer function:
     1 + z^-1
-------------------
1 + 2 z^-1 + 3 z^-2

Sampling time: 0.1
```

A continuación se especifica el modelo cero-polo-ganancia correspondiente a la función de transferencia:

$$H(z) = \begin{bmatrix} \dfrac{1}{z-0.3} \\ \dfrac{2(z+0.5)}{(z-0.1+j)(z-0.1-j)} \end{bmatrix}$$

```
>> z = {[] ; -0.5}
p = {0.3 ; [0.1+i 0.1-i]}
k = [1 ; 2]
H = zpk(z,p,k,-1)

z =

            []
    [-0.5000]

p =

    [    0.3000]
    [1x2 double]

k =

     1
     2


Zero/pole/gain from input to output...

          1
 #1:  -------
      (z-0.3)

            2 (z+0.5)
 #2:  --------------------
      (z^2 - 0.2z + 1.01)

Sampling time: unspecified
```

En el ejemplo siguiente se convierte la función de transferencia definida como `tf([-10 20 0],[1 7 20 28 19 5])` en su correspondiente modelo cero-polo-ganancia.

```
>> h = tf([-10 20 0],[1 7 20 28 19 5])
```

```
Transfer function:

              -10 s^2 + 20 s
---------------------------------------
s^5 + 7 s^4 + 20 s^3 + 28 s^2 + 19 s + 5

>> zpk(h)

Zero/pole/gain:

     -10 s (s-2)
----------------------
(s+1)^3 (s^2 + 4s + 5)
```

Comandos sobre características del modelo

Comando	*Descripción*
str = class(object)	*Muestra una cadena con la clase de objeto en modelos (`'tf'`, `'zpk'`, `'ss'`, o `'frd'`)*
hasdelay(sys)	*Devuelve 1 si el modelo LTI tiene desfase de entrada, salida o entrada/salida*
K = isa(obj,'clase)	*Devuelve 1 si el objeto es de la clase dada*
boo = isct(sys)	*Da 1 si el modelo LTI sys es continuo*
boo = isdt(sys)	*Da 1 si el modelo LTI sys es discreto*
boo = isempty(sys)	*Da 1 si el modelo LTI sys no tiene entrada o salida*
boo = isproper(sys)	*Devuelve 1 si el modelo LTI sys es adecuado*
boo = issiso(sys)	*Devuelve 1 si el modelo LTI sys es SISO*
n = ndims(sys)	*Muestra el número de dimensiones del modelo/array LTI sys*
size(sys)	*Muestra dimensiones output/input/array de sys*
d = size(sys)	*Muestra el número de inputs/outputs de sys*
Ny = size(sys,1)	*Muestra el número de outputs de sys*
Nu = size(sys,2)	*Muestra el número de inputs de sys*
Sk = size(sys,2+k)	*Da la longitud de la k-ésima dimensión del array cuando sys es un array LTI*
Ns = size(sys,'orden')	*Devuelve el orden del modelo sys (TS, SS o ZPK)*
Nf = size(sys,'frecuencia')	*Da el número de frecuencias si sys es un modelo FRD*

Comandos de conversión de modelos

Comando	Descripción
sysd = c2d(sys,Ts)	*Convierte a continuo el modelo discreto sys usando control de orden cero en la entrada y muestra de tiempo Ts*
sysd = c2d(sys,Ts,método)	*Usa los métodos de discretización de orden cero (zoh), aproximación tiangular (foh), aproximación bilineal o de Tustin (tustin), frecuencia de Tustin (prewarp) y polos-cero (matched)*
[sysd,G] = c2d(sys,Ts,método)	*Devuelve una matriz G que mapea las condiciones iniciales continuas x0, u0 y sus correspondientes x[0], u[0] usando el método especificado*
sys = chgunits(sys,unidades)	*Convierte las unidades de los puntos de frecuencia almacenados en un modelo FRD a Hz o rad/seg*
sysc = d2c(sysd)	*Convierte a discreto el modelo continuo sys usando control de orden cero en la entrada*
sysc = d2c(sysd,método)	*Usa los métodos de conversión zoh, tustin, prewarp o matched*
sys1 = d2d(sys,Ts)	*Remuestrea el modelo de tiempo discreto sys a un modelo sys1 con muestra de tiempo Ts*
sys = delay2z(sys)	*Convierte desfases en modelos de tiempo discreto o FRD (un desfase k períodos de muestreo se reemplaza por $1/z^k$ en la función de transferencia)*
sys = frd(r,f)	*Crea un modelo de frecuencia de respuesta de datos (FRD) almacenados en el array r y vector de frecuencias f*
sys = frd(r,f,Ts)	*Crea un modelo FRD de array r, vector de frecuencias f y muestra de tiempo Ts en segundos*
sys = frd	*Crea un modelo FRD vacío*
sys = frd(r,f,ltisys)	*Crea un modelo FRRD de array r, vector de frecuencias f y propiedades genéricas LTI del modelo ltisys*
sysfrd = frd(sys,f)	*Convierte un modelo TF, SS o ZPK a modelo FRD de frecuencia f*
sysfrd = frd(sys,f,'U',u)	*Convierte un modelo TF, SS o ZPK a modelo FRD de frecuencia f especificando las unidades para la frecuencia (`'rad/s'` or `'Hz'`)*
[num,den] = pade(T,N)	*Aproximación de Padé de orden N para el modelo de tiempo continuo con desfase e^{-sT} en la función de transferencia*
pade(T,N)	*Realiza un gráfico*
sysx = pade(sys,N)	*Aproxima sys por sysx mediante un desfase libre*
sysx = pade(sys,NI,NO,Nio)	*Especifica aproximación de órdenes de desfase de entrada, salida y entrada salida, respectivamente*

sys = reshape(sys,s1,s2,...,sk) **sys = reshape(sys,[s1 s2 ... sk])**	*Cambia la forma (shape) del modelo LTI sys a un array de modelos LTI*
[r,p,k] = residue(b,a) **[b,a] = residue(r,p,k)**	*Encuentra residuos, polos y términos directos de la expansión en fracciones parciales del ratio de dos polinomios a(s) y b(s)* *Convierte la expansión en fracciones parciales del ratio de dos polinomios a(s) y b(s) a polinomios con coeficientes en b y a.*
sys=ss(a,b,c,d,e)	*Crea el modelo continuo de espacio de los estados:* $\dot{x} = Ax + Bu$ $y = Cx + Du$
sys(a,b,c,d,e,Ts)	*Crea el modelo discreto de espacio de los estados (con muestra de tiempo Ts en segundos)* $x[n+1] = Ax[n] + Bu[n]$ $y[n] = Cx[n] + Du[n]$
sys=ss(a,b,c,d,e,ltisys)	*Crea el modelo de espacio de los estados con propiedades genéricas LTI heredadas del modelo ltisys*
sys=ss(a,b,c,d,e,p1,v1,p2,v2,...)	*Crea el modelo de espacio de los estados con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
sys=ss(a,b,c,d,e,Ts,p1,v1,p2,v2,...)	*Crea el modelo de espacio de los estados con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi) y muestra de tiempo Ts en segundos*
sys_ss = ss(sys) **sys_ss = ss(sys,'minimal')**	*Pasa el modelo sys (TF o ZPK) al espacio de los estados* *Conversión con estados inobservables*
sys = tf(num,den)	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados y muestra de tiempo Ts unitaria*
sys = tf(num,den,Ts)	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados y muestra de tiempo Ts en segundos*
sys = tf(M)	*Crea una matriz de ganancia estática M*
sys = tf(num,den,ltisys)	*Crea la función de transferencia con propiedades genéricas heredadas del modelo LTI ltisys*
sys = tf(num,den, **p1,v1,p2,v2,...)**	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados y propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi)*

sys = tf(num,den,Ts, p1,v1,p2,v2,...))	*Crea la función de transferencia en tiempo continuo con numerador y denominador dados, muestra de tiempo Ts en segundos y propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*

`s = tf('s')`
 	
`z = tf('z',Ts)`
	Especifica un modelo TF usando una función racional en la variable de Laplace s
	Especifica un modelo de tiempo TF con muestra de tiempo Ts usando una función racional en la variable discreta de tiempo z

tfsys = tf(sys) **tfsys = tf(sys,'inv')**	*Convierte el modelo sys (TF o ZPK) a forma de función de transferencia* *Conversión usando fórmulas de inversión*
sys = zpk(z,p,k) **sys = zpk(z,p,k,Ts)** **sys = zpk(M)** **sys = zpk(z,p,k,ltisys)**	*Crea el modelo con ceros z, polos p y ganancias k* *Crea el modelo con ceros z, polos p, ganancias k y muestra de tiempo en segundos Ts* *Especifica una matriz de ganancia estática M* *Crea el modelo con ceros z, polos p y ganancias k con propiedades genéricas heredadas del modelo LTI ltisys*
sys=zpk(z,p,k,p1,v1,p2,v2,...)	*Crea el modelo con ceros z, polos p y ganancias k con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi).*
sys=zpk(z,p,k,Ts,p1,v1,p2,v2,..)	*Crea el modelo con ceros z, polos p y ganancias k con propiedades genéricas LTI dadas por los pares propiedad valor (pi,vi) y muestra de tiempo Ts en segundos*

`sys = zpk('s')`
 	
`sys = zpk('z',Ts)`
	Especifica un modelo ZPK usando una función racional en la variable de Laplace s
	Especifica un modelo de tiempo ZPK con muestra de tiempo Ts usando una función racional en la variable discreta de tiempo z

zsys = zpk(sys) **zsys = zpk(sys,'inv')**	*Convierte el modelo sys LTI a tipo cero-polo-ganancia* *Convierte el modelo sys LTI a tipo cero-polo-ganancia usando fórmulas de inversión*

Como primer ejemplo consideramos el sistema:

$$H(s) = \frac{s-1}{s^2+4s+5}$$

con desfase de entrada $Td = 0{,}35$ segundos. Se trata de discretizar este sistema usando aproximación triangular con tiempo de muestreo $Ts = 0{,}1$ segundos.

```
>> H = tf([1 -1],[1 4 5],'inputdelay',0.35)

Transfer function:

                    s - 1
exp(-0.35*s) * -------------
               s^2 + 4 s + 5

>> Hd = c2d(H,0.1,'foh')

Transfer function:

          0.0115 z^3 + 0.0456 z^2 - 0.0562 z - 0.009104
z^(-3) * ---------------------------------------------
                   z^3 - 1.629 z^2 + 0.6703 z

Sampling time: 0.1
```

Si queremos comparar la respuesta continua y la discretizada (Figura 12-9) podemos utilizar el siguiente comando:

```
>> step(H,'-',Hd,'--')
```

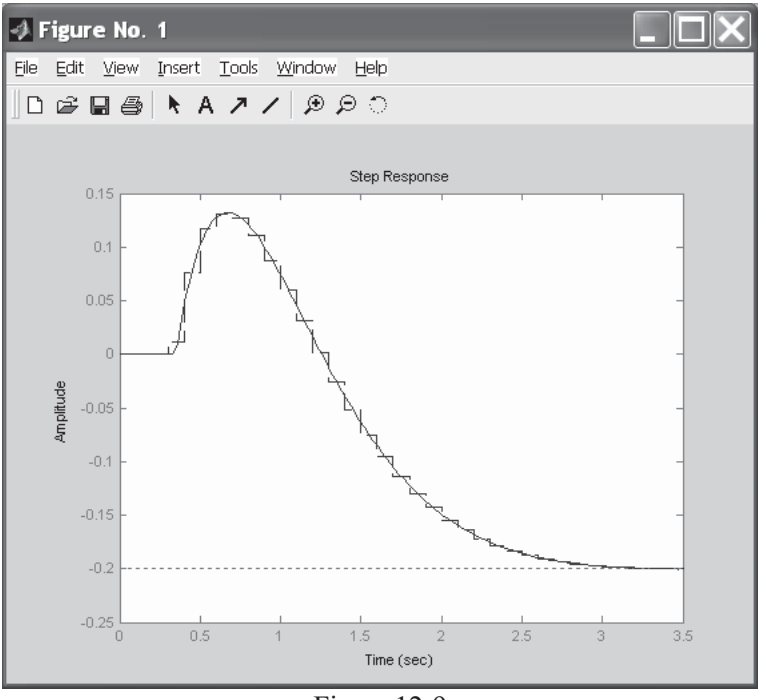


Figura 12-9

En el ejemplo se computa una aproximación de Pade de tercer orden de 0,1 segundos de desfase I/O y se compara el tiempo y la frecuencia de respuesta del desfase verdadero y el de su aproximación (Figura 12-10).

```
>> pade(0.1,3)
Step response of 3rd-order Pade approximation
```

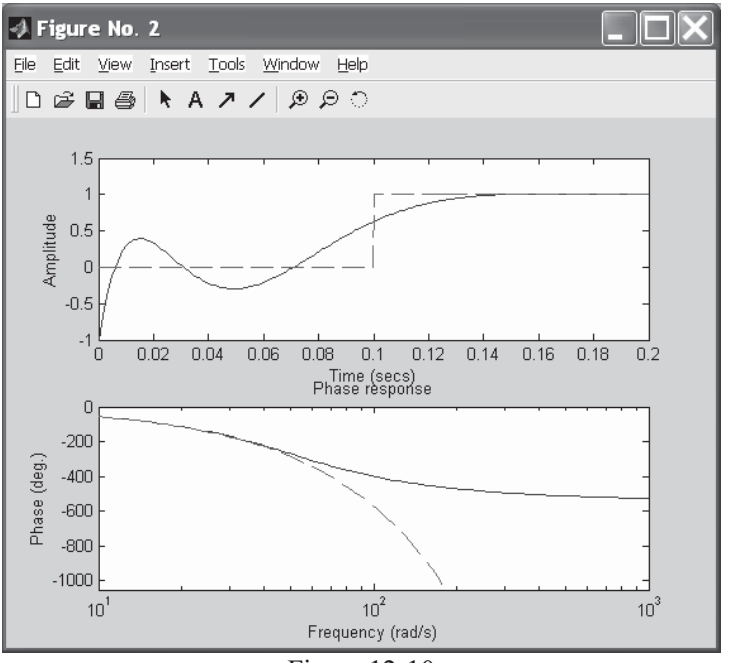


Figura 12-10

Comandos de reducción de orden en los modelos

Comando	Descripción
sysb = balreal(sys) **[sysb,g,T,Ti] = balreal(sys)**	*Calcula una realización balanceada I/O de sys* *Calcula además el vector g con la diagonal de la grammiana balanceada, el estado de similaridad T de la transformación $x_b = Tx$ usando para convertir sys a sysb y la inversa de la transformación $T_i = T^{-1}$*
sysr = minreal(sys) **sysr = minreal(sys,tol)** **[sysr,u] = minreal(sys,tol)**	*Elimina estados incontrolables e inobservables en modelos de espacio de los estados o cancela pares polo-cero en funciones de transferencia o modelos cero-polo-ganancia. Se puede especificar la tolerancia tol para la eliminación. Por defecto tol = sqrt(eps). También es posible obtener la matriz ortogonal U de la descomposición de Kalman*

rsys = modred(sys,elim) **rsys = modred(sys,elim,'mdc')** **rsys = modred(sys,elim,'del')**	*Borra los estados declarados en elim reduciendo el orden del modelo pudiendo incluso simplificar los estados borrados*
msys = sminreal(sys)	*Calcula reducción de modelo estructurada eliminando los estados del modelo del espacio de los estados sys que no afectan a la respuesta I/O*

En el ejemplo que se presenta a continuación se considera el modelo cero-polo-ganancia definido como *sys = zpk*([-10 -20.01],[-5 -9.9 -20.1],1) y se calcula una realización balanceada presentando la diagonal de la grammiana balanceada.

```
>> sys = zpk([-10 -20.01],[-5 -9.9 -20.1],1)

Zero/pole/gain:

   (s+10) (s+20.01)
----------------------
(s+5) (s+9.9) (s+20.1)

>> [sysb,g] = balreal(sys)

a =
            x1       x2       x3
   x1    -4.97   0.2399   0.2262
   x2  -0.2399   -4.276   -9.467
   x3   0.2262    9.467   -25.75

b =
             u1
   x1        -1
   x2  -0.02412
   x3   0.02276

c =
            x1       x2       x3
   y1       -1  0.02412  0.02276

d =
       u1
   y1   0

Continuous-time model.

g =
    0.1006
    0.0001
    0.0000
```

El resultado muestra que los dos últimos estados están débilmente acoplados a la entrada y salida, por lo que será conveniente eliminarlos mediante la sintaxis:

```
>> sysr = modred(sysb,[2 3],'del')

a =
           x1
   x1   -4.97

b =
        u1
   x1   -1

c =
        x1
   y1   -1

d =
        u1
   y1    0

Continuous-time model.
```

Ahora podemos comparar las respuestas de los modelos original y reducido (Figura 12-11) mediante la sintaxis siguiente:

```
>> bode(sys,'-',sysr,'x')
```

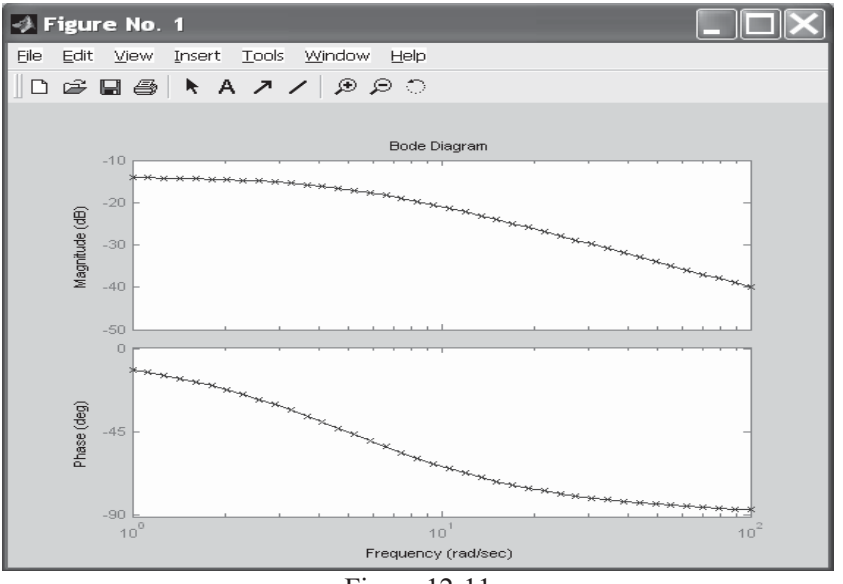


Figura 12-11

Comandos de realización del espacio de los estados

Comando	Descripción
csys = canon(sys,*'type'*) **[csys,T] = canon(sys,*'type'*)**	*Computa realizaciones canónicas del espacio de los estados para el sistema LTI sys discreto o continuo. Son posibles dos tipos de forma canónica, 'modal' y 'companion'. También es posible obtener la transformación T de la forma canónica a la original*
Co = ctrb(A,B) **Co = ctrb(sys)**	*Devuelve la matriz de controlabilidad para sistemas del espacio de los estados*
[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C) **[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C,tol)**	*Computa la controlabilidad para sistemas de espacio de los estados [A B C] obteniendo el sistema en forma de escalera [Abar Bbar Cbar] pudiendo incluso fijar un nivel de tolerancia tol. T es la matriz de transformación de similaridad y k es un vector de longitud n, siendo n el orden del sistema representado por A. El número de valores no nulos de k indica las iteraciones para calcular T*
Wc = gram(sys,'c') **Wo = gram(sys,'o')**	*Calcula las grammianas de controlabilidad y observabilidad*
Ob = obsv(A,B) **Ob = obsv(sys)**	*Calcula las matrices de observabilidad para sistemas del espacio de los estados*
[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C) **[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C,tol)**	*Calcula la observabilidad en forma de escalera*
sysT = ss2ss(sys,T)	*Devuelve el modelo transformado de espacio de los estados para la transformación de coordenadas T*
[sysb,T] = ssbal(sys) **[sysb,T] = ssbal(sys,condT)**	*Computa el modelo balanceado del espacio de los estados mediante la transformación T de similaridad diagonal.*

Como primer ejemplo consideramos el modelo continuo de espacio de los estados siguiente:

$$A = \begin{bmatrix} 1 & 10^4 & 10^2 \\ 0 & 10^2 & 10^5 \\ 10 & 1 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \qquad C = \begin{bmatrix} 0.1 & 10 & 100 \end{bmatrix}$$

Calcularemos su modelo balanceado como sigue:

```
>> a = [1 1e4 1e2;0 1e2 1e5;10 1 0];
b = [1;1;1];
c = [0.1 10 1e2];
sys = ss(a,b,c,0)

a =
           x1      x2      x3
   x1       1  1e+004     100
   x2       0     100  1e+005
   x3      10       1       0

b =
       u1
   x1   1
   x2   1
   x3   1

c =
        x1   x2   x3
   y1  0.1   10  100

d =
       u1
   y1   0

Continuous-time model.
```

En el ejemplo siguiente calculamos la matriz de observabilidad en escalera correspondiente al sistema *A*=[1, 1;4, -2], *B*=[1,-1;1,-1], *C*=[1,0;0,1].

```
>> A=[1, 1;4, -2]; B=[1,-1;1,-1]; C=[1,0;0,1];
>> [Abar,Bbar,Cbar,T,k] = obsvf(A,B,C)

Abar =

     1     1
     4    -2

Bbar =

     1    -1
     1    -1

Cbar =

     1     0
     0     1
```

```
T =

     1     0
     0     1

k =

     2     0
```

En el ejemplo siguiente calculamos la matriz de controlabilidad en escalera correspondiente al sistema del ejemplo anterior.

```
>> A=[1, 1;4, -2]; B=[1,-1;1,-1]; C=[1,0;0,1];
>> [Abar,Bbar,Cbar,T,k]=ctrbf(A,B,C)

Abar =

   -3.0000    0.0000
    3.0000    2.0000


Bbar =

         0         0
   -1.4142    1.4142


Cbar =

   -0.7071   -0.7071
    0.7071   -0.7071


T =

   -0.7071    0.7071
   -0.7071   -0.7071


k =

     1     0
```

Comandos de modelos dinámicos

Comando	Descripción
[Wn,Z] = damp(sys) **[Wn,Z,P] = damp(sys)**	*Calcula el factor damping (vector columna Wn) y las frecuencias naturales (vector columna Z) de los polos del modelo LTI sys. También se puede obtener el vector P de los polos de sys*
k = dcgain(sys)	*Calcula la ganancia de baja frecuencia del modelo sys*
[P,Q] = covar(sys,W)	*Calcula la covarianza estacionaria del output en el modelo LTI sys manejado por el input ruido blanco gaussiano W. P es la covarianza de respuesta y Q es la covarianza de los estados*
s = dsort(p) **[s,ndx] = dsort(p)**	*Ordena los polos de tiempo completo del vector p por magnitud, siendo ndx los índices de ordenación*
s = esort(p) **[s,ndx] = esort(p)**	*Ordena los polos p de tiempo continuo según parte real siendo posible obtener los índices de ordenación ndx*
norm(sys) **norm(sys,2)** **norm(sys,inf)** **norm(sys,inf,tol)** **[ninf,fpeak] = norm(sys)**	*Calcula la H_2 norma del modelo sys* *Calcula la H_2 norma del modelo sys* *Calcula la norma infinita H_∞ del modelo sys* *Calcula la norma infinita H_∞ del modelo sys con tolerancia tol* *Calcula, además de la norma infinita, el valor para el que alcanza la cima*
p = pole(sys) **d = eig(A)** **d = eig(A,B)** **[V,D] = eig(A)** **[V,D] = eig(A,'nobalance')** **[V,D] = eig(A,B)** **[V,D] = eig(A,B,flag)**	*Calcula los polos del modelo LTI* *Devuelve el vector de autovalores de A* *Devuelve los autovalores generalizados de A y B* *Devuelve autovalores y autovectores de la matriz A* *Da autovalores y autovectores de A sin balanceo previo* *Da autovalores y autovectores generalizados de A y B* *Se usa el método de factorización flag ('chol' o 'qz')*
pzmap(sys) **pzmap(sys1,sys2,...,sysN)** **[p,z] = pzmap(sys)**	*Grafica el mapa polo/cero del sistema LTI sys o de varios sistemas LTI a la vez sys1, sys2,..., sysn. [p,z] da los polos y los ceros y no el gráfico*
rlocus(sys) **rlocus(sys,k)** **rlocus(sys1,sys2,...)** **[r,k] = rlocus(sys)** **r = rlocus(sys,k)**	*Calcula y grafica root locus para el sistema LTI sys* *Usa el vector k de ganancias para graficar rot locus* *Grafica rot locus de varios sistemas en un gráfico simple* *Calcula root locus y ganancias del sistema LTI sys* *Calcula root locus del sistema sys para unas ganancias k*
r = roots(c)	*Calcula el vector columna de las raíces del polinomio c*
sgrid, zgrid	*Superpone mallas de planos s y z para root locus o mapas polo/cero*
z = zero(sys) **[z,gain] = zero(sys)**	*Calcula los ceros del modelo LTI sys* *Calcula ceros y ganancia del sistema LTI sys*

Como primer ejemplo calculamos autovalores, frecuencias naturales y factores damping para el modelo con función de transferencia continua:

$$H(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

```
>> H = tf([2 5 1],[1 2 3])

Transfer function:

2 s^2 + 5 s + 1
---------------
 s^2 + 2 s + 3

>> damp(H)

        Eigenvalue             Damping      Freq. (rad/s)

 -1.00e+000 + 1.41e+000i     5.77e-001       1.73e+000
 -1.00e+000 - 1.41e+000i     5.77e-001       1.73e+000
```

En el ejemplo siguiente calculamos la ganancia DC del modelo MIMO con función de transferencia:

$$H(s) = \begin{bmatrix} 1 & \dfrac{s-1}{s^2+s+3} \\ \dfrac{1}{s+1} & \dfrac{s+2}{s-3} \end{bmatrix}$$

```
>> H = [1 tf([1 -1],[1 1 3]) ; tf(1,[1 1]) tf([1 2],[1 -3])]
dcgain(H)

Transfer function from input 1 to output...

 #1:  1

        1
 #2:  -----
      s + 1

Transfer function from input 2 to output...

         s - 1
 #1:  -----------
      s^2 + s + 3
```

```
      s + 2
 #2:  -----
      s - 3

ans =

    1.0000   -0.3333
    1.0000   -0.6667
```

A continuación consideramos la función de transferencia de tiempo discreto siguiente:

$$H(z) = \frac{z^3 - 2.841z^2 + 2.875z - 1.004}{z^3 - 2.417z^2 + 2.003z - 0.5488}$$

con tiempo de muestreo 0,1 segundo y calculamos la 2-norma y la norma infinita con su valor óptimo.

```
>> H = tf([1 -2.841 2.875 -1.004],[1 -2.417 2.003 -0.5488],0.1)
norm(H)

Transfer function:

z^3 - 2.841 z^2 + 2.875 z - 1.004
---------------------------------
z^3 - 2.417 z^2 + 2.003 z - 0.5488

Sampling time: 0.1

ans =

    1.2438

>> [ninf,fpeak] = norm(H,inf)

ninf =
    2.5488
fpeak =

    3.0844
```

A continuación confirmamos los valores anteriores (Figura 12-12) mediante el gráfico Bode de $H(z)$.

```
>> bode(H)
```

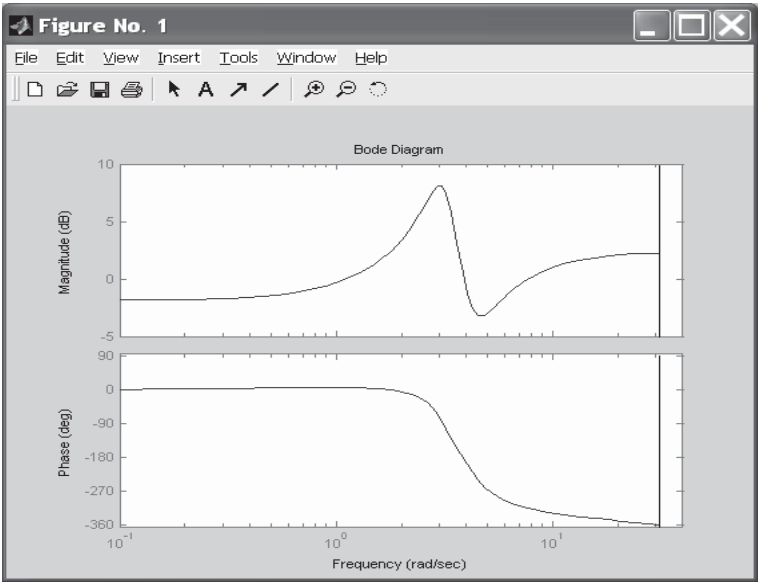


Figura 12-12

A continuación se calculan y grafican (Figura 12-13) *root locus* del sistema:

$$h(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

```
>> h = tf([2 5 1],[1 2 3]);
rlocus(h)
```

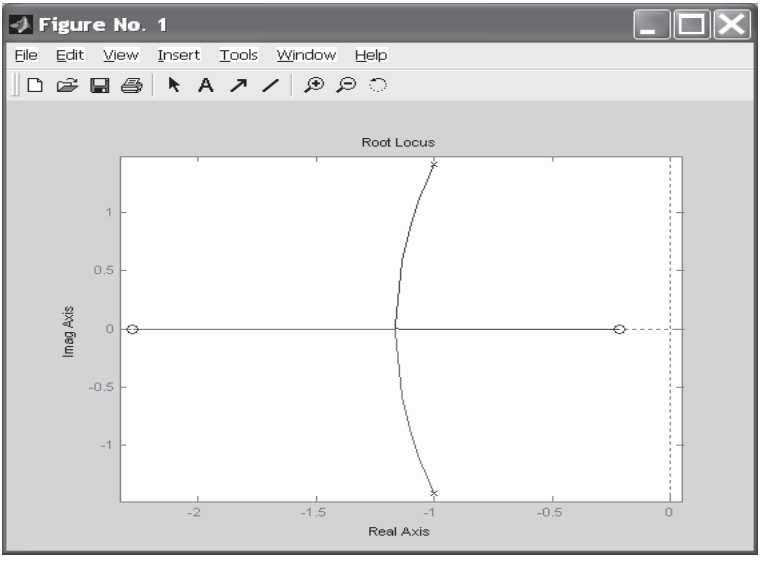


Figura 12-13

En el ejemplo siguiente graficamos las líneas de malla del plano *Z* de *root locus* (Figura 12-14) para el sistema:

$$H(z) = \frac{2z^2 - 3.4z + 1.5}{z^2 - 1.6z + 0.8}$$

```
> H = tf([2 -3.4 1.5],[1 -1.6 0.8],-1)

Transfer function:

2 z^2 - 3.4 z + 1.5
-------------------
 z^2 - 1.6 z + 0.8

Sampling time: unspecified

>> rlocus(H)
zgrid
axis('square')
```

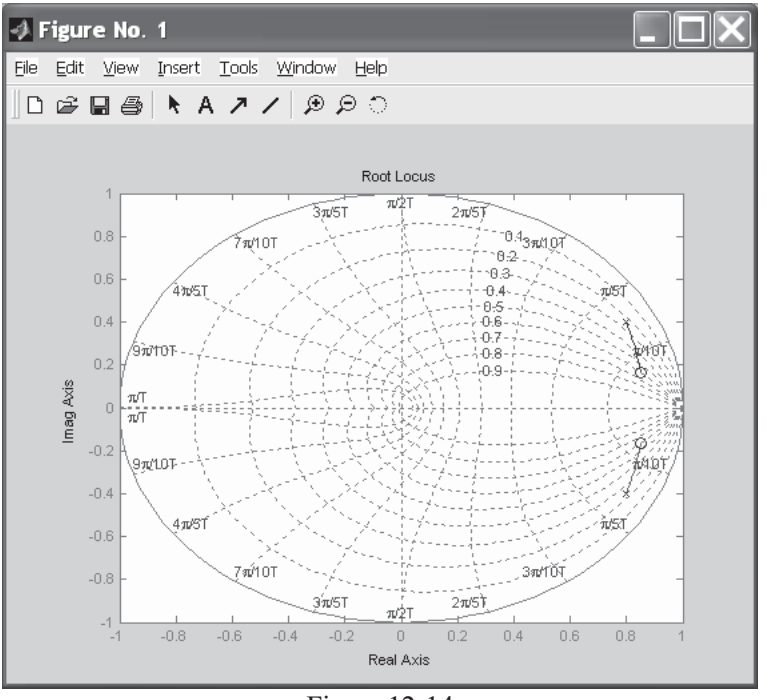


Figura 12-14

Comandos de interconexión de modelos

Comando	Descripción
sys = append(sys1,sys2,...,sysN)	*Añade los modelos en un bloque de configuración diagonal. Agrupa los modelos juntando inputs y outputs (Figura 12-15)*
asys = augstate(sys)	*Aumenta el output añadiendo estados*
sysc = connect(sys,Q,inputs,outputs)	*Conecta los subsistemas de un modelo en bloque de acuerdo a un esquema de interconexión elegido (conexiones dadas por la matriz Q)*
sys = feedback(sys1,sys2) **sys = feedback(sys1,sys2,sign)** **sys = feedback(sys1,sys2,feedin,feedout,sign)**	*Calcula conexión feedback de modelos (Figura 12-16) pudiendo incluir signo y bucle cerrado (Figura 12-17)*
sys = lft(sys1,sys2) **sys = lft(sys1,sys2,nu,ny)**	*Realiza interconexión LFT de modelos (Figura 12-18)*
[A,B,C,D] = ord2(wn,z) **[num,den] = ord2(wn,z)**	*Genera modelos de segundo orden de espacio de los estados para frecuencias Wn y factores damping z dados*
sys = parallel(sys1,sys2) **sys = parallel(sys1,sys2,inp1,inp2,out1,out2)**	*Crea conexiones generalizadas en pararlelo de sistemas (Figura 12-19)*
sys = series(sys1,sys2) **sys = series(sys1,sys2,outputs1,inputs2)**	*Crea conexiones generalizadas en serie de sistemas (Figura 12-20)*
sys = stack(arraydim,sys1,sys2,...)	*Modelos LTI apilados en un array de modelos*

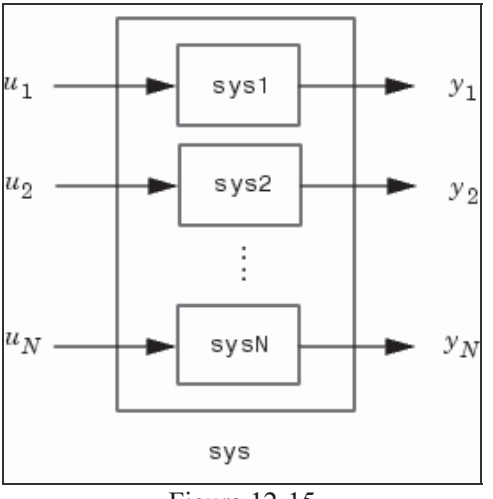


Figura 12-15

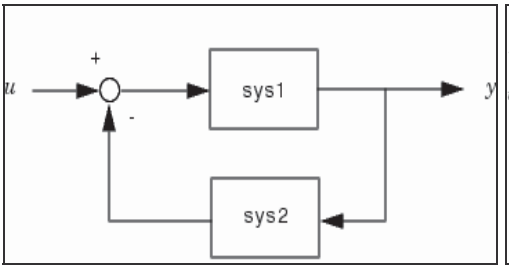


Figura 12-16

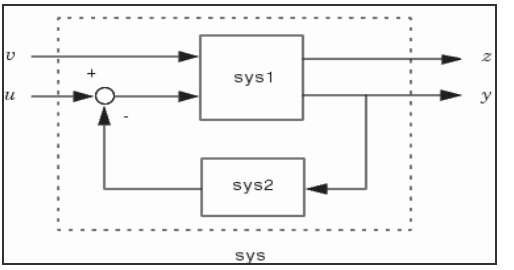


Figura 12-17

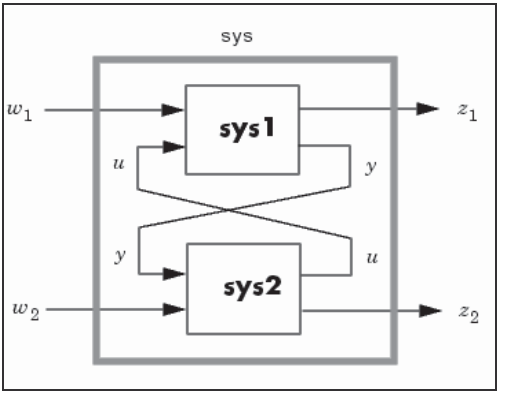


Figura 12-18

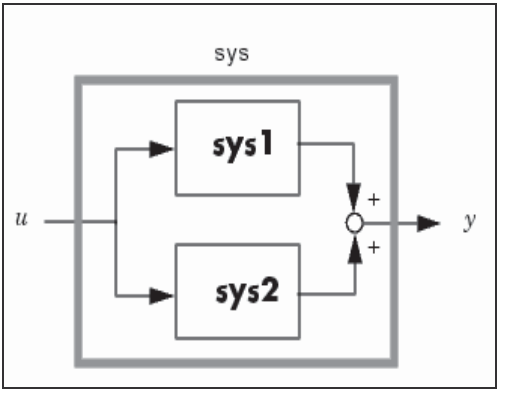


Figura 12-19

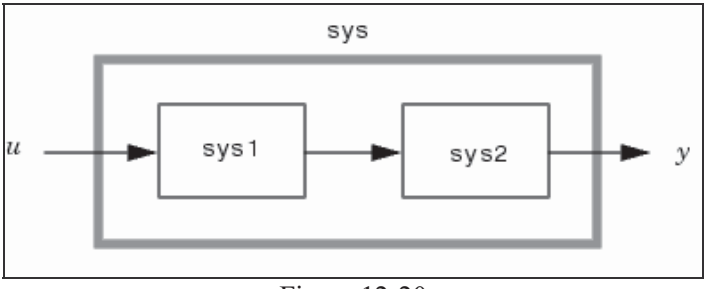


Figura 12-20

Como primer ejemplo añadimos los sistemas *tf*(1,[1 0]) *y ss*(1,2,3,4). Hay que tener presente que para sistemas con funciones de transferencia *H*1(*s*), *H*2(*s*),…, *Hn*(*s*), el sistema resultante al añadir todos ellos tiene como función de transferencia:

$$\begin{bmatrix} H_1(s) & 0 & .. & 0 \\ 0 & H_2(s) & . & : \\ : & . & . & 0 \\ 0 & .. & 0 & H_N(s) \end{bmatrix}$$

Para dos sistemas *sys*1 y *sys*2 definidos por (*A*1,*B*1,*C*1, *D*1) y (*A*2,*B*2,*C*2,*D*2), se tiene que *append*(*sys1,sys2*) resulta ser el sistema:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}A_1 & 0\\ 0 & A_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}B_1 & 0\\ 0 & B_2\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$\begin{bmatrix}y_1\\ y_2\end{bmatrix} = \begin{bmatrix}C_1 & 0\\ 0 & C_2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}D_1 & 0\\ 0 & D_2\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

Para nuestro ejemplo tenemos:

```
>> sys1 = tf(1,[1 0])
sys2 = ss(1,2,3,4)
sys = append(sys1,10,sys2)

Transfer function:

1
-
s

a =
       x1
   x1   1


b =
       u1
   x1   2


c =
       x1
   y1   3


d =
       u1
   y1   4

Continuous-time model.

a =
       x1  x2
   x1   0   0
   x2   0   1
```

```
b =
       u1  u2  u3
   x1   1   0   0
   x2   0   0   2

c =
       x1  x2
   y1   1   0
   y2   0   0
   y3   0   3


d =
       u1  u2  u3
   y1   0   0   0
   y2   0  10   0
   y3   0   0   4

Continuous-time model.
```

En el ejemplo siguiente se conecta la planta G(s) con el controlador H(s) representados en la Figura 12-21 (usando feedback negativo) y definidos como sigue:

$$G(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

$$H(s) = \frac{5(s+2)}{s+10}$$

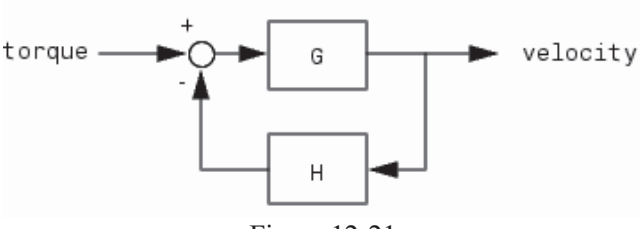


Figura 12-21

```
>> G = tf([2 5 1],[1 2 3],'inputname','torque',...
                          'outputname','velocity');
H = zpk(-2,-10,5)
Cloop = feedback(G,H)

Zero/pole/gain:
```

```
5 (s+2)
-------
(s+10)

Zero/pole/gain from input "torque" to output "velocity":

0.18182 (s+10) (s+2.281) (s+0.2192)
-----------------------------------
 (s+3.419) (s^2 + 1.763s + 1.064)
```

En el ejemplo siguiente se genera un modelo con función de transferencia de segundo orden, con factor damping 0,4 y con frecuencia natural 2.4 rad/seg.

```
>> [num,den] = ord2(2.4,0.4)

num =

     1

den =

    1.0000    1.9200    5.7600

>> sys = tf(num,den)

Transfer function:
          1
-------------------
s^2 + 1.92 s + 5.76
```

Comandos de tiempo de respuesta

Comando	Descripción
[u,t] = gensig(*type*,tau)	*Genera una señal escalar u de la clase type y período tau. Los tipos pueden ser sin, square y pulse*
[u,t] = gensig(*type*,tau,Tf,Ts)	*Se especifica también el tiempo de duración de la señal Tf y el espaciado Ts*
impulse(sys)	*Calcula y grafica el impulso respuesta unitario del sistema LTI sys*
impulse(sys,t)	*Calcula y grafica el impulso respuesta en el tiempo t del sistema LTI sys*
impulse(sys1,sys2,...,sysN)	*Calcula y grafica el impulso respuesta unitario de varios sistemas LTI*
impulse(sys1,sys2,...,sysN,t)	*Calcula y grafica el impulso respuesta en el tiempo t de varios sistemas LTI*

impulse(sys1,'PlotStyle1',... ,sysN,'PlotStyleN') **[y,t,x] = impulse(sys)**	*Además se define estilo para los gráficos* *Devuelve el output de respuesta y, el vector de tiempo t usado para la simulación y el estado de las trayectorias x*
initial(sys,x0) **initial(sys,x0,t)** **initial(sys1,sys2,...,sysN,x0)** **initial(sys1,sys2,...,sysN,x0,t)** **initial(sys1,'PlotStyle1',..., sysN,'PlotStyleN',x0)** **[y,t,x] = initial(sys,x0)**	*Calcula y grafica condiciones iniciales de respuesta unitarias y en el tiempo t de uno y varios sistemas y con posibilidad de usar estilos gráficos. Se puede obtener el output de respuesta y, el vector de tiempo t usado para la simulación y el estado de las trayectorias x*
lsim(sys,u,t) **lsim(sys,u,t,x0)** **lsim(sys,u,t,x0,'zoh')** **lsim(sys,u,t,x0,'foh')** **lsim(sys1,sys2,...,sysN,u,t)** **lsim(sys1,sys2,...,sysN,u,t,x0)** **lsim(sys1,'PlotStyle1',...,sysN,' PlotStyleN',u,t)** **[y,t,x] = lsim(sys,u,t,x0)**	*Simula y grafica la respuesta del modelo LTI sys para inputs arbitrarios, siendo t el tiempo muestral, u los valores de input, xo una condición inicial para los estados del sistema y zoh y foh métodos de interpolación. Se puede simular la respuesta de varios sistemas a la vez, dar estilo a los gráficos y obtener el output de respuesta y, el vector de tiempo t usado para la simulación y el estado de las trayectorias x*
step(sys) **step(sys,t)** **step(sys1,sys2,...,sysN)** **step(sys1,sys2,...,sysN,t)** **step(sys1,'PlotStyle1',...,sysN,' PlotStyleN')** **[y,t,x] = step(sys)**	*Calcula el paso de respuesta del modelo LTI en condiciones iniciales de respuesta unitarias y en el tiempo t de uno y varios sistemas y con posibilidad de usar estilos gráficos. Se puede obtener el output de respuesta y, el vector de tiempo t usado para la simulación y el estado de las trayectorias x*
ltiview **ltiview(sys1,sys2,...,sysn)** **ltiview('plottype',sys1,sys2,...,sysn)** **ltiview('plottype',sys,extras)** **ltiview('clear',viewers)** ltiview('current',sys1,sys2,..., **sysn,viewers)**	*Abre el visor LTI para modelos de respuesta lineal para uno o varios sistemas y con distintas opciones gráficas definidas por plottype ('step', 'impulse', 'initial', 'lsim', 'pzmap', 'bode', 'nyquist', 'nichols' y 'sigma')*

Como primer ejemplo generamos y graficamos (Figura 12-22) una señal cuadrada con período 5 segundos, duración 30 segundos y muestreo cada 0,1 segundos.

```
>> [u,t] = gensig('square',5,30,0.1);
>> plot(t,u)
axis([0 30 -1 2])
```

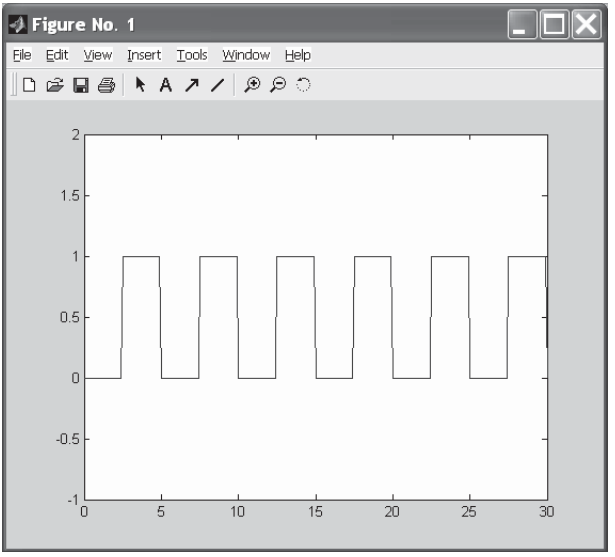


Figura 12-22

En el ejemplo siguiente graficamos (Figura 12-23) la respuesta del modelo de espacio de los estados siguiente:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-0.5572 & -0.7814\\ 0.7814 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

$$y = \begin{bmatrix}1.9691 & 6.4493\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

con condiciones iniciales

$$x(0) = \begin{bmatrix}1\\ 0\end{bmatrix}$$

```
>> a = [-0.5572   -0.7814;0.7814  0];
c = [1.9691  6.4493];
x0 = [1 ; 0]
sys = ss(a,[],c,[]);
initial(sys,x0)

x0 =

     1
     0
```

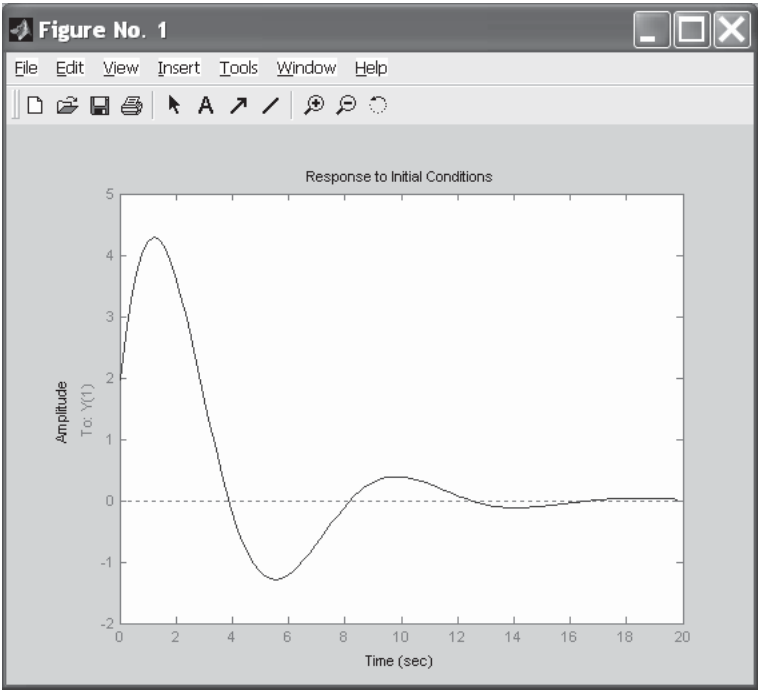


Figura 12-23

A continuación se grafica (Figura 12-24) el paso de respuesta para el modelo de espacio de los estados de segundo orden siguiente:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.5572 & -0.7814 \\ 0.7814 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$y = \begin{bmatrix} 1.9691 & 6.4493 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Se utilizará la sintaxis siguiente:

```
>> a = [-0.5572   -0.7814;0.7814  0];
b = [1 -1;0 2];
c = [1.9691  6.4493];
sys = ss(a,b,c,0);
step(sys)
```

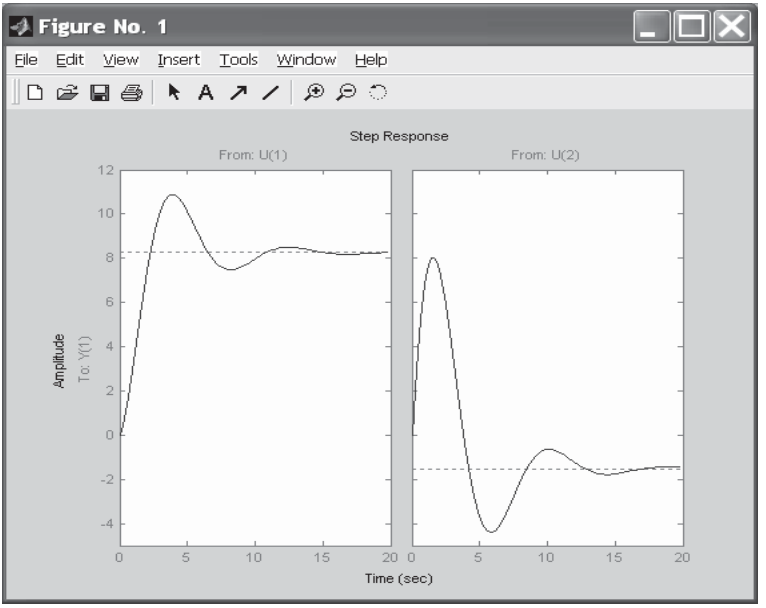


Figura 12-24

Comandos de frecuencia de respuesta

Comando	Descripción
S = allmargin(sys)	*Halla frecuencias cruzadas, ganancias asociadas, fase y márgenes desfasados para el sistema LTI sys*
bode(sys) **bode(sys,w)** **bode(sys1,sys2,...,sysN)** **bode(sys1,sys2,...,sysN,w)** **bode(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[mag,phase,w] = bode(sys)**	*Calcula un gráfico Bode de respuesta para el modelo LTI sys. Puede especificarse el rango de frecuencias w para los puntos del gráfico y condiciones de estilo para el mismo. Se puede calcular el gráfico Bode para varios sistemas a la vez y obtener la magnitud, fase y rango de frecuencias*
bodemag(sys) **bodemag(sys,{wmin,wmax})** **bodemag(sys,w)** **bodemag(sys1,sys2,...,sysN,w)** **bodemag(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')**	*Halla un gráfico Bode sólo con magnitud*
frsp = evalfr(sys,f)	*Evalúa la respuesta en frecuencia compleja simple f*
H = freqresp(sys,w)	*Evalúa frecuencia de respuesta para una malla w de frecuencias dadas*

isys = interp(sys,freqs)	*Interpola modelo FRD entre puntos de frecuencia freqs dados*
y = linspace(a,b) **y = linspace(a,b,n)**	*Crea un vector con 100 o n valores igualmente espaciados entre a y b*
y = logspace(a,b) **y = logspace(a,b,n)** **y = logspace(a,pi)**	*Crea un vector con espaciado logarítmico uniforme entre a y b (50 puntos entre 10^a y 10^b, n puntos entre 10^a y 10^b o puntos entre 10^a y π)*
[Gm,Pm,Wcg,Wcp] = margin(sys) **[Gm,Pm,Wcg,Wcp] = margin(mag,phase,w)** **margin(sys)**	*Calcula márgenes de ganancia Gm y fase Pm y las correspondientes frecuencias cruzadas Wcg y Wcp, pudiendo utilizar también un vector de frecuencias w. También puede obtenerse el gráfico Bode de sys*
ngrid	*Superpone líneas de malla en un gráfico de Nichols*
nichols(sys) **nichols(sys,w)** **nichols(sys1,sys2,...,sysN)** **nichols(sys1,sys2,...,sysN,w)** **nichols(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[mag,phase,w] = nichols(sys)** **[mag,phase] = nichols(sys,w)**	*Calcula gráficos de Nichols. Los argumentos tienen los mismos significados que para el gráfico Bode*
nyquist(sys) **nyquist(sys,w)** **nyquist(sys1,sys2,...,sysN)** **nyquist(sys1,sys2,...,sysN,w)** **nyquist(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[re,im,w] = nyquist(sys)** **[re,im] = nyquist(sys,w)**	*Calcula gráficos de Nyquist. Los argumentos tienen los mismos significados que para el gráfico Bode*
sigma(sys) **sigma(sys,w)** **sigma(sys,w,type)** **sigma(sys1,sys2,...,sysN)** **sigma(sys1,sys2,...,sysN,w)** **sigma(sys1,sys2,...,sysN,w,type)** **sigma(sys1,'PlotStyle1',...,** **sysN,'PlotStyleN')** **[sv,w] = sigma(sys)** **sv = sigma(sys,w)**	*Calcula gráficos de valores singulares*

Como primer ejemplo realizamos un gráfico de respuesta Bode (Figura 12-25) para el sistema continuo SISO siguiente:

$$H(s) = \frac{s^2 + 0.1s + 7.5}{s^4 + 0.12s^3 + 9s^2}$$

```
>> g = tf([1 0.1 7.5],[1 0.12 9 0 0]);
bode(g)
```

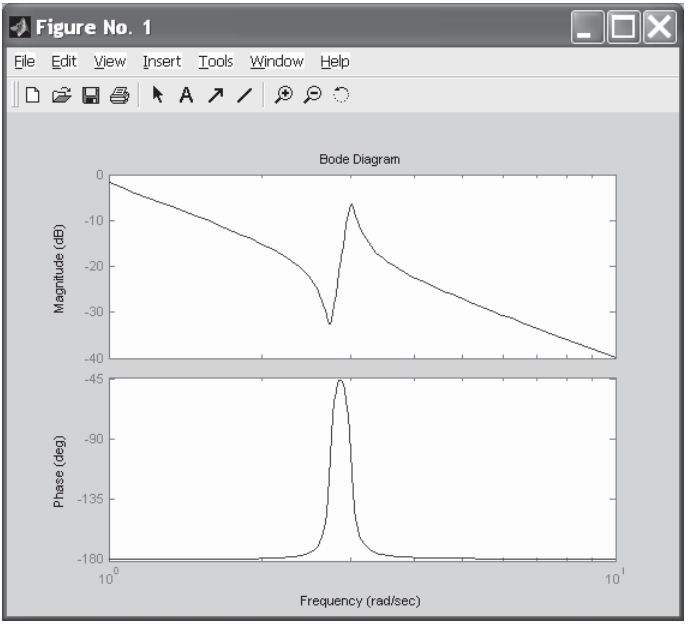


Figura 12-25

A continuación se evalúa en $z = 1+i$ la función de transferencia de tiempo discreto:

$$H(z) = \frac{z - 1}{z^2 + z + 1}$$

```
>> H = tf([1 -1],[1 1 1],-1)
z = 1+j
evalfr(H,z)

Transfer function:

   z - 1
-----------
z^2 + z + 1
```

```
Sampling time: unspecified
z =
   1.0000 + 1.0000i
ans =
   0.2308 + 0.1538i
```

A continuación se grafica la respuesta de Nichols con líneas de malla (Figura 12-26) para el sistema:

$$H(s) = \frac{-4s^4 + 48s^3 - 18s^2 + 250s + 600}{s^4 + 30s^3 + 282s^2 + 525s + 60}$$

```
>> H = tf([-4 48 -18 250 600],[1 30 282 525 60])
Transfer function:
-4 s^4 + 48 s^3 - 18 s^2 + 250 s + 600
--------------------------------------
 s^4 + 30 s^3 + 282 s^2 + 525 s + 60
>> nichols(H)
>> ngrid
```

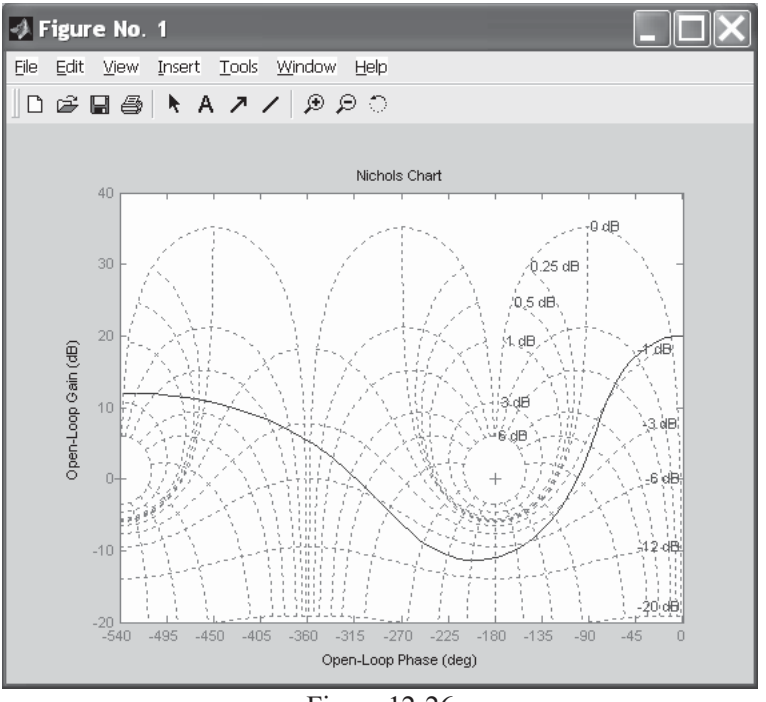


Figura 12-26

Comandos de ubicación de polos

Comando	Descripción
k = acker(A,b,p)	*Calcula ubicación de polos SISO (diseño) de* $\dot{x} = ax + bu$
K = place(A,B,p)	*Calcula ubicación de polos MIMO (diseño) de* $\dot{x} = ax + bu$
est = estim(sys,L) **est = estim(sys,L,sensors,known)**	*Produce un estimador de estado/output dado el modelo de planta/estado sys y el estimador de la ganancia L, pudiendo especificarse los outputs medidos (sensors) y los inputs conocidos (know)*
rsys = reg(sys,K,L) **rsys = reg(sys,K,L,sensors,known,controls)**	*Produce compensadores output-feedback dado el state-feedback K y la ganancia estimada L*

Comandos de diseño LQG

Comando	Descripción
[K,S,e] = lqr(A,B,Q,R) **[K,S,e] = lqr(A,B,Q,R,N)**	*Calcula ganancia LQ-óptima para modelos continuos*
[K,S,e] = dlqr(a,b,Q,R) **[K,S,e] = dlqr(a,b,Q,R,N)**	*Calcula ganancia LQ-óptima para modelos discretos*
[K,S,e] = lqry(sys,Q,R) **[K,S,e] = lqry(sys,Q,R,N)**	*Calcula ganancia LQ-óptima con output ponderado*
[Kd,S,e] = lqrd(A,B,Q,R,Ts) **[Kd,S,e] = lqrd(A,B,Q,R,N,Ts)**	*Calcula ganancia discreta LQ para modelos continuos*
[kest,L,P] = kalman(sys,Qn,Rn,Nn) **[kest,L,P,M,Z] = kalman(sys,Qn,Rn,Nn)**	*Calcula el estimador de Kalman para modelos discretos y continuos*
[kest,L,P,M,Z] = kalmd(sys,Qn,Rn,Ts)	*Calcula el estimador discreto de Kalman para modelos continuos*
rlqg = lqgreg(kest,k) **rlqg = lqgreg(kest,k,controls)**	*Forma el regulador LQG dada la ganancia LQ y el filtro de Kalman*

Comandos de solución de ecuaciones

Comando	Descripción
[X,L,G,rr] = care(A,B,Q) **[X,L,G,rr] = care(A,B,Q,R,S,E)** **[X,L,G,report] = care(A,B,Q,...,'report')** **[X1,X2,L,report] = care(A,B,Q,...,'implicit')**	*Resuelve ecuaciones algebraicas de Riccati en tiempo continuo*
[X,L,G,rr] = dare(A,B,Q,R) **[X,L,G,rr] = dare(A,B,Q,R,S,E)** **[X,L,G,report] = dare(A,B,Q,...,'report')** **[X1,X2,L,report] = dare(A,B,Q,...,'implicit')**	*Resuelve ecuaciones algebraicas de Riccati en tiempo discreto*
X = lyap(A,Q) **X = lyap(A,B,C)**	*Resuelve ecuaciones de tiempo continuo de Lyapunov*
X = dlyap(A,Q)	*Resuelve ecuaciones de tiempo discreto de Lyapunov*

Como ejemplo resolvemos la ecuación de Riccati:

$$A^T X + XA - XBR^{-1}B^T X + C^T C = 0$$

donde:

$$A = \begin{bmatrix} -3 & 2 \\ 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad R = 3$$

```
>> a = [-3 2;1 1]; b = [0 ; 1]; c = [1 -1]; r = 3;
[x,l,g] = care(a,b,c'*c,r)

x =

    0.5895    1.8216
    1.8216    8.8188

l =

   -3.5026
   -1.4370

g =

    0.6072    2.9396
```

Ejercicio 12-1. Crear el modelo continuo de espacio de los estados y posteriormente computar la realización del espacio de los estados para la función de transferencia H(s) que se define en este problema. Obtener también una realización mínima de H(s)

$$H(s) = \begin{bmatrix} \dfrac{s+1}{s^3+3s^2+3s+2} \\ \dfrac{s^2+3}{s^2+s+1} \end{bmatrix}$$

```
>> H = [tf([1 1],[1 3 3 2]) ; tf([1 0 3],[1 1 1])];
>> sys = ss(H)

a =
                        x1                  x2                  x3
x4            x5
           x1                -3              -1.5                -1
0             0
           x2                 2                 0                 0
0             0
           x3                 0                 1                 0
0             0
           x4              0             0             0             -
1          -0.5
           x5                 0                 0                 0
2             0

b =
                       u1
           x1           1
           x2           0
           x3           0
           x4           1
           x5           0

c =
                    x1          x2          x3          x4          x5
         y1          0         0.5         0.5          0           0
         y2          0           0           0          -1           1

d =
                       u1
           y1           0
           y2           1

 Continuous-time model.
```

```
>> size(sys)

State-space model with 2 outputs, 1 input, and 5 states.
```

Se ha obtenido un modelo de espacio de los estados con 2 outputs, 1 input y 5 estados. La realización mínima de $H(s)$ se obtiene mediante la sintaxis:

```
>> sys = ss(H,'min')

a =
                x1          x2          x3
   x1      -1.4183     -1.5188     0.21961
   x2     -0.14192     -1.7933    -0.70974
   x3     -0.44853      1.7658     0.21165

b =
                u1
   x1      0.19787
   x2       1.4001
   x3      0.02171

c =
                x1          x2          x3
   y1     -0.15944    0.018224     0.27783
   y2      0.35997    -0.77729     0.78688

d =
                u1
   y1            0
   y2            1

Continuous-time model.

>> size(sys)

State-space model with 2 outputs, 1 input, and 3 states.
```

Como realización mínima se ha obtenido un modelo de espacio de los estados con 2 outputs, 1 input y 3 estados.

El resultado está de acuerdo con la factorización de $H(s)$ como producto de un sistema de primer orden con uno de segundo orden

$$H(s) = \begin{bmatrix} \dfrac{1}{s+2} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dfrac{s+1}{s^2+s+1} \\ \dfrac{s^2+3}{s^2+s+1} \end{bmatrix}$$

Ejercicio 12-2. Especificar la función de transferencia discreta MIMO definida como H(z) en el problema con tiempo de muestra de 0,2 segundos.

$$H(z) = \begin{bmatrix} \dfrac{1}{z+0.3} & \dfrac{z}{z+0.3} \\ \dfrac{-z+2}{z+0.3} & \dfrac{3}{z+0.3} \end{bmatrix}$$

```
>> nums = {1 [1 0];[-1 2] 3}
Ts = 0.2
H = tf(nums,[1 0.3],Ts)

nums =

    [      1.00]    [1x2 double]
    [1x2 double]    [      3.00]


Ts =

          0.20


Transfer function from input 1 to output...
          1
 #1:  -------
      z + 0.3

      -z + 2
 #2:  -------
      z + 0.3

Transfer function from input 2 to output...
          z
 #1:  -------
      z + 0.3

          3
 #2:  -------
      z + 0.3

Sampling time: 0.2
```

Ejercicio 12-3. Dado el modelo cero-polo-ganancia:

$$H(z)=\frac{z-0.7}{z-0.5}$$

y con tiempo de muestreo 0,01, se trata de realizar remuestreo a 0,05 segundos. Posteriormente deshacer el remuestreo para comprobar que se obtiene el modelo original.

```
>> H = zpk(0.7,0.5,1,0.1)
H2 = d2d(H,0.05)

Zero/pole/gain:

(z-0.7)
-------
(z-0.5)

Sampling time: 0.1

Zero/pole/gain:

(z-0.8243)
----------
(z-0.7071)

Sampling time: 0.05
```

Ahora deshacemos el remuestreo de la forma siguiente:

```
>> d2d(H2,0.1)

Zero/pole/gain:

(z-0.7)
-------
(z-0.5)

Sampling time: 0.1
```

Se observa que se obtiene el modelo original.

Ejercicio 12-4. Considerar el modelo continuo de cuarto orden dado por la función de transferencia h(s) definida en el problema. Se trata de reducir su orden eliminando los estados correspondientes a los valores más pequeños del vector g de valores de la diagonal grammiana balanceada. Comparar los modelos original y reducido.

$$h(s) = \frac{s^3 + 11s^2 + 36s + 26}{s^4 + 14.6s^3 + 74.96s^2 + 153.7s + 99.65}$$

Comenzamos definiendo el modelo y computando una realización balanceada del espacio de los estados como sigue:

```
>> h = tf([1 11 36 26],[1 14.6 74.96 153.7 99.65])
[hb,g] = balreal(h)
g'

 Transfer function:

          s^3 + 11 s^2 + 36 s + 26
-----------------------------------------
s^4 + 14.6 s^3 + 74.96 s^2 + 153.7 s + 99.65

a =
             x1        x2        x3        x4
   x1    -3.601   -0.8212   -0.6163   0.05831
   x2    0.8212    -0.593    -1.027   0.09033
   x3   -0.6163     1.027    -5.914     1.127
   x4  -0.05831   0.09033    -1.127    -4.492

b =
              u1
   x1     -1.002
   x2     0.1064
   x3   -0.08612
   x4  -0.008112

c =
             x1        x2        x3        x4
   y1    -1.002   -0.1064  -0.08612  0.008112

d =
       u1
   y1   0

Continuous-time model.

g =

    0.1394
    0.0095
    0.0006
    0.0000
```

```
ans =

    0.1394    0.0095    0.0006    0.0000
```

Ahora eliminamos los tres estados relativos a los tres últimos valores de g usando dos métodos distintos.

```
>> hmdc = modred(hb,2:4,'mdc')
hdel = modred(hb,2:4,'del')

a =
           x1
   x1  -4.655

b =
           u1
   x1  -1.139

c =
           x1
   y1  -1.139

d =
             u1
   y1  -0.01786

Continuous-time model.

a =
           x1
   x1  -3.601

b =
           u1
   x1  -1.002

c =
           x1
   y1  -1.002

d =
       u1
   y1   0

Continuous-time model.
```

A continuación comparamos las respuestas con el modelo original (Figura 12- 27).

```
>> bode(h,'-',hmdc,'x',hdel,'*')
```

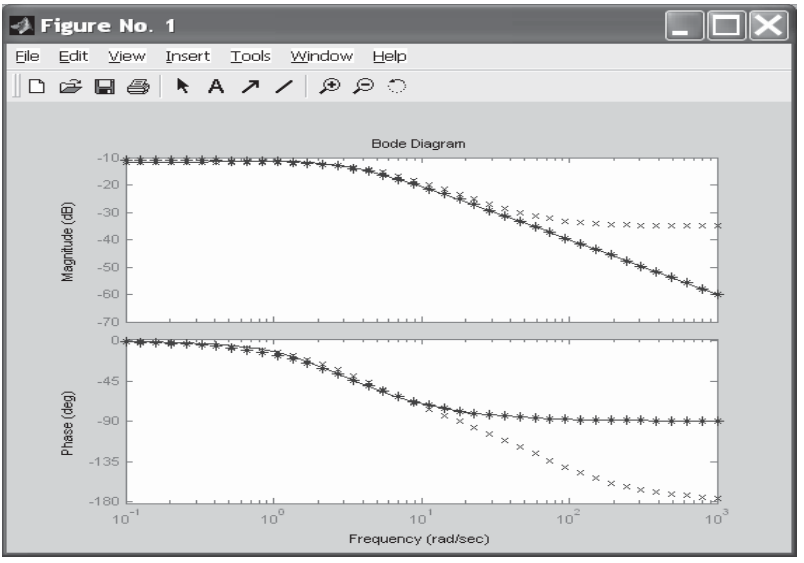


Figura 12-27

Resulta que en ambos casos el modelo reducido es mejor que el original. Ahora comparamos los pasos de respuesta (Figura 12-28)

```
>> step(h,'-',hmdc,'-.',hdel,'--')
```

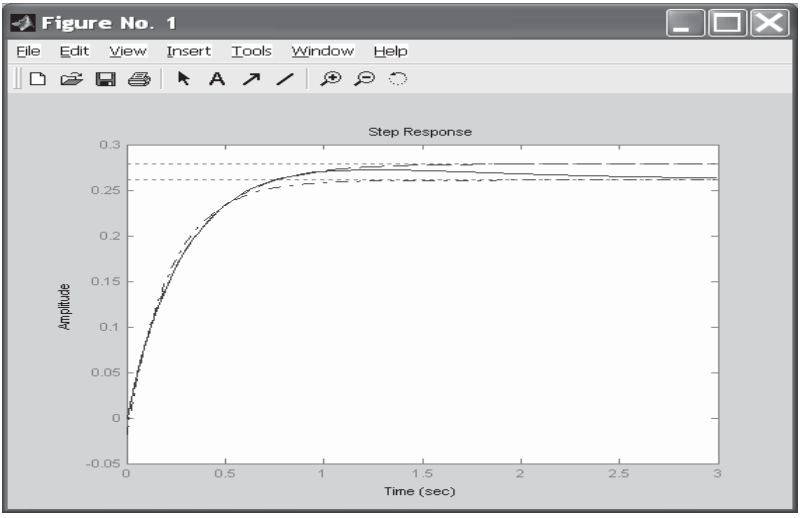


Figura 12-28

Ejercicio 12-5. Calcular la covarianza del output de respuesta del sistema discreto SISO, con H(z) y Ts definidos en el problema, correspondiente a un ruido blanco gaussiano de intensidad W=5.

$$H(z) = \frac{2z+1}{z^2+0.2z+0.5}, \qquad T_s = 0.1$$

```
>> sys = tf([2 1],[1 0.2 0.5],0.1)

Transfer function:
     2 z + 1
-----------------
z^2 + 0.2 z + 0.5

Sampling time: 0.1
>> p = covar(sys,5)

p =

   30.3167
```

Ejercicio 12-6. Graficar polos y ceros del sistema de tiempo continuo definido por la función de transferencia:

$$H(s) = \frac{2s^2+5s+1}{s^2+2s+3}$$

```
>> H = tf([2 5 1],[1 2 3])
 Transfer function:

2 s^2 + 5 s + 1
---------------
 s^2 + 2 s + 3

>> pzmap(H)
>> sgrid
```

La Figura 12-29 presenta el resultado.

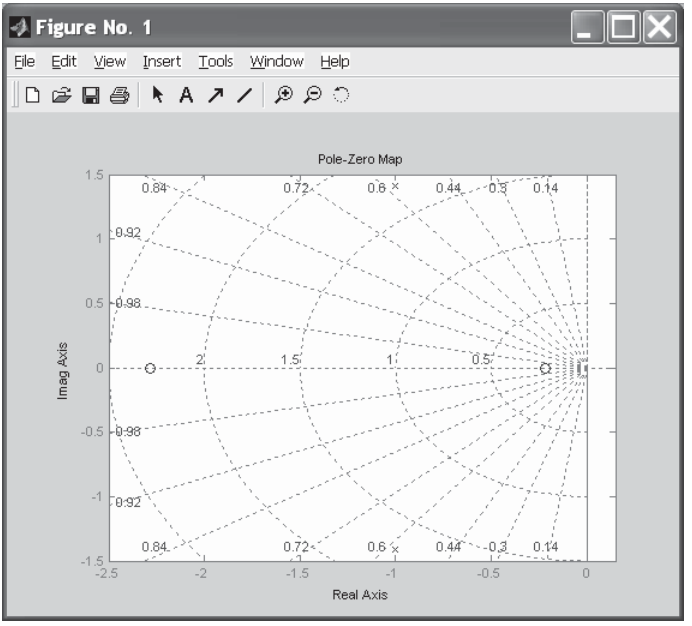


Figura 12-29

Ejercicio 12-7. Consideramos el diagrama de la Figura 12-30 en el que las matrices del espacio de los estados del modelo sys2 son:

$$A = [\ -9.0201,\ 17.7791;\ -1.6943,\ 3.2138\ ];$$
$$B = [\ -.5112,\ .5362;\ -.002\ -1.8470];$$
$$C = [\ -3.2897,\ 2.4544;\ -13.5009,\ 18.0745];$$
$$D = [-.5476,\ -.1410;\ -.6459,\ .2958\ ];$$

Se trata en primer lugar de unir estos bloques de forma no conectada, y en segundo lugar se trata de obtener el modelo de espacio de los estados para la interconexión global a través de la matriz Q = [3,1,-4;4,3,0] con inputs = [1,2] y outputs = [2,3].

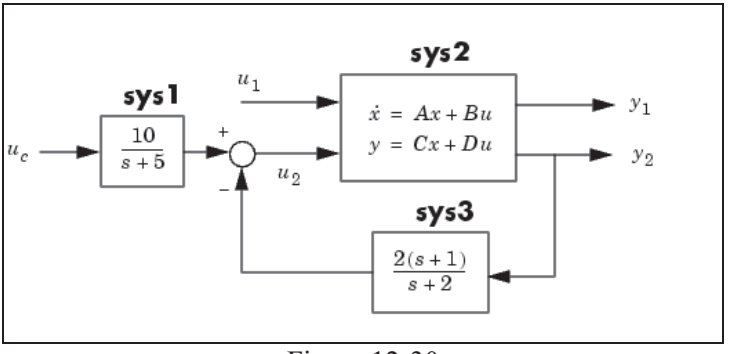


Figura 12-30

La unión se realiza mediante la siguiente sintaxis:

```
>> A = [ -9.0201,  17.7791; -1.6943  3.2138 ];
B = [ -.5112,  .5362;  -.002  -1.8470];
C = [ -3.2897,  2.4544;  -13.5009  18.0745];
D = [-.5476,  -.1410;  -.6459  .2958 ];
>> sys1 = tf(10,[1 5],'inputname','uc')
sys2 = ss(A,B,C,D,'inputname',{'u1' 'u2'},...
                    'outputname',{'y1' 'y2'})
sys3 = zpk(-1,-2,2)

Transfer function from input "uc" to output:

 10
-----
s + 5

a =
           x1      x2
   x1   -9.02   17.78
   x2  -1.694   3.214

b =
            u1       u2
   x1  -0.5112   0.5362
   x2   -0.002   -1.847

c =
          x1     x2
   y1  -3.29  2.454
   y2  -13.5  18.07

d =
            u1       u2
   y1  -0.5476   -0.141
   y2  -0.6459   0.2958

Continuous-time model.

Zero/pole/gain:

2 (s+1)
-------
 (s+2)
```

La unión de bloques de forma no conectada puede realizarse como sigue:

```
sys = append(sys1,sys2,sys3)
```

```
a =
           x1       x2       x3       x4
   x1      -5        0        0        0
   x2       0    -9.02    17.78        0
   x3       0   -1.694    3.214        0
   x4       0        0        0       -2

b =
           uc       u1       u2        ?
   x1       4        0        0        0
   x2       0  -0.5112   0.5362        0
   x3       0   -0.002   -1.847        0
   x4       0        0        0    1.414

c =
           x1       x2       x3       x4
   ?      2.5        0        0        0
   y1       0    -3.29    2.454        0
   y2       0    -13.5    18.07        0
   ?        0        0        0   -1.414

d =
           uc       u1       u2        ?
   ?        0        0        0        0
   y1       0  -0.5476   -0.141        0
   y2       0  -0.6459   0.2958        0
   ?        0        0        0        2

Continuous-time model.
```

A continuación se obtiene el modelo de espacio de los estados para la interconexión global.

```
>> Q = [3, 1, -4; 4, 3, 0];
>> inputs = [1 2];
>> outputs = [2 3];
>> sysc = connect(sys,Q,inputs,outputs)

a =
           x1       x2       x3       x4
   x1      -5        0        0        0
   x2   0.8422  0.07664    5.601   0.4764
   x3   -2.901   -33.03    45.16   -1.641
   x4   0.6571      -12    16.06   -1.628

b =
           uc       u1
   x1       4        0
   x2       0   -0.076
   x3       0   -1.501
   x4       0  -0.5739
```

```
c =
            x1        x2        x3        x4
   y1  -0.2215    -5.682     5.657   -0.1253
   y2   0.4646    -8.483     11.36    0.2628

d =
            uc        u1
   y1        0    -0.662
   y2        0   -0.4058

Continuous-time model.
```

Ejercicio 12-8. Graficar el impulso respuesta unitario del modelo de espacio de los estados de segundo orden definido a continuación en este problema y almacenar los resultados en un array con output de respuesta y tiempo de simulación.

El modelo a que hace referencia este problema es el siguiente:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-0.5572 & -0.7814\\ 0.7814 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1 & -1\\ 0 & 2\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

$$y = \begin{bmatrix}1.9691 & 6.4493\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

La graficación pedida (Figura 12- 31) se obtiene mediante la siguiente sintaxis:

```
>> a = [-0.5572 -0.7814;0.7814  0];
b = [1 -1;0 2];
c = [1.9691  6.4493];
sys = ss(a,b,c,0);
impulse(sys)
```

El output de respuesta y tiempo de simulación se obtienen mediante la sintaxis:

```
>> [y,t] = impulse(sys)

y(:,:,1) =

    1.9691
    2.6831
    3.2617
    3.7059
    4.0197
    4.2096
      .
      .
```

```
y(:,:,2) =

   10.9295
    9.4915
    7.9888
    6.4622
    4.9487
       .
       .
```

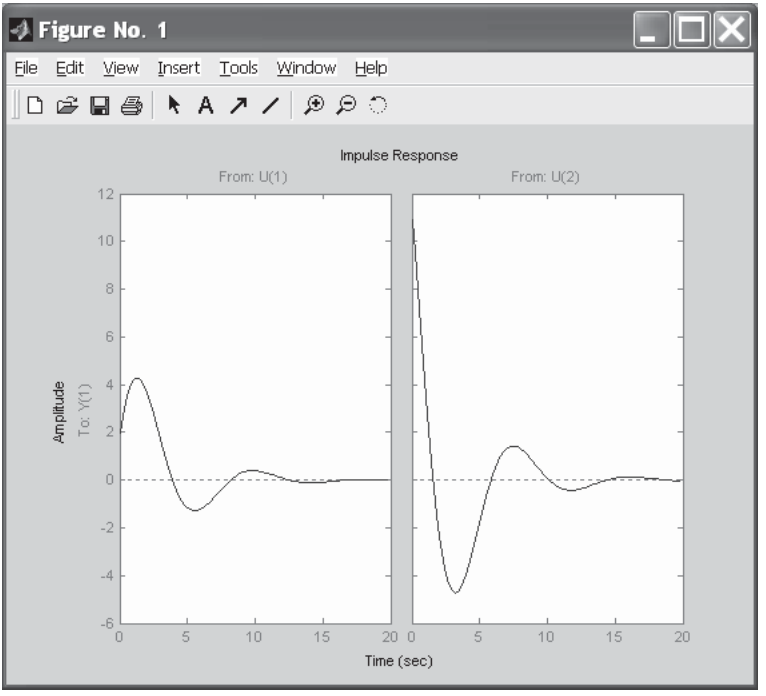


Figura 12-31

Ejercicio 12-9. Graficar y simular la respuesta del sistema con función de transferencia H(s) definida en el problema a la señal cuadrada de período 4 segundos con muestreo cada 0,1 segundos y cada 10 segundos.

$$H(s) = \begin{bmatrix} \dfrac{2s^2 + 5s + 1}{s^2 + 2s + 3} \\ \dfrac{s - 1}{s^2 + s + 5} \end{bmatrix}$$

Comenzamos generando la señal cuadrada con *gensys* y luego realizamos la simulación con *lsim* (Figura 12-32) como sigue:

```
>> [u,t] = gensig('square',4,10,0.1);
>> H = [tf([2 5 1],[1 2 3]) ; tf([1 -1],[1 1 5])]
lsim(H,u,t)

Transfer function from input to output...

      2 s^2 + 5 s + 1
 #1:  ---------------
       s^2 + 2 s + 3

        s - 1
 #2:  -----------
      s^2 + s + 5
```

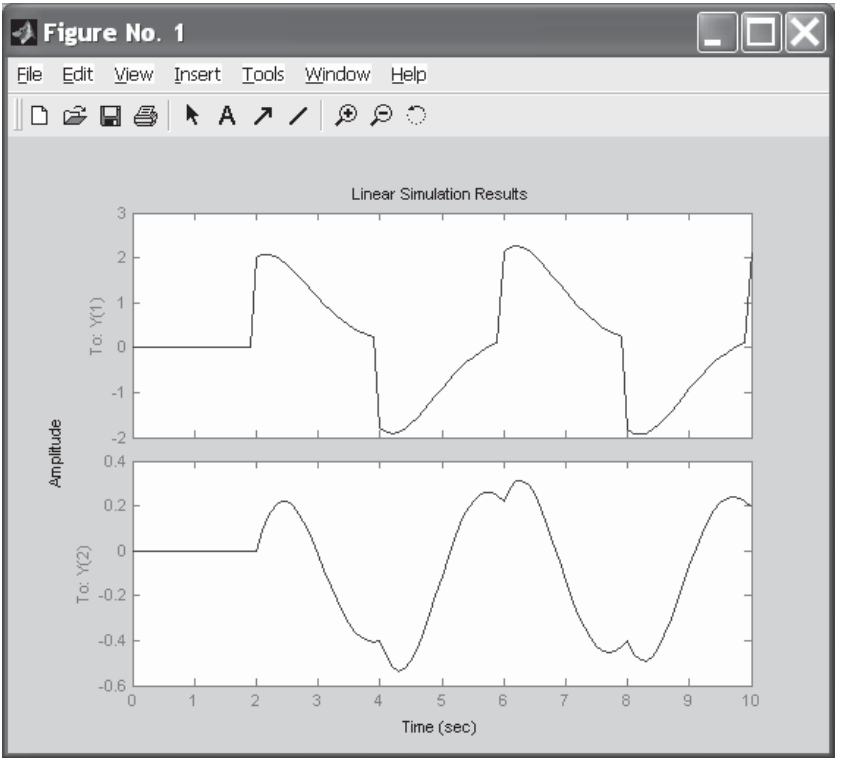


Figura 12-32

13

Control predictivo y robusto

13.1 Estrategias de control predictivo: *Model Predictive Control Toolbox*

Model Predictive Control Toolbox es un conjunto completo de herramientas para implementar estrategias de control de modelos predictivos. Los métodos de control de modelos predictivos son utilizados en ingeniería química y en procesos continuos de control en otras industrias.

Las características más importantes de este toolbox son las siguientes:

- Modelado, identificación y validación.
- Soporte para MISO, MIMO, paso de respuesta y modelo de espacio de los estados.
- Análisis de sistemas.
- Modelo de conversión entre espacio de los estados, función de transferencia y paso de respuesta.

El control de modelos predictivos aproxima un modelo lineal dinámico de planta para predecir el efecto de movimientos futuros y variables manipuladas. El problema de optimización *online* se formula como programación cuadrática que se resuelve repetidamente utilizando las medidas más recientes.

Model Predictive Control Toolbox incluye más de 50 funciones especializadas de MATLAB para realizar diseño, análisis y simulación de sistemas dinámicos utilizando aproximación de control de modelos predictivos. El toolbox soporta los formatos de respuesta de paso finito (o impulso), función de transferencia de tiempo discreto y continuo y espacio de los estados. El toolbox maneja sistemas no cuadrados y soporta variedad de técnicas de estimación de estado. Las herramientas de simulación permiten testear la respuesta para sistemas con o sin restricciones. Para la identificación de modelos, el toolbox dispone de una función de interfaz que hace fácil utilizar modelos desarrollados con el toolbox de identificación de sistemas.

Comandos de identificación

[ax,mx,stdx] = autosc(x)	*Escala automáticamente una matriz por sus columnas de medias mx y desviaciones típicas stdx*
sx = scal(x,mx)	*Escala una matriz por sus columnas de medias*
sx = scal(x,mx,stdx)	*Escala una matriz por sus columnas de medias y desviaciones típicas*
rx = rescal(x,mx)	*Reescala una matriz por sus columnas de medias*
rx = rescal(x,mx,stdx)	*Reescala una matriz por sus columnas de medias y desviaciones típicas*
plant = imp2step(delt,nout,theta1, theta2, ..., theta25)	*Construye un modelo MIMO (multi-input multi-output) en formato paso MPC a partir de matrices $theta_i$ de modelos de respuesta a impulsos MISO (multi-input single-output) siendo delt el intervalo de muestreo y nout el indicador de estabilidad de salida*
[theta, yres] = mlr(xreg,yreg,ninput) **[theta, yres] = mlr(xreg,yreg,ninput,plotopt, wtheta, wdeltheta)**	*Calcula modelos de respuesta a impulso MISO vía regresión lineal multivariante, donde xreg e yreg son la matriz de inputs y el vector de outputs producidos por rutinas como wtreg, ninput es el número de inputs, las columnas de theta son los coeficientes de respuesta a impulsos de cada input e yres es un vector de residuos*
[theta,w,cw,ssqdif,yres] = plsr(xreg,yreg,ninput,lv) **[theta,w,cw,ssqdif,yres] = plsr(xreg,yreg,ninput,lv, plotopt)**	*Calcula modelos de respuesta a implusos MISO vía regresión parcial de mínimos cuadrados*
yres = validmod(xreg,yreg,theta) **yres = validmod(xreg,yreg,theta,plotopt)**	*Valida modelos de respuesta a impulsos MISO para un nuevo conjunto de datos*
[xreg,yreg] = wrtreg(x,y,n)	*Escribe matrices de datos para ser utilizadas en regresión.*

Comandos de graficado de la matriz de información

mpcinfo(mat)	*Determina el tipo de matriz y devuelve información sobre ella*
plotall(y,u) **plotall(y,u,t)**	*Grafica outputs e inputs de ejecuciones de simulación, donde y y u son matrices de outputs y variables manipuladas (t=periodo).*
plotfrsp(vmat) **plotfrsp(vmat,out,in)**	*Grafica frecuencias de respuesta de un sistema como un gráfico Bode, siendo vmat la matriz que contiene los datos*
ploteach(y) **ploteach(y,u)** **ploteach([ ],u)** **ploteach(y,[ ],t)** **ploteach([ ],u,t)** **ploteach(y,u,t)**	*Separa gráficos de outputs e inputs en ejecuciones de simulación, siendo y y u matrices de outputs y variables manipuladas, respectivamente (t=período).*
plotstep(plant) **plotstep(plant,opt)**	*Grafica los coeficientes de un modelo en formato paso MPC, dónde plant es la matriz de respuesta creada por mod2step, ss2step o tfd2step y opt es un escalar que selecciona distintas salidas gráficas*

Comandos de conversión de modelos

c2dmp	*Convierte el modelo de espacio de los estados de continuo a discreto. (Equivale a c2d en el Control System Toolbox)*
[numd,dend] = cp2dp(num,den,delt) **[numd,dend] = cp2dp(num,den,delt,delay)**	*Convierte de continua a discreta la función de transferencia en formato poly (delt es el período de muestreo y delay es el desfase)*
d2cmp	*Convierte el modelo de espacio de los estados de discreto a continuo. (Equivale a d2cen el Control System Toolbox)*
newmod = mod2mod(oldmod,delt2)	*Cambia el período de muestreo de un modelo MPC en formato mod (delt2 es el nuevo período)*
[phi,gam,c,d] = mod2ss(mod) **[phi,gam,c,d,minfo] = mod2ss(mod)**	*Convierte un modelo MPC en formato mod a un modelo de espacio de los estados*
plant = mod2step(mod,tfinal) **[plant,dplant] = mod2step(mod,tfinal,delt2,nout)**	*Convierte un modelo MPC en formato mod a un modelo MPC en formato paso.*
g = poly2tfd(num,den) **g = poly2tfd(num,den,delt,delay)**	*Convierte una función de transferencia en formato poly en modelo MPC en formato tf*
pmod = ss2mod(phi,gam,c,d) **pmod = ss2mod(phi,gam,c,d,minfo)**	*Convierte un modelo de espacio de los estados a modelo MPC en formato mod*

plant = ss2step(phi,gam,c,d,tfinal) **plant = ss2step(phi,gam,c,d,tfinal,delt1,delt2,nout)**	*Convierte un modelo de espacio de los estados a modelo MPC en formato paso*
ss2tf2	*Convierte un modelo de espacio de los estados a function de transferencia. (Equivale a ss2tf en el Control System Toolbox)*
tf2ssm	*Convierte function de transferencia a modelo de espacio de los estados. (Equivale a a tf2ss en el Control System Toolbox)*
model = tfd2mod(delt2,ny,g1,g2,g3,...,g25)	*Convierte un modelo MPC en formato tf a modelo MPC en formato mod.*
plant = tfd2step(tfinal,delt2,nout,g1) **plant = tfd2step(tfinal,delt2,nout,g1,...,g25)**	*Convierte un modelo MPC en formato tf a modelo MPC en formato paso*
umod = th2mod(th) **[umod,emod] = th2mod(th1,th2,...,thN)**	*Convierte un modelo en formato theta (System Identification Toolbox) a MPC en formato mod.*

Comandos de construcción de modelos – MPC formato mod

model = addmd(pmod,dmod)	*Añade una o más disturbancias medidas a un modelo de planta*
pmod = addmod(mod1,mod2)	*Combina dos modelos añadiendo el output del primero al input del segundo.*
model = addumd(pmod,dmod)	*Añade una o más disturbancias no medidas a un modelo de planta*
pmod = appmod(mod1,mod2)	*Añade dos modelos en estructura paralela no conectada*
pmod = paramod(mod1,mod2)	*Sitúa dos modelos en paralelo compartiendo output común*
pmod = sermod(mod1,mod2)	*Sitúa dos modelos en serie*

Comandos de control de diseño y simulación – MPC formato paso

yp = cmpc(plant,model,ywt,uwt,M,P,tend,r) **[yp,u,ym] = cmpc(plant,model,ywt,uwt,M,P,tend,...**	*Resuelve un problema de programación cuadrática para simular la representación de un sistema en bucle cerrado con input y output constantes*
[clmod] = mpccl(plant,model,Kmpc) **[clmod,cmod] = mpccl(plant,model,Kmpc,tfilter,... dplant, dmodel)**	*Crea un modelo MPC en formato mod de un sistema en bucle cerrado con un controlador MPC sin restricción.*
Kmpc = mpccon(model) **Kmpc = mpccon(model,ywt,uwt,M,P)**	*Calcula la matriz de ganancia (controlador sin restricciones) para MPC*
yp = mpcsim(plant,model,Kmpc,tend,r) **[yp,u,ym] = mpcsim(plant,model,Kmpc,tend,r,usat,... tfilter,dplant,dmodel,dstep)**	*Simula un sistema en bucle cerrado con restricciones de saturación opcionales en las variables manipuladas*
nlcmpc	*Simula S-function block para controlador MPC con restricciones input y output (resuelve el programa cuadrático).*
nlmpcsim	*Simula S-function block para controlador MPC con restricciones opcionales de saturación*

Comandos de control de diseño y simulación – MPC formato mod

yp = scmpc(pmod,imod,ywt,uwt,M,P,tend,r) **[yp,u,ym] = scmpc(pmod,imod,ywt,uwt,M,P,tend, ... r,ulim,ylim,Kest,z,d,w,wu)**	*Resuelve un problema de programación cuadrática para simular la representación de un sistema en bucle cerrado con input y output constantes*
[clmod,cmod] = smpccl(pmod,imod,Ks) **[clmod,cmod] = smpccl(pmod,imod,Ks,Kest)**	*Crea un modelo MPC en formato mod de un sistema en bucle cerrado con un controlador MPC sin restricción*
Ks = smpccon(imod) **Ks = smpccon(imod,ywt,uwt,M,P)**	*Calcula la matriz de ganancia (controlador sin restricciones) para MPC*
[Kest] = smpcest(imod,Q,R)	*Diseña un estimador de estado para uso en MPC*

yp = smpcsim(pmod,imod,Ks,tend,r) **[yp,u,ym] =** **smpcsim(pmod,imod,Ks,tend,r,usat,...** **Kest,z,d,w,wu)**	*Simula un sistema en bucle cerrado con restricciones de saturación opcionales en las variables manipuladas*

Comandos de análisis

frsp = mod2frsp(mod,freq) **[frsp,eyefrsp] =** **mod2frsp(mod,freq,out,in,balflg)**	*Calcula frecuencias de respuesta para un sistema en formato mod MPC*
g = smpcgain(mod) **poles = smpcpole(mod)**	*Calcula matriz de ganancia de los estados en un sistema MPC en formato mod*
smpcpole	*Calcula polos de un sistema en formato MPC mod*
[sigma,omega] = svdfrsp(vmat)	*Calcula valores singulares de frecuencia de respuesta*

13.2 Sistemas de control robustos: *Robust Control Toolbox*

Robust Control Toolbox aporta herramientas para el diseño y análisis de sistemas de control multivariantes robustos. Incluye sistemas en los que es posible modelizar errores, sistemas dinámicos que no son completamente conocidos o con parámetros que pueden variar durante la duración del producto. Los potentes algoritmos de este toolbox permiten ejecutar cálculos complejos considerando un número importante de variaciones en los parámetros.

Las características más importantes de este toolbox son las siguientes:

- Control H2 y H_∞ basado en LQG (síntesis).
- Frecuencia de respuesta multivariable.
- Construcción del modelo de espacio de los estados.
- Valores singulares basados en modelos de conversión.
- Reducción de modelos de alto orden.
- Factorización espectral e *inner/outer*.

Comandos para estructura de datos opcional del sistema

[b1,b2,,bn] = branch(tr,PATH1,PATH2,,PATHN)	*Extrae ramas de un árbol determinadas por los caminos PATH1,PATH2,,PATHN*
TR = graft(TR1,B) **TR = graft(TR1,B,NM)**	*Añade la rama B a la variable árbol TR1. NM sería el nombre de la nueva rama*
[I,TY,N] = issystem(S)	*Identifica variables del sistema*
[i] = istree(T) **[i,b] = istree(T,path)**	*Identifica variables de árbol con estructura T*
S = mksys(a,b,c,d) **S = mksys(v1,v2,v3,vn, TY)**	*Construye variables de árbol para el sistema*
T = tree(nm,b1,b2,,bn)	*Construye una variable árbol*
[VARS,N] = vrsys(NAM)	*Da nombres de variables de sistema estandarizado*

Comandos para construcción de modelos

[a,b1,b2,c1,c2,d11,d12,d21,d22] = ... **augss(ag,bg,,aw1,bw1,,aw2,bw2,,aw3,bw3,)** **[a,b1,b2,c1,c2,d11,d12,d21,d22] = ...** **augss(ag,bg,,aw1,bw1,,aw2,bw2,,aw3,bw3,,w3poly)** **[a,b1,b2,c1,c2,d11,d12,d21,d22] = ...** **augtf(ag,bg,cg,dg,w1,w2,w3)** **[tss] = augss(ssg,ssw1,ssw2,ssw3,w3poly)** **[tss] = augtf(ssg,w1,w2,w3)** **[tss] = augss(ssg,ssw1,ssw2,ssw3)**	*Aumento de planta en función de transferencia o espacio de los estados para uso en diseños mezclados ponderados H_2 y H_∞ (pesos en e, u e y)*
[acl,bcl,ccl,dcl] = interc(a,b,c,d,m,n,f) **[sscl] = interc(ss,m,n,f)**	*Interconexión general multivariable de sistemas*

Comandos para conversión de modelos

[ab,bb,cb,db] = bilin(a,b,c,d,ver,Type,aug) **[ssb] = bilin(ss,ver,Type,aug)**	*Transformación bilineal multivariable de frrecuencias (s o z) donde ver denota la dirección de la transformación (1 o -1), type denota el tipo de transformación ('Tustin', 'P_Tust', 'BwdRec', FwdRec, 'S_Tust' 'S_ftjw' o 'G_Bilin') y aug = [$\alpha,\beta,\gamma,\delta$] donde s = $(\alpha z+\delta)/(\beta+\gamma z)$*

[aa,bb,cc,dd] = des2ss(a,b,c,d,E,k) **[ss1] = des2ss(ss,E,k)**	*Convierte el descriptor del sistema al espacio de los estados vía SVD*
[a,b1,b2,c1,c2,d11,d12,d21,d22] = lftf(A,B1,B2,,a,b1,b2,) **[aa,bb,cc,dd] = lftf(a,b1,b2,c1,c2,d11,d12,d21,d22,aw,bw,cw,dw)** **[aa,bb,cc,dd] = lftf(aw,bw,cw,dw,a,b1,b2,c1,c2,d11,d12,d21,d22)** **tss = lftf(tss1,tss2)** **ss = lftf(tss1,ss2)** **ss = lftf(ss1,tss2)**	*Transformación lineal fraccional*
[ag,bg1,,dg22,at,bt1,,dt21,dt22] = sectf(af,bf1,,df22,secf,secg) **[ag,bg,cg,dg,at,bt1,,dt21,dt22] = sectf(af,bf,cf,df,secf,secg)** **[tssg,tsst] = sectf(tssf,secf,secg)** **[ssg,tsst] = sectf(ssf,secf,secg)**	*Transformación bilineal sector del espacio de los estados*
[a1,b1,c1,d1,a2,b2,c2,d2,m] = stabproj(a,b,c,d) **[a1,b1,c1,d1,a2,b2,c2,d2] = slowfast(a,b,c,d,cut)** **[ss1,ss2,m] = stabproj(ss)** **[ss1,ss2] = slowfast(ss,cut)**	*Proyección estable e inestable y modos de descomposición rápido y lento*
[a,b,c,d] = tfm2ss(num,den,r,c) **[ss] = tfm2ss(tf,r,c)**	*Convierte una matriz de función de transferencia (MIMO) en forma de controlador de bloque en el espacio de los estados*

Comandos de utilidades

[p1,p2,lamp,perr,wellposed,p] = aresolv(a,q,r) **[p1,p2,lamp,perr,wellposed,p] = aresolv(a,q,r,Type)**	*Resuelve la ecuación continua generalizada de Riccati $A^TP+PA-PRP+Q = 0$ con $P = p1/p2$*
[p1,p2,lamp,perr,wellposed,p] = daresolv(a,b,q,r) **[p1,p2,lamp,perr,wellposed,p] = daresolv(a,b,q,r,Type)**	*Resuelve la ecuación discreta generalizada de Riccati $A^TPA - A^TPB(R+B^TPB)^{-1}P+B^TPA +Q=0$ con $P = p2/p1$*
[tot] = riccond(a,b,qrn,p1,p2) **[tot] = driccond(a,b,q,r,p1,p2)**	*Número de condición de la ecuación algebraica continua de Riccati* *Número de condición de la ecuación algebraica discreta de Riccati*

[v,t,m] = blkrsch(a,Type,cut) **[v,t,m,swap] = cschur(a,Type)**	*Bloque ordenado en la forma real de Schur vía cschur*
[v,t,m] = blkrsch(a,Type,cut) **[v,t,m,swap] = cschur(a,Type)**	*Bloque ordenado en la forma compleja de Schur vía rotación deGivens*

Comandos sobre gráficos Bode multivariables

[cg,ph,w] = cgloci(a,b,c,d(,Ts)) **[cg,ph,w] = cgloci(a,b,c,d(,Ts),'inv')** **[cg,ph,w] = cgloci(a,b,c,d(,Ts),w)** **[cg,ph,w] = cgloci(a,b,c,d(,Ts),w,'inv')** **[cg,ph,w] = cgloci(ss,)**	*Ganancia característica continua loci*
[cg,ph,w] = dcgloci(a,b,c,d(,Ts)) **[cg,ph,w] = dcgloci(a,b,c,d(,Ts),'inv')** **[cg,ph,w] = dcgloci(a,b,c,d(,Ts),w)** **[cg,ph,w] = dcgloci(a,b,c,d(,Ts),w,'inv')** **[cg,ph,w] = dcgloci(ss,)**	*Ganancia característica discreta loci*
[sv,w] = dsigma(a,b,c,d(,Ts)) **[sv,w] = dsigma(a,b,c,d(,Ts),'inv')** **[sv,w] = dsigma(a,b,c,d(,Ts),w)** **[sv,w] = dsigma(a,b,c,d(,Ts),w,'inv')** **[sv,w] = dsigma(ss,..)**	*Gráfico Bode de valores singulares discretos*
[sv,w] = sigma(a,b,c,d(,Ts)) **[sv,w] = sigma(a,b,c,d(,Ts),'inv')** **[sv,w] = sigma(a,b,c,d(,Ts),w)** **[sv,w] = sigma(a,b,c,d(,Ts),w,'inv')** **[sv,w] = sigma(ss,..)**	*Gráfico Bode de valores singulares continuos*
[mu,ascaled,logm,x] = muopt(a) **[mu,ascaled,logm,x] = muopt(a,k)**	*Frontera superior en los valores singulares estructurados usando aproximación de multiplicador*
[mu,ascaled,logd] = osborne(a) **[mu,ascaled,logd] = osborne(a,k)**	*Frontera superior en los valores singulares estructurados usando el método de Osborne*
[mu] = perron(a) **[mu] = perron(a,k)** **[mu,ascaled,logd] = psv(a)** **[mu,ascaled,logd] = psv(a,k)**	*Frontera superior en los valores singulares estructurados usando los métodos de Perron y valores singulares*
[mu,logd] = ssv(a,b,c,d,w) **[mu,logd] = ssv(a,b,c,d,w,k)** **[mu,logd] = ssv(a,b,c,d,w,k,opt)** **[mu,logd] = ssv(ss,)**	*Halla el gráfico Bode de los valores singulares estructurados*

Ejercicio 13-1. Dado el modelo y(s) definido en el problema con entrada doble y salida simple, cuyos datos de entrada y salida se encuentran en el fichero mlrdat, determinar la desviación típica para los datos de entrada utilizando la función autoesc y escalar la entrada por su desviación típica solamente. Situar los datos de entrada y salida en la forma que permite calcular los coeficientes de respuesta impulso (35 coeficientes) y hallar dichos coeficientes mediante mlr. Por último escalar theta basada en la desviación típica del escalado de entrada y convertir el modelo respuesta a impulso a un modelo de paso para ser usado en diseño MPC graficando los coeficientes de paso de respuesta.

$$y(s) = \begin{bmatrix} \dfrac{5.72e^{-14s}}{60s+1} & \dfrac{1.52e^{-15s}}{25s+1} \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix}$$

Para obtener la Figura 13-1 se utilizará la siguiente sintaxis de MATLAB:

```
>> load mlrdat;
>> [ax,mx,stdx] = autosc(x);
>> mx = [0,0];
sx = scal(x,mx,stdx);
>> n = 35;
[xreg,yreg] = wrtreg(sx,y,n);
>> ninput = 2;
plotopt = 2;
[theta,yres] = mlr(xreg,yreg,ninput,plotopt);
```

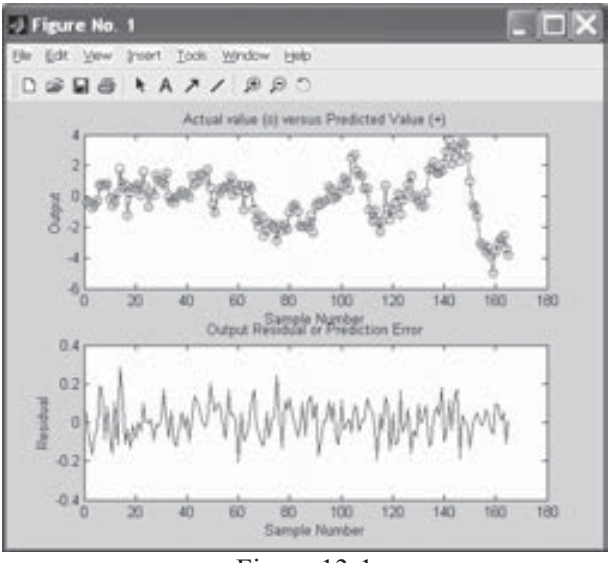


Figura 13-1

Para el escalado de theta, la conversión del modelo y el graficado de los coeficientes de paso de respuesta (Figura 13-2) usando tiempo de muestreo de 7 minutos para hallar el impulso, se utilizará la siguiente sintaxis:

```
>> theta = scal(theta,mx,stdx);
>> nout = 1;
delt = 7;
model = imp2step(delt,nout,theta);
>> plotstep(model)
```

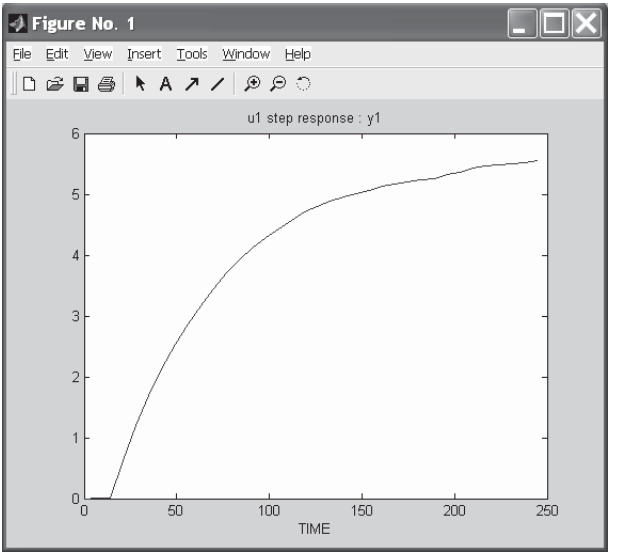


Figura 13-2

Ejercicio 13-2. Dado el modelo de función de transferencia en tiempo continuo G(s) definido en el problema sin desfase, crear el modelo MPC correspondiente en formato tf. Realizar la misma tarea suponiendo un desfase de 2.5 y hallar la función de transferencia equivalente en forma discreta

El modelo $G(s)$ sin desfase viene definido como:

$$\frac{3s-1}{5s^2+2s+1}$$

que se transforma en formato *tf* como sigue:

```
>> g=poly2tfd(0.5*[3 -1],[5 2 1])

g =
```

```
         0    1.5000   -0.5000
    5.0000    2.0000    1.0000
         0         0         0
```

Si existe un desfase de 2,5 el modelo se representa como:

$$\frac{3s-1}{5s^2+2s+1}e^{-2.5s}$$

y la transformación a formato *tf* se realiza como sigue:

```
>> g=poly2tfd(0.5*[3 -1],[5 2 1],0,2.5)

g =

         0    1.5000   -0.5000
    5.0000    2.0000    1.0000
         0    2.5000         0
```

Para transformar la función de transferencia a forma discreta usando un periodo de muestreo de 0,75 unidades se utiliza la siguiente sintaxis:

```
>> delt=0.75;
[numd,dend]=cp2dp(0.5*[3 -1],[5 2 1],delt,rem(2.5,delt))

numd =

         0    0.1232   -0.1106   -0.0607


dend =

    1.0000   -1.6445    0.7408         0
```

Ejercicio 13-3. Dado el modelo especificado en el problema construir modelos separados de respuesta para las variables u y w con un tiempo de muestreo T=3 y combinarlos para formar el modelo del sistema completo.

$$\begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \begin{bmatrix} \frac{12.8e^{-s}}{16.7s+1} & \frac{-18.9e^{-3s}}{21.0s+1} \\ \frac{6.6e^{-7s}}{10.9s+1} & \frac{-19.4e^{-3s}}{14.4s+1} \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix} + \begin{bmatrix} \frac{3.8e^{-8s}}{14.9s+1} \\ \frac{4.9e^{-3s}}{13.2s+1} \end{bmatrix} w(s)$$

```
>> g11=poly2tfd(12.8,[16.7 1],0,1);
g21=poly2tfd(6.6,[10.9 1],0,7);
g12=poly2tfd(-18.9,[21.0 1],0,3);
g22=poly2tfd(-19.4,[14.4 1],0,3);
delt=3; ny=2;
umod=tfd2mod(delt,ny,g11,g21,g12,g22);
gw1=poly2tfd(3.8,[14.9 1],0,8);
gw2=poly2tfd(4.9,[13.2 1],0,3);
wmod=tfd2mod(delt,ny,gw1,gw2);
pmod=addumd(umod,wmod)

pmod =

Columns 1 through 14

   3.0000   13.0000    2.0000         0    1.0000    2.0000         0         0         0         0         0         0         0         0
      NaN    1.5950   -0.6345         0         0         0         0         0         0         0         0         0         0         0
        0    1.0000         0         0         0         0         0         0         0         0         0         0         0         0
        0         0    1.0000         0         0         0         0         0         0         0         0         0         0         0
        0         0         0    1.0000         0         0         0         0         0         0         0         0         0         0
        0         0         0         0    1.0000         0         0         0         0         0         0         0         0         0
        0         0         0         0         0         0    1.6788   -0.7038         0         0         0         0         0         0
        0         0         0         0         0         0    1.0000         0         0         0         0         0         0         0
        0         0         0         0         0         0         0    1.0000         0         0         0         0         0         0
        0         0         0         0         0         0         0         0         0    1.6143   -0.6514         0         0         0
        0         0         0         0         0         0         0         0         0    1.0000         0         0         0         0
        0         0         0         0         0         0         0         0         0         0    1.0000         0         0         0
        0         0         0         0         0         0         0         0         0         0         0    1.0000         0         0
        0         0         0         0         0         0         0         0         0         0         0         0    1.0000         0
        0    1.4447   -0.4371   -0.5012         0         0         0   -2.5160    2.0428         0         0    0.2467    0.2498   -0.3556
        0         0         0    1.1064   -0.4429   -0.4024         0   -3.6484    3.1627         0    0.9962   -0.8145         0         0

Columns 15 through 17

        0         0         0
   1.0000         0         0
        0         0         0
        0         0         0
        0         0         0
        0         0         0
        0    1.0000         0
        0         0         0
        0         0         0
        0         0    1.0000
        0         0         0
        0         0         0
        0         0         0
        0         0         0
        0         0         0
        0         0         0
```

Ejercicio 13-4. Considerando el sistema lineal especificado en el problema construir modelos individuales en formato tf que calculen y grafiquen el modelo MIMO paso de respuesta.

$$\begin{bmatrix} y_1(s) \\ y_2(s) \end{bmatrix} = \begin{bmatrix} \dfrac{12.8e^{-s}}{16.7s+1} & \dfrac{-18.9e^{-3s}}{21.0s+1} \\ \dfrac{6.6e^{-7s}}{10.9s+1} & \dfrac{-19.4e^{-3s}}{14.4s+1} \end{bmatrix} \begin{bmatrix} u_1(s) \\ u_2(s) \end{bmatrix}$$

Para obtener la gráfica de la Figura 13-3 se utilizará la siguiente sintaxis:

```
>> g11=poly2tfd(12.8,[16.7 1],0,1);
g21=poly2tfd(6.6,[10.9 1],0,7);
g12=poly2tfd(-18.9,[21.0 1],0,3);
g22=poly2tfd(-19.4,[14.4 1],0,3);
delt=3; ny=2; tfinal=90;
plant=tfd2step(tfinal,delt,ny,g11,g21,g12,g22,gw1,gw2);
plotstep(plant)
 Percent error in the last step response coefficient
 of output yi for input uj is :
   0.48%       1.6%      0.41%
  0.049%      0.24%      0.14%
```

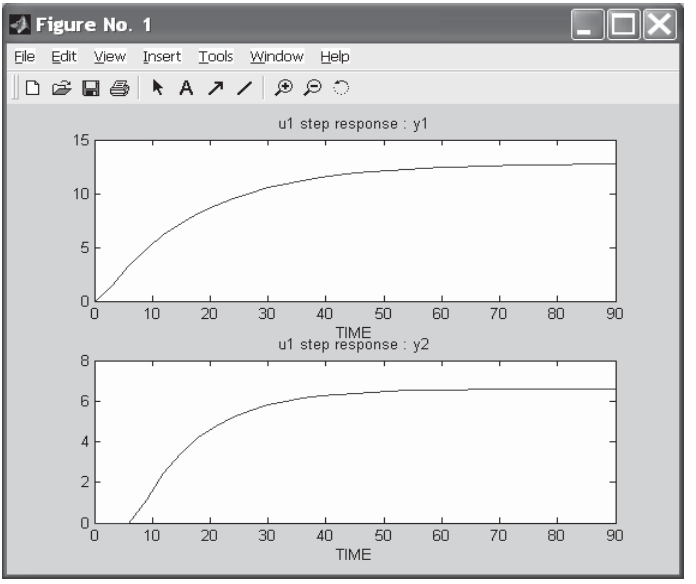


Figura 13-3

Ejercicio 13-5. Considerando el sistema lineal especificado en el problema anterior medir el efecto de situar un límite de 0,1 en la tasa de cambio de u_1 y un mínimo de -0,15 para u_2. Posteriormente aplicar un cota inferior de cero a ambas salidas.

Se construye el modelo mediante la siguiente sintaxis:

```
>> g11=poly2tfd(12.8,[16.7 1],0,1);
g21=poly2tfd(6.6,[10.9 1],0,7);
g12=poly2tfd(-18.9,[21.0 1],0,3);
g22=poly2tfd(-19.4,[14.4 1],0,3);
delt=3; ny=2; tfinal=90;
model=tfd2step(tfinal,delt,ny,g11,g21,g12,g22);
plant=model;
P=6;  M=2;  ywt=[ ]; uwt=[1 1];
tend=30;  r=[0 1];
```

```
 Percent error in the last step response coefficient
 of output yi for input uj is :
   0.48%       1.6%
  0.049%      0.24%
```

El efecto de las restricciones (Figura 13-4) se observa mediante la siguiente sintaxis:

```
>> ulim=[-inf -0.15 inf inf 0.1 100];
ylim=[ ];
[y,u]=cmpc(plant,model,ywt,uwt,M,P,tend,r,ulim,ylim);
plotall(y,u,delt),pause

 Time remaining 30/30
 Time remaining 0/30
Simulation time is 0.03 seconds.
```

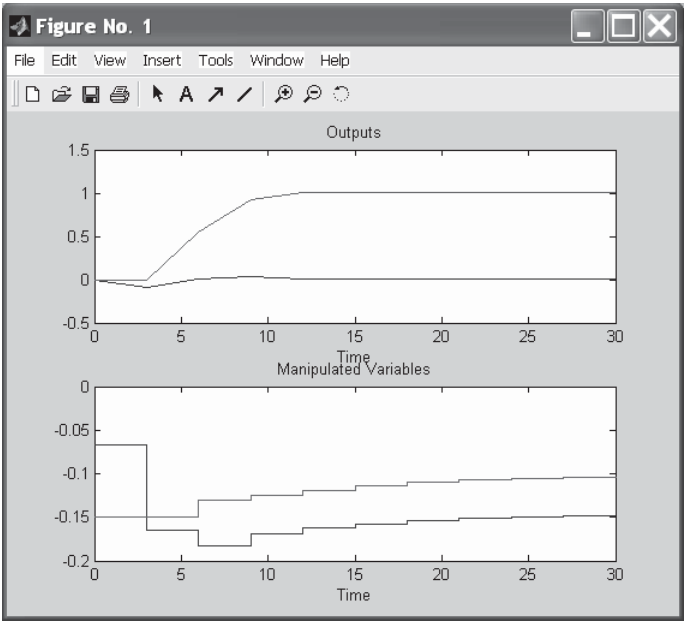


Figura 13-4

Se aplica límite inferior cero a ambos outputs (Figura 13-5) mediante la siguiente sintaxis:

```
>> ulim=[-inf -0.15 inf inf 0.1 100];
ylim=[0 0 inf inf];
[y,u]=cmpc(plant,model,ywt,uwt,M,P,tend,r,ulim,ylim);
plotall(y,u,delt),pause
```

```
 Time remaining 30/30
 Time remaining 0/30
Simulation time is 0.03 seconds.
```

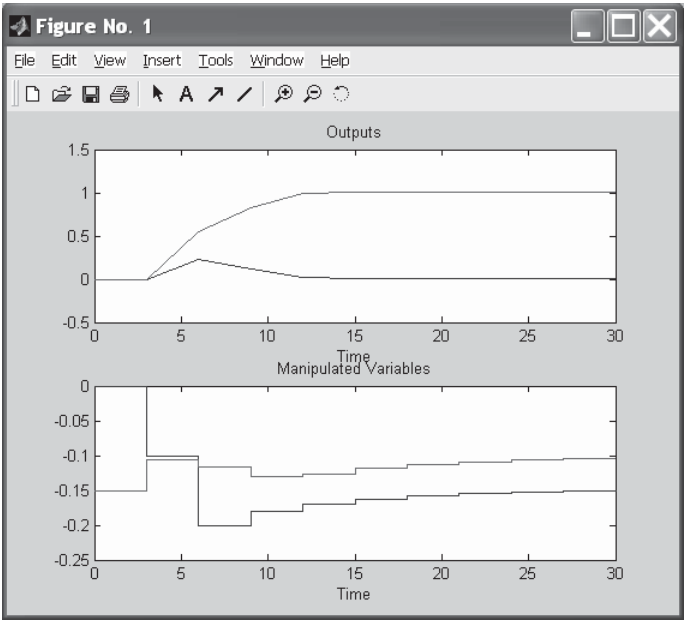


Figura 13-5

Ejercicio 13-6. Considerando el sistema lineal especificado en el problema anterior diseñar el controlador para otros parámetros de ajuste por defecto, calcular el modelo de bucle cerrado del sistema y chequear polos para estabilidad. Posteriormente realizar un gráfico de frecuencia de respuesta de las funciones de sensibilidad y su complementario y calcular y graficar los valores singulares para la sensibilidad.

```
>> g11=poly2tfd(12.8,[16.7 1],0,1);
g21=poly2tfd(6.6,[10.9 1],0,7);
g12=poly2tfd(-18.9,[21.0 1],0,3);
g22=poly2tfd(-19.4,[14.4 1],0,3);
delt=3; ny=2;
imod=tfd2mod(delt,ny,g11,g21,g12,g22);
pmod=imod;

>> P=6;.
M=2;
ywt=[ ];
uwt=[ ];
Ks=smpccon(imod,ywt,uwt,M,P);
```

```
>> clmod=smpccl(pmod,imod,Ks);
maxpole=max(abs(smpcpole(clmod)))
```

maxpole =

0.8869

El gráfico de la frecuencia de respuesta de la sensibilidad (Figura 13-6) y su complementario (Figura 13-7) se realizan como sigue:

```
>> freq = [-3,0,30];
in = [1:ny]; % input is r for comp. sensitivity
out = [1:ny]; % output is yp for comp. sensitivity
[frsp,eyefrsp] = mod2frsp(clmod,freq,out,in);
plotfrsp(eyefrsp); % Sensitivity
pause;
```

```
over estimated time to perform the frequency response: 0.61 sec
```

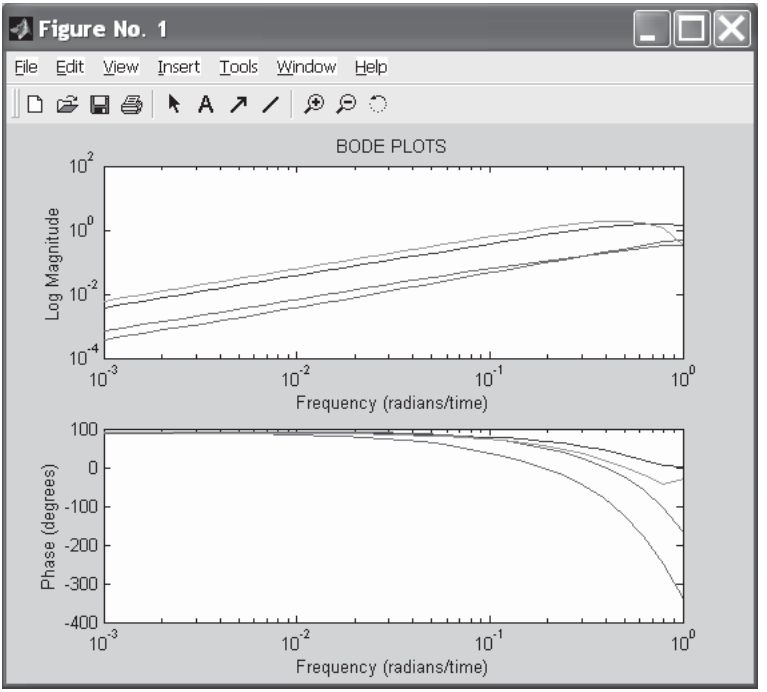


Figura 13-6

La sintaxis para el complementario es la siguiente:

```
>> plotfrsp(frsp); % Complementary Sensitivity pause;
```

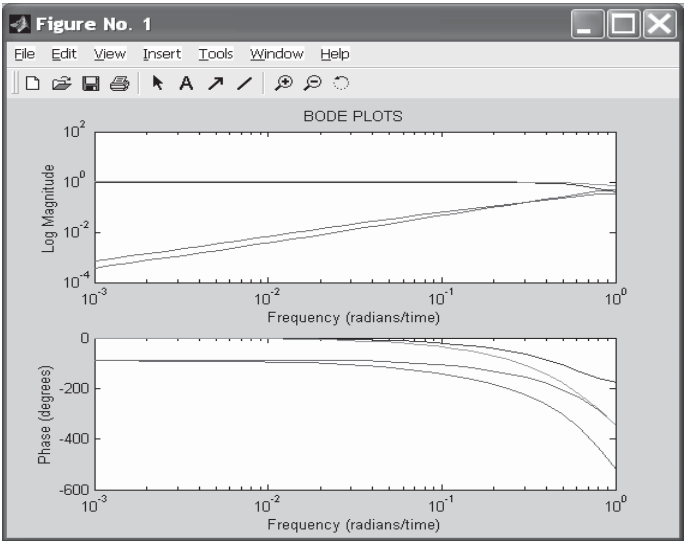


Figura 13-7

Para calcular y graficar los valores singulares para la sensibilidad (Figura 13-8) se utiliza la siguiente sintaxis:

```
>> [sigma,omega] = svdfrsp(eyefrsp);
clg;
semilogx(omega,sigma);
title('Singular Values vs. Frequency');
xlabel('Frequency (radians/time)');
ylabel('Singular Values');
```

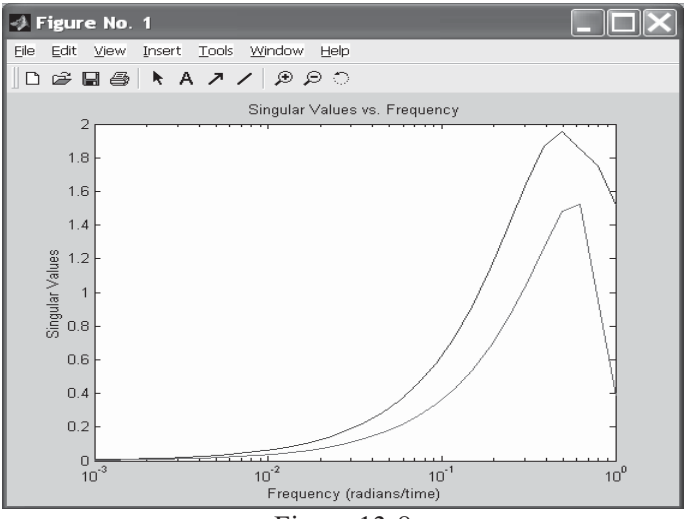


Figura 13-8

14

Técnicas de optimización

14.1 Optimization Toolbox

Optimization Toolbox proporciona algoritmos para resolución de problemas de optimización, tales como programación lineal, programación cuadrática, mínimos cuadrados no lineales y resolución de ecuaciones no lineales. Contiene rutinas que ponen en práctica los métodos más ampliamente utilizados para realizar minimización y maximización.

El toolbox incluye algoritmos para resolver muchos tipos de problemas de optimización, incluyendo algoritmos estándar para optimización al nivel actual de la técnica, minimización no lineal sin restricciones, minimización no lineal restringida, incluyendo minimax, logro de objetivos y problemas de minimización semiinfinitos, programación cuadrática y lineal, mínimos cuadrados no lineales y ajuste de curvas con límites, sistema no lineal de resolución de ecuaciones y mínimos cuadrados lineales restringidos.

Asimismo, este toolbox contiene también algoritmos a gran escala especializados para resolver problemas dispersos, ajuste de datos usando ajuste de curvas, mínimos cuadrados no lineales, búsqueda no lineal de cero y resolución de sistemas no lineales de ecuaciones. El entorno funciona con entradas escalares, vectoriales o matriciales. Las funciones que hay que optimizar pueden escribirse como una función guardada o interactivamente en la línea de comandos de MATLAB.

Algoritmos estándar

El toolbox pone en práctica el estado actual de la técnica en algoritmos de optimización. Los principales algoritmos para minimización no limitada son el método BFGS quasi-Newton y el método de investigación directa Nelder-Mead con investigación lineal. Para minimización con límites, logro de objetivos y optimizaciones semiinfinitas se usan variaciones de programación cuadrática secuencial (SQP). Los problemas de mínimos cuadrados no lineales se resuelven usando los métodos de Gauss-Newton o de Levenberg-Marquardt. Las rutinas para resolver problemas de programación cuadrática y lineal usan un método de series activas combinado con técnicas de proyección. Las rutinas ofrecen una gama de algoritmos y estrategias de investigación lineal. Las estrategias de investigación lineal son métodos de interpolación y extrapolación cuadrática y cúbica protegidos.

Algoritmos a gran escala

Optimization Toolbox incluye también algoritmos para problemas con dispersión o estructura. Los métodos a gran escala aprovechan las prestaciones de matrices dispersas de MATLAB. El toolbox incluye algoritmos para resolver problemas de programación lineal, mínimos cuadrados no lineales con límites, minimización no lineal sin restricciones, minimización no lineal con restricciones de límites, minimización no lineal con igualdades lineales, sistema no lineal de resolución de ecuaciones, minimización cuadrática con restricciones de límites, minimización cuadrática con igualdades lineales y mínimos cuadrados lineales con restricciones de límites. Se implementa también el nuevo algoritmo de programación lineal a gran escala; se basa en el método 1 LIPSOL (*Linear programming Interior-Point SOLver*) de Yin Zhang, un algoritmo de punto interior primal-dual basado en el método de predicción-corrección de Mahrota. También hay disponibles métodos a gran escala para algunas formulaciones de programación cuadrática y objetivos no lineales con restricciones de límites o restricciones de igualdad lineal. Estos métodos son algoritmos de región de confianza a gran escala, desarrollados por Thomas F. Coleman y usan métodos de Newton reflexivos y de proyección para manejar las restricciones.

14.2 Algoritmos de minimización

La mayoría de las opciones de trabajo en minimización sin y con restricción en el campo multivariable están presentes como funciones específicas en este módulo de MATLAB.

En el cuadro siguiente se resumen las funciones sobre minimización que proporciona Optimization Toolbox.

fgoalattain	*Problemas multiobjetivo*
fminbnd	*Minimización no lineal escalar con fronteras*
fmincon	*Minimización no lineal con restricciones*
fminimax	*Optimización minimax*
fminsearch **fminunc**	*Minimización no lineal sin restricciones*
fseminf	*Minimización semiinfinita*
linprog	*Programación cuadrática*
quadprog	*Programación lineal*

Problemas multiobjetivo

Un problema general de este tipo puede definirse como sigue:

$$\underset{x,\gamma}{\text{minimize}} \ \gamma$$

sujeto a las restricciones siguientes:

$$\begin{aligned} &F(x) - weight \cdot \gamma \le goal \\ &c(x) \le 0 \\ &ceq(x) = 0 \\ &A \cdot x \le b \\ &Aeq \cdot x = beq \\ &lb \le x \le ub \end{aligned}$$

donde *x*, *weight*, *goal*, *b*, *beq*, *lb*, y *ub* son vectores, *A* y *Aeq* son matrices, y *c(x)*, *ceq(x)*, y *F(x)* son funciones que devuelven vectores. *F(x)*, *c(x)*, y *ceq(x)* pueden ser funciones no lineales.

La función *fgoalattain* resuelve este tipo de problemas con la sintaxis siguiente:

```
x = fgoalattain(fun,x0,goal,weight)
x = fgoalattain(fun,x0,goal,weight,A,b)
x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq)
x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub)
x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,lb,ub,nonlcon)
x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,...
                lb,ub,nonlcon,options)
x = fgoalattain(fun,x0,goal,weight,A,b,Aeq,beq,...
                lb,ub,nonlcon,options,P1,P2,...)
[x,fval] = fgoalattain(...)
[x,fval,attainfactor] = fgoalattain(...)
[x,fval,attainfactor,exitflag] = fgoalattain(...)
[x,fval,attainfactor,exitflag,output] = fgoalattain(...)
[x,fval,attainfactor,exitflag,output,lambda] = fgoalattain(...)
```

Las distintas formas de la sintaxis de la función son casos particulares del problema global. Comienzan considerando el problema en su mínima expresión y la van ampliando hasta su generalización más amplia. La solución del problema es *x*, y *fval* es el valor de la función objetivo en *x*. La realización del factor en *x* es *attaintfactor*, *exitflag* es un indicador de salida y *output* presenta información sobre el proceso de optimización y *lambda* contiene información sobre los multiplicadores de Lagrange.

Como ejemplo consideremos un controlador *K* que produce un sistema en bucle cerrado:

$$\begin{aligned} x &= (A + BKC)x + Bu \\ y &= Cx \end{aligned}$$

Los autovalores del sistema se determinan mediante las matrices *A, B, C* y *K* utilizando *eig(A+B*K*C)*. Los autovalores deben estar en el eje real o en el plano complejo a la izquierda de [-5, -3, -1]. Además, para no saturar las entradas los elementos de *K* han de estar entre -4 y 4. Se trata de un sistema inestable con dos entradas y dos salidas con bucle abierto y las siguientes matrices de espacio de los estados:

$$A = \begin{bmatrix} -0.5 & 0 & 0 \\ 0 & -2 & 10 \\ 0 & 1 & -2 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 \\ -2 & 2 \\ 0 & 1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

El conjunto de valores objetivo para los autovalores del ciclo cerrado se inicializa como [-5,-3,-1].

Comenzamos con el controlador *K*=[-1,-1;-1,-1] y definimos el M-fichero *eigfun.m* como se indica en la Figura 14-1.

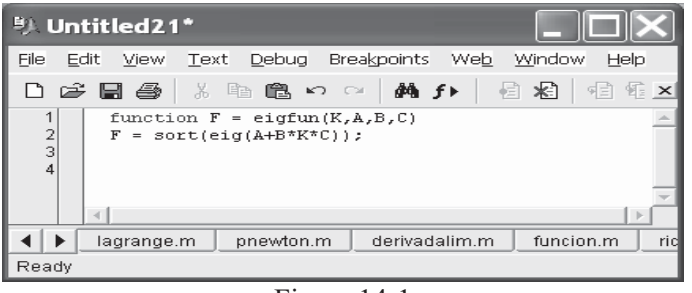


Figura 14-1

A continuación se introducen las matrices del sistema y se invoca la rutina de optimización.

```
>> A = [-0.5 0 0; 0 -2 10; 0 1 -2];
B = [1 0; -2 2; 0 1];
C = [1 0 0; 0 0 1];
K0 = [-1 -1; -1 -1];     % Inicializa la matriz del controlador
goal = [-5 -3 -1];       % Sitúa valores objetivo de autovalores
weight = abs(goal)       % Sitúa pesos
lb = -4*ones(size(K0)); %Sitúa cotas inferiores en el controlador
ub = 4*ones(size(K0));  %Sitúa cotas superiores en el controlador
options = optimset('Display','iter');
[K,fval,attainfactor] = fgoalattain(@eigfun,K0,...
    goal,weight,[],[],[],[],lb,ub,[],options,A,B,C)

weight =

     5     3     1

                    Attainment               Directional
 Iter   F-count       factor      Step-size     derivative    Procedure
    1       6          1.885           1           1.03
    2      13          1.061           1         -0.679
    3      20         0.4211           1         -0.523    Hessian modified
    4      27       -0.06352           1         -0.053    Hessian modified twice
    5      34        -0.1571           1         -0.133
    6      41        -0.3489           1        -0.00768   Hessian modified
    7      48        -0.3643           1       -4.25e-005   Hessian modified
    8      55        -0.3645           1        -0.00303    Hessian modified twice
    9      62        -0.3674           1         -0.0213    Hessian modified
   10      69        -0.3806           1         0.00266
   11      76        -0.3862           1       -2.73e-005   Hessian modified twice
   12      83        -0.3863           1       -1.25e-013   Hessian modified twice

Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon

Active Constraints:

     1
     2
     4
     9
    10

K =

   -4.0000   -0.2564
   -4.0000   -4.0000

fval =

   -6.9313
   -4.1588
   -1.4099

attainfactor =

   -0.3863
```

Minimización no lineal escalar con fronteras

Un problema general de este tipo puede definirse como sigue:

$$\min_x f(x)$$

sujeto a la restricción:

$$x_1 < x < x_2$$

donde x, x_1 y x_2 son escalares y $f(x)$ es una función que devuelve un escalar.

Este problema lo resuelve la función *fminbd,* cuya sintaxis es la siguiente:

```
x = fminbnd(fun,x1,x2)
x = fminbnd(fun,x1,x2,options)
x = fminbnd(fun,x1,x2,options,P1,P2,...)
[x,fval] = fminbnd(...)
[x,fval,exitflag] = fminbnd(...)
[x,fval,exitflag,output] = fminbnd(...)
```

Como ejemplo minimizamos la función *Sen*(x) en [0,2π].

```
>> x = fminbnd(@sin,0,2*pi)

x =

    4.7124
```

Minimización no lineal con restricciones

Un problema general de este tipo puede definirse como sigue:

$$\min_x f(x)$$

sujeto a las restricciones:

$$\begin{aligned} &c(x) \le 0 \\ &ceq(x) = 0 \\ &A \cdot x \le b \\ &Aeq \cdot x = beq \\ &lb \le x \le ub \end{aligned}$$

donde x, b, *beq*, *lb* y *ub* son vectores, A y *Aeq* son matrices y *c(x)*, *ceq(x)* y *F(x)* son funciones que devuelven vectores. *F(x)*, *c(x)* y *ceq(x)* pueden ser funciones no lineales.

Este problema lo resuelve la función *mincon*, cuya sintaxis es la siguiente:

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x =
fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,
...)
[x,fval] = fmincon(...)
[x,fval,exitflag] = fmincon(...)
[x,fval,exitflag,output] = fmincon(...)
[x,fval,exitflag,output,lambda] = fmincon(...)
[x,fval,exitflag,output,lambda,grad] = fmincon(...)
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(...)
```

Como ejemplo minimizamos la función $f(x) = -x_1*x_2*x_3$ sujeta a la restricción $0 \leq -x_1+2x_2+2x_3 \leq 72$ comenzando en el punto $x_0 = [10;10;10]$

Reescribiendo la restricción como:

$$-x_1 - 2x_2 - 2x_3 \leq 0$$
$$x_1 + 2x_2 + 2x_3 \leq 72$$

podemos utilizar las matrices

$$A = \begin{bmatrix} -1 & -2 & -2 \\ 1 & 2 & 2 \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ 72 \end{bmatrix}$$

para plantear la restricción de la forma $A*x \leq b$

Definimos la función objetivo mediante el M-fichero de la Figura 14-2.

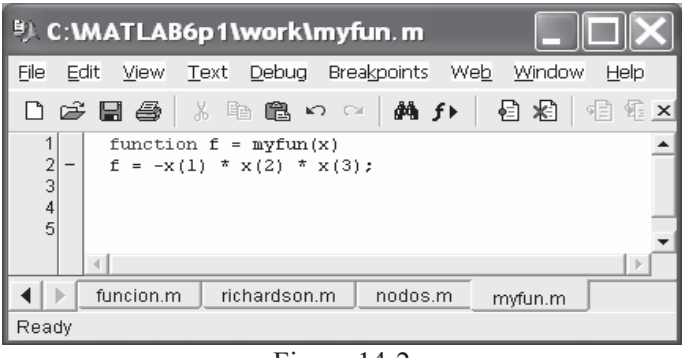


Figura 14-2

Ahora ya se puede resolver el problema mediante la sintaxis:

```
>> A=[-1 -2 -2; 1 2 2];
>> b=[0 72]';
>> x0 = [10; 10; 10];
>> [x,fval] = fmincon(@myfun,x0,A,b)

Optimization terminated successfully:
 Magnitude of directional derivative in search direction
  less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon
Active Constraints:
     2

x =

   24.0000
   12.0000
   12.0000

fval =

       -3456
```

Optimización mínima: *fminimax* y *fminuc*

Un problema general de este tipo puede definirse como sigue:

$$\min_{x} \max_{\{F_i\}} \{F_i(x)\}$$

sujeto a las restricciones:

$$\begin{aligned} c(x) &\le 0 \\ ceq(x) &= 0 \\ A \cdot x &\le b \\ Aeq \cdot x &= beq \\ lb \le x &\le ub \end{aligned}$$

donde *x*, *b*, *beq*, *lb* y *ub* son vectores, *A* y *Aeq* son matrices y *c(x)*, *ceq(x)* y *F(x)* son funciones que devuelven vectores. *F(x)*, *c(x)* y *ceq(x)* pueden ser funciones no lineales.

Este problema lo resuelve la función *fminimax,* cuya sintaxis es la siguiente:

```
x = fminimax(fun,x0)
x = fminimax(fun,x0,A,b)
x = fminimax(fun,x0,A,b,Aeq,beq)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x =
fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,
...)
[x,fval] = fminimax(...)
[x,fval,maxfval] = fminimax(...)
[x,fval,maxfval,exitflag] = fminimax(...)
[x,fval,maxfval,exitflag,output] = fminimax(...)
[x,fval,maxfval,exitflag,output,lambda] = fminimax(...)
```

Minimiza las funciones definidas partiendo del valor inicial *x*0 con sujeción a *A**x* <= *b* o a *Aeq**x* = *beq* o soluciones *x* en el rango *lb* <= *x* <= *ub*. El valor *maxfval* es el máximo valor de la función.

La función *fminuc* encuentra el mínimo de una función multivariante sin restricciones

$$\min_x f(x)$$

donde *x* es un vector y *f*(*x*) es una función que devuelve un escalar.

La sintaxis es la siguiente:

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
x = fminunc(fun,x0,options,P1,P2,...)
[x,fval] = fminunc(...)
[x,fval,exitflag] = fminunc(...)
[x,fval,exitflag,output] = fminunc(...)
[x,fval,exitflag,output,grad] = fminunc(...)
[x,fval,exitflag,output,grad,hessian] = fminunc(...)
```

Optimización *minimax*

Un problema general de este tipo puede definirse como sigue:

$$\min_x \max_{\{F_i\}} \{F_i(x)\}$$

sujeto a las restricciones:

$$c(x) \leq 0$$
$$ceq(x) = 0$$
$$A \cdot x \leq b$$
$$Aeq \cdot x = beq$$
$$lb \leq x \leq ub$$

donde *x*, *b*, *beq*, *lb* y *ub* son vectores, *A* y *Aeq* son matrices y *c(x)*, *ceq(x)* y *F(x)* son funciones que devuelven vectores. *F(x)*, *c(x)* y *ceq(x)* pueden ser funciones no lineales.

Este problema lo resuelve la función *fminimax,* cuya sintaxis es la siguiente:

```
x = fminimax(fun,x0)
x = fminimax(fun,x0,A,b)
x = fminimax(fun,x0,A,b,Aeq,beq)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x =
fminimax(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,
...)
[x,fval] = fminimax(...)
[x,fval,maxfval] = fminimax(...)
[x,fval,maxfval,exitflag] = fminimax(...)
[x,fval,maxfval,exitflag,output] = fminimax(...)
[x,fval,maxfval,exitflag,output,lambda] = fminimax(...)
```

Minimiza las funciones definidas partiendo del valor inicial *x*0 con sujeción a *A***x* <= *b* o a *Aeq***x* = *beq* o soluciones *x* en el rango *lb* <= *x* <= *ub*. El valor *maxfval* es el máximo valor de la función.

Optimización mínima: *fminsearch* y *fminuc*

La función *fminsearch* encuentra el mínimo de una función multivariante sin restricciones

$$\min_x f(x)$$

donde *x* es un vector y *f*(*x*) es una función que devuelve un escalar.

La sintaxis es la siguiente:

```
x = fminsearch(fun,x0)
x = fminsearch(fun,x0,options)
x = fminsearch(fun,x0,options,P1,P2,...)
[x,fval] = fminsearch(...)
[x,fval,exitflag] = fminsearch(...)
[x,fval,exitflag,output] = fminsearch(...)
```

Como ejemplo se minimiza la función $f(x) = Sen(x) + 3$ como sigue:

```
>> x = fminsearch('sin(x)+3',2)

x =

    4.7124
```

La función *fminuc* encuentra el mínimo de una función multivariante sin restricciones

$$\min_x f(x)$$

donde x es un vector y $f(x)$ es una función que devuelve un escalar.

La sintaxis es la siguiente:

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
x = fminunc(fun,x0,options,P1,P2,...)
[x,fval] = fminunc(...)
[x,fval,exitflag] = fminunc(...)
[x,fval,exitflag,output] = fminunc(...)
[x,fval,exitflag,output,grad] = fminunc(...)
[x,fval,exitflag,output,grad,hessian] = fminunc(...)
```

Minimización semiinfinita

Un problema general de este tipo encuentra mínimos de funciones semiinfinitas muiltivariables con restricciones y puede definirse como sigue:

$$\min_x f(x)$$

sujeto a las restricciones:

$$
\begin{aligned}
&c(x) \le 0, \\
&ceq(x) = 0 \\
&A \cdot x \le b \\
&Aeq \cdot x = beq \\
&lb \le x \le ub \\
&K_1(x, w_1) \le 0 \\
&K_2(x, w_2) \le 0 \\
&\cdots \\
&K_n(x, w_n) \le 0
\end{aligned}
$$

donde *x*, *b*, *beq*, *lb* y *ub* son vectores, *A* y *Aeq* son matrices y *c(x)*, *ceq(x)* y *F(x)* son funciones que devuelven vectores. *F(x)*, *c(x)* y *ceq(x)* pueden ser funciones no lineales. K_i y (x,w_i) son funciones que devuelven vectores y w_i son vectores de al menos longitud 2.

Este problema lo resuelve la función *fseminf* , cuya sintaxis es la siguiente:

```
x = fseminf(fun,x0,ntheta,seminfcon)
x = fseminf(fun,x0,ntheta,seminfcon,A,b)
x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq)
x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,lb,ub)
x =
fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,lb,ub,option
s)
x = fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,...
            lb,ub,options,P1,P2,...)
[x,fval] = fseminf(...)
[x,fval,exitflag] = fseminf(...)
[x,fval,exitflag,output] = fseminf(...)
[x,fval,exitflag,output,lambda] = fseminf(...)
```

Programación lineal

Un problema general de este tipo puede definirse como sigue:

$$\min_x f^T x$$

sujeto a las restricciones:

$$A \cdot x \leq b$$
$$Aeq \cdot x = beq$$
$$lb \leq x \leq ub$$

donde *f*, *x*, *b*, *beq*, *lb* y *ub* son vectores y *A* y *Aeq* son matrices.

Este problema lo resuelve la función *linprog,* cuya sintaxis es la siguiente:

```
x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0)
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
[x,fval] = linprog(...)
[x,fval,exitflag] = linprog(...)
[x,fval,exitflag,output] = linprog(...)
[x,fval,exitflag,output,lambda] = linprog(...)
```

Minimiza *f'*x* con sujeción a *A*x <= b* o a *Aeq*x = beq* o a que *x* esté en el rango *lb <= x <= ub*, pudiendo utilizar un valor inicial *x*0.

Como ejemplo minimizamos la función:

$$f(x) = -5x_1 - 4x_2 - 6x_3$$

sujeta a las restricciones:

$$x_1 - x_2 + x_3 \leq 20$$
$$3x_1 + 2x_2 + 4x_3 \leq 42$$
$$3x_1 + 2x_2 \leq 30$$
$$0 \leq x_1, 0 \leq x_2, 0 \leq x_3$$

Se utiliza la siguiente sintaxis:

```
>> f = [-5; -4; -6]
A =  [1 -1  1
      3  2  4
      3  2  0];
b = [20; 42; 30];
lb = zeros(3,1);
```

```
f =

    -5
    -4
    -6

>> [x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)

Optimization terminated successfully.

x =

    0.0000
   15.0000
    3.0000

fval =

  -78.0000

exitflag =

     1

output =

      iterations: 6
    cgiterations: 0
       algorithm: 'lipsol'

lambda =

    ineqlin: [3x1 double]
      eqlin: [0x1 double]
      upper: [3x1 double]
      lower: [3x1 double]

>> lambda.ineqlin

ans =

    0.0000
    1.5000
    0.5000

>> lambda.lower

ans =

    1.0000
    0.0000
    0.0000
```

Programación cuadrática

Un problema general de este tipo puede definirse como sigue:

$$\min_x \ \frac{1}{2}x^T Hx + f^T x$$

sujeto a las restricciones:

$$\begin{aligned} A \cdot x &\le b \\ Aeq \cdot x &= beq \\ lb \le x &\le ub \end{aligned}$$

donde *f*, *x*, *b*, *beq*, *lb* y *ub* son vectores y *H*, *A* y *Aeq* son matrices.

Este problema lo resuelve la función *quadprog,* cuya sintaxis es la siguiente:

```
x = quadprog(H,f,A,b)
x = quadprog(H,f,A,b,Aeq,beq)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options,p1,p2,...)
[x,fval] = quadprog(...)
[x,fval,exitflag] = quadprog(...)
[x,fval,exitflag,output] = quadprog(...)
[x,fval,exitflag,output,lambda] = quadprog(...)
```

Minimiza *1/2*x'*H*x + f'*x* con sujeción a *A*x <= b* o a *Aeq*x = beq* o a que *x* esté en el rango *lb <= x <= ub*, pudiendo utilizar un valor inicial *x*0.

Como ejemplo minimizamos la función:

$$f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2$$

sujeta a las restricciones:

$$\begin{aligned} x_1 + x_2 &\le 2 \\ -x_1 + 2x_2 &\le 2 \\ 2x_1 + x_2 &\le 3 \\ 0 \le x_1, \quad 0 &\le x_2 \end{aligned}$$

Comenzamos escribiendo la función como:

$$f(x) = \frac{1}{2}x^T Hx + f^T x$$

donde:

$$H = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}, \qquad f = \begin{bmatrix} -2 \\ -6 \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

```
>> H = [1 -1; -1 2] ;
f = [-2; -6];
A = [1 1; -1 2; 2 1];
b = [2; 2; 3];
lb = zeros(2,1);

>> ;[x,fval,exitflag,output,lambda] =
quadprog(H,f,A,b,[],[],lb)

Optimization terminated successfully.

x =

    0.6667
    1.3333

fval =

   -8.2222

exitflag =

     1

output =

       iterations: 3
        algorithm: 'medium-scale: active-set'
    firstorderopt: []
     cgiterations: []

lambda =

      lower: [2x1 double]
      upper: [2x1 double]
      eqlin: [0x1 double]
    ineqlin: [3x1 double]
```

14.3 Algoritmos de resolución de ecuaciones

En el cuadro siguiente se resumen las funciones sobre resolución de ecuaciones y sistemas que proporciona Optimization Toolbox.

fsolve	*Resuelve ecuaciones y sistemas no lineales*
fzero	*Resuelve ecuaciones no lineales escalares*

Resolución de ecuaciones y sistemas

La función *fsolve* resuelve sistemas de ecuaciones no lineales $F(x) = 0$, donde x es un vector y $F(x)$ es una función que devuelve un valor vectorial. Su sintaxis es la siguiente:

```
x = fsolve(fun,x0)
x = fsolve(fun,x0,options)
x = fsolve(fun,x0,options,P1,P2, ... )
[x,fval] = fsolve(...)
[x,fval,exitflag] = fsolve(...)
[x,fval,exitflag,output] = fsolve(...)
[x,fval,exitflag,output,jacobian] = fsolve(...)
```

Como ejemplo resolvemos el sistema:

$$\begin{aligned} 2x_1 - x_2 &= e^{-x_1} \\ -x_1 + 2x_2 &= e^{-x_2} \end{aligned}$$

con condiciones iniciales [-5 5].

Comenzamos escribiendo el sistema en la forma:

$$\begin{aligned} 2x_1 - x_2 - e^{-x_1} &= 0 \\ -x_1 + 2x_2 - e^{-x_2} &= 0 \end{aligned}$$

Las ecuaciones las construimos en el M-fichero de la Figura 14-3.

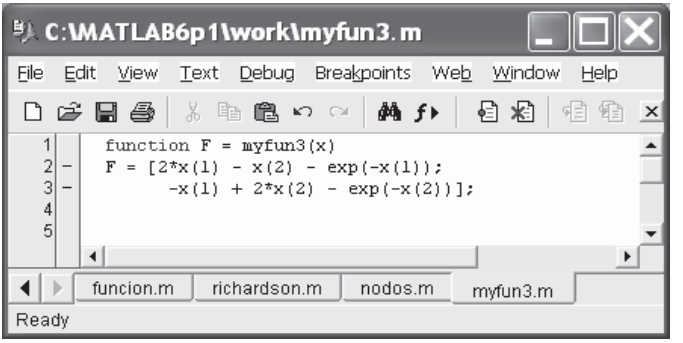


Figura 14-3

El sistema se resuelve mediante la sintaxis:

```
>> x0 = [-5; -5];
>> [x,fval] = fsolve(@myfun3,x0)

Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
    0.5671
    0.5671
fval =
  1.0e-008 *
   -0.5319
   -0.5319
```

La function *fzero* resuelve ecuaciones no lineales mediante la siguiente sintaxis:

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
x = fzero(fun,x0,options,P1,P2,...)
[x,fval] = fzero(...)
[x,fval,exitflag] = fzero(...)
[x,fval,exitflag,output] = fzero(...)
```

Como ejemplo se resuelve la ecuación x^3-2x-5 en un entorno de x=2.

```
>> z = fzero(inline('x^3-2*x-5'),2)
z =
    2.0946
```

14.4 Ajuste de curvas por mínimos cuadrados

MATLAB permite ajustar curvas por mínimos cuadrados con restricciones, por mínimos cuadrados no lineales y por mínimos cuadrados lineales no negativos. Las funciones que implementa para estas tareas son las siguientes:

lsqlin	*Mínimos cudrados lineales con restricciones*
lsqcurvefit	*Ajuste de curvas no lineales*
lsqnonlin	*Mínimos cuadrados no lineales*
lsqnonneg	*Mínimos cuadrados lineales no negativos*

Mínimos cuadrados condicionados

El problema de mínimos cuadrados condicionados tiene la siguiente estructura:

$$\min_x \frac{1}{2}\|Cx - d\|_2^2$$

sujeto a las restricciones:

$$A \cdot x \le b$$
$$Aeq \cdot x = beq$$
$$lb \le x \le ub$$

donde *d*, *x*, *b*, *beq*, *lb* y *ub* son vectores y C, *A* y *Aeq* son matrices.

Este problema lo resuelve la función *lsqlin,* cuya sintaxis es la siguiente:

```
x = lsqlin(C,d,A,b)
x = lsqlin(C,d,A,b,Aeq,beq)
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub)
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0)
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0,options)
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0,options,p1,p2,...)
[x,resnorm] = lsqlin(...)
[x,resnorm,residual] = lsqlin(...)
[x,resnorm,residual,exitflag] = lsqlin(...)
[x,resnorm,residual,exitflag,output] = lsqlin(...)
[x,resnorm,residual,exitflag,output,lambda] =
lsqlin(...)
```

Resuelve *C*x = d* sujeto a *A*x <= b* o a *Aeq*x = beq* o a que *x* esté en el rango *lb <= x <= ub*, pudiendo utilizar un valor inicial *x*0.

Mínimos cuadrados no lineales

La función *lscurvefit* realiza ajustes de curvas no lineales por mínimos cuadrados. Dado un conjunto de datos de entrada *xdata* y un conjunto de datos observados de salida *ydata*, se encuentran los coeficientes *x* que mejor ajustan la función *F(x, xdata):*

$$\min_x \frac{1}{2}\|F(x,xdata)-ydata\|_2^2 = \frac{1}{2}\sum_i (F(x,xdata_i)-ydata_i)^2$$

La sintaxis es la siguiente:

```
x = lsqcurvefit(fun,x0,xdata,ydata)
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub)
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options)
x =
lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options,P1,P2,...)
[x,resnorm] = lsqcurvefit(...)
[x,resnorm,residual] = lsqcurvefit(...)
[x,resnorm,residual,exitflag] = lsqcurvefit(...)
[x,resnorm,residual,exitflag,output] = lsqcurvefit(...)
[x,resnorm,residual,exitflag,output,lambda] =
lsqcurvefit(...)
[x,resnorm,residual,exitflag,output,lambda,jacobian] =
    lsqcurvefit(...)
```

La función *lsqnonlin* resuelve el problema de mínimos cuadrados no lineales siguiente:

$$\min_x f(x) = f_1(x)^2 + f_2(x)^2 + f_3(x)^2 + \ldots + f_m(x)^2 + L$$

mediante la sintaxis:

```
x = lsqnonlin(fun,x0)
x = lsqnonlin(fun,x0,lb,ub)
x = lsqnonlin(fun,x0,lb,ub,options)
x = lsqnonlin(fun,x0,eb,ub,options,P1,P2, ... )
[x,resnorm] = lsqnonlin(...)
[x,resnorm,residual] = lsqnonlin(...)
[x,resnorm,residual,exitflag] = lsqnonlin(...)
[x,resnorm,residual,exitflag,output] = lsqnonlin(...)
[x,resnorm,residual,exitflag,output,lambda] =
lsqnonlin(...)
[x,resnorm,residual,exitflag,output,lambda,jacobian] =
    lsqnonlin(...)
```

Mínimos cuadrados lineales no negativos

La función *lsqnonneg* resuelve el problema de mínimos cuadrados no negativos siguiente:

$$\min_x \frac{1}{2}\|Cx-d\|_2^2 \quad x \geq 0$$

donde la matriz *C* y el vector *d* son los coeficientes de la función objetivo. Su sintaxis es la siguiente:

```
x = lsqnonneg(C,d)
x = lsqnonneg(C,d,x0)
x = lsqnonneg(C,d,x0,options)
[x,resnorm] = lsqnonneg(...)
[x,resnorm,residual] = lsqnonneg(...)
[x,resnorm,residual,exitflag] = lsqnonneg(...)
[x,resnorm,residual,exitflag,output] = lsqnonneg(...)
[x,resnorm,residual,exitflag,output,lambda] =
lsqnonneg(...)
```

En el ejemplo siguiente se compara la solución del problema 4x2 definido por *C* y *d* resuelto mediante el método normal y mediante *lsqnonneg*.

```
>> C = [
     0.0372    0.2869
     0.6861    0.7071
     0.6233    0.6245
     0.6344    0.6170];
d = [
     0.8587
     0.1781
     0.0747
     0.8405];
[C\d, lsqnonneg(C,d)]
ans =
   -2.5627         0
    3.1108    0.6929
>> [norm(C*(C\d)-d), norm(C*lsqnonneg(C,d)-d)]
ans =
    0.6674    0.9118
```

Ejercicio 15-1. Minimizar la función $f(x) = (x-3)^2 - 1$ en el intervalo (0,5).

```
>> x = fminbnd(inline('(x-3)^2-1'),0,5)

x =

      3
```

Ejercicio 15-2. Encontrar el valor de x que minimiza el valor máximo de:

$$[f_1(x), f_2(x), f_3(x), f_4(x), f_5(x)]$$

estando las funciones fi(x) definidas como se indica en el problema.

$$\begin{aligned} f_1(x) &= 2x_1^2 + x_2^2 - 48x_1 - 40x_2 + 304 \\ f_2(x) &= -x_2^2 - 3x_2^2 \\ f_3(x) &= x_1 + 3x_2 - 18 \\ f_4(x) &= -x_1 - x_2 \\ f_5(x) &= x_1 + x_2 - 8. \end{aligned}$$

Comenzamos construyendo el fichero *myfun1* que define las funciones (Figura 14-4).

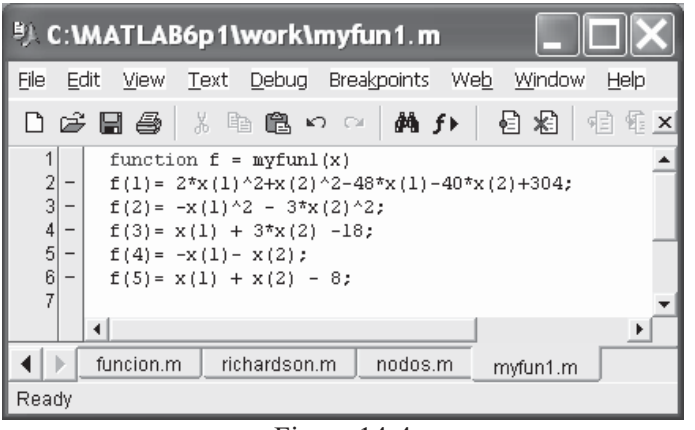


Figura 14-4

Utilizando como valor inicial [0,1 0.1] se resuelve el problema mediante la siguiente sintaxis:

```
>> x0 = [0.1; 0.1];
>> [x,fval] = fminimax(@myfun1,x0)

Optimization terminated successfully:
 Magnitude of directional derivative in search direction
  less than 2*options.TolFun and maximum constraint violation
  is less than options.TolCon
Active Constraints:
     1
     5

x =

    4.0000
    4.0000

fval =

    0.0000  -64.0000   -2.0000   -8.0000   -0.0000
```

Ejercicio 15-3. Minimizar la función siguiente:

$$f(x) = 3*x_1^2 + 2*x1*x2 + x_2^2$$

con valores iniciales [1,1].

```
>> [x,fval] = fminunc(inline('3*x(1)^2 + 2*x(1)*x(2) + x(2)^2'),x0)

Warning: Gradient must be provided for trust-region method;
   using line-search method instead.

> In C:\MATLAB6p1\toolbox\optim\fminunc.m at line 211

Optimization terminated successfully:
 Search direction less than 2*options.TolX

x =

  1.0e-008 *

   -0.7591    0.2665


fval =

  1.3953e-016
```

Ejercicio 15-4. Encontrar los valores de x que minimizan la función f(x) sujeta a las restricciones $k_1(x,w_1)$ y $k_2(x,w_2)$ con w_1 y w_2 en [1, 100]. La función y las restricciones se definen en el problema y elk punto inicial es (0,5 0,2 0,3).

$$f(x) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 + (x_3 - 0.5)^2$$

$$K_1(x, w_1) = \sin(w_1 x_1)\cos(w_1 x_2) - \frac{1}{1000}(w_1 - 50)^2 - \sin(w_1 x_3) - x_3 \le 1$$

$$K_2(x, w_2) = \sin(w_2 x_2)\cos(w_2 x_1) - \frac{1}{1000}(w_2 - 50)^2 - \sin(w_2 x_3) - x_3 \le 1$$

Comenzamos creando un M-fichero con las restricciones (Figura 14-5).

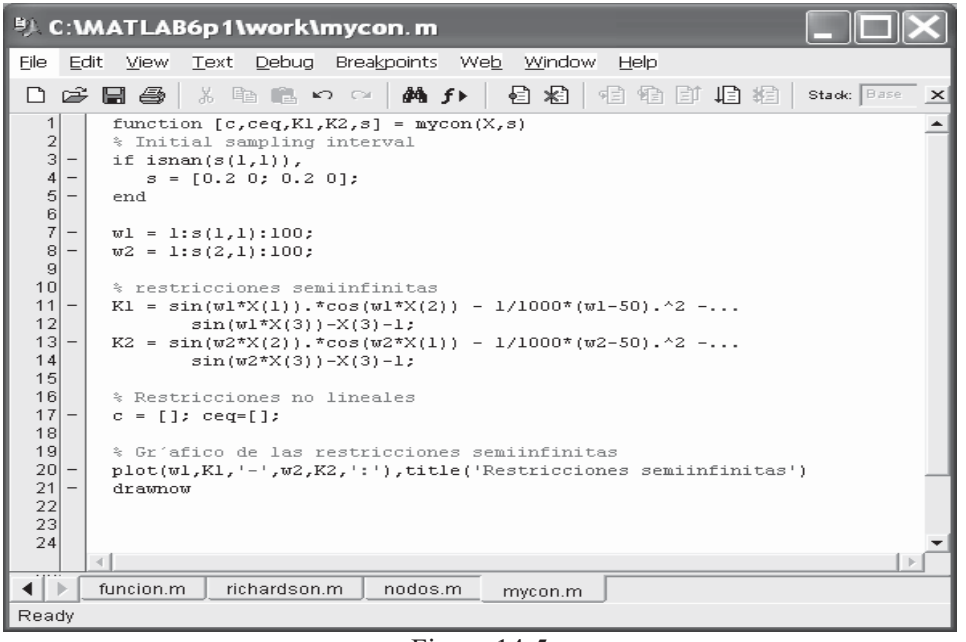


Figura 14-5

También creamos un fichero para la función objetivo (Figura 14-6).

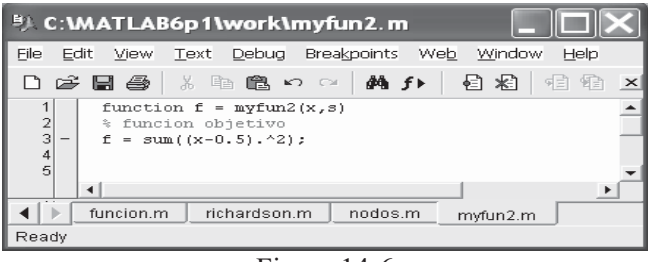


Figura 14-6

Ya se puede resolver el problema y obtener la solución gráfica (Figura 14-7) mediante la sintaxis siguiente:

```
>> [x,fval] = fseminf(@myfun2,x0,2,@mycon)

Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon
Active Constraints:
     7
    10

x =

    0.6673
    0.3013
    0.4023

fval =

    0.0770
```

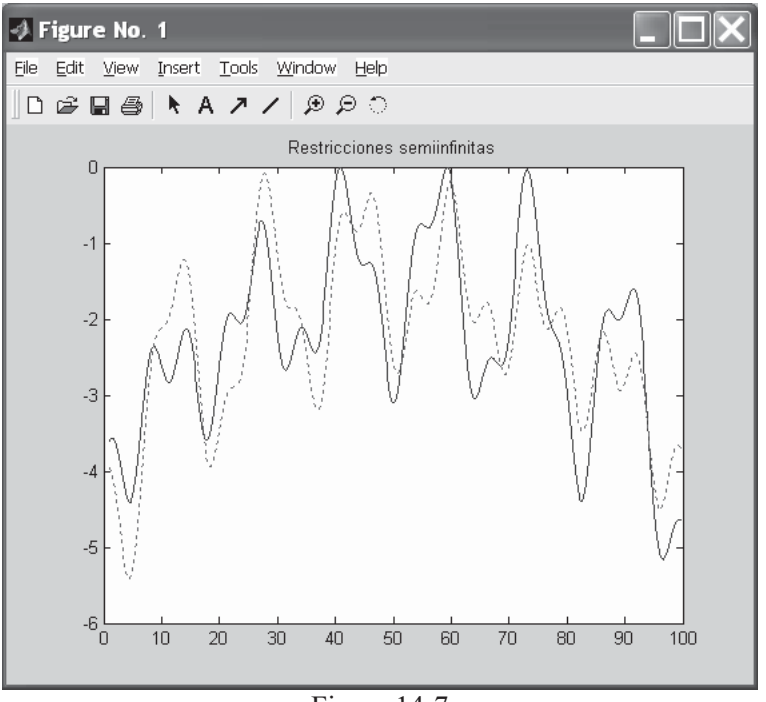


Figura 14-7

Ejercicio 15-5. Dados los conjuntos de datos:

```
xdata = [3.6 7.7 9.3 4.1 8.6 2.8 1.3 7.9 10.0 5.4];
ydata = [16.5 150.6 263.1 24.7 208.5 9.9 2.7 163.9 325.0 54.3];
```

se trata de encontrar los coeficientes x que minimicen la función ydata(i) del tipo definido en el problema.

$$ydata(i) = x(1)\cdot xdata(i)^2 + x(2)\cdot \sin(xdata(i)) + x(3)\cdot xdata(i)^3$$

Nuestro problema puede escribirse como:

$$\min_x \ \frac{1}{2}\sum_{i=1}^{n} (F(x, xdata_i) - ydata_i)^2$$

Comenzamos escribiendo la función *F* en el M-fichero de la Figura 14-8:

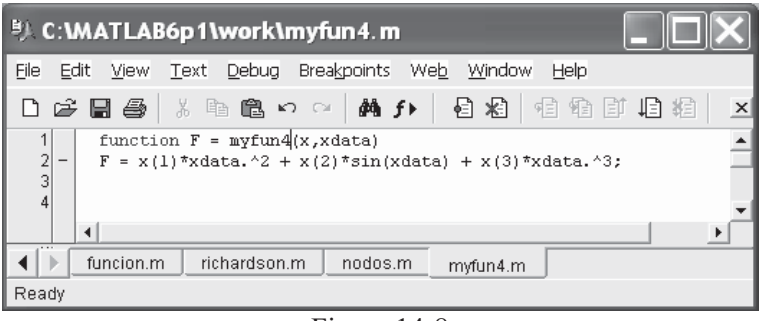


Figura 14-8

El problema, con valores iniciales en [10,10,10], se resuelve mediante la siguiente sintaxis:

```
>> xdata = [3.6 7.7 9.3 4.1 8.6 2.8 1.3 7.9 10.0 5.4];
ydata = [16.5 150.6 263.1 24.7 208.5 9.9 2.7 163.9 325.0
54.3];
>> x0 = [10, 10, 10];
>> [x,resnorm] = lsqcurvefit(@myfun4,x0,xdata,ydata)
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
x =
    0.2269    0.3385    0.3021
resnorm =
    6.2950
```

Matlab®
y sus Aplicaciones en las Ciencias y la Ingeniería
Pérez

MATLAB® es un sistema general de software para matemáticas y otras aplicaciones científicas. Es utilizado por investigadores, ingenieros y analistas, así como en el entorno universitario. Las aplicaciones de **MATLAB** comprenden la mayoría de las áreas de la ciencia, tecnología y negocios en donde se aplican métodos cuantitativos.

Este libro comienza situando al lector en el entorno de trabajo de **MATLAB®**, para posteriormente analizar de forma muy precisa el módulo básico del programa incluyendo las técnicas de programación y sus aplicaciones en el cálculo numérico.

A continuación se presenta de forma completa el módulo de matemática simbólica y sus aplicaciones en el Análisis Matemático, Álgebra Lineal y Geometría. Más adelante se trata de forma extensa el módulo de estadística y sus aplicaciones en la industria incluyendo estadística descriptiva, estadística matemática y técnicas de ajuste de modelos, análisis de la varianza, control de calidad y diseño de experimentos.

Otra parcela importante del libro la constituyen los sistemas de control y sus aplicaciones en la ingeniería (control predictivo, control robusto, etc.). Estas aplicaciones se completan con las técnicas de optimización que proporcionan algoritmos para solucionar problemas de optimización no lineales, tanto generales como a gran escala.

El contenido se presenta en orden secuencial de dificultad con gran variedad de ejemplos prácticos al final de los capítulos. Todo el trabajo se realiza con la última versión de **MATLAB®** para Windows, pero salvo características específicas, el libro puede adaptarse también a versiones anteriores del programa.

CÉSAR PÉREZ LÓPEZ es Licenciado en Ciencias Matemáticas con la especialidad de Estadística, y Licenciado en Ciencias Económicas. Es, asimismo, Estadístico Facultativo del Instituto Nacional de Estadística, Profesor Asociado en el Departamento de Estadística e Investigación Operativa III, de la Escuela Universitaria de Estadística de la Universidad Complutense de Madrid, y Coordinador de Área de Investigación en el Instituto de Estudios Fiscales.

PEARSON
Educación

www.pearsoneducacion.com

ISBN 84-205-3537-0

9 788420 535371