AAU

AMERICAN ANDRAGOGY UNIVERSITY

MATLAB

APLICADO A ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS

Alfaomega

Contenido

Plataforma de contenidos interactivos

Para tener acceso al código fuente de los programas de ejemplos y ejercicios presentados en **MATLAB Aplicado a Robótica y Mecatrónica**, siga los siguientes pasos:

1) Ir a la página

http://virtual.alfaomega.com.mx

2) Registrarse como usuario del sitio.

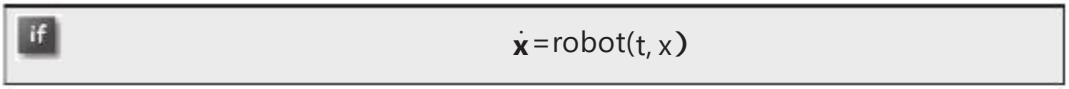
3) En el catálogo identificar este libro y descargar el material adicional.

Simbolog´ıa e iconograf´ıa utilizada

Diversos recursos did´acticos est´an presentes en esta obra; particularmente se resaltan las siguientes herramientas:

Librer´ıas

La sintaxis de las librer´ıas que se desarrollan son representadas como:



Los ejemplos ilustrativos se presentan de la siguiente manera:

♠ ♠ Ejemplo 5.1

El enunciado de cada ejemplo se encuentra dentro de un recuadro con fondo gris. Los ejemplos son presentados por nivel de complejidad, para el nivel simple o b´asico se emplea la marca ♠, ejemplos del nivel intermedio ♠♠ y complejos son denotados por ♠♠♠

Soluci´on

Se detalla la respuesta de cada ejemplo por ecuaciones y programas en c´odigo fuente. Adicionalmente, todos los ejemplos incluyen un n´umero de referencia que identifica al cap´ıtulo donde fue definido.

Todos los programas de esta obra han sido implementados en lenguaje **MATLAB** versi´on 11. El c´odigo fuente se identifica por el siguiente recuadro:

C´odigo Fuente 5.1 robot.m

%MATLAB Aplicado a Rob´otica y Mecatr´onica.
%Editorial Alfaomega, Fernando Reyes Cort´es.
%Simulaci´on de robots manipuladores

robot.m

```
1  clc;
2  clear all;
3  close all;
4  format short g
5  ti=0; h=0.001; tf = 10;%vector tiempo
6  t=ti:h:tf;%tiempo de simulaci´on
7  ci=[0; 0; 0; 0];
8  opciones=odeset('RelTol',1e-3,'InitialStep',1e-3,'MaxStep',1e-3);
9  [t,x]=ode45('robot',t,ci,opciones);
10 plot(x(:,1),x(:,2))
```

Para representar una idea general de una instrucci´on de programaci´on, se emplea pseudo-c´odigo como el siguiente:

Estructura de c´odigo 5.1

Pseudo c´odigo

```
while k<1000
    qp(k)=(q(k)-q(k-1))/h;
    if q(k)>100
        for j=1:1000
            qpp(j)=robot(q(j),qp(j));
        end
    end
    k=k+1;
    otro grupo de instrucciones;
end
```

Pr´ologo

Rob´otica y mecatr´onica representan en la actualidad a´reas estrat´egicas y claves para todo pa´ıs que aspire a la modernidad y bienestar, ya que su impacto no s´olo repercute en aspectos pol´ıticos y econ´omicos, tambi´en forma parte importante de la vida cotidiana, educaci´on, cultura, y en todos los a´mbitos de la sociedad.

La simulaci´on es una herramienta imprescindible para reproducir los fen´omenos f´ısicos de un robot o de un sistema mecatr´onico, permite estudiar y analizar a detalle los aspectos pr´acticos que intervienen en tareas espec´ıficas que debe realizar un robot industrial. La simulaci´on es un proceso previo a la etapa experimental donde es posible entender los conceptos claves de la rob´otica y mecatr´onica. Bajo este escenario se ubica la presente obra a trav´es de la propuesta de un conjunto de librer´ıas para **MATLAB** que permitan realizar estudio, an´alisis, simulaci´on y aplicaciones de robots manipuladores y sistemas mecatr´onicos.

Esta obra presenta la propuesta y desarrollo de una clase particular de librear´ıas toolbox para rob´otica y mecatr´onica. Las librer´ıas son funciones en c´odigo fuente para **MATLAB** que permiten modelar la cinem´atica directa e inversa, transformaciones homog´eneas (rotaci´on y traslaci´on), din´amica, identificaci´on param´etrica, control de robots manipuladores. El contenido y material incluido en el libro est´a dirigido a estudiantes de ingenier´ıa en electr´onica, sistemas, computaci´on, industrial, rob´otica y mecatr´onica. No obstante, tambi´en puede ser adecuado para nivel posgrado.

La organizaci´on de este libro consta de cuatro partes: la **Parte I Programaci´on** incluye dos cap´ıtulos. El **Cap´ıtulo 1 Conceptos ba´sicos** de programacio´n del lenguaje de **MATLAB** tiene la finalidad de que el lector adquiera los conocimientos necesarios para adquirir solvencia en programar aplicaciones en c´odigo fuente. El cap´ıtulo 2 **M´etodos num´ericos** presenta aspectos relacionados con las t´ecnicas de m´etodos num´ericos enfocados a resolver problemas de diferenciaci´on e integraci´on num´erica de sistemas din´amicos.

La **Parte II Cinem´atica** est´a relacionada con el an´alisis cinem´atico de las principales clases de robots manipuladores. Se componen de tres cap´ıtulos; el

cap´ıtulo 3 **Preliminares matem´aticos** contiene las bases de las reglas de las matrices de rotaci´on, traslaci´on y transformaciones homog´eneas. En el cap´ıtulo 4 **Cinem´atica directa** se detalla la metodolog´ıa de Denavit-Hartenberg para describir la cinem´atica directa; asimismo se describe la cinem´atica inversa, cinem´atica diferencial y jacobianos. Un compendio de librer´ıas se desarrollaron con variables simb´olicas y aplicaciones num´ericas para los principales tipos de robots manipuladores en el cap´ıtulo 5 **Cinem´atica directa cartesiana**.

La **Parte III Din´amica** describe la t´ecnica de simulaci´on para sistemas din´amicos e identificaci´on param´etrica. Formada por el cap´ıtulo 6 **Din´amica** donde se realiza simulaci´on de sistemas mecatr´onicos y robots manipuladores con la estructura de una ecuaci´on diferencial ordinaria de primer orden descrita por variables de estado. El cap´ıtulo 7 **Identificaci´on param´etrica** presenta cinco esquemas de regresi´on con la t´ecnica de m´ınimos cuadrados. Se proponen extensos ejemplos para ilustrar el procedimiento para encontrar el valor num´erico de los par´ametros del robot.

Finalmente, la **Parte IV** cap´ıtulo 8 **Control de posici´on** contiene la simulaci´on de algoritmos cl´asicos como el control PD y PID; as´ı como simulaci´on de nuevas estrategias de control con mejor desempeño que pertenecen a la filosof´ıa de diseño moderno conocida como moldeo de energ´ıa. Aplicaciones de control punto a punto se ilustran para describir la t´ecnica de simulaci´on.

La realizaci´on de esta obra no hubiera sido posible sin el apoyo de la **Benem´erita Universidad Aut´onoma de Puebla** (BUAP), particularmente agradezco a la Facultad de Ciencias de la Electr´onica por todas las facilidades brindadas. El autor desea agradecer principalmente al Dr. Jaime Cid, M. C. Fernando Porras, Dra. Aurora Vargas, Dr. Sergio Vergara, Dra. Amparo Palomino y al M. C. Pablo S´anchez, as´ı como al programa de financiamiento de proyectos de investigaci´on de la Vicerrector´ıa de Investigaci´on y Estudios de Posgrado (VIEP), especialmente a la Dra. Rosa Graciela Montes y al Dr. Pedro Hugo Hern´andez Tejeda.

Fernando Reyes Cort´es
H. Puebla de Z., a 3 de diciembre de 2011
Facultad de Ciencias de la Electr´onica
Benem´erita Universidad Aut´onoma de Puebla

Parte I

Programaci´on

La **Parte I** de la presente obra está destinada a estudiar el lenguaje de programación de **MATLAB** y la implementación de métodos numéricos aplicados a la robótica y mecatrónica. Esta primera parte se compone de dos capítulos. El capítulo 1 **Conceptos básicos** presenta los aspectos básicos del lenguaje **MATLAB** y tiene la finalidad de que el lector adquiera los conocimientos fundamentales para adquirir solvencia en programar en código fuente. El capítulo 2 **Métodos numéricos** presenta aspectos relacionados con las técnicas de métodos numéricos enfocados a resolver problemas de modelado y control de las áreas de robótica y mecatrónica.

Capítulo 1 Conceptos básicos

Descripción de los principales componentes del ambiente de programación de **MATLAB**. Sintaxis y programación, declaración de variables, tipo de datos, operadores, programación, manipulación numérica de matrices y arreglos, funciones, gráficas, funciones importantes, matrices, operadores, instrucciones, lazos de programación, etcétera. Un conjunto de ejemplos se desarrollan para ilustrar los temas presentados.

Capítulo 2 Métodos numéricos

Conocer los principales métodos de diferenciación e integración numérica que se aplican en robótica y mecatrónica. Sistemas de ecuaciones lineales, método de Euler e integración numérica (trapezoidal, Simpson). Simulación de sistemas dinámicos con la técnica de Runge-Kutta.

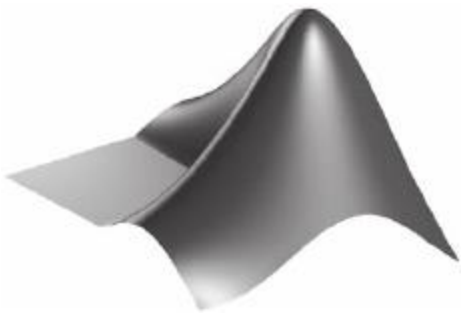
La parte I concluye con referencias selectas y un conjunto de problemas para mejorar las habilidades y conocimientos de los capítulos tratados.

Referencias selectas

Problemas propuestos

Cap´itulo 1

Conceptos b´asicos



Objetivos

Presentar la plataforma y ambiente de programaci´on de **MATLAB** con los fundamentos y bases de su lenguaje enfocado para adquirir solvencia y dominio en la implementaci´on de algoritmos con aplicaciones en mecatr´onica y rob´otica.

Objetivos particulares:

- Aprender el Lenguaje **MATLAB**.
- Conocer los operadores.
- Programar con matrices y arreglos.
- Manejar funciones e instrucciones.
- Graficar funciones y datos en 2D y 3D.
- Desarrollar formatos para manejar datos experimentales.

1.1 Introducci´on

Hoy en d´ıa, **MATLAB** es un lenguaje de programaci´on matem´atico de alto nivel integrado con entorno gr´afico amigable, visualizaci´on de datos, funciones, gr´aficas 2D y 3D, procesamiento de im´agenes, video, computaci´on num´erica para desarrollar algoritmos matem´aticos con aplicaciones en ingenier´ıa y ciencias exactas. Particularmente, en ingenier´ıa es una herramienta muy poderosa para realizar aplicaciones en mecatr´onica, rob´otica, control y automatizaci´on.

MATLAB es un acr´onimo que proviene de **matrix laboratory** (laboratorio matricial) creado por el profesor y matem´atico Cleve Moler en 1970. La primera versi´on de **MATLAB** fue escrita en lenguaje fortran la cual contempl´o la idea de emplear subrutinas para los cursos de a̕lgebra lineal y an´alisis num´erico de los paquetes LINPACK y EISPACK; posteriormente se desarroll´o software de matrices para acceder a esos paquetes sin necesidad de usar programas en fortran. La empresa The Mathworks (http://www.mathworks.com), propietaria de **MATLAB** fue fundada por Jack Little y Cleve Moler en 1984. El actual paquete de **MATLAB** se encuentra escrito en lenguaje C, y su primera versi´on en este lenguaje fue escrita por Steve Bangert quien desarroll´o el int´erprete parser. Steve Kleiman implement´o los gr´aficos, las rutinas de an´alisis, la gu´ıa de usuario, mientras que la mayor´ıa de los archivos.m fueron escritos por Jack Little y Cleve Moler. Actualmente, el lenguaje de programaci´on de **MATLAB** proporciona un sencillo acceso a algoritmos num´ericos que incluyen matrices.

Las versiones recientes del lenguaje **MATLAB** se caracterizan por ser multiplataforma, es decir se encuentra disponible para sistemas operativos como Unix, Windows y Apple Mac OS X. **MATLAB** posee varias caracter´ısticas computacionales y visuales, entre las que sobresalen la caja de herramientas (toolbox) la cual representa un amplio compendio de funciones y utiler´ıas para analizar y desarrollar una amplia gama de aplicaciones en las ´areas de ingenier´ıa y ciencias exactas. Un rasgo distintivo de **MATLAB** es que ofrece un entorno gr´afico de programaci´on amigable al usuario a trav´es de herramientas y utiler´ıas para realizar simulaci´on de sistemas din´amicos, an´alisis de datos, procesamiento de im´agenes y video, gr´aficas y m´etodos

de visualización, desarrollo de interfaces de usuario (GUI); también permite realizar interfaces para sistemas electrónicos, por ejemplo adquisición de datos con una versatilidad de tarjetas de instrumentación electrónica comerciales con plataforma de microprocesadores, DSP's, PIC's, FPGA's, manejo de puertos como USB, COM, paralelo y diseños electrónicos propios.

MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones: plataforma de simulación multidominio **Simulink** y **GUIDE** editor de interfaces de usuario. Se recomienda al lector visitar:

www.mathworks.com/products/matlab

en ese sitio WEB se pueden encontrar productos actuales de **MATLAB**, manuales, notas técnicas, cursos, aplicaciones y foros internacionales, así como sociedades de desarrollo. Actualmente, **MATLAB** se ha convertido en un referente de cómputo a nivel internacional, debido que en la mayoría de las universidades, centros de investigación e industria lo utilizan para desarrollar y llevar a cabo proyectos de ciencia y tecnología.

Específicamente en el área de ingeniería **MATLAB** permite realizar simulaciones de sistemas mecatrónicos y robots manipuladores. El proceso de simulación resulta importante cuando no se dispone de una adecuada infraestructura experimental. Sin embargo, la simulación depende de un buen modelo matemático que permita reproducir fielmente todos los fenómenos físicos del sistema real. La simulación es flexible ya que permite detectar posibles deficiencias en el diseñõ del modelado. Una vez depurado el modelo dinámico, entonces la simulación facilita el proceso para analizar, estudiar y comprender el comportamiento de la dinámica del sistema. Esta etapa es fundamental para el diseñõ de algoritmos de control. Cuando el modelo matemático del sistema mecatrónico o robot manipulador es lo suficientemente completo entonces la simulación proporciona un medio virtual del sistema real.

El propósito de este capítulo se ubica en preparar al lector para conocer mejor el entorno de programación, programar y manejar en forma solvente el lenguaje de **MATLAB**. En otras palabras, proporcionar todos los elementos necesarios de programación (variables, datos, instrucciones y funciones) para realizar diversas

aplicaciones para las ´areas de rob´otica y mecatr´onica.

1.2 Componentes

El ambiente de programaci´on de **MATLAB** es amigable al usuario, y est´a compuesto por una interface gr´afica con varias herramientas distribuidas en ventanas que permiten programar, revisar, analizar, registrar datos, utilizar funciones, historial de comandos y desarrollar diversas aplicaciones. La forma de iniciar el paquete **MATLAB** es realizando doble click sobre el icono colocado en el escritorio de Windows (Windows desktop). La pantalla principal de **MATLAB** se presenta en la figura 1.1 la cual contiene la interface gr´afica de usuario con varias ventanas como la de comandos (Command Window **fx** >>), manejo de archivos, espacio de trabajo (Workspace), historial de comandos (Command History), directorio de trabajo y aplicaciones como Simulink.

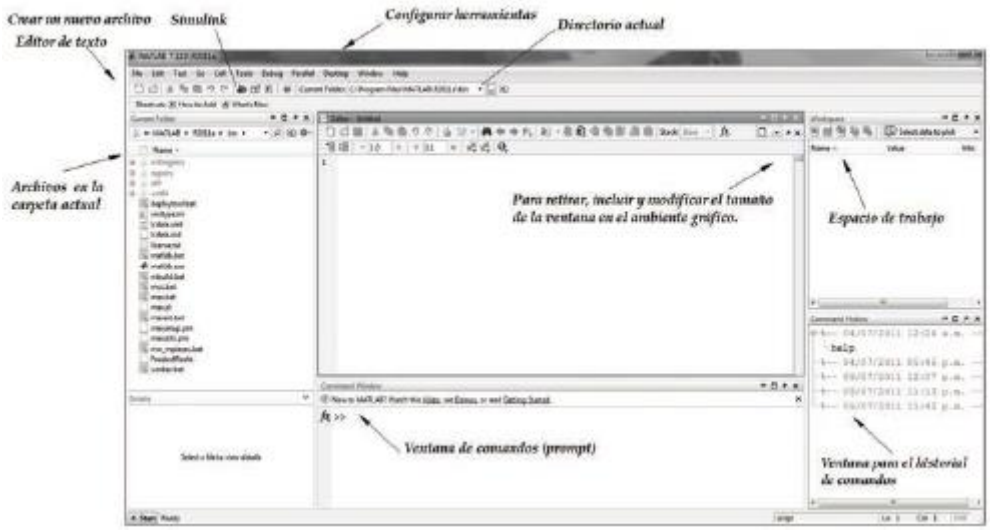


Figura 1.1 Ambiente de programaci´on de **MATLAB**.

Es recomendable personalizar el ambiente de programaci´on de **MATLAB** para una f´acil interacci´on con manejos de archivos, datos y herramientas de programaci´on. Por ejemplo, para integrar la ventana del editor dentro del ambiente gr´afico se logra oprimiendo el icono del editor de texto colocado en la esquina superior izquierda,

justo abajo de la opci´on File del men´u principal, posteriormente se hace click sobre la flecha colocada en la esquina superior derecha de la ventana del editor. La figura 1.2 muestra el ambiente configurado.

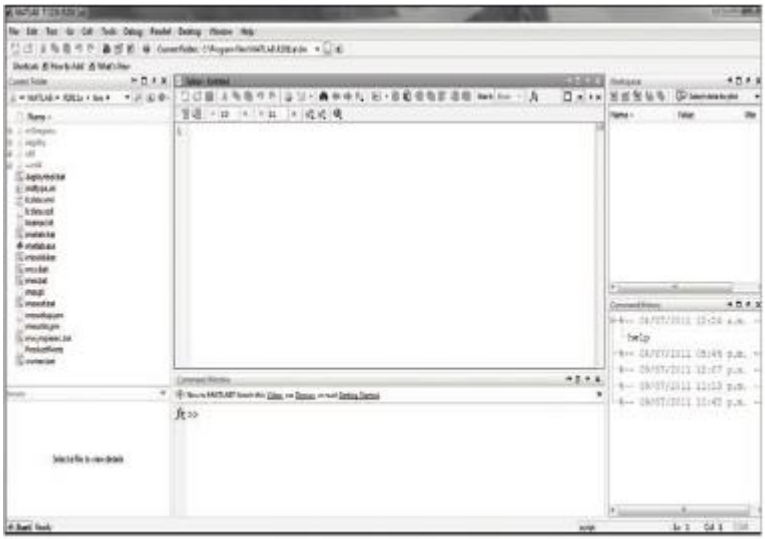


Figura 1.2 Editor de texto integrado en el ambiente de programaci´on.

Generalmente el usuario registra sus archivos en una carpeta o directorio predefinido; para dar de alta dicho directorio al momento de simular aparecer´a el mensaje que se muestra en la figura 1.3. Seleccionar Add to Path para que **MATLAB** realice la simulaci´on desde esa trayectoria de trabajo, de esta forma las trayectorias de otros directorios que est´en habilitadas no se perder´an, por lo que la trayectoria del usuario se añade a las ya existentes.
Las componentes principales del ambiente de programaci´on de **MATLAB** son:

- Herramientas de escritorio y ambiente de desarrollo.
- Librer´ıas.
- Lenguaje **MATLAB**.
- Gr´aficas.
- Interfaces externas/API.

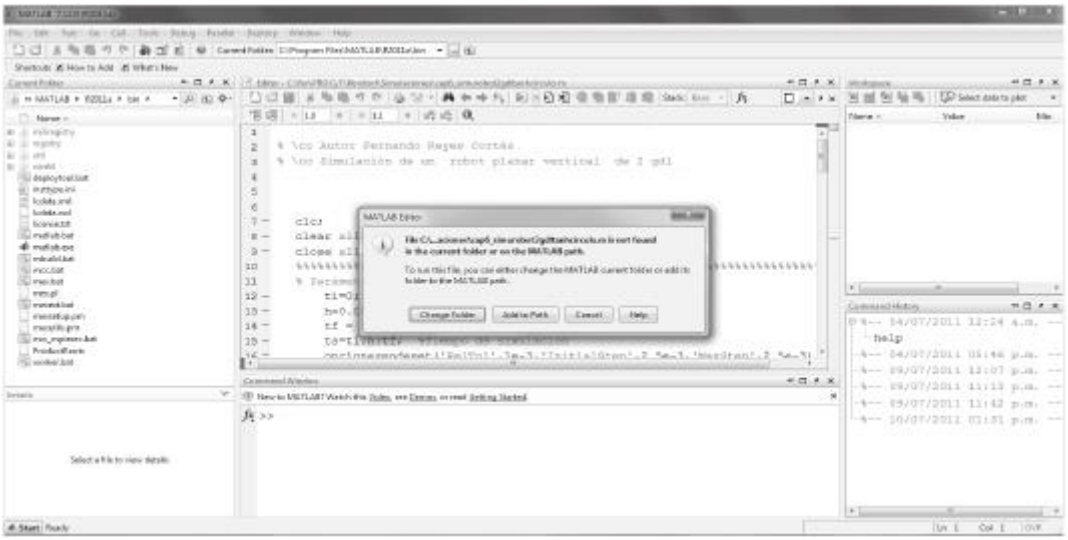


Figura 1.3 Habilitaci´on del directorio de trabajo de usuario.

Herramientas de escritorio y ambiente de desarrollo

MATLAB tiene un conjunto de herramientas que facilitan el uso de funciones, comandos, programas, datos, variables y archivos (desktop tools and development environment). La mayor´ıa de las herramientas son interfaces gr´aficas donde el usuario interacciona con comandos, funciones, editor de texto, variables y datos. Por ejemplo **MATLAB** desktop, Command Windows, Simulink, ayuda en l´ınea, analizador de c´odigo, navegador de archivos, etc´etera.

Librer´ıas

Las librer´ıas de **MATLAB** son un conjunto muy grande de funciones y algoritmos matem´aticos que van desde funciones trigonom´etricas, operaciones b´asicas con matrices hasta funciones complejas como an´alisis de regresi´on, inversa de matrices, funciones Bessel, transformada r´apida de Fourier, Laplace, programaci´on de DSP's, microprocesadores, procesamiento de im´agenes, video y tarjetas de instrumentaci´on electr´onica, entre otras ventajas.

Lenguaje

El lenguaje de programación de **MATLAB** es de alto nivel y permite programar matrices, arreglos e incorporar instrucciones de control de flujo del programa, funciones, comandos y estructura de datos.

MATLAB contiene una diversidad en comandos que facilitan al usuario la implementación del problema a simular. Los comandos son funciones muy específicas que tienen un código depurado y que no se encuentra disponible al usuario.

El lenguaje de programación de **MATLAB** contiene todos los elementos de programación necesarios para desarrollar aplicaciones en realidad virtual, programación de robots manipuladores, análisis de regresión y estadísticos de señales experimentales, procesamiento y extracción de rasgos distintivos de imágenes y video, así como varias aplicaciones más en ingeniería y ciencias exactas.

Gráficos

MATLAB contiene un enorme número de funciones que facilitan la representación gráfica y visualización de variables, funciones, vectores, matrices y datos que pueden ser graficados en 2 y 3 dimensiones. Incluye también funciones de alto nivel para el análisis y procesamiento de gráficas, datos experimentales o de simulación, video e imágenes, animación y presentación de sólidos, así como el desarrollo de aplicaciones con interface gráfica para incluir menús con botones, barras deslizadoras, diagramas a bloques, instrumentos de medición y ventanas para seleccionar opciones o herramientas de la aplicación.

Interfaces externas/API

Hay varias herramientas especiales para escribir programas en lenguaje C, C++, fortran y Java que interactúen con programas en lenguaje **MATLAB** facilitando el proceso de llamada a rutinas desde **MATLAB** y pase de parámetros, así como la

programaci´on de puertos (USB, COM, paralelo y serial) que permiten conectar tarjetas de instrumentaci´on electr´onica para adquisici´on de datos y evaluaci´on experimental de algoritmos de control.

1.3 Inicio

Una vez instalado **MATLAB**, la forma m´as simple de interaccionar con este paquete es introduciendo expresiones directamente en la venta de comandos, por ejemplo iniciando con el tradicional letrero "Hola mundo..." sobre el prompt:

```
fx >> disp('Hola mundo...') ←

      Hola mundo...
fx >> 9+9 oprimir la tecla Enter ←
      ans=

                  18

fx >> sin(10) oprimir Enter ←
      ans=

                -0.5440
```

Cuando no se especifique el nombre de la variable, **MATLAB** usar´a el nombre por default ans que corresponde a un nombre corto de answer. Al especificar una variable el resultado se desplegar´a con el nombre de esa variable, por ejemplo:

```
fx >> z=9+9 Enter ←
      z=

                  18

fx >> y=sin(10) Enter ←
      y=

                -0.5440
```

MATLAB funciona como calculadora, sobre el prompt de la ventana de comandos se pueden escribir expresiones aritméticas mediante los operadores "+","-","/", "*", "^" para la suma, resta, división, multiplicación y potencia, respectivamente. La tabla 1.1 contiene los operadores aritméticos básicos.

Por ejemplo, similar a una calculadora se pueden evaluar expresiones como:

```
fx >> (10.3+8*5/3.33)^5 ←
ans=

                    5.5296e+006

fx >> ans^2 ←
ans=

                    3.0576e+013
```

Tabla 1.1 Operadores aritméticos básicos

Adición	+
Sustracción	-
Multiplicación	*
División	/
División left	\
Potencia	^
Evaluación	()

MATLAB utiliza el operador **;** para deshabilitar la opción de desplegado en la ventana de comandos. Por ejemplo,

```
fx >> z=9+9; ←
fx >>
```

es decir, con el operador **;** **MATLAB** no exhibir´a ning´un valor de variable o funci´on.

El lenguaje **MATLAB** permite insertar comentarios usando el operador **%** el cual deber´a emplearse por cada l´ınea de comentarios. Por ejemplo:

```
fx >> w=10+20%Suma de dos enteros. ←
w=

    30

fx >> sign(-8999.4) %Obtiene el signo de un n´umero. ←
ans=

    -1
```

Obs´ervese que al ejecutarse la instrucci´on o la sentencia el comentario no se despliega. Los comentarios sirven para documentar aspectos t´ecnicos de un programa.

Herramienta de ayuda de MATLAB

MATLAB tiene una extensa gama de herramientas para solicitar ayuda en l´ınea (help), ya sea de manera general o en forma espec´ıfica buscar informaci´on para una instrucci´on comando o funci´on, as´ı como sus principales caracter´ısticas y ejemplos did´acticos (demos).

En la ventana de comandos puede solicitar ayuda de la siguiente forma:

```
fx >> help ←
fx >> helpwin ←
fx >> help general ←
fx >> helpdesk ←
fx >> help if ←
fx >> help demo ←
```

La ayuda en l´ınea del programa **MATLAB** tambi´en proporciona informaci´on sobre los demos y tutoriales que facilitan el aprendizaje del paquete.

Para suspender la ejecución de un programa

Cuando **MATLAB** se encuentra realizando la ejecución de un programa o cálculo computacional en la esquina inferior izquierda aparecería un icono que indica busy. Sin embargo, hay ocasiones que toma mucho tiempo la ejecución del programa debido a que se encuentra realizando operaciones complicadas, funciones discontinuas o con números muy grandes, en cuyo caso despliega **Inf**, que significa infinito.

```
fx >> sinh(1345e34) %Obtiene el seno hiperbólico de un número. ←
      ans=

                    Inf
```

La siguiente expresión genera un lazo infinito de ejecución. Es decir el programa se quedará de manera indefinida en ejecución.

```
fx >> while 1 end%Genera un ciclo infinito de ejecución del programa. ←
```

En este caso, hay ocasiones donde es recomendable interrumpir el proceso mediante la combinación de teclas CRTL C (presionar simultáneamente CONTROL y la tecla C).

> Para que tenga efecto CRTL C es muy importante que se encuentre sobre la ventana de comandos; si no es el caso primero haga click sobre dicha ventana, y posteriormente CRTL C. De lo contrario no tendría efecto la acción de suspender programa.

Finalizar la sesión de MATLAB

Para finalizar **MATLAB** simplemente escribir en la ventana de comandos lo siguiente:

```
fx >> quit ←
fx >> exit ←
```

También se pueden usar las siguientes opciones para finalizar la sesión con **MATLAB**:

Desde el teclado, presionar simult´aneamente CRTL Q o CONTROL con q

Presionar simult´aneamente las teclas ALT F y despu´es la opci´on Exit

Con el mouse sobre el men´u de **MATLAB** click sobre File y click en Exit.

1.4 Lenguaje

El lenguaje de programaci´on de **MATLAB** est´a compuesto por un conjunto de reglas gramaticales y sintaxis para escribir correctamente las variables, constantes, operadores, expresiones, funciones y todos los elementos que forman parte de la programaci´on.

Las variables, constantes, operadores y funciones forman una **expresi´on** la cual ser´a procesada por un analizador l´exico y sint´actico antes de ser ejecutada por la computadora. A diferencia de otros lenguajes, las expresiones en **MATLAB** involucran matrices.

Variables

En el lenguaje de **MATLAB** las variables no requieren ning´un tipo de declaraci´on o definici´on. Esto tiene varias ventajas ya que facilita la programaci´on. Cuando **MATLAB** encuentra el nombre de una nueva variable, autom´aticamente crea la variable y le asigna una localidad de memoria. Por ejemplo,

```
fx >> error_posici´on=20
```

esta variable es vista por **MATLAB** desde el punto de vista de programaci´on como una matriz de dimensi´on 1×1.

Los nombres usados para referenciar a las variables, funciones y otro tipo de objetos o estructuras definidos por el usuario se les conocen como **identificadores**. Los nombres de identificadores constan de una letra del idioma castellano como caracter

inicial (**con excepción de la letra ñ y sin acentos**) seguido de cualquier número de letras, dígitos, y también se puede usar el guión bajo (underscore _). **MATLAB** distingue las letras mayúsculas de las minúsculas; por ejemplo la variable A es diferente a la variable a. La longitud de los nombres de las variables puede ser cualquier número de caracteres. Sin embargo, **MATLAB** usa los primeros 63 caracteres del nombre (namelengthmax=63) e ignora los restantes. Por lo tanto, para distinguir variables es importante escribir los nombres de las variables con un máximo de 63 caracteres.

La función genvarname es útil para crear nombres de variables que son válidas y únicas. Por ejemplo, cadena=genvarname({'A', 'A', 'A', 'A'}) produce la siguiente salida: cadena = 'A' 'A1' 'A2' 'A3.

MATLAB puede trabajar con varios tipos de variables: en punto flotante, flotante doble, enteros, enteros dobles, enteros cortos o tipo char, cadenas de caracteres, expresiones simbólicas, etc. Los diversos tipos de variables son interpretados como matrices.

Para saber qué tipos de variables estamos usando podemos utilizar el comando whos como en los ejemplos siguientes, primero se declaran las variables y posteriormente se usa la función whos:

```
fx >>  i=9 ←
       i=

                              9

fx >>  x=3.35 ←
       x=

                            3.3500

fx >>  cadena= 'esta es una cadena de caracteres' ←
       cadena=

                'esta es una cadena de caracteres'

fx >>  A=[2 3; 3 4] ←
```

```
A=
     2     3
     3     4

fx >> whos ←
=
```

Name	Size	Bytes	Class	Attributes
A	2 × 2	32	double	
cadena	1 × 32	64	char	
i	1 × 1	8	double	
x	1 × 1	8	double	

La columna Size indica la cantidad de memoria asignada a esa variable, mientras que la columna Bytes Class indica el tipo de variable. Obsérvese que **MATLAB** trata a todas las variables como matrices para propósitos de programación.

Al terminar una sesión, o cuando ya no se usen la variables debido a que se desea trabajar con otro programa es recomendable limpiar todas las variables para liberar memoria. Para realizar lo anterior, se puede hacer con cada variable, por ejemplo `clear i`, posteriormente `clear x` y finalmente `clear cadena`. Otra forma de realizarlo es limpiando simultáneamente las tres variables a través de: `clear i x cadena` o cuando hay una gran cantidad de variables, entonces es conveniente usar `clear all`.

Números

La notación convencional que usa **MATLAB** para la representación de números es la decimal con un punto decimal, signo $\pm$ y un número determinado de dígitos después del punto decimal, esto depende del tipo de formato numérico empleado (ver format). En la notación científica se usan potencias de diez. También se emplean constantes tales como e, π. Los números complejos o imaginarios emplean i o j. Todos los números son almacenados internamente utilizando el formato long en notación estándar de la IEEE para punto flotante con precisión finita de 16 dígitos decimales y un rango finito de 10^{-308} a 10^{308}.

Algunos ejemplos de n´umeros son:

2.33	−105	0.005
9.999	8.345e88	9 + ¡3.2
3.1416	8e5i	$\frac{90}{\sqrt{60}}$
e^{π}	sen(π)	$\sqrt{2}$

La tabla 1.2 muestra algunas de las constantes m´as usadas en el ´area de la ingenier´ıa.

Tabla 1.2 Constantes utíles

pi	3.141592653589793.
N´umero imaginario **i**	$\sqrt{-1}$
j	Igual que el n´umero imaginario **i**
eps	Punto flotante con precisi´on relativa, $\varepsilon = 2^{-52}$
realmin	El n´umero en punto flotante m´as pequeño, 2^{-1022}
realmax	El n´umero m´as grande en punto flotante, $(2-\varepsilon)^{1023}$
inf	Infinito ∞
NaN	No es un n´umero

Formato num´erico

La forma para desplegar funciones, variables y cualquier tipo de dato por **MATLAB** se realiza a trav´es del comando format la cual controla el formato num´erico de los valores desplegados. Es decir, modifica el n´umero de d´ıgitos para el desplegado de los datos. Este comando solo afecta a los n´umeros que son desplegados, no al proceso de c´omputo num´erico o al registro de las variables o datos.

```
fx >>  format short ←
fx >>  x= [ 9/8 8.3456e-8] ←
```

```
x=

    1.1250    0.0000
```

Observe que en el caso de la constante 8.3456e-8 lo despliega con 0.0000 debido a los l´ımites del formato corto format short.

Ahora note la diferencia con format short e

```
fx >>  format short e ←
fx >>  x ←
       x=

    1.1250e+000    8.3450e-008
```

Es posible eliminar el formato extendido para desplegar el resultado de 9/8 y al mismo tiempo mantener la caracter´ıstica de desplegado para la segunda componente de la variable x de la siguiente forma:

```
fx >>  format short g ←
fx >>  x ←
       x=

    1.1250    8.3450e-008
```

Para desplegar la informaci´on de x con mayor n´umero de d´ıgitos sin utilizar la notaci´on cient´ıfica, entonces

```
fx >>  format long ←
       x=

    1.125000000000000    0.000000083450000
```

Para saber qu´e tipo de format est´a actualmente activo se usa el comando get:

```
fx >>  get(0,'format') ←
       ans=

    long
```

El formato numérico que por default emplea **MATLAB** es el formato corto (short): format short. Sin embargo, es posible utilizar otros tipos de formatos como: extendido, notación científica, punto fijo, punto flotante, formato de ingeniería, etc. Los formatos numéricos que contiene el comando format se encuentran resumidos en la tabla 1.3.

Tabla 1.3 Opciones del comando format

Tipo	Descripción
short	Formato por default. Despliega 4 dígitos después del punto decimal. Por ejemplo 3.1416
long	Formato para punto fijo con 15 dígitos después del punto decimal para variables tipo doble y 7 dígitos para variables tipo simple. Por ejemplo x=4.123456789012345
ShortE	Formato para punto flotante con 4 dígitos después del punto decimal. Por ejemplo: 1.4567e+000. Variables enteras que almacenan números flotantes con más de 9 dígitos no serán desplegados en la notación científica.
longE	Punto flotante con 15 dígitos después del punto decimal para datos tipo doble y 7 dígitos después del punto decimal para variables tipo simple. Ejemplo x=4.123456789012345+e000. Variables enteras que almacenan números flotantes con más de 9 dígitos no serán desplegados en notación científica.
shortG	Formato para punto fijo o punto flotante, 4 dígitos después del punto decimal. Ejemplo 3.1416.
longG	Formato para punto flotante o fijo con 14 o 15 dígitos después del punto decimal para variables tipo doble y 6 o 7 dígitos después del punto decimal para variables tipo simple. Ejemplo 8.01234567891234.
shortEng	Formato para ingeniería tiene 4 dígitos después del punto decimal y una potencia que es múltiplo de tres. Ejemplo 3.1416e+000.
longEng	Formato para ingeniería tiene 16 dígitos después del punto decimal y una potencia que es múltipo de tres. Ejemplo 3.14159265358979e+000.

El comando format tambi´en proporciona representaci´on hexadecimal, tipo proporci´on y de tipo bancaria como se explica en la tabla 1.4.

Tabla 1.4 Opciones del comando format

Tipo	Descripci´on
rat	Despliega en forma de proporci´on de n´umeros enteros. Ejemplo 17/50
hex	Representaci´on en formato hexadecimal. Ejemplo a0fcbdf34
bank	Despliega tipo cantidad bancaria (pesos y centavos). Ejemplo 3.14

Operadores

Los operadores en **MATLAB** juegan un papel determinante ya que manipulan a las variables y funciones. Adicional a los operadores aritm´eticos de la tabla 1.1, hay operadores espec´ificos para funciones, operaciones l´ogicas, relacionales, estructuras de datos, uniones y matrices.

A continuaci´on se presentan diversos tipos de operadores de utilidad en **MATLAB**.

Operador colon :

El operador colon **:** (dos puntos verticales) es uno de los operadores m´as importantes para programar, se emplea para diferentes formas, como por ejemplo en **MATLAB** t´ecnicamente se conoce como vectorizaci´on al proceso de generar una secuencia de n´umero usando 1:10.

```
fx >>  1:10  ←
       ans =

    1   2   3   4   5   6   7   8   9   10
```

En este caso el incremento es de uno en uno. Si se requiere un paso de incremento

específico, por ejemplo 2, entonces se procede de la siguiente forma

```
fx >>  1:2:10 ←
       ans=

    1   3   5   7   9   10
```

El valor del paso de incremento también puede ser menor que 1, por ejemplo

```
fx >>  1:0.1:2 ←
       ans=

    1   1.1   1.2   1.3   1.4   1.5   1.6   1.7   1.8   1.9   2
```

En ocasiones es conveniente que sea negativo

```
fx >>  10:-1:2 ←
       ans=

    10   9   8   7   6   5   4   3   2
```

En otras aplicaciones es útil realizar una variación desde -8 a 8 con incrementos de pasos de 0.5, por ejemplo

```
fx >>  -8:0.5:8 ←
ans=
    -8   -7.5   -7   -6.5   -6   -5.5   -5   -4.5   -4   -3..5   -3   -2.5   -2
    -1.5   -1   -0.5   0   0.5   1   1.5   2   2.5   3   3.5   4   4.5
     5   5.5   6   6.5   7   7.5   8
```

Operador ()

Los operadores paréntesis **()** son utilizados en diversas operaciones matemáticas, por ejemplo: evaluar funciones como en el caso de `y=sin(t)`. También para referenciar a elementos de matrices `A(2, 3)`. En expresiones aritméticas indican precedencia y la forma de agrupar variables `x=3*(y+j)*r-i*(4+r)`. Resultan de utilidad para evaluar condiciones lógicas `if (x>3) x=3 end`. Cuando se trata de evaluar potencias los paréntesis determinan los niveles de jerarquía para realizar operaciones desde las

m´as internas hasta las externas $(7 + (r * (x^2 + 8))^6 - j + \text{sen}(t^3)^5)^3$.

Operador semicolon ;

El operador punto y coma alineados en forma vertical o mejor conocido como semicolon **;** tiene varias funciones. Una de ellas se encuentra relacionada con el desplegar el resultado que tienen las variables, constantes, funciones o gr´aficas. Cuando se inserta al final de la expresi´on, instrucci´on o comando se inhabilita el desplegado. Por ejemplo:

```
fx >>  w=sin(pi/2); ←
fx >>
```

Si el operador **;** no se coloca al final de la instrucci´on, entonces se produce el desplegado del valor de la variable w

```
fx >>  w=sin(pi/2)  ←
       w=

     1
```

Otra funcionalidad del operador **;** es generar renglones en matrices. Por ejemplo,

```
fx >>  A=[ 19 3; 4 5 ]  ←
       A=
        19   3
         4   5
```

El operador **;** que precede al n´umero 3 y antecede al n´umero 4 genera un rengl´on de esta matriz. En este caso el primer rengl´on lo forman los n´umeros 19 y 3, y para el segundo rengl´on formado por 4 y 5. Obs´ervese que despu´es del corchete **]** no se inserta el operador **;** ya que en este caso afecta el desplegado, mientras que dentro de los corchetes **[;]** significa que genera un rengl´on en la matriz. Es decir el operador **;** tiene dos funciones muy diferentes. Por cada **;** dentro de corchetes se generar´a un rengl´on de la matriz. Es muy importante que quede claro la aplicaci´on de este operador para evitar problemas de programaci´on.

Si en el anterior ejemplo, en la matriz A no se insertara el operador **;** entonces el resultado quedaría de la siguiente forma:

```
fx >>  A=[ 19 3 4 5 ] ←
       A=

    19   3   4   5
```

Para ver más detalles del operador **;** referirse a la sección de **Matrices y arreglos**.

Operador ,

El operador coma **,** tiene más de una función en **MATLAB**. Por ejemplo, cuando se emplea en funciones indica la separación de los argumentos como en el caso de y=sin(t,x) o en w=sinh(t,x,y,z,p). Para referenciar a los elementos de una matriz se especifica el número de renglón y de la columna separados por una coma A(3,4). En matrices indica la separación de los elementos de un renglón A=[5,6,7,8; 8,3,0,2].

Operador '

El operador ' se relaciona con el manejo de datos tipo char o cadena de caracteres. También representa la transpuesta de una matriz.

En relación a caracteres se emplea de la siguiente forma:

```
fx >>  dato_char= 'a' ←
       dato_char=
```

Para almacenar una cadena de caracteres, se procede de la siguiente forma:

```
fx >>  cadena= 'una cadena de letras' ←
       cadena=

            una cadena de letras
```

El operador ' es ampliamente utilizado en funciones donde el pase de par´ametros es a trav´es de cadenas de texto como en disp('Hola, mundo...').

En matrices el operador ' representa la matriz transpuesta, es decir $A' = A^T$. Para mayor detalle de este operador en matrices ver la secci´on de **Matrices y arreglos**.

Operador %

En el lenguaje de **MATLAB** se pueden insertar comentarios usando el operador **%**. Los comentarios son importantes en todo programa ya que permiten la documentaci´on t´ecnica del algoritmo o aplicaci´on a implementar. Por ejemplo,

```
clc ; %limpia la pantalla de la ventana de comandos\
clear ; %limpia espacio de trabajo\
      ; %as´ı como todas las variables que ocupen memoria
t=0:0.1:t; %genera base de tiempo
y=sin(t); %genera el vector senoidal
plot(t,y) %gr´afica  onda senoidal
```

Operador ~

El operador tilde ~ se emplea para deshabilitar una variable de salida de una funci´on. Es muy útil cuando la funci´on retorna m´as de una variable y no se requiere usar todas las variables; sup´ongase que la funci´on control_robot retorna dos variables (error y par) y la sintaxis es: [error, par]=control_robot(q). No se requiere usar la variable error, únicamente par, entonces se usa de la forma siguiente:

```
[~ , par]= control_robot(q).
```

Tambi´en se emplea como negaci´on en operadores l´ogicos, por ejemplo en ~ = que significa no es igual a.

En las siguientes secciones se ir´an explicando otros tipos de operadores que se encuentran directamente relacionados con matrices y en general con la programaci´on del lenguaje de **MATLAB**.

1.5 Matrices y arreglos

En **MATLAB** una matriz es un arreglo rectangular de datos o números, tienen n renglones por p columnas; la notación matemática más común para representar a una matriz es: $A \in IR^{n \times p}$. Matrices con una sola columna o renglón significan vectores para **MATLAB**, por ejemplo $\mathbf{x} \in IR^{n \times 1}$ o $\mathbf{y} \in IR^{1 \times n}$, respectivamente. Especial significado representan los escalares cuya interpretación para propósitos de programación corresponde a una representación de matriz del tipo $IR^{1 \times 1}$.

Las entradas de la matriz se conocen como elementos y pueden ser números reales, números complejos, funciones, operadores, inclusive también pueden ser matrices de menor dimensión. Es importante destacar que una matriz es un objeto matemático por sí misma, esto es, no representa un número o un escalar. Sin embargo, en lenguaje **MATLAB** es posible trabajar a una matriz como un escalar, por ejemplo $\alpha \in IR^{1 \times 1}$. La matrices tienen operaciones y propiedades bien definidas. Las operaciones entre matrices producen una matriz, en contraste operaciones entre vectores pueden producir un escalar, vector o matriz.

En la figura 1.4 se muestra la representación clásica de una matriz rectangular de n renglones por p columnas (matriz n por p). Las columnas son arreglos verticales. Por ejemplo, la segunda columna está formada por los elementos $A(1, 2), A(2, 2), \cdots, A(n, 2)$ mientras que los renglones son filas horizontales dentro del arreglo rectangular; el segundo renglón está compuesto por $A(2, 1), A(2, 2), \cdots, A(2, p)$. Las columnas se incrementan de izquierda a derecha, mientras que los renglones se incrementan de arriba hacia abajo.

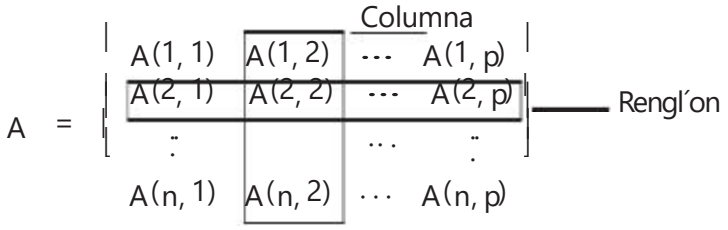


Figura 1.4 Componentes de una matriz: elementos $A(i, j)$, renglones y columnas.

Los elementos de una matriz se denotan en **MATLAB** por $A(i, j)$ donde $i = 1 \ldots n$ representa el i-ésimo renglón y $j = 1 \ldots p$ denota la j-ésima columna. Por ejemplo, en la matriz $A \in \mathbb{R}^{5\times 5}$ el elemento $A(2, 5)$ significa la componente del segundo renglón y quinta columna, el cual se muestra en la siguiente expresión:

$$A = \begin{bmatrix} A(1,1) & A(1,2) & A(1,3) & A(1,4) & A(1,5) \\ A(2,1) & A(2,2) & A(2,3) & A(2,4) & \boxed{A(2,5)} \\ A(3,1) & A(3,2) & A(3,3) & A(3,4) & A(3,5) \\ A(4,1) & A(4,2) & A(4,3) & A(4,4) & A(4,5) \\ A(5,1) & A(5,2) & A(5,3) & A(5,5) & A(5,5) \end{bmatrix}.$$

En esta matriz en particular los pivotes pueden adquirir valores para referenciar a los elementos de la matriz $A(i, j)$. Para los renglones $i = 1, 2, \cdots, 5$ y en las columnas $j = 1, 2 \cdots, 5$.

Para los propósitos de identificar los elementos de una matriz, éstos pueden ser vistos sobre un sistema de referencia cartesiano planar cuyo origen se encuentra localizado en la esquina superior izquierda. Las referencias a elementos se hace con $A(i, j)$. Es importante aclarar que dicho origen no inicia en $A(0, 0)$; lo que sería un error de sintaxis. El origen inicia con los pivotes asignados en uno, como en $A(1, 1)$; con esta asignación se hace referencia al primer elemento de la matriz, ubicado en el primer renglón y primera columna. El elemento $A(1, 1)$ representa el origen o punto de partida para los pivotes (i, j) donde el índice del i-ésimo renglón está dado antes del j-ésimo índice de la columna. Los renglones de una matriz son numerados de arriba hacia abajo y las columnas de izquierda a derecha.

Se debe tener claro que el lenguaje **MATLAB** no admite referencias a elementos de una matriz con pivotes (i, j) asignados en cero, negativos, números reales o números enteros de magnitud mayor a la dimensión de la matriz. En otras palabras, en las referencias a elementos de la matriz únicamente con números enteros o naturales dentro de los límites de la dimensión de la matriz. Por ejemplo, los siguientes elementos $A(0, 0)$, $A(0, 5)$, $A(-5, -1)$, $A(5, 0)$ son referencias inválidas de una matriz.

Es muy común cometer errores al referenciar los elementos o entradas de una matriz. Por ejemplo las entradas $A(i, j)$ de una matriz $A \in IR^{n \times p}$ deben satisfacer lo siguiente:

No usar en la referencia de las entradas de una matriz A la posición $A(0, 0)$ o números negativos $A(-3, -5)$, ni números reales como $A(1.32, 2.67)$. En todos estos ejemplos existe error de sintaxis y la ejecución de un programa no puede realizarse hasta no corregir dicho error.

Los pivotes i, j son números acotados por $0 < i \leq n$, $0 < j \leq p$. Por ejemplo si $A \in IR^{3 \times 5}$, existe un error en la referencia del elemento $A(4, 6)$, puesto que $i \leq 3$ y $j \leq 5$

Los pivotes i, j son números enteros positivos (números naturales) $i, j \in N$.

Inicializando una matriz

La matriz se inicializa por corchetes $A = []$ directamente en la ventana de comandos (Command Window) **fx** >> realizar los siguientes ejercicios:

Las entradas de una matriz puede ser de la siguiente forma: separando los elementos de un renglón por espacios en blanco o por comas:

```
fx >>  A=[2, 3, 4, 78.3, 45] ←
       A=
                    2   3   4   78.3   45
```

Es posible emplear únicamente espacios en blanco

```
fx >>  A=[2 3 4 78.3 45] ←
       A=
                    2   3   4   78.3   45
```

Combinando comas y espacios en blanco es otra posibilidad de inicializar una matriz

```
fx >>  A=[2, 3 4, 78.3 45] ←
       A=
```

```
2  3  4  78.3  45
```

Usar el operador semicolon **;** para indicar el fin del renglón y generar otro más

```
fx >>  A=[2, 3 ,4 ; 5 6  8 ; 7, 8, 9] ←
A=
     2  3  4
     5  6  8
     7  8  9
```

De tomarse en cuenta que el operador **;** además de emplearse en matrices tiene otra función cuando se inserta al finalizar una variable, constante o función desactiva la opción de desplegado.

Los elementos del i-ésimo renglón o de la j-ésima columna de una matriz A son denotados como: $A(i, j)$, la expresión $A(3, 4)$ representa un elemento de la matriz que se encuentra localizado en el tercer renglón y cuarta columna.

Considérese la siguiente matriz $A \in \mathbb{R}^{4\times 4}$

$$A = \begin{bmatrix} 1 & 3 & 3 & 2 \\ 0 & 3 & 4 & 12 \\ 12 & 34 & 1 & 23 \\ 9 & 7 & 2 & 3 \end{bmatrix}$$

Se inicializa como:

```
fx >>  A=[1,3,3,2; 0, 3, 4,12; 12,34,1,23;9,7,2,3] ←
```

Para sumar todos los elementos del cuarto renglón de A se procede de la siguiente forma: $A(4, 1) + A(4, 2) + A(4, 3) + A(4, 4) = 9 + 7 + 2 + 3 = 21$, es decir

```
fx >>  A(4,1)+A(4,2)+A(4,3)+A(4,4) ←
       ans =
```

21

El resultado es diferente a sumar $A(1, 4)+A(2, 4)+A(3, 4)+A(4, 4) = 2+12+23+3 = 40$.

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

    40
```

En general $A(i,j) = A(j,i) \circ A = A^T$. Si la matriz es simétrica, entonces se cumple $A(i,j) = A(j,i) \circ A = A^T$. En **MATLAB** el operador que representa la matriz transpuesta es '. Por ejemplo,

```
fx >>  B=A '  ←
       B=
    1   0   12   9
    3   3   34   7
    3   4    1   2
    2  12   23   3
```

Debe tenerse cuidado con el empleo del operador ', además de su uso en matrices también se emplea en cadenas de caracteres.

El operador colon **:** se puede emplear en matrices para referenciar a una porción de la matriz. Por ejemplo, sea una matriz $A \in IR^{5\times 5}$

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}$$

entonces las referencias $A(1:5,1)$, $A(1:5,2)$, $A(1:5,3)$, $A(1:5,4)$, $A(1:5,5)$ representan los 5 renglones de las columnas $j = 1 \cdots 5$, respectivamente

$$A(1:5,1) = \begin{bmatrix} 1 \\ 6 \\ 11 \\ 16 \\ 21 \end{bmatrix} \quad A(1:5,2) = \begin{bmatrix} 2 \\ 7 \\ 12 \\ 17 \\ 22 \end{bmatrix}$$

$$A(1:5,3) = \begin{bmatrix} 3 \\ 8 \\ 13 \\ 18 \\ 23 \end{bmatrix} \quad A(1:5,4) = \begin{bmatrix} 4 \\ 9 \\ 14 \\ 19 \\ 24 \end{bmatrix}$$

$$A(1:5,5) = \begin{bmatrix} 5 \\ 10 \\ 15 \\ 20 \\ 25 \end{bmatrix}$$

Para obtener las 5 columnas de cada renglón se representa por $A(1, 1:5)$, $A(2, 1:5)$, $A(3, 1:5)$, $A(4, 1:5)$, $A(5, 1:5)$

$A(1, 1:5) = [1 \quad 2 \quad 3 \quad 4 \quad 5]$ $A(2, 1:5) = [6 \quad 7 \quad 8 \quad 9 \quad 10]$

$A(3, 1:5) = [11 \quad 12 \quad 13 \quad 14 \quad 15]$ $A(4, 1:5) = [16 \quad 17 \quad 18 \quad 19 \quad 20]$

$A(5, 1:5) = [21 \quad 22 \quad 23 \quad 24 \quad 25]$

Generando matrices básicas

Existen varias formas para generar matrices básicas en **MATLAB**, se pueden generar con funciones específicas, con funciones definidas por el usuario, o con el uso del operador colon :. A continuación se enlistan varias opciones para generar matrices básicas en **MATLAB**:

- Introduciendo una lista de elementos explícitos: A = [2, 3, 67; 5.3, 3.3, 2.4; 2, 9, 1]
- Cargando matrices de archivos de datos experimentales o externos en formato de columnas: matriz=load('datos.dat')
- Generando matrices usando funciones que retornan matrices: matriz=mi matriz(n,m)
- Usando funciones de **MATLAB** por ejemplo: A=zeros(3,3).

La tabla 1.5 muestra las opciones m´as comunes (funciones especiales y por asignaci´on de datos) que permiten generar matrices b´asicas.

Tabla 1.5 Funciones especiales para generar matrices b´asicas

Nombre de la funci´on	Descripci´on y caracter´ısticas
zeros(n,m)	Genera una matriz $IR^{n\times m}$ donde todos sus elementos son cero.
ones(n,m)	Genera una matriz $IR^{n\times m}$ donde todos los elementos tienen el valor 1.
rand(n,m)	Genera una matriz $IR^{n\times m}$ donde las entradas son elementos aleatorios distribuidos de manera uniforme.
randn(n,m)	Genera una matriz $IR^{n\times m}$ donde las entradas son elementos aleatorios distribuidos normalmente.
magic(n)	Retorna una matriz cuadrada $IR^{n\times n}$ donde las entradas son n´umeros enteros desde 1 hasta n^2. El n´umero n debe ser mayor o igual a 3. .
eye(n,m)	Genera una matriz rectangular $IR^{n\times m}$ donde los elementos con ´ındices (i,j) tal que $i = j$ tienen el valor de 1, y para entradas $i = j$ tienen valor de 0. Si $n = m$, entonces la matriz es cuadrada y diagonal.
A(1:n, 1:m)=α	Genera una matriz rectangular $IR^{n\times m}$ donde todos los elementos de la matriz A adquieren el valor del escalar α. Cuando $n = m$, entonces retorna una matriz cuadrada.
A=load	Genera una matriz rectangular desde un archivo ASCII o de texto load('nombre archivo.tex'); el archivo puede tener datos experimentales y asignarlos a una matriz para su procesamiento o an´alisis.

Por ejemplo, en la ventana de comandos se puede comprobar lo siguiente:

```
fx >> A=zeros(3,3) ←
A=
     0     0     0
     0     0     0
     0     0     0

fx >> B=10*ones(3,3) ←
B=
    10    10    10
    10    10    10
    10    10    10

fx >> C=randn(3,3) ←
C=
    0.5377    0.8622   -0.4336
    1.8339    0.3188    0.3426
   -2.2588   -1.3077    3.5784
```

Otra forma simple de generar matrices es por medio de la siguiente expresi´on:

```
fx >> D(1:3,1:3)=3.1416 ←
D=
    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416
```

Por medio de un archivo con extensi´on `m` es posible generar una matriz de la siguiente forma. Crear en el editor de texto de **MATLAB** un archivo con la siguiente informaci´on:

```
A=[ 8     3     7;
    6     7.3   3.1416;
    7.98  6.67  0.001];
```

salvar el archivo como datos.m **en la carpeta del usuario**, entonces sobre el prompt de la ventana de comandos teclear:

```
fx >> datos ←
      A=
          8.0000   3.0000   7.0000
          6.0000   7.3000   3.1416
          7.9800   6.6700   0.0010
```

Otra forma de generar matrices es por medio de la funci´on load la cual lee archivos binarios y de texto. Por medio de un editor de texto generar un archivo de datos de texto en arreglo rectangular separados con espacios en blanco. Por ejemplo, grabar en la **carpeta de trabajo del usuario** un archivo de texto denominado datos1.dat con la siguiente informaci´on:

```
                datos1.dat
   2.0000    3.0000    4.0000
   5.0000    6.0000    7.0000
   6.7000    8.3000    4.5450
```

Para generar la matriz con la informaci´on del archivo datos1.dat utilizar la siguiente sentencia en el prompt de la ventana de comandos:

```
fx >> A=load('datos1.dat') ←
      A=
          2.0000   3.0000   4.0000
          5.0000   6.0000   7.0000
          6.7000   8.3000   4.5450
```

Observe que con esta asignaci´on en la matriz A, la informaci´on se encuentra lista para ser procesada por cualquier operaci´on de matrices previamente vista. En rob´otica y mecatr´onica es com´un grabar archivos experimentales o resultados de simulaci´on para ser analizados con **MATLAB** usando un procedimiento parecido al planteado. El formato del archivo debe ser que los datos est´en distribuidos en renglones y columnas separadas por espacios en blanco.

Concatenación

Al proceso de unir pequeñas matrices para hacer una más grande se le denomina concatenación. Los símbolos [], mejor conocidos como corchetes, representan en **MATLAB** el operador concatenación. Por ejemplo, se puede iniciar con una matriz diagonal 4 × 4 de la siguiente forma:

```
fx >> A=eye(4,4) ←
A=
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Escribir en el prompt de la ventana de comandos la siguientes expresión para obtener una matriz de dimensión 8 × 8.

```
fx >> B=[A A+10; A+20 A+30] ←
B=
     1     0     0     0    11    10    10    10
     0     1     0     0    10    11    10    10
     0     0     1     0    10    10    11    10
     0     0     0     1    10    10    10    11
    21    20    20    20    31    30    30    30
    20    21    20    20    30    31    30    30
    20    20    21    20    30    30    31    30
    20    20    20    21    30    30    30    31
```

Borrando columnas y renglones

MATLAB permite borrar renglones y columnas de una matriz usando el operador concatenación []. Por ejemplo, para borrar la segunda columna de la matriz A previamente definida en el apartado inmediato anterior se procede de la siguiente forma:

```
fx >>  A(:,2)=[]  ←
        A=
```

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

obs´ervese que por borrar la segunda columna, la nueva matriz resultante A adquiere una dimensi´on de 4×3.

Ahora, si se desea borrar el primer renglón de la anterior matriz resultante A, entonces

```
fx >>  A(1,:)=[]  ←
        A=
```

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

resulta una matriz $A \in IR^{3\times 3}$.

> Si un elemento de una matriz se quiere borrar, resulta un error, la expresi´on $A(1,1)=[]$ es inv´alida. Sin embargo, usando sub´ındices borra un simple elemento o una secuencia de elementos de la matriz y rehace los elementos restantes de la matriz en un vector renglón.

Por ejemplo, para borrar el primer elemento de A se emplea la siguiente expresi´on

```
fx >>  A(1:1)=[]  ←
        A=
```

$$\begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{matrix}$$

N´otese que los elementos restantes se quedan arreglados en la forma de un vector

renglón. Si en lugar de borrar el primer elemento, se desea borrar los 5 primeros elementos, entonces se procede de la siguiente forma:

```
fx >>  A(1:5)=[]  ←
       A=

    0   0   0   1
```

Expansión de escalares

Los escalares pueden ser combinados con algunas operaciones con matrices. Por ejemplo, en la sustracción un escalar y una matriz A puede ser combinados como A-3.3 de la siguiente manera:

```
fx >>  A=[1 2; 5 6];  ←
fx >>  B=A-3.3  ←
       B=

   -2.3000   -1.3000
    1.7000    2.7000
```

es necesario resaltar que la expresión A-3.3 no produce error, es una operación válida de acuerdo a la sintaxis del lenguaje de **MATLAB**.

Además debe tomarse en cuenta que las operaciones entre escalares y matrices respetan la propiedad conmutativa. Es decir, A-3.3=-3.3+A. De manera similar para la suma, división y multiplicación entre escalares y matrices: A+3.3=3.3+A, A*3.3=3.3*A, A/3.3=1/3.3*A.

Usando la expansión de escalares, **MATLAB** asigna un escalar específico a todos los índices para el rango indicado en la matriz. Por ejemplo, la siguiente expresión genera una matriz de dimensión 3×3 donde todos sus elementos tienen asignado el valor 9:

```
fx >>  B(1:3,1:3)=9  ←
```

Si la matriz ya existe, entonces esta expresión solo asigna el valor del escalar 9 a

los elementos de la matrices (renglones y columnas especificados) de acuerdo a los ´ındices indicados. Si la matriz no existe, entonces genera una nueva matriz en este caso de dimensi´on 3 × 3 donde todos sus elementos son iguales a 9.

Operaciones b´asicas con matrices

Hay varias operaciones elementales entre matrices que se realizan con frecuencia en mecatr´onica y rob´otica. Hay que considerar que se deben respetar las reglas del ´algebra de matrices. Por ejemplo, se debe satisfacer la compatibilidad entre el n´umero de renglones y columnas de la matriz A con la matriz B:

1) Declarar las matrices por ejemplo: A=[2 3 4; 6 7 8; 9 0 9] y B=[3 0 1; 5 2 4; 7 8 1]

2) Usar el operador + para realizar la suma: A+B

3) La sustracci´on se realiza con: A-B

4) La multiplicaci´on: A*B (el n´umero de columnas de la matriz A debe ser igual al n´umero de renglones de la matriz B).

5) En general la multiplicaci´on no es conmutativa: B*A (el n´umero de columnas de la matriz B debe ser igual al n´umero de renglones de la matriz A). Cuando las matrices son diagonales, entonces se satisface la propiedad de conmutatividad de la multiplicaci´on.

6) El determinante de una matriz se obtiene como: det(A)

7) La inversa de una matriz existe si su determinante es diferente a cero: inv(A)

Debido a que las expresiones de programaci´on van creciendo, es conveniente que el lector en lugar de seguir utilizando la ventana de comandos, utilice el editor de texto para grabar todas las operaciones que realiza en nombre archivo.m (M-files) el cual almacenar´a instrucciones, variables, operadores, comentarios, funciones, etc. De esta forma el archivo queda grabado en el directorio del usuario y se puede reproducir sin ning´un problema. Por lo tanto, de aqu´ı en adelante ser´a de utilidad emplear el editor de texto que se encuentra integrado en el ambiente de programaci´on de **MATLAB**.

♣ Ejemplo 1.1

Escribir un programa en **MATLAB** para realizar operaciones b´asicas entre matrices (adici´on , sustracci´on, inversa, sim´etrica, antisim´etrica, etc). Considere las siguientes matrices $A, B \in \mathbb{R}^{3\times 3}$:

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 9 & 0 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} 3 & 0 & 1 \\ 5 & 2 & 4 \\ 7 & 8 & 1 \end{bmatrix}$$

Soluci´on

El cuadro de c´odigo fuente 1.1 contiene las operaciones m´as importantes que se realizan en mecatr´onica y rob´otica usando matrices. Edite el programa que se muestra a continuaci´on; se recomienda salvarlo como cap1_matrices1.m; y ejec´utese oprimiendo la tecla F5 o realizando click sobre el icono play que se encuentra localizado en la parte superior del editor de texto.

Recuerde que para visualizar los resultados de las operaciones que le interesan al final de la l´ınea correspondiente elimine el operador **;**.

C´odigo Fuente 1.1 Operaciones b´asicas con matrices

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo cap1_matrices1.m
```

Operaciones b´asicas con matrices

```
%se definen las matrices de prueba
A=[ 2 3 4;
    6 7 8;
    9 0 9];
 B=[ 3 0 1;
    5 2 4;
    7 8 1];
A+B; % suma de matrices
A-B; %resta de matrices
%en general la multiplicaci´on no es conmutativa
A*B;
B*A;
A' % transpuesta de la matriz A
(A')' % transpuesta de la transpuesta = A
As=(A+A')/2; % parte sim´etrica
Ask=(A-A')/2; % parte antisim´etrica
As+Ask % matriz original A
A'*A %obtiene una matriz sim´etrica
A^(-1); %inversa de la matriz
det(A) % determinante
inv(A); %inversa de la matriz A
A*inv(A);
inv(A)*A; % matriz identidad
eig(A) % valores propios de la matriz A
norm(A) % norma de la matriz A
p=poly(A) %retorna coeficientes del polinomio caracter´ıstico de A
r=root(p); %retorna ra´ıces del polinomio caracter´ıstico de A
```

Arreglos

Las operaciones aritm´eticas con arreglos o registros son realizadas elemento por elemento. Es decir, la adici´on y sustracci´on son las mismas para arreglos y matrices, pero la multiplicaci´on es diferente. **MATLAB** usa como parte de la notaci´on de multiplicaci´on con arreglos un punto decimal.

La tabla 1.6 muestra la lista de operadores que se utilizan en operaciones con arreglos.

Tabla 1.6 Operadores de arreglos

Suma	**+**
Sustracci´on	**-**
Multiplicaci´on elemento por elemento	**.***
Divisi´on elemento por elemento	**./**
Divisi´on izquierda elemento por elemento	**.**
Elevar a una potencia elemento por elemento	**.^**
Arreglo transpuesto	**.'**

Como ejemplo ilustrativo de estos operadores considere lo siguiente:

```
fx >> A=[1 3; 4 5] ←
A=
     1   3
     4   5

fx >> A.*A ←
ans=
```

```
     1     9
    16    25
```

observese que cada elementos se multiplica por sí mismo, es decir los elementos de la matriz resultante son el cuadrado de cada elemento. Lo mismo sucede para los demás operadores:

```
fx >>  A./A  ←
       ans=

     1     1
     1     1

fx >>  A.^A  ←
       ans=

     1    27
   256  3125
```

Las operaciones con arreglos son muy útiles para construir tablas. Por ejemplo:

```
fx >>  x=(0:5)'  ←
       x=

     0
     1
     2
     3
     4
     5

fx >>  [n, m]=size(x);  ←
       n=

     6

fx >>  tabla=[x x.^2 x.^n]  ←
       tabla=
```

0	0	0
1	1	1
2	4	64
3	9	729
4	16	4096
5	25	15625

1.6 Gr´aficas

La representaci´on gr´afica y visualizaci´on de resultados es una herramienta muy importante en el an´alisis y diseño de esquemas de control y comportamiento din´amico. Permite exhibir como el robot o el sistema mecatr´onico responde en sus etapas transitoria y estacionaria cuando se encuentra bajo la acci´on de control de una estructura matem´atica determinada. **MATLAB** tiene varios tipos de gr´aficas (plots): incluye desde la est´andar con ejes lineales, logar´ıtmicas y semi-logar´ıtmicas, barras, polares, en el espacio tridimensional (3D), contornos de superficies entre otras. Las gr´aficas se pueden formatear para tener una apariencia espec´ıfica, es decir con un tipo de l´ınea y colores. Tambi´en es posible utilizar una sola ventana con varias gr´aficas, as´ı como t´ıtulos y letreros sobre los ejes.

Para inicializar ventanas espec´ıficas donde se despliegue resultados gr´aficos se emplea la funci´on `figure`; la cual crea figuras y objetos gr´aficos, para ser utilizados por funciones como `plot`, `subplot`, `plot3`, `mesh`, etc. La informaci´on completa y detallada de las funciones gr´afica se puede consultar en helpwin, help plot, help figure, etc. Sin embargo, a continuaci´on se describe algunas funciones que de manera particular se emplean en rob´otica y mecatr´onica.

plot

La funci´on `plot` es utilizada para generar una gr´afica en dos dimensiones:

```
plot(t,x)
```

Los argumentos `t,y` son vectores (arreglos de una dimensión), ambos vectores deben tener el mismo número de elementos o dimensión. Cuando el comando `plot` se ejecuta genera automáticamente una simple curva de los datos de la variable x (eje vertical o eje de ordenadas) vs `t` (eje horizontal o eje de las abscisas). Con la función `plot` también se puede graficar diagramas de fase, respuesta transitoria y estacionaria de robots manipuladores. De esta forma en forma gráfica es más fácil la interpretación de las variables de estado.

♣ Ejemplo 1.2

Escribir un programa en **MATLAB** para graficar la curva de la siguiente función:

$$y = 9x^5 + 3\,\text{sen}(x^3)$$

para el intervalo $x \in [-10, 10]$.

Solución

El código fuente que se presenta en el cuadro 1.2 contiene la forma de obtener la evolución en el tiempo de una función exponencial requerida para el intervalo $x \in [-10, 10]$. En la línea 1 se emplean las instrucciones `clear all; close all; clc;` requeridas para liberar cualquier variable, figura, o archivo que ocupe espacio de memoria.

En la línea 3 se define el intervalo de la función con desplazamiento de una milésima. La función a graficar $y = 9x^5 + 3\,\text{sen}(x^3)$ se encuentra implementada con operaciones de arreglos.

En esta parte del código se destaca la ventaja que ofrece emplear operaciones con arreglos, debido que la variable x es definida como un vector de un renglón y n columnas, entonces la forma más adecuada de realizar las operaciones con los exponentes de la función y es a través de la manipulación de datos usando arreglos.

Finalmente, en la línea 5 se grafica la función $y = 9x^5 + 3\,\text{sen}(x^3)$ por medio de `plot(x,y)` tal y como se muestra en la figura 1.5.

C´odigo Fuente 1.2 Gr´afica de la funci´on $9x^5 + 3\ \mathbf{sen}(x^3)$

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo cap1_grafica.m
```

Gr´afica de la funci´on $9x^5 + 3\ \mathrm{sen}(x^3)$

```
1 clear all; close all; clc;
2 %vector del eje de ordenadas
3 x=-10:0.001:10;%intervalo de graficado
4 y=9*x.^5+3*sin(x.^3); y es el vector de las abscisas
5 plot(x,y)
```

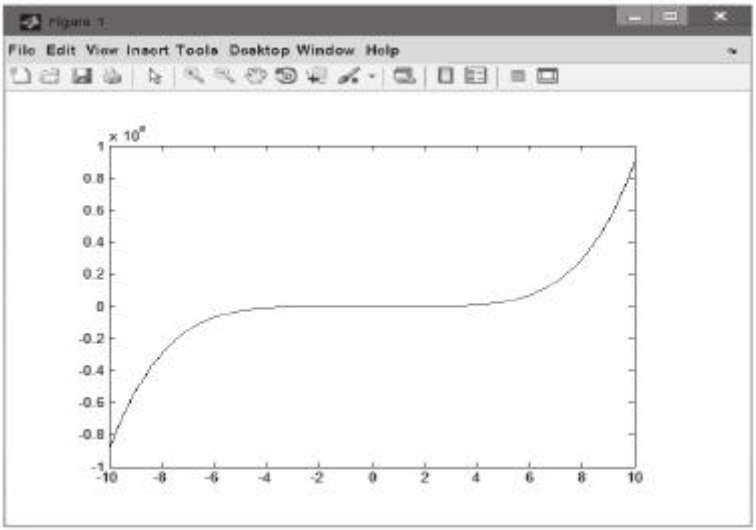


Figura 1.5 Gr´afica de $9x^5 + 3\ \mathrm{sen}(x^3)$.

Mayor informaci´on de la funci´on `plot` se puede consultar en:

fx >> `help plot` ←

fplot

El comando fplot grafica una función de la forma y = f(x) entre límites específicos.

La sintaxis del comando fplot es la siguiente:

```
fplot('function',limits,line)
```

- function: es la función a graficar. Puede ser escrita como una cadena de caracteres entre comillas. La función puede ser propia de **MATLAB** (built-in) o definida por el usuario (M-file).
- La función no puede incluir variables previamente definidas.
- limits: puede ser de dos formas: la primera forma incluye un vector que especifica el incio y término del intervalo a graficar (el dominio de x): [xmin, xmax]. La segunda forma cosiste de 4 parámetros [xmin,xmax,ymin,ymax], es decir los 2 primeros parámetros indican el dominio de x, mientras que los 2 últimos parámetros representan los límites del eje de las abscisas (eje vertical y).
- line: indica especificaciones del tipo de línea son las mismas que el comando plot.

♣ Ejemplo 1.3

Escribir un programa en **MATLAB** para graficar en línea la función:

$$y = 9x^5 + 3\,\text{sen}(x^3) \quad x \in [-10, 10].$$

Solución

El programa 1.3 contiene el código para obtener la gráfica de la función solicitada, reproduciendo la misma gráfica de la figura 1.5.

C´odigo Fuente 1.3 Gr´afica de '9*x^5+3* sen(x^3)'

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo cap1_exponencial.m
```

Gr´afica de '9*x^5+3* sen(x^3)'

```
1 clear all; close all; clc;
2 fplot('9*x^5+3* sen(x^3)',[-10,10])
```

subplot

La funci´on `subplot(m,n,p)` divide una ventana en una matriz de $m \times n$, p indica la posici´on de la gr´afica dentro de esa ventana.

plot3

La funci´on `plot3(x,y,z)` grafica las coordenadas en el espacio tridimensional.

mesh

La funci´on mesh genera superficies en el espacio tridimensional especificadas por las coordenadas x, y, z. Un aspecto interesante de esta funci´on es que la superficie de la funci´on la presenta con estilo de malla para resaltar el aspecto tridimensional. Hay otras funciones para graficar superficies que se auxilian de mesh; como es el caso de la funci´on ezmesh.

♣ Ejemplo 1.4

Escribir un programa en **MATLAB** para graficar funciones en 3D

Soluci´on

El programa que se presenta en el cuadro 1.4 muestra la forma de usar las funciones

3D. Con el uso de la funci´on figure (l´ınea 3) se abre una ventana para indicar las gr´aficas generadas con subplot(2,2,1)...subplot(2,2,4). Observe la diferencia de presentaci´on entre las funciones surf(x,y,z) y plot3(x,y,z); la primera tiene un aspecto de s´olido, mientras que la segunda usa l´ıneas suaves (ver figura 1.6).

C´odigo Fuente 1.4 Funciones 3D.

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo cap1_graficas3D.m
```

Funciones 3D.

```
1  clc; clear all; close all;
2  t=0:pi/10:2*pi;
3  figure
4  [x,y,z]=cylinder(4*cos(t));
5  subplot(2,2,1); mesh(x,y,z)
6  subplot(2,2,2); ezmesh('x.*exp(-x.^2-y.^2)',40);
7  subplot(2,2,2); ezmesh(fh,40)
8  [x,y,z] = sphere;
9  subplot(2,2,3); plot3(x,y,z)
10 subplot(2,2,4); surf(x,y,z)
```

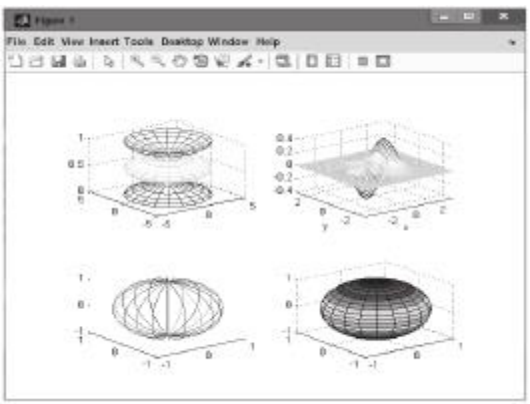


Figura 1.6 Gr´aficas en 3D.

1.7 Funciones

Una de las principales caracter´ısticas de **MATLAB** es que provee un n´umero muy grande de funciones matem´aticas. La lista completa de funciones matem´aticas se puede consultar en la ventana de comandos escribiendo:

```
fx >> help elfun ←
```

Una lista de funciones matem´aticas avanzadas se puede ver en:

```
fx >> help specfun ←
fx >> help elmat ←
```

La gran mayor´ıa de las funciones matem´aticas acepta como argumento n´umeros imaginarios, de esta forma cuando se calcula por ejemplo la ra´ız cuadrada o logaritmo de un n´umero negativo, no produce un error, autom´aticamente produce un n´umero complejo o imaginario.

Algunas funciones tales como `sqrt`, `sin`, `cos` son funciones propias (built in) de **MATLAB** ya que contienen un c´odigo muy eficiente, entonces no se encuentran disponibles los detalles computacionales y c´odigo fuente. Otras funciones como `gamma`, `sinh` son implementadas en archivos (M-files), por lo que el c´odigo fuente est´a disponible, y adem´as se puede modificar.

Las funciones pueden generar o retornar `infinity` cuando ocurre una divisi´on entre cero o tambi´en por generar un c´alculo matem´atico m´as all´a del l´ımite permitido por la variable realmax, lo cual se le conoce t´ecnicamente como overflow. Como ejemplos de este escenario considere los siguientes ejercicios:

```
fx >> z=pi*exp(log(realmax)) ←
      z =
           Inf
fx >> z=cos(0)/sin(0) ←
      z =
           Inf
```

```
fx >>  z=cosh(12e10) ←
       z =
            Inf
```

Los nombres de las funciones no son reservados, es decir se puede utilizar para asignarlo como nombre de una nueva variable, por ejemplo:

```
sin=1.2345
```

La función original `sin` puede ser restaurada con

```
clear sin
```

Funciones en línea

Cuando se requiere simples cálculos matemáticos pueden ser usarse las funciones en línea (inline functions).

La funciones en línea pueden ser generadas con el comando `inline('expresión matemática en forma de cadena')`. Por ejemplo:

```
fx >>  z=inline('x/(1+x^2'))  ←
       z =
            inline function:

             z(x)=x/(1+x^2)
```

Entonces la función `z(x)` se puede evaluar numéricamente

```
fx >>  z(2)  ←
       ans
            0.4000
```

Otra opción para evaluar funciones en línea es por medio del comando `feval`

```
fx >>  feval('sqrt', 100)  ←
       ans
            10
```

El comando feval también puede evaluar funciones definidas por el usuario.

Cuando se requiere de complicadas funciones matemáticas, programación extensiva, subprogramas, etc., lo adecuado son funciones archivo (M-file functions).

Funciones archivo

Las funciones archivo (M-file function) aceptan datos como argumentos de entrada y retornan resultado a través de la variable de salida. Las funciones archivo tienen una estructura sintáctica muy bien definida. Emplean la palabra reservada function para indicar que es una función. En seguida, se especifica el nombre de la variable de salida (por ejemplo w), seguida del signo =, a continuación del nombre de la función y entre paréntesis los argumentos de entrada separados por comas. Posteriormente viene el grupo de instrucciones que contenga la función, la variable de salida tiene el conjunto de variables a retornar (w=[var1; var2]). Finalmente se inserta el delimitador end.

La sintaxis de una función se ilustra en el código 1.1.

Estructura de código 1.1

Sintaxis de una función

```
function w=nombrefuncion(x1, x2, x3,...,xn)
    var1=grupo de instrucciones1;
    var2=grupo de instrucciones2;
    w=[var1; var2];
end
```

Usando el editor de textos integrado en **MATLAB** se pueden desarrollar funciones como la que se muestra en el cuadro 1.5.

Una función archivo M-file function tiene que ser salvada antes que pueda ser usada. Es recomendable salvar al archivo con el mismo nombre de la función y verificar que tenga extensión m.

C´odigo Fuente 1.5 norma euclidiana de un vector

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo normaev.m
```

norma euclidiana de un vector

```
1 function y =normaev(x)
2     %Norma euclidiana de un vector x ∈ IR^n
3     %hay varias formas para calcular la norma euclidiana
4     % x = sqrt(x21 + x22 + ... + xn^2)
5     % x = sqrt(x^T x)
6     % x = sqrt(x21 + x22 + ... + xn^2) = sqrt(sum_{i=1}^{n} x_i^2)
7     %en MATLAB se calcula como:
8     y=norma(x,2);
9 end
```

N´otese que se utiliza la palabra reservada function para definir una funci´on, despu´es se indica la variable donde regresa el valor (en este caso es la variable y), posteriormente contin´ua el signo igual (=) y el nombre de la funci´on definida por el usuario normaev entre par´entesis el argumento de entrada x. Al finalizar el c´odigo se inserta la palabra clave end para delimitar el grupo de instrucciones que estar´a entre la palabras reservadas function y end.

MATLAB autom´aticamente ajustar´a la naturaleza del argumento de entrada, as´ı como su salida. En otras palabras, el argumento x puede ser un escalar, vector o matriz, de acuerdo al tipo de operaciones que se realizan dentro de la funci´on la naturaleza de la salida y se ajustar´a de manera adecuada. En el caso del c´odigo fuente del cuadro 1.5 el argumento puede ser un vector o matriz y la salida correspondiente es escalar. Algunas funciones pueden tener m´as de un argumento de entrada y tambi´en retornar varias variables de salida, como el caso que a continuaci´on se presenta en el cuadro 1.6.

Código Fuente 1.6 cinemática directa robot de 2 gdl

```
%MATLAB Aplicado a Robótica y Mecatrónica
%Capítulo 1 Introducción
%Editorial Alfaomega Fernando Reyes Cortés
%Archivo cinematica_directa_robot2gdl.m
```

cinemática directa robot de 2 gdl

```
1 function [y, x]=cinematica_directa_robot2gdl(l1,l2,q1,q2)
2     % convierte variables articulares a coordenadas cartesianas
3     x=l1*cos(q1)+l2*cos(q1+q2);
4     y=l1*sin(q1)+l2*sin(q1+q2);
5 end
```

Funciones anidadas

Una función anidada (nested function) es aquella función que se encuentra como parte del código de cualquier función archivo (M-file function). En el siguiente ejemplo la función B está anidada en la función A:

Estructura de código 1.2

Función anidada

```
function y=A(x1,x2)
    y=B(x1)+x2;
    function w=B(x3)
        w=x3*x3*x3;
    end
end
```

Variables locales y globales

Todas las variables que se definan en una función son locales por naturaleza, esto significa que sólo en esa función existen y no pueden ser reconocidas por alguna otra función. Aun cuando dos funciones tengan variables con el mismo nombre, dichas variables sólo son reconocidas por sus respectivas funciones.

Sin embargo, es posible lograr que una o m´as variables comunes sean reconocidas por varias funciones (variables globales) listando el nombre de las variables y separadas por espacios en blanco en el comando global (global command) sin comas intermedias. Usar la siguiente declaraci´on:

```
global X Y Z
```

- Las variables han sido declaradas globales en cada funci´on que el usuario quiere que sean reconocidas. Las variables ser´an comunes s´olo en esas funciones.
- El comando global debe aparecer antes que la variable sea usada. Es recomendable usar la declaraci´on para ese comando al inicio del archivo.
- Las variables globales est´an separadas por espacios en blanco, si usar comas y se recomienda que al final de la l´ınea no se inserte el operador **;**.
- El valor de las variables globales puede ser modificado, asignado o reasignado en cualquiera de las funciones comunes.
- Se debe tener mucho cuidado en el empleo de variables globales. Es recomendable usar letras may´usculas o alg´un nombre que el usuario puede identificar f´acilmente para las variables globales con la finalidad de no confundirlas con variables regulares o locales.

Comparaci´on entre archivos scripts y archivos funciones

Con la finalidad de entender exactamente la diferencia entre archivos script y funciones archivo (M-file function) se enlistan a continuaci´on las principales caracter´ısticas entre ellos:

- Ambos archivos: script y funciones son salvados con la misma extensi´on .m por lo cual se conocen como M-files.
- La primera l´ınea de c´odigo en el archivo funci´on es la l´ınea de definici´on de una funci´on. Ejemplo: function y=reloj(t)


Las variables dentro de una funci´on son locales.


Las variables en un archivo script son reconocidas en la ventana de comandos.


Los archivos script contienen una secuencia de comandos de **MATLAB**.


Los archivos funciones pueden aceptar datos a trav´es de la entrada de sus argumentos y pueden retornar datos por medio de sus variables de salida.


Cuando una funci´on archivo es salvada, entonces el nombre del archivo es el mismo que el nombre de la funci´on.


Los nombres de los archivos script pueden tener cualquier nombre ya que no tienen funciones.

Usando una funci´on archivo

Una funci´on definida por el usuario en un archivo funci´on (M-file function) se usa en la misma forma que una funci´on propia (built-in function) de **MATLAB**. Esto significa que la funci´on puede ser llamada desde la ventana de comandos (Command Window), desde un archivo script o desde cualquier funci´on. Si la funci´on del usuario se encuentra en alg´un directorio diferente al de trabajo de **MATLAB**, entonces debe ser habilitado dicho directorio con la opci´on add to path (ver p´agina 9).

♣ ♣ Ejemplo 1.5

Escribir un programa principal para graficar la curva de un c´ırculo de radio $r = 0.25$ m, con centro en $x_0 = 0.3$m, $y_0 = 0.3$m

$$x = x_0 + r\ \text{sen}(t) \qquad y = y_0 + r\cos(t) \qquad z = t$$

donde t la variable tiempo.
Asimismo implementar la ecuaci´on del c´ırculo en una funci´on.m; utilizar variables globales para almacenar la coordenada z.

Solución

La función círculo se encuentra descrita en el cuadro 1.7, los parámetros de entrada de esta función son las coordenadas del centro del círculo x_0, y_0, el radio r y el vector tiempo. Las variables de salida son las coordenadas cartesianas x, y del círculo. En la línea 2 se encuentra definida la variable global `altura`, para almacenar las coordenadas sobre el eje z. Esta misma variable global `altura` se encuentra en el programa principal 1.8 (ver la línea 3). La función círculo y el programa principal pueden reconocer a la variable `altura`, lo que representa una ventaja debido a que cualquier de los dos programas puede manipular datos de esa variable.

Código Fuente 1.7 Función círculo

```
%MATLAB Aplicado a Robótica y Mecatrónica.
%Editorial Alfaomega, Fernando Reyes Cortés.
%Capítulo 3 Cinemática función circulo.m
```

Función círculo

```
1 function [x, y]=circulo(x0,y0,r,t)
2     global altura % variable global que se emplea en el programa principal 1.8
3     %ecuación del círculo
4     x=x0+r*sin(t);
5     y=y0+r*cos(t);
6     %coordenada sobre el eje z
7     altura=t;
8 end
```

Observe que el programa principal 1.8 no es una función, puesto que no emplea la palabra reservada `function`, entonces se puede salvar el código de este programa con cualquier nombre, añadiendo la extensión `m`; es decir: (nombre archivo.m).

Para ejecutar el programa principal se puede hacer con la tecla F5 o realizando click sobre el icono play localizado en el menú superior del editor de texto. Las gráficas de `x vs t`, `y vs t` y (x ,y , z) se obtienen con las funciones `subplot`, `plot` y `plot3` (líneas 9 a la 13) cuyo resultado se encuentra en la figura 1.7.

C´odigo Fuente 1.8 Programa principal del c´ırculo

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%archivo circulo_simu.m
```

Programa principal del c´ırculo

```
1  clc; clear all; close all;  %limpia a´rea de memoria
2  format short
3  global altura % declaraci´on de la variable global
4  ti=0; %tiempo inicial
5  h=0.001; %incremento de una mil´esima de segundo
6  tf = 10; %tiempo final
7  t=ti:h:tf; %vector tiempo
8  %se invoca a la funci´on c´ırculo
9  [x, y]=circulo(0.3,0.3,0.25,t);
10 subplot(2,2,1); plot(t,x) % gr´afica de x vs t
11 subplot(2,2,2); plot(t,y) %gr´afica de y vs t
12 subplot(2,2,3); plot(x,y) % c´ırculo
13 subplot(2,2,4); plot3(x,y,altura)%figura tridimensional
```

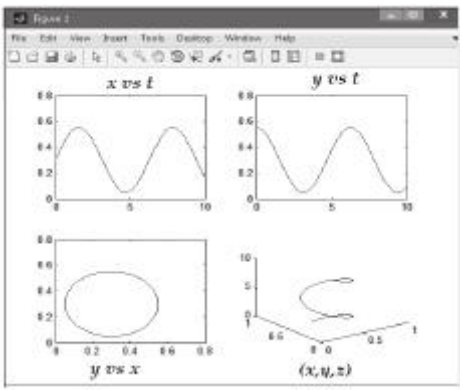


Figura 1.7 C´ırculo.

1.8 Programaci´on

Un programa es una secuencia de instrucciones, expresiones, funciones, comandos y declaraciones para realizar aplicaciones en ingenier´ıa mecatr´onica y rob´otica. En **MATLAB** las instrucciones son ejecutadas una tras otra, en forma secuencial, es decir en el mismo orden como van apareciendo. La secuencia del programa depende de las instrucciones que controlan el flujo del programa, dependiendo de ciertas condiciones se ejecuta un determinado conjunto de instrucciones o bloque del programa.

Dentro de las condiciones que determinan el flujo del programa se encuentran operadores l´ogicos como los que se presentan en la tabla 1.7. A trav´es de ellos se eval´ua expresiones para realizar un cierto bloque de c´odigo o funciones espec´ıficas de la aplicaci´on implementada.

Tabla 1.7 Operadores l´ogicos

Igual a	==
No es igual a	~=
Menor que	<
Mayor que	>
Menor que o igual que	<=
Mayor que o igual que	>=

El lenguaje de programaci´on **MATLAB** es muy rico en instrucciones y funciones toolbox, ya que cuenta con un amplio compendio de librer´ıas o funciones para diversas ´areas de ingenier´ıa y ciencias exactas.

A continuaci´on se describen las instrucciones condicionales y lazos de programas como **if, for,** while, switch que permiten programar una gran cantidad de aplicaciones en rob´otica y mecatr´onica.

Instrucciones condicionales

Una instrucci´on condicional permite a **MATLAB** realizar decisiones para ejecutar un grupo de funciones, comandos, etc. Si la condici´on en la instrucci´on condicional es verdadera, entonces realiza un grupo de expresiones del programa. Si la condici´on es falsa, entonces no ejecuta o salta ese grupo de expresiones.

if

La instrucci´on **if** eval´ua si una expresi´on l´ogica es verdadera, entonces realiza un conjunto de instrucciones o declaraciones del programa, de otra forma realiza la ejecuci´on de otro bloque del programa. La estructura de esta instrucci´on condicional consta de la palabra clave **if**, un conjunto de instrucciones o bloque de programa delimitadas por la palabra clave end.

Si la condici´on l´ogica que se eval´ua por la instrucci´on **if** es falsa, el programa salta ese bloque de instrucciones y contin´ua con las instrucciones que se encuentran justo despu´es de end.

La estructura de la instrucci´on condicional **if** se muestra en el c´odigo 1.3.

Estructura de c´odigo 1.3

Sintaxis de la instrucci´on if — end

```
if condicion_verdadera_if
   instrucciones_1;
   ---------------
   instrucciones_n;
end
otro grupo de instrucciones;
```

La figura 1.8 muestra el diagrama de flujo de la instrucci´on **if**.
Es importante resaltar que en el entorno de programaci´on de **MATLAB** las palabras claves **if** end aparecen en color azul. Adem´as, las expresiones que aparecen como el bloque de instrucciones de la instrucci´on **if** aparecen como texto con sangr´ıa

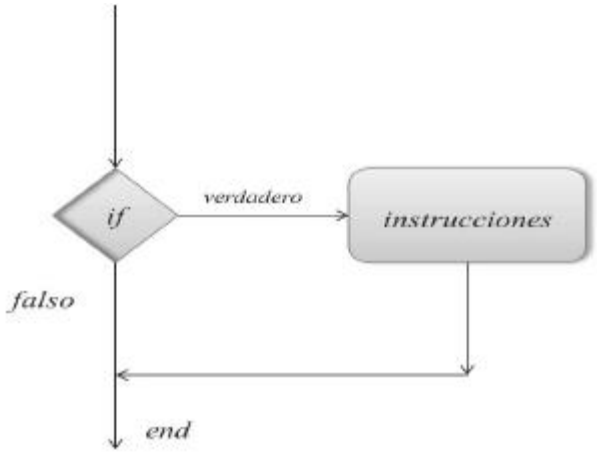


Figura 1.8 Diagrama de flujo de la instrucci´on **if**.

(indented), lo que resulta una f´acil lectura del c´odigo de programaci´on.

if, else, elseif

La instrucci´on **if** puede tener m´as opciones dentro de su estructura **if...end**. Es decir, se puede utilizar **if** con la palabra clave **else** para evaluar otro grupo de instrucciones de la manera que se muestra en la estructura de c´odigo 1.4:

Estructura de c´odigo 1.4

Sintaxis de if...else...end

```
if condici´on verdadera
    grupo de instrucciones;
    else
    grupo de instrucciones;
end
```

Adem´as de la opic´on **else** en la instrucci´on **if...end**, tambi´en se puede combinar con la palabra clave **elseif**; en este caso incluye una declaraci´on condicional adicional para la ejecuci´on de m´as grupos de instrucciones. Si en la instrucci´on **if** la condici´on es verdadera, entonces el programa ejecuta el grupo (1) de instrucciones

y pregunta por una nueva expresi´on condicional **elseif**, si es verdadera entonces ejecuta el grupo (2) de instrucciones y si es falsa se ejecuta el grupo (3) de instrucciones.

La sintaxis de**f...elseif...else...end** se presenta en el c´odigo 1.5:

Estructura de c´odigo 1.5

Sintaxis def...elseif...else...end

```
if condici´on verdadera if
   grupo de instrucciones 1; %grupo(1)
   elseif condici´on verdadera elseif
    | grupos de instrucciones 2; %grupo(2)
   else
   grupo de instrucciones 3; %grupo(3)
end
```

Debe tomarse en cuenta que tanto **else**, como **elseif** quedan dentro de la instrucci´on **if** determinado por el delimitador end.

La figura 1.9 muestra el diagrama de flujo de la instrucci´on**f...elseif...else...end**.

for

La instrucci´on **for** genera lazos repetitivos (flujos o ciclos de instrucciones) para repetirse un n´umero determinado de veces. La instrucci´on **for** requiere de una variable contador para realizar n veces la iteraci´on, la condici´on de salida se alcanza cuando la variable contador ha llegado a la cuenta especificada.

La sintaxis de la instrucci´on **for** inicia con esa palabra clave, a continuaci´on la inicializaci´on de una variable contador que determinar´a el n´umero de veces que realizar´a el ciclo o flujo del grupo de instrucciones. Finalmente se incluye la palabra clave end para delimitar la instrucci´on **for**.

El c´odigo 1.6 presenta la estructura de la instrucci´on **for...end**:

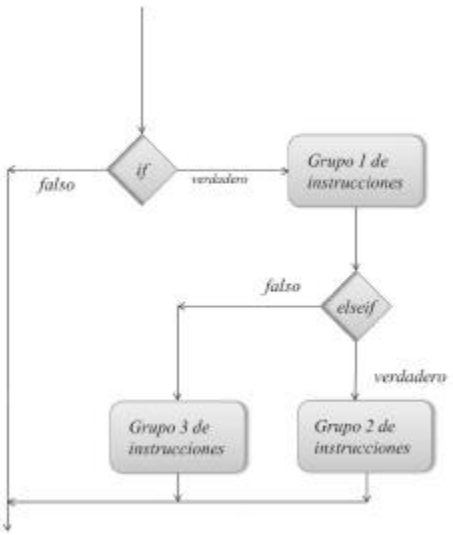


Figura 1.9 Diagrama de flujo de la instrucci´on **if...elsefi...end**.

Estructura de c´odigo 1.6

Sintaxis de la instrucci´on for

```
for  contador=1:n
|  instrucci´on 1;
|  ..............;
|  instrucci´on n;
end
```

Como un ejemplo sencillo para mostrar la utilidad de la instrucci´on for...end se presenta en el c´odigo 1.7 un programa que despliega 10 veces el contador **i** y la variable **j** que contiene el valor de una funci´on trigonom´etrica.

Estructura de c´odigo 1.7

Ejemplo de iteraciones con la instrucci´on for

```
j=0;
for i=1:10
|  disp(i,j)
|  j=sin(2*pi*i);
end
```

En este caso for ejecuta 10 veces el incremento sobre la variable **i**. Inicia **i** con el valor de 1 hasta llegar al valor de i=10, entonces la variable **i** realiza el papel de contador, con incrementos unitarios. La funci´on **disp(i,j)** despliega los valores num´ericos de las variables **i,j**.

El siguiente ejemplo muestra el empleo de for usando incrementos en mil´esimas para la variable **t** por medio del operador colon **:**. Esta aplicaci´on de la instrucci´on for es muy útil para graficar funciones como es el caso de la funci´on seno dentro de un intervalo de tiempo.

♣ Ejemplo 1.6

Escribir un programa en **MATLAB** para graficar una senal˜ senoidal.

Soluci´on

El programa 1.9 presenta el c´odigo fuente en **MATLAB** para graficar una senal˜ senoidal. En la l´ınea 3 se inicializa el vector tiempo de 0 a 10 segundos con incrementos de una mil´esima de segundo. El ciclo iterativo de la instrucci´on for abarca las l´ıneas 3 a la 7. El lector puede realizar el paso del tiempo de simulaci´on m´as fino, por ejemplo en diezmil´esimas. En la figura 1.10 se muestra la gr´afica que genera el programa 1.9.

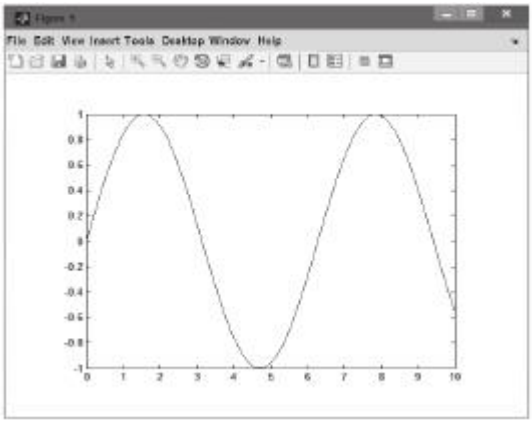


Figura 1.10 Funci´on senoidal.

C´odigo Fuente 1.9 Gr´afica de una onda senoidal

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo senoide.m
```

Gr´afica de una onda senoidal

```
1  disp('Programa para graficar una onda senoidal')
2  i=1;%inicializa pivote para registro
3  for t=0:0.001:10   % incrementos de t en mil´esimas
4      y(i)=sin(t);% arreglo de datos para la onda senoidal
5      tiempo(i)=t;% arreglo de datos para el tiempo
6      i=i+1;% incrementa pivote
7  end
8  disp('valor de y')
9  y
10 plot(tiempo,y)
```

El empleo de instrucciones `for` con la estructura`f...elseif...else...end` diversifica las aplicaciones de programaci´on. Como una aplicaci´on de este conjunto de instrucciones se presenta la implementaci´on de la funci´on signo la cual es ampliamente utilizada en din´amica y esquemas de control de sistemas mecatr´onicos y robots manipuladores; espec´ıficamente se emplea en el modelado del fen´omeno de fricci´on de Coulomb. Tambi´en se usa en algunos algoritmos de control.

Una de las principales desventajas de la funci´on signo es su discontinuidad que presenta y debido a esta caracter´ıstica muy particularidad produce retardo en el proceso de integraci´on num´erica de sistemas din´amicos, por lo que no se recomienda el emplear de esta funci´on cuando se realice simulaci´on de la din´amica de robots manipuladores.

Una posible forma de implementar la funci´on signo es por medio del uso de `for` y `if elseif else end`, el cuadro 1.10 presenta la funci´on signo( ).

C´odigo Fuente 1.10 Funci´on signo

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo signo.m
```

Funci´on signo

```
1  function y =signo(x)
2      [n,m]=size(x);
3      for i=1:n
4          if x(i) >0
5              y(i) = 1;%si es positivo retorna 1
6              elseif x(i) == 0
7                  y(i) = 0;%si es cero retorna 0
8              else
9              y(i) = -1;% si es negativo retorna -1
10         end
11     end
12 end
```

♣ Ejemplo 1.7

Escribir un programa en **MATLAB** para convertir una onda senoidal en onda cuadrada (emplear la funci´on `signo()` para realizar dicha conversi´on).

Soluci´on

Una forma de convertir una onda senoidal del tipo sen(t) a señal cuadrada (tren de pulsos cuadrados) es por medio del uso de la funci´on `signo(x)`. Considere el programa 1.11; el tiempo de simulaci´on es de 0 a 10 segundos, en la l´ınea 5 se obtiene la onda senoidal y la conversi´on a onda cuadrada se realiza en la l´ınea 6. La figura 1.11 contiene la conversi´on de la onda senoidal a onda cuadrada.

C´odigo Fuente 1.11 Aplicaci´on de la funci´on signo

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo signo_simu.m
```

Aplicaci´on de la funci´on signo

```
1 clc; clear all;
2 close all;
3 format short
4 ti=0;h=0.001; tf = 10;%par´ametros del tiempo de simulaci´on
5 t=(ti:h:tf)';%vector columna de tiempo
6 y=sin(t);% onda senoidal y1=signo(y);% funci´on signo
7 plot(t,y1)
```

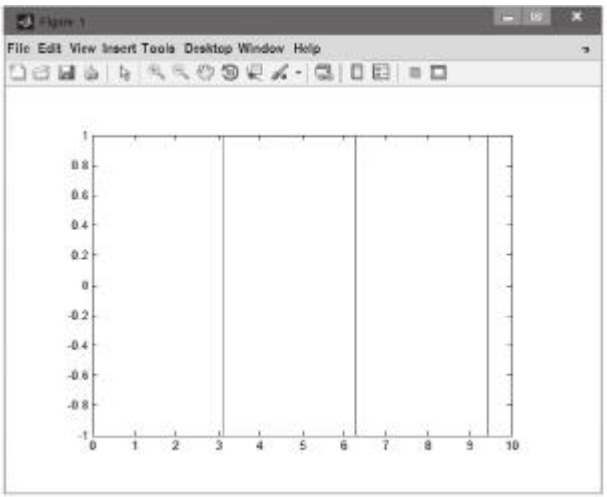


Figura 1.11 Funci´on signo().

Debido a que la funci´on signo est´a implementada bajo la estructura function es necesario realizar un programa principal donde se pueda invocar con el adecuado pase de par´ametros y su correcta sintaxis. Esta es la tarea que realiza el programa principal 1.11

Tambi´en es posible programar for anidados, es decir un for insertado dentro de otro for. Este tipo de aplicaciones resulta de utilidad en esquemas de control, c´alculos iterativos o recursivos, simulaci´on de sistemas din´amicos, procesamiento de se˜nales, etc.

El c´odigo 1.8 presenta la sintaxis gramatical para el caso de tener instrucciones for anidadas. Obs´ervese que el for externo realizar´a un n´umero de veces determinado por la variable contador_1 al grupo de instrucciones 1 y al segundo for. Por otro lado, el grupo de instrucciones 2 ser´a realizado un n´umero de veces indicado por el contador_2. El n´umero total de veces que se ejecutar´a el grupo (2) de instrucciones es contador_1 * contador_2.

Para un correcto funcionamiento de las instrucciones for anidadas es necesario ubicar adecuadamente los delimitadores end. Por ejemplo para la instrucci´on for externa abarca el grupo (1) de instrucciones, as´ı como el for interno con su segundo grupo de instrucciones.

Por lo tanto, est´an delimitados por el for externo y la palabra clave end. Mientras que para el segundo for el segundo grupo de instrucciones est´a delimitado por la palabra clave end correspondiente a la instrucci´on for del contador 2.

Estructura de c´odigo 1.8

Sintaxis para el caso de for anidados

```
for contador_1
    grupo de instrucciones_1;
    for contador_2
        ---------------------;
        grupo de instrucciones_2;
        ---------------------;
    end
end
```

Un ejemplo t´ıpico del empleo de for anidados es la aplicaci´on de suma de matrices realizada en forma recursiva. Este algoritmo emplea dos instrucciones for, el primero sirve para recorrer los renglones usando la variable **i**, mientras que el segundo for accede a las columnas de las matrices a trav´es del pivote **j**.

♣ Ejemplo 1.8

Escribir un programa en **MATLAB** para sumar dos matrices $A, B \in IR^{n \times p}$ en forma recursiva.

Soluci´on

El c´odigo fuente para sumar matrices en forma recursiva se presenta en el programa 1.12:

C´odigo Fuente 1.12 Algoritmo para sumar matrices

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 1 Introducci´on
%Editorial Alfaomega Fernando Reyes Cort´es
%Archivo sumamatrices.m

Algoritmo para sumar matrices
1  disp('Realiza la suma de dos matrices de orden n')
2  %Inicializaci´on de las matrices A, B dimensi´on 3 x 3
3  A=[ 1 1 3; 2 4 5; 8 9 0];
4  B=[3 2 5; 6 7 8; 9 0 0];
5  %Algoritmo de suma de matrices.
6  [ n p ] =size(A); % Obtiene orden de la matriz
7  %Algoritmo para sumar matrices
8  for i=1:n
9      for j=1:p
10         %Suma algor´ıtmica
11         C(i,j)=A(i,j)+B(i,j);
12     end
13 end
14 disp('Resultado:')
15 C
```

♣ Ejemplo 1.9

Escribir un programa en **MATLAB** para encontrar el m´inimo y m´aximo de la siguiente funci´on:

$$y = \text{sen}(t) - \cos(\pi t) - 2\tanh(t) + 0.1\log(t^3 + 1)$$

determinar los tiempos de ocurrencia.

Soluci´on

Para buscar el m´aximo y m´ınimo de la funci´on indicada (ver figura 1.12) y tiempos de ocurrencia puede usarse el c´odigo del programa 1.13.

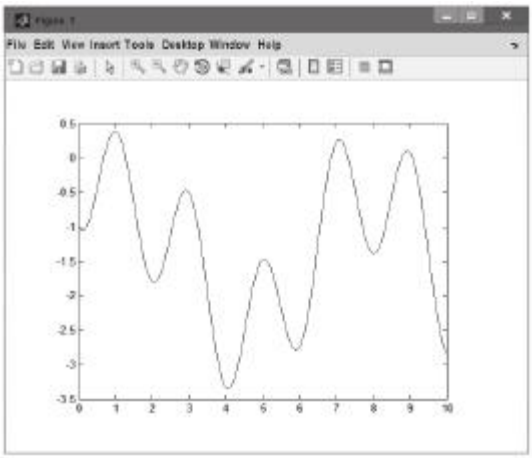


Figura 1.12 B´usqueda de m´ınimo y m´aximo.

La salida del programa tiene el siguiente resultado:

```
Max    Tmax    Min    Tmin

0.3887   0.9800   -3.3539   4.0600
```

Código Fuente 1.13 Mínimos y máximos

%MATLAB Aplicado a Robótica y Mecatrónica
%Capítulo 1 Introducción
%Editorial Alfaomega Fernando Reyes Cortés
archivo cap1_minimax.m

Mínimos y máximos

```
1  clc; clear all; close all;
2  disp('Encuentra mínimo y máximo de una función')
3  %variables para registrar máximo y mínimo
4  max=0; min=0; t_min=0; t_max=0;
5  %generación del vector de tiempo
6  t=[0:0.01:10]';%tiempo de 0 a 10 segundos con incremento de 0.01 seg.
7  [n m]=size(t); %dimensión del arreglo
8  y=zeros(n,m); %variables de registro
9  %función de prueba para generar la gráfica del proceso
10 for i=1:n
11 |   y(i)=sin(t(i))-cos(3.1416*t(i))-2*tanh(y(i))+0.1*log(t(i)^3+1);
12 end
13 %gráfica de la función del proceso
14 plot(t,y)
15 for i=1:n
16 |   if (y(i)>= max)
17 |       max=y(i);
18 |       t_max=t(i);
19 |       elseif (y(i)<= min)
20 |           min=y(i);
21 |           t_min=t(i);
22 |
23 |   end
24 end
25 disp( 'Max Tmax Min Tmin')
26 Resultado=[max t_max min t_min ];
27 disp(Resultado)
```

while

La instrucci´on while genera un lazo o ciclo de instrucciones que se repiten un n´umero finito o infinito de veces dependiendo de la condici´on l´ogica especificada. A diferencia de la instrucci´on for, while puede ejecutar un grupo de instrucciones de manera indefinida. La instrucci´on while tambi´en tiene su delimitador end. La forma general del lazo de programa while es:

Estructura de c´odigo 1.9

Sintaxis para el caso de for anidados

```
while condici´on boleana verdadera
    %grupo de instrucciones o sentencias
    instrucci´on 1;
    ------------;
    instrucci´on n;
end
grupo de instrucciones;
```

Un ejemplo de c´omo utilizar la instrucci´on while se presenta a continuaci´on:

Estructura de c´odigo 1.10

Uso de la instrucci´on while

```
a=10;
b=15;
i=0;
while  a>i
    i=i+1;
    b=b-i;
    if b>a
        disp('Robot activo')
        else disp('Proceso desactivo')
    end
end
```

switch, case

La instrucci´on switch ejecuta un grupo de sentencias con base al valor de la variable o expresi´on que emplea al lado derecho de su sintaxis. Para dividir los grupos de sentencias se emplean las palabras claves case y othewise. La instrucci´on switch finaliza con end.

En el siguiente c´odigo se ilustra la forma en que funciona la instrucci´on switch:

Estructura de c´odigo 1.11

Sintaxis de la instrucci´on switch

```
for i=1:3
    switch i
        case 1
            disp('caso 1')
            grupo de instrucciones;
        case 2
            disp('caso 2')
            grupo de instrucciones;
        case 3
            disp('caso 3')
            grupo de instrucciones;
        otherwise
        disp('fuera de rango')
        grupo de instrucciones;
    end
end
contin´ua con otro grupo de instrucciones;
```

MATLAB distingue los diferentes grupos de instrucciones asociados a un respectivo case. Por ejemplo, si entra a ejecutar el grupo de instrucciones del caso 1, se sale de la instrucci´on switch cuando encuentra el caso 2 y contin´ua con otro grupo de instrucciones ya fuera de switch. Para el caso de otherwise el grupo de instrucciones queda delimitado entre otherwise y el delimitador end perteneciente a switch.

break

La sentencia break permite abandonar o salirse del lazo generado por las instrucciones for o while. El uso de break se puede ilustrar por medio del siguiente ejemplo. Se genera un lazo usando while, como la condici´on de salida es una constante positiva, el lazo se generar´a de manera infinita. Sin embargo, la condici´on de salida queda determinada por **if** t==0.0 y break. En este ejemplo se realiza la operaci´on $\frac{sen(t)}{t}$, la variable **t** se inicializa en -1 y se decrementa en 0.001 en cada iteraci´on. Cuando t=0 se evita realizar una divisi´on entre cero usando break.

Estructura de c´odigo 1.12

Forma de usarse la instrucci´on break

```
t=-10;
while 1
    t=t+1;
    % c´odigo correspondiente para detectar cruce por cero
    if t==0
        disp('Advertencia divisi´on entre cero')
        break;
    end
    x= sin(t)/t;
end
disp(x)
```

Tambi´en es posible usar break en lazos anidados.

return

La expresi´on return termina la secuencia actual de comandos y retorna el control a la funci´on que invoc´o. Una llamada a funci´on normalmente transfiere el control a la funci´on que la invoc´o cuando alcanza el fin de la funci´on. Es posible insertar una sentencia return para forzar la terminaci´on de la funci´on y transferir el control a la funci´on que realizo la llamada.

Estructura de código 1.13

Forma de usarse la instrucción return

```
function y =cap1_raiz(x)
  if x<0
    disp('error: número imaginario');
    y=i;
    return;
    else y=sqrt(x);
  end
end
```

continue

La sentencia continue temporalmente interrumpe la ejecución del lazo del programa (for o while). A diferencia de break y return no causa una inmediata salida del lazo de control, permanece ahí suspendiendo la ejecución del programa hasta que la condición de salida del lazo del programa sea falsa.

Por ejemplo, en el siguiente código 1.14 la variable **i** se inicializa con cero y entra al lazo while permaneciendo ahí mientras la condición i<10 sea verdadera. Dentro de este lazo se pregunta si la variable **i** ya tiene el valor de 8 (por medio de **if** i==8), si esta condición es falsa, entonces continúa incrementando la variable **i** con i=i+1. Cuando la variable **i** es igual a 8, entonces se despliega en la ventana de comandos la leyenda Robot activo; se asigna el valor i=20; para que la condición de salida de while sea falsa, el control del programa sale del lazo de while por medio de la sentencia continue para ejecutar b=100 fuera de dicho lazo.

Observe que efectivamente la sentencia continue hace que el programa abandone el lazo de while, ya que ya no pasará por la instrucción **if** (i>10) i=0; end, la cual fue insertada intencionalmente para inicializar a la variable **i** en cero y provocar que la condición sea verdadera dentro del lazo while. Es necesario recalcar que la sentencia continue suspenderá el programa hasta que la condición de salida del lazo de control sea falsa, a diferencia con la forma de trabajar en break o return.

Estructura de c´odigo 1.14

Forma de usarse la instrucci´on continue en el lazo while

```
i=0;
while i<10
    if i==8
        disp('Robot activo')
        i=20; %condici´on de salida del lazo while
        continue; %contin´ua en b=100
    end
    if i>9
        i=0; %asegura condici´on verdadera para permanecer
        dentro del lazo while
    end
    i=i+1;
end
b=100
otro grupo de instrucciones;
```

El siguiente ejemplo 1.15 muestra el uso de la sentencia continue dentro del lazo generado por for:

Estructura de c´odigo 1.15

Forma de usarse la instrucci´on continue en el lazo for

```
for t=-10:0.1:10
    y=sin(t);
    if y==0.0
        disp('detecta cruce por cero')
        t=100; %condici´on de salida de lazo for
        % evita divisi´on entre cero
        continue; %salta en b=300
    end
    x=(sin(t)^2+cos(t)^2)/sin(t);
end
b=300
otro grupo de instrucciones;
```

1.9 Formato para datos experimentales

En las ´areas de rob´otica y mecatr´onica es com´un trabajar archivos que tengan datos experimentales o de simulaci´on con informaci´on de variables como error de posici´on, velocidad, aceleraci´on y par aplicado. No hay un formato espec´ıfico para generar los archivos. No obstante, es recomendable que tengan forma tabular, tipo matriz teniendo la primera columna destinada al tiempo, la segunda columna y posteriores para la informaci´on de las variables. Puede haber tantas columnas como variables se requiera registrar. Este formato para archivos de datos experimentales permite portabilidad para todos de paquetes de c´omputo de cualquier plataforma. Los datos experimentales de robots manipuladores son importantes para elaborar reportes t´ecnicos, art´ıculos cient´ıficos, tesis, memorias para congresos, etc.

¿C´omo generar un formato de archivo con datos experimentales?

MATLAB tiene las funciones fopen, fprintf, fclose, load y save para realizar dicho formato. El procedimiento para generar archivos de datos experimentales consiste en generar un archivo de tipo ASCII o texto, con la funci´on fopen o con la funci´on save por medio del siguiente **procedimiento**:

Indicar la trayectoria donde se generar´a el archivo de texto, por ejemplo con nombre robot.dat

```
fid = fopen('c:\robot\experimentos\robot.dat', 'wt');
```

donde el atributo 'wt' significa que se crea un nuevo archivo de texto para escritura y lectura. La variable fid es un identificador o apuntador asociado al archivo robot.dat

Agrupar en una matriz todas la variables que se desean grabar, por ejemplo: datos=[t, q, qp, qpp]; donde t es el tiempo, posici´on q, velocidad qp y aceleraci´on qpp. Posteriormente usar la funci´on fprintf para grabar las variables:

```
datos=[t, q, qp, qpp];
fprintf(fid,'\%3.3f \%3.3f    \%3.3f \%3.3f \n',datos);
```

El formato '%3.3f %3.3f %3.3f %3.3f \n' significa que se grabarán cuatro columnas de datos en formato de punto flotante, manteniendo una precisión numérica de tres dígitos con tres fracciones. Observe que para separar las columnas de datos hay al menos un espacio en blanco. El comando \n se usa para generar renglones, por cada dato **t(i), q(i), qp(i),** qpp(i), para i = 1, 2, ···, n.

Cerrar el archivo robot.dat de la siguiente forma:

```
fclose(fid);
```

Otra opción para generar el archivo de datos experimentales es usando la función save; por ejemplo:

```
save -ascii 'c:\robot\experimentos\robot.dat'    datos
```

Es recomendable que después de salvar el archivo, se libere la memoria ocupada por la variable datos, por medio de clear datos; de esta forma no se tendrán problemas de espacio.

Estructura de código 1.16

Generar formato de archivo con datos experimentales

```
clc; clear all; close all;
t=(0:0.001:10)'; %vector columna de la variable tiempo
q=sin(t); % posición
qp=cos(t); % velocidad
qpp=-sin(t); %aceleración
datos=[t, q, qp, qpp];
fid = fopen('c:\robot\experimentos\robot.dat', 'wt');
fprintf(fid, '%3.3f %3.3f %3.3f %3.3f \n',datos);
fclose(fid);
      :
clear datos;
```

Estructura de código 1.17

Opción para generar formato del archivo experimental

```
clc; clear all; close all;
t=(0:0.001:10)'; %vector columna de la variable tiempo
q=sin(t); % posición
qp=cos(t); % velocidad
qpp=-sin(t); %aceleración
datos=[t, q, qp, qpp];
%una forma alterna para grabar los datos
save -ascii 'c:\robot\experimentos\robot.dat'    datos
      ::
clear datos;
```

El programa que se muestra en la estructura de código 1.16 genera el formato del archivo con datos experimentales. Mientras que en la estructura de código 1.17 se presenta otra opción de generar el mismo formato. Para el caso específico del archivo robot.dat el formato se indica en la tabla 1.8. Note que la primera columna corresponde a la evolución del tiempo, la posición, velocidad y aceleración corresponden a la segunda, tercera y cuarta columna, respectivamente.

¿Cómo leer el archivo de datos experimentales?

Una vez generado un archivo con datos experimentales, se emplea la función load para abrir y manipular su información.

A continuación se describe el seguimiento procedimiento:

Usar la función load para cargar en memoria el archivo con datos experimentales:

```
datos_experimentales=load('c:\robot\experimentos\robot.dat});
t=datos_experimentales(:,1);
```

la primera columna tiene el tiempo y las demás columnas contienen información de las variables de interés al usuario.

Tabla 1.8 Formato de datos experimentales para rob´otica y mecatr´onica

t	q	$\dot{q}$	$\ddot{q}$
0	0	1.0000	0
0.0010	0.0010	1.0000	−0.0010
0.0020	0.0020	1.0000	−0.0020
0.0030	0.0030	1.0000	−0.0030
0.0040	0.0040	1.0000	−0.0040
⋮	⋮	⋮	⋮
0.0050	0.0050	1.0000	−0.0050
0.0060	0.0060	1.0000	−0.0060
0.0070	0.0070	1.0000	−0.0070
0.0080	0.0080	1.0000	−0.0080
0.0090	0.0090	1.0000	−0.0090
0.0100	0.0100	1.0000	−0.0100
0.0110	0.0110	0.9999	−0.0110
⋮	⋮	⋮	⋮

El programa que se indica en la estructura de c´odigo 1.18 permite abrir y manipular la informaci´on del archivo de prueba robot.dat.

Estructura de c´odigo 1.18

Abrir formato de archivo experimental

```
clc;
clear all;
close all;
datos_experimentales=load('c:\robot\experimentos\robot.dat');
t=datos_experimentales(:,1); %tiempo
qe=datos_experimentales(:,2); %posici´on
qpe=datos_experimentales(:,3); % velocidad
qppe=datos_experimentales(:,4); %aceleraci´on
```

1.10 Resumen

Hoy en d´ıa, **MATLAB** es un ambiente de programaci´on amigable e interactivo que se ha ubicado como un referente a nivel internacional en todas las universidades, institutos de investigaci´on e industria.

El lenguaje **MATLAB** es una potente herramienta de dise˜no en simuladores de sistemas mecatr´onicos y robots manipuladores. Contiene un amplio conjunto de instrucciones y librer´ıas toolbox para estudiar en detalle todos los aspectos de modelado din´amico, sistemas de control, identificaci´on param´etrica, procesamiento de im´agenes, etc´etera.

Particularmente, en los cursos de control autom´atico se emplea para caracterizar, evaluar y desarrollar esquemas de control. **MATLAB** tiene una gran cantidad de aplicaciones para las ´areas de rob´otica y mecatr´onica. Facilita el desarrollo de la investigaci´on y fortalecer al mismo tiempo aspectos pedag´ogicos en docencia, as´ı como la transmisi´on del conocimiento. Por eso, resulta clave dominar y programar con solvencia sobre todo en c´odigo fuente para realizar cualquier tipo de aplicaci´on.

En este cap´ıtulo se ha presentado las principales funciones e instrucciones del lenguaje **MATLAB**. Desde los aspectos b´asicos del entorno de programaci´on hasta aspectos m´as detallados donde se involucra declaraci´on de variables, manipulaci´on de matrices y arreglos, manejo de operadores, gr´aficas y funciones, as´ı como ejemplos en c´odigo fuente que ilustran su inmediata aplicaci´on a control y din´amica de robots manipuladores y sistemas mecatr´onicos.

Se propone un formato libre para archivos con datos de resultados experimentales o de simulaci´on; es muy importante tener un m´etodo computacional para registrar variables de estado e informaci´on relevante del robot. Para este prop´osito se explica el procedimiento para generar este tipo de archivos compuesto por columnas y renglones tipo matriz, tambi´en se describe el procedimiento para cargarlo a memoria y extraer la informaci´on registrada, para su an´alisis e interpretaci´on.

Finalmente, se recomienda consultar peri´odicamente: www.mathworks.com/.

Cap´ıtulo 2

M´etodos num´ericos

$$I = \int_0^t f(x)dx \Rightarrow I_k = I_{k-1} + hf(x_{k-1})$$

$$\dot{f} = \lim_{t \to 0} \frac{f(t + \quad t) - f(t)}{\quad} \Rightarrow \dot{f}_k = \frac{\quad}{f_k \dashv f_{k-1}}$$

Objetivos

Presentar los m´etodos num´ericos m´as utilizados en el ´area de ingenier´ıa a trav´es de ejemplos did´acticos y aplicaciones.

Objetivos particulares:

- Sistemas de ecuaciones lineales.
- M´etodos de diferenciaci´on num´erica (Euler, diff).
- M´etodos de integraci´on num´erica: Trapezoidal, Simpson, Euler.
- Sistemas din´amicos de primer orden (Runge-Kutta).
- Funciones ode (ecuaciones diferenciales ordinarias).
- Simulaci´on de sistemas din´amicos de primer orden.

2.1 Consideraciones computacionales

Resolver problemas de ingenier´ıa requiere del conocimiento solvente de m´etodos num´ericos para su adecuada implementaci´on pr´actica en simulaci´on, an´alisis y estudio de los resultados experimentales.

Un simulador se caracteriza por reproducir fielmente todos los fen´omenos f´ısicos del sistema mecatr´onico o robot manipulador. Desarrollar el proceso de simulaci´on es una etapa no trivial, involucra aspectos desde seleccionar el m´etodo num´erico m´as adecuado tanto en exactitud de resultados como en desempeño computacional. Tambi´en depende de las finalidades del problema a resolver, puede ser control en tiempo real o un escenario donde analizar´a resultados experimentales fuera de l´ınea (off-line). Este es el caso de algunos t´opicos de control y de automatizaci´on donde interviene identificaci´on param´etrica, caracterizaci´on de sensores y servomotores, estimaci´on de señales usando la posici´on articular por ejemplo, obtener velocidad y aceleraci´on de movimiento.

Cuando se llega a la etapa de resolver num´ericamente un sistema din´amico (lineal y no lineal), es importante seleccionar el m´etodo de integraci´on m´as adecuado de tal forma que no consuma muchos recursos computacionales (tiempo, eficiencia y desempeño) y que tenga la exactitud adecuada en la soluci´on num´erica para que ´esta sea confiable para su correcta interpretaci´on de resultados. Con estas caracter´ısticas del m´etodo de integraci´on, el esfuerzo de diseño se concentra en mejorar los atributos y propiedades del esquema de control y posteriormente evaluar el desempeño en un robot real. De esta forma, las potenciales aplicaciones del esquema de control son mucho m´as amplias, as´ı como su correcta ejecuci´on.

En el presente cap´ıtulo se presentan los principales m´etodos num´ericos para abordar problemas de control en sistemas mecatr´onicos y robots manipuladores. Se hace un especial ´enfasis en determinar cu´ales son m´as adecuadas para su utilizaci´on en control de procesos en tiempo real y cu´ales son m´as apropiadas para aspectos de simulaci´on donde el grado de exactitud es el apropiado para analizar y estudiar sistemas mecatr´onicos, as´ı como la din´amica compleja no lineal y multivariable de robots manipuladores.

2.2 Sistemas de ecuaciones lineales

En el ´area de ingenier´ıa es muy com´un encontrar sistemas de ecuaciones lineales, en donde se involucran variables desconocidas que fundamentalmente definen el problema a resolver; tambi´en participan matrices, as´ı como su matriz inversa y su determinante.

Consid´erese un conjunto de n ecuaciones lineales con m inc´ognitas dadas de la siguiente forma:

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ y_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots &= \qquad\qquad \vdots \\ y_m &= a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{aligned} \tag{2.1}$$

donde a_{ij} son coeficientes conocidos, con $i = 1, 2 \cdots, m$, $j = 1, 2, \cdots, n$; x_j son las inc´ognitas, y_i son variables conocidas. En forma compacta el sistema de ecuaciones lineales 2.1 puede ser representado por:

$$\mathbf{y} = A\mathbf{x} \tag{2.2}$$

donde $A \in IR^{m \times n}$, $\mathbf{x} \in IR^n$, $\mathbf{y} \in IR^m$

entonces la soluci´on consiste en encontrar $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$ en funci´on de la matriz A y de B, quedando de la siguiente forma:

$$\mathbf{x} = A^{-1}\mathbf{y}. \tag{2.3}$$

Obs´ervese que la soluci´on involucra la inversa de la matriz A. Por lo tanto, es necesario primero abordar los aspectos computacionales a la inversi´on de matrices.

Determinantes de matrices

El determinante de una matriz cuadrada $A \in \mathbb{R}^{n \times n}$ es un número o escalar y tiene varias aplicaciones en ingeniería: se utiliza en matrices inversas, solución de ecuaciones simultaneas; en robótica indica los valores numéricos de las variables articulares del robot para producir singularidades en su jacobiano, etc.

La sintaxis en **MATLAB** de la función determinante `det` de una matriz $A \in \mathbb{R}^{n \times n}$ es la siguiente:

```
det(A)
```

donde A es una matriz cuadrada de dimensión n previamente definidas sus entradas.

♣ Ejemplo 2.1

Considere el siguiente sistema de ecuaciones con tres incógnitas:

$$\begin{aligned} 9 &= 8x_1 + 5x_2 + x_3 \\ 6 &= -x_1 + 4x_2 - 10x_3 \\ 0 &= 5x_1 + 7x_2 + 2x_3 \end{aligned}$$

este sistema de ecuaciones puede ser re-escrito de la siguiente forma:

$$B = A\mathbf{x} \quad \text{donde} \quad B = \begin{bmatrix} 9 \\ 6 \\ 0 \end{bmatrix} \quad A = \begin{bmatrix} 8 & 5 & 1 \\ -1 & 4 & -10 \\ 5 & 7 & 2 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Obtener el determinante de la matriz A.

Solución

En el cuadro 2.1 se presenta el código fuente para obtener el determinante de la matriz A. El resultado es presentado en forma general usando variables simbólicas para expresar el determinante analítico de una matriz cuadrada de dimensión 3. En

varias aplicaciones de ingenier´ıa y para prop´ositos did´acticos es importante trabajar con variables simb´olicas; la funci´on det tambi´en acepta representaci´on simb´olica.

De las l´ıneas 12 a la 18, se asignan valores a las variables simb´olicas de la matriz A para mostrar el resultado en forma num´erica.

C´odigo Fuente 2.1 Determinante simb´olico

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 2 M´etodos num´ericos
%Editorial Alfaomega
%Fernando Reyes Cort´es
%Archivo cap2_detsim.m

Determinante simb´olico
 1 clear all;
 2 close all;
 3 clc;
 4 syms a11 a12 a13 a21 a22 a23 a31 a32 a33
 5 % matriz simb´olica
 6 Asim=[ a11, a12, a13;
 7          a21, a22, a23;
 8          a31, a32, a33];
 9 disp('Determinante simb´olico')
10 det_sim=det(Asim)
11 % se asignan valores a las variables simb´olicas
12 a11=8;
13 a12=5;
14 a13=1;
15 a21=-1;
16 a22=4; a23=-10;
17 a31=5; a32=7;
18 a33=2;
```

Código Fuente 2.2 Determinante simbólico

```
% Continúa programa 2.1
%MATLAB Aplicado a Robótica y Mecatrónica
%Capítulo 2 Métodos numéricos
%Editorial Alfaomega
%Fernando Reyes Cortés
%Archivo cap2_detsim.m
Determinante simbólico
%matriz numérica
Anum=[ a11, a12, a13;
       a21, a22, a23;
       a31, a32, a33];
disp('Determinante numérico')
det_num=det(Anum)
```

El programa cap2_detsim.m produce el siguiente resultado:

```
det_sim=
 a11a22a33 - a11a23a32 - a12a21a33 + a12a23a31 + a13a21a32 - a13a22a31
det_num=
   357
```

Inversión de matrices

En mecatrónica y robótica la representación de una matriz inversa está dada por A^{-1}. Una matriz $A \in IR^{n \times n}$ tiene inversa si la matriz A es cuadrada y su determinante es = 0. En **MATLAB** hay varias formas de obtener la inversa de una matriz cuadrada:

Ainv=inv(A) retorna la inversa de la matriz A

Un mensaje de advertencia se despliega si la matriz A tiene si tiene una singularidad (su determinante es un escalar muy pequeño o cercano a cero).

Ainv=A^(-1)

La ecuaci´on lineal (2.3) considera los siguientes casos: $m = n$ ($A \in \mathbb{R}^{n \times n}$ matriz cuadrada). Problema subdeterminado es cuando $m < n$ el n´umero de ecuaciones es menor que el n´umero de inc´ognitas. Para el caso en que $m > n$ el n´umero de ecuaciones es mayor que el n´umero de inc´ognitas se le conoce como problema sobre determinado.

Para el caso $m = n$, la matriz A es cuadrada, entonces la soluci´on se encuentra dada por:

$$\mathbf{x} = A^{-1}\mathbf{y}. \tag{2.4}$$

La soluci´on existe y es única si el determinante de la matriz A es diferente de cero. Una forma de verificar si la matriz A existe es analizando su determinante como en el c´odigo 2.1:

Estructura de c´odigo 2.1

```
Caso 1: m = n
if  det(A)~=0
 | x=inv(A)*y;
end
```

El c´odigo 2.1 no es único. Otro tipo de expresiones que producen el mismo resultado son:

x=A^(-1)*y;

x=A\y; Esta forma es recomendable, ya que la soluci´on emplea eliminaci´on gausiana, y por lo tanto no se utiliza la forma tradicional de obtener la inversa.

Usando la funci´on: x=linsolve(A,y).

linsolve

La funci´on `linsolve` representa una opci´on para resolver un sistema lineal de ecuaciones de la forma 2.3. La sintaxis de la funci´on `linsolve` est´a dada como:

```
y=linsolve(A,x)
```

donde A es la matriz del sistema de ecuaciones lineales y $\mathbf{x}$ es el vector inc´ognita.

Mayor informaci´on de la funci´on `linsolve` es proporcionada en:

```
fx >> help linsolve ←
```

♣ Ejemplo 2.2

Considere el siguiente sistema de ecuaciones:

$$1 = x_1 + 4x_2 + 3x_3 \quad (2.5)$$
$$4 = 8x_1 + 5x_2 + 9x_3 \quad (2.6)$$
$$6 = 3x_1 + 2x_2 + 4x_3 \quad (2.7)$$

Obtener la soluci´on del sistema $\mathbf{x}$ y compararla con los m´etodos:
$\mathbf{x} = \text{inv}(A)\mathbf{y}$, $\mathbf{x} = A^{-1}\mathbf{y}$ y $\mathbf{x} = A\backslash y$

Soluci´on

Para el sistema de ecuaciones planteado se tiene que la matriz A y el vector $\mathbf{y}$ est´an dados como:

$$A = \begin{bmatrix} 1 & 4 & 3 \\ 8 & 5 & 9 \\ 3 & 2 & 4 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix}.$$

El cuadro 2.3 contiene el c´odigo fuente para resolver el problema planteado y donde se han implementado los m´etodos solicitados.

C´odigo Fuente 2.3 Comparaci´on de soluciones SLE

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 2 M´etodos num´ericos
%Editorial Alfaomega
%Fernando Reyes Cort´es
%cap2_sle
```

Comparaci´on de soluciones SLE

```
1  %matriz A
2  A=[ 1 4 3;
3        8 5 9;
4        3 2 4];
5  y=[1; 4; 6];
6  % primera soluci´on
7  x=inv(A)*y;
8  % segunda soluci´on
9  x1=A^(-1)*y;
10 % tercera soluci´on
11 x2=A\y;
12 % cuarta soluci´on
13 x3= linsolve(A,y);
14 % comparaci´on de todas las soluciones
15 disp(' x x1 x2 x3 ')
16 disp([x x1 x2 x3])
```

La salida del programa cap2_sle arroja resultados id´enticos:

```
fx >>
```

x	x1	x2	x3
-5.8667	-5.8667	-5.8667	-5.8667
-4.3333	-4.3333	-4.3333	-4.3333
8.0667	8.0667	8.0667	8.0667

Regla de Cramer

La regla de Cramer representa otra opci´on para encontrar la soluci´on a sistemas de ecuaciones lineales de la forma (2.3). El procedimiento de la regla de Cramer consiste en obtener la soluci´on para cada componente x_i del vector $\mathbf{x}$ a trav´es de calcular el determinante ¡-´esimo obtenido como por sustituir el vector $\mathbf{y}$ en la ¡-´esima columna de la matriz A vista como un determinante, y dividirlo entre el determinante de la matriz A. Es decir:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \det_1 \\ \det_2 \\ \vdots \\ \det_n \end{bmatrix} \qquad (2.8)$$

$$\det_i = \frac{\begin{matrix} a_{11} & \cdots & \vdots & \cdots & a_{1n} \\ a_{21} & \cdots & \mathbf{y} & \cdots & a_{2n} \\ a_{n1} & \cdots & \vdots & \cdots & a_{nn} \end{matrix}}{\det(A)} \quad \text{columna}_i \quad \text{para } i = 1 \ldots n.$$

♣ Ejemplo 2.3

Considere nuevamente el sistema (2.5) del ejemplo 2.2. Obtener la soluci´on para $\mathbf{x}$ usando el m´etodo de Cramer.

Soluci´on

De acuerdo con el procedimiento de la regla de Cramer se establece que:

$$\det_1 = \frac{\begin{matrix} 1 & 4 & 3 \\ 4 & 5 & 9 \\ 6 & 2 & 4 \end{matrix}}{\begin{matrix} 1 & 4 & 3 \\ 8 & 5 & 9 \\ 3 & 2 & 4 \end{matrix}} = -5.8667 \qquad \det_2 = \frac{\begin{matrix} 1 & 1 & 3 \\ 8 & 4 & 9 \\ 3 & 6 & 4 \end{matrix}}{\begin{matrix} 1 & 4 & 3 \\ 8 & 5 & 9 \\ 3 & 2 & 4 \end{matrix}} = -4.3333$$

$$\det_3 = \frac{\begin{vmatrix} 1 & 4 & 1 \\ 8 & 5 & 4 \\ 3 & 2 & 6 \end{vmatrix}}{\begin{vmatrix} 1 & 4 & 3 \\ 8 & 5 & 9 \\ 3 & 2 & 4 \end{vmatrix}} = 8.0667.$$

Por lo tanto la soluci´on est´a dada por:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \det_1 \\ \det_2 \\ \det_3 \end{bmatrix} = \begin{bmatrix} -5.8667 \\ -4.3333 \\ 8.0667 \end{bmatrix}.$$

2.3 Diferenciaci´on num´erica

Diferenciación num´erica es la t´ecnica de aproximar a la derivada de una funci´on. La derivada es una operaci´on matem´atica sobre funciones o variables de estado, y que tiene una diversidad de aplicaciones en ingenier´ıa. La derivada de una funci´on representa la raz´on de cambio con respecto al tiempo; se utilizan en esquemas de control y en procesos de automatizaci´on. Por ejemplo, en el ´area de control de posici´on de robots manipuladores, los algoritmos incluyen un t´ermino denominado acci´on de control derivativo con la finalidad de generar amortiguamiento o freno mec´anico y con dicha acci´on se pretende mejorar la respuesta transitoria del robot. Para lograr esto, se inyecta la velocidad de movimiento, la cual representa la variaci´on temporal de la posici´on.

La fricci´on viscosa y de Coulomb es uno de los fen´omenos f´ısicos que se encuentra presente en los sistemas mec´anicos, la derivada de la posici´on forma parte de este fen´omeno. El modelado din´amico de sistemas mecatr´onicos y robots incluye derivadas de las variables de estado para conformar la estructura matem´atica adecuada para estudiar y analizar todos los fen´omenos f´ısicos del sistema. La teor´ıa de estabilidad de sistemas din´amicos es otro ejemplo donde se utilizan derivadas; por ejemplo, la derivada de la energ´ıa (potencia) se emplea para llevar a cabo el an´alisis de estabilidad del punto de equilibrio. Hoy en d´ıa, la derivada se ha convertido en una herramienta imprescindible para realizar automatizaci´on de sistemas.

La notación de la derivada de una función $f(t)$ con respecto al tiempo es definida como $\dot{f}(t)$, la cual es igual a la razón de cambio de f con respecto al tiempo t. Matemáticamente la derivada de una función se define como:

$$\dot{f}(t) = \frac{d}{dt} f(t) = \lim_{t \to 0} \frac{f(t + \Delta t) - f(t)}{\Delta t} \tag{2.9}$$

donde Δt es un infinitésimo intervalo de tiempo.

En mecatrónica hay varios procesos donde se requiere medir la razón de cambio de las variables del robot. Por ejemplo la razón de cambio de la posición es la velocidad de movimiento; el cambio temporal de la velocidad es la aceleración.

Por otro lado, la integral de la aceleración es la velocidad, y la integral de la velocidad es la posición. Es decir, existe una relación muy cercana entre la integral y la derivada; se puede considerar que son operaciones inversas una de la otra. La integral de una derivada:

$$\int_0^t \dot{f}(t)\,dt = \int_0^t \frac{df(t)}{dt} dt = \int_0^t df(t) = f(t) \Big|_0^t = f(t) - f(0)$$

retorna la función original $f(t)$ más una constante que depende de las condiciones iniciales.

La derivada de una integral

$$\frac{d}{dt} \int_0^t f(t)\,dt = f(t)$$

retorna la función original.

Geométricamente, la derivada $\dot{f}$ puede ser descrita como la pendiente de la línea tangente en la función $f(t)$ del punto específico t. La línea tangente está especificada por la pendiente $\frac{f(t+\Delta t) - f(t)}{\Delta t}$ (ver figura 2.3.)

Debido a esto, la derivada de una constante es cero, puesto que la línea tangente es horizontal (no hay variación temporal). Los puntos donde la derivada de la función f es cero se llaman **puntos críticos** y pueden representar regiones horizontales de la función f o **puntos extremos** (puntos que son máximo o mínimo local o también globales). Si evaluamos la derivada de la función en varios puntos y observamos que

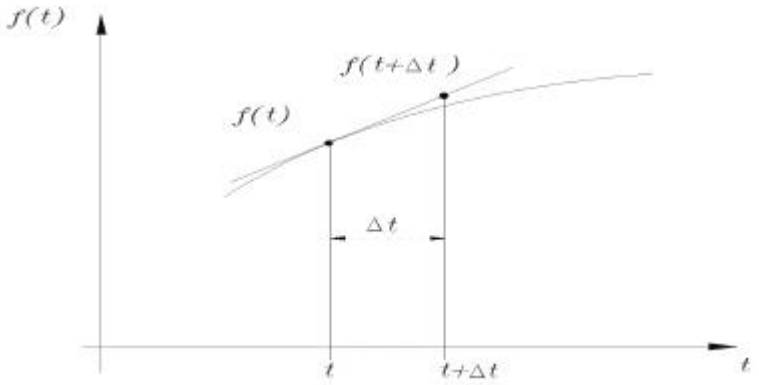


Figura 2.1 L´ınea tangente de la funci´on f en el punto t_k.

el signo de la derivada cambia, entonces un m´ınimo o m´aximo local ocurre en ese intervalo. La segunda derivada de la funci´on $\ddot{f}(t)$ determina cu´ando el punto cr´ıtico es m´ınimo o m´aximo. Si la segunda derivada en un punto extremo es positiva, entonces el valor de la funci´on en el punto extremo es un m´ınimo local. Por otro lado, si la segunda derivada de la funci´on en un punto extremo es negativo, entonces la funci´on evaluada en el punto extremo es un m´aximo local (ver figura 2.2).

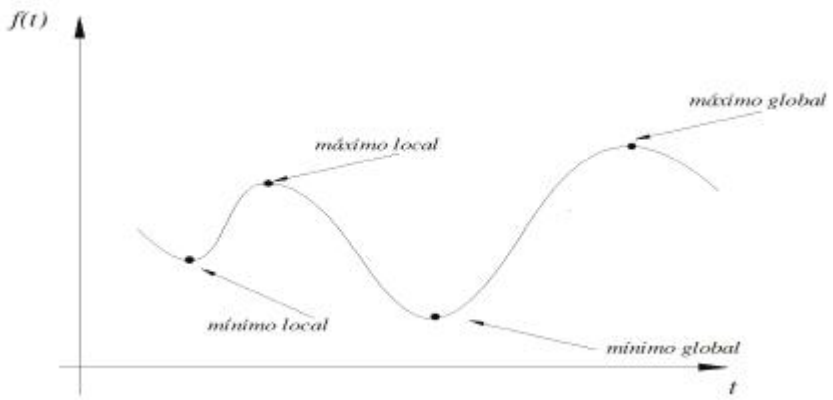


Figura 2.2 Puntos extremos: m´ınimo y m´aximo local (global).

Las t´ecnicas de diferenciaci´on num´erica estiman la derivada de una funci´on f en un punto t_k aproxima la pendiente de la l´ınea tangente en t_k usando valores de la funci´on en puntos cercanos a t_k. Si denotamos al intervalo de tiempo t como la diferencia entre dos puntos consecutivos, $t = t_k - t_{k-1} = h$, donde h es la longitud de t, (ver figura 2.3).

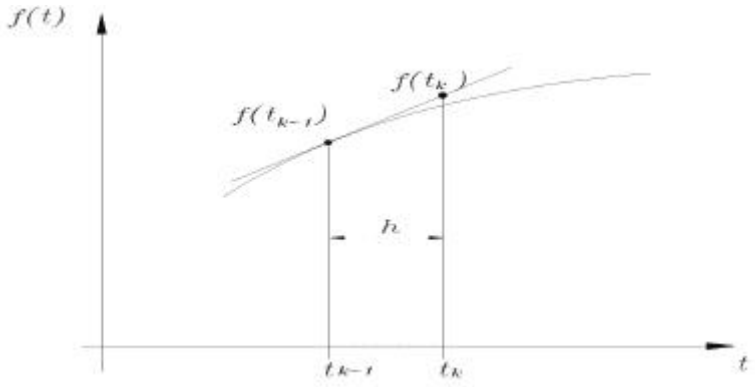


Figura 2.3 Línea tangente de la función f en el punto t_k.

En control digital y sistemas discretos la longitud del intervalo h se mantiene constante (periodo de muestreo) y se toman datos cada determinado tiempo h, igualmente espaciadas, esto significa que el tiempo discreto transcurre como múltiplos del periodo de muestreo $t_k = t + h$; para n muestras se tiene que el tiempo discreto se puede expresar como $t_k = kh$, donde $k = 1, 2, 3, \cdots, n$.

Una forma ampliamente usada es el método de Euler (diferenciación numérica)

$$\dot{f}(t_k) \approx \frac{f(t_k) - f(t_{k-1})}{h} = \frac{f(t_k) - f(t_{k-1})}{h}$$

particularmente esta aproximación es conocida como diferenciación con un paso atrás (backward difference).

También se puede obtener la aproximación de la derivada por computar la pendiente entre $f(t_k)$ y $f(t_{k+1})$ conocida como diferenciación hacia adelante (forward difference):

$$\dot{f}(t_k) \approx \frac{f(t_{k+1}) - f(t_k)}{h} = \frac{f(t_{k+1}) - f(t_k)}{h}$$

la figura 2.4 ilustra ambos métodos.

Evidentemente la calidad de la derivada depende de la distancia entre esos puntos o del periodo de muestreo $h = t_k - t_{k-1} = t_{k+1} - t_k$. De esta forma se puede obtener la aceleración por diferenciación numérica de la velocidad:

$$\ddot{f}(t_k) = \frac{\dot{f}(t_k) - \dot{f}(t_{k-1})}{h} = \frac{f(t_k) - 2f(t_{k-1}) + f(t_{k-2})}{h^2}$$

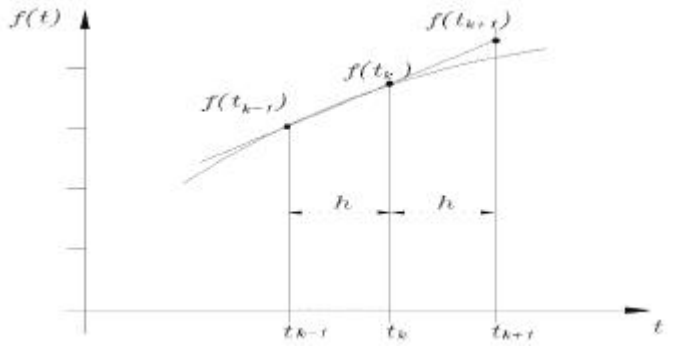


Figura 2.4 Diferenciaci´on num´erica: backward y forward.

♣ Ejemplo 2.4

Emplear el m´etodo de Euler para convertir la siguiente ecuaci´on diferencial en su equivalente expresi´on por diferenciaci´on num´erica.

$$\alpha u(t) = a_2\ddot{y}(t) + a_1\dot{y}(t) + a_0 y(t) \tag{2.10}$$

Soluci´on

Empleando el m´etodo de Euler, la velocidad y aceleraci´on toman la siguiente forma:

$$y(t) \qquad y(t_k) = \frac{y(t_k) - y(t_{k-1})}{h}$$

$$\ddot{y}(t) \qquad \ddot{y}(t_k) = \frac{\dot{y}(t_k) - \dot{y}(t_{k-1})}{h} = \frac{\frac{y(t_k) - y(t_{k-1})}{h} - \frac{\quad}{h}}{h}$$

$$\ddot{y}(t_k) = \frac{y(t_k) - 2y(t_{k-1}) + y(t_{k-2})}{h^2}$$

Por lo tanto la ecuaci´on diferencial 2.10 queda como:

$$\alpha u(t_k) = a_2\ddot{y}(t_k) + a_1 y(t_k) + a_0 y(t_k)$$

$$\alpha u(t_k) = a_2\left[\frac{y(t_k) - 2y(t_{k-1}) + y(t_{k-2})}{h}\right] + a_1\frac{y(t_k) - y(t_{k-1})}{h} + a_0 y(t_k).$$

La solución para $y(t_k)$ está dada por:

$$y(t_k) = [a_0 + \frac{a_1}{h} + \frac{2}{h^2}]^{-1} [\alpha u(t_k) + [\frac{2a_0}{h^2} + \frac{a_1}{h}] y(t_{k-1}) - \frac{a_0}{h^2} y(t_{k-2})].$$

Para implementar la solución discreta $y(t_k)$ de la ecuación (2.10) se requieren los estados $y(t_{k-1})$ y $y(t_{k-2})$ y el periodo de muestre h. El método de Euler es ampliamente utilizado para resolver ecuaciones diferenciales lineales, usando el procedimiento de este ejemplo se obtiene la solución en forma discreta de sistemas dinámicos lineales.

Función diff

MATLAB tiene la función **diff** la cual realiza la diferenciación numérica entre valores adyacentes de un vector **x**.

La sintaxis de la función **diff** es la siguiente:

```
y= diff(x)
```

diff Retorna un nuevo vector **y** conteniendo la diferenciación entre valores adyacentes del vector de entrada **x**

La función **diff** también se aplica a matrices, entonces opera en cada columna de la matriz, y retorna una matriz con el mismo número de columnas, en este caso la sintaxis es:

```
B= diff(A)
```

diff Retorna una nueva matriz B conteniendo la diferenciación entre valores adyacentes de cada columna de la matriz de entrada A.

♣ Ejemplo 2.5

Aproximar la derivada con respecto al tiempo de la funci´on sen(t) mediante diferenciaci´on.

Soluci´on

Para obtener la derivada temporal por diferenciaci´on num´erica de la funci´on sen(t) se hace uso de **diff**. El intervalo de diferenciaci´on num´erica se selecciona de 0 a 10 segundos, con incrementos de un milisegundo. El m´etodo anal´ıtico conduce $\frac{d}{dt}\text{sen}(t) = \cos(t)$. Por lo tanto, se comparar´a la soluci´on aproximada con la soluci´on anal´ıtica.

El programa que contiene el c´odigo fuente para resolver este problema se presenta en el cuadro 2.5. El resultado de dicho programa se exhibe en la figura 2.5 donde se han superpuesto ambas gr´aficas: por diferenciaci´on num´erica y el m´etodo anal´ıtico.

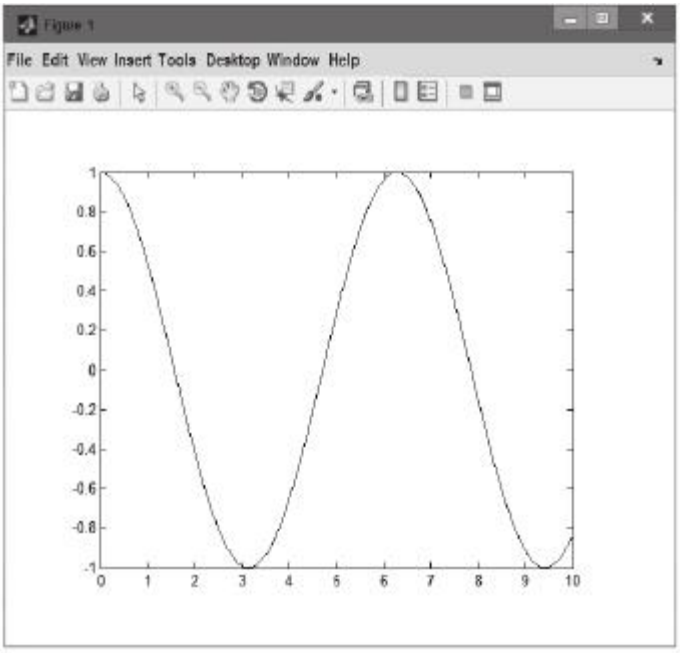


Figura 2.5 Comparaci´on del m´etodo anal´ıtico y por diferenciaci´on num´erica.

C´odigo Fuente 2.4 Diferenciaci´on num´erica

```
%MATLAB Aplicado a Rob´otica y Mecatr´onica
%Cap´ıtulo 2 M´etodos num´ericos
%Editorial Alfaomega
%Fernando Reyes Cort´es
%Archivo cap2_diffnum.m
```

Diferenciaci´on num´erica

```
1  clc;
2  clear all;
3  close all;
4  % intervalo de tiempo
5  t=0:0.001:10;
6  % funci´on a derivar
7  f=sin(t);
8  % derivada de la funci´on sen(t) con respecto al tiempo.
9  % es decir se obtiene f'(tk) = df(t)/dtk
10 dfdt = diff(f)./diff(t);
11 % el vector df tiene dimensi´on n-1
12 t1=0:0.001:9.999;% nueva base de tiempo con dimensi´on n-1
13 %compara la derivada aproximada con el m´etodo anal´ıtico
14 plot(t1,dfdt,t1,cos(t1))
```

2.4 Integraci´on num´erica

Integración num´erica es la t´ecnica de aproximar integrales de funciones. La integraci´on de funciones es un t´opico especial y de inter´es en aplicaciones pr´acticas para ingenier´ıa mecatr´onica y rob´otica. Desde el punto de vista anal´ıtico y experimental las integrales representan una herramienta fundamental en el an´alisis y diseño de algoritmos de control, desarrollo y construcci´on de robots manipuladores. La aplicaci´on inmediata de integraci´on num´erica se encuentra en el diseñ˜o de simuladores, la calidad de los resultados depende en gran medida de la exactitud

para aproximar a una integral de función.

Esquemas de control bien conocidos que se emplean en robots manipuladores como es el caso del proporcional integral derivativo (PID) involucra la integral del error de posición para mejorar las características operativas de la estrategia de control

$$\boldsymbol{\tau} = K_p\tilde{\mathbf{q}}(t) + K_i \int_{t_0}^{t} \tilde{\mathbf{q}}(t)\, dt - K_v\dot{\mathbf{q}}(t)$$

La acción de control integral almacena energía (área bajo la curva del error de posición), y mediante una adecuada sintonía en la ganancia integral influye sobre la cantidad de energía aplicada al robot, repercutiendo en la respuesta transitoria y reduciendo el error en estado estacionario.

Es necesario remarcar que en el caso del esquema PID, la acción de control integral se realiza en cada instante de tiempo conforme el tiempo evoluciona. Es decir, para cada valor de t se calcula la integral, de tal forma que no significa realizar la integral una sola vez, más bien es un cálculo continuo, uno tras otro como el tiempo evolucione.

En contraste con otros métodos, el cálculo de la integral sólo se requiere una vez, tal es el caso de la medición de desempeño de reguladores usando la norma $L_2[\tilde{\mathbf{q}}]$, cuya expresión matemática está dada por:

$$L_2[\tilde{\mathbf{q}}] = \sqrt{\frac{1}{T}\int_0^T \tilde{\mathbf{q}}(t)^2\, dt}$$

donde T representa el tiempo de experimentación o simulación (intervalo de integración).

Una magnitud pequeña en la norma L_2 significa alto desempeño del esquema de control. Esto significa que el área bajo la curva del error de posición no fue significativa debido a que el robot se posicionó de manera inmediata, logrando la convergencia hacia cero del error de posición $\tilde{\mathbf{q}}(t) \rightarrow \mathbf{0}$. Por lo tanto, en el periodo de integración el transitorio fue rápido, sin sobretiros y el desempeño del algoritmo de control es muy bueno. En contra parte, un alto valor de la norma L_2 representa pobre desempeño. La norma L_2 es muy útil cuando se compara varios algoritmos de control, entonces se determina cuál de ese conjunto de esquemas tiene mejores prestaciones y por lo tanto determina la selección para una aplicación específica.

La adecuada interpretaci´on de resultados experimentales, anal´ıticos y funcionamiento cualitativo de estrategias de control requiere del buen entendimiento de t´opicos espec´ıficos de c´alculo integral y diferencial. M´as a´un, c´omo implementar una integral, o seleccionar el m´etodo num´erico m´as adecuado son conocimientos fundamentales que se ven reflejados en la exactitud de posicionamiento de un robot manipulador y en el desempeño del esquema de control.

La integral definida de una funci´on $f(x)$ sobre un intervalo finito $[a, b]$ es interpretada como el ´area sobre la curva de $f(x)$ como se muestra en la figura 2.6. Para varias funciones la integral se puede obtener en forma anal´ıtica. Sin embargo, para un tipo de funciones su integral no se puede obtener por medios anal´ıticos, y por lo tanto se requiere de m´etodos num´ericos para estimar su valor.

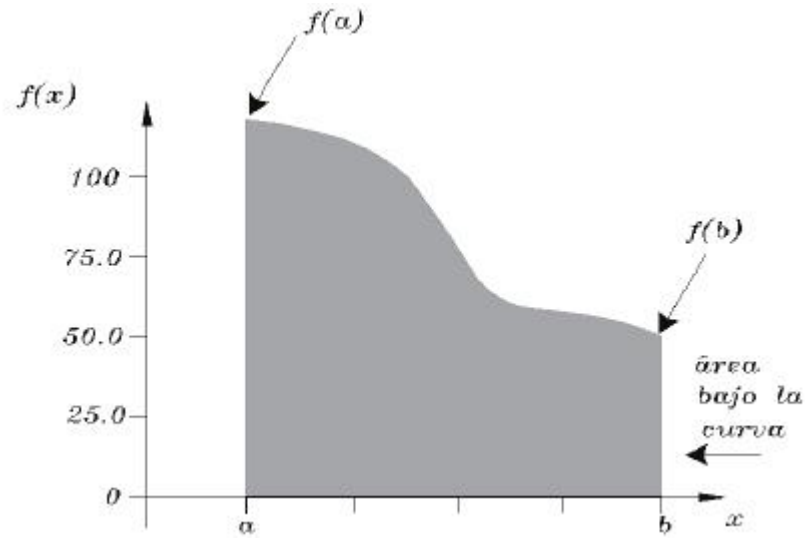


Figura 2.6 La integral de $f(x)$ significa el ´area bajo la curva.

Hay varias t´ecnicas para aproximar la integral de una funci´on $f(x)$; la evaluaci´on num´erica de una integral se le denomina **cuadratura** cuyo nombre proviene de un problema geom´etrico ancestral.

A continuaci´on se presentan dos m´etodos geom´etricos ampliamente conocidos: m´etodo trapezoidal y regla de Simpson.

2.4.1. Regla trapezoidal

Cuando el área bajo la curva de la función $f(x)$ es representada por trapezoides y el intervalo $[a, b]$ es dividido en n secciones del mismo tamaño, entonces el área de $f(x)$ puede ser aproximada por la siguiente expresión:

$$I_T = \frac{b-a}{2n}[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)] \qquad (2.11)$$

donde los valores x_i representan los valores finales de los trapecios, para $i = 1, 2, \cdots, n-1$; $x_0 = a$ y $x_n = b$.

MATLAB tiene la función `trapz` para aproximar la integral de una función $f(x)$ por el método trapezoidal, cuya sintaxis es la siguiente:

```
y=trapz(f)
```

```
y=trapz(x,f)
```

```
y=trapz(x,f,dim)
```

La función `trapz(f)` calcula la aproximación numérica de la integral de la función **f** por el método trapezoidal.

Si **f** es un vector, entonces `y=trapz(f)` es la integral de **f**. Cuando **f** representa a una matriz `y=trapz(f)` retorna un vector renglón con la integral sobre cada columna.

Cuando la función `trapz` tiene la sintaxis `y=trapz(x,f)` realiza la integral de **f** con respecto a x.

Para el caso `y=trapz(x,f, dim)`, dim es un escalar que indica la dimensión de la función **f**. En ambos casos, el vector x debe tener la misma dimensión dim.

♣ Ejemplo 2.6

Calcular la integral de la función sen(x) por el método trapezoidal para el intervalo $x \in [0, \pi]$

$$I_{trapz} = \int_0^{\pi} sen(x)\, dx.$$

Solución

El programa 2.5 contiene el código fuente para calcular la integral de la función $f(x) = sen(x)$ en el intervalo $[0, \pi]$. De la línea 4 a la 7 se define el intervalo de tiempo de integración. El incremento del paso de integración es cada $\frac{\pi}{1000}$. En la línea 8 se define la función sen(x) y en la línea 9 se realiza la integración numérica usando la función y=trapz(x,f). Este resultado se compara con el valor analítico, es decir $I_{trapz} = \int_0^{\pi} sen(x)dx = 1 - cos(\pi) = 2$. El resultado que genera el programa 2.5 es: $I_{trapz} = 2.0$.

Código Fuente 2.5 Integración numérica trapezoidal

```
%MATLAB Aplicado a Robótica y Mecatrónica
%Capítulo 2 Métodos numéricos
%Editorial Alfaomega
%Fernando Reyes Cortés
%Archivo cap2_trap.m
```

Integración numérica trapezoidal

```
1 clc;
2 clear all;
3 close all;
4 tini=0;
5 tinc=pi/1000;
```