Fundamentos de Programación para Ciencias e Ingeniería

Eduardo Tusa Jumbo, Hugo Sanmartín Cuenca

Universidad Técnica de Machala

Fundamentos de Programación para Ciencias e Ingeniería

Ing. César Quezada Abad, MBA
Rector

Ing. Amarilis Borja Herrera, Mg. Sc.
Vicerrectora Académica

Soc. Ramiro Ordóñez Morejón, Mg. Sc.
Vicerrector Administrativo

COORDINACIÓN EDITORIAL
VICERRECTORADO ACADÉMICO

Tomás Fontaines-Ruiz, PhD.
Investigador Becario Prometeo-Utmach
Asesor Del Programa De Reingeniería

Ing. Karina Lozano Zambrano
Coordinadora Editorial

Ing. Jorge Maza Córdova, Ms.
Ing. Cyndi Aguilar
Equipo de Publicaciones

Fundamentos de Programación para Ciencias e Ingeniería

Eduardo Alejando Tusa Jumbo

Hugo Dennys San Martín Cuenca

UNIVERSIDAD TÉCNICA DE MACHALA
2015

Dedicatoria

Este libro está dedicado a mis estudiantes de la Carrera de Ingeniería Civil de la Universidad Técnica de Machala (UTMACH).

Ellos y ellas son la fuente de motivación para la construcción de esta obra.

A mis padres Contardo y Alva, por sus sabios consejos y su apoyo constante.

Son verdaderos modelos de tolerancia y amor incondicional.

A mis hermanas Karla y Fernanda, por la enorme pasión que entregan a todas las cosas que hacen en sus vidas.

A mis sobrinos Santiago y Martin, por ser destellos de alegría en el devenir de mis días, y razones de desvelo en el pasar de mis noches.

A mis familiares, por ser maravillosos ejemplos de unidad en la adversidad.

A mis amigos, por su tiempo, su paciencia y su comprensión.

A las autoridades y colegas docentes de la UTMACH, por la oportunidad de crecer como profesional junto a ustedes.

Eduardo Tusa

Este libro está dedicado a mis padres Hugo y Gloria, por el apoyo incondicional tanto económico como emocional, quienes a lo largo de nuestra vida han velado por nuestro bienestar y educación siendo nuestro pilar fundamental.

A mis hermanos Jimmy y Nahomi, por el apoyo y motivación que me han dado para nunca rendirme en el proceso de alcanzar mis metas.

A mis amigas y compañeras Rosita, Ninoska y Miriam, quienes me han apoyado dedicándome un poco de su tiempo, su paciencia y su comprensión.

Dennys San Martin

Primera edición 2015

ISBN: 978-9978-316-80-1

D.R. © 2015, UNIVERSIDAD TÉCNICA DE MACHALA
Ediciones UTMACH
Km. 5 1/2 Vía Machala Pasaje
www.utmachala.edu.ec

ESTE TEXTO HA SIDO SOMETIDO A UN PROCESO DE EVALUACIÓN POR PARES EXTERNOS CON BASE EN LA NORMATIVA EDITORIAL DE LA UTMACH.

Portada:
Concepto editorial: Jorge Maza Córdova
Samanta Cabezas (EST. COMUNICACIÓN SOCIAL)
Fotografia: Dir. de Comunicación UTMACH

Diseño, montaje y producción editorial: UTMACH

Impreso y hecho en Ecuador
Printed and made in Ecuador

Advertencia: "Se prohíbe la reproducción, el registro o la transmisión parcial o total de esta obra por cualquier sistema de recuperación de información, sea mecánico, fotoquímico, electrónico, magnético, electroóptico, por fotocopia o cualquier otro, existente o por existir, sin el permiso previo por escrito del titular de los derechos correspondientes".

Índice

Introducción 13

Algoritmos de Programación

Lenguaje de programación 15
Generaciones de los Lenguajes de Programación 16
Etapas de Desarrollo de un Programa..... 18
Variables..... 19
Propiedades de las Variables 20
Nombres de Variables 21
Vocabulario..... 22
Jerarquía de Variables 22
Ejercicios sobre Operaciones con Variables 23
Algoritmo 24
Desarrollo de Algoritmos 25
Diagrama de Flujo..... 26
Comandos básicos en MATLAB 28
Ejercicios sobre Algoritmos y Comandos Básicos en MATLAB .. 30

Estructuras condicionales

Estructuras de Control Condicionales 33
Estructura Condicional Simple IF/END..... 33
Estructura Condicional Doble IF/ELSE/END 35

Estructura Condicional Múltiple IF/ELSEIF/ELSE/END 36
Estructura Condicional Múltiple SWITCH/CASE 38
Ejercicios sobre Estructuras Condicionales 40

Estructuras Repetitivas
Estructuras de Control Repetitivas 43
Estructura Repetitiva FOR 43
Estructura Repetitiva WHILE 45
Interrupción con BREAK 46
Interrupción con CONTINUE 47
Ejercicios sobre Estructuras Repetitivas 49

Arreglos en MATLAB
Arreglo Unidimensional 53
Manipulación de vectores 53
Creación de vectores 56
Comandos 57
Operaciones entre vector 60
Ejercicios sobre Vectores 61
Arreglo Multidimensional 62
Manipulación de matrices 62
Manipulación filas y columnas 65
Matrices Especiales 66
Operaciones entre matrices 68
Comandos Especiales 69
Ejercicios sobre Matrices 71

Graficación en MATLAB
Gráficos Bidimensionales 2D 73
Comando plot(x,y) 73
Comando grid on 74
Comando plot(x,y,'r*') 75
Comandos de etiqueta 75
Comandos para curvas paramétricas 77
Comando para curvas polares 79
Ejercicios sobre Graficación 2D 80

Gráficos Tridimensionales 3D 80
Comando plot3(x,y,z) 81
Comando quiver3 82
Comandos meshgrid y plot3 83
Comandos meshgrid y mesh 83
Comandos meshgrid y surf 84
Comando subplot 85
Comando colorbar 86
Comandos para curvas de nivel 87
Comando SPHERE (Esfera) 89
Comando CYLINDER (Cilindro) 90
Ejercicios sobre Graficación 3D 91

Funciones
Definición 93
Funciones en línea 94
Funciones definidas mediante script 96
Declaracion de una función 96
Ejercicios sobre Funciones 100

Bibliografía 103

Índices de cuadros, gráficas, imágenes y fotografías 105
Índice de gráficas 106

Biografía 107

Introducción

El desarrollo de los dispositivos computacionales con un alto rendimiento en términos de memoria, capacidad de almacenamiento y velocidad de procesamiento de la información; ha desafiado inminentemente a los desarrolladores de software. La actual generación de programadores se ve en la necesidad de implementar aplicaciones que sean capaces de responder a los requerimientos del usuario, aprovechando los recursos tecnológicos disponibles.

La actual demanda de programas pretende satisfacer un sin número de campos profesionales como los negocios, la educación, el sector social, entre otros. Particularmente, dedicaremos nuestra atención a los programas que tienen principales aplicaciones en Ciencias e Ingeniería.

Este tipo de programas han emergido como una poderosa herramienta en la investigación y desarrollo de nueva tecnología. Permiten la evaluación de fórmulas complejas con una alta precisión. Además, estos programas interactúan con un conjunto de datos provenientes de sensores, y que son procesados a velocidades significativas.

Este libro propone revisar los fundamentos necesarios para la elaboración de programas en una plataforma de programación conocida como MATLAB. Este entorno de programación ofrece la posibilidad de introducir al estudiante en el desarrollo de programación secuencial, programación estructurada y programación modular.

Para cumplir el propósito de aplicar estos diferentes tipos de programación, se propone un capítulo 1 donde se explica sobre las reglas básicas para la utilización de variables y el planteamiento de algoritmos de programación secuencial, utilizando diagramas de flujo.

En los capítulos 2 y 3, se presentan las estructuras de control condicionales y repetitivas, respectivamente. Entre las estructuras condicionales se destacan las condicionales simples, dobles y múltiples. Mientras que las estructuras repetitivas destacan aquellas que son controladas por índice y las que son controladas por condición.

En el capítulo 4 se presentan los arreglos unidimensionales representados por vectores y los arreglos bidimensionales representados por matrices. Se hace una amplia descripción sobre la manipulación de la información que estas estructuras almacenan.

En el capítulo 5 se propone un conjunto de estrategias para el desarrollo de gráficos en 2 y 3 dimensiones. Ésto permite fundamentalmente la visualización de la información para un apropiado análisis de los fenómenos presentes en la naturaleza.

Finalmente, el capítulo 6 explica la construcción de funciones que serán herramientas importantes en la programación modular. Se hace una diferenciación en la manera que se hace programación secuencial mediante ficheros, y las ventajas que posee el uso de funciones.

Los autores de este libro expresan su profunda expectativa de que esta obra se constituya en una referencia que despierte en los estudiantes de ciencias e ingeniería, la pasión y el interés en la resolución de problemas mediante el uso de programas computacionales. De igual manera, esperamos que el lector encuentre en cada problema planteado en este libro, como una oportunidad para atreverse a aprender: Sapere aude.

Algoritmos de programación

El presente capítulo tiene como finalidad familiarizar al estudiante con la idea de lenguaje de programación, destacando su evolución y etapas de desarrollo. A continuación, se conceptualiza la definición de variable y la manipulación de la información que estas entidades almacenan.

Finalmente, se define la idea de algoritmo y su representación mediante diagramas de flujo. Los algoritmos serán implementados en MATLAB a través de los comandos básicos que permiten el ingreso y la salida de información. Con ello, el lector será capaz de producir programas mediante una programación secuencial.

Lenguaje de programación

Un lenguaje de programación es un vocabulario o conjunto de reglas gramaticales que instruyen a una computadora a desarrollar una tarea específica [1]. Es decir, es un sistema notacional para describir instrucciones de una forma legible tanto para la máquina como para el ser humano.

Los lenguajes de programación se caracterizan por su eficiencia, expresividad, capacidad de mantenimiento, legibilidad, confiabilidad, seguridad, simplicidad, y productividad.

Las ventajas que existen en el estudio de los lenguajes de programación, son las siguientes:

- Aumenta la capacidad para producir software.
- Mejora la habilidad para desarrollar mejores algoritmos.
- Potencia el uso de los lenguajes de programación.

- Incrementa el vocabulario.
- Permite elegir mejor el lenguaje a utilizar para cada tarea.
- Hace más fácil aprender nuevos lenguajes de programación.

Generaciones de los Lenguajes de Programación

Existen muchos tipos de lenguajes de programación, por ejemplo: C++, VB.NET, Java, Python, Ensamblador. Es necesario revisar la historia de cómo se produjeron estos lenguajes y las razones por las que todavía son útiles. En todos los casos, es necesario tener en mente que lo único que se ejecutará en un ordenador es el código de máquina, una vez que éste se ha convertido desde un lenguaje para ejecutarse en un procesador [2].

Primera Generación - 1940

La primera generación de lenguajes de programación se basa en código de máquina que está conformado por unos y ceros. Los programadores tienen que diseñar su código a mano y luego transferirlo a un equipo con una tarjeta perforada, cinta perforada o mediante la conmutación de interruptores. No hay necesidad de traducir el código, el cual se ejecuta inmediatamente. Entre los beneficios, se puede destacar que el código es rápido y eficiente; y hace uso de características específicas del procesador como los registros especiales. Por esta razón, el código no puede ser utilizado en otros sistemas y debe ser reescrito, más allá de la dificultades existentes para editar y actualizar el código.

Segunda Generación - 1950

Los lenguajes de programación de segunda generación se caracterizan por la utilización de código ensamblador. La programación se hace mucho más fácil de leer y escribir mediante el uso de códigos nemotécnicos que se asemejan a un lenguaje de comunicación como el Inglés. Para convertir un programa de código ensamblador en código

de máquina que permita su ejecución en un equipo, se requiere de un Ensamblador; el cual sustituye cada línea de código ensamblador por el equivalente a una línea de código de máquina.

Tercera Generación - Mediados de 1950

Los lenguajes de programación de tercera generación son conocidos como lenguajes de alto nivel. Esta generación introduce nuevas características amigables con el programador con el fin de codificar bucles, condicionales, clases, etc. Esto significa que una línea de código de tercera generación pueden producir muchas líneas de máquina, ahorrando tiempo significativo en la edición de programas. Los lenguajes de tercera generación puede ser independientes de la plataforma, lo que significa que el código escrito para un sistema puede funcionar en otro. Además, la conversión de un programa de tercera generación en código de máquina requiere de un Compilador o un Interpretador.

Cuarta Generación - 1970

Los lenguajes de cuarta generación están diseñados para reducir el esfuerzo de programación y el tiempo que se necesita para el desarrollo de software, lo que resulta en una reducción en el costo de desarrollo del mismo. A veces, estos programas no presentan el mejor desempeño en las tareas para las que fueron creados, resultando un código poco elegante y difícil de mantener.

Los lenguajes de esta generación han sido elaborados con un propósito específico en mente y ésto incluye los lenguajes para consultar bases de datos (SQL), los lenguajes para realizar informes (Oracle Reports) y los lenguajes para la construcción de la interfaz de usuario (XUL).

Quinta Generación - 1980

Una quinta generación de lenguajes de programación resulta de la agrupación de los lenguajes que se basan en la premisa de que un

problema se puede resolver con una aplicación creada para resolverlo, proporcionando restricciones al programa (programación basada en restricciones), en lugar de especificar algorítmicamente el proceso para resolver el problema (programación imperativa).

En esencia, el lenguaje de programación se utiliza para denotar las propiedades, o la lógica de una solución, en lugar de establecer la forma en que se llega a la solución. La mayoría de los lenguajes de programación basados en restricciones y programación lógica son lenguajes de quinta programación. PROLOG (acrónimo de lógica de programación) es un ejemplo de un lenguaje de programación lógica. Éste utiliza una forma de lógica matemática (cálculo de predicados) para resolver las consultas en una base de datos de hechos y reglas proporcionadas por el programador [3].

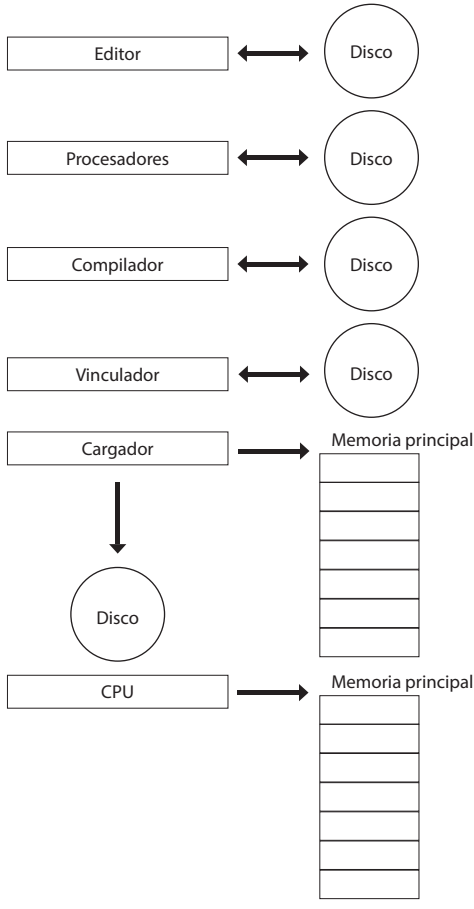
Etapas de Desarrollo de un Programa

El programador crea el programa en el editor y lo almacena en el disco. El preprocesador procesa el código. El compilador crea el código, objeto y lo almacena en el disco. El vinculador enlaza el código objeto con las bibliotecas, crea un archivo ejecutable y lo almacena en el disco.

El cargador pone el programa en la memoria. Durante la ejecución, el CPU toma cada instrucción y la ejecuta. Posiblemente; almacenando nuevos valores de los datos conforme el programa se ejecuta.

Las etapas para la creación de un programa se presentan a continuación en la Figura 1 [4]:

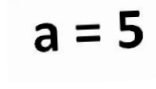
Figura 1: Representación de los pasos para la creacion de un programa



Variables

Las variables se definen como símbolos con los que se representan valores. Es una porción de memoria para almacenar un valor como se observa en la Figura 2 [5].

Figura 2: Representación de la variable "a"



Una variable debe ser nombrada para representar todos los valores posibles que puede contener, por ejemplo; Nota_mediosemestre, notas_mediosemestre, dato_puntos, nombre_curso. La información en la variable representa el último valor almacenado, desapareciendo el contenido que estaba previamente guardado. Las variables se usan para obtener una copia de la información preservando el contenido original.

Propiedades de las Variables

Las propiedades principales de una variable que se deben tomar en cuentan son:

1. Un nombre
2. Un tipo
3. Un valor
4. Una tiempo de vida
5. Una ubicación

Nombre

El nombre es simbólico. Representa el título de la información que se almacena en la variable. El nombre define la forma en cómo el programador accede al contenido de la variable. Cada variable debe tener un nombre único.

Tipo

En MATLAB, el tipo de la variable se infiere de los datos almacenados en la variable. Generalmente, se definen cuatro tipos de variables: numéricas, lógicas, caracteres y arreglos.

Valor

El valor de una variable, por su nombre, cambia con el tiempo. Si a la variable edad_juan se le asigna el valor 21, en otro momento, esa misma variable, edad_juan; se le puede asignar otro valor como 27.

Tiempo de vida

El tiempo de vida de una variable es fuertemente relacionada con el alcance de la variable. Cuando un programa se inicia, las variables cobran vida en el momento en que el programa llega a la línea de código donde están declaradas. Las variables mueren cuando el programa sale del alcance de la variable.

En nuestros primeros programas que no tienen funciones, el tiempo de vida y el alcance de las variables será el programa completo.

Ubicación

Afortunadamente, no tenemos que preocuparnos demasiado en qué parte del equipo se almacena el contenido la variable. Durante el desarrollo de un programa, se va definiendo la ubicación de la variable.

Se debe ser consciente de que existe una casilla o espacio de memoria en el hardware para cada variable que se declara. En el caso de una matriz, se necesitará de un grupo de casillas. Cada casilla puede contener un solo valor.

Nombres de Variables

Los principales aspectos que se debe tomar en cuenta para la determinación del nombre de una variable se detallan a continuación:

1. Las nombres de variables empiezan con letras.
2. Utilice subguiones para mostrar mayor claridad.
3. Puede utilizar números
4. No utilice símbolos especiales
5. No puede dejar espacios
6. Deben tener nombres significativos

Por ejemplo, g = −9,81 es un mal ejemplo de nombre significativo si desea definir la constante gravitacional. Alternativamente, un buen ejemplo sería constante_gravitacional = −9,81.

Vocabulario

Acceder a una variable significa leer la información dentro de la variable. Para acceder a una variable, simplemente se debe escribir el nombre de la variable.

Asignar un valor a una variable es reemplazar el último valor de la variable por uno nuevo.

El valor anterior se borra para siempre. Algunas veces, se reemplaza el antiguo valor de una variable en función del valor actual de la variable. Por ejemplo,

numero_de_estudiantes = numero_de_estudiantes + 1.

Declarar una variable es crear la variable. En MATLAB, se declara una variable simplemente escribiendo su nombre y asignándole un valor. Por ejemplo, edad_juan = 21.

Inicializar una variable es otra manera de decir, declarar una variable, pero además, implica que le da un valor base.

La información almacenada en una variable es la última pieza de contenido asignada a esa variable. Después de realizar una nueva asignación, toda la información anterior se pierde.

Jerarquía de Variables

Las variables numéricas en MATLAB se definen mediante su uso. No hay la necesidad de una sección explícita donde se declaren variables. Las variables numéricas son por defecto definidas como dobles, es decir utilizan 64 bits para almacenar un valor.

Las operaciones más básicas con MATLAB son las operaciones aritméticas. Los operadores aritméticos básicos son +, −, /, * y ^ potencia. Estos operadores se pueden utilizar conjuntamente con los paréntesis ().

Al igual que con todos los lenguajes de programación, la atención especial se debe dar a cómo se escribe una expresión matemática. Por ejemplo, 5 + 10/2 * 3 = 5+(10/2) * 3 = 20, que no es igual a 5 + 10/(2 * 3) = 5 + 53. En general, MATLAB sigue el presente orden jerárquico:

1. Los paréntesis ()
2. Exponenciación
3. La multiplicación y división, *, /, de izquierda a derecha.
4. Más y menos, +, −, de izquierda a derecha.

Ejemplo:
3 + 5/2 * 4 − 2 ^ 3 + (5 * 2) = 3 + 5/2 * 4 − 2 ^ 3 + 10
3 + 5/2 * 4 − 2 ^ 3 + (5 * 2) = 3 + 5/2 * 4 − 8 + 10
3 + 5/2 * 4 − 2 ^ 3 + (5 * 2) = 3 + 2,5 * 4 − 8 + 10
3 + 5/2 * 4 − 2 ^ 3 + (5 * 2) = 3 + 10 − 8 + 10
3 + 5/2 * 4 − 2 ^ 3 + (5 * 2) = 15

Ejercicios sobre Operaciones con Variables

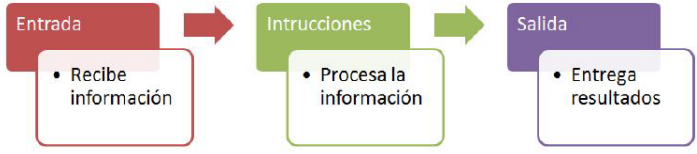
MATLAB almacena el último resultado en una variable denominada ans. Para almacenar variables por un largo tiempo, se puede especificar el nombre. Ejecute cada una de las siguientes líneas de instrucciones en la ventana de comandos de MATLAB:

```
>> x = 5+2 ^ 2
>> y = 2*x
>> z = x ^ 2 + y ^ 2
>> 3 * (5/3 − 4/3) − 1
>> pi
>> (−1) ^ (0,5)
>> log(0)
>> 0/0
>> S = 2 > 4
>> S |~ S
>> (2 < 3) * 3
>> true * 3
>> false | −4,03
>> w ='Goodmorning'
>> w
```

Algoritmo

Es una descripción ordenada de las instrucciones que deben realizarse para resolver un problema en un tiempo finito [6]. La estructura de un algoritmo se presenta a continuación en la Figura 3.

Figura 3: Estructura de un Algoritmo

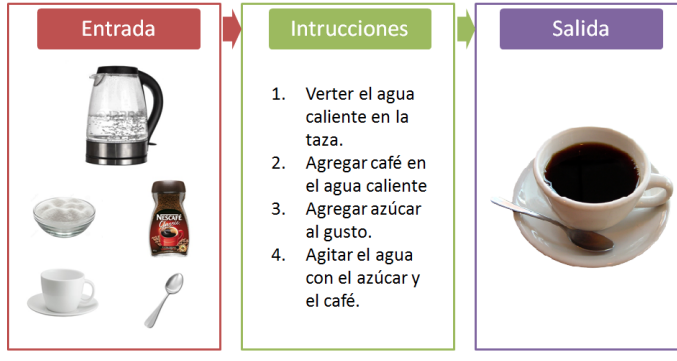


Ejemplos de Algoritmos

1. El algoritmo para la preparación de una taza de café se presenta en la Figura 4.

En este ejemplo la información de entrada son los ingredientes: el agua, azúcar, café, una taza y una cuchara, las instrucciones son el proceso a seguir en orden, mientras que en el de salida tenemos como resultado la taza de café.

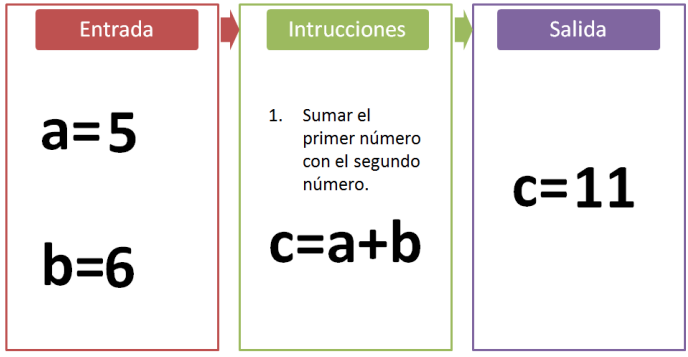
Figura 4: Algoritmo para la preparación de una taza de café



2. El algoritmo para la suma de dos números se presenta en la Figura 5.

En este ejemplo la información de entrada son las variables con su respectivo valor: a = 5 y b = 6, en donde las instrucciones son el proceso matemático c = a+b, mientras que en el de salida tenemos el resultado de c.

Figura 5: Algoritmo para la suma de dos números



Desarrollo de Algoritmos

La metodología para el desarrollo de un algoritmo se describe en los siguientes cuatro pasos que son ilustrados en el Ejemplo 1 [6].

1. Objetivo
2. Componentes o Variables
3. Instrucciones necesarias
4. Prueba de validación.

Ejemplo

Describa un algoritmo para calcular el área de un triángulo dados sus tres lados (suponer que son 5, 6, 8).

Datos: a, b, c: Lados del triángulo.

Resultado: s: Área del triángulo.

Fórmula geométrica: $s = \sqrt{t(t-a)(t-b)(t-c)}$, siendo $t = \frac{a+b+c}{2}$

Resolución:

1. Objetivo: Calcular el área de un triángulo dado sus tres lados.

2. Componentes o variables:

a = 5

b = 6

c = 8

s =área del triangulo

3. Instrucciones necesarias:

Fórmula geométrica para calcular el área de un triángulo

$s = \sqrt{t(t-a)(t-b)(t-c)}$

Donde primero debemos calcular el valor de t dada la siguiente fórmula $t = \frac{a+b+c}{2}$

Una vez calculado t se procede a reemplazar en la fórmula para el área de un triángulo

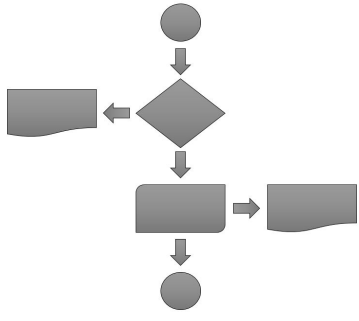
4. Prueba de validacion:

S	a	b	c	t
14.9812	5	6	8	9,5

Diagrama de Flujo

Es la representación gráfica de la solución algorítmica de un problema tal como se observa en la Figura 6. Para diseñarlos se utilizan determinados símbolos o figuras que representan una acción dentro del procedimiento.

Figura 6: Diagrama de Flujo



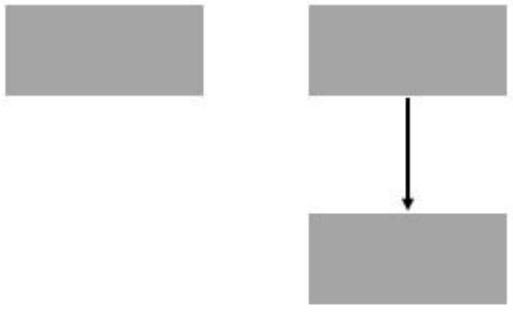
Las características de los diagramas de flujo son las siguientes:

- Se escriben de arriba hacia abajo y de izquierda a derecha.
- Siempre se usan flechas verticales u horizontales, jamás curvas.
- Se debe evitar cruce de flujos.
- En cada paso se debe expresar una acción concreta.
- Secuencia de flujo normal en una solución de problema.
- Tiene un inicio.
- Una lectura o entrada de datos.
- El proceso de datos.
- Una salida de información.
- Un final

Desarrollo

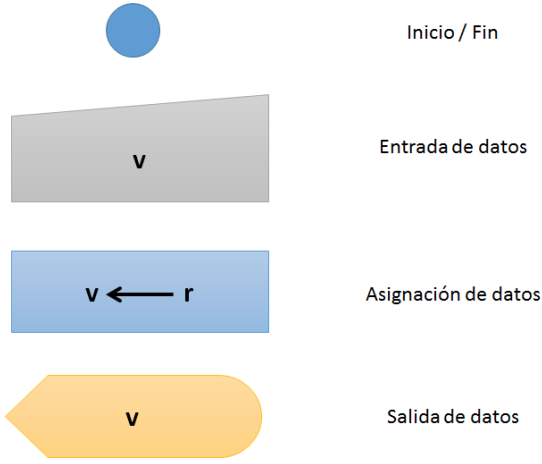
• Cada componente puede incluir una instrucción simple o un conjunto de instrucciones como en la Figura 7.

Figura 7: Enlaces de un diagrama de flujo



•Sea "v" una variable y "r" es un valor que se desea usar en el algoritmo, las diferentes representaciones de un diagrama de flujo se observan en la Figura 8.

Figura 8: Componentes de un diagrama de flujo

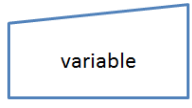


Comandos básicos en MATLAB

Ingreso de datos

Esta instrucción se utiliza para describir la acción de ingresar algún valor para una variable desde fuera del algoritmo, cuando sea este ejecutado. Esta instrucción permite que los datos no estén dentro del algoritmo y poder realizar diferentes pruebas con diferentes datos. Su sintaxis en MATLAB junto a su representación en Diagrama de flujo se observa en la Figura 9.

Figura 9: Ingreso de datos en MATLAB



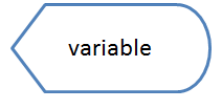

```
>> variable = input('mensaje');
```

Salida de datos

Esta instrucción básica se utiliza para describir la acción de mostrar o desplegar el contenido de las variables y que representan los resultados

de interés. Su sintaxis en MATLAB junto a su representación en Diagrama de flujo se observa en la Figura 10.

Figura 10: Salida de datos en MATLAB




```
>> disp('variable');
```

Ejemplo

Describa un algoritmo para calcular el área de un triángulo dados sus tres lados (suponer que son 5, 6, 8).

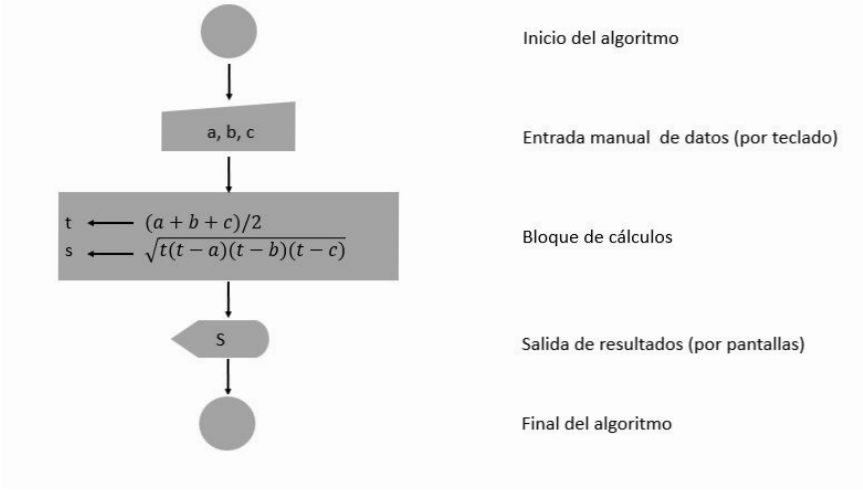
Datos: a, b, c: Lados del triángulo.

Resultado: S=Área del triángulo.

Fórmula geométrica: $s = \sqrt{t(t-a)(t-b)(t-c)}$, siendo $t = \frac{a+b+c}{2}$

Diagrama de flujo del ejemplo se observa en la Figura 11.

Figura 11: Diagrama de flujo



```
1 %este programa calcula el area de un triangulos
2 a = 5; % primer lado del triangulo
3 b = 6; % segundo lado del triangulo
4 c = 8; % tercer lado del triangulo
5 t = (a + b + c)/2;
6 s = sqrt((t)*(t-a)*(t-b)*(t-c));
7 disp('area del trianguloes es:');
8 disp(s);
```

Funciones fix y mod

La función fix trunca los decimales de un número y entrega la parte decimal.

Command Window

```
>> c = fix (20/6)

    c =

       2
fx
```

La función mod entrega el residuo de la división entera entre dos números.

Command Window

```
>> c = mod(20,6)

     c =

       2
fx
```

Ejemplo 1. Dado un número entero de dos cifras, sumar sus cifras

```
1 % Sumar las cifras de de un numero entero de dos cifras
2 n = input('Ingrese un numero de dos cifras');
3 d = fix(n/10);
4 u = mod(n/10);
5 s = d+u;
6 disp('Suma');
7 disp('s');
```

Ejercicios sobre Algoritmos y Comandos Básicos en MATLAB

Los siguientes ejercicios han sido propuestos por el Profesor Luis Rodríguez en [6].

1. Dados el radio y la altura de un cilindro calcule el área total y el volumen con el fin de mostrar los resultados al usuario.

2. Se tiene un recipiente cilíndrico con capacidad de x litros. Su altura es h metros. Determine el diámetro de la base.

3. Calcule el promedio de tres calificaciones de las lecciones que obtuvo un estudiante.

4. La siguiente fórmula proporciona el enésimo término u de una progresión aritmética: u = a + (n − 1) r, en donde a es el primer término, n es la cantidad de términos y r es la razón entre dos números consecutivos. Calcule el valor de r dados u, a y r

5. Un modelo de crecimiento poblacional está dado por $n = 5t+2e^{0,1t}$, en donde n es el número de habitantes, t es el tiempo en años. Calcule el número de habitantes que habría en tres años consecutivos.

6. Una empresa produce varillas de construcción. Cada mes el ingreso por ventas en miles de dólares se describe con v = 0,4x(30 − x) mientras que el costo de producción en miles de dólares es v = 5+10ln(x), siendo x la cantidad producida en toneladas entre $1 < x < 30$.

Determine un valor del ingreso neto, dado un valor para x.

7. Dado un número de dos cifras, mostrarlo con las cifras en orden opuesto.

8. Lea dos números de tres cifras cada uno. Sume la cifra central del primer número con la cifra central del segundo número y muestre el resultado.

9. Dado un número de tres cifras. Determine si la cifra central es igual a la magnitud de la diferencia entre la primera y la tercera cifras.

Estructuras Condicionales

Las estructuras de control son parte fundamental de cualquier lenguaje. Sin ellas, las instrucciones de un programa sólo podrían ejecutarse en el orden en que están escritas(orden secuencial).

Las estructuras de control permiten modificar este orden. Hay dos categorías de estructuras de control: Condicionales o bifurcaciones, y Bucles o repeticiones[7]. En este capítulo revisaremos las estructuras condicionales, sus tipos, sus sintaxis y sus representaciones en diagrama de flujo.

Estructuras de Control Condicionales

Las estructuras de control condicionales o bifurcaciones permiten que se ejecuten conjuntos distintos de instrucciones, en función de que se verifique o no determinada condición.

Los tipos de estructuras condicionales o de decisión que describiremos a continuación son:

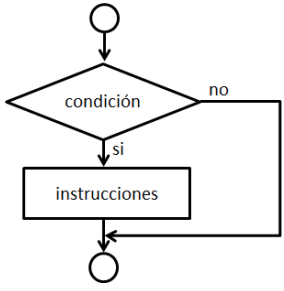
- Estructura Condicional Simple
- Estructura Condicional Doble
- Estructura Condicional Múltiple

Estructura Condicional Simple IF/END

Si se verifica una determinada condición, se ejecuta una serie de instrucciones y luego se sigue adelante. Si la condición no se cumple,

no se ejecutan dichas instrucciones y se sigue adelante. La estructura condicional simple se representa en el diagrama de flujo de la Figura 12, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

Figura 12: Estructura Condicional Simple

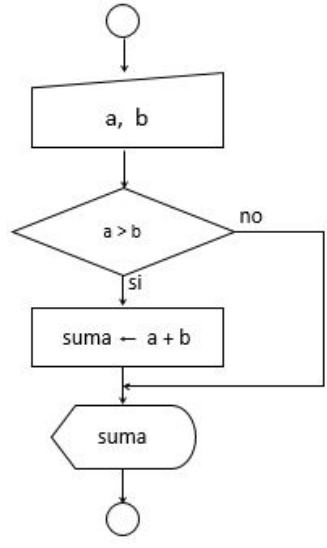



```
1 % Estructura Condicional Simple
2
3 if condicion
4     instrucciones
5 end
```

Ejemplo: Este programa compara dos números y si a > b entonces calcula la suma = a + b.

A Continuación se representa el diagrama de flujo en la Figura 13.

Figura 13: Diagrama de flujo de condicional simple IF/END

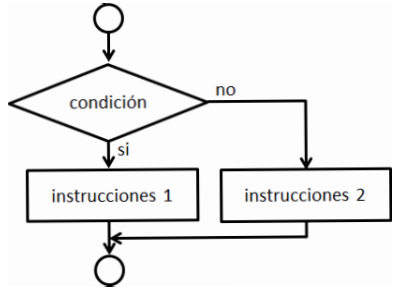


```
1 % Si a > b entonces calcula la suma = a + b
2
3 a = input('Ingrese el valor de a: ');
4 b = input('Ingrese el valor de b: ');
5
6 if a > b                          % condicion
7      suma = a + b;                % instrucciones
8      disp('La suma es: ');
9      disp(suma);
10 end
```

Estructura Condicional Doble IF/ELSE/END

Si se verifica una determinada condición, se ejecuta una serie de instrucciones 1. Si la condición no se verifica, se ejecuta una serie de instrucciones 2. La estructura condicional doble se representa en el diagrama de flujo de la Figura 14, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

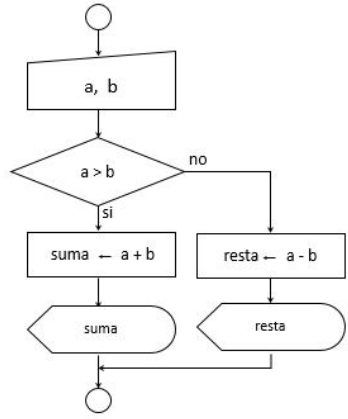
Figura 14: Estructura Condicional Doble




```
1 % Estructura Condicional Doble
2
3 if condicion
4     instrucciones1;
5 else
6     instrucciones2;
7 end
```

Ejemplo: Este programa compara dos números y si a > b entonces calcula la suma = a + b, de lo contrario calcula la resta = a − b.
A continuación se representa el diagrama de flujo en la Figura 15.

Figura 15: Diagrama de flujo de condicional doble IF/ELSE/END

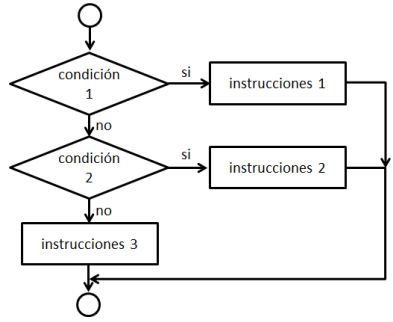



```
1 % Si a > b entonces calcula la suma = a + b
2 % de lo contrario calcula la resta = a - b
3
4 a = input('Ingrese el valor de a: ');
5 b = input('Ingrese el valor de b: ');
6
7 if a > b % condicion
8      suma = a + b; % instrucciones 1
9      disp('La suma es: ');
10     disp(suma);
11 else % de lo contrario
12     resta = a - b; % instrucciones 2
13     disp('La resta es: ');
14     disp(resta);
15 end
```

Estructura Condicional Múltiple IF/ELSEIF/ELSE/END

Si se verifica la condición 1, se ejecuta un bloque de instrucciones 1. Si no se verifica la condición 1, pero si se verifica la condición 2 , se ejecuta un bloque de instrucciones 2. Si no se ha verificado ninguna de las condiciones anteriores, se ejecuta el bloque de instrucciones 3. La estructura condicional múltiple se representa en el diagrama de flujo de la Figura 16, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

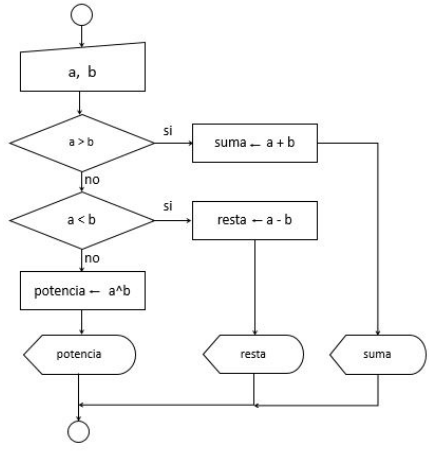
Figura 16: Estructura Condicional Múltiple




```
1 % Estructura Condicional Multiple
2
3 if condicion1
4     instrucciones1;
5 elseif condicion2
6     instrucciones2;
7 else
8     instrucciones3;
9 end
```

Ejemplo: Este programa compara dos números. Si a > b entonces calcula la suma = a + b. Si a < b calcula la resta = a − b. Si a = b entonces calcula potencia = a^b.
A continuación se representa el diagrama de flujo en la Figura 17.

Figura 17: Diagrama de flujo de condicional múltiple IF/ELSEIF/ELSE/END.

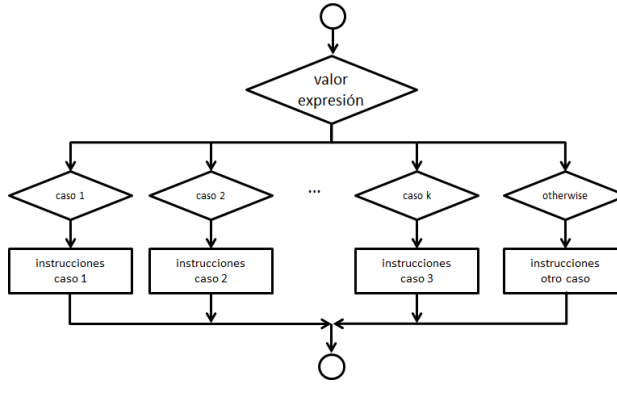


```
% Si a > b entonces calcula la suma = a + b.
% Si a < b calcula la resta = a - b.
% Si a=b entonces calcula potencia = a^b

a = input('Ingrese el valor de a: ');
b = input('Ingrese el valor de b: ');

if a > b                          % condicion 1
     suma = a + b;                % instrucciones 1
     disp('La suma es: ');
     disp(suma);
elseif a < b                      % condicion 2
     resta = a - b;               % instrucciones 2
     disp('La resta es: ');
     disp(resta);
else
     potencia = a^b;              % instrucciones 3
     disp('La potencia es: ');
     disp(potencia);
endd
```

Estructura Condicional Múltiple SWITCH/CASE

En la sentencia SWITCH/CASE, se compara el valor de una variable o el resultado de evaluar una expresión, con un conjunto de valores caso 1, caso 2, ..., caso k. Cuando coinciden, se ejecuta el bloque de instrucciones que están asociadas. Si el compilador no encuentra coincidencia, se ejecuta la sentencia OTHERWISE, si es que está presente en el código. La estructura condicional múltiple SWITCH/CASE se representa en el diagrama de flujo de la Figura 18, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

Figura 18: Estructura Condicional SWITCH/CASE

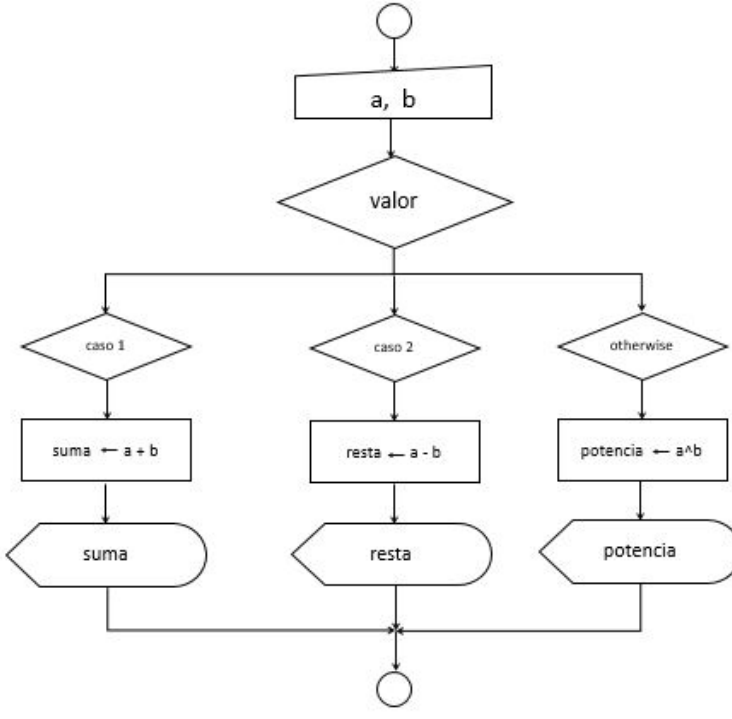


```
1 % Estructura Condicional SWITCH/CASE
2
3 switch valor
4     case 1
5         instrucciones caso1;
6     case 2
7         instrucciones caso2;
8     case 3
9         instrucciones caso3;
10    otherwise
11        instrucciones otro caso;
12 end
```

Ejemplo: El usuario ingresa un valor y dos números a y b. Si el valor es igual 1 entonces el programa calcula la suma = a+b. Si el valor es igual 2 entonces el programa calcula la resta = a − b. En otro caso, el programa calcula potencia = a^b.

A continuación se representa el diagrama de flujo en la Figura 19

Figura 19: Diagrama de flujo de condicional múltiple SWITCH/CASE.



```
% Si valor = 1 entonces calcula la suma = a + b.
% Si valor = 2 calcula la resta = a - b.
% en otro caso calcula potencia = a^b

disp('Este programa realiza las siguientes operaciones:');
disp('Si escoge 1 se calcula la suma de dos numeros')
disp('Si escoge 2 se calcula la resta de dos numeros')
disp('cualquier valor se calcula la potencia de dos numeros')

valor = input('Ingrese el valor de la operacion a realizar: ');
a = input('Ingrese el valor de a: ');
b = input('Ingrese el valor de b: ');

switch valor
    case 1                      % condicion 1
       suma = a + b;            % instrucciones 1
       disp('La suma es: ');
       disp(suma);
    case 2                      % condicion 2
       resta = a - b;           % instrucciones 2
       disp('La resta es: ');
       disp(resta);
    otherwise
       potencia = a^b; % instrucciones 3
       disp('La potencia es: ');
       disp(potencia);
end
```

Ejercicios sobre Estructuras Condicionales

1. El siguiente programa presenta errores. Describa dónde están esos errores y corríjalos para que el programa determine si dos números son: el mayor o el menor, y si son par o impar.

```
pnum = (Ingrese el primer numero: );
snum = (Ingrese el segundo numero: );

if pnum > snum
    disp('El menor de los numeros es: ');
    disp('pnum');
    if mod(pnum,2) = 0
        disp('Ademas, el numero menor es impar');
    else
        disp('Ademas, el numero menor es par');
    end
else
    disp('El mayor de los numeros es: ');
    disp('snum');
    if mod(pnum,2) = 0
        disp('Ademas, el numero mayor es impar');
    else
        disp('Ademas, el numero mayor es par');
    end
end
```

2. Determine los valores de a, b, c y d después de que los algoritmos representados en las Figuras 20 y 21 han sido ejecutados.

Figura 20: Ejercicio 1 con Diagrama de flujo de Condicionales

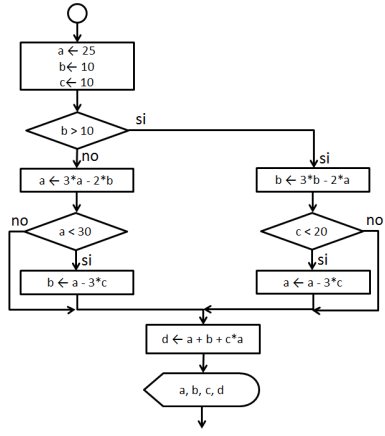
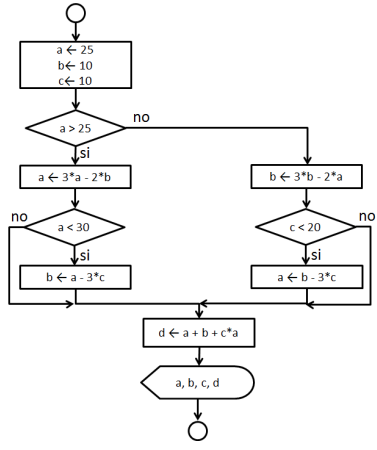


Figura 21: Ejercicio 2 con Diagrama de flujo de Condicionales



Los siguientes ejercicios han sido propuestos por el Profesor Luis Rodríguez en [6].

3. Dados el radio y la altura de un cilindro, si la altura es mayor al radio, calcule y muestre el volumen del cilindro, caso contrario muestre el mensaje: Error.

4. Lea un número de dos cifras. Determinar si la suma de ambas cifras es una número par o impar. Muestre un mensaje.

5. Lea la cantidad en Kw que ha consumido una familia y el precio por Kw. Si la cantidad es mayor a 700, incremente el precio un 5% para el exceso de Kw sobre 700. Muestre el valor total a pagar.

6. Lea un valor de temperatura t y un código p que puede ser 1 o 2. Si el código es 1 convierta la temperatura t de grados f a grados c con la fórmula $c = \frac{5}{9}(t-32)$. Si el código es 2 convierta la temperatura t de grados c a grados f con la fórmula $f = 32+ \frac{9}{5}t$.

7. Dadas las tres calificaciones de un estudiante, encuentre y muestre la calificación más alta.

8. Leer las dimensiones a, b, c de un bloque rectangular, y el diámetro d de un agujero. Determine si es posible que el bloque pueda pasar por el agujero. Debe encontrar la diagonal de menor longitud y compararla con el diámetro.

9. Dado un número par n, sume los primeros n números impares. Si n = 4, entonces s = 1 + 3 + 5 + 7 = 16. Verifique si esta suma s es igual n^2 y muestre un mensaje.

Estructuras Repetitivas

Como hemos visto en el capítulo anterior, las estructuras de control son una parte fundamental de cualquier lenguaje porque nos permiten modificar el orden de una programación secuencial. En este capítulo revisaremos las estructuras repetitivas, sus tipos, sus sintaxis y sus representaciones en diagrama de flujo. Además, presentaremos dos sentencias que nos permitirán salir de los bucles de repetición mediante el uso de estructuras condicionales.

Estructuras de Control Repetitivas

Los bucles o repeticiones permiten que se ejecute repetidamente un conjunto de instrucciones, bien un número predeterminado de veces, o bien hasta que se verifique una determinada condición [7]. Existen dos tipos de estructuras de repetición:

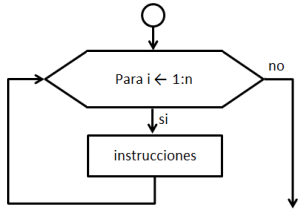
1. Estructuras de Repetición Indexada: FOR
2. Estructura de Repetición Condicional: WHILE

A estas estructuras de repetición, están asociadas dos tipos de sentencias para la ruptura de ciclos de repetición: BREAK y CONTINUE.

Estructura Repetitiva FOR

La estructura repetitiva o de bucle FOR realiza n repeticiones o iteraciones. La estructura repetitiva FOR se representa en el diagrama de flujo de la Figura 22, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

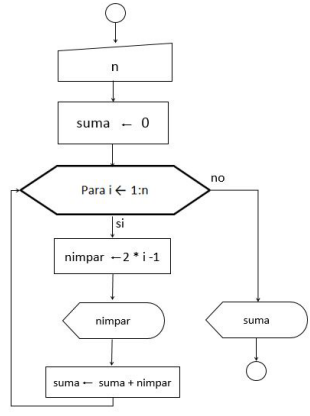
Figura 22: Estructura Repetitiva FOR




```
1 % Estructura Repetitiva FOR
2
3 for i=1:n
4     instrucciones;
5 end
```

Ejemplo: Dado un entero, n, calcular la suma de los n primeros números impares.
A continuación se representa el diagrama de flujo en la Figura 23.

Figura 23: Diagrama de flujo de estructura repetitiva FOR

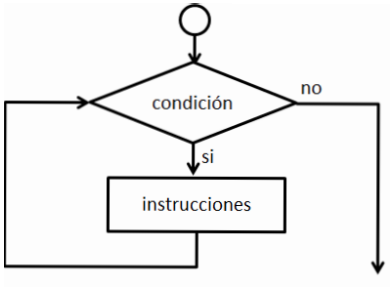



```
1 % Suma de los n primeros numeros impares
2
3 n = input('Ingrese el valor de n: ');
4 suma = 0;                          % inicializa la suma
5
6 for i=1:n                          % indice
7     nimpar = 2*i - 1;              % instrucciones
8     disp(nimpar);
9     suma = suma + nimpar;
10 end
11
12 disp('La suma de los n numeros impares es: ');
13 disp(suma);
```

Estructura Repetitiva WHILE

La estructura repetitiva o de bucle WHILE permite implementar la repetición de un mismo conjunto de instrucciones mientras se verifique que una determinada condición es verdadera. Si el resultado es falso, se detiene la ejecución del ciclo WHILE. La estructura repetitiva WHILE se representa en el diagrama de flujo de la Figura 24, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

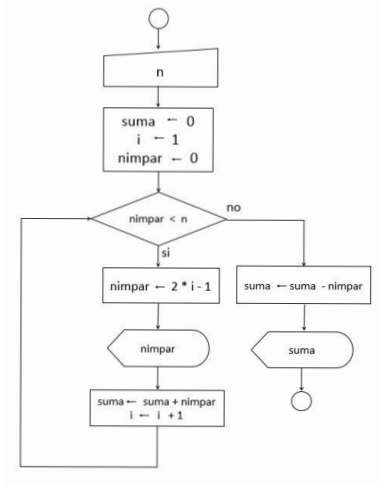
Figura 24: Estructura Repetitiva WHILE




```
1 % Estructura Repetitiva WHILE
2
3 while condicion
4     instrucciones;
5 end
```

Ejemplo: Dado un entero, n, calcular la suma de los primeros números impares menores a n. A continuación se representa el diagrama de flujo en la Figura 25.

Figura 25: Diagrama de flujo de repetitiva WHILE

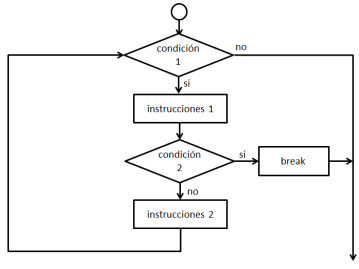


```
1 % Suma de los primeros numeros impares menores a n.
2
3 n = input('Ingrese el valor de n: ');
4 suma = 0;              % inicializa la suma
5 i = 1;                 % inicializa el indice
6 nimpar = 0;            % inicializa la variable que guarda numeros impares
7
8 while nimpar < n       % condicion
9     nimpar = 2*i - 1;  % instrucciones
10    disp(nimpar);
11    suma = suma + nimpar;
12    i = i + 1;
13 end
14
15 suma = suma - nimpar; % Se resta el ultimo numero impar que es mayor o igual a n
16 disp('La suma de los numeros impares menores a n es: ');
17 disp(suma);
```

Interrupción con BREAK

En ocasiones es necesario interrumpir la ejecución de un ciclo de repetición en algún punto interno del bloque de instrucciones que se repiten mediante alguna condición. Cuando la interrupción se hace para abandonar el ciclo de repetición definitivamente, se hace uso de la interrupción con BREAK. La interrupción con BREAK se representa en el diagrama de flujo de la Figura 26, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

Figura 26: Interrupción con BREAK

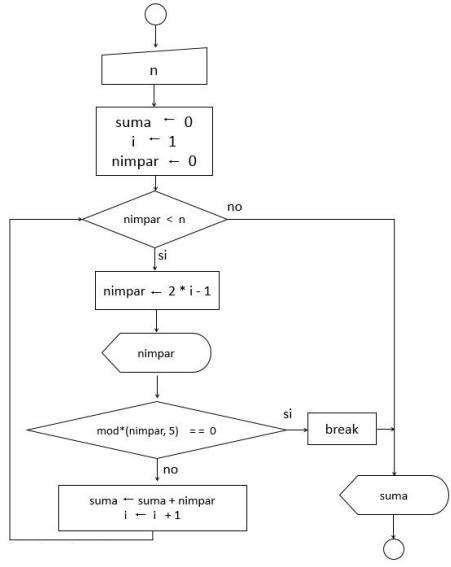



```
1 % Interrupcion con BREAK
2
3 while condicion1
4     instrucciones 1
5     if condicion2
6         break;
7     else
8     instrucciones 2
9     end
10 end
```

Ejemplo: Dado un entero, n, calcular la suma de los primeros números impares menores a n; descartando aquellos números múltiplos de 5.

A continuación se representa el diagrama de flujo en la Figura 27.

Figura 27: Diagrama de flujo de repetitiva BREAK



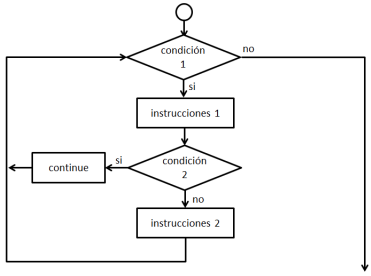

```
1 % Suma de los primeros numeros impares menores a n, terminando cuando
aparece 5.
2
3 n = input('Ingrese el valor de n: ');
4 suma = 0;               % inicializa la suma
5 i = 1;                  % inicializa el indice
6 nimpar = 0;             % inicializa la variable que guarda numeros impares
7
8 while nimpar < n        % condicion 1
9     nimpar = 2*i - 1;   % instrucciones
10    disp(nimpar);
11    if mod(nimpar,5) == 0 % condicion 2
12        break;
13    end
14    suma = suma + nimpar;
15    i = i + 1;
16 end
17
18 disp('La suma de los numeros impares menores a n es: ');
19 disp(suma);
```

Interrupción con CONTINUE

Cuando la interrupción se hace para abandonar la iteración en curso, pero comenzando la siguiente; se hace uso de la interrupción con CONTINUE. La interrupción con CONTINUE se representa en el

diagrama de flujo de la Figura 28, mostrando en el lado derecho la sintaxis de esta estructura en MATLAB.

Figura 28: Interrupción con CONTINUE

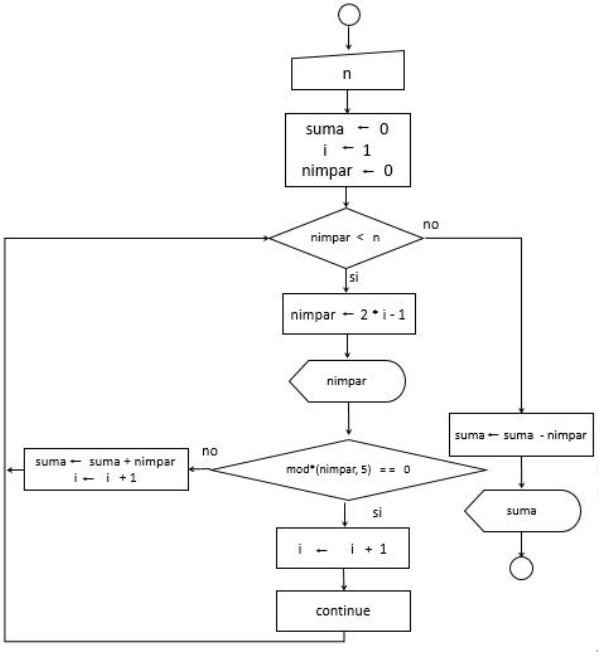



```
1 % Interrupcion con CONTINUE
2
3 while condicion1
4     instrucciones 1
5     if condicion2
6         continue;
7     else
8         instrucciones 2
9     end
10 end
```

Ejemplo: Dado un entero, n, calcular la suma de los primeros números impares menores a n; descartando aquellos números múltiplos de 5.

A continuación se representa el diagrama de flujo en la Figura 29.

Figura 29: Diagrama de flujo de repetitiva CONTINUE



```
1 % Suma de los primeros numeros impares menores a n, descartando aquellos
numeros multiplos de 5.
2
3 n = input('Ingrese el valor de n: ');
4 suma = 0;              % inicializa la suma
5 i = 1;                 % inicializa el indice
6 nimpar = 0;            % inicializa la variable que guarda numeros impares
7
8 while nimpar < n       % condicion 1
9 nimpar = 2*i - 1;      % instrucciones
10 disp(nimpar);
11 if mod(nimpar,5) == 0 % condicion 2
12 i = i + 1;
13 continue;
14 end
15 suma = suma + nimpar;
16 i = i + 1;
17 end
18
19 suma = suma - nimpar; % Se resta el ultimo numero impar que es mayor o
igual a n
20
21 disp('La suma de los numeros impares menores a n es: ');
22 disp(suma);
```

Ejercicios sobre Estructuras Repetitivas

1. Describa en una tabla, el valor de las variables j, x, y; en cada iteración generada por los algoritmos representados en las Figuras 30 y 31 durante su ejecución.

Figura 30: Ejercicio 1 con Diagrama de flujo de Bucles

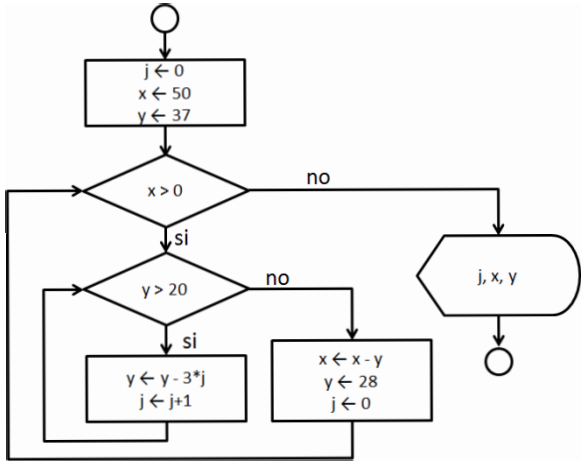
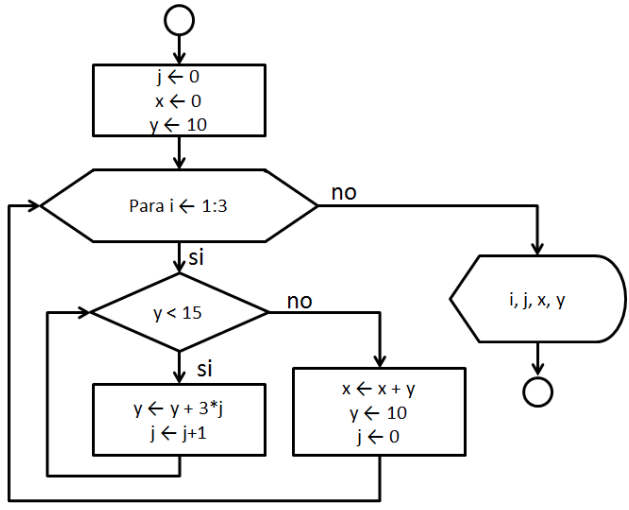


Figura 31: Ejercicio 2 con Diagrama de flujo de Bucles



2. Represente mediante un diagrama de flujo la suma de los primeros n números múltiplos de 3, sin tomar en cuenta aquellos valores que son pares. El programa debe mostrar la suma total y la cantidad de números considerados en la suma total.

3. Determine cuantas veces el programa despliega el mensaje: listo!

```
1 x =3;
2 while x < 14
3     disp('listo!')
4     x = x + 3.5;
5 end
```

La respuesta es

4. Reescriba una forma alternativa del mismo programa que se presenta a continuación, conservando su funcionamiento. Debería ser capaz únicamente de cambiar la síntaxis de la estructura for por una estructura while.

```
1 n = 10;
2 x = 0;
3 y = 0;
4 for i=1:n
5     x = x + 5;
6     y = x + 3;
7 end
```

Los siguientes ejercicios han sido propuestos por el Profesor Luis Rodríguez en [6].

5. Dadas los tres lados de un triángulo, determine su tipo: escaleno, isósceles, o equilatero.

6. Calcule el promedio del menor y mayor valor de los pesos de n paquetes en una bodega. Estos datos se ingresan uno a la vez dentro de un ciclo. n es un dato ingresado al inicio.

7. Dados dos números enteros a, b, determine su máximo común divisor m. Si a = 36, b = 45 entonces m = 9.

8. Dado un número entero positivo n, descompóngalo en sus factores primos. Si n = 72, entonces los factores primos son 2,2,2,3,3.

9. Lea los votos de n personas. Cada voto es un número 1, 2 o 3 correspondiente a tres candidatos. Si el dato es 0 es un voto en blanco. Si es otro número es un voto nulo. Determine el total de los votos de cada candidato y el total de votos blancos y nulos.

10. Lea las coordenadas (u,v) de la ubicación de una fábrica y las coordenadas (x,y) de n sitios de distribución. Encuentre cuál es la distancia del sitio más alejado de la fábrica.

11. Encuentre el mayor valor de la función f(x) = sen(x)+ln(x) para los valores x = 1.0, 1.1, 1.2, 1.3, ..., 4.

12. Determine la suma de los n primeros números de la serie: 1, 1, 2, 3, 5, 8, 13, 21, ... En cada término, a partir del tercero, se obtiene sumando los dos términos anteriores.

Arreglos en MATLAB

Hasta el momento, se ha revisado variables numéricas y lógicas para la elaboración de programas secuenciales y estructurados. En este capítulo se presentan los arreglos unidimensionales representados por vectores y los arreglos bidimensionales representados por matrices. Se hace una amplia descripción sobre la manipulación de la información que estas estructuras almacenan.

Arreglo Unidimensional

Los vectores son dispositivos para representar y manejar variables que pueden tener varias componentes.

Manipulación de vectores

Sintaxis de un vector

nombre = [componentes]

Notación para las componentes de un vector.

Notación Matemática: x

Notación de las componentes en MATLAB: x(i)

x es el nombre del vector.
i es el número de la celda(numeradas en forma natural desde 1).

Crear un vector fila de tres componentes

Command Window

```
>> x = [5 , 7, 4];
>> x
    x =
        5 7 4
fx
```

Las componentes pueden separarse con espacios o con comas. Si se desea crear un vector columna, los elementos deben separarse con punto y coma.

Crear un vector columna de tres o mas componentes.

Command Window

```
>> t = [6; 9;3];
>> t
    t =
        5 7 4
fx
```

Los vectores pueden manejarse dinámicamente:

Agregar un nuevo elemento al final del vector x anterior:

Command Window

```
>> x = [x, 8];
>> x
     x =
         5 7 4 8
fx
```

Agregar un nuevo elemento al inicio del vector x anterior:

Command Window

```
>> x = [6 , x] ;
>> x
     x =
         6 5 7 4 8
fx
```

Eliminar el tercer elemento del vector x

Command Window

```
>> x (3) = [ ] ;
>> x
     x =
         6 5 4 8
fx
```

Eliminar el vector completo

Command Window

```
>> x = [ ] ;
>> x
     x =
         [ ]
fx
```

También se puede crear un vector asignando valores a sus componentes:

Crear un vector fila de tres componentes.

Command Window

```
>> x (1) = 5;
>> x (2) = 7;
>> x (3) = 4;
>> x
     x =
         5 7 4
fx
```

El manejo individual de los componentes requiere el uso de un índice.

Crear un vector fila.

Command Window

```
>> x = [6 , 7, 4, 8, 3];
>> x
     x =
         6 7 4 8 3
fx
```

Mostrar el tercer componente.

Command Window

```
>> x (3) ;
   ans =
        4
fx
```

Sustituir el cuarto componente por el 9.

Command Window

```
>> x (4) = 9;
    x =
        6 7 4 9 3
fx
```

Mostrar los componentes 2, 3 y 4.

Command Window

```
>> x (2:4) ;
    ans =
         7 4 9
fx
```

Insertar en la posicion 3 el valor 8.

Command Window

```
>> x = [x(1:2) , 8, x(3:5) ] ;
>> x
    x =
        6 7 8 4 9 3
fx
```

Creación de vectores

Crear un vector fila usando el comando: input

Command Window

```
>> y = input ( ' ingrese un vector : ' ) ;

ingrese un vector : [10 12 14 16 18 20]
       y =
          10 12 14 16 18 20
fx
```

Crear un vector fila usando la sintaxis inicio:incremento:fin

Command Window

```
>> y = 10:2:20

    y =
        10 12 14 16 18 20
fx
```

Si no se define un valor de incremento, se asume por defecto que el incremento es 1.

Command Window

```
>> y = 10:20

     y =
         10 11 12 13 14 15 16 17 18 19 20
fx
```

Crear un vector fila usando el comando:
linspace(inicio, fin, número de componentes)

Command Window

```
>> y = linspace (10 , 20 ,6)

     y =
         10 12 14 16 18 20
fx
```

Comandos

Algunas funciones de MATLAB para manejo de vectores.

Longitud de un vector.

Command Window

```
>> v = [2 4 7 3 5 8 6];
>> n = length (v) ;
    n =
        8
fx
```

Suma de las componentes en el vector.

Command Window

```
>> s = sum(v) ;
    s =
        42
fx
```

Máximo valor de las componentes en el vector.

Command Window

```
>> m = max(v) ;
    m =
        8
fx
```

Máximo valor de las componentes y su posición en el vector.

Command Window

```
>> [m, p ] = max(v) ;
    m =
        8
    p =
        7
fx
```

Media o promedio aritmético.

Command Window

```
>> p = mean(v) ;
    p =
        3 5.2500
fx
```

Determinar si algún elemento pertenece al vector.

Command Window

```
>> v = [2 4 7 3 5 8 6];
>> e = ismember(8,v) ;
    e =
        1
>> e =ismember(9,v) ;
    e =
        0
fx
```

Determinar adicionalmente la posición del elemento en el vector.

Command Window

```
>> [ e ,p] = ismember (8 ,v) ;
          e =
          1
     p =
          7
fx
```

Determinar el producto cruz entre los vectores a y b

Command Window

```
>> a = [ 1 3 2];
>> b = [ -2 2 -2];
>> c = cross (a , b)

    c =
        -10   -2    8
fx
```

Determinar el producto punto entre los vectores a y b

Command Window

```
>> a = [ 1 3 2];
>> b = [ -2 2 -2];
>> c = dot (a , b)

    c =
        0
fx
```

La raiz de un vector.

Command Window

```
>> x = [0 ,1 ,4 ,9 ,16 ,25];
>> z = sqrt (x)

    z =
       0    1    2    3    4    5
fx
```

Operaciones entre vectores

Considere a los vectores a y b, que poseen igual número de componentes. Entonces, se puede realizar las siguientes operaciones entre vectores:

Suma de vectores

Command Window

```
>> x = a + b ;
fx
```

Resta de vectores

Command Window

```
>> x = a - b ;
fx
```

Multplicación de vectores

Command Window

```
>> x = a * b ;
fx
```

Multplicación elemento a elemento entre vectores

Command Window

```
>> x = a .* b ;
fx
```

División elemento a elemento entre vectores

Command Window

```
>> x = a ./ b ;
fx
```

Potenciación del elemento a elemento del vector.

Command Window

```
>> x = a .^2 ;
fx
```

Ejercicios sobre Vectores

Ejecute las siguientes instrucciones en la ventana de comandos de MATLAB:

1. (1:7)
2. 1:7
3. 1:3:10
4. 1:4:10
5. 1:0.1:1
6. 50:-7:1
7. linspace(2,6,3)
8. linspace(2,6,4)
9. linspace(2,10)

Sean a=[1 2 3], b=[-2 3 5])

1. Sume 3 a cada elemento de a y divida cada elemento de b entre 2.
2. Realiza las operaciones que se indican: a+b, a-b, a.b y a x b.
3. Divide los elementos de a entre los elementos de b
4. Un vector columna que contenga los números impares entre 1 y 1000.
5. Un vector fila que contenga los números pares entre 2 y 1000.
6. Si x=0:2:20, escribe el comando de MATLAB que eleva al cuadrado cada componente de x.
7. Verifique si x=(1,3,2) e y=(-2,2,-2) son ortogonales.
8. Obtenga un vector ortogonal x=(1,3,2) e y=(2,2,-2).
9. Si x=[0,1,4,9,16,25], calcula la raíz cuadrada de cada componente de x.
10. Si x=0:.1:1, eleva cada componente de x a 2/3.
11. Si x=0:pi/2:2*pi, calcula el coseno de cada componente de x.
12. Si x=-1:.1:1, calcula el seno inverso de cada componente de x.
13. Si x=linspace(0,2*pi,1000), ¿cuál es la entrada 50 de x? ¿Cuál es la longitud de x?

14. Si k=0:100, ¿cuál es la entrada número 12 de y = 0,5. ^ k?
15. Evaluar la expresión $\frac{sen(x)}{x}$

para valores de x entre −1 y 1 con incrementos de 0.1 unidades.

Arreglo Multidimensional

Una matriz es considerado un arreglo multidimensional que recoge un conjunto de variables, cuyo acceso se realiza por índices o líneas, o subarreglos de números organizados en filas y columnas.

Manipulación de matrices

Creación de una matriz a partir de vectores fila

```
Command Window

>> a = [10 20 30 40; 5 7 9 11; 13 15 17 19; 2 4 6 8];
>> a
a =

    10   20   30   40
     5    7    9   11
    13   15   17   19
     2    4    6    8
fx
```

Creación de una matriz a partir de vectores columna

```
Command Window

>> a = [10;20;30;40]
>> b = [5;7;9;11]
>> c = [13;15;17;19]
>> d = [2;4;6;8]
>> x = [a,b,c ,d]
x =

     10    5    13    2
     20    7    15    4
     30    9    17    6
     40   11    19    8
fx
```

Considere la matriz a que se presenta a continuación.

Command Window

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

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

Acceder a un elemento de la fila 3, columna 2.

Command Window

```
>> a = (3 ,2)
ans =
      9
fx
```

Cambiar el valor del elemento de la fila 3, columna 2.

Command Window

```
>> a (3 ,2) = 10
ans =
      2     5     7
      4     6     2
      8     10    3
fx
```

Acceder a la fila 2.

Command Window

```
>> d = a(2,:)
d =
   4     6     2
fx
```

Acceder a la columna 2.

Command Window

```
>> d = a(:,2)
d =
     5
     6
     9
fx
```

Acceder a una submatriz comprendida entre las filas 1 y 2, columnas 2 y 3

Command Window

```
>> d = a(1:2 , 2:3) ;
d =

      5     7
      6     2
fx
```

Acceder a una submatriz comprendida entre las filas 1 y 3, columnas 2 y 3.

Command Window

```
>> d = a(1:3 , 2:3) ;
d =
     5     7
     6     2
     9     3
fx
```

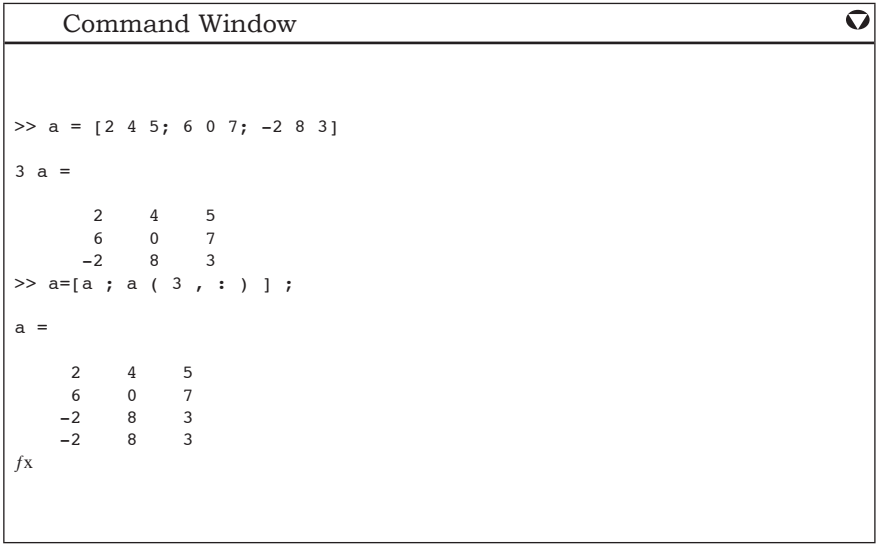
Acceder a una submatriz formada solo por las filas 1 y 3, y las columnas 1 y 3

Command Window

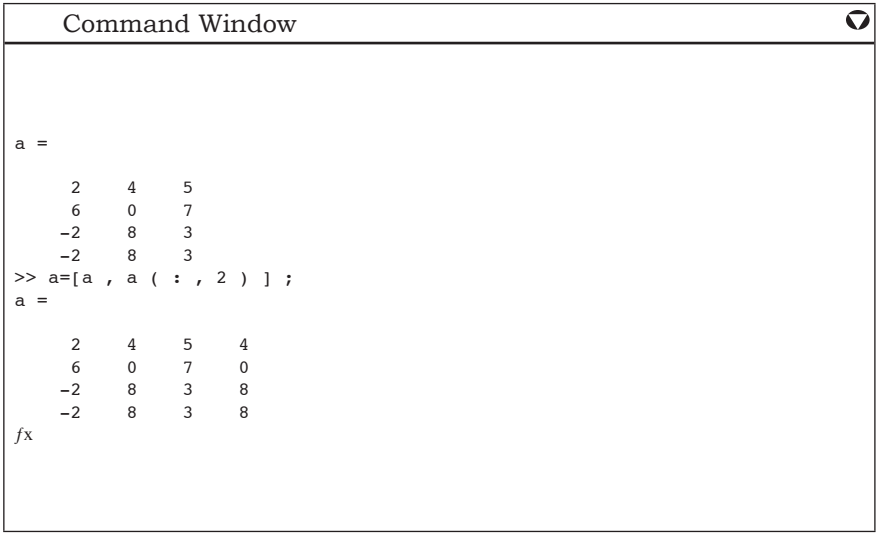
```
>> d = a([1 ,3] ,[1 ,3]) ;
d =
     2     7
     8     3
fx
```

Manipulación de filas y columnas.

Insertar la fila 3 de la matriz a en la fila 4.



Insertar la columna 2 de la matriz a en la columna 4.



Eliminar la fila 1 de la matriz *a*

Command Window

```
a =

      2     4     5     4
      6     0     7     0
     -2     8     3     8
     -2     8     3     8
>> a (1 , : )= [ ] ;
a =

      6     0     7     0
     -2     8     3     8
     -2     8     3     8
fx
```

Elimina la columna 3 de la matriz *a*

Command Window

```
a =

     6     0     7     0
    -2     8     3     8
    -2     8     3     8
>> a ( : , 3 )= [ ] ;
a =

     6     0     0
    -2     8     8
    -2     8     8
fx
```

Matrices Especiales

A continuación se presentan algunos comandos que permiten la generación de Matrices especiales

Matriz de ceros de 3 filas y 4 columnas.

Command Window

```
>> a = zeros (3 , 4 )

a =

     0     0     0     0
     0     0     0     0
     0     0     0     0
fx
```

Matriz de unos de 4 filas y 3 columnas.

Command Window

```
>> b = ones (4 ,3)

b =

     1     1     1
     1     1     1
     1     1     1
     1     1     1
fx
```

Matriz identidad de 4 x 4.

Command Window

```
>> c = eye (4 , 4 )

c =
      1     0     0     0
      0     1     0     0
      0     0     1     0
      0     0     0     1
fx
```

Matriz con números aleatorios entre 0 y 1, con 5 filas y 6 columnas.

Command Window

```
>> d =rand (5 , 6 )

d =
     0.8147    0.0975    0.1576    0.1419    0.6557    0.7577
     0.9058    0.2785    0.9706    0.4218    0.0357    0.7431
     0.1270    0.5469    0.9572    0.9157    0.8491    0.3922
     0.9134    0.9575    0.4854    0.7922    0.9340    0.6555
     0.6324    0.9649    0.8003    0.9595    0.6787    0.1712
fx
```

Matriz con números aleatorios entre 0 y 10, con 5 filas y 6 columnas.

Command Window

```
>> d =10*rand (5 , 6 )

d =
     7.0605    8.2346    4.3874    4.8976    2.7603    4.9836
     0.3183    6.9483    3.8156    4.4559    6.7970    9.5974
     2.7692    3.1710    7.6552    6.4631    6.5510    3.4039
     0.4617    9.5022    7.9520    7.0936    1.6261    5.8527
     0.9713    0.3445    1.8687    7.5469    1.1900    2.2381
fx
```

Matriz con números aleatorios enteros entre 0 y 10, con 5 filas y 6 columnas

Command Window

```
>> d =fix (10* rand (5 , 6 ) )

d =

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

Operaciones entre matrices

Considere las matrices a y b que poseen el mismo número de filas y el mismo número de columnas.

Suma de matrices

Command Window

```
>> x = a + b ;
fx
```

Resta de matrices.

Command Window

```
>> x = a - b ;
fx
```

Multplicación de matrices (el número de filas de a es igual al número de columnas de b)

Command Window

```
>> x = a * b ;
fx
```

Multplicación elemento a elemento de matrices.

Command Window

```
>> x = a .* b ;
fx
```

División elemento a elemento de matrices.

Command Window

```
>> x = a ./ b ;
fx
```

Comandos Especiales

Dimensiones de una matriz

Command Window

```
>> a = [2 4 5; 6 0 7; -2 8 3];

a =

     2 4 5
     6 0 7
    -2 8 3
>> [ n ,m]=size ( a ) ;
n =
    3
m =
    3
fx
```

Suma entre columnas

Command Window

```
>> c = sum( a ) ;
c =
    6    12    15
fx
```

Suma entre filas

Command Window

```
>> f = sum(a') ;
f =
    11
    13
    9
fx
```

Suma de todos los elementos

Command Window

```
>> s = sum(sum( a ) ) ;
s =
    33
fx
```

El mayor valor de cada columna

Command Window

```
>> max( a )
ans =
      6    8    7
fx
```

Elementos de la diagonal de una matriz.

Command Window

```
>> a = [2 4 5; 6 0 7; -2 8 3]

a =

     2    4    5
     6    0    7
    -2    8    3
>> d =diag ( a )
 d =
     2
     0
     3
fx
```

Matriz diagonal a partir de un vector columna.

Command Window

```
>> e =diag (d)
e =
    2    0    0
    0    0    0
    0    0    3
fx
```

Determinante de una matriz.

Command Window

```
>> a = [2 4 5; 6 0 7; -2 8 3]

a =

     2    4    5
     6    0    7
    -2    8    3
>> det ( a )

ans =

2.1316 e -14
fx
```

Inversa de una matriz.

Command Window

```
>> a = [2 4 5; 6 0 7; -2 8 3]

a =

     2 4 5
     6 0 7
    -2 8 3
>> inv ( a )

ans =

     1 .0 e+15 *

     -2.6271    1.3135    1.3135
     -1.5012    0.7506    0.7506
      2.2518   -1.1259   -1.1259
fx
```

Ejercicios sobre Matrices

1. Almacena en memoria principal la siguiente matriz, en una variable que se llame M1.

$$\begin{pmatrix} 1 & 2 & 3 \\ -3 & -4 & 4 \\ 3 & 7 & 2 \end{pmatrix}$$

2. Calcule la transpuesta de M1 y guardala en M2.

3. Calcule el producto elemento a elemento de M1 y M2.

4. Calcule la suma M1 y M2.

5. Calcule la division elemento a elemento de M1 y M2.

6. Calcula el producto matricial de M1 y M2 y guardalo en PRODM1M2.

7. Calcula el producto matricial de M2 y M1 y guárdalo en PRODM2M1.

8. Calcula la división matricial de M1 y M2.

9. Cambia el valor del elemento central de M1 a 9.

10. Guarda en una matriz llamada ESQUINASM1 de tamaño 2x2 los elementos de las esquinas de M1.

11. Guarda en un vector fila V, los elementos de la diagonal principal de M1.

12. Guarda en un vector columna W, los elementos de la diagonal secundaria de M2.

13. Calcula el producto escalar de V y W.

14. Calcula el producto vector de V y W.

15. Guarda en FILA1, los elementos de la primera fila de la matriz M1.

16. Guarda en COLUMNA1, los elementos de la primera columna de la matriz M1.

17. Convierte FILA1 en un vector columna y COLUMNA1 en un vector fila.

Resolver los siguientes sistemas de ecuaciones linelaes

$$\begin{cases} x_1 - 2x_2 + x_3 + x_4 = 1 \\ 3x_1 + 2x_3 - 2x_4 = 0 \\ 4x_2 - x_3 - x_4 = 0 \\ 5x_1 + 3x_3 - x_4 = 2 \end{cases}$$

$$\begin{cases} x_1 + x_2 - x_3 = 15 \\ 4x_1 - 2x_2 + 7x_3 = 5 \end{cases}$$

$$\begin{cases} x_1 + x_2 - x_3 = 6 \\ 3x_1 - 3x_2 + 2x_3 = 3 \\ -x_1 - 11x_2 + 6x_3 = 12 \end{cases}$$

Graficación en MATLAB

En el presente capítulo, se propone un conjunto de estrategias para el desarrollo de gráficos en 2 y 3 dimensiones. La familiaridad con el uso de vectores y matrices es imperativa en el planteamiento de los dominios de las gráficas. Ésto permite fundamentalmente la visualización de la información para un apropiado análisis de los fenómenos presentes en la naturaleza. Además, las herramientas gráficas de MATLAB proporcionan robustez en la presentación de resultados de los trabajos investigativos en Ciencias e Ingeniería.

Gráficos Bidimensionales 2D

Las gráficas 2D de MATLAB están fundamentalmente orientadas a la representación gráfica de vectores y matrices.

Para hacer gráficas de funciones de la forma y = f(x), de una variable con MATLAB, primero se debe crear un vector de valores de la variable para después dibujar la función.

Comando plot(x,y)

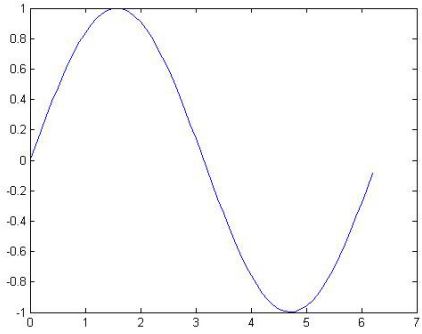
Este comando dibuja los pares de puntos (x, y) unidos por segmentos (x e y representan las componentes de los vectores x e y respectivamente).

Ejemplo: Dibujar y = sen(x). Primero creamos un vector de valores para x de la siguiente manera x=0:0.1:2*π.

Ahora calculamos los valores de y como se observa en la Figura 32.

```
1 x = 0:0.1:2*pi
2 y = sin(x);
3 figure(1);
4 plot(x,y);fx
```

Figura 32: Función y = sen(x)

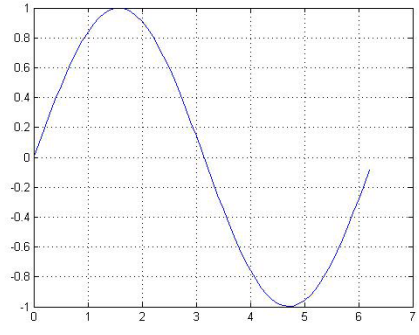


Comando grid on

Si se desea que aparezca una cuadrícula sobre la gráfica de la función tal como aparece en la Figura 33.

```
1 x = 0:0.1:2*pi
2 y = sin(x);
3 figure(1);
4 plot(x,y);
5 grid on:
```

Figura 33: Función y = sen(x) con cuadrícula



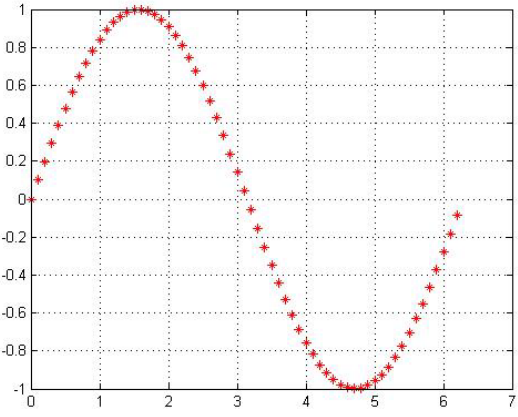
Comando plot(x,y,'r*')

El comando plot ofrece múltiples posibilidades de color y forma de trazo de la gráfica plot(x,y,'r*') tal como aparece en la Figura 34.

Ejemplo: y = sen(x)

```
1 x = 0:0.1:2*pi
2 y = sin(x);
3 figure(1);
4 plot(x,y'r*');
5 grid on;
```

Figura 34: Función y = sen(x) trazada con asteriscos rojos

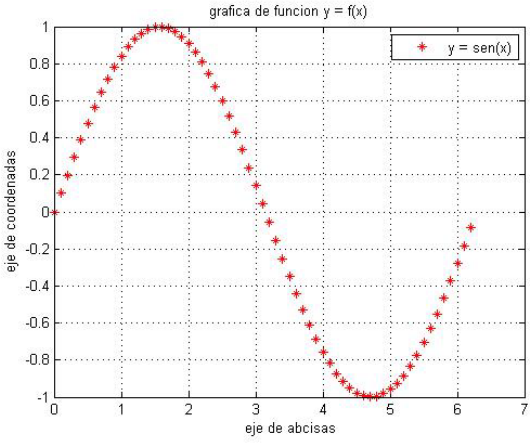


Comandos de etiqueta

Existen diversas posibilidades para el etiquetado de las gráficas como se puede apreciar en la Figura 35.

```
1 x = 0:0.1:2*pi
2 y = sin(x);
3 figure(1);
4 plot(x,y,'r*');
5 grid on;
6 title('Grafica de funcion y = f(x)');
7 xlabel('Eje de abcisas');
8 ylabel('Eje de coordenadas');
9 legend('y = sen(x)');
```

Figura 35: Función y = sen(x) con etiquetado en el eje x, en el eje y, y el título.



Para graficar funciones por partes, se observa los tramos donde se satisface el dominio de la función utilizando unos y ceros. Para crear los dominios de los tramos en que se define la función, se puede utilizar los siguientes operadores relacionales:

<	menor que
>	mayor que
<=	menor o igual
>=	mayor o igual
==	igual
~=	distinto

Estos operadores se pueden combinar utilizando los operadores lógicos:

&	y
|	o
~	no

Ejemplo:

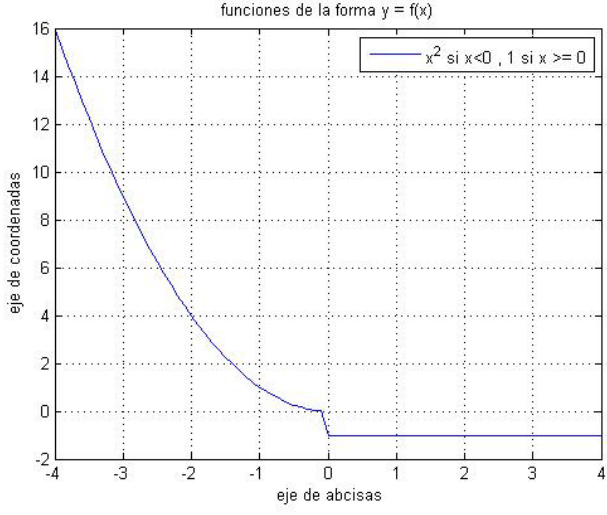
$$\begin{cases} x^2 & \text{si } x < 0 \\ -1 & \text{si } x \geq 0 \end{cases}$$

Ahora se define la función, multiplicando cada porción por el dominio que describa el lugar en el que queremos generar la función como se observa en la Figura 36

$$y = (x. \wedge 2). * (x < 0) + (-1). * (x >= 0);$$

```
1 x = -4:0.1:4;
2 y = (x.^2).*(x<0)+(-1).*(x>=0);
3 figure(4);
4 plot(x,y);
5 grid on;
6 title('funciones de la forma y = f(x)');
7 xlabel('Eje de abcisas');
8 ylabel('Eje de coordenadas');
9 legend('x^2 si x<0 , 1 si x>=0);
```

Figura 36: Trazado de funciones por partes.



Comandos para curvas paramétricas

Esta sección tiene como propósito representar curvas en el plano dadas en forma paramétrica como se presenta a continuación.

$$r(t) = (x(t), y(t)),\ t \in [a, b]$$

Ejemplo: Representar la curva que se presenta a continuación

$$r(t) = \frac{t^2 - 1}{t^2 + 1}, \frac{t^2 - 1}{t^2 + 1}, -5 \leq t \leq 5$$

En primer instancia, se procede a generar los valores de t en el intervalo indicado:

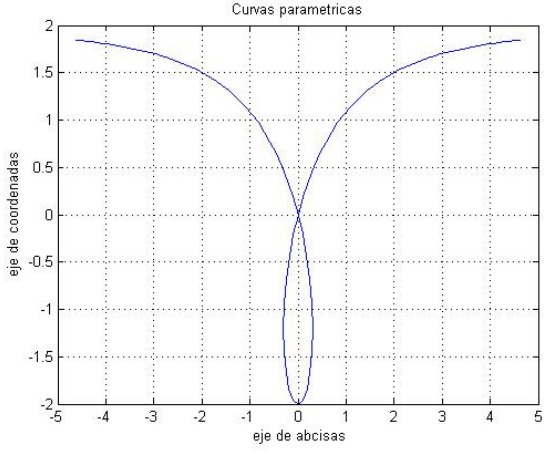
$$t = -5 : 0{,}1 : 5;$$

A continuación, se define las coordenadas (x, y) de la función paramétrica como se observa en la Figura 38.

```
x = t.* (t. ^ 2 − 1)./(t. ^ 2 + 1);
y = 2.* (t. ^ 2 − 1)./(t. ^ 2 + 1);
```

```
1 t = -5:(0.1):5;
2 x = t.*(t.^2 -1)./(t.^2 +1);
3 y = 2.*(t.^2 -1)./(t.^2 +1);
4 figure(5);
5 plot(x,y);
6 grid on;
7
8 title('Curvas parametricas')
9 xlabel('Eje de abcisas')
10 ylabel('Eje de coordenadas')
```

Figura 37: Trazado de curvas paramétricas.



Y otra forma de hacerlo es utilizar el comando comet. Los dos comandos producen el mismo resultado, sin embargo, la forma de ejecución es diferente. El comando comet permite aparecer un circulito (el cometa) que va dibujando la curva. La velocidad de ejecución depende del número de puntos que hayamos generado en el intervalo.

Comando para curvas polares

Una curva en coordenadas polares es la imagen de la función

$$r = h(\theta);\ \theta \epsilon\ [\theta_1\ ,\ \theta_2]$$

Un punto de la curva en polares (r_0, θ_0) tiene distancia al origen r_0 y el ángulo que forma el vector de posición del punto con el eje horizontal, medido en sentido positivo, es θ_0.

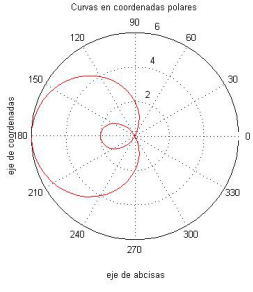
Para dibujar una curva en polares con MATLAB se utiliza el comando polar. Por ejemplo, para dibujar la gráfica de

$$r = 2 - 4\cos(\theta); -\pi \leq \theta \leq \pi$$

Generamos los valores del ángulo θ y calculamos los valores de r.

```
1 teta = -pi:0.1:pi,
2 r =2 - 4 * cos(teta),
3 figure(6);
4 polar(teta,r,'yellow');
5 title('Curvas en coordenadas polares')
6 xlabel(' eje de abcisas')
7 ylabel(' eje de coordenadas')
```

Figura 38: Trazado de curvas polares.



Ejercicios sobre Graficación 2D

Los siguientes ejercicios han sido propuestos por el Profesor Roberto Rodríguez del Río en [8].

Funciones de la forma y = f(x)

1. $f(x) = \dfrac{x(x-2)}{(x+1)\ \ (x+2)}$

2. $f(x) = sen(1/x)$

3. $f(x) = \dfrac{x}{e^{|x-1|}}$

4. $f(x) = \begin{cases} x^2 & si\ x < 0 \\ -1 & si\ x \geq 0 \end{cases}$

5. $f(x) = \begin{cases} \sqrt{1-x} & si\ x < -1 \\ 1-x^2 & si\ -1 < x < 1 \\ \sqrt{x-1} & si\ x < 1 \end{cases}$

Curvas Paramétricas

1. $\vec{r}(t) = (2\cos^3(t), 2sen^3(t)),\ -\pi \leq t \leq \pi$
2. $\vec{r}(t) = (\frac{3}{2}\ \cos(t)(\cos(t) + 1), 2sen(2t)),\ -\pi \leq t \leq \pi$
3. $\vec{r}(t) = (sen(2t) + sen(t), -\cos(2t) - \cos(t)),\ -\pi \leq t \leq \pi$
4. $\vec{r}(t) = (e^{\frac{t}{4}} sen(2t), e^{\frac{t}{4}} \cos(2t)),\ -\pi \leq t \leq \pi$
5. $\vec{r}(t) = (t - \frac{11}{10} sen(3t), -\frac{22}{10} \cos(3t)),\ -\pi \leq t \leq \pi$

Curvas Polares

1. $r = 7\ sen(\theta),\ -\pi \leq \theta \leq \pi$
2. $r = 3 - 6sen(\theta),\ -\pi \leq \theta \leq \pi$
3. $r = sen(6\theta),\ -\pi \leq \theta \leq \pi$
4. $r = \cos(8\theta),\ -\pi \leq \theta \leq \pi$
5. $r = \sqrt{5\cos(2\theta)},\ -\pi \leq \theta \leq \pi$

Gráficos Tridimensionales 3D

Estos gráficos se generan de una manera similar a las gráficas bidimensionales, con la diferencia de que aqui se utilizan los comandos plot3

o comet3, tambien existe un comando quiver3 para dibujar vectores velocidad sobre las curvas.

Ejemplo: Dibujar la hélice que describa los vectores velocidad.

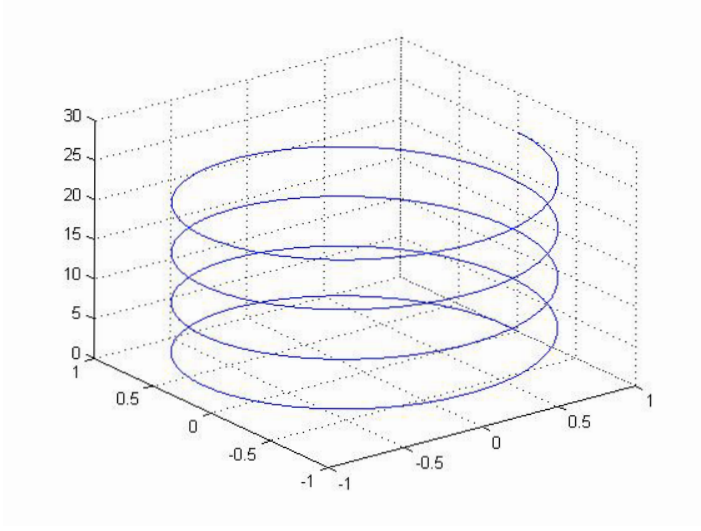
r = (t) (cos(t),sin(t),t),0 ≤ t ≤ 8π

Comando plot3(x,y,z)

Para usar este comando, se debe generar un vector para los valores de t en el intervalo señalado en el problema. Luego, se utiliza el comando plot3 como se muestra en la Figura 39.

```
1 t = 0:0.01:8*pi;
2 plot3(cos(t),sin(t),t),grid on;
```

Figura 39: Trazado de curvas tridimensionales con plot3(x,y,z)



El comando comet3 funciona de manera análoga a como lo hacía el comando comet en las curvas en el plano.

comet3(sin(t), cos(t), t)

Comando quiver3

El comando quiver3 posee la siguiente sintaxis

$$quiver3 = (x(t), y(t), z(t), \frac{dx(t)}{dt}, \frac{dy(t)}{dt}, \frac{dz(t)}{dt})$$

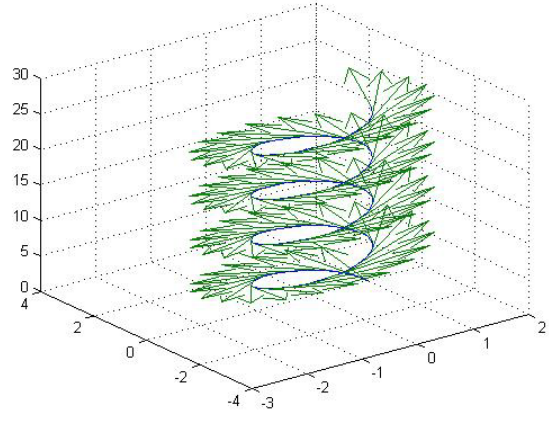
Ejemplo: Trazar la curva paramétrica que se presenta a continuación, y sobre ella los vectores velocidad.:

$$r(t) = (sen(t), cos(t), t),\ 0 \leq t \leq 8\pi$$

Para usar este comando, se debe generar un vector para los valores de t en el intervalo señalado en el problema. Luego, se utiliza el comando quiver3 como se muestra en la Figura 40.

```
1 t = 0:0.3:8*pi;
2 x = cos(t);
3 y = sin(t);
4 z = t;
5 dx = -sin(t);
6 dy = cos(t);
7 dz = ones(1,length(t));
8 plot3(x,y,z),hold on, grid on;      %sirve para mantener la grafica
9 quiver3(x,y,z,dx,dy,dz);
```

Figura 40: Trazado de curvas tridimensionales con quiver3(x,y,z)



Comandos meshgrid y plot3

Para dibujar gráficos de funciones de dos variables z = f(x, y), al igual que para funciones de una variable, en primer lugar hay que generar vectores para las variables x e y. Ahora, es necesario generar un mallado sobre el plano XY. Para eso se utiliza el comando meshgrid.

Ejemplo: Dibujar la gráfica de la función

$$z = e^{-(x^2 + y^2)}$$

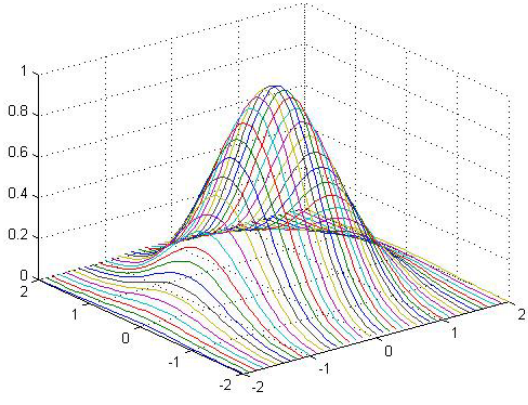
Primero se genera el mallado usando el siguiente comando

[x, y] = meshgrid(–2 : 0,1 : 2)

Sustituimos en la función para calcular los valores de z como se muestra en la Figura 41.

```
1 [x,y]=meshgrid(-2:0.1:2);
2 z = exp(-x.^2 - y.^2);
3 plot3(x,y,z),grid on;
```

Figura 41: Trazado de curvas tridimensionales con plot3(x,y,z)

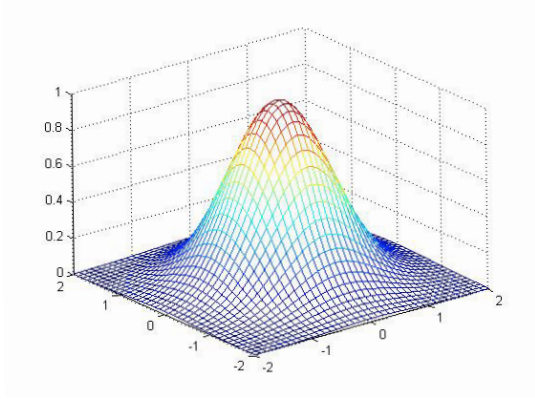


Comandos meshgrid y mesh

El comando mesh cambia la textura de la superficie graficada en 3 dimensiones, como se puede apreciar en la Figura 42

```
1 [x,y]=meshgrid(-2:0.1:2);
2 z = exp(-x.^2 - y.^2);
3 mesh (x,y,z);
```

Figura 42: Trazado de curvas tridimensionales con mesh(x,y,z)

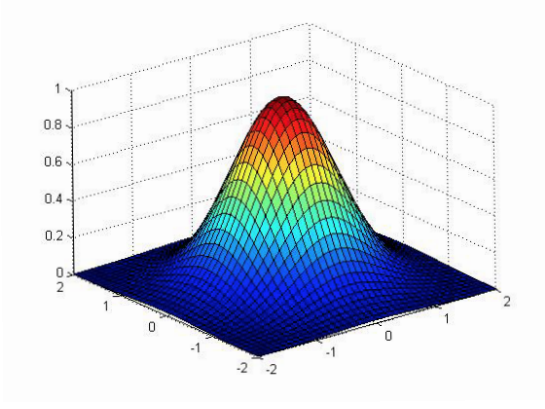


Comandos meshgrid y surf

El comando surf agrega color a la malla de la superficie graficada en 3 dimensiones, como se puede apreciar en la Figura 43.

```
1 [x,y]=meshgrid(-2:0.1:2);
2 z = exp(-x.^2 - y.^2);
3 surf (x,y,z);
```

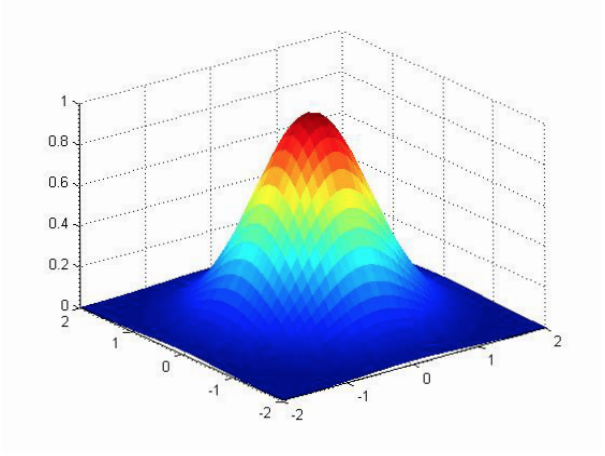
Figura 43: Trazado de curvas tridimensionales con surf(x,y,z)



El comando surf en combinación con el comando shading flat proporciona un efecto de sombreado como se puede apreciar en la Figura 44.

```
1 [x,y]=meshgrid(-2:0.1:2);
2 z = exp(-x.^2 - y.^2);
3 surf (x,y,z), shading flat;
```

Figura 44: Trazado de curvas tridimensionales con surf(x,y,z) y shading flat



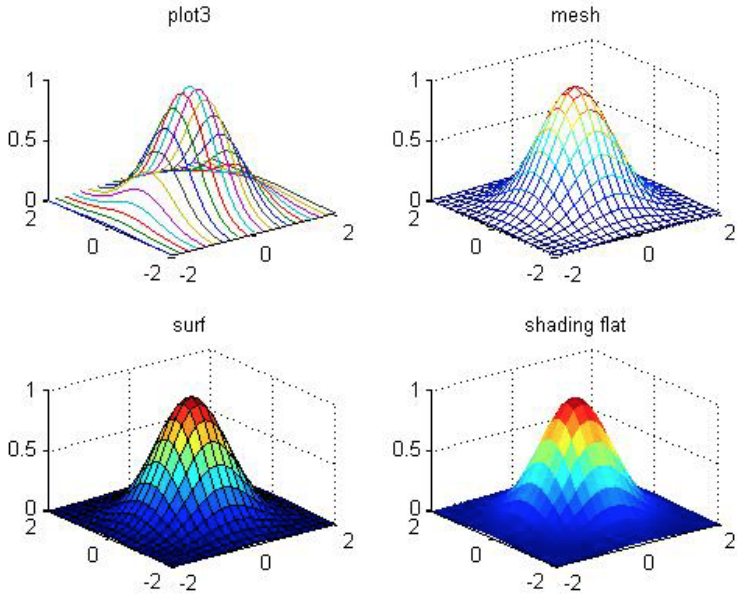
Comando subplot

El comando subplot(x,y,z) permite mostrar múltiples gráficas en una misma ventana de figura.

Los argumento del comando son: x = número de filas, y = número de columnas, z = la ubicación de la gráfica indexada de izquierda a derecha, y de arriba hacia abajo como se puede observar en la Figura 45.

```
1 [x,y]=meshgrid(-2:0.2:2);
2 z = exp(-x.^2 - y.^2);
3 subplot(2,2,1), plot3(x,y,z), title('plot3');
4 subplot(2,2,2), mesh(x,y,z), title('mesh');
5 subplot(2,2,3), surf(x,y,z), title('surf');
6 subplot(2,2,4), surf(x,y,z), shading flat, title('shading flat');
```

Figura 45: Representación de las cuatro figuras en una sola ventana mediante el comando subplot.



Comando colorbar

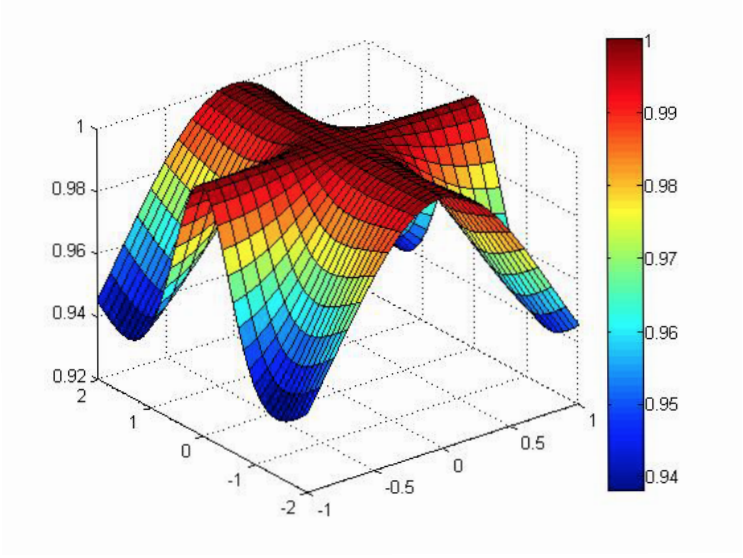
En la interpretación y visualización de la información puede resultar interesante añadir una escala de colores al dibujo con la finalidad de relacionar la tonalidad del color con variables como la altura (coordenada z) de los diferentes puntos de la gráfica. Ésto se consigue con el comando colorbar como se observa en la Figura 46.

Ejemplo: Dibujar la gráfica de la función que se presenta a continuación

$$z = \left(\frac{cos\,(xy)}{x^2 + y^2 + 1} \right)$$

```
1 [x,y]=meshgrid(-1:0.1:1, -2:0.1:2);
2 z = cos((x.*y)./(x.^2 + y.^2 + 1));
3 surf(x,y,z), colorbar;
```

Figura 46: Relación de los valores del relieve de los puntos con el color mediante el uso del comando colorbar



Comandos para curvas de nivel

Dada una función z = f(x, y), las curvas sobre el plano XY, determinadas por la ecuación f(x, y) = k, donde k es una constante se llaman curvas de nivel. Hay varias formas de obtenerlas usando MATLAB.

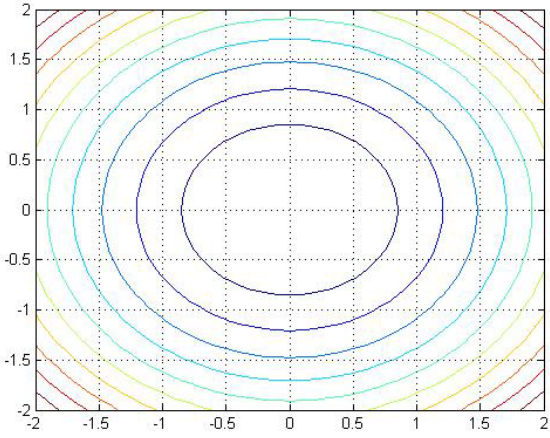
Ejemplo: Representar la gráfica de la función que se presenta a continuación, dibujando algunas curvas de nivel.

$$z = x^2 + y^2$$

Una vez creado el mallado con la función meshgrid, se calcula los valores de la función. Finalmente, se utiliza el comando contour(x,y,z,10) para dibujar 10 curvas de nivel como se muestra en la Figura 47.

```
1 [x,y]=meshgrid(-2:0.1:2);
2 z = x.^2 + y.^2;
3 contour(x,y,z,10);
```

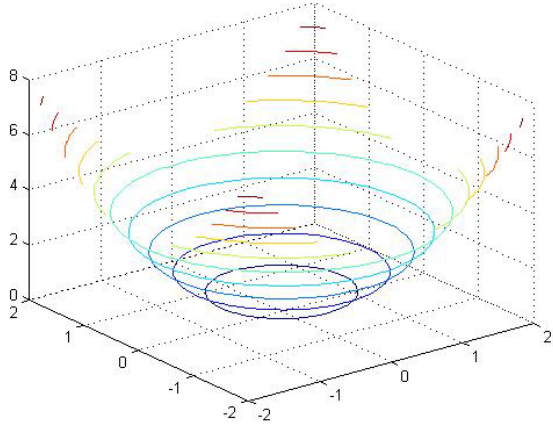
Figura 47: Curva de nivel de la función $z = x^2 + y^2$ usando el comando contour.



El comando contour3(x,y,z,10) permite dibujar 10 curvas de nivel en el espacio tridimensional como se muestra en la Figura 48.

```
1 [x,y]=meshgrid(-2:0.1:2);
2 z = x.^2 + y.^2;
3 contour3(x,y,z,10);
```

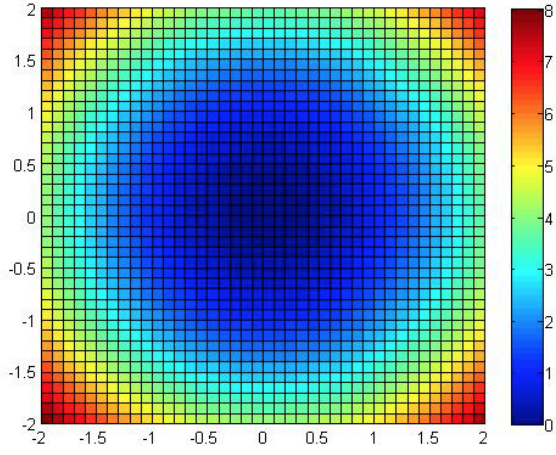
Figura 48: Curva de nivel de la función $z = x^2 + y^2$ usando el comando contour3.



El comando pcolor(x,y,z) dibuja un mapa de colores por niveles. El comando colorbar hace aparecer una escala de valores según el color como se muestra en la Figura 49.

```
1 [x,y]=meshgrid(-2:0.1:2);
2 z = x.^2 + y.^2;
3 pcolor(x,y,z), colorbar;
```

Figura 49: Mapa de colores por niveles usando el comando colorbar.

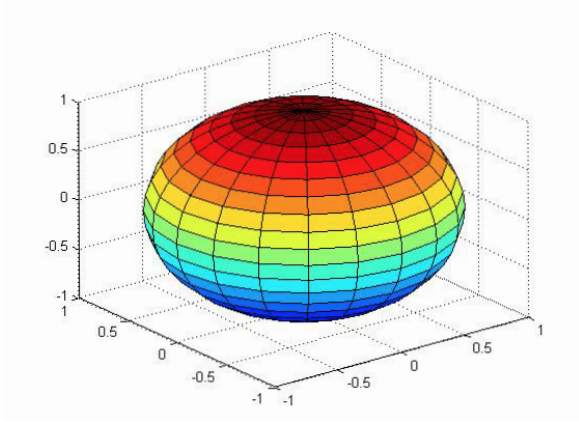


Comando SPHERE (Esfera)

El comando sphere(n) genera una esfera, donde n es el número de puntos en los que queda dividido el ecuador de la esfera. Cuanto mayor sea n, mayor será la aproximación a la curvatura real de la esfera de radio 1, centrada en el origen. El comando sphere() utilizará el valor por defecto n igual a 20, tal como se observa en la Figura 50.

```
1 [x,y,z] = sphere();
2 surf(x,y,z);
```

Figura 50: Esfera generada usando el comando sphere

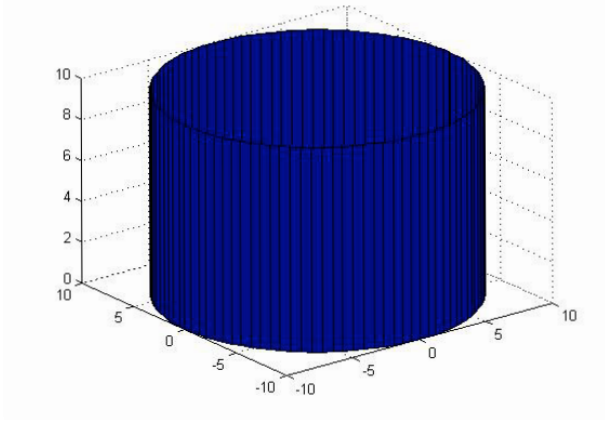


Comando CYLINDER (Cilindro)

El comando cylinder(R,n), genera automáticamente un cilindro de revolución de radio R, donde n es el número de puntos de la circunferencia de la base del cilindro. Como en el caso de la esfera, si usamos sólo cylinder(R), el número n es, por defecto, 20. Un ejemplo de este comando se puede apreciar en la Figura 51.

```
1 [x,y,z] = cylinder(10,100);
2 surf(x,y,10*z);
```

Figura 51: Esfera generada usando el comando cylinder.



Ejercicios sobre Graficación 3D

Los siguientes ejercicios han sido propuestos por el Profesor Roberto Rodríguez del Río en [8].

Curvas en el espacio

Utilice el comando plot3 para graficar cada una de las siguientes curvas.

1. $\vec{r}(t) = (2\cos^3(t), 2\text{sen}^3(t), t),\ -4 \le t \le 3$
2. $\vec{r}(t) = (\cos(t), \cos^3(t), \frac{1}{4}\,\text{sen}(t)),\ -\pi \le t \le \pi$
3. $\vec{r}(t) = (\frac{t}{6}\cos(t), \frac{t}{6}\text{sen}(t)\frac{t}{36}),\ -12 \le t \le 19$
4. $\vec{r}(t) = (e^{\frac{t}{4}}\text{sen}(2t), e^{\frac{t}{4}}\cos(2t), \frac{t}{4}),\ -10 \le t \le 4{,}8$
5. $\vec{r}(t) = (\text{sen}(2t) + \text{sen}(t), -\cos(2t) - \cos(t), \frac{t}{6}),\ -9 \le t \le 10$

Vectores de curvas en el espacio

Por cada curva del ejercicio anterior, utilice el comando quiver3 para generar algunos vectores de velocidad de las curvas. Utilice la resolución adecuada. Recuerde que el comando quiver3 posee la siguiente sintaxis

$$quiver3 = \left(x(t), y(t), z(t), \frac{dx(t)}{dt}, \frac{dy(t)}{dt}, \frac{dz(t)}{dt}\right)$$

Curvas de la forma $z = f(x, y)$

Por cada curva que se presenta a continuación, utilice los comandos plot3, mesh, surf y surf con shading flat para hacer una representación de las cuatro gráficas en una sola ventana a través del comando subplot.

1. $z = \dfrac{1}{9 + x^2 + y^2}$

2. $z = -\sqrt{|xy|}$

3. $z = \dfrac{\cos\left(\dfrac{x^2 + y^2}{4}\right)}{3 + x^2 + y^2}$

4. $z = \dfrac{y^2}{5} - 3|x|$

5. $z = e^{-(x^2+y^2)}$

Contornos de las curvas z = f(x, y)

Utilice los comandos contour, contour3 y pcolor para obtener la reprsentación de curvas de nivel de los ejercicios de la sección anterior. Las tres gráficas resultantes deben aparecer en una sola ventana simultáneamente.

Funciones

En los capítulos anteriores, se ha proporcionado los fundamentos de programación secuencial a través de los comandos de entrada y salida de datos, las operaciones con variables, vectores y matrices; y sus diferentes aplicaciones como la graficación en MATLAB. También se ha revisado la programación estructurada que utiliza las estructuras condicionales y repetitivas.

La programación modular es un paradigma de programación que pretende dividir nuestros programas en componentes que contengan un conjunto de instrucciones que realice la ejecución de algún proceso determinado. El propósito de la programación modular es gestionar eficazmente programas que contendrían cientos o miles de líneas de código en su interior.

En este capítulo se pretende abordar una herramienta importante para la programación modular que son las funciones. Se definirá la sintaxis de una función en MATLAB y la diferencia con los scripts que se venían desarrollando hasta ahora.

Definición

Una función representa un conjunto de instrucciones que se escriben separadamente del programa y que realizan alguna tarea específica. Los usuarios pueden definir sus propias funciones y utilizarlas con las funciones implementadas en MATLAB [6]. A continuación describiremos un tipo de función en línea que permite evaluar expresiones matemáticas analíticas.

Funciones en línea

MATLAB nos proporciona lo que conocemos como inline functions o en español como funciones en Línea. Ésto nos permite crear en la misma ventana de comandos, una corta función que puede ser llamada de forma repetida [9].

Ejemplo: Crear una función en línea utilizando la expresión

$$f(x) = \frac{sen\ (x)}{x}$$

La síntaxis resulta de escribir el nombre la función, seguidamente el signo igual (=) y luego el comando inline(). Dentro de éste, el cuerpo de la función debe ser escrito entre comillas simples (' ') como si estuviésemos definiendo una cadena de caracteres.

```
Command Window
>> f=inline ('sin(x)/x' )

f =

     Inline function :
     f(x) = sin(x)/x
fx
```

De este modo, cuando ingresemos un valor, éste será evaluado en la función y se retornará el valor respectivo al terminar la ejecución de la misma como se observa a continuación.

```
Command Window
>> f(2)

ans =

      0.4546

>> f (15)

ans =

      0.0434
fx
```

La ventaja de estas funciones en línea es que pueden ser graficadas utilizando el comando fplot(), el cual recibe dos argumentos: el nombre

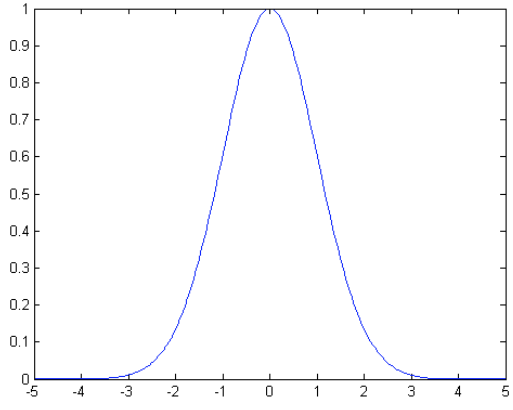
de la función en línea y un vector de dos componentes que describe los valores extremos del dominio de la función.

Ejemplo: Graficar la función $g(x) = e^{-\frac{x^2}{2}}$ en el intervalo [−5, 5] utilizando el comando fplot().

Para este ejemplo, se declara la función en línea g y luego se la grafica en el intervalo [−5, 5] utilizado el comando fplot() como se observa en la Figura 52.

```
Command Window
>> g=inline('exp(-(x^2)/2)')

g =

     Inl ine function :
     g (x) = exp (-(x^2)/2)

>> fplot(g,[-5 5] )
fx
```

Figura 52: Gráfica de la función $g(x) = e^{-\frac{x^2}{2}}$ generada usando el comando fplot()



Existen unas reglas básicas para el uso efectivo de este tipo de funciones en Matlab y son las siguientes:

La expresión matemática puede contener una o más variables independientes, $f(x_1, x_2, x_3, ...,x_n)$.

Se puede usar cualquier letra como variable independiente excepto i y j (porque son las constantes complejas de MATLAB).

La expresión matemática puede contener cualquier función de MATLAB o las que sean definidas por el usuario.

La expresión se debe escribir según la dimensión de los argumentos de entrada. Por ejemplo, operaciones elemento a elemento cuando se trabaja con vectores o matrices.

La expresión no puede incluir variables predefinidas en el Workspace debido a que no tiene acceso a ellas.

Una vez que se define la función, ésta puede ser usada tecleando su nombre y pasando sus argumentos de entrada como en el ejemplo anterior.

Una funcion inline() también se puede usar como argumento de entrada en otras funciones

Funciones definidas mediante script

Este tipo de funciones se crean del mismo modo que un fichero o script, a través del menú File/New/Function. El archivo de la función se guarda en un fichero que tiene el mismo nombre de la función con extensión .m.

La forma para trasmitirle datos a las funciones es mediante el uso de una lista de variables que se denominan parámetros o argumentos de la función. No obstante, es necesario señalar que las variables que se usan dentro de una función, no estarán disponibles fuera de ella.

Declaracion de una función

La sintaxis utilizada para la declaración de una función se presenta a continuación:

```
1 function variable = nombre (argumentos)
2         instrucciones
3
4 % variable              contiene el valor que entrega la funcion
5 % argumentos            son variables que reciben los datos que
6 %                       ingresan a la funcion.
7 % nombre                identificacion de la funcion
8 % instrucciones         describen las tareas que realiza la funcion
```

El nombre asignado a una función debe coincidir con el nombre usado para identificar al archivo que contiene la función.

Las funciones se escriben en la ventana de edicion de MATLAB y se las almacena en alguna carpeta. En la ventana de comandos, se debe especificar la ubicación de esta carpeta.

El uso de una funcion es similar al uso de las funciones comunes en MATLAB. Pese a que los argumentos pueden tener nombres diferentes, su uso debe ser coherente.

Las principales diferencias entre un script o fichero de MATLAB con una función, se puede apreciar en el Cuadro 6.1.

Cuadro 6.1: Scripts vs Funciones

Scripts o Ficheros	**Funciones**
No reciben argumentos de entrada ni producen resultados de salida.	Reciben argumentos de entrada y producen resultados.
Se trabaja sobre las variables en el Workspace.	Las variables internas son locales a la función.
Automatiza una serie de pasos que se repiten con bastante frecuencia.	Se extiende a diferentes aplicaciones que puedan ser implementadas en MATLAB.

Los scripts trabajan sobre variables en el Workspace de la línea de comandos o crean nuevas variables que son añadidas a dicho Workspace, de modo que todas esa variables puden ser luego manipuladas desde la linea de comandos.

Ejemplo: Crear una función que calcule el área y el volumen del cilindro de radio r y altura h. La función debería desplegar la gráfica del cilindro de radio $r = 2m$ y altura $h = 10m$

Para la elaboración de la función, se ha considerado los siguientes pasos:

1. Se guarda la función con el nombre del archivo: grafica_cilindro
2. Se escribe una descripción de la función y de las variables.
3. Se procedea escribir el codigo del programa.

Se crea la función como se muestra a continuación:

```
1  function [v,a] = grafica_cilindro(r,h)
2  % [v,a] = grafica_cilindro(r,h)
3  % Esta funcion calcula el volumen y el area del cilindro de radio r
4  % y altura h
5  % variables de entrada
6  %        r = radio
7  %        h = altura
8  % variables de salida
9  %        v = volumen
10 %        a = area
11 % Este programa fue creado por Hugo San Martin, Estudiante de
12 % Ingenieria Civil de la Universidad Tecnica de Machala
13
14 a = 2 *pi*r*h +2*pi*r^2;       % area
15 v = pi*r^2*h;                  % volumen
16
17 [x,y,z] = cylinder(r,100);     % genera coordenadas (x,y,z) del cilindro
18 surf(x,y,z*h)                  % grafica cilindro de altura h
```

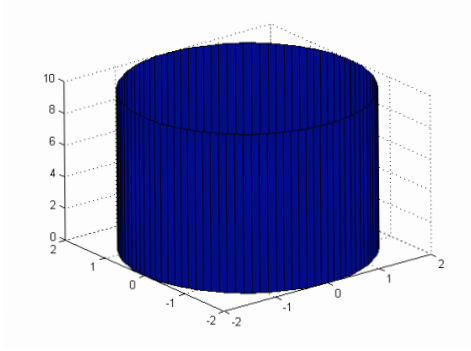
Una vez guardada la función, se puede desplegar la descripción de la función haciendo uso del comando help seguido del nombre de la función.

Command Window

```
>> help grafica_cilindro
[v, a] = grafica_cilindro (r, h)
Esta funcion calcula el volumen y el area del cilindro de radio r
y altura h
variables de entrada
       r = radio
       h = altura
variables de salida
       v = volumen
       a = area
Este programa fue creado por Hugo San Martin , Estudiante de
Ingenieria Civil de la Universidad Tecnica de Machala
fx
```

Para llamar o invocar a la función, se debe escribir el nombre de la función en la ventana de comandos, inicializando los valores de los argumentos r y h con 2 m y 10 m, respectivamente. La gráfica se puede apreciar en la Figura 53.

Figura 53: Gráfica de un cilindro de radio r = 2m y altura h = 10m utilizando la función



Ejemplo: Calcular las raices de un polinomio de grado 2 cuyo ecuación es de la forma $p(x) = ax^2 + bx + c$

Para la elaboración de la función grafica_cilindro, se ha considerado como argumentos los coeficientes del polinomio, y se ha utilizado la fórmula cuadrática $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$ con el fin de calcular las raíces del polinomio p(x).

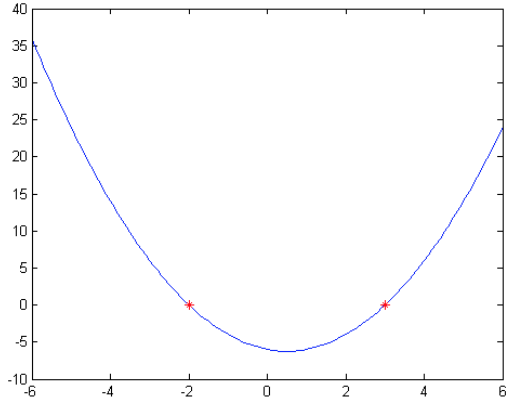
```
1  function [x1,x2] = calcularaices(a,b,c)
2  % [x1,x2] = calcularaices(a,b,c)
3  % Esta funcion calcula la raices de un polinomio de grado 2
4  % de la forma ax^2 + bx + c
5  % variables de entrada
6  %       a = coeficiente de x^2
7  %       b = coeficiente de x
8  %       c = coeficiente independiente
9  % variables de salida
10 %       x1 = raiz 1
11 %       x2 = raiz 2
12 % Este programa fue creado por Edwin Guaman, Estudiante de
13 % Ingenieria Civil de la Universidad Tecnica de Machala
14
15 x1 = (-b + sqrt(b^2 - 4*a*c))/(2*a);    % raiz 1
16 x2 = (-b - sqrt(b^2 - 4*a*c))/(2*a);    % raiz 2
17
18 x = -6:0.1:6;               % dominio
19 y = a*x.^2 + b.*x +c;       % rango
20 y1 = a*x1^2 + b*x1 +c;      % coordenada-y de raiz 1
21 y2 = a*x2^2 + b*x2 +c;      % coordenada-y de raiz 2
22
23 figure(1);
24 plot(x,y), hold on;         % grafica la curva
25 plot(x1,y1,'*r'), hold on;  % grafica raiz 1
26 plot(x2,y2,'*r'), hold on;  % grafica raiz 2
```

Se utiliza la función calcularaices para encontrar las raices del polinomio p(x) = x^2 − x − 6 que se pueden visualizar en la Figura 54.

Command Window

```
>> [ raiz1 , raiz2 ] = calcularaices (1 , -1 , -6)

raiz1 =

     3


raiz2 =

    -2

fx
```

Figura 54: Gráfica del polinomio $p(x) = x^2 - x - 6$ con las raices $x_1 = 3$ y $x_2 = -2$, representadas mediante asteriscos.



Ejercicios sobre Funciones

Los siguientes ejercicios han sido propuestos en su mayoría por el Profesor Enric Cervera en [10].

1. Evaluar las siguientes expresiones para el valor especificado en x utilizando funciones en línea:

a) $y = -\frac{1}{x^3} - \frac{5}{x^2 - x - 6}$, x = 2

b) $y = \dfrac{tan(x) - \text{sen}(x)}{x - sen(x)}$, $x = \dfrac{\pi}{6}$

b) $y = \dfrac{1}{2(\sqrt{1-\sqrt{x}})} - \dfrac{1}{3(\sqrt{1-\sqrt[3]{x}})}$, $x = 3$

2. Escribir una función que convierta grados F (Fahrenheit) a grados C (Celsius) de acuerdo con la siguiente fórmula $C = \frac{5}{9}(F - 32)$

3. Escribir una función que calcule el perímetro y el área de un círculo dado su radio.

4. Crear una función que convierta una medida de longitud expresada en mm, cm, dm, m, km; en una medida convertida a m (metros). Por ejemplo, medida=50 y unidad='cm' aparece 0,5 m.

5. Escribir una función que convierta las coordenadas rectangulares x e y a polares r, θ.

6. Realice una función que implemente la siguiente aproximación de la función seno.

$$sen\ (x) = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \frac{1}{9!}x^9 + \ldots + \frac{(-1)^n - 1}{(2n-1)!}x^{2n-1}$$

El argumento de entrada de la función debe ser n que representa el número de términos a sumar.

7. Elabore una función que implemente y grafique en una sola ventana los primeros n polinomios de Legendre, solicitados al usuario y que se presentan a continuación:

$P0(x) = 1$
$P1(x) = x$
$P2(x) = \frac{1}{2}(3x^2 - 1)$
$P3(x) = \frac{1}{2}(5x^3 - 3x)$
$P4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$
$P5(x) = \frac{1}{8}(63x^5 - 70x^3 - 15x)$
$P6(x) = \frac{1}{16}(231x^6 - 315x^4 + 105x^2 - 5)$

...

$(n + 1)P_{n+1} = (2n + 1)xP_n - nP_{n-1}$

8. Sean dos números enteros positivos m y n, escriba el código de una función que calcule el número combinatorio $\binom{a}{b}$

9. Implemente una función que calcule el adjunto Adj(a_{kj}) en una matriz A ϵ R^{mxn} dados *A, k y j*

10. Realice una función que implemente una la Suma de Riemman como estimación del área bajo de un función en un intervalo dado. Las entradas de esta función deben ser la expresión analítica de la función, el intervalo [a, b] y el número de divisiones n utilizadas para la estimación.

Bibliografía

[1] B. Vangie, “Programming language,” 2015.

[2] Wikibooks, “Fundamentals of computer systems: Generations of programming language,”2015.

[3] Techopedia, “Fifth generation (programming) language (5gl),” 2015.

[4] H. Deitel, Paul & Deitel, C++ How to Program. Prentice Hall, 2012.

[5] A. Cubillos, “Guía de matlab para ing. mecánica,” 2007.

[6] L. Rodríguez, “Matlab programación,” 2014.

[7] U. d. S. Departamento de Ecuaciones Diferenciales y Análisis Numérico, “Algoritmos y estructuras de programación,” 2009.

[8] E. Rodríguez del Río, Roberto & Zuazua, De la aritmética al análisis : historia y desarrollos recientes en matemáticas. Secretaría General Técnica, Subdirección General de Información y Publicaciones, 2002.

[9] J. Echeverri, “Funciones en línea de matlab,” 2014.

[10] E. Cervera, “Práctica 3: Funciones,” 2010.

Índices de imágenes y fotografías

Índice de imágenes y fotografías

Número	Título de la foto/Autor/Año/Fuente	Página
1	Representación de los pasos para la creacion de un programa	19
2	Representación de la variable a	19
3	Estructura de un Algoritmo	24
4	Algoritmo para la preparación de una taza de café.	24
5	Algoritmo para la suma de dos números	25
6	Diagrama de Flujo	26
7	Enlaces de un diagrama de flujo	27
8	Componentes de un diagrama de flujo	28
9	Ingreso de datos en MATLAB	28
10	Salida de datos en MATLAB	29
11	Diagrama de flujo	29
12	Estructura Condicional Simple	34
13	Diagrama de flujo de condicional simple IF/END	34
14	Estructura Condicional Doble	35
15	Diagrama de flujo de condicional doble IF/ELSE/END	36
16	Estructura Condicional Múltiple	37
17	Diagrama de flujo de condicional múltiple IF/ELSEIF/ELSE/END	37
18	Estructura Condicional SWITCH/CASE	38
19	Diagrama de flujo de condicional múltiple SWITCH/CASE.	39
20	Ejercicio 1 con Diagrama de flujo de Condicionales	41
21	Ejercicio 2 con Diagrama de flujo de Condicionales	41
22	Estructura Repetitiva FOR	44
23	Diagrama de flujo de estructura repetitiva FOR	44
24	Estructura Repetitiva WHILE	45
25	Diagrama de flujo de repetitiva WHILE	45
26	Interrupción con BREAK	46
27	Diagrama de flujo de repetitiva BREAK	47
28	Interrupción con CONTINUE	48

29	Diagrama de flujo de repetitiva CONTINUE	48
30	Ejercicio 1 con Diagrama de flujo de Bucles	49
31	Ejercicio 2 con Diagrama de flujo de Bucles	50
32	Función y = sen(x)	74
33	Función y = sen(x) con cuadrícula	74
34	Función y = sen(x) trazada con asteriscos rojos	75
35	Función y = sen(x) con etiquetado en el eje x, en el eje y, y el título	76
36	Trazado de funciones por partes	77
37	Trazado de curvas paramétricas	78
38	Trazado de curvas polares	79
39	Trazado de curvas tridimensionales con plot3(x,y,z)	81
40	Trazado de curvas tridimensionales con quiver3(x,y,z)	82
41	Trazado de curvas tridimensionales con plot3(x,y,z)	83
42	Trazado de curvas tridimensionales con mesh(x,y,z)	84
43	Trazado de curvas tridimensionales con surf(x,y,z)	84
44	Trazado de curvas tridimensionales con surf(x,y,z) y shading flat	85
45	Representación de las cuatro figuras en una sola ventana mediante el comando subplot.	86
46	Relación de los valores del relieve de los puntos con el color mediante el uso del comando colorbar	87
47	Curva de nivel de la función $z = x^2 + y^2$ usando el comando contour	88
48	Curva de nivel de la función $z = x^2 + y^2$ usando el comando contour3	88
49	Mapa de colores por niveles usando el comando colorbar	89
50	Esfera generada usando el comando sphere	90
51	Esfera generada usando el comando cylinder	90
52	Gráfica de la función $g(x) = e^{-\frac{x^2}{x}}$ generada usando el comando fplot	95
53	Gráfica de un cilindro de radio r = 2m y altura h = 10m utilizando la función	99
54	Gráfica del polinomio $p(x) = x^2 - x - 6$ con las raices $x_1 = 3$ y $x_2 = -2$, representadas mediante asteriscos.	100

Biografía

Eduardo Alejandro Tusa Jumbo

Ingeniero Electrónico. Máster de Ciencias en Visión, Imagen y Robótica; otorgado por el consorcio de Universidades de Heriot-Watt en Reino Unido, la Universidad de Borgoña en Francia y la Universidad de Girona en España. Como Docente de la Unidad Académica de Ingeniería Civil, dirige el proyecto de investigación "Desarrollo de Herramientas de Análisis de Imágenes Espectrales y Modelización Matemática para el Estudio de áreas Geográficas", aprobado mediante resolución Nº. 297/2014 del H. Consejo Universitario de la UNIVERSIDAD TÉCNICA DE MACHALA

Hugo Dennys San Martin Cuenca

Bachiller de Físico Matemático Químico Biológico del Colegio Nacional Mixto Machala. Tiene estudios en Informática Avanzada y Música Clásica.

Actualmente se desempeña como Estudiante de la Unidad Académica de Ingeniería Civil de la UNIVERSIDAD TÉCNICA DE MACHALA.

Fundamentos de programación para ciencias e ingeniería
Se terminó de imprimir en marzo de 2016 en la imprenta de la UTMACH, calle Loja y 25 de Junio (campus Machala)
Esta edición consta de 300 ejemplares.

www.utmachala.edu.ec

El programa de Reingeniería del Conocimiento en la Universidad Técnica de Machala (UTMACH) es un modelo emergente de gestión de la investigación que promueve saberes científicos con pertinencia social. Desde el Vicerrectorado Académico impulsamos la investigación colectivista, donde docentes y estudiantes se engranan en la construcción y divulgación del resultado de sus ejercicios pedagógicos, heurísticos y de vinculación social, en aras de contribuir con el fortalecimiento de nuestras ventajas comparativas y competitivas a nivel transfronterizo.

Mediante este programa estratégico la UTMACH impacta sus imaginarios respecto a la relación de la docencia con la investigación, muestra de ello es la presente obra donde se cristaliza el empoderamiento y profesionalismo de sus actores y redes al servicio de la formación crítica de profesionales de avanzada.

En la UTMACH seguimos conquistando el conocimiento a través de la investigación, por ello en cada acción emprendida ***proyectamos nuestra historia***.

Ing. Amarilis Borja Herrera, Mg. Sc.
VICERRECTORA ACADÉMICA.

ISBN: 978-9978-316-80-1

REINGENIERÍA DE LA INVESTIGACIÓN

UTMACH