TUTORIAL DE MATLAB

1. ¿QUÉ ES MATLAB?

MatLab es un programa interactivo para computación numérica y visualización de datos. Es ampliamente usado por Ingenieros de Control en el análisis y diseño, posee además una extraordinaria versatilidad y capacidad para resolver problemas en matemática aplicada, física, química, ingeniería, finanzas y muchas otras aplicaciones. Está basado en un sofisticado software de matrices para el análisis de sistemas de ecuaciones. Permite resolver complicados problemas numéricos sin necesidad de escribir un programa.

MATLAB es un entorno de computación y desarrollo de aplicaciones totalmente integrado orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos.

MATLAB integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían tradicionalmente, sin necesidad de hacer uso de la programación tradicional.

El nombre de MATLAB proviene de la contracción de los términos MATrix LABoratory y fue inicialmente concebido para proporcionar fácil acceso a las librerías LINPACK y EISPACK, las cuales representan hoy en dia dos de las librerías más importantes en computación y cálculo matricial.

MATLAB es un sistema de trabajo interactivo cuyo elemento básico de trabajo son las matrices. El programa permite realizar de un modo rápido la resolución numérica de problemas en un tiempo mucho menor que si se quisiesen resolver estos mismos problemas con lenguajes de programación tradicionales como pueden ser los lenguajes Fortran, Basic o C.

MATLAB goza en la actualidad de un alto nivel de implantación en escuelas y centros universitarios, así como en departamentos de investigación y desarrollo de muchas compañías industriales nacionales e internacionales. En entornos universitarios, por ejemplo, MATLAB se ha convertido en una herramienta básica, tanto para los profesionales e investigadores de centros docentes, como una importante herramienta para la impartición de cursos universitarios, tales como sistemas e ingenieria de control, álgebra lineal, proceso digital de imagen, señal, etc. En el mundo industrial, MATLAB está siendo utilizado como herramienta de investigación para la resolución de complejos problemas planteados en la realización y aplicación de modelos matemáticos en ingeniería. Los usos más característicos de la herramienta los encontramos en áreas de computación y cálculo numérico tradicional, prototipaje algorítmico, teoría de control automático, estadística, análisis de series temporales para el proceso digital de señal.

MATLAB dispone también en la actualidad de un amplio abanico de programas de apoyo especializados, denominados Toolboxes, que extienden significativamente el número de funciones incorporadas en el programa principal. Estos Toolboxes cubren en la actualidad prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación, destacando entre ellos el 'toolbox' de proceso de imágenes, señal, control robusto, estadística, análisis financiero, matemáticas simbólicas, redes neurales, lógica difusa, identificación de sistemas, simulación de sistemas dinámicos, etc.

Además también se dispone del programa Simulink que es un entorno gráfico interactivo con el que se puede analizar, modelizar y simular la dinámica de sistemas no lineales.

1.1 Uso de Matrices

MatLab emplea matrices porque con ellas se puede describir infinidad de cosas de una forma altamente flexible y matemáticamente eficiente. Una matriz de pixeles puede ser una imagen o una película. Una matriz de fluctuaciones de una señal puede ser un sonido o una voz humana. Y tal vez más significativamente, una matriz puede describir una relación lineal entre los componentes de un modelo matemático. En este último sentido, una matriz puede describir el comportamiento de un sistema extremadamente complejo. Por ejemplo una matriz puede representar el vuelo de una avión a 40.000 pies de altura, o un filtro digital de procesamiento de señales.

1.2 Origen de MatLab

MatLab fue originalmente desarrollado en lenguaje FORTRAN para ser usado en computadoras mainframe. Fue el resultado de los proyectos Linpack y Eispack desarrollados en el Argonne National Laboratory. Su nombre proviene de MATrix LABoratory. Al pasar de los años fue complementado y reimplementado en lenguaje C. Actualmente la licencia de MatLab es propiedad de MathWorks Inc .

1.3 Plataformas

MatLab está disponible para una amplio número de plataformas: estaciones de trabajo SUN, Apollo, VAXstation y HP, VAX, MicroVAX, Gould, Apple Macintosh y PC AT compatibles 80386 o superiores. Opera bajo sistemas operativos UNIX, Macintosh y Windows.

1.4 Productos

La empresa MathWorks ofrece MatLab como su principal producto para computación numérica, análisis y visualización de datos. También ofrece Simulink

como un anexo a MatLab y que interactua con él en lenguaje de MatLab y lenguaje de bajo nivel C. Simulink es usado para simulación modelado no lineal avanzado. Se ofrecen además numerosas herramientas especiales en "Toolboxes" para resolver problemas de aplicaciones específicas, por ejemplo control, procesamiento de señales, redes neurales, etc. Estas herramientas son colecciones de rutinas escritas en MatLab.

2. Librería de Aplicaciones de MATLAB

2.1 SIGNAL PROCESSING TOOLBOX

MATLAB tiene una gran colección de funciones para el procesamiento de señal en el Signal Processing Toolbox. Este incluye funciones para:

- Análisis de filtros digitales incluyendo respuesta en frecuencia, retardo de grupo, retardo de fase.
- Implementación de filtros, tanto directo como usando técnicas en el dominio de la frecuencia basadas en la FFT.
- Diseño de filtros IIR, incluyendo Butterworth, Chebyschev tipo I, Chebyshebv tipo II y elíptico.
- Diseño de filtros FIR mediante el algorítmo óptimo de Parks-McClellan.
- Procesamiento de la transformada rápida de Fourier FFT, incluyendo la transformación para potencias de dos y su inversa, y transformada para no potencias de dos.

2.2 THE MATLAB C MATH LIBRARY

La MATLAB C Math Library proporciona al usuario la capacidad computacional de MATLAB en una libreria en formato objeto enlazable. El objetivo principal de la C Math Library es soportar el desarrollo de aplicaciones 'stand alone' utilizando MATLAB y su compilador. Puede ser utilizada independientemente de MATLAB por programadores avezados en lenguaje C que necesiten prestaciones computacionales robustas y de alto rendimiento.

Junto con el compilador de MATLAB, la C Math Library permitirá a los programadores de aplicaciones utilizar MATLAB para la creación de aplicaciones 'stand alone'. Para los usuarios clásicos de MATLAB, se elimina así cualquier necesidad de volver a reescribir algoritmos en lenguaje C para ser utilizada por programas externos. Para aquellos usuarios que sean nuevos en la tecnología MATLAB, esta tecnología ofrece una nueva vía para la reducción del tiempo de desarrollo y puesta a punto de aplicaciones.

La MATLAB C Math Library proporciona una amplia gama de funciones clásicas del programa MATLAB, proporcionadas como librerias objeto, incluyendo básicamente las siguientes categorías de funciones presentes en MATLAB y ficheros M compilados:

- Algebra lineal.
- Funciones matemáticas elementales y especializadas.
- Operadores lógicos y aritméticos.

- Matrices elementales y manipulación de vectores.
- Matrices especiales.
- Estadística básica y análisis de datos.
- Polinomios e interpolación.
- Gestión de cadenas de caracteres.
- Entradas y Salidas.
- Gestión de memoria y errores.

(Nota: Las funciones del tipo Handle Graphics no están incluidas en la C Math Library).

2.2.1 Desarrollo de aplicaciones utilizando la MATLAB C Math Library

La construcción y desarrollo de aplicaciones utilizando esta librería es un proceso de amplias perspectivas una vez se tiene un dominio adecuado de su operativa. El producto está dividido en dos categorías (como librerías objeto): la librería (built-in library) contiene versiones de las funciones de MATLAB en lenguaje C del tipo numérico, lógico y utilidades. Por otra parte la librería de toolboxes (toolbox library) contiene versiones compiladas de la mayoría de ficheros M de MATLAB para cálculo numérico, análisis de datos y funciones de acceso a ficheros y matrices.

En equipos UNIX estas librerias pueden ser igualmente obtenidas como librerías de tipo estático (static libraries) o bien como librerías compartidas (shared libraries). Respecto al mundo PC, estas librerías pueden obtenerse como DLL's en el entorno Microsoft Windows o como librerias compartidas en equipos Apple MacIntosh.

2.2.2 Utilización de MATLAB y de su compilador

Para construir una aplicación del tipo 'stand alone' que incorpore código originalmente desarrollado como ficheros M de MATLAB , deberán seguirse los pasos siguientes:

1. Utilizar el compilador de MATLAB para convertir ficheros M en C mediante la utilización de la instrucción mcc -e (la cual es externa a MATLAB).
2. Compilar el código C fuente en código objeto utilizando un compilador ANSI C.
3. Enlazar el código resultante con la MATLAB C Math Library y con cualquier tipo de ficheros y programas específicos que hayan sido previamente definidos por el usuario.

2.2.3 Velocidad y Precisión

Los algoritmos utilizados en la MATLAB C Math Library han sido desarrollados por un grupo de renombrados expertos en programación algorítmica de funciones de tipo matemático (algebra lineal y cálculo numérico). Las funciones de álgebra lineal han sido obtenidas de las librerias mundialmente reconocidas LINPACK y EISPACK. La MATLAB C Math Library contiene más de 300 funciones numéricas, lógicas y de utilidad. Todas estas funciones le permitirán operar en datos de tipo escalar, vectorial o matricial con la misma facilidad sintáctica.

2.2.4 Lista parcial de funciones

Funciones matemáticas

Funcionales especiales y elementales

- Funciones gamma, beta y elípticas.
- Transformación de sistemas de coordenadas.
- Matriz identidad y otras matrices elementales.
- Matrices de Hilbert, Toeplitz, Vandermonde, Hadamard, etc.
- Partes reales, imaginarias y complejas conjugadas.
- Funciones trigonométricas y de potencias.

Algebra lineal numérica

- Valores propios y descomposición de matrices.
- Funciones generales de evaluación de matrices.
- Determinantes, normas, rangos, etc.
- Matrices inversas y factorización de matrices.
- Matriz exponencial, logarítmica y raíces cuadradas.

Polinomios e interpolación

- Interpolación 1-D y 2-D.
- Construcción polinomial.
- Interpolación por splines cúbicos.
- Diferenciación de polinomios.
- Evaluación de polinomios.
- Multiplicación y división de polinomios.
- Residuos de polinomios y residuos.

Métodos numéricos no lineales

Búsqueda de ceros en funciones de una única variable.

Minimización de funciones de una o más variables.

Resolución numérica de integrales.

Solución numérica de ecuaciones diferenciales ordinarias.

Estadística y análisis de Fourier

Convolución 1-D y 2-D.

Filtros digitales 1-D y 2-D.

Transformadas de Fourier 1-D y 2-D y su inversa.

Coeficientes de correlación y matrices de covarianza.

Deconvolución.

Magnitudes y ángulos de fase.

Funciones max, min, sum, mean y otras funciones de estadística básica.

Operaciones algebráicas y lógicas

Suma, resta, multiplicación, división y potencias de matrices.

Matrix traspuesta.

Operadores lógicos AND, OR, NOT y XOR.

2.2.5 Utilidades

Gestión y mantenimiento de errores.

Conversión de tipos de datos Fortran.

Funciones de fecha y hora.

Clasificación de matrices.

Conversión de números a cadenas y viceversa.

2.2.6 Requerimientos

La libreria MATLAB C Math Library cumple con la normativa estándar ANSI para compiladores C.

Finalmente, la librería trabajará con aquellos enlazadores que vienen suministrados con la mayoría de compiladores ANSI C.

2.3 THE MATLAB COMPILER TOOLBOX

El nuevo compilador de MATLAB -The MATLAB Compiler- permite crear código C optimizado procedente de ficheros M -M files- de MATLAB. Este compilador puede ser utilizado de dos modos:

1. Como un generador MEX automático. Pueden convertirse ficheros M en funciones C ejecutables que se ejecutaran desde dentro de MATLAB. Como un generador de código C fuente.
2. Pueden construirse aplicaciones que se ejecutaran independientemente de MATLAB. Estas aplicaciones externas requieren de la MATLAB C Math Library, que está disponible separadamente.

Mediante la conversión automática de ficheros M en código C fuente, el compilador MATLAB elimina consumo de tiempo y la conversión manual de código.

Todo el proceso de conversión, compilación y enlazado se inicia a través de una simple instrucción de MATLAB.

2.3.1 Generación Automática de ficheros MEX.

El compilador de MATLAB automatiza la creación de ficheros MEX de C (MATLAB Ejecutables).

Los ficheros MEX contienen código objeto que es dinámicamente enlazado como 'runtime' en el entorno MATLAB por el intérprete del programa.

El proceso en cuestión se realiza en tres pasos:

1. El compilador de MATLAB traduce las funciones MATLAB en sus funciones equivalente en lenguaje C.
2. La instrucción MATLAB cmex llama al compilador y al enlazador del sistema para construir un fichero MEX objeto.
3. El intérprete de MATLAB enlaza automáticamente la función de MATLAB como 'runtime'.

Mientras se efectúa una conversión de los ficheros M en ficheros MEX, el compilador realiza llamadas a las rutinas de la libreria C para muchas de las instrucciones contenidas en el propio núcleo de MATLAB. Existen algunas funciones, incluyendo las rutinas 'Handle Graphics', para las cuales se generan de nuevo llamadas 'callbacks' a MATLAB.

Pueden convertirse convenientemente ficheros M en código fuente C para incorporarlos posteriormente en los ficheros externos desarrollados en lenguaje C, si ese es el caso. Esta opción es ideal para usuarios que quieren sacar la máxima ventaja de MATLAB desde cualquier otra aplicación o producir código C eficiente a partir de los algoritmos desarrollados con MATLAB. Los desarrollos

del tipo 'stand-alone' requieren para ello de la MATLAB C Math Library. Obsérvese que las funciones gráficas de MATLAB no están incluidas.

Para construir aplicaciones 'stand-alone' se debería seguir los siguientes pasos:

1. Utilizar el compilador de MATLAB para convertir ficheros M en C con la instrucción externa mcc -e.
2. Compilar el código C fuente en código objeto utilizando un compilador C.
3. Enlazar el código resultante con las librerías matemáticas C de MATLAB y los ficheros específicos que dispongamos.

2.3.2 Rendimiento del compilador

Mediante la compilación de los ficheros M se puede obtener un rendimiento significativo. La velocidad de mejora de este rendimiento, depende fuertemente de cada aplicación. En algunos casos el rendimiento puede mejorar hasta en 200 veces la ejecución si la comparamos con el modo de trabajo interpretado del programa. Las operaciones matriciales y vectoriales ejecutadas desde MATLAB ya están fuertemente optimizadas en su diseño. Sin embargo, mediante la utilización del compilador se obtendrán significativas mejoras.

2.3.3 Opciones de ajuste del rendimiento

El compilador de MATLAB ofrece varias opciones que permiten generar el programa final de la forma más eficiente. Por ejemplo, Ud. puede directamente:

- Tratar todas las variables en ficheros como datos enteros y/o reales.
- Utilizar una variable concreta como variable escalar, vectorial, entera, real o una combinación de estas.
- Desactivar el control de parámetros de entrada y el redimensionamiento dinámico de vectores.

2.3.4 Requerimientos del sistema

Para utilizar el compilador de MATLAB para crear ficheros MEX se necesita la versión de MATLAB 4.2c y tener instalado uno de los siguientes compiladores de lenguaje C:

PC/Microsoft Windows

Metaware High C/C++ V.3.0 o superior.

Watcom C V.10.0 o superior

Power MacIntosh

MetroWerks CodeWarrior C V.7

MPW MrC V.1.0b2 o PPCC version 1.0.5

680x0 MacIntosh

MPW C Versión 3.4

UNIX y VMS

Cualquier compilador ANSI C (Nota: El compilador de SunOS 4.1.X no es un compilador ANSI C).

Cualquiera que sea el equipo informático que vaya a utilizarse para desarrollar aplicaciones 'stand alone' se requiere, además del compilador de MATLAB, que se tengan las MATLAB C Math Library y un compilador ANSI C.

2.3.5 Limitaciones del código compilado

Ciertas instrucciones, como load y eval, no están soportadas por el compilador de MATLAB . Este no puede generar código de los diagramas de bloques de SIMULINK. Los toolboxes de MATLAB pueden incluir ficheros MEX y otros componentes que no son compilables.

2.4 SYMBOLIC MATH TOOLBOX

El Toolbox de Matemática Simbólica, añade a MATLAB la capacidad de realizar cálculos simbólicos basados en MAPLE V © soportando además (The Extended Symbolic Math Toolbox) las librerías especializadas, y los programas realizados para este último. Entre otros, los principales tipos de operaciones soportados son los siguientes:

- Algebra simbólica: Derivación, integración y simplificación de expresiones matemáticas.
- Algebra lineal exacta: Inversas, determinantes, autovalores y formas canónicas de matrices simbólicas.
- Aritmética de precisión variable: Evaluación de expresiones matemáticas con diversos grados de precisión.
- Resolución de ecuaciones: Resolución numérica y simbólica de ecuaciones algebraicas y diferenciales.
- Funciones matemáticas especiales: Evaluación de la mayoría de las funciones utilizadas en matemáticas aplicadas.

Existen dos versiones del mismo Toolbox. The Basic Symbolic Math Toolbox es una colección de más de 50 funciones MATLAB las cuales permiten acceder al

kernel de MAPLE utilizando la Sintaxis y el estilo del lenguaje MATLAB. The Extended Symbolic Math Toolbox aumenta esta funcionalidad incluyendo todas las características de programación de MAPLE, y el acceso a los paquetes de funciones de más de veinte campos de las matemáticas especiales aplicadas.

Es posible utilizar este Toolbox sin conocimiento previos de MAPLE, ya que los ficheros contenidos en él son totalmente autónomos. Sin embargo, si lo que se desea es obtener toda la potencia de cálculo del entorno, será necesario un amplio conocimiento del manejo y la programación de MAPLE

2.5 OPTIMIZATION TOOLBOX

El toolbox de optimización consta de un conjunto de funciones que resuelven problemas de extremos, con o sin condiciones, de funciones reales las cuales son generalmente multivariables y no lineales.

Asimismo, posee funciones para la resolución de algunos tipos de problemas matriciales en extremos.

Resulta conveniente para una comprensión y mejor manejo de la toolbox poseer conocimientos básicos previos de análisis de funciones reales, matrices y teoría de extremos.

Algunas de las áreas básicas que cubre este toolbox para MATLAB son las siguientes:

- Cálculo de un extremo local (máximo o mínimo) de una función real f(x), en general multivariable y no lineal, sin imponer ninguna restricción o condición a la solución. Como caso particular, se incluye una rutina especial para problemas de mínimos cuadrados no lineales.
- Cálculo de un extremo local (máximo o mínimo) de una función real f(x), en general multivariable y no lineal, condicionado a que la solución satisfaga ciertas condiciones de desigualdad (g(x)<=0) y/o igualdad (g(x)=0).
- Problemas de aproximación a un conjunto de objetivos.
- Cálculo de soluciones de un sistema de ecuaciones continuas y, en general, no lineales.
- Solución de problemas minimax.
- Programación lineal.
- Programación cuadrática.
- Problemas de mínimos cuadrados no negativos.

2.6 IMAGE PROCESSING TOOLBOX

Este Toolbox proporciona a MATLAB de un conjunto de funciones que amplia las capacidades del producto para realizar desarrollo de aplicaciones y de nuevos algoritmos en el campo del proceso y análisis de imagenes. El entorno matemático y de creación de MATLAB es ideal para el procesado de imágenes, ya que estas imágenes son, al fin y al cabo, matrices. Este toolbox incorpora funciones para:

- Diseño de filtros.
- Mejora y retocado de imágenes.
- Análisis y estadística de imágenes.
- Operaciones morfológicas, geométricas y de color.
- Transformaciones 2D.

El proceso de imágenes es un campo de trabajo absolutamente crucial para aquellos colectivos e industrias que estén trabajando en áreas como diagnóstico médico, astronomía, geofísica, ciencia medioambientales, análisis de datos en laboratorios, inspección industrial, etc. Los programas actuales de procesado y análisis de imágenes se clasifican actualmente en dos categorías: librerías de bajo nivel para programadores profesionales y paquetes de aplicación con capacidades limitadas de personalización. Ambos tipos de aplicaciones están, generalmente, pensados para tareas básicas de visualización de datos y 'rendering'. Sin embargo, muchos de ellos adolecen de la posibilidad de efectuar análisis numéricos de los mismos. El Image Processing Toolbox entra dentro de la categoría de familias de funciones que, desde el entorno de trabajo de MATLAB , permitirá al profesional efectuar una exploración exhaustiva y desde un punto de vista matemático de las imágenes y gráficos que se deseen tratar o analizar.

Algunas de las funciones más importantes incluidas dentro de este toolbox son las siguientes:

- Análisis de imágenes y estadística.
- Diseño de filtros y recuperación de imágenes.
- Mejora de imágenes.
- Operaciones morfológicas.
- Definición de mapas de colores y modificación gráfica.
- Operaciones geométricas.
- Transformación de imágenes.
- Proceso de bloques

2.7 Neural Network Toolbox

Este toolbox proporciona funciones para el diseño, inicialización, simulación y entrenamiento de los modelos neuronales de uso más extendido en la actualidad: Perceptrón, redes lineales, redes de retropropagación, redes de base radial, aprendizaje asociativo y competitivo, aplicaciones autoorganizativas, aprendizaje de cuantización vectorial, redes de Elman y redes de Hopfield.

Mediante la inclusión de un amplio abanico de funciones y procedimientos escritos para MATLAB, el usuario puede mediante el Neural Network Toolbox efectuar el diseño de arquitecturas complejas, combinando los modelos que ya estan proporcionados por defecto en el toolbox. Asimismo, el usuario puede definir sus propias funciones de transferencia e inicialización, reglas de aprendizaje, funciones de entrenamiento y estimación de error para usarlas posteriormente con las funciones básicas.

El toolbox, aporta las facilidades y prestaciones gráficas de MATLAB para el estudio del comportamiento de las redes: visualización gráfica de la matriz de pesos y vector de desplazamiento mediante diagramas de Hinton, representación de errores a lo largo del entrenamiento, mapas de superficie de error en función de pesos y vector de desplazamiento, etc. Estos gráficos resultan muy útiles en el estudio de la convergencia y estabilidad de los algoritmos de aprendizaje. Este toolbox incluye un manual de introducción al campo de las redes neuronales junto con una colección de demostraciones y aplicaciones muy didácticas, útiles para el estudio y la profundización en las cuestiones fundamentales de los paradigmas de redes neuronales básicos. Asimismo, se proporcionan las referencias bibliográficas más significativas referidas a los distintos modelos que aparecen en la aplicación.

A pesar de que el estudio de las redes neuronales se inició ya hace algunas decadas, las primeras aplicaciones sólidas dentro de este campo no han tenido lugar hasta hace unos doce años y aun ahora constituyen un área de investigación en rápido desarrollo. Este toolbox tiene por tanto una orientación diferente a aquellos destinados a campos como el de sistemas de control u optimización donde la terminología, fundamentos matemáticos y procedimientos de diseño estan ya firmemente establecidos y se han aplicado durante años. Este toolbox pretende que sea utilizado para la valoración y diseño de diseños neuronales en la industria y sobre todo en educación e investigación.

Esta herramienta tiene el soporte de MATLAB 4.2c y SIMULINK. La librería de SIMULINK contiene modelos de capas de redes neuronales de cada tipo de neurona implementada en el toolbox de redes neuronales. Es posible por tanto diseñar sistemas SIMULINK para simular redes neuronales creadas usando esta herramienta. Simplemente, las capas se conectan de acuerdo con la arquitectura de la red y se proporcionan como entrada a la caja de diálogo de cada capa la matriz de pesos apropiada y el vector de desplazamiento. Usando el generador de código C de SIMULINK es posible generar automáticamente el código correspondiente a un diseño neuronal.

Dentro de las aplicaciones básicas de este toolbox, cabe destacar aquellas que están orientadas a aquellas que se enmarcan dentro del campo de la industria aeroespacial y automoción (simulación, sistemas de control, autopilotaje), banca, defensa (reconocimiento de patrones, procesamiento de señales, identificación de imágenes, extracción de características, compresión de datos), electrónica (control de procesos, análisis de errores, modelado no lineal, síntesis de voz, visión por ordenador), economía (análisis financiero, análisis predictivo), industria (control de procesos, identificación en tiempo real, sistemas de inspección), medicina, robótica (control de trayectorias, sistemas de visión), reconocimiento y síntesis del habla, telecomunicaciones (control de datos e imágenes, servicios de información automatizada, traducción del lenguaje hablado en tiempo real, diagnosis, sistemas de enrutamiento), etc. El toolbox contiene muchos ejemplos de algunas de estas aplicaciones.

2.8 NON LINEAR CONTROL DESIGN TOOLBOX

Se trata del primer producto comercialmente disponible en la actualidad para el diseño de controladores automáticos en entornos de sistemas no lineales. Este nuevo toolbox está pensado para ser utilizado exhaustivamente por ingenieros que diseñan controladores para industrias avanzadas, destacando el sector del automóvil, ingenieria aeroespacial, control de procesos y empresas petroquímicas. Según indica Jim Tung, Vicepresidente del área de desarrollo de The MathWorks Group, Inc. "El proceso de aproximación tradicional en el diseño de controladores en sistemas no lineales ha sido hasta la fecha linealizarlos de algún modo para aplicar posteriomente un método de diseño lineal que requiere de importantes ajustes manuales. El toolbox NCD permite por primera vez a los ingenieros de control diseñar directamente sus controladores en un ambiente no lineal, obviando la aproximación lineal y otros procedimientos auxiliares que antes se necesitaban de modo imperativo.

Los resultados ahora son de elevada calidad, controladores más robustos y un ciclo de diseño mucho más rápido.

El toolbox NCD extiende, además, las prestaciones que incorpora SIMULINK, el entorno de desarrollo de diagramas de bloques para la modelación y análisis de sistemas dinámicos de The MathWorks, Inc. El usuario puede incluir uno o más bloques NCD en el sistema y describir posteriormente de modo totalmente gráfico las restricciones, tolerancias y límites de permisividad de cada uno de estos bloques. Los métodos avanzados de optimización y la simulación del proceso son posteriormente analizados y ajustados mediante la inclusión de unas ciertas variables de contorno para poder obtener los tiempos de respuesta deseados. Este toolbox puede ser utilizado para ajustar una amplia variedad de controladores que se utilicen en un sistema, destacando los controladores PID, LQR, LQG y estructuras H infinito. El diseñador de sistemas puede utilizar el método de Montecarlo para el diseño y análisis de controladores robustos,

siempre que se detecten determinadas variaciones en los componentes del sistema.

El toolbox NCD es un componente avanzado del entorno integrado de desarrollo que ofrecen a los especialistas los programas MATLAB y SIMULINK. Por ello, los diseñadores podrán beneficiarse de muchos de los toolboxes desarrollados para este entorno en materia de diseño de sistemas lineales.

Por ejemplo, podrán utilizarse toolboxes para el análisis de sistemas lineales para el diseño inicial; posteriormente, podrán utilizarse modelos no lineales más sofisticados utilizando SIMULINK.

Además, puede invocarse NCD para un mejor ajuste paramétrico y para la optimización de los controladores. Este toolbox se encuentra actualmente disponible para una amplia variedad de plataformas informáticas, destacando ordenadores personales tipo PC o Apple MacIntosh, numerosas estaciones UNIX y ordenadores Digital VAX VMS.

2.9 NAG FOUNDATION TOOLBOX

Este toolbox proporciona un acceso interactivo, desde dentro de MATLAB, a un amplio conjunto de funciones matemáticas y estadísticas contenidas en las clásicas NAG Fortran Libraries de la empresa The Numerical Algorithms Group Incorpora más de 200 ficheros M, los cuales cubren un amplio espectro de áreas de interés, entre las que cabe destacar optimización, ecuaciones diferenciales ordinarias y en derivadas parciales, cuadratura, estadística, etc. La NAG Foundation Toolbox añade también rutinas concretas para campos específicos tales como la resolución de problemas con condiciones de contorno, problemas de cuadratura adaptativa multidimensional, ajuste de curvas y superficies y el acceso a los algoritmos LAPACK para la resolución de ecuaciones lineales. Los nombre de las funciones han sido directamente tomados de las especificaciones de función clásica que añade The Numerical Algorithms Group para sus librerías. Como resultado de esto, aquellos usuarios de las librerías Fortran de NAG que a la vez sean usuarios de MATLAB, encontraran bastante cómodo acceder a las rutinas NAG utilizando la nomenclatura original.

La NAG Foundation Toolbox es resultado de la colaboración corporativa que actualmente están llevando a cabo The MathWorks Group y The Numerical Algoriths Group para proporcionar un rápido acceso desde MATLAB a un importante de rutinas matemáticas contenidas en la NAG Foundation Library. Actualmente, este toolbox incorpora 250 rutinas matemáticas.

Algunas de las áreas de cobertura de la NAG Foundation Toolbox son las siguientes:

- Ceros de polinomios
- Raíces de una o más ecuaciones de tipo trascendental.

- Suma de series.
- Cuadraturas.
- Ecuaciones diferenciales ordinarias.
- Ecuaciones diferenciales en derivadas parciales.
- Estadística no paramétrica.
- Análisis de series temporales.
- Rutinas de clasificación.
- Aproximación de funciones especiales.
- Aproximación de curvas y superficies.
- Maximización y minimización de funciones.
- Factorización de matrices.
- Valores y vectores propios.
- Resolución de ecuaciones lineales simultáneas.
- Ecuaciones lineales (LAPACK).
- Estadística básica.
- Análisis de correlación y regresiones.
- Métodos multivariantes.
- Generación de números aleatorios.

3. INICIANDO MATLAB

Después de ejecutar el programa MatLab desde el sistema operativo empleado, por ejemplo haciendo doble click sobre el icono de MatLab en ambientes Windows, aparece el indicador de comandos el cual está listo para recibir instrucciones en lenguaje MatLab. Este indicador es de la siguiente forma:

>>

Al iniciar el uso de MatLab están disponibles dos comandos de ayuda y demostración. Para ejecutarlos se escribe el comando en la línea de comandos después del símbolo >> y se presiona la tecla Enter. Por ejemplo:

>>help

permite obtener una ayuda sobre los diferentes comandos de MatLab.

>>demo

hace una demostración de las diferentes aplicaciones de MatLab.

Para cerrar o finalizar el uso de MatLab se usa el comando quit.

>>quit

4. USO DE COMANDOS

La primera forma de interactuar con MatLab es a través de la línea de comandos. Puede ejecutarse un comando si este está escrito después del símbolo >> y se presiona la tecla Enter.

MATLAB trabaja esencialmente con matrices numéricas rectangulares. La manera más fácil de entrar matrices pequeñas es enumerando los elementos de ésta de tal manera que:

- los elementos estén separados por blancos ó comas.
- los elementos estén cerrados entre corchetes, [ ].
- muestre el final de cada fila con ; (punto y coma).

Ejemplo:

A = [1 2 3; 4 5 6; 7 8 9]

resultaría en la matriz

A =

1 2 3

4 5 6

7 8 9

MATLAB guarda esta matriz para utilizarla luego bajo el nombre de A.

Si la matriz a introducir es muy grande se puede utilizar el siguiente formato:

```
A = [1 2 3
4 5 6
7 8 9]
```

El comando load y la función fread pueden leer matrices generadas en sesiones anteriores ó generadas por otros programas.

Ya que MatLab se basa en el álgebra de matrices como ejemplo crearemos una matriz. Estas pueden estar formadas por un sólo elementos (escalar), por una fila o una columna (vector) o por una serie de filas y columnas (matriz propiamente dicha).

```
>>A=1
```

define A como un escalar de valor 1. Al definir A automáticamente MatLab presenta en pantalla su valor.

```
A =
1
```

Para no presentar el valor de la variable creada, debe agregarse punto y coma (;) al final del comando.

Después de crear una variable, puede presentarse su valor en pantalla escribiendo la variable después del prompt (>>).

```
>>A
```

Se pueden redefinir variables, por ejemplo:

```
>>A=[1 2 3]
```

define A como un vector de tres elementos, A(1)=1, A(2)=2 y A(3)=3. Estos elementos deben separase con espacios en blanco o comas (,).

Para definir una matriz se deben separar las filas con punto y coma (;) o con retorno (Enter).

```
>>A=[1 2 3; 4 5 6]
```

o

```
>>A=[1 2 3
4 5 6]
```

ambos comandos producen el mismo efecto:

```
A =
1 2 3
4 5 6
```

4.1 Elementos de matrices

Los elementos de una matriz pueden ser cualquier expresión de MATLAB.

Ejemplo:

```
x = [-1.3,sqrt(3),(1+2+3) *4/5]
```

resultaría en

```
x =

-1.3000 1.7321 4.8000
```

Nos podemos referir a elementos individuales de la matriz con índices entre paréntesis.

Ejemplo: En el ejemplo anterior

```
x(4) = abs(x(1))
```

resultaría

```
x =

-1.3000 1.7321 4.8000 0 1.3000
```

Para añadir otra fila a la matriz A de arriba podemos hacer lo siguiente:

```
r = [10 11 12];

A = [A; r]
```

y resultaría

```
A =

1 2 3

4 5 6

7 8 9

10 11 12
```

4.2 Instrucciones de MATLAB y Variables

Si omites el nombre de la variable y el signo "=", MATLAB automáticamente crea la variable `ans` para guardar el resultado. También distingue las letras mayúsculas de las minúsculas. Todos los nombres de funciones deben ser en letras minúsculas.

4.3 Obteniendo Información del Espacio de Trabajo

Los ejemplos que hemos dado se han guardado en variables que están en el espacio de trabajo de MATLAB. Para listar las variables en el espacio de trabajo se utiliza el comando `who`. Para ver información adicional acerca de estas variables se utiliza el comando `whos`.

4.4 Variables Permanentes

Las variables permanentes son aquellas con significado especial, y que no se pueden eliminar. Estas son por ejemplo las variables `ans` y `eps`.

La variable `eps` es una tolerancia para determinar. Por ejemplo la singularidad y el rango. Su valor inicial es la distancia de 1.0 al próximo número de punto flotante mayor.

4.5 Funciones

Las funciones que utiliza MATLAB son intrínsecas al procesador de éste. Otras funciones están disponibles en la librería externa de archivos-M. Además de éstas funciones todo usuario también puede crear otras funciones. Puedes combinar las funciones de acuerdo a tu necesidad.

Ejemplo:

```
x = sqrt(log(z))
```

4.6 Saliendo y Guardando el Espacio de Trabajo

Para salir de MATLAB se escribe `quit` ó `exit`. Al terminar una sesión de MATLAB, las variables en el espacio de trabajo se borran. Si deseas guardar tu espacio de trabajo escribes `save`.

`save` guarda todas las variables en un archivo llamado matlab.mat.

Se puede utilizar `save` y `load` con otros nombres de archivos, ó para guardar solo variables seleccionadas

Ejemplo:

```
save temp X Y Z
```

Este ejemplo guarda las variables X, Y, Z en el archivo temp.mat. Usando el comando `load temp` las obtienes nuevamente del archivo temp.mat. `load` y `save` también pueden importar y exportar información de archivos ASCII.

4.7 Manipulación de Vectores y Matrices

Generando Vectores

Los dos puntos, :, son importantes en MATLAB. Por ejemplo

```
x = 1:5
```

genera un vector fila que contiene los números enteros del 1 al 5:

```
x =

1 2 3 4 5
```

No necesariamente se tiene que incrementar por números enteros, pueden ser decimales, números negativos ó constantes.

Índices

Podemos referirnos a elementos individuales de matrices encerrando sus índices en paréntesis.

Ejemplo:

```
A =

1 2 3

4 5 6

7 8 9
```

```
A(3, 3) = A(1, 3) + A(3, 1)
```

resultaría

```
A =

1 2 3

4 5 6

7 8 10
```

Un índice puede ser un vector. Si x y v son vectores, entonces x(v) es [x(v(1)), x(v(2)), ...,x(v(n))]. Para matrices, los índices de vectores permiten acceso a submatrices contiguas y no-contiguas.

Por ejemplo, suponga que A es una matriz 10 por 10. Entonces

A(1:5, 3)

especifica la submatriz 5 x 1, ó vector columna, que consiste de los primeros cinco elementos en la tercera columna de A.

También

A(1:5, 7:10)

es la submatriz 5 x 4 de las primeras cinco filas y las últimas cuatro columnas.

Utilizando solo los dos puntos denota todo lo correspondiente a la fila ó columna. Podríamos tener una instrucción como:

A(:, [3 5 10]) = B(:, 1:3)

que reemplaza la tercera, quinta y décima columna de A con las primeras tres columnas de B.

Manipulación de Matrices

diag - extrae ó crea una diagonal

tril - parte inferior triangular

triu - parte superior triangular

' - transposición

4.8 Operaciones de Matrices

Matrices Transpuestas

El caracter ' (apóstrofe) denota la transpuesta de la matriz. Si tenemos la matriz A y llamamos B = A', B es la transpuesta de la matriz A.

Sumando y Restando Matrices

Las operaciones suma (+) y resta (-) son definidas para las matrices siempre y cuando éstas tengan la misma dimensión. Es decir, si A y B son matrices 3 x 3, entonces A + B se puede calcular.

Las operaciones suma y resta también está definidas si uno de los operandos es un escalar, es decir, una matriz 1 x 1.

Ejemplo:

x =

-1

0

2

y = x - 1

resultaría

y =

-2

-1

1

Ejemplo:

>>A=[1 2 3;4 5 6]; B=[6 5 4; 3 2 1];

define las matrices A y B. Para sumarlas se escribe la operación:

>>A+B

El resultado de la operación es por defecto almacenado en la variable ans e inmediatamente presentado en pantalla:

ans =

7 7 7

7 7 7

Para almacenar la suma de A y B en la variable C:

>>C=A+B

C =

7 7 7

7 7 7

Multiplicando Matrices

La operación de multiplicación de matrices está definida siempre que el número de columnas de la primera matriz sea igual a el número de filas de la segunda matriz.

Producto escalar

El producto interior (producto escalar ó producto punto) se consigue de la siguiente manera:

x' * y

asumiendo que x y y son vectores columnas. Note que y' * x produce el mismo resultado.

Producto de una matriz por un vector

El producto de una matriz y un vector es un caso especial del producto matriz-matriz y naturalmente, un escalar como pi, puede multiplicar, ó ser multiplicado por, cualquier matriz.

Dividiendo Matrices

En división de matrices, si A es una matriz cuadrada no-singular, entonces A\B y B/A corresponden a la multiplicación izquierda y derecha de B por el inverso de A, esto es, inv(A) * B y B * inv(A) respectivamente. El resultado es obtenido directamente sin la computación del inverso.

X = A\B es una solución a A * X = B

X = B/A es una solución a X * A = B

A\B es definido cuando B tiene la misma cantidad de filas que A. Si A es cuadrada, el método usado es la Eliminación Gaussiana. El resultado es una matriz X con las mismas dimensiones que B.

Si A no es cuadrada, se factoriza utilizando la ortogonalización de Householder con pivoteo de columnas.

Los factores son usados para resolver sistemas de ecuaciones sub-determinados y sobre-determinados. El resultado es una matriz X m-por-n donde m es el número de columnas de A y n es el número de columnas de B. Cada columna de X tiene, al menos, k componentes diferentes de cero, donde k es el rango efectivo de A.

B/A esta definido en términos de A\B por B/A = (A'\B') '.

Usando Exponentes con Matrices

La expresión A^n eleva A a la n-ésima potencia y esta definido si A es una matriz cuadrada y n un escalar.

Funciones Matriciales Trascendentales y Elementales

MATLAB considera expresiones como exp(A) y sqrt(A) como operaciones de arreglos, definidas en los elementos individuales de A. También puede calcular funciones trascendentales de matrices, como la matriz exponencial y la matriz

logarítmica. Estas operaciones especiales están definidas solamente para matrices cuadradas.

Otras funciones elementales de matrices son:

poly - polinomio característico

det - determinante

trace - traza

kron - producto tensorial de Kronecker

eig - calcula los valores propios de la matriz

4.9 Operaciones de Arreglos

El término operaciones de arreglo se refiere a las operaciones de aritmética elemento por elemento. Un punto (.) antes de un operador indica una operación de arreglos elemento por elemento.

Suma y Resta de Arreglos

Para suma y resta, las operaciones de arreglos y las operaciones de matrices son iguales.

Multiplicación y División de Arreglos

El símbolo .* denota multiplicación de arreglos elemento por elemento.

Ejemplo:

```
x = [1 2 3]; y = [4 5 6];
z = x. *y
```

resulta

```
z =
4 10 18
```

Las expresiones A./B y A.\B dan los cocientes de los elementos individuales.

Ejemplo:

```
z = x.\y
```

resulta

```
z =
4.0000 2.5000 2.0000
```

Exponentes con Arreglos

El símbolo .^ denota exponenciación elemento por elemento.

4.10 Ejemplos: Operaciones Aritméticas

Ejemplos:

```
>> 1/2
ans =
0.5000

>> 2\1
ans =
0.5000

>> a=[2;1;2]
a =
2
1
2


>> b=[1;2;3]
b =
1
2
3

>> a'
ans =
2 1 2

>> b'
ans =
1 2 3
```

```
>> a*b
??? Error using ==> *
Inner matrix dimensions must agree.


>> a.*b
ans =
2
2
6

>> a*b'
ans =
2 4 6
1 2 3
2 4 6

>> a.*b'
??? Error using ==> .*
Matrix dimensions must agree.


>> a*3
ans =
6
3
6

>> b.*3
ans =
3
6
9
```

```
>> a/3
ans =
0.6667
0.3333
0.6667

>> a./3
ans =
0.6667
0.3333
0.6667

>> a^b
??? Error using ==> ^
Matrix dimensions must agree.

>> a.^b
ans =
2
1
8

>> a^2
??? Error using ==> ^
Matrix must be square.

>> a.^2
ans =
4
1
4

>> 2^a
??? Error using ==> ^
Matrix must be square.
```

```
>> 2.^a
ans =
4
2
4
```

Precisión utilizada.- Aproximadamente 16 dígitos significativos en computadoras utilizando aritmética flotante IEEE. El rango aproximado es:

10^-308 a 10^308.

Formatos de salida:

```
4/3
a) format short
1.3333
b) format short e
1.3333e+00
c) format long
1.33333333333333
d) format long e
1.33333333333333e00
e) format bank
1.33
f) format hex
3ff5555555555555
```

5. PROGRAMANDO CON MATLAB

5.1 Generalidades

Programar en MatLab es usar una serie de comandos que permitan realizar una tarea o función específica. Estos pueden ser escritos uno por uno a través de la línea de comandos:

```
>>A=[1 2 3;4 5 6;7 8 9]
A =
1 2 3
4 5 6
7 8 9
>>A'
ans =
1 4 7
2 5 8
3 6 9
```

El primer comando A=[1 2 3;4 5 6;7 8 9] define la matriz A y el siguiente comando A' calcula y presenta en pantalla la transpuesta de A.

5.1.1 Archivos-M: Comandos y Funciones

Los archivos de disco que contienen instrucciones de MATLAB se llaman archivos-M. Esto es así porque siempre tienen una extención de ".m" como la última parte de su nombre de archivo.

Un archivo-M consiste de una secuencia de instrucciones normales de MATLAB, que probablemente incluyen referencias a otros archivos-M. Un archivo-M se puede llamar a sí mismo recursivamente. Puedes crear archivos-M utilizando un editor de texto ó procesador de palabras.

Hay dos tipos de archivos-M: los de comandos y las funciones. Los archivos de comandos, automatizan secuencias largas de comandos. Los archivos de funciones, permiten añadir a MATLAB funciones adicionales expandiendo asi la capacidad de este programa. Ambos, comandos y funciones, son archivosordinarios de texto ASCII.

Archivos de Comandos

Cuando un archivo de comandos es invocado, MATLAB simplemente ejecuta los comandos encontrados en dicho archivo. Las instrucciones en un archivo de comando operan globalmente en los datos en el espacio de trabajo. Los comandos son utilizados para hacer análisis, resolver problemas, ó diseñar secuencias

largas de comandos que se conviertan en interactivas. Por ejemplo, suponga que el archivo fibo.m contiene los siguientes comandos de MATLAB:

```
% Un archivo-M para calcular los elementos de la serie de Fibonacci
f = [1 1]; i = 1;
while f(i) + f(i+1) < 1000
f(i+2) = f(i) + f(i+1);
i = i + 1;
end
plot(f)
```

Si escribimos fibo en una ventana de MATLAB seguido de "enter" vemos que MATLAB calcula los primeros 16 números de Fibonacci, y luego grafica estos. Luego que la ejecución del archivo es completada, las variables f y i permanecen en el espacio de trabajo.

Los programas de demostraciones incluidos en MATLAB son ejemplos de como usar comandos para hacer tareas más complicadas. Para utilizar estos escriba demos en el "prompt" de MATLAB.

Archivos de Funciones

Un archivo-M que contiene la palabra function al principio de la primera línea, es un archivo de función. En una función, a diferencia de un comando, se deben de pasar los argumentos. Las variables definidas y manipuladas dentro de la función son locales a esta y no operan globalmente en el espacio de trabajo. Los archivos de funciones se utilizan para extender a MATLAB, i.e., crear nuevas funciones para MATLAB utilizando el lenguaje propio de MATLAB.

El archivo mean.m contiene las instrucciones:

```
function y = mean(x)
% Valor medio.
% Para vectores, mean(x) retorna el valor medio de los elementos del vector x.
% Para matrices, mean(x) es un vector fila conteniendo el valor medio de cada columna.
[m, n] = size(x);
if m == 1
m = n;
end
y = sum(x)/m;
```

(Las lineas que comienzan con "%" son interpretadas como comentarios por MATLAB). La existencia de este archivo en el disco duro define una nueva función en MATLAB llamada mean. Si z es un vector de los enteros desde 1 a 99, por ejemplo,

```
z = 1:99;
```

entonces, el valor promedio es encontrado escribiendo

```
mean(z)
```

que resultaría

```
ans =
50
```

Veamos algunos detalles de mean.m:

La primera línea declara el nombre de la función, los argumentos de entrada, y los argumentos de salida. Sin esta línea sería un archivo de comando.

% indica que el resto de la línea es un comentario.

Las primeras líneas documentan el archivo-M y aparecen en la pantalla cuando escribimos help mean.

Las variables m, n, e y son locales a mean y no existen en el espacio de trabajo. (O si existen, permanecen sin cambios.)

No es necesario asignar los enteros de 1 al 99 en la variable x. Utilizamos mean con una variable llamada z.

Este vector que contenía los enteros de 1 a 99 fue pasado ó copiado a mean donde se convirtió en una variable local llamada x.

Ejemplo

```
% Ejemplo de un archivo-m
% Creación del vector x usando el comando for
n=5;
for i=1:n
x(i)=i^2;
end
x
% Fin del archivo-m
```

Este ejemplo es un archivo-m tipo comando. Para ejecutarlo, en la línea de comandos se debe escribir el nombre del archivo:

```
>>ejemplo
x =
1 4 9 16 25
```

Ejemplo

```
% Calcula el promedio de los elementos de un vector y dibuja dicho vector
% Sintaxis : promedio(x) donde x es el vector a promediar
function p = promedio(x)
n=length(x);
p=0;
for i=1:n
p=p+x(i);
end
p=p/n;
plot(x);
```

Para ejecutar la función, se hace la llamada en la línea de comandos incluyendo el parámetro. La función promedio usa por parámetro un vector. Este vector debe ser definido previamente.

```
>>A=[1 2 4 3 7 5 6 1 2 0 8 5];
>>promedio(A)
ans =
3.6667
```

MatLab presenta las imágenes en una ventana de figuras. Al observar el contenido de dicha ventana luego de ejecutar la función promedio, se tiene:

Esta imagen es el resultado del comando plot(x) al ejecutar la función promedio. MatLab posee un conjunto de archivos-m incorporados (built-in). Puede agregársele archivos-m definidos por el usuario almacenando los mismos en el directorio principal de MatLab. Los comentarios incluidos en estos scripts y funciones se visualizan al usar el comando help seguido del nombre del archivo.

>>help promedio

Calcula el promedio de los elementos de un vector y dibuja dicho vector

Sintaxis : promedio(x) donde x es el vector a promediar

Para ver el contenido de un archivo-m se usa el comando type seguido del nombre del archivo.

5.1.2 Otras funciones

Funciones Matemáticas

Algunas funciones trigonométricas utilizadas por MATLAB son:

sin - seno

cos - coseno

tan - tangente

asin - seno inverso

acos - coseno inverso

atan - tangente inversa

Algunas funciones elementales son:

real(a) Parte real

imag(a) Parte imaginaria

conj(a) Conjugado de a

fft(x) Transformada discreta de Fourier del vector x

fft(x,n) FFT de n puntos muestrales

ifft(x) Transformada inversa rápida de Fourier del vector x

ifft(x,n) FFT inversa de n puntos muestrados

zeros Inicializa a ceros

zeros(n) Matriz de nxn de ceros

zeros(m,n) Matriz de mxn de ceros

y=zeros(size(A) Matriz del tamaño de A, todos ceros

Ejemplo

size Regresa el número de filas y columnas

```
A =
0 7 -6
1 0 0
0 1 0

>> [m n]=size(A)
m =
3
n =
3
```

Funciones matriciales

tril(A) Matriz triangular inferior

triu(A) Matriz triangular superior

pascal Triangulo de Pascal

tocplitz Tocplitz

Ejemplos

```
>> A =
0 7 -6
1 0 0
0 1 0
>> toeplitz(A)
ans =
0 1 0 7 0 1 -6 0 0
1 0 1 0 7 0 1 -6 0
0 1 0 1 0 7 0 1 -6
```

Producto de dos matrices triangulares.

Esta factorización se utiliza para obtener el inverso y el determinante. También es la base para la solución de sistemas lineales. Para obtener la factorización LU de A escribimos,

[L, U] = lu(A).

Factorización Ortogonal ó Factorización QR.

Se utiliza para matrices cuadradas ó rectangulares. Esta factorización se utiliza para resolver sistemas lineales con más ecuaciones que desconocidas. Esta factorización también es la base para las funciones null y orth, que generan bases ortonormales para el espacio nulo y rango de una matriz rectangular dada.

Descomposición de Valores Singulares

La descomposición de Valores Singulares es importante para el análisis de problemas que envuelvan matrices.

La asignación triple [U, S, V] = svd(A) produce los tres factores en la descomposición de valores singulares A = U*S*V'. Las matrices U y V son ortogonales y la matriz S es diagonal.

La función svd(A) devuelve solamente los elementos de la diagonal de S, que son los valores singulares de A.

Descomposición de Valores Propios

La Descomposición de Valores Propios se utiliza para obtener los valores y vectores propios de una matriz cuadrada A.

La función eig(A) devuelve los valores propios de A en un vector columna.

La asignación [X,D]=eig(A) produce una matriz diagonal D cuyos elementos diagonales son los valores propios de A y las columnas de X son los vectores propios correspondientes.

Las Funciones de norma, rango y acondicionamiento asociadas son:

cond - número de condición en la norma 2

norm - norma 1, norma 2, norma F, norma

rank - rango

rcond - estimado del número de condición

Funciones de Funciones

MATLAB representa funciones matemáticas mediante archivos-M de tipo función. Un ejemplo de una función es el archivo-M llamado humps.m.

Ejemplo: El archivo-M llamado humps.m contiene las siguientes instrucciones:

```
function y = humps(x)
```

```
y = 1./((x-.3).^2 +.01) + 1./((x-.9).^2 +.04) - 6;
```

y para la gráfica de la función escribimos

```
x = -1:.01:2;
plot(x, humps(x))
```

Integración Numérica (Cuadratura)

El área bajo la gráfica de la función f(x) se puede aproximar integrando f(x) numéricamente mediante una regla de cuadratura. Para integrar la función definida por humps.m desde 0 hasta 1 escribimos:

```
q = quad('humps', 0, 1)
q =
29.8583
```

Note que el argumento de quad contiene un nombre de una función. Por esto quad se llama una función de función, i.e., es una función que opera en otras funciones.

Ecuaciones No-lineales y Funciones de Optimización

Las funciones de funciones para ecuaciones no-lineales y optimización incluyen:

fmin - mínimo de una función de una variable

fmins - mínimo de una función multi-variable (minimización no-lineal sin restricciones)

fzero - cero de una función de una variable

constr - minimización con restricciones

fsolve - solución de ecuación no-lineal

leastsq - cuadrados mínimos no-lineales

Funciones para Ecuaciones Diferenciales

Las funciones de MATLAB para resolver problemas de valor inicial para ecuaciones diferenciales ordinarias son:

ode23 - método Runge-Kutta de largo de paso variable que combina un método de orden dos con uno de orden tres.

ode45 - método Runge-Kutta-Fehlberg de largo de paso variable que combina un método de orden cuatro con uno de orden cinco.

Ejemplo

```
to=0; tf=10;
[t,x]=ode23(`edif',to,tf,xo);

[t,x]=ode23(`deriv',to,tf,xo);
ode45

[t,x]=ode23(`deriv',to,tf,xo,to1,trace);
ode45
trace => 0 - no resuntados intermedios
1 - resultados intermedios

default tol: ode23 -> 1.0e-03
ode45 -> 1.oe-06
```

5.1.3 Declaración function

Sintaxis:

```
function nombre_1=nombre_2(parametro_1, ..., parametro_n)
```

Ejemplos:

```
function y=promedio(x)
function i=inodal(t,v)
function xpunto=vdpol(t,x)
xpunto=zeros(2,1);
xpunto(1)=x(1).*(1-x(2).^2)-x(2);
xpunto(2)=x(1);
```

5.2 Operadores relacionales

Los operadores relacionales de MatLab son:

```
<  menor que
<=  menor o igual a
>  mayor que
>=  mayor o igual a
==  igual a
=~  no igual a
```

Ejemplo:

```
if n< maxn
...
if n>=0, break, end
```

5.3 Operadores lógicos

Los operadores &, | y ~ son los operadores de lógica "y", "ó" y "no" respectivamente.

El resultado de C = A & B es una matriz cuyos elementos son unos donde A y B sean ambos distintos de cero, y ceros donde A ó B sean cero. A y B deben de ser matrices con las mismas dimensiones, a menos que una de ellas sea un escalar.

El resultado de C = A | B es una matriz cuyos elementos son unos donde A ó B tienen un elemento diferentede cero, y ceros donde ambas tienen elementos cero. A y B deben de ser matrices con las mismas dimensiones, a menos que una sea un escalar.

El resultado de B = ~A es una matriz cuyos elementos son uno donde A tiene un elemento cero, y ceros donde A tiene elementos diferentes de cero.

Funciónes **any, all**

La función any(x) devuelve 1 si cualquiera de los elementos de x es diferente de cero, de lo contrario devuelve 0.

La función all(x) devuelve 1 solamente si todos los elementos de x son diferentes de cero.

Estas funciones se usan en cláusulas if. Por ejemplo:

```
if all(A <.5)
. . .
end
```

Para argumentos matriciales, any y all trabajan por columnas para devolver un vector fila con el resultado para cada columna. Aplicando la función dos veces, any(any(A)), siempre reduce la matriz a una condición escalar.

Las funciones relacionales y lógicas en MATLAB son:

any - condiciones lógicas

all - condiciones lógicas

find - halla índices de arreglos de valores lógicos

exist - verifica si existen variables

isinf - detecta infinitos

finite - verifica para los valores finitos

5.4 Caracteres especiales

Los caracteres especiales de MatLab son:

[ ] Se utilizan para formar vectores y matrices

( ) Define precedencia en expresiones aritméticas. Encierra argumentos de funciones en forma usual

, Separador de elementos de una matriz, argumentos de funciones y declaraciones en líneas con declaraciones múltiples

; Termina filas de una matriz, separador de declaraciones

% Comentario

Ejemplos:

```
[6.0 9.0 3.4 ]
sqrt(2)
for i=1:n, a(i)=0, end
for i=1:n; a(i)=0; end
% inicia vector a en 0
```

5.5 Control de flujo

5.5.1 Declaración FOR simple

Sintaxis

```
for variable=incio:paso:final
declaración 1;
...
declaración n;
end

for variable=inicio:final
declaración 1;
...
declaración n;
end
```

Ejemplo:

```
for i=1:n
c(i)=a(i)*b(i);
end
```

o

```
for i=1:n; c(i)=a(i)*b(i); end
```

El ciclo FOR permite que una instrucción, ó grupo de instrucciones, pueda repetirse un número determinado de veces. Por ejemplo,

```
for i = 1:n, x(i) = 0, end
```

asigna 0 a los primeros n elementos de x. Si n es menor de 1, el ciclo sigue siendo válido pero MATLAB no ejecuta la instrucción intermedia. Si x no esta definido, ó si tiene menos de n elementos, entonces un espacio adicional es localizado automáticamente a x cada vez que sea necesario.

5.5.2 Declaración FOR anidada.

Sintaxis

```
for variable 1 = inicio1:paso1:fin1
for variable2 = inicio2:paso2:fin2
declaración 1;
...
declaración n;
end
end
```

Ejemplo

```
y=1
for t1=0:0.1:1
for t2=1:-0.1:0
y(1)=sin(t1*t2)
end
i=i+1;
end
```

Ejemplo

```
for i = 1:m
for j = 1:n
A(i, j) = 1/(i+j-1);
end
end
A
```

La "A" al terminar el ciclo muestra en la pantalla el resultado final. Es importante que para cada for halla un end.

5.5.3 Declaración WHILE

Sintaxis:

```
while expresion
proposición 1;
...
proposición 2;
end
```

Ejemplos

```
e=1.0;
while (1.0+e)>1.0001
e=e/2.0;
end
```

```
it=1; t=0; wo=2.0*pi*60.0;
while it<=npts, ut=sin(wo*t);t=t+dt;end
```

El ciclo WHILE permite a una instrucción, ó grupo de instrucciones, repetirse un número indefinido de veces, bajo el control de una condición lógica. El siguiente ciclo while halla el primer entero n para el cual n! es un número de 100 digitos:

```
n = 1;
while prod(1:n) < 1.0e100, n = n+1; end
n
```

Un cálculo más práctico ilustrando el ciclo while es en el cómputo del exponencial de una matriz, llamado `expm(A)` en MATLAB. Una posible definición de la función exponencial es mediante la serie:

```
expm(A) = I + A + A^2/2! + A^3/3! + ...
```

La idea es sumar todos los términos necesarios hasta producir un resultado que, en la precisión finita la de computadora, no cambie aunque más términos sean añadidos. Para esto procedemos de la forma siguiente:

```
E = zeros(size(A));
F = eye(size(A));
k = 1;
while norm(E+F-E, 1) > 0
E = E + F;
F = A*F/k
k = k+1;
end
```

Aqui A es la matriz dada, E representa la suma parcial de la serie, F es un término individual en la serie, y k es el índice de este término.

5.5.4 Declaraciones IF, ELSE, ELSEIF y BREAK

Sintaxis

```
a) if expresión
proposición 1;
...
proposición n;
end

b) if expresión
proposición 1;
...
proposición n;
else
proposición 1;
...
proposición m;
end
```

c) if expresión
proposición 1;
...
proposición n;
elseif
proposición 1;
...
proposición m;
else
proposición 1;
...
proposición r;
end

d) if expresión, break, end

Ejemplos

```
if dv(i) > maxer
maxer=dv(i);
nmaxe=i;
end


sum=0.0; y=1;
while i<=so
n=input(`Introduzca n, interrumpe con valor negativo `);
if n<0, break, end;
if n==0
sum=sum+n;
elseif n<=10
sum=sum+n/2;
else
sum=sum+n/10;
end
end
```

A continuación se muestra como un cálculo se puede dividir en tres casos, dependiendo del signo ó paridad de un entero n:

```
if n < 0
A = negative(n)
else if rem(n, 2) == 0
A = even(n)
else
A = odd(n)
end
```

En el segundo, partiendo de un entero positivo n, si este es par, se divide entre dos; si es impar, se multiplica por tres y se le suma uno. ¿Habrá algún entero para el cual el proceso nunca termine? Aquí se ilustran los enunciados while y if, también se muestra la función input (en este caso es una entrada del teclado), y el enunciado break, que provee salidas abruptas de los ciclos. Veamos:

```
% Problema "3n+1" clásico de la teoria de números.
while 1
n = input('Entre n, negativo termina. ');
if n <= 0, break, end
while n > 1
if rem(n, 2) == 0
n = n/2
else
n = 3*n+1
end
end
end
```

5.6 Algebra Matricial

5.6.1 Creación de una matriz

Ejemplo

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

5.6.2 Cambio del orden de una matriz: **reshape**

Sintaxis:

```
matriz_modificada = reshape(matriz_original, filas, columnas)
```

Ejemplo

```
>> A=[1 4 7 10; 2 5 8 11; 3 6 9 12]
A =
1 4 7 10
2 5 8 11
3 6 9 12

>> B=reshape(A,2,6)
B =
1 3 5 7 9 11
2 4 6 8 10 12
```

5.6.3 Modificación individual de elementos

Ejemplos

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

>> A(1,1)=A(1,2)+A(2,1)
A =
5 2
3 4
```

```
>> A(1,2)=A(2,1)
A =
5 3
3 4

>> A(2,2)=10
A =
5 3
3 10
```

5.6.4 Modificaciones adicionales de una matriz

Ejemplo

```
>> A=[1 2; 3 4; 5 6 ]
A =
1 2
3 4
5 6
```

Conversión de una matriz en un vector

```
>> A=[1 2; 3 4; 5 6 ]
A =
1 2
3 4
5 6

>> b=A(:)
b =
1
3
5
2
4
6
```

Modificación de los elementos

```
>> A(:)=10:15
A =
10 13
11 14
12 15
```

Generación de vectores:

Ejemplos

```
>> x=1:5
x =
1 2 3 4 5

>> x=5:-1:1
x =
5 4 3 2 1

>> x=0:0.25:1
x =
0 0.2500 0.5000 0.7500 1.0000
```

Acceso a submatrices contiguas y no contiguas

Ejemplos

Si la matriz original A es de 10*10, entonces:

A(1:3,5) matriz de 3x1 que tiene los tres primeros elementos de la columna 5 de A

A(1:3, 5:9) matriz de 3x4 que tiene los tres primeros filas y las columnas de 5 a 9 de A

A(:,5) quinta columna de A

A(1:5,:) primeras cinco filas de A

A(:,[4 6])=B(:,1:2) remplaza la cuarta y sexta columnas de A con las dos primeras de A

Matrices vacias

La declaración

x = [ ]

asigna una matriz de dimensión 0x0 a x

Para la matriz A considerada previamente

A(:,[3,5])=[ ] borra columnas 3 y 5 de A

A([3,5],:)=[ ] borra filas 3 y 5 de A

Declaración de matrices complejas

A=[1 2; 3 4] + i*[5 6 ; 7 8]

o

A=[1 2; 3 4] + i*[5 6 ; 7 8]

o

A=[1+5i 2+6i; 3+7i 4+8i]

A =

1.0000 + 5.0000i 2.0000 + 6.0000i

3.0000 + 7.0000i 4.0000 + 8.0000i

Generación de tablas

```
>> x=(0.0:0.2:3.0);
>> y=exp(-x).*sin(x);
>> [x;y]
ans =
Columns 1 through 7
0 0.2000 0.4000 0.6000 0.8000 1.0000 1.2000
0 0.1627 0.2610 0.3099 0.3223 0.3096 0.2807
Columns 8 through 14
1.4000 1.6000 1.8000 2.0000 2.2000 2.4000 2.6000
0.2430 0.2018 0.1610 0.1231 0.0896 0.0613 0.0383
Columns 15 through 16
2.8000 3.0000
0.0204 0.0070
```

Determinante de A: **det(A)**

```
>> A=[1 2 3;4 5 6; 7 8 9]
A =
1 2 3
4 5 6
7 8 9
>> det(A)
ans =0
```

Diagonal de A: **diag(A)**

```
>> diag(A)
ans =
1
5
9
```

Valores y vectores característicos: **eig(A)**

```
>> A=[0 7 -6; 1 0 0;0 1 0]
A =
0 7 -6
1 0 0
0 1 0

>> eig(A)
ans =
-3.0000
2.0000
1.0000
```

v - Vectores característicos

d - valores característicos

```
>> [v d]=eig(A)
v =
0.9435 -0.8729 0.5774
-0.3145 -0.4364 0.5774
0.1048 -0.2182 0.5774
d =
-3.0000 0 0
0 2.0000 0
0 0 1.0000
```

- Exponencial de una matriz**: expm(A)**

```
>> A=[0 7 -6; 1 0 0;0 1 0]
A =
0 7 -6
1 0 0
0 1 0

>> expm(A)
ans =
5.2541 11.0757 -13.6115
2.2686 5.2541 -4.8044
0.8007 2.2686 -0.3510
```

- Factorización LU de A: **lu(A)**

```
>> [L U]=lu(A)
L =
0 1.0000 0
1.0000 0 0
0 0.1429 1.0000
U =
1.0000 0 0
0 7.0000 -6.0000
0 0 0.8571
```

- Inversa de A: **inv(A)**

```
>> inv(A)
ans =
0 1.0000 0
0 0 1.0000
-0.1667 0 1.1667
```

- Ecuación característica de la matriz A: **poly(A)**

```
>> p=poly(A)
p =
1.0000 0.0000 -7.0000 6.0000
```

- Raices de la ecuación característica: **roots(p)**

```
>> r=roots(p)
r =
-3.0000
2.0000
1.0000
```

5.7 Archivos de E/S

5.7.1 Declaración **fopen**

Sintaxis

id = fopen(`nombre.dat', `permiso')

donde permiso puede ser:

`r' Abre archivo para lectura

`r+ Abre archivo para lectura y escritura

`w' Borra el contenido del archivo existente o crea un nuevo archivo y lo abre para escritura

`w+' Idem que `w' únicamente que el archivo se abre para lectura y escritura

`a' Crea y abre un nuevo archivo o abre un archivo

`a+' Idem que `a' únicamente que el archivo es abierto para lectura y escritura

Ejemplo

fid = fopen(`archivo.dat','r')

fid = -1, error

0, lectura/escritura normal

[fid, mensaje = fopen(`archivo.dat','r')

5.7.2 Declaración **fclose**

Sintaxis

status = fclose(fid)

o

status = fclose (`all') - cierra todos los archivos abiertos

5.7.3 Declaración **fread**

Lee un archivo abierto con una precisión indicada

Sintaxis

fread(fid,registros,'precision')

registros

`char' o `uchar'

`short' o `long'

`float' o `double'

Ejemplo:

```
A = fread(fid,10,'float')
```

5.7.4 Declaración **fwrite**

Sintaxis

```
fwrite(fid,A,'short')
```

5.7.5 Declaración **fprintf**

Salida con formato

Ejemplos:

```
fprintf(fid,'titulo\n');
fprintf(fid,'%f %12.7f\n', y);
```

Formato

%s - cadena decimal

%d - número decimal

%f - punto flotante

% g - formato g

5.8 Variables globales

Son variables, de las cuales una sola copia es compartida por el programa principal y sus funciones.

Sintaxis:

```
global variable1, ..., variable_N
```

Ejemplo

```
function x=ccdifs(t,x)
global ka,kb
xp=[x(1)-ka*x(1)*x(2); -x(2)+kb*x(1)*x(2)];
...
global ka,kb
ka=0.01
kb=0.02
[t,x]=ode23('ccdifs',0,10,[1:1]);
```

5.9 Vectorización de algoritmos y estructuras (for, while)

Para que los programas en MATLAB ejecuten más rápido, debemos vectorizar estos siempre que sea posible. Esto es, debemos convertir los ciclos for y while a operaciones de vectores ó de matrices. Por ejemplo, un modo de calcular la función "sin" para 1001 números entre 1 y 10 es:

```
i = 0;
for t = 0:.01:10
i = i + 1;
y(i) = sin(t);
end
```

Una versión vectorizada del mismo código es

```
t = 0:.01:10; y = sin(t);
```

En una computadora lenta, el primer ejemplo tomó 15 segundos, mientras que el segundo tomó 0.6 segundos.

Vectores Pre-Asignados

Si no podemos vectorizar un pedazo de código, podemos hacer que los ciclos for vayan más rápido pre-asignando cualquier vector en el cual el resultado de salida sea guardado. Veamos un ejemplo:

```
y = zeros (1,100);
for i = 1:100
y(i) = det(X^i);
end
```

Si no pre-asignamos el vector "y", el interpretador de MATLAB irá aumentando el tamaño de "y" por uno cada vez que se itera en el ciclo.
Permite incrementar la velocidad de proceso de MATLAB

Sintaxis

variable=inicio:incremento:final

Ejemplo

```
i=1, wo=2*pi*fo;
for t=0:dt:per
f(i)=sin(wo*t);
i=i+1;
end
```

ó

```
t=0:dt:per;
fi=sin(wo*t);
```

5.10 Gráficas en Dos Dimensiones

5.10.1 Funciones elementales para graficar

plot - crea una gráfica de vectores ó columnas de matrices.

loglog - crea una gráfica utilizando una escala logarítmica para ambos ejes.

semilogx - crea una gráfica utilizando una escala logarítmica para el eje-x y una escala lineal para el eje-y.

semilogy - crea una gráfica utilizando una escala logarítmica para el eje-y y una escala lineal para el eje-x.

Puedes añadir títulos, encabezamientos de ejes, líneas entre cortadas y texto a tus gráficas utilizando:

tittle - añade título a la gráfica

xlabel - añade encabezamiento al eje-x

ylabel - añade encabezamiento al eje-y

text - añade una cadena de texto en una localización específica

gtext - añade texto a la gráfica utilizando el ratón

grid - crea líneas entrecortadas

5.10.2 Creando una gráfica

Comando Plot

Sintaxis:

a) plot(y)

b) plot(x,y)

c) plot(x,y,'tipo_línea')

d) plot(x1,y1,'tipo_línea_1',x2,y2,'tipo_línea_2', ... , xn,yn,'tipo_línea_n')

Si y es un vector, plot(y) produce una gráfica lineal de los elementos de y versus el índice de estos. Si especifica dos vectores como argumentos, plot(x, y) produce una gráfica de y versus x.

Símbolo Color

y amarillo

m magenta

c cyan (azul claro)

r rojo

g verde

b azul

w blanco

k negro

Símbolo Estilo de línea

. punto

o circulo

x marca

+ mas

* asterisco

- sólido

: punteado

-. segmento punto

-- segmento

Ejemplo

```
t=0:pi/200:2*pi;x=sin(t);y1=sin(t+0.5);y2=sin(t+1.0);
plot(x,y1,'r-',x,y2,'g--'); title('Angulo difuso'); xlabel('x=sin(t)'); ...
ylabel('y=sin(t+)')
```

5.10.3 Graficando Matrices

plot(Y) - dibuja una línea para cada columna de Y. El eje-x es encabezado por el vector índice de fila, 1:m, donde m es el número de filas en Y.

Si plot es usado con dos argumentos y si X ó Y tienen más de una fila ó columna, entonces:

si Y es una matriz, y x es un vector, plot(x,Y) grafica las filas ó columnas de Y versus el vector x;

si X es una matriz y y es un vector, plot(X,y) grafica cada fila ó columna de X versus el vector y;

si X y Y son ambas matrices del mismo tamaño, plot(X, Y) grafica las columnas de X versus las columnas de Y.

También puedes usar la función plot con múltiples pares de argumentos matriciales:

plot (X1, Y1, X2, Y2, ...)

Cada par X-Y es graficado, generando líneas múltiples. Los pares diferentes pueden ser de dimensiones diferentes.

5.10.4 Importando Datos

Puede importar y graficar datos generados fuera de MATLAB utilizando el comando load.

5.10.5 Graficando Funciones Matemáticas

Hay diferentes formas de graficar funciones y = f(x). Una de estas formas es evaluar la función en miles de puntos en el intervalo de interés. La siguiente función oscila infinitamente rápido en el intervalo, 0 x 1.

Podemos gráficarla como sigue:

```
x = (0:1/2000:1)';
plot(x, cos(tan(pi*x)))
```

Para hacer esto más eficiente podemos usar la función fplot la cual concentra su evaluación sobre las regiones donde la rapidez de cambio de la función es más grande.

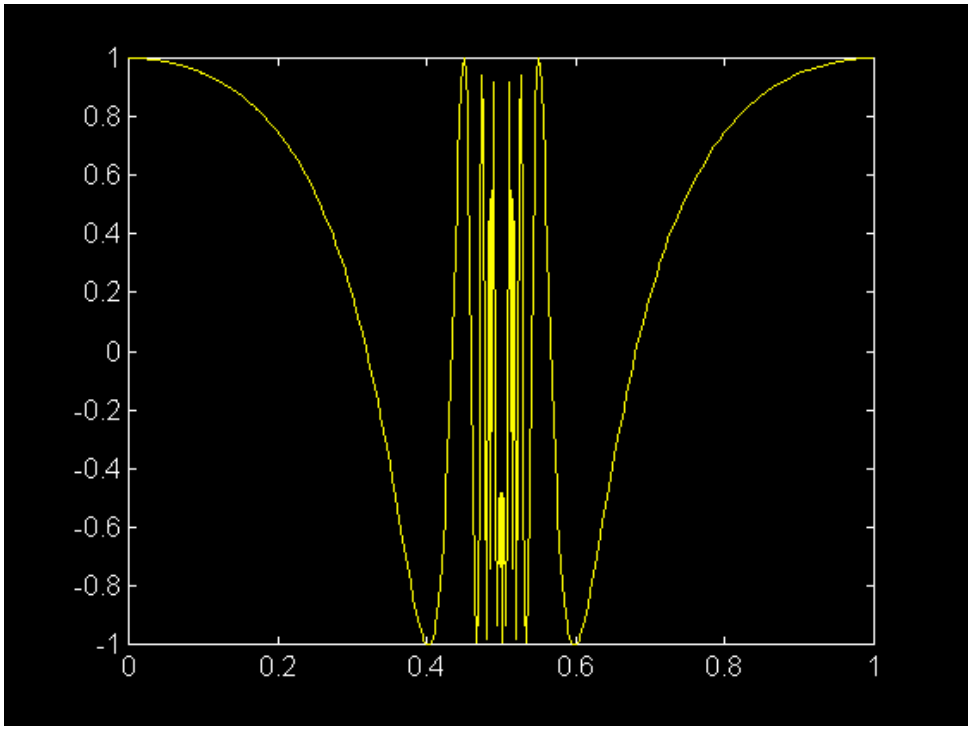
Para evaluar una función, se crea un archivo de esta función y se le pasa el nombre del archivo a fplot. El siguiente archivo-M de tipo función define la función anterior como fofx.

```
function y = fofx(x)
y = cos(tan(pi*x));
```

Este archivo se guarda con el nombre de fofx.m. Ahora la instrucción

```
fplot('fofx', [0 1])
```

produce la gráfica:



Aquí, fplot usa menos puntos para evaluar la misma función a intervalos más cerrados en la región donde la rapidez de cambio es mayor.

5.10.6 Comandos gráficos

hold Permite añadir líneas al dibujo previo

on Activa hold

off Desactiva hold

Ejemplo

plot(x); hold on; plot(y',':');plot(yz,'-.')

loglog

Sintaxis

a) loglog(x,y)

b) loglog(x,y,'tipo_línea')

c)loglog(x1',y1','tipo_línea_1',...,

xn,yn,'tipo_línea_n')

Ejemplo

x=logspace(-1,3); loglog(x,exp(x))

donde logspace tiene las formas:

logspace(a,b)

logspace(a,b,n)

a,b exponentes de los límites. Es decir, 10^a y 10^b

semilog(x), semilog(y)

Sintaxis

a) semilogx(x,y)

b) semilogy(x,y)

Ejemplo

x=0:.1:20; semilogy(x,10.^x)

fill

Dibuja el area interior de una curva en determinado color

Sintaxis:

a) fill(x,y,'c')

b) fill(x1,y1,'c1',...,xn,yn,cn)

Ejemplo

```
t=0:0.5:2*pi; x=sin(t); fill(t,x,'b')
t=0:0.5:2*pi; x=sin(t); y=cos(t); fill(y,x,'r')
```

subplot

Dibuja la pantalla en mxn subdivisioens, numeradas por el parámetro p, de izquierda a derecha, iniciando por la fila superior

Sintaxis:

subplot(m,n,p)

Ejemplo:

```
vt=smvars(:,1);
it=smvars(:,2);
rang=smvars(:,3);
ikd=smvars(:,4);
subplot(2,2,1);
plot(vt)
subplot(2,2,2)
plot(it)
suplot(2,2,3)
plot(rang)
subplot(2,2,4)
plot(ikd)
```

bar

Crea una gráfica de barras

Sintaxis:

a) bar(y);

b) bar(x,y);

c) [xb,yb]=bar(y); => plot(xb,yb)

d) [xb,yb]=bar(x,y); => plot(xb,yb)

Ejemplo

```
x=-2.8:0.2:2.8
bar(x,exp(-x.*x)
```

Nota: Los valores de x deben estar igualmente espaciados

stairs

Igual que bar, únicamente sin líneas internas

fplot

Dibuja la gráfica de una función

Sintaxis:

a) fplot(`función', [inicio,final])

b) fplot(`función', [inicio,final],n)

c) fplot(`función', [inicio,final],n,ángulo)

d) [x,y]=fplot(`función', [inicio,final]) => plot(x,y)

n - número de puntos

ángulo - ángulo entre segmentos sucesivos de la función

Ejemplo

```
fplot(`sin',[0,pi])
fplot(`tanh',[-2 2])
function y=func(x)
y=200*sin(x(:))./x(:);
fplot(`func',[-30 30],60,2)
```

polar

Dibujo en coordenadas polares

Sintaxis:

a) polar(ángulo,radio)

b) polar(ángulo, radio, `tipo_línea')

Ejemplo

```
t=0:0.01:2*pi;
polar(t,sin(5*t))
```

colormap

Colorea con sombreado el interior de una curva o polígono

Sintaxis

colormap(colorbase)

donde colorbase es:

gray

hot

cool

copper

pink

Ejemplo

t=0:0.05:2*pi; x=sin(t); y=cos(t); colormap(hot(130)); ...

Nota: 130 es opcional el rango 0-255

fill(y,x,x) => sombreado horizontal

fill(y,x,y) => sombreado vertical

5.11 Gráficos en 3 dimensiones

plot3

Dibuja líneas y puntos en 3 dimensiones

Sintaxis:

a) plot3(x,y,z)

b) plot3(x,y,z)

c) plot3(x,y,z,'tipo_línea)

d) plot3(x1,y1,z1,'tipo_línea',...,xn,yn,zn,'tipo_línea')

Ejemplo

t=0:0.05:10*pi; plot3(sin(t),cos(t),t)

contour, contour3

Genera dibujos compuestos de líneas de valores de datos constantes obtenidos de una matriz de entrada

Sintaxis:

a) contour(z)

b) contour(z,n)

c) contour(x,y,z)

d) contour(x,y,z,n)

Ejemplo

contour(peeks)

contour(peeks,30)

contour3

Igual función de contour en 3 dimensiones

Sintaxis:

a) contour3(z)

b) contour3(z,n)

c) contour3(x,y,z)

d) contour3(x,y,z,n)

Ejemplo

contour3(peaks,30)

meshgrid

Genera arreglos X y Y para dibujos en 3 dimensiones

Sintaxis:

a) [X,Y] = meshgrid(x,y)

b) [X,Y] = meshgrid(x) => meshgrid(x,y)

Ejemplo. Evalue y dibuje la funcion z=x*exp(-x^2-y^2) sobre el rango -2<=x<=2, -2<=y<=2

```
[X,Y]=meshgrid(-2:2:2); z=x.*exp(-x^2-y^2); ...
mesh(Z)
```

```
x=-8:0.5,8; y=x; [x,y]=meshgrid(x,y);...
R=sqrt(x.^2+y.^2)+0.001; z=sin(R)./R;...
mesh(z)
```

mesh, meshc y meshz

Dibujan una superficie de malla tridimensional, crando una perspectiva del dibujo, sobre y bajo el plano de referencia.

Sintaxis:

a) mesh(x,y,z,c)

b) mesh(x,y,z)

c) mesh(x,y,z,c)

d) mesh(x,y,z)

e) mesh(z,c)

f) mesh(z)

g) meshc(...) => mismo que mesh

h) meshc(...) => mismo que mesh

meshc

Añade un plano de contorno debajo del dibujo

meshz

Añade un plano de referencia o cortina al dibujo

Ejemplo:

```
[x,y] = meshgrid(-3:2:3); z=peaks(x,y); meshc(x,y,z)
[x,y] = meshgrid(-3:2:3); z=peaks(x,y); meshz(x,y,z)
```

surf, surfc

Crean superficies sombreadas en 3 dimensiones

Sintaxis:

a) surf(x,y,z,c)

b) surf(x,y,z)

c) surf(x,y,z,c)

d) surf(x,y,z)

e) surf(z,c)

f) surf(z)

g) surfc(...) => misma Sintaxis que surf

Ejemplo

```
[x,y]=meshgrid(-3:.2:3); z=peaks(x,y); surf(x,y,z)
k=5; n=2^k-1; [x,y,z]=sphere(n); c=hadamard(2^k);...
surf(x,y,z,c); colormap(hot)
```

hadamard

Matriz hadamard compuesta de 1's y -1's, empleada en procesamiento de señales y análisis numérico

Ejemplo (matriz de 4*4)

```
1 1 1 1
1 -1 1 -1
1 1 -1 -1
1 -1 -1 1
```

sphere

Genera una esfera

Sintaxis

[x,y,z] = sphere(n)

n - número de meridianos

Ejemplo

```
[x,y,z]=sphere(20);
mesh(x,y,z)
```

o con :

```
colormap(hot)
```

surfl

Superficie sombreada tridimensioanl con efecto de reflexión de luz

Sintaxis:

a) surfl(z)

b) surfl(z,s)

c) surfl(x,y,z)

d) surfl(x,y,z,s)

s - dirección de la luz

Ejemplo

```
[x,y]=meshgrid(-3:0.01:3);
z=peaks(x,y);
surfl(x,y,z)
```

shading

Establece las propiedades de sombreado con color

Sintaxis

```
shading faceted
shading interp
shading flat
```

shading flat - cada segmento de la superficie tiene un valor constante determinado por el color de los puntos extremos del segmento o sus esquinas

shading interp - el color en cada segmento varia linealmente e interpolo los valores extremos o esquinas

shading faceted - utiliza sombreado "flat" con líneas de malla negras superpuestas

Para el ejemplo anterior, añadiendo:

```
shading interp
```

y posteriormente:

```
colormap(gray);
```

axis

Escala y apariencia de los ejes

Sintaxis:

```
a) axis([xmin, xmax, ymin, ymax])
b) axis([xmin, xmax, ymin, ymax, zmin, zmax])
c) axis(`auto')
d) axis(`ij')
e) axis(`xy')
f) axis(`square')
g) axis(`equal')
h) axis(`off')
```

i) axis(`on')

donde:

axis(`auto') realiza el escalamiento de ejes a su modo de autoescalamiento por defecto.

axis(`ij') dibuja nuevamente la gráfica.

El eje y es vertical y es numerado de arriba hacia abajo.

El eje j es horizontal y es numerado de izquierda a derecha.

axis(`xy') regresa la forma de ejes cartesianos que existe por defecto. El eje x es horizontal y se numera de izquierda a derecha. El eje y es vertical y se numera de abajo hacia arriba

axis(`square') determina que la región de los ejes es cuadrada

axis(`equal') indica que los factores de escalamiento y marcas incrementales a lo largo de los ejes x y y son iguales.

axis(`off') desactiva las etiquetas de los ejes y las marcas

axis(`on') activa las etiquetas de los ejes y las marcas

Para el ejemplo último:

...

```
axis([-3 3 -3 3 -8 8])
```

fill3 colorea polígonos de 3 dimensiones

a) fill3(x,y,z,'c')

b) fill3(x1,y1,z1,...,xn,yn,zn)

Ejemplo

```
colormap(hot)
fill3(rand(3,4), rand(3,4), rand(3,4), rand(3,4))
```

rand matrices y números aleatorios distribuidos uniformemente

Sintaxis:

a) rand(n) - matriz de nxn

b) rnad(m,n) - matriz de mxn

Ejemplo

```
fill3(rand(20), rand(20), rand(20), rand(20))
```

load carga en el area de trabajo un archivo (imagen, sonido, datos, etc)

Sintaxis

a) load archivo

b) load archivo.ext

donde:

ext - extensión

Ejemplo

```
load clown
```

image crea un objeto imagen y lo presenta

Sintaxis:

a) image(x)

b) image(x,y,x)

c) presenta la matriz c como una imagen

d) especifica los límites de los datos de la imagen en los ejes x e y. En b), x e y son vectores

Ejemplo

```
load clown
colormap(map)
image(x)
```

brighten hace más brillante o más obscura la imagen

Sintaxis:

a) brighten(alfa)

b) brighten(map,alfa)

donde:

0<alfa<1 más brillante

-1<alfa<0 más obscuro

Del ejemplo anterior:

```
...
brighten(0.6)
ó brighten(-0.6)
```

clf borra la figura

sound convierte un vector en sonido (en computadoras sparc y macintosh)

Sintaxis

a) sound(y)

b) sound(y,Fs)

donde:

Fs frecuencia especificada en Hz

Ejemplo

```
load train
sound(y,Fs)
t=(0:length(y)-1)/Fs;
plot(t,y)
```

5.12 Archivos de disco

5.12.1 Manipulación de Archivos de Disco

Algunos comandos utilizados para la manipulación de archivos de disco son dir, type, delete y cd. Si la extención no se especifica, MATLAB utiliza .m automáticamente. El comando diary crea un diario de tu sesión de MATLAB en un archivo de disco. Para más información utiliza la Guía de Referencia de MATLAB ó el comando help.

5.12.2 Ejecutando Programas Externos

El simbolo "!" le indica a MATLAB que el resto de la línea de entrada es un comando para el sistema operativo. Por ejemplo,

```
! edt darwin.m
```

invoca un editor llamado edt en un archivo llamado darwin.m. Luego que este programa sea completado, el sistema operativo devuelve el control a MATLAB.

5.12.3 Importando y Exportando Datos

Puedes introducir datos de otros programas a MATLAB por varios métodos. Similarmente, puedes exportar datos de MATLAB a otros programas. También puedes hacer que tus programas manipulen datos directamente en archivos-MAT,

el cúal es el formato de archivo utilizado por MATLAB. Para información acerca de las técnicas utilizadas para importar y exportar datos consulte la sección de Importando y Exportando Datos de la guía de MATLAB ó utilice al comando help de MATLAB.

5.13 INDICE ALFABETICO

axis, all, any, bar, break, brighten, clf, for, colormap, conj, contour, contour3, det, diag, else, elseif, eig, expm, fclose, fill, fill3, fft, fftn, function, fopen, fplot, fread, fwrite, grid, gtext, hadamard, hold, if, ifft, ifftn, imag, image, inv, loglog, load, lu, mesh, meshc, meshz, meshgrid, ode23, ode45, peaks, plot, plot3, poly, reshape, rot90, sphere, tril, triu, pascal, polar, real, toplitz, rand, reshape, semilog, semilogy, shading (flat interp faceted), size, sound, stairs, subplot, surf, surfc, surfl, text, title, xlabel, ylabel, while, zeros.

6. SIMULINK

Simulink es una herramienta para el modelaje, análisis y simulación de una amplia variedad de sistemas físicos y matemáticos, inclusive aquellos con elementos no lineales y aquellos que hacen uso de tiempos continuos y discretos. Como una extensión de MatLab, Simulink adiciona muchas características específicas a los sistemas dinámicos, mientras conserva toda la funcionalidad de propósito general de MatLab. Así Simulink no es completamente un programa separado de MatLab, sino un anexo a él. El ambiente de MatLab está siempre disponible mientras se ejecuta una simulación en Simulink.

Simulink tiene dos fases de uso: la definición del modelo y el análisis del modelo. La definición del modelo significa construir el modelo a partir de elementos básicos construidos previamente, tal como, integradores, bloques de ganancia o servomotores. El análisis del modelo significa realizar la simulación, linealización y determinar el punto de equilibrio de un modelo previamente definido.

Para simplificar la definición del modelo Simulink usa diferentes clases de ventanas llamadas ventanas de diagramas de bloques. En estas ventanas se puede crear y editar un modelo gráficamente usando el ratón. Simulink usa un ambiente gráfico lo que hace sencillo la creación de los modelos de sistemas.

Después de definir un modelo este puede ser analizado seleccionando una opción desde los menús de Simulink o entrando comandos desde la línea de comandos de MatLab.

Simulink puede simular cualquier sistema que pueda ser definido por ecuaciones diferenciales continuas y ecuaciones diferenciales discretas. Esto significa que se puede modelar sistemas continuos en el tiempo, discretos en el tiempo o sistemas híbridos.

Simulink usa diagramas de bloques para representar sistemas dinámicos. Mediante una interface gráfica con el usuario se pueden arrastrar los componentes desde una librería de bloques existentes y luego interconectarlos mediante conectores y alambre. La ventana principal de Simulink se activa escribiendo simulink en la línea de comandos de MatLab, y se muestra a continuación:

Haciendo doble click en cualquiera de las librerías presentes en esta ventana se abrirá otra ventana conteniendo una cantidad de bloques relativos a dicha librería.

Para realizar un sistema debe abrirse una nueva ventana de diagrama de bloques seleccionando la opción file del menú principal del Simulink y allí la opción new. En esta nueva ventana se colocarán todos los bloques interconectados que formarán el sistema deseado.

Como ejemplo se ha tomado un generador de ondas seno de la librería de fuentes "sources" y un osciloscopio de la librería "sinks", ambos se unieron mediante un conector usando el ratón. Este sistema se almacena como un archivo-m.

Haciendo doble click sobre cada elemento del sistema se pueden ver y modificar sus características.

Por ejemplo, al generador seno se le puede modificar su amplitud, frecuencia y fase. Al osciloscopio se le definen las escalas horizontal y vertical.

Para ejecutar el programa se usa la opción simulation en el menú de la ventana del archivo-m creado.

En este submenú está la opción start que permite ejecutar el programa. También está la opción parameters que activa el panel de control de Simulink en donde se definen los métodos y parámetros usados para la simulación, tal como se muestra a continuación:

Al ejecutar el programa seno.m creado mediante simulink, se puede observar la respuesta al hacer doble click en el osciloscopio.

Existen numerosos bloques y funciones incorporados en las librerías de simulink que pueden ser empleados para simular cualquier sistema.

Por ejemplo, para implementar un sistema que emplea un controlador PID tenemos:

En este diagrama se tiene al bloque llamado PID que fue definido previamente y agrupado como uno solo. El contenido de dicho bloque se obtiene haciendo doble click sobre él. A continuación se muestra el bloque PID:

6.1 Acelerador de Simulink

Para incrementar la velocidad de Simulink se debe instalar el acelerador "Accelerator". Este permite automáticamente generar una versión mejorada de los modelos los cuales correrán diez veces más rápido que el original. El acelerador puede ser usado sobre modelos continuos, discretos en el tiempo y híbridos.

El acelerador trabaja generando y compilando un código-C para un modelo dado. Una vez se completa la compilación, la simulación es ejecutada en la ventada de modelos de Simulink exactamente igual que antes sólo que más rápidamente. El propósito del acelerador es aumentar la velocidad de simulación.

Si el programa MatLab posee instalado el "Accelerator" podrá iniciarse la acción aceleradora seleccionando la opción simulation en el menú principal del Simulink y dentro de esta seleccionando la opción Accelerate. Esta acción es totalmente transparente en el sentido de que el incremento de la velocidad se presenta sin ningún otro requerimiento por parte del usuario.

6.2 Generador de código-C en Simulink

Una vez se ha creado un modelo dinámico en Simulink, se puede invocar el generador de código-C que permite convertir el diagrama de bloques implementado en un código C. Este puede ser útil para varios propósitos: puede ser usado para control en tiempo real, simulación en tiempo real o simulación acelerada en tiempo no real. Sus aplicaciones pueden ser control de movimiento, control de procesos, sistemas automotores, equipos médicos, robótica, etc.

El código-C es diseñado tal que puede ser ejecutado en tiempo real. No requiere ser escrito manualmente por un programador pues es creado a nivel de diagramas de bloques en Simulink. El código generado puede correr sobre un amplio rango de hardware ubicado en estaciones de trabajo, PC o microprocesadores. Este código es la forma en la que puede usare el Simulink para adquisición de datos.

7. COMANDOS DE MATLAB

7.1 General purpose commands:

Managing commands and functions:

help - On-line documentation.

what - Directory listing of M-, MAT- and MEX-files.

type - List M-file.

lookfor - Keyword search through the HELP entries.

which - Locate functions and files.

demo - Run demos.

path - Control MATLAB's search path.

Managing variables and the workspace:

who - List current variables.

whos - List current variables, long form.

load - Retrieve variables from disk.

save - Save workspace variables to disk.

clear - Clear variables and functions from memory.

pack - Consolidate workspace memory.

size - Size of matrix.

length - Length of vector.

disp - Display matrix or text.

Working with files and the operating system:

cd - Change current working directory.

dir - Directory listing.

delete - Delete file.

getenv - Get environment value.

! - Execute operating system command.

unix - Execute operating system command & return result.

diary - Save text of MATLAB session.

Controlling the command window:

cedit - Set command line edit/recall facility parameters.

clc - Clear command window.

home - Send cursor home.

format - Set output format.

echo - Echo commands inside script files.

more - Control paged output in command window.

Starting and quitting from MATLAB:

quit - Terminate MATLAB.

startup - M-file executed when MATLAB is invoked.

matlabrc - Master startup M-file.

General information:

info - Information about MATLAB and The MathWorks, Inc.

subscribe - Become subscribing user of MATLAB.

hostid - MATLAB server host identification number.

whatsnew - Information about new features not yet documented.

ver - MATLAB, SIMULINK, and TOOLBOX version information.

Operators and special characters:

Char Name HELP topic

+ Plus arith

- Minus arith

* Matrix multiplication arith

.* Array multiplication arith

^ Matrix power arith

.^ Array power arith

\ Backslash or left division slash

/ Slash or right division slash

./ Array division slash

kron Kronecker tensor product kron

: Colon colon

( ) Parentheses paren

[ ] Brackets paren

. Decimal point punct

.. Parent directory punct

... Continuation punct

, Comma punct

; Semicolon punct

% Comment punct

! Exclamation point punct

' Transpose and quote punct

= Assignment punct

== Equality relop

< > Relational operators relop

& Logical AND relop

| Logical OR relop

~ Logical NOT relop

xor Logical EXCLUSIVE OR xor

Logical characteristics:

exist - Check if variables or functions are defined.

any - True if any element of vector is true.

all - True if all elements of vector are true.

find - Find indices of non-zero elements.

isnan - True for Not-A-Number.

isinf - True for infinite elements.

finite - True for finite elements.

isempty - True for empty matrix.

issparse - True for sparse matrix.

isstr - True for text string.

isglobal - True for global variables.

Control System Toolbox Commands:

***Model building*:**

append - Append system dynamics.

augstate - Augment states as outputs.

blkbuild - Build state-space system from block diagram.

cloop - Close loops of system.

connect - Block diagram modeling.

conv - Convolution of two polynomials.

destim - Form discrete state estimator from gain matrix.

dreg - Form discrete controller/estimator from gain matrices.

drmodel - Generate random discrete model.

estim - Form continuous state estimator from gain matrix.

feedback - Feedback system connection.

ord2 - Generate A,B,C,D for a second-order system.

pade - Pade approximation to time delay.

parallel - Parallel system connection.

reg - Form continuous controller/estimator from gain matrices.

rmodel - Generate random continuous model.

series - Series system connection.

ssdelete - Delete inputs, outputs, or states from model.

ssselect - Select subsystem from larger system.

Model conversions>:

c2d - Continuous to discrete-time conversion.

c2dm - Continuous to discrete-time conversion with method.

c2dt - Continuous to discrete conversion with delay.

d2c - Discrete to continuous-time conversion.

d2cm - Discrete to continuous-time conversion with method.

poly - Roots to polynomial conversion.

residue - Partial fraction expansion.

ss2tf - State-space to transfer function conversion.

ss2zp - State-space to zero-pole conversion.

tf2ss - Transfer function to state-space conversion.

tf2zp - Transfer function to zero-pole conversion.

zp2tf - Zero-pole to transfer function conversion.

zp2ss - Zero-pole to state-space conversion.

***Model reduction*:**

balreal - Balanced realization.

dbalreal - Discrete balanced realization.

dmodred - Discrete model order reduction.

minreal - Minimal realization and pole-zero cancellation.

modred - Model order reduction.

***Model realizations*:**

canon - Canonical form.

ctrbf - Controllability staircase form.

obsvf - Observability staircase form.

ss2ss - Apply similarity transform.

***Model properties*:**

covar - Continuous covariance response to white noise.

ctrb - Controllability matrix.

damp - Damping factors and natural frequencies.

dcgain - Continuous steady state (D.C.) gain.

dcovar - Discrete covariance response to white noise.

ddamp - Discrete damping factors and natural frequencies.

ddcgain - Discrete steady state (D.C.) gain.

dgram - Discrete controllability and observability gramians.

dsort - Sort discrete eigenvalues by magnitude.

eig - Eigenvalues and eigenvectors.

esort - Sort continuous eigenvalues by real part.

gram - Controllability and observability gramians.

obsv - Observability matrix.

printsys - Display system in formatted form.

roots - Polynomial roots.

tzero - Transmission zeros.

tzero2 - Transmission zeros using random perturbation method.

Time response:

dimpulse - Discrete unit sample response.

dinitial - Discrete initial condition response.

dlsim - Discrete simulation to arbitrary inputs.

dstep - Discrete step response.

filter - SISO z-transform simulation.

impulse - Impulse response.

initial - Continuous initial condition response.

lsim - Continuous simulation to arbitrary inputs.

ltitr - Low level time response function.

step - Step response.

stepfun - Step function.

Frequency response:

bode - Bode plot (frequency response).

dbode - Discrete Bode plot (frequency response).

dnichols - Discrete Nichols plot.

dnyquist - Discrete Nyquist plot.

dsigma - Discrete singular value frequency plot.

fbode - Fast Bode plot for continuous systems.

freqs - Laplace-transform frequency response.

freqz - Z-transform frequency response.

ltifr - Low level frequency response function.

margin - Gain and phase margins.

nichols - Nichols plot.

ngrid - Draw grid lines for Nichols plot.

nyquist - Nyquist plot.

sigma - Singular value frequency plot.

Root locus:

pzmap - Pole-zero map.

rlocfind - Interactive root locus gain determination.

rlocus - Evans root-locus.

sgrid - Draw continuous root locus wn,z grid.

zgrid - Draw discrete root locus wn,z grid.

Gain selection:

acker - SISO pole placement.

dlqe - Discrete linear-quadratic estimator design.

dlqew - General discrete linear quadratic estimator design.

dlqr - Discrete linear-quadratic regulator design.

dlqry - Discrete regulator design with weighting on outputs.

lqe - Linear-quadratic estimator design.

lqed - Discrete estimator design from continuous cost function.

lqe2 - Linear quadratic estimator design using Schur method.

lqew - General linear-quadratic estimator design.

lqr - Linear-quadratic regulator design.

lqrd - Discrete regulator design from continuous cost function.

lqry - Regulator design with weighting on outputs.

lqr2 - Linear quadratic regulator design using Schur method.

place - Pole placement.

Equation solution:

are - Algebraic Riccati equation solution.

dlyap - Discrete Lyapunov equation solution.

lyap - Continuous Lyapunov equation solution.

lyap2 - Lyapunov equation solution using diagonalization.

Demonstrations:

ctrldemo - Introduction to the Control Toolbox.

boildemo - LQG design of boiler system.

jetdemo - Classical design of jet transport yaw damper.

diskdemo - Digital control design of hard disk controller.

kalmdemo - Kalman filter design and simulation.

8. APLICANDO MATLAB AL CONTROL DE PROCESOS

8.1 Respuesta en el dominio del tiempo

Para obtener la respuesta de un sistema en el tiempo ante una entrada estándar, debe primero definirse el sistema. Para ello puede definirse en MatLab la función de transferencia propia del sistema o las ecuaciones de estado.

La función de transferencia de un sistema es una relación formada por un numerador y un denominador:

En MatLab debe definirse el numerador Y(s) y el denominador U(s) como vectores, cuyos elementos son los coeficientes de los polinomios del numerador y del denominador en potencias decrecientes de S. Por ejemplo, para definir la función de transferencia:

```
>>y=[1];
>>u=[1 0.25 1];
```

Para determinar la respuesta en el tiempo para una entrada escalón unitario de este sistema se usa el comandos step indicando el vector del numerador y del denominador entre paréntesis. step(num,den)

```
>>step(y,u)
```

MatLab presenta la respuesta en el tiempo en la ventana de figuras:

Puede definirse el tiempo en el cual se desea la respuesta al escalón, mediante un vector de tiempo T, step(num,den,T)

```
>>t=0:0.1:20;

>>step(y,u,t)
```

Se define t como un vector cuyo elemento inicial es 0, su elemento final es 20 y existen elementos que son el incremento desde 0 hasta 20 de 0.1 en 0.1. Al ejecutar el comando step para y y u se obtiene en la ventana de figuras la respuesta escalón para los primeros 20 segundos.

Otra forma de definir el sistema en MatLab es usando las ecuaciones de estado de la forma:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

MatLab permite hacer la conversión de una función de transferencia a su equivalente en ecuaciones de estado, mediante el comando tf2ss. Se deben especificar las cuatro matrices de estado de la forma:

```
[A,B,C,D]=tf2ss(num,den)
```

Para el ejemplo anterior tenemos:

```
>>[a,b,c,d]=tf2ss(y,u)

a =

-0.2500 -1.0000

1.0000 0

b =

1

0

c =

0 1

d =

0
```

Se puede hacer la conversión de una ecuación de estado a su equivalente función de transferencia, mediante el comando ss2tf. Se deben especificar los vectores para almacenar los coeficientes del polinomio numerador y del denominador. Su

Sintaxis es:

[num,den]=ss2tf(a,b,c,d)

Ejemplo

```
>>[num,den]=ss2tf(a,b,c,d)
num =
0 0 1.0000
den =
1.0000 0.2500 1.0000
```

Para obtener la respuesta escalón de un sistema a partir de las ecuaciones de estado se usa el

comando step con la

Sintaxis:

step(A,B,C,D)

Ejemplo

```
>>step(a,b,c,d)
```

Para obtener la respuesta en el tiempo para una entrada impulso unitario se usa el comando impulse, con Sintaxis idéntica a la utilizada con el comando step:

Si se define el sistema en MatLab por los polinomios del numerador y denominador de la función de transferencia tenemos:

```
» y=[1 5 4];
» u=[1 6 11 6];
» impulse(y,u)
```

Si por el contrario el sistema se define en MatLab por las ecuaciones de estado:

```
» [A,B,C,D]=tf2ss(y,u)
A =
-6 -11 -6
1 0 0
0 1 0
```

```
B =
1
0
0
C =
1 5 4
D =
0
» impulse(A,B,C,D)
```

En ambos casos, MatLab presenta la respuesta en el tiempo en la ventana de figuras:

MatLab permite, además de obtener la respuesta en el tiempo para una entrada escalón o impulso, también obtener respuesta para otras entradas tal como rampas o sinusoides. El comando lsim permite obtener la respuesta en el tiempo para un sistema con una entrada u, donde u se define como una función del tiempo.

La Sintaxis de este comando es: lsim(A,B,C,D,U,T) usando las matrices de estado o lsim(NUM,DEN,U,T) usando la función de transferencia.

Para obtener la respuesta en el tiempo para una función rampa, se define U de la siguiente forma:

```
>>T=0:0.1:10
>>U=T;
>>NUM=[1];
>>DEN=[1 0.25 1];
>>[Y,X]=lsim(NUM,DEN,U,T);
>>PLOT(T,Y,T,U)
```

Al hacer U=T se está definiendo la función rampa. T es el vector de tiempo variando desde 0 hasta 10

seg. NUM y DEN son los vectores de los coeficientes decrecientes en potencia de S de los polinomios del numerador y del denominador respectivamente. En la variable Y se almacena la salida del sistema en función del tiempo T. El comando plot permite presentar en la ventana de figuras la variable Y (salida) y la entrada U (rampa) en función del tiempo, obteniéndose:

8.2 Respuesta en el dominio de la frecuencia

Para el estudio de un sistema en el dominio de la frecuencia existen tres herramientas disponibles en MatLab como son: los diagramas de Bode, de Nyquist y de Nichols.

Para obtener el diagrama de Bode de una función de transferencia, se definen dos vectores cuyos elementos son los coeficientes de los polinomios del numerador y del denominador en potencias decrecientes de S. Estos vectores son usados en el comando bode con la siguiente

Sintaxis:

bode(num,den).

Se define la función de transferencia:

Ejemplo

```
>>y=[1];
>>u=[1 0.25 1];
>>bode(y,u)
```

MatLab presenta el diagrama de bode en la ventana de figuras:

Otro formato mediante el cual el comando bode presenta el diagrama de bode, es a través de las ecuaciones de estado representadas por las matrices de estado (A,B,C,D). Su

Sintaxis es:

bode(A,B,C,D).

Para especificar un rango deseado de frecuencias en las cuales se desea obtener el diagrama de Bode, se emplea un vector de frecuencias en el que se especifica la frecuencia inicial, el incremento y la frecuencia final. Por ejemplo:

```
>>W=0:0.1:100;
>>bode(y,u,W)
```

Este comando muestra el diagrama de Bode entre 0 y 100 rad/s.

Otra herramienta de análisis en el dominio en la frecuencia que ofrece MatLab es el diagrama de Nichols. Para obtener el diagrama de Nichols se utiliza el comando nichols, cuya Sintaxis es idéntica a la del comando bode: nichols(A,B,C,D,W) si se emplean las matrices de estado o nichols(num,den,W) si se emplea la función de transferencia.

Si se define y como el vector de los coeficientes del polinomio del numerador y u como el deldenominador:

```
>>y=[0 0 100];
>>u=[0.04 1 0];
>>nichols(y,u)
```

MatLab presenta en la ventana de figuras el diagrama de Nichols:

Otra herramienta de análisis en el dominio en la frecuencia que ofrece MatLab es el diagrama de Nyquist. Para obtenerlo se utiliza el comando nyquist, cuya Sintaxis es idéntica a la del comando bode y nichols: nyquist(A,B,C,D,W) si se emplean las matrices de estado o nyquist(num,den,W) si se emplea la función de transferencia.

Si se define y como el vector de los coeficientes del polinomio del numerador y u como el del denominador:

```
>>y=[1];
>>u=[1 6 5];
>>nyquist(y,u)
```

MatLab presenta en la ventana de figuras el diagrama de Nyquist:

Para obtener el margen de ganancia, el margen de fase, la frecuencia de cruce de ganancia y la frecuencia de cruce de fase MatLab dispone del comando margin. Las diferentes formas de utilizar este comando son:

[Gm,Pm,Wcg,Wcp] = MARGIN(A,B,C,D) retorna los valores de margen de ganancia (Gm), margen de fase (Pm), frecuencia de cruce de ganancia (Wcg) y la frecuencia de cruce de fase (Wcp) cuando se trabaja con las matrices de estado (A,B,C,D).

[Gm,Pm,Wcg,Wcp] = MARGIN(NUM,DEN) cuando se trabaja con la función de transferencia.

[Gm,Pm,Wcg,Wcp] = MARGIN(MAG,PHASE,W) toma los vectores de magnitud, fase y frecuencia del diagrama de Bode.

MARGIN(A,B,C,D) dibuja el diagrama de Bode y muestra con líneas verticales los márgenes deganancia y de fase.

```
>>num=10;
>>den=[1 0.25 1];
>>[Gm,Pm,Wcg,Wcp] =margin(num,den)
Gm =
Inf
Pm =
4.7487
Wcg =
NaN
Wcp =
3.3114
>>margin(num,den)
```

8.3 Lugar de las raíces

Para obtener el lugar de las raíces de un sistema como el mostrado en el siguiente diagrama:

Se debe determinar su ecuación característica, la cual es de la forma:

Para obtener el lugar de las raíces, MatLab dispone del comando rlocus. Las diferentes

Sintaxis para utilizar este comando son:

rlocus(NUM,DEN) calcula y dibuja el lugar de las raíces cuando se trabaja con la función de transferencia donde NUM y DEN son los vectores de los coeficientes en potencia descendiente de S de los polinomios del numerador y denominador de la función de transferencia G(S). MatLab generará automáticamente un conjunto de valores de la ganancia K.

rlocus(NUM,DEN,K): calcula y dibuja el lugar de las raíces cuando se trabaja con la función de transferencia y ha sido previamente definido el rango de valores de K. Por ejemplo de 0 a 100 con incrementos de 10: k=0:10:100

R = rlocus(NUM,DEN,K) o [R,K] = rlocus(NUM,DEN) no dibuja el lugar de las raíces pero almacena en la matriz R, de longitud igual al número de elementos de K, la localización de las raíces. R tendrá tantas columnas como raíces existan, estas pueden además ser complejas.

rlocus(A,B,C,D), R=rlocus(A,B,C,D,K), o [R,K]=rlocus(A,B,C,D) son equivalentes a las Sintaxis anteriores pero empleando las matrices de estado para hallar el lugar de las raíces.

Para la siguiente forma modificada de la ecuación característica de un sistema se desea hallar el lugar de las raíces mediante MatLab:

```
>>num=[0,0,0,1];
>>den=[1,3,2,0];
>>rlocus(num,den)
```

MatLab dispone del comando rlocfind que permite determinar los polos del sistema para una valor determinado de k. Su

Sintaxis es:

[K,POLES] = rlocfind(num,den) permite determinar los polos para un valor determinado de k, cuando se trabaja con la función de transferencia. Por medio del curso en el lugar de las raíces se selecciona una localización, MatLab retorna el valor de k para esta localización y los polos asociados a esta ganancia.

Cuando se trabaja con las matrices de estado, las Sintaxis para el comando rlocfind es: **[K,POLES] = rlocfind(A,B,C,D).**

Al ejecutar el comando rlocfind con la función de transferencia anterior, MatLab activa la ventana de figuras en espera de que el usuario seleccione un punto del lugar de las raíces mediante el cursor. En este caso el punto seleccionado fue -2.4623 en la parte real y - 0.0132 en la parte imaginaria.

```
» [k,poles]=rlocfind(num,den)
Select a point in the graphics window
selected_point =
-2.4623 - 0.0132i
k =
1.6655
poles =
-2.4625
-0.2688 + 0.7773i
-0.2688 - 0.7773i
```

Para seleccionar el punto en el cual calcular los polos del lugar de las raíces sin usar el cursor se agrega un parámetro al comando rlocfind. Este debe ser el punto o los puntos en donde se desea tomar el valor de k. La nueva

Sintaxis es:

[K,POLES] = rlocfind(A,B,C,D,P) o [K,POLES] = rlocfind(num,den,P)

P debe definirse previamente indicando la parte real e imaginaria del mismo. Por ejemplo: P=3+0i o P=1-0.555i.

8.4 Controladores PID

Para implementar los diferentes tipos de controladores (P, PD, PI, PID) en MatLab se hace uso de la función de transferencia propia del sistema a objeto de estudio. Si dicho sistema es de la forma:

donde G(S) es la función de transferencia de la planta o proceso; mientras que C(S) es la función de transferencia del controlador.

Para el caso del controlador proporcional, C(S)=Kp, que es una constante o valor escalar. El controlador PI es C(S)=Kp + Ki/S que puede representarse como una relación ente dos polinomios.

El controlador PID es C(S)=Kp + Ki/S + Kd S que se representa como:

que es de nuevo una relación entre dos polinomios. Los coeficientes decrecientes en potencias de S de estos polinomio pueden ser almacenados en vectores en MatLab. Si se multiplica el controlador C(S) por la función de transferencia del proceso o planta G(S) se formará la función de transferencia de lazo abierto. Por ejemplo un G(S) puede ser:

Para obtener la respuesta en lazo abierto ante una entrada escalón unitario tenemos:

```
>>Kp=50;
>>Ki=1;
>>Kd=10;
>>num=[Kd Kp Ki];
>>den=[1 10 20 0 0];
>>step(num,den)
```

Para obtener la respuesta de lazo cerrado en el tiempo para una entrada escalón unitario se emplea el comando cloop, el cual genera los polinomios del numerador (numc) y denominador (denc) de la función de transferencia de lazo cerrado con realimentación unitaria a partir de los polinomios de la función de transferencia de lazo abierto (num y den). Su

Sintaxis es:

[numc,denc]=cloop(num,den,sign)

El signo de la realimentación viene dado por sign. Para el ejemplo anterior, tenemos:

```
>>Kp=500;
>>Ki=1;
>>Kd=100;
>>num1=[Kd Kp Ki];
>>den1=[1 0];
>>num2=1;
>>den2=[1 10 20 0];
>>[numc,numd]=cloop(conv(num1,num2),conv(den1,den2),-1);
>>step(numc,denc)
```

Se usa el comando conv para obtener la convolución y multiplicación polinomial de dos vectores. La salida obtenida mediante el comando step se muestra a continuación:

9. TRUCOS EN MATLAB®

Paper semilogarítmico gratis: papelbod.m

Para cotejar sus diagramas de Bode:

>>bode(num,den)

donde num y den son vectores que contienen los coeficientes del numerador y denominador de H(s) en orden de potencias descendentes de s.

Nota: Esto da las curvas exactas, no las aproximaciones asintóticas con líneas rectas.

Ejemplo: Para ,

Escribimos

>>bode([158.11 15.811],[1 5 0])

Precaución: El punto "." puede significar operación elemento-por-elemento o punto decimal.

Cuando escribimos un dígito pegado al punto como "2.", el interpretador cree que es el número "2.0". Entonces si queremos calcular A2B2, donde A y B son arreglos y no matrices (o sea, queremos operación elemento-por-elemento), debemos escribir

>>A.^2 .*B.^2 (notar el espacio después del primer 2)

y no

>>A.^2.*B.^2

Para remover ejes de la gráfica:

>>set(gca,'Visible','off')

o simplemente

>>axis off

Para cambiar el color de trasfondo de la gráfica:

```
>> whitebg('c')
```

donde c es el código del color descrito en help plot.

Para establecer propiedades de la gráfica, es más fácil hacerlo al crearla que después. Por ejemplo, para graficar con una línea gruesa,

```
>>plot(x,y,'linewidth',3)
```
(En el momento de creación)

```
>>set(get(gca,'children'),'linewidth',3)
```
(Después de creada)