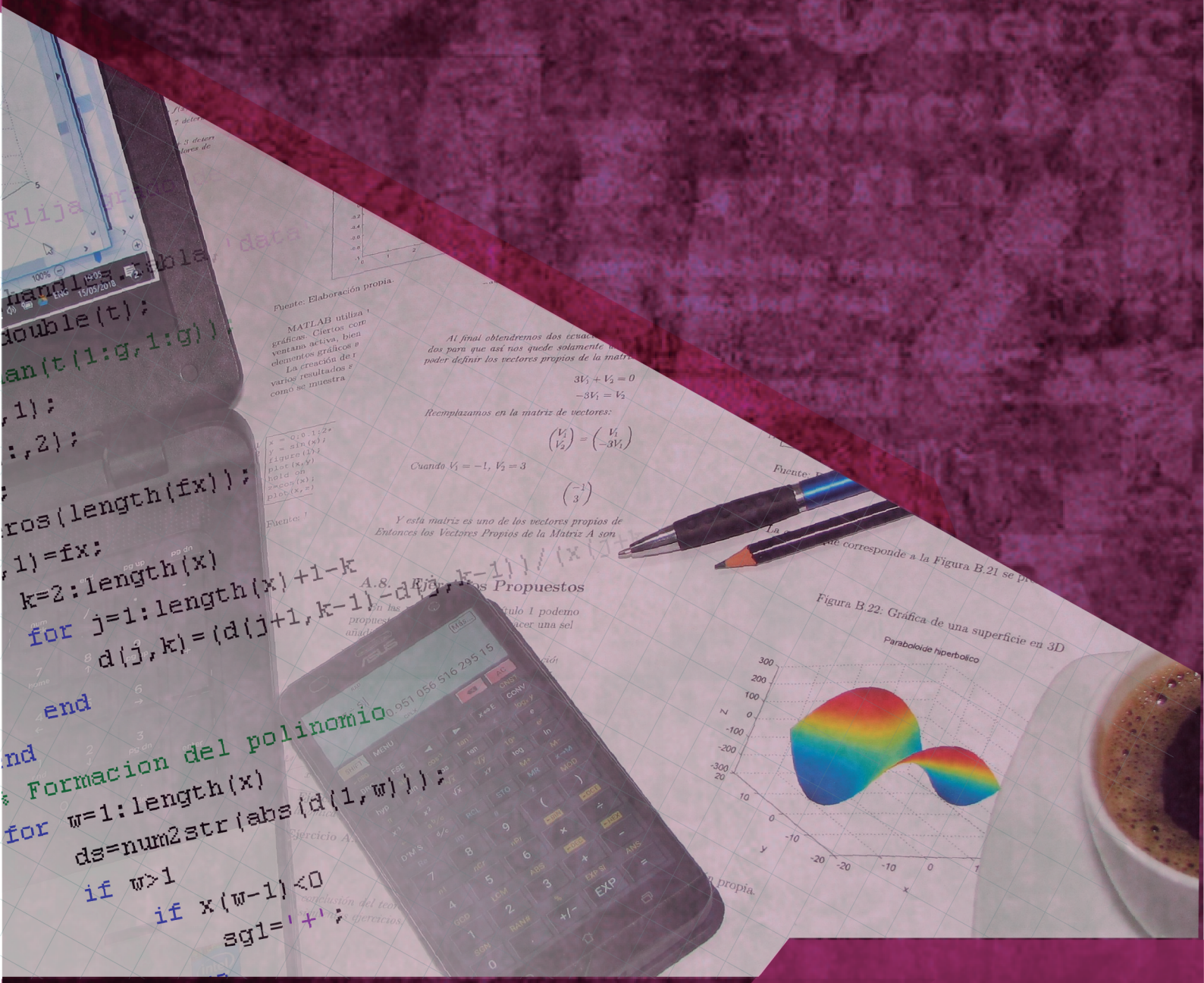
MÉTODOS NUMÉRICOS PARA EL ANÁLISIS MATEMÁTICO CON MATLAB

JUAN F. MAÑAS-MAÑAS / MARITZA A. PINTA



Editorial UTMACH

REDES 2017

COLECCIÓN EDITORIAL

Primera edición en español, 2018

Este texto ha sido sometido a un proceso de evaluación por pares externos con base en la normativa editorial de la UTMACH

Ediciones UTMACH

Gestión de proyectos editoriales universitarios

306 pag; 22X19cm - (Colección REDES 2017)

Título: Métodos Numéricos para el Análisis Matemático con Matlab.

Juan F. Mañas Mañas / Maritza A. Pinta (Coordinadores)

ISBN: 978-9942-24-104 -7

Publicación digital

Título del libro: Métodos Numéricos para el Análisis Matemático con Matlab.

ISBN: 978-9942-24-104 -7

Comentarios y sugerencias: editorial@utmachala.edu.ec

Diseño de portada: MZ Diseño Editorial

Diagramación: MZ Diseño Editorial

Diseño y comunicación digital: Jorge Maza Córdova, Ms.

Km. 5 1/2 Vía Machala Pasaje

www.utmachala.edu.ec

Machala - Ecuador

Métodos Numéricos para el Análisis Matemático con MATLAB

Juan F. Mañas-Mañas
Maritza A. Pinta

Coordinadores

Resumen

El objetivo principal de esta obra es introducir al lector en el estudio de los **Métodos Numéricos** utilizando la plataforma computacional MATLAB, e impulsar en el estudiante el desarrollo de habilidades para el Análisis Matemático. Este libro es un trabajo conjunto de docentes de la Universidad Técnica de Machala (Ecuador) y de la Universidad de Almería (España) empeñados en difundir los Métodos Numéricos.

Los Métodos Numéricos son de gran importancia porque constituyen hoy en día en una herramienta fundamental para la solución de muchos problemas de las ciencias cuya solución exacta no es alcanzable y, es por tanto, necesario obtener una solución aproximada. Por otro lado, el desarrollo computacional actual, a través de software como MATLAB, permite resolver rápidamente mediante la implementación de algoritmos numéricos eficientes problemas que antes tenían una solución numérica poco factible, debido al tiempo que se requería para ello.

Este libro se compone de 6 capítulos. En el primero de ellos, se realizará un repaso de las bases matemáticas necesarias para la comprensión de los temas a tratar en los capítulos subsiguientes; se abordarán conceptos referentes a: funciones, derivación, integración, ecuaciones diferenciales, matrices y vectores.

MATLAB se fundamenta en cuatro paradigmas básicos de la programación: la programación secuencial, la programación estructurada, la programación modular y la programación orientada a objetos. Por ello se ha considerado necesario en el segundo capítulo hacer una presentación de la sintaxis de comandos secuenciales que sean de relevancia significativa para la construcción de algoritmos de métodos numéricos y la visualización de resultados. Para ello, se empieza con la caracterización de las variables en el entorno de MATLAB; luego, se presentan los comandos que implementan diferentes funciones matemáticas. De esta forma, se aprovecha los recursos disponibles en este paquete computacional para fortalecer las soluciones numéricas de nuestros algoritmos.

El capítulo 3 nos introduce en el estudio de la derivación numérica de funciones, la cual tiene muchas aplicaciones, especialmente en la resolución numérica de ecuaciones diferenciales. Además, nos permite determinar la derivada de un orden determinado de una función en un punto dado, utilizando solamente los valores que toma la función en una serie de puntos. Se realizará un estudio con detalle pero con enfoque práctico de estas fórmulas y del error teórico cometido, prestando atención a un método relevante de aceleración de la convergencia conocido como método de extrapolación de Richardson; haciendo la implementación de estos métodos en MATLAB a través de ejemplos prácticos.

Posteriormente los capítulos 4 y 5 están dedicados a la integración numérica

que tienen como objetivo aproximar numéricamente integrales definidas, las cuales tienen muchas aplicaciones tanto en matemáticas como en procesos científicos-técnicos. Este cálculo suele ser complicado y en la mayoría de los casos es inviable si se pretende expresar el valor de la integral definida como la evaluación de combinación de funciones elementales.

El capítulo 4 tiene por objetivo obtener expresiones, usualmente denominadas fórmulas de cuadratura, que permitan aproximar de la forma más exacta posible una integral definida. Obtendremos fórmulas basadas en polinomios interpoladores, denominadas fórmulas de Newton-Cotes. Como es habitual en el análisis numérico proporcionaremos expresiones para el error cometido al usar estas fórmulas. Adicionalmente aplicaremos el proceso de aceleración o método de Romberg y algunas breves notas sobre cuadraturas adaptativas.

En el capítulo 5, dedicado a la integración numérica, se presentarán las fórmulas de cuadratura gaussianas. La ventaja de estas fórmulas es que los nodos involucrados no son fijos, sino que van a ser los ceros de determinados polinomios ortogonales; de esta forma natural, imbricamos la integración numérica con la Teoría de Aproximación a través del uso de los polinomios ortogonales. Haremos un estudio detallado y práctico de las fórmulas gaussianas, analizando su exactitud máxima, el error y el cálculo eficiente de sus nodos y pesos mediante los valores y vectores propios de la matriz de Jacobi.

Nuestro último capítulo está dedicado a las ecuaciones diferenciales, que es la herramienta matemática más útil a la hora de describir problemas en todos los ámbitos de las ciencias y también en otras ramas del conocimiento. Es bien conocido su uso en la modelización matemática en biología, ingeniería, medicina, informática y cualquier área científico-técnica, pero también en otras áreas como en el estudio de comportamientos sociales o en economía.

En este libro introductorio a los métodos numéricos pretendemos acercar al lector a la resolución numérica de ecuaciones diferenciales ordinarias (EDO). Las ecuaciones, o de forma más natural, los sistemas de EDO que aparecen en la modelación matemática, raramente son resolubles utilizando solamente el análisis matemático y es imprescindible el uso del análisis numérico. Se asumirá que el lector posee un cierto conocimiento de EDO, y se presentarán de forma práctica métodos útiles de resolución numérica de problemas de valores iniciales y problemas de contorno. Se prestará atención a los denominados problemas stiff. La aplicación de los métodos presentados en este capítulo hará uso necesariamente del ordenador y del programa MATLAB.

Listado de autores por capítulos

- Capítulo 1:
  - Maritza A. Pinta.
  - Fausto F. Redrován.
- Capítulo 2:
  - Eduardo Tusa.
  - Carlos Loor.
- Capítulo 3:
  - Juan F. Mañas-Mañas.
  - Maritza A. Pinta.
- Capítulo 4:
  - Juan F. Mañas-Mañas.
  - Maritza A. Pinta.
- Capítulo 5:
  - Juan F. Mañas-Mañas.
  - Maritza A. Pinta.
- Capítulo 6:
  - Juan J. Moreno-Balcázar.

Dedicatorias

Dedico esta obra a mi amado esposo Eguipto y a mi hijo Juan José, por su amor y apoyo incondicional.
Maritza A. Pinta

Esta obra es dedicada amis estudiantes, quiénes han tenido la libertad de compartirme sus inquietudes, inconformidades y preocupaciones; en torno al área de la programación y métodos numéricos en MATLAB. Dedico este trabajo a toda mi familia, mis padres, mis hermanas, mis sobrinos; quiénes no han dudado en demostrarme su enorme afecto, cariño y paciencia en cada acierto y desacierto que se concibe a lo largo de mi vida. A aquellos grandes amigos y amigas que a pesar del tiempo, no han renunciado a caminar junto a mi lado. Finalmente, a todos los grandes soñadores del mundo que se levantan cada mañana dispuestos a cambiar el curso de su destino.
Eduardo Tusa

A toda mi familia. En especial, a mi mujer y compañera, Loli, y a mis hijas, Rocío y Mar, por ser tan maravillosas y a mis padres, Juan José y Rosa, gracias a ellos empezó todo, siempre serán mis referentes.
Juan J. Moreno-Balcázar

Dedico esta obra a mis padres, Francisco y Rosalía, ya que gracias a ellos he llegado hasta donde estoy hoy en día. A mi hermana Rosa María y mi novia Mónica por el apoyo recibido, y a toda mi familia y amigos por los ánimos dados.
Juan F. Mañas-Mañas

A mi esposa Rosita, mi consejera e inspiración.
A mis hijos Abigaíl y Mateo, mi motivación y razón.
A mis padres Fausto y Mirian, mi guía y apoyo.
A la memoria de mi abuelito Fausto,
mi primer profesor de matemáticas.

Fausto F. Redrován

Agradecimientos

Los autores compiladores, Maritza A. Pinta y Juan F. Mañas-Mañas agradecen en primer lugar a todos los autores su trabajo, perseverancia y todo el esfuerzo dedicado a la hora de realizar este libro, ya que gracias a ellos el mismo ha sido posible.

Todos los autores aprovechamos estas líneas para agradecer a las principales autoridades de la Universidad Técnica de Machala, quienes han brindado el tiempo, los recursos y el escenario para impulsar la producción científica de sus principales actores. Asimismo, expresamos nuestra gratitud a la Editorial UTMACH y al Centro de Investigaciones por fortalecer iniciativas orientadas a preservar el legado científico de los investigadores.

Los autores de la Universidad Técnica de Machala expresamos también gratitud al decano, subdecano, coordinadores y colegas docentes de la Unidad Académica de Ingeniería de Civil, por su cooperación en potenciar los espacios de investigación alcanzados y a los miembros del Grupo de Investigación de Automatización, Matemática y Tecnologías de la Información y las Comunicaciones, AutoMathTIC.

Por parte de la Universidad de Almería, el autor Juan J. Moreno Balcázar está parcialmente financiado por los siguientes proyectos e instituciones: Grupo de Investigación FQM-229 de la Junta de Andalucía adscrito al Campus de Excelencia Internacional del Mar (CEIMAR); Ministerio de Economía y Competitividad de España y el Fondo Europeo de Desarrollo Regional (FEDER) con el proyecto MTM2014-53963-P; Centro para el Desarrollo y la Transferencia de la Innovación Matemática en la Empresa (CDTIME). El autor Juan F. Mañas-Mañas está financiado por una beca predoctoral del Plan Propio de la Universidad de Almería, además, parcialmente financiado por los siguientes proyectos e instituciones: Grupo de Investigación FQM-229 de la Junta de Andalucía adscrito al Campus de Excelencia Internacional del Mar (CEIMAR); Ministerio de Economía y Competitividad de España y el Fondo Europeo de Desarrollo Regional (FEDER) con el proyecto MTM2014-53963-P.

Finalmente, y no por ello menos importante, agradecemos también a los árbitros anónimos la revisión de los capítulos, gracias a sus comentarios y aportaciones esta obra ha ganado valor y relevancia.

Índice general

Índice de tablas

Índice de figuras

Capítulo 2

INTRODUCCIÓN A MATLAB

Autores: Eduardo Tusa[1], Carlos Loor[2]

[1,2]Unidad Académica de Ingeniería Civil, Universidad Técnica de Machala, Ecuador.

[1]etusa@utmachala.edu.ec, [2]cloor@utmachala.edu.ec

2.1. Introducción

La evolución del hardware computacional tuvo una importante incidencia en el desarrollo de los lenguajes de programación. La invención de los microprocesadores ofreció las condiciones para que los lenguajes de programación de cuarta generación brindaran soporte para la gestión de bases de datos, la generación de reportes, el desarrollo de interfaces de usuario y la optimización matemática [8]. La plataforma computacional de MATLAB se constituye en uno de los principales representantes de esta generación debido a las facilidades que ofrece su entorno computacional numérico.

MATLAB se fundamenta en cuatro paradigmas básicos de la programación: la programación secuencial, la programación estructurada, la programación modular y la programación orientada a objetos. Este capítulo centra sus esfuerzos en la presentación de la sintaxis de comandos secuenciales que sean de relevancia significativa para la construcción de algoritmos de métodos numéricos y la visualización de resultados.

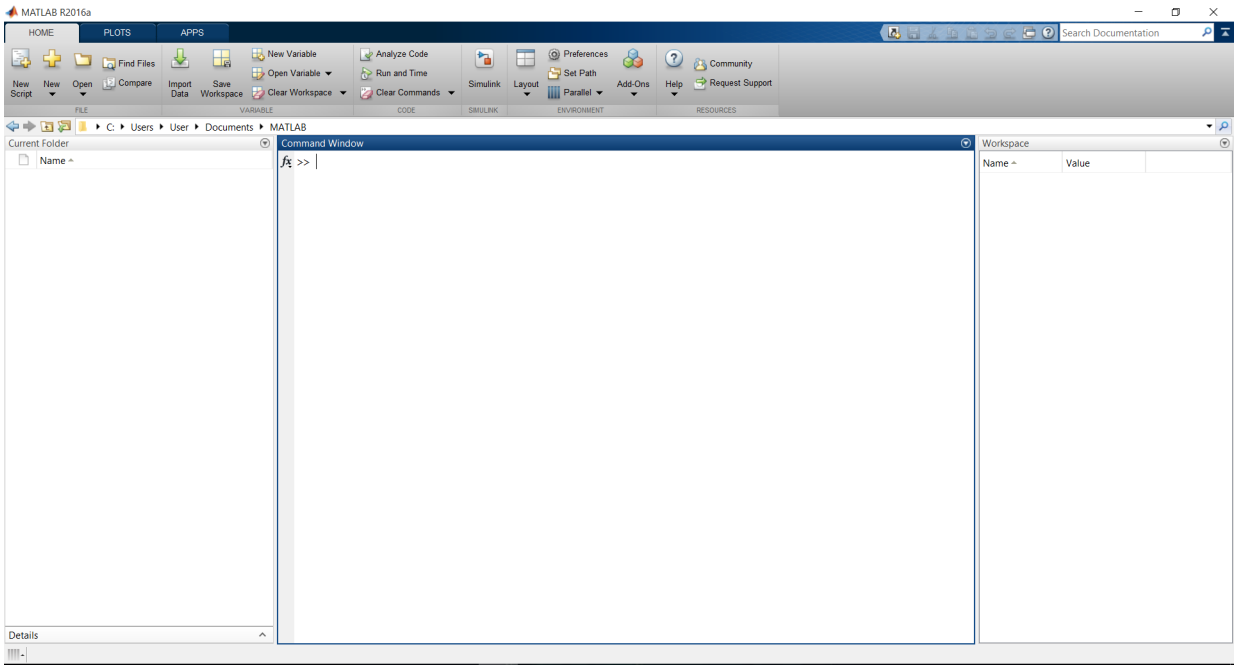
Para ello, se empieza con la caracterización del entorno de MATLAB, sabiendo que la fortaleza de MATLAB reside en la manipulación de matrices, a continuación se definen matrices y sus operaciones. La manipulación de arreglos de datos posee una considerable aplicación al momento de realizar gráficas en 2 y 3 dimensiones mediante los comandos de MATLAB. Además, se hace una descripción de las estructuras de control para acciones de decisión y repetición. Finalmente, se presenta la sintaxis para la elaboración de funciones con el fin de aprovechar los recursos disponibles en este paquete computacional para fortalecer las soluciones numéricas de los diferentes algoritmos.

2.2. Entorno de MATLAB

Una vez que se inicia MATLAB, se presenta una interface dividida en tres ventanas [1] como se observa en la Figura 2.1, las cuales se detallan a continuación:

- Current Folder (Ventana Izquierda)
- Command Window (Ventana Central)
- Workspace (Ventana Derecha)

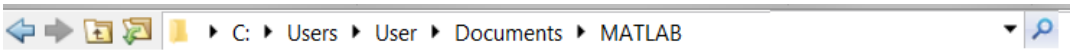
Figura 2.1: Entorno de la interface de MATLAB o Ventana Principal



Fuente: Elaboración propia.

Current Folder: Esta ventana es un explorador de los archivos contenidos en la ubicación descrita en la barra de herramientas del Current Folder [4]. En la Figura 2.2, se aprecia la barra de herramientas del Current Folder y la ubicación o raíz de la carpeta que se visualiza en esta ventana.

Figura 2.2: Barra de herramientas del Current Folder



Fuente: Elaboración propia.

Si se desea cambiar a una carpeta de trabajo, se debe modificar la raíz o el *search path* en la barra de herramientas del Current Folder. Para la ejecución de un programa en MATLAB, el archivo debe tener la extensión *.m* y visualizarse en la ventana del Current Folder, o cualquier otra carpeta que contenga un archivo con la extensión *.m* creado por el usuario.

Command Window: La Figura 2.3 es la Ventana Command Window donde se solicita ejecutar la suma de dos números enteros y se muestra el correspondiente resultado de la ejecución. Por defecto, la variable *ans* representa la respuesta (answer, en inglés) y almacena el resultado de la operación.

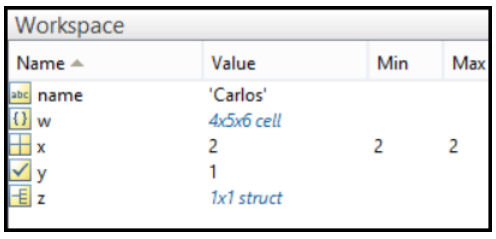
Figura 2.3: Interface de la ventana Command Window en MATLAB

Fuente: Elaboración propia.

El símbolo >> indica que esta ventana está disponible para la escritura de comandos. En la esquina inferior izquierda de la ventana principal de MATLAB (ver Figura 2.1), se despliega un mensaje con las palabras *Ready* o *Busy*, cada vez que el programa está esperando por datos a ser ingresados por el usuario, o el programa esté realizando cálculos.

Workspace: Presenta las variables que se definen en la ventana Command Window (ver Figura 2.4), como resultado de la ejecución de los programas de MATLAB. El Workspace muestra la información de cada variable, incluyendo sus dimensiones, valores mínimos o valores máximos [10]. En la parte superior de la ventana, se presentan botones que permiten realizar varias tareas básicas como crear, guardar, eliminar, entre otras. Todas las variables que se han definido en una sesión de trabajo, se pueden guardar utilizando ***File*** > ***Save Workspace As*** (Ctrl-S). La extensión para un archivo del Workspace es .mat [11].

Figura 2.4: Ventana del workspace con 5 tipos de variables inicializadas



Fuente: Elaboración propia.

2.3. Variables

El análisis de los modelos matemáticos requiere la caracterización de los elementos de estudio. A menudo, estas características tienen una naturaleza variable debido a los diferentes valores que pueden asumir. Así, las variables se consideran como símbolos que representan valores. Las variables están asociadas a porciones de memoria que almacenan estos valores. El nombre de una variable representa todos los valores posibles que puede contener. Por esta razón, existen ciertas nomenclaturas para la definición del nombre de variables [12]:

- MATLAB es sensible a las letras mayúsculas con respecto a las variables. Por lo tanto las variables $x = 2$ y $X = 3$ se consideran dos variables diferentes.
- Todos los nombres de variables deben empezar con una letra.
- MATLAB utiliza sólo los primeros 31 caracteres para el nombre de variables.
- Los signos de puntuación no están permitidos en el nombre de las variables.
- El signo de subguión "_" está permitido en el nombre de variables.
- Tanto letras como dígitos pueden conformar el nombre de una variable.
- No puede contener espacios en blanco. Por ejemplo: *constante gravitacional* no es un nombre de variable permitido.

Todas las variables deben estar definidas o inicializadas con un valor, de lo contrario se produce errores de indefinición.

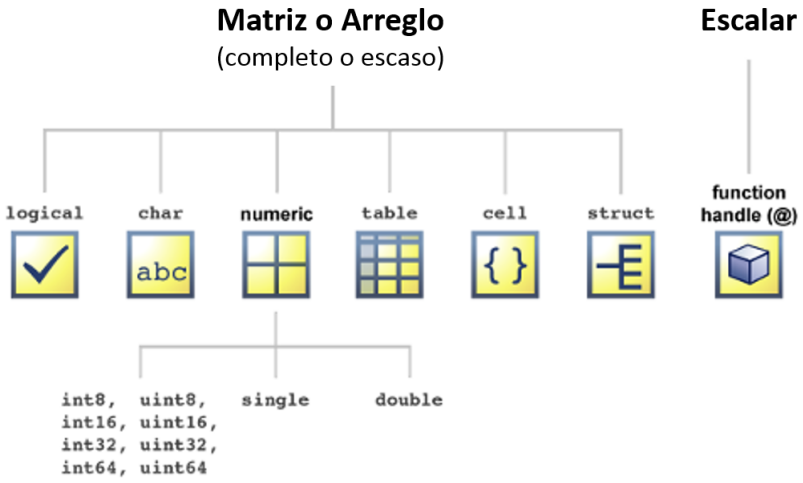
2.3.1. Tipo de variables

Existen algunos tipos de variables en MATLAB que están asociados al tipo de datos que almacenan. La Figura 2.5 obtenida del sitio de [15] presenta los tipos de datos o clases con los que puede trabajar en MATLAB.

Se puede crear matrices y arreglos de datos de punto flotante y enteros (variables numéricas o en inglés, numeric), caracteres y cadenas (variables tipo caracter o en inglés, char), y estados lógicos de verdadero y falso (variables lógicas o en inglés, logical) [20]. Las funciones handle es un tipo de dato en MATLAB que almacena una asociación a una función. El tema de funciones se tratará en la sección 2.9. Los arreglos de tablas (table), de estructuras (struct) y de celdas (cell) proporcionan una alternativa para almacenar diferentes tipos de datos en una mismo entidad.

Hay 16 clases fundamentales en MATLAB. Cada una de estas clases está en la forma de una matriz o arreglo. Con la excepción de las funciones handles, las matrices o arreglos son de un tamaño mínimo de 0 por 0 y puede crecer hasta una matriz n-dimensional de cualquier tamaño. Una función handle es siempre escalar (1 por 1).

Figura 2.5: Tipo de variables disponibles en MATLAB



Fuente: Obtenida de [15].

2.3.2. Jerarquía de operaciones artiméticas

Las operaciones más básicas con MATLAB se realizan con variables numéricas, cuyos operadores aritméticos son $+$, $-$, $/$, $*$ y $\wedge$ potencia. Estos operadores se combinan con paréntesis () para agrupar operaciones. MATLAB brinda una atención especial a la manera en que se escribe una expresión matemática. Por ejemplo, $10+48/2*4 = 10+(48/2)*4 = 106$ no es igual a $10+48/(2*4) = 10+6$. En general, MATLAB ejecuta las operaciones en el siguiente orden:

1. Se da atención a las operaciones dentro de los paréntesis ()
2. La exponenciación, $\wedge$, de izquierda a la derecha.
3. La multiplicación y división, $*$, $/$, de izquierda a derecha.
4. La suma y la resta, $+$, $-$, de izquierda a derecha.

Un ejemplo se presenta a continuación. En el lado izquierdo, se presenta la operación a desarrollar, mientras que en lado derecho; se presenta paso a paso, el resultado de la operación a la cual se ha dado prioridad, de acuerdo al orden jerárquico establecido previamente.

$$
\begin{aligned}
&2+5/2*4-2\wedge 3+(5*2) = 2+5/2*4-2\wedge 3+10\\
&2+5/2*4-2\wedge 3+(5*2) = 2+5/2*4-8+10\\
&2+5/2*4-2\wedge 3+(5*2) = 2+2{,}5*4-8+10\\
&2+5/2*4-2\wedge 3+(5*2) = 2+10-8+10\\
&2+5/2*4-2\wedge 3+(5*2) = 14
\end{aligned}
$$

En el apéndice B1 se presentan cuestiones resueltas en relación a la jerarquía de operacionalización y reconocimiento de variables.

2.4. Vectores

El vector, al igual que la matriz, forman el tipo de datos más común dentro del entorno de MATLAB [3]. El vector está formado por una fila y n columnas (vector fila de $1 \times n$) o m filas y una columna (vector columna $m \times 1$). La dimensión del vector fila se denota como $1 \times n$ y $m \times 1$ para el vector columna.

2.4.1. Ingreso de vectores

La forma de ingresar un vector en la ventana Command Window es escribiendo el nombre del vector (variable) y luego del signo igual, se ingresan los elementos del vector entre corchetes y separados por espacios o comas, si es un vector fila o punto y coma, si es un vector columna. El signo igual (=) asigna valores a las variables, se puede ver en la Figura 2.6 que el vector fila a sido asignado a la variable A y el vector columna a la variable B.

Figura 2.6: Ingreso de vectores fila y columna en MATLAB

```
Command Window
>> x = [2 4 5 7 8 10]    % Vector fila
x =
     2     4     5     7     8    10
>> y = [2;1;5;4;3]       % Vector columna
y =
     2
     1
     5
     4
     3
fx
```

Fuente: Elaboración propia.

MATLAB ofrece comandos y estructuras que definen un vector y que operan con elementos de un vector. Por ejemplo, para crear un vector con elementos igualmente espaciados se usa la siguiente estructura

variable=***inicio:incremento:fin***

Si el incremento es 1 es suficiente la estructura

variable=***inicio:fin***

Se puede usar el comando ***linspace*** siguiendo la siguiente estructura,

variable=***linspace***(***inicio, fin, número de elementos***)

La Figura 2.7 presenta un ejemplo de creación de vectores igualmente espaciados.

Figura 2.7: Creación de un vector fila igualmente espaciado

```
Command Window
>> x1=0:2:10                % vector de 0 a 10 con incremento de 2

x1 =
     0     2     4     6     8    10

>> x2=0:10                  % vector de 0 a 8 con incremento de 1

x2 =
     0     1     2     3     4     5     6     7     8

>> x3=linspace(0,10,6)      % vector de 0 a 10 con incremento de
                            %(10-0)/(6-1)
x3 =
fx   0     2     4     6     8    10
```

Fuente: Elaboración propia.

Se puede crear un vector solo de unos con el comando ***ones***, y de ceros con el comando ***zeros***

variable=***ones*(*número de filas*, *número de columnas*)**

variable=***zeros*(*número de filas*, *número de columnas*)**

Un ejemplo de cada comando se aprecia en la Figura 2.8.

Figura 2.8: Vectores fila de unos y ceros

```
Command Window
>> x2=ones(1,8)          % vector de 1 fila y 8 columnas de unos
x2 =

     1     1     1     1     1     1     1     1

>> x3=zeros(1,10)        % vector de 1 fila y 10 columnas de ceros
x3 =

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

Fuente: Elaboración propia.

En MATLAB el apostrofe simple (') es utilizado para determinar la transpuesta de un vector fila a un vector columna, o un vector columna a un vector fila, por ejemplo en la Figura 2.9 se a transpuesto el vector fila a vector columna. Para no mostrar los resultados en pantalla se usa el ";"

Figura 2.9: Transpuesta de un vector fila a un vector columna

```
Command Window
>> x=[2 4 6 8];          % ingreso vector sin mostrar resultado
>> x1=x'                 % transpuesta de un vector

x1 =
     2
     4
     6
     8
fx
```

Fuente: Elaboración propia.

Después de ingresar o crear un vector, es posible acceder a cada uno de sus componentes y cambiarlos si fuera el caso. Algunos ejemplos se dan en la Figura 2.10.

Figura 2.10: Consulta de los elementos de un vector

```
Command Window
>> x=[2 4 6 8];              % ingreso del vector sin mostrar
                             % resultado por pantalla
>> c=x(2)                    % asigna a la variable c el segundo
                             % elemento del vector x
c =
     4

>> x(3)=7                    % modifica el valor del tercer
x =                          % elemento
     2     4     7     8
>>  y=x(2:4)                 % asigna a la variable y los
y =                          % elementos 2 a 4
     4     7     8
fx
```

Fuente: Elaboración propia.

2.4.2. Operaciones con vectores

Las operaciones entre vectores se realizan elemento a elemento, y en consecuencia, deben tener la misma dimensión y están sujetas a la jerarquía de operaciones aritméticas mencionadas anteriormente [19].

Suma En la Figura 2.11, en primer lugar se introducen los vectores filas x y y sin mostrar sus resultados en pantalla mediante el signo de ";" al final de la instrucción, y luego se realiza la suma.

Figura 2.11: Suma de un vector con otro vector y un escalar

```
Command Window
>> x=[2  4  6];
>> y=[1  3  5];
>> c=2;
>> z=x+y                    %suma de vectores
z =
     3     7    11
>> u=x+c                    %suma de un vector con un escalar
u =
     4     6     8
fx
```

Fuente: Elaboración propia.

Resta Un ejemplo se presentea en la Figura 2.12.

Figura 2.12: Resta de un vector con otro vector y un escalar

```
Command Window
>> xx=x-y                   %resta de vectores
xx =
     1     1     1
>> yy=x-c                   %resta de un vector con un escalar
yy =
     0     2     4
fx
```

Fuente: Elaboración propia.

En MATLAB el punto (.) es necesario siempre que se quieran realizar las operaciones de producto, división y potencia entre vectores.

Producto La Figura 2.13 presenta ejemplos de productos con vectores.

Figura 2.13: Producto de vectores

```
Command Window
>> zz=3*y                   %producto de un escalar por un vector
zz =
     3    15    27
>> uu=x.*y                  %producto de vectores
uu =
     2    12    30
fx
```

Fuente: Elaboración propia.

División La Figura 2.14 presentan ejemplos de división de vectores.

Figura 2.14: División de vectores

```
Command Window
>> xx=x/c                  % division de un vector entre un escalar
xx =
    1.0000    2.0000    3.0000
>> yy=x./y                 % division de vectores
yy =
    2.0000    1.3333    1.2000
fx
```

Fuente: Elaboración propia.

Potencia La Figura 2.15 presenta ejemplos de potenciación de vectores.

Figura 2.15: Potencia de vectores

```
Command Window
>> xx=x.^c                 % vector elevado a un escalar
xx =
     4    16    36
>> yy=x.^y                 % vector elevado a un vector
yy =
     2     64   7776
fx
```

Fuente: Elaboración propia.

2.5. Matrices

La matriz es un conjunto de vectores formado por m filas y n columnas, la dimensión de una matriz se denota con $m \times n$ [13]. Al igual que con vectores, MATLAB ofrece comandos para definir matrices con ciertas características. A continuación, se especifican los comandos para obtener matrices de unos, ceros y la matriz identidad.

2.5.1. Ingreso de matrices

En la Figura 2.16 se da un ejemplo de como ingresar una matriz 3×3.

Figura 2.16: ingreso de una matriz 3×3

```
Command Window
>> matriz1=[1 2 3;4 5 6;7 8 9]
matriz1 =
     1     2     3
     4     5     6
     7     8     9
fx
```

Fuente: Elaboración propia.

Se pueden crear matriz de unos y ceros con los comandos ***ones*** y ***zeros***, un ejemplo de matriz 3×3 se muestra en la Figura 2.17.

Figura 2.17: Creación de matrices 3×3 de unos y ceros

```
Command Window
>> matriz_unos=ones(3,3)
matriz_unos =
     1     1     1
     1     1     1
     1     1     1
>> matriz_ceros=zeros(3,3)
matriz_ceros =
     0     0     0
     0     0     0
     0     0     0
fx
```

Fuente: Elaboración propia.

En la Figura 2.18, se puede crear la matriz identidad usando el comando ***eye*** y la siguiente estructura.

I=***eye(número de filas, número de columnas)***

Figura 2.18: Creación de una matriz identidad 3×3

```
Command Window
>> I=eye(3,3)
I =
     1     0     0
     0     1     0
     0     0     1
fx
```

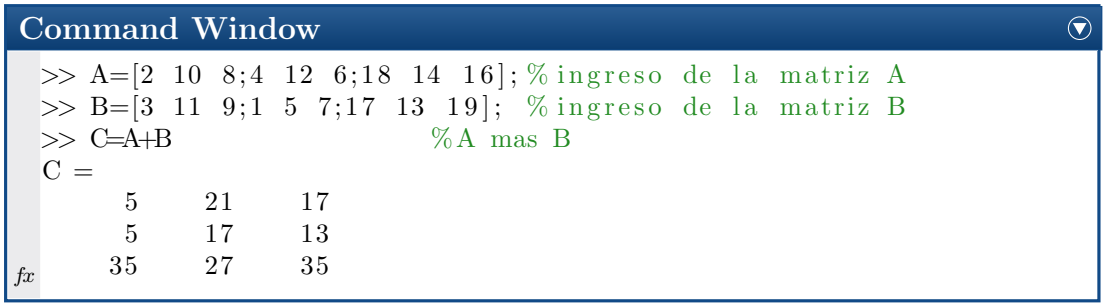
Fuente: Elaboración propia.

2.5.2. Operaciones con matrices

Las operaciones entre matrices se rigen por las reglas del algebra matricial.

Suma La Figura 2.19 presenta un ejemplo.

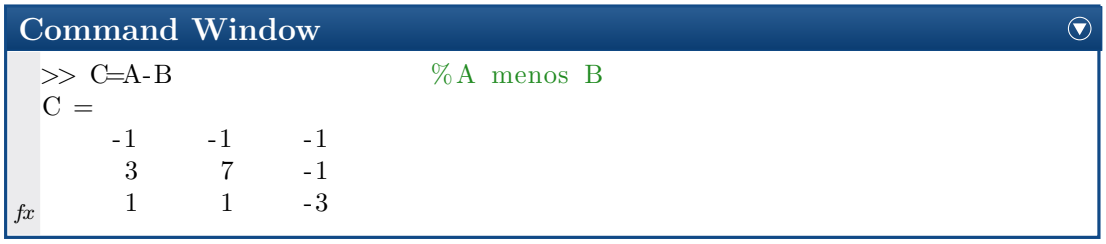
Figura 2.19: Suma de matrices



Fuente: Elaboración propia.

Resta En la Figura 2.20, se describe un ejemplo.

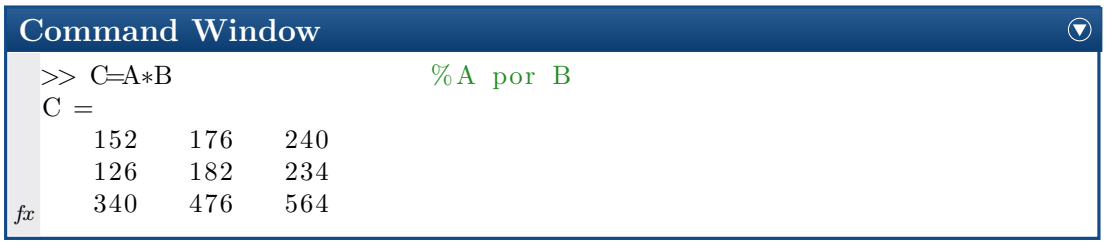
Figura 2.20: Resta de matrices



Fuente: Elaboración propia.

Producto En la Figura 2.21, se describe un ejemplo de $A \times B$.

Figura 2.21: Producto de matrices



Fuente: Elaboración propia.

Transpuesta Un ejemplo de transposición se aprecia en la Figura 2.22.

Figura 2.22: Transpuesta de una matriz

```
Command Window
>> A=[2 10 8;4 12 6;18 14 16]; % ingreso de la matriz A
>> C=A'                        % transpuesta de la matriz A

C =
     2     4    18
    10    12    14
     8     6    16
```

Fuente: Elaboración propia.

Determinante El comando ***det*** se presenta en la Figura 2.23.

Figura 2.23: Determinante de una matriz

```
Command Window
>> c=det(B)                    % determinante de la matriz B
c =
   464.0000
```

Fuente: Elaboración propia.

Inversa El comando ***inv*** se presenta en la Figura 2.24.

Figura 2.24: Inversa de una matriz

```
Command Window
>> C=inv(A)                    %inversa de la matriz A
C =
   -0.1731    0.0769    0.0577
   -0.0705    0.1795   -0.0321
    0.2564   -0.2436    0.0256
```

Fuente: Elaboración propia.

En el apéndice B2, se presentan comandos con ejemplos para generar matrices con diferentes características. Además, se calcula el rango, la norma, la descomposición QR, los valores y vectores propios de una matriz, entre otras operaciones.

2.6. Polinomios

El manejo de polinomios en MATLAB resulta de mucha utilidad en la derivación numérica con Polynomios de Taylor que son tratados en el capítulo 3.

Asimismo, los polinomios son importantes para las cuadraturas interpolarias tratadas en el capítulo 4 y los polinomios ortogonales, cubiertos en el capítulo 5. En MATLAB, los polinomios son vectores fila cuyos elementos son los coeficientes del polinomio en orden de potencia decreciente. Si un término del polinomio no existe se coloca un cero en la posición correspondiente [18].

2.6.1. Ingreso de polinomios

Por ejemplo, para los siguientes polinomios:
$p1 = 2x^3 + 5x^2 - 6x + 8$
$p2 = 2x^3 - x - 5$
El ingreso en MATLAB es como se presenta en la Figura 2.25.

Figura 2.25: Polinomios

```
Command Window
>> p1=[2 5 -6 8];          %polinomio de grado 3
p1 =
     2     5    -6     8
>> p2=[2 0 -1 -5];         %polinomio de grado 3
p2 =
     2     0    -1    -5
fx
```

Fuente: Elaboración propia.

2.6.2. Operaciones con polinomios

Raíces Se obtiene con el comando ***roots*** y la siguiente estructura.

variable=***roots***(variable asignada al polinomio)

En la Figura 2.26 se ilustra el uso del comando roots para el polinomio p1 de la Figura 2.26. Hay que aclarar que MATLAB devuelve las raíces en un vector columna, en el ejemplo de la Figura 2.27 el vector columna es r1.

Figura 2.26: Raíces de un polinomio

```
Command Window
>> r1=roots(p1)

r1 =

  -3.6300 + 0.0000i
   0.5650 + 0.8847i
   0.5650 - 0.8847i
fx
```

Fuente: Elaboración propia.

Creación Un polinomio se puede formar si se conocen las raíces del mismo con el comando ***poly*** y la siguiente estructura (ver ejemplo en Figura 2.27).

variable=***poly***(variable asignada al vector con las raíces del polinomio)

Figura 2.27: Polinomio a partir de las raíces

```
Command Window
>> r1=[-3.6;0.565+0.8847i;0.565-0.8847i]; % raices del polinomio
>> p3=poly(r1)              % cracion del polinomio

p3 =

    1.0000    2.4700   -2.9661    3.9669
```

Fuente: Elaboración propia.

El resultado que MATLAB entrega es un vector fila con las coeficientes, por lo tanto el polinomio es:

$$p3 = x^3 + 2{,}47x^2 - 2{,}9661x + 3{,}9669$$

Evaluación MATLAB puede evaluar un polinomio para un determinado valor, en la Figura 2.28 se muestra un ejemplo y el uso de la estructura:

variable=***polyval***(nombre del polinomio, valor a evaluar)

Figura 2.28: Evaluación de un polinomio

```
Command Window
>> p1=[1 0 -8 6 -10];
>> x=polyval(p1,2)          % polinomio p1 evaluado en 2; p1(2)= -14
x =
   -14
```

Fuente: Elaboración propia.

Producto La estructura para calcular el producto de polinomios en MATLAB es mediante la utilización de la función ***conv*** (de producto de convolución):

variable=***conv***(primer polinomio, segundo polinomio)

La Figura 2.29 presenta un ejemplo.

Figura 2.29: Producto de polinomios

```
Command Window
>> p1=[1 -2 4];
>> p2=[1 0 3 -4];
>> p3=conv(p1,p2)
p3 =
     1     -2      7    -10     20    -16
```

Fuente: Elaboración propia.

División Para dividir polinomios existe la función ***deconv***, con la que se obtiene el cociente y el resto de dos polinomios, siguiendo la estructura.

[cociente, resto]=***deconv***(primer polinomio, segundo polinomio)

La Figura 2.30 presenta un ejemplo.

Figura 2.30: División de polinomios

```
Command Window
>> p1=[1 0 -8 6 -10];
>> p2=[1 0 3 -4];
>> [cociente,resto]=deconv(p1,p2)
cociente =
     1     0
resto =
     0     0    -11     10    -10
```

Fuente: Elaboración propia.

2.7. Script de MATLAB

En MATLAB, se utiliza archivos con la extensión *.m* [16] que poseen conjuntos de instrucciones o definición de funciones. Estos archivos se ejecutan al teclear su nombre en el Command Windows y presionar enter, si éste se encuentra en el Current Folder. También se ejecutan al presionar la tecla *RUN*. Un archivo *.m* o script puede invocar funciones y ejecutar otros achivos *.m*. Para la creación de un archivo *.m*, también conocido como fichero se utiliza el Editor de MATLAB [14]. Para ello, ***Editor*** > ***New*** > ***Script*** (Ctrl-N) y se desplegará la ventana que se aprecia en la Figura 2.31.

Figura 2.31: Ventana del Editor en MATLAB

Fuente: Elaboración propia.

El editor presenta los diferentes tipos de comandos que se escriben en un programa (ver ejemplo en la Figura 2.32.) de acuerdo a lo siguiente nomenclatura:

- Los comentarios se pintan de color verde y van precedidos del signo %.
- Las cadenas de caracteres están de color violeta.
- Las sentencias se colocan de color negro.
- Las sentencias de estructuras de control se colorean de azul.

Figura 2.32: Ejemplo de un script editado en MATLAB

```
1  % Esta linea ilustra la edicion de un comentario
2  % Los comentarios en MATLAB comienzan con el signo %
3  x = 'La cadena de caracteres se escribe entre comillas simples'
4  n = 10;              % Esta sentencia va con color negro
5  % if-else-end es una sentencia de estructuras de control condicional
6  if n > 5
7      disp('n es mayor que 5');    % disp presenta los caracteres
8  else
9      disp('n es menor o igual que 5');
10 end
```

Fuente: Elaboración propia.

2.8. Estructuras de control

Son parte fundamental de cualquier lenguaje de programación porque permiten que las instrucciones de un programa no sólo se ejecuten en el orden en que están escritas (orden secuencial),sino que permiten modificar esta secuencia; para lo cual existen dos categorías de estructuras de control [9]:

- Estructuras de decisión o condicionales
- Estructuras de repetición

2.8.1. Estructuras de decisión

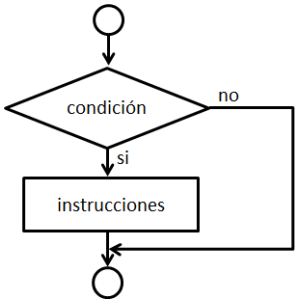
Durante la resolución de cualquier problema de ingeniería, es normal tener en consideración el uso de condiciones que influyan sobre la secuencia de nuestras instrucciones. Para ello, las estructuras de decisión o condicionales son una herramienta para la selección de lo que se debería ejecutar o no en un programa [5]. MATLAB considera tres tipos de estructuras condicionales: Simple, doble y múltiple.

Estructura condicional simple

Este tipo de estructura se caracteriza porque utiliza una sentencia *if* que permite elegir si se ejecuta o no un bloque de instrucciones, tal como se aprecia en la Figura 2.33.

Figura 2.33: Estructura Condicional Simple: Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

```
1 % Condicional simple
2
3 if condicion
4     instrucciones;
5 end
```



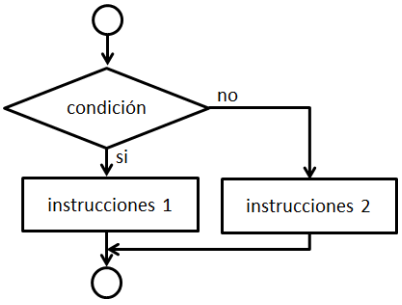
Fuente: Obtenida de [17].

Estructura condicional doble

La estructura condicional doble se compone además de la cláusula *if*, de la sentencia *else*, la cual permite ejecutar un bloque de instrucciones si la condición es verdadera. Si la condición es falsa entonces ejecuta otro bloque de instrucciones diferente (ver Figura 2.34).

Figura 2.34: Estructura Condicional Doble: Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

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



Fuente: Obtenida de [17].

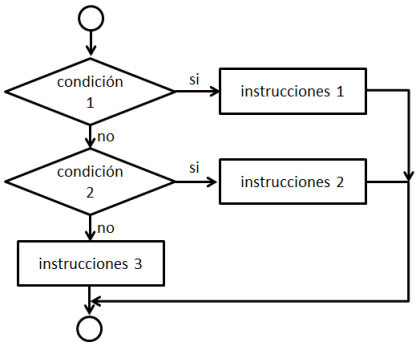
Estructura condicional múltiple

En su forma más general, la estructura $if - elseif - else$ permite implementar condicionales más complejas (ver Figura 2.35), en las que se encadenan condiciones de la siguiente forma:

- *Si* se verifica la *condición 1*, se ejecuta el conjunto de instrucciones del *bloque 1*.
- *Si no* se verifica la *condición 1*, pero si se verifica la *condición 2*, se ejecuta el conjunto de instrucciones del *bloque 2*.
- *Si no*, lo que significa que no se ha verificado ninguna de las condiciones anteriores, se ejecuta el conjunto de instrucciones del *bloque 3*.

Figura 2.35: Estructura Condicional Múltiple: Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

```
% Condicional Multiple
if condicion1
    instrucciones1;
elseif condicion2
    instrucciones2;
else
    instrucciones3;
end
```

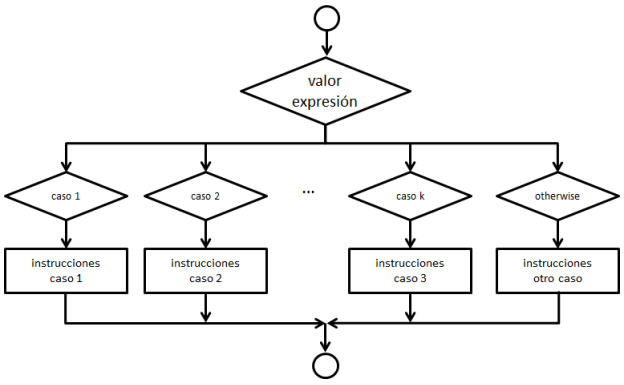


Fuente: Obtenida de [17].

Otra variante de la estructura condicional múltiple, se presenta en la sentencia *switch*, la cual es un tipo de estructura que permite decidir entre varios caminos posibles, en función del *valor* que tome una determinada instrucción, tal como se aprecia en la Figura 2.36

Figura 2.36: Estructura Condicional Múltiple SWITCH/CASE: Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

```
% Estructura Condicional ...
    SWITCH/CASE
switch valor
    case 1
        instrucciones_caso1;
    case 2
        instrucciones_caso2;
    otherwise
        instrucciones_otro_caso;
end
```



Fuente: Obtenida de [17].

2.8.2. Estructuras de repetición

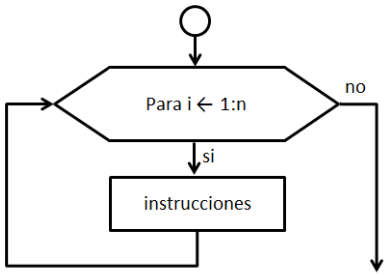
Este tipo de sentencias permiten la ejecución reiterada de un conjunto de instrucciones, ya sea un número predeterminado de veces, o bien hasta que se verifique una determinada condición. En general, MATLAB utiliza dos tipos de estructuras de repetición: Estructura de Repetición Indexada y Condicionada.

Estructura de Repetición Indexada

Este tipo de estructura permite implementar la repetición de un cierto conjunto de instrucciones un número pre-determinado de veces (ver Figura 2.37). Para ello se utiliza una variable de control del bucle, llamada también *índice*, que va recorriendo un conjunto prefijado de valores en un orden determinado. Para cada valor del índice en dicho conjunto, se ejecuta el mismo conjunto de instrucciones una vez.

Figura 2.37: Estructura de Repetición Indexada : Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

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



Fuente: Obtenida de [17].

Estructura de Repetición Condicionada

Permite implementar la repetición de un mismo conjunto de instrucciones mientras se verifique una determinada condición. El número de veces que se repetiría el ciclo no está definido previamente, sino que está definido por una condición (ver Figura 2.38).

1. Al comienzo de cada iteración se evalúa la condición.

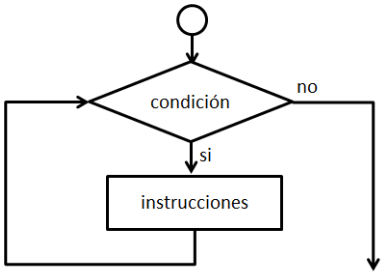
2. Si el resultado es verdadero, se ejecuta el conjunto de instrucciones y se vuelve a iterar, es decir, se repite el paso 1.

3. Si el resultado es falso, se detiene la ejecución del ciclo *while* y el programa se sigue ejecutando por la instrucción siguiente al *end*

Figura 2.38: Estructura de Repetición Condicionada : Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

```
% Estructura Repetitiva WHILE

while condicion
    instrucciones;
end
```



Fuente: Obtenida de [17].

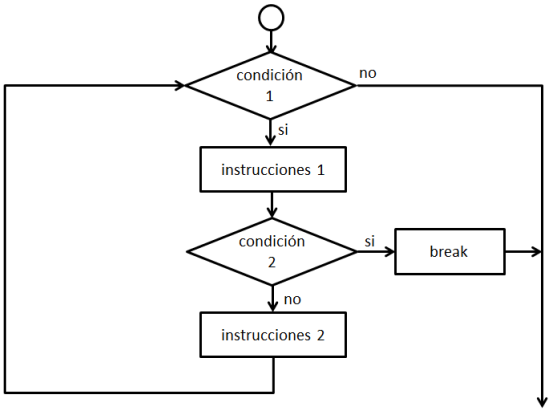
Interrupción con BREAK

En ocasiones es necesario interrumpir la ejecución de un ciclo de repetición en algún punto interno del bloque de instrucciones que se repiten. Esta interrupción está vinculada a la verificación o no de alguna condición. La sentencia *break* abandona el ciclo de repetición definitivamente como se aprecia en la Figura 2.39.

Figura 2.39: Interrupción con BREAK: Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

```
% Interrupcion con BREAK

while condicion1
    instrucciones
    if condicion2
        break;
    end
end
```



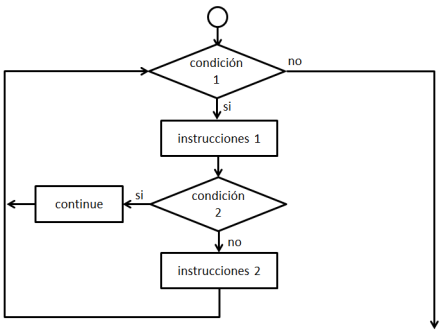
Fuente: Obtenida de [17].

Interrupción con CONTINUE

La interrupción del bucle o lazo con la sentencia *continue* garantiza que se abandone la iteración en curso, pero comenzando la siguiente como se muestra en la Figura 2.40.

Figura 2.40: Interrupción con CONTINUE: Síntaxis en MATLAB (izquierda) y Diagrama de flujo (derecha)

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



Fuente: Obtenida de [17].

En al apéndice B4 se desarrollan varios programas utilizando estructuras condicionales y repetitivas en sus diferentes variantes.

2.9. Funciones

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 [7]. Una función comprende un conjunto de instrucciones que se escriben separadamente del programa para realizar alguna tarea específica En MATLAB, los usuarios pueden definir dos tipos de funciones: Funciones en línea y modulares.

2.9.1. Funciones en línea

Las funciones en línea (*inline functions*, en inglés) se crean en la misma ventana de comandos que puede ser llamada de forma repetida. Por ejemplo, la creación de la función en línea $f(x) = e^{-x^2}$, resulta de escribir el nombre la función, seguidamente el signo igual (=) y luego el comando ***inline***. Dentro de este, el cuerpo de la función debe ser escrito entre comillas simples (' ') como si estuviésemos definiendo una cadena de caracteres como se aprecia en la Figura 2.41.

Figura 2.41: Creación de la función en línea $f(x) = e^{-x^2}$

```
Command Window
>> f=inline('exp(-x^2)')
f =
     Inline function:
     f(x) = exp(-x^2)
fx >>
```

Fuente: Elaboración propia.

De este modo, cuando ingresemos un valor, este será evaluado en la función y retornará el valor respectivo al terminar la ejecución de la misma como se observa a continuación en la Figura 2.42.

Figura 2.42: Evaluación de la función en línea $f(x) = e^{-x^2}$

```
Command Window
>> f(4)
ans =
    1.1254e-07

>> f(0.02)
ans =
    0.9996
fx
```

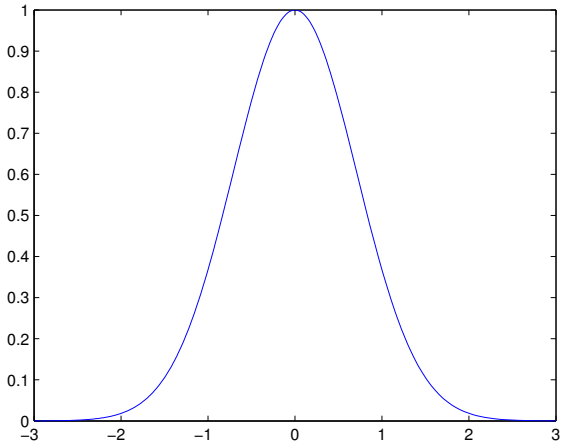
Fuente: Elaboración propia.

Esta misma función se puede graficar utilizando el comando ***fplot***, en un intervarlo de interés; por ejemplo, sería suficiente ejecutar la sentencia

fplot(función,[mínimo máximo])

para graficar la función $f(x)$ en el intervalo de $[-3, 3]$ como se aprecia en la Figura 2.43.

Figura 2.43: Gráfica de la función en línea $f(x) = e^{-x^2}$



Fuente: Elaboración propia.

En el apéndice B3, se presentan ejemplos de varios comandos de graficación en dos y tres dimensiones; asi como también, se puede encontrar implementaciones de funciones en el apéndice B5.

2.9.2. Funciones modulares

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*. Los datos son evaluados en las funciones 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 [2]. La sintaxis utilizada para la declaración de una función se presenta a continuación en la Figura 2.44

Figura 2.44: Declaracion de una función en MATLAB

```
function [variables_salida] = nombre (variables_entrada)
        instrucciones
% variables_salida:         contienen los valores que
%                           entrega la funcion
% variables_entrada:        son variables que reciben los datos que
%                           ingresan a la funcion.
% nombre:                   identificacion de la funcion
% instrucciones:            describen las tareas que realiza
%                           la funcion
```

Fuente: Elaboración propia.

- 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. En la ventana de comandos, se debe especificar la ubicación de la carpeta que contiene la función.
- La función modular se usa similarmente a las funciones comunes en MATLAB. Los argumentos poseen nombres diferentes, pero su uso debe ser coherente.

Las principales diferencias entre un script o fichero de MATLAB con una función, se puede apreciar en la Tabla 2.1 [6].

Tabla 2.1: 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.

Fuente: Elaboración propia.

Bibliografía

[1] Ataurima, M. (2013) *MATLAB & Simulink para Ingeniería Nivel I*. Peru: Universidad de Ciencias y Humanidades.

[2] Attaway, S. (2016) *Matlab: A Practical Introduction to Programming and Problem Solving*. Elsevier Science. https://goo.gl/gLL8MX

[3] Bermúdez, E. P. and Espada, F. M. (2010) *Fundamentos de control con MATLAB*. Pearson Educación.

[4] Chapman, S. J. (2015) *MATLAB Programming for Engineers*. Cengage Learning. https://goo.gl/skvwup

[5] Chapman, S. J. (2016) *Essentials of MATLAB programming*. Cengage Learning.

[6] García de Jalón, J. Rodríguez, J. and Vidal, J. (2005) *Aprenda Matlab 7.0 como si estuviera en primero [on line]*. Madrid: Universidad Politécnica de Madrid. Escuela Técnica Superior de Ingenieros Industriales.

[7] Gdeisat, M. and Lilley, F. (2012) *MATLAB® by Example: Programming Basics*. Elsevier Science: Elsevier insights. https://goo.gl/6JqwaT

[8] Gilat, A. (2006) *Matlab: Una introducción con ejemplos prácticos*. Reverté.

[9] Hahn, B. and Valentine, D. T. (2016) *Essential MATLAB for engineers and scientists*. Academic Press.

[10] Higham, D. J. and Higham, N. J. (2005) *MATLAB guide*. SIAM.

[11] Hunt, B. R. Lipsman, R. L. and Rosenberg, J. M. (2014) *A guide to MATLAB: for beginners and experienced users*. Cambridge, Reino Unido: Cambridge University Press.

[12] Kattan, P. (2008) *Matlab for Beginners: A gentle approach*. Petra Books.

[13] Lee, H. (2016) *Programming with MATLAB 2016*. SDC Publications. https://goo.gl/WFznYE

[14] Lockhart, S. and Tilleson, E. (2017) *An Engineer's Introduction to Programming with MATLAB 2017*. SDC Publications. https://goo.gl/siHzEy

[15] MathWorks (2017) *Fundamental MATLAB Classes*. https://goo.gl/fr5tNi

[16] Moler, C. B. (2004) *Numerical computing with MATLAB*. SIAM.

[17] San Martín Cuenca, H. D. and Tusa Jumbo, E. A. (2015) *Fundamentos de programación para ciencias e ingeniería*. Machala, Ecuador.

[18] Sayood, K. (2007) *Learning Programming Using MATLAB*. CMorgan & Claypool Publishers: Online access: IEEE (Institute of Electrical and Electronics Engineers) IEEE Morgan & Claypool Synthesis eBooks Library. https://goo.gl/j9kEhY

[19] SINGH, Y. K. and CHAUDHURI, B. B. (2007) *MATLAB PROGRAMMING*. PHI Learning. https://goo.gl/peD2bP

[20] Venkataraman, P. (2009) *Applied Optimization with MATLAB Programming*. Wiley. https://goo.gl/W1BWD1

Bibliografía

[1] Cordero-Barbero, A. Hueso-Pagoaga, J. L. Martínez-Molada, E. y Rorregrosa-Sánchez, J. R. (2006) *Problemas Resueltos de Métodos Numéricos*. Madrid, España: Thomson.

[2] Doubova, A. y Guillén-González F. (2007) *Un Curso de Cálculo Numérico: Interpolación, Aproximación, Integración y Resolución de Ecuaciones Diferenciales*. Sevilla, España: Servicio de Publicaciones de la Universidad de Sevilla.

[3] Gander, W. y Gautschi, W. (2000) Adaptive Quadrature - Revisited. BIT. *CS technical report*, *40*(1), 84-101.

[4] Gautschi, W. (2002) *Numerical Analysis*. New York, Estados Unidos: Springer.

[5] Johansen, A. M. y Evers, L. (2011) *Simulation and the Monte Carlo Methods*. Bristol, Reino Unido: Nick Whiteley.

[6] Kalos, M. H. y Whitlock, P. A. (2008) *Monte Carlo Methods*. New York. Estados Unidso: Wiley Online Library.

[7] Kincaid, D. y Cheney, W. (1994) *Análisis numérico. Las matemáticas del cálculo científico*. Mexico D. F. México: Addison-Wesley Iberoamericana, S. A.

[8] Martínez-Finkelshtein, A. y Moreno-Balcázar, J. J. (1999) *Métodos Numéricos: Aproximación en* $\mathbb{R}$. Almería, España: Servicio de Publicaciones de la Universidad de Almería.

[9] Mathews, J. H. y Fink K. D. (2000) *Métodos Numéricos con Matlab*. Madrid, España: Prentice Hall.

[10] Quarteroni, A. y Saleri, F. (2006) *Cálculo Cientifico con MATLAB y Octave*. Milan, Italia: Springer.

[11] Ramírez-González, V. Barrera-Rosillo, D. Pasadas-Fernández, M. y González-Rodelas, P. (2001) *Cálculo Numérico con Mathematica*. Barcelona, España: Editorial Ariel, S. A.

[12] Temme, N. M. (2010) *Numerical Methods, in NIST Handbook of Mathematical Functions*. Cambridge, UK: Cambridge University Press.

Apéndice B

En este Apéndice se plantea escribir todas las implementaciones en MATLAB que han sido necesarias durante el desarrollo del Capítulo 2.

B.1. Cuestiones resueltas

Determine el resultado de las siguientes operaciones desarrollando paso a paso la jerarquía de variables

B.1.1. $4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2$

Desarrollo:

$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2$
$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 4 * 2 + (9 + 2) \wedge 2 - 9 \wedge 2$
$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 4 * 2 + 11 \wedge 2 - 9 \wedge 2$
$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 4 * 2 + 121 - 9 \wedge 2$
$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 4 * 2 + 121 - 81$
$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 8 + 121 - 81$
$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 129 - 81$
$4 * 2 + (3 * 3 + 2) \wedge 2 - 9 \wedge 2 = 48$

B.1.2. $7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2$

Desarrollo:

$7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2 = 7 \wedge 2 + 3 * 4 + (6 + 5) \wedge 2$
$7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2 = 7 \wedge 2 + 3 * 4 + 11 \wedge 2$
$7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2 = 49 + 3 * 4 + 11 \wedge 2$
$7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2 = 49 + 3 * 4 + 121$
$7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2 = 49 + 12 + 121$
$7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2 = 61 + 121$
$7 \wedge 2 + 3 * 4 + (2 * 3 + 5) \wedge 2 = 182$

B.1.3. Seleccione un nombre de variable que no sea reconocido en MATLAB:

A. $a\&$

B. $aceleracion$

C. a_1

D. $a1$

Desarrollo: La opción correcta es A. ya que el símbolo & no puede ser parte del nombre de una variable.

B.1.4. Seleccione un nombre de variable que no sea reconocido en MATLAB:

A. a

B. $aceleracion$

C. $a\ 1$

D. $ac98$

Desarrollo: La opción correcta es C ya que no pueden existir espacios en blanco en el nombre de la variable.

B.1.5. ¿Cuál de las siguientes variables es tipo caracter?

A. $a = 9.8$

B. $aceleracion =$ '9.80'

C. $a\ 10 = 9.8$

D. $ac98 = 3 * 3{,}2$

Desarrollo: La opción correcta es B ya que el contenido de la variable aceleracion está entre comillas simples.

B.1.6. ¿Cuál de las siguientes variables es tipo lógico?:

A. $a =$ 'false'

B. $aceleracion = falso$

C. $a\ 10 = 3 * true$

D. $ac98 = true$

Desarrollo: La opción correcta es D ya que la variable $ac98$ posee un valor lógico de verdadero.

B.2. Matrices

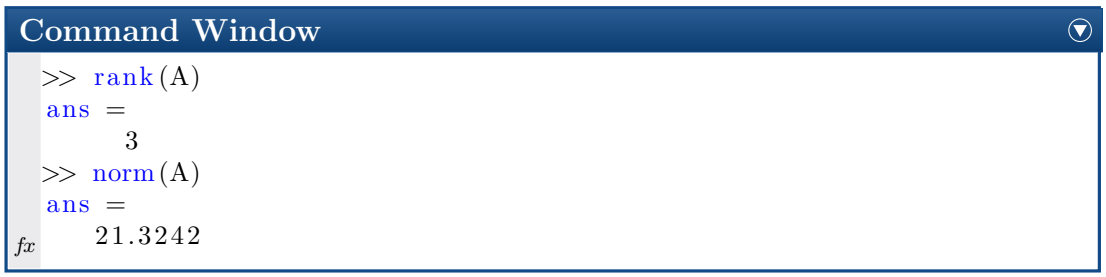
Se puede generar una matriz de 3×3 de números aleatorios enteros entre 0 y 10, mediante la siguiente instrucción visualizada en la Figura B.1

Figura B.1: Matriz de 3×3 de números aleatorios enteros entre 0 y 10

Fuente: Elaboración propia.

El rango y la norma de una matriz puede ser encontrada fácilmente mediante los comandos ***rank*** y ***norm***, respectivamente; como se aprecia en la Figura B.2

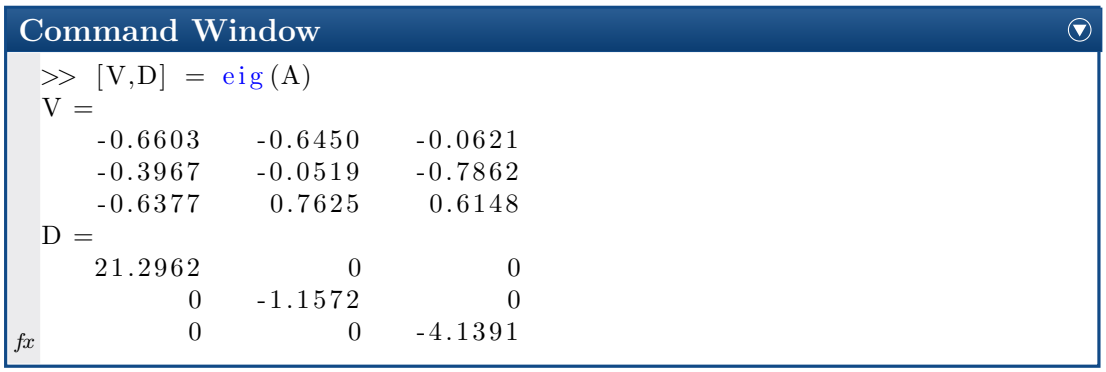
Figura B.2: Rango y norma de una matriz



Fuente: Elaboración propia.

Los valores y vectores propios de la matriz se obtienen a partir del comando ***eig*** como se presenta en la Figura B.3. La matriz V contiene los vectores propios y la matriz D contiene los valores propios en su diagonal principal.

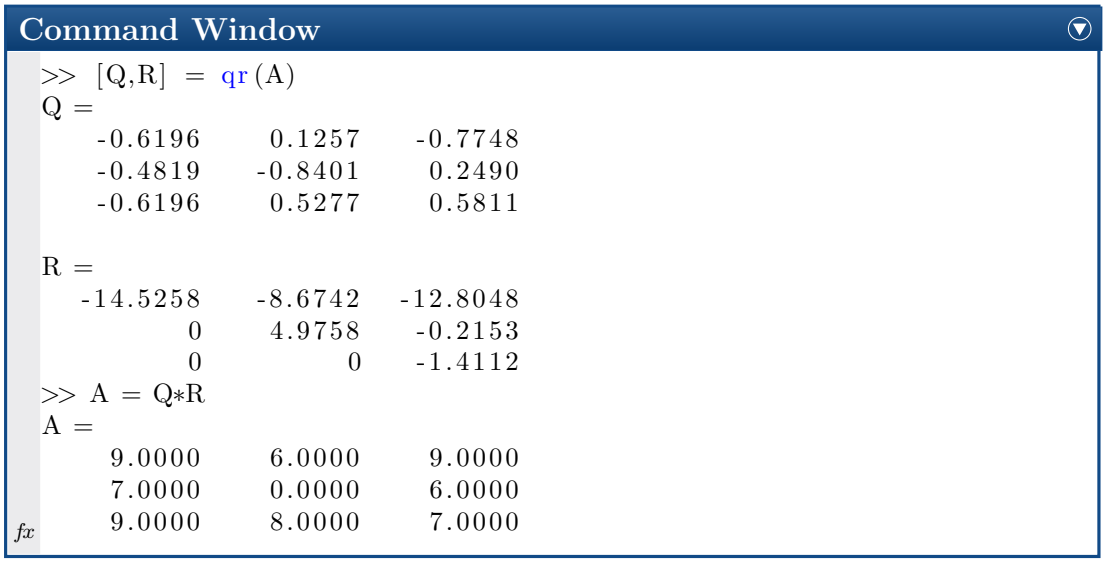
Figura B.3: Valores y vectores propios



Fuente: Elaboración propia.

La descomposición QR que genera una matriz ortogonal Q por una triangular superior R, tal que $A = Q * R$. Esta operación está implementada en MATLAB mediante el comando ***qr*** como se aprecia en la Figura B.4.

Figura B.4: Descomposición QR



Fuente: Elaboración propia.

La descomposición LU que genera una matriz triangular inferior L por una triangular superior U, tal que $A = L * U$. Esta operación está implementada en MATLAB mediante el comando ***lu*** como se aprecia en la Figura B.5.

Figura B.5: Descomposición LU

```
Command Window
>> [L,U] = lu(A)
L =
    1.0000         0         0
    0.7778    1.0000         0
    1.0000   -0.4286    1.0000

U =
    9.0000    6.0000    9.0000
         0   -4.6667   -1.0000
         0         0   -2.4286
>> A = L*U
A =
    9.0000    6.0000    9.0000
    7.0000    0.0000    6.0000
    9.0000    8.0000    7.0000
fx
```

Fuente: Elaboración propia.

B.3. Representaciones Gráficas

Los gráficos son herramientas muy utilizadas para presentar todo tipo de información; información que puede proceder de cualquier campo del conocimiento, pero especialmente de las disciplinas relacionadas con las ciencias y la ingeniería, donde MATLAB es ampliamente utilizado. Con los comandos de MATLAB se pueden crear distintos tipos de gráficos. En esta sección se describe cómo se utiliza MATLAB para crear y dar forma a gráficos de dos y tres dimensiones.

B.3.1. Comandos para Gráficos 2D

Las gráficas 2D de MATLAB están fundamentalmente orientados a la representación gráfica de vectores. El ***Comando plot*** realiza gráficas de funciones de la forma $y = f(x)$, con MATLAB, el comando más elemental es ***plot***, por ejemplo: dibujar $y = sen(x)$. Primero, se crea un vector de valores para x, luego se calcula los valores de y como se observa en la Figura B.6.

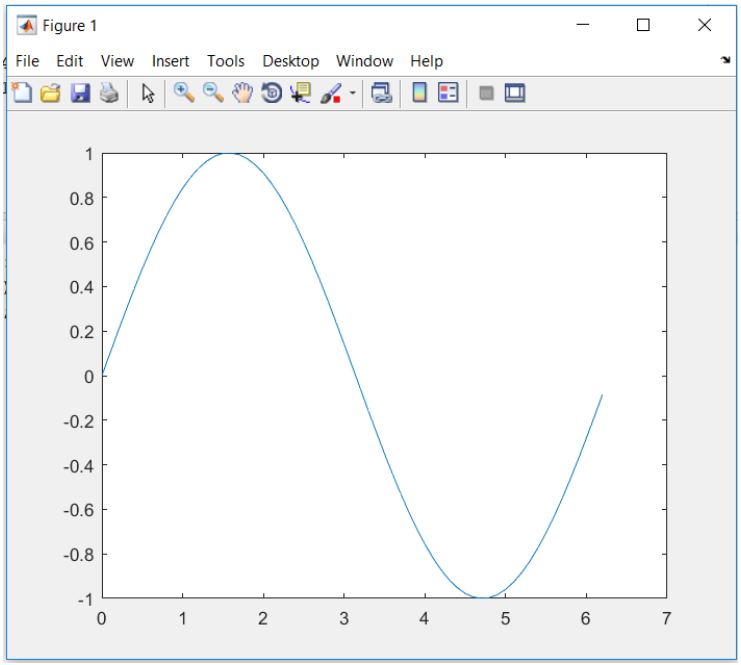
Figura B.6: Comando plot

```
1 x = 0:0.1:2*pi;          % ingreso del vector x
2 y = sin(x);              % calculo del la funcion y=f(x)
3 figure(1);               % nueva figura
4 plot(x,y);               % grafico de la funcion y=f(x)
```

Fuente: Elaboración propia.

El resultado del ejemplo de la Figura B.6 es que se abre una ventana mostrando el gráfico de la Figura B.7. Por defecto, los distintos puntos del gráfico se unen con una línea continua. También por defecto, el color que se utiliza para la primera línea es el azul.

Figura B.7: Función $y = sen(x)$



Fuente: Elaboración propia.

MATLAB utiliza un tipo especial de ventanas para realizar las operaciones gráficas. Ciertos comandos abren una ventana nueva y otros dibujan sobre la ventana activa, bien sustituyendo lo que hubiera en ella, bien añadiendo nuevos elementos gráficos a un dibujo anterior.

La creación de nuevas gráficas se realiza con el comando ***figure***. Para mostrar varios resultados sobre una misma gráfica, es necesario activar el comando ***hold*** como se muestra en la Figura B.8

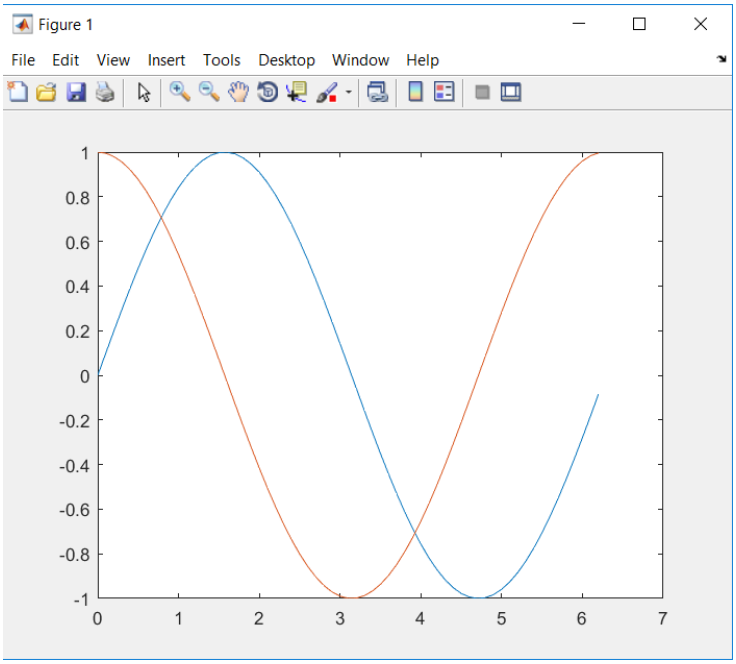
Figura B.8: Comando hold on

```
1 x = 0:0.1:2*pi;      % ingreso del vector x
2 y = sin(x);          % calculo del la funcion y=f(x)
3 figure(1);           % nueva figura
4 plot(x,y)            % grafico de la funcion y=f(x)
5 hold on              % mantiene la grafica anterior
6 z=cos(x);            % calcula nueva funcion z=g(x)
7 plot(x,z)            % adiciona la representacion de z
```

Fuente: Elaboración propia.

El resultado del uso del comando ***hold*** en la Figura B.8 se muestra en la Figura B.9

Figura B.9: Funciones $y = sen(x)$ y $z = cos(x)$



Fuente: Elaboración propia.

Y cuando no se quiera superponer más resultados sobre la misma gráfica se debe desactivar el comando ***hold***; a partir de las siguientes lineas una nueva ejecución del comando ***plot*** generará únicamente los resultados posteriores. Un ejemplo se da en la Figura B.10.

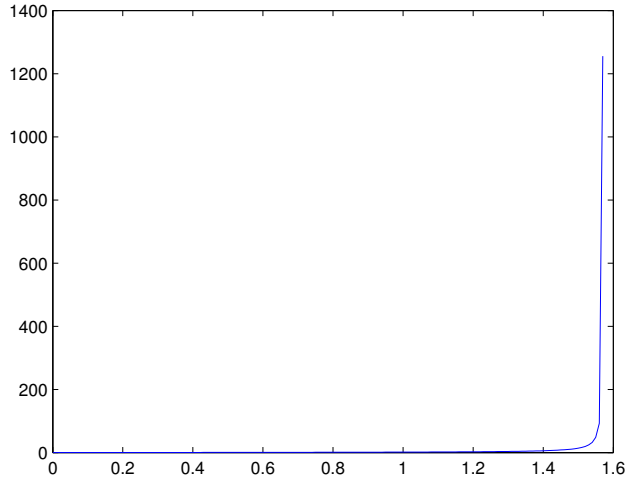
Figura B.10: Comando hold off

```
1  x = 0:0.1:2*pi;      % ingreso del vector x
2  y = sin(x);
3
4  figure(1);
5  plot(x,y)            % grafico de y
6  hold on              % mantiene la grafica anterior
7  z = cos(x);
8  plot(x,z)            % adiciona la representacion de z
9  hold off             % desactiva el comando hold
10
11 x = 0:0.01:pi/2;     % nueva entrada
12 w = tan(x);
13 plot(x,w)            % nueva representacion grafica
```

Fuente: Elaboración propia.

El gráfico del ejemplo de la Figura B.10 se observa en la Figura B.11, en este caso solo la gráfica de la tangente se presenta como resultado final del código.

Figura B.11: Grafica de la función $w = tan(x)$



Fuente: Elaboración propia.

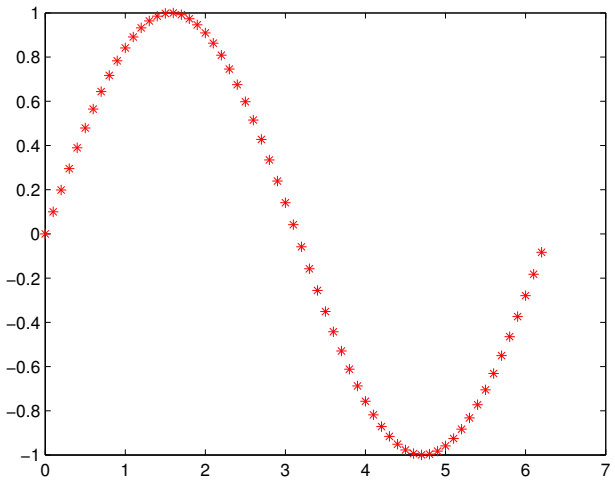
El comando ***plot*** ofrece múltiples posibilidades, en realidad, el conjunto básico de argumentos de esta función es una tripleta formada por dos vectores y una cadena de 1, 2 ó 3 caracteres entre comillas simples que indica el color, símbolo y tipo de linea. El atributo color se especifica con la primera letra en inglés del mismo: 'r' para rojo, 'b' para azul, 'g' para verde, etc. El atributo símbolo, empleado para remarcar los resultados gráficos, pueden ser: 'o', '*', '+', '×', etc. El tipo de línea se especifica con los caracteres: '-' para línea sólida, ':'para línea discontinua, 'none' sin línea, etc. En la Figura B.12 se da un código de ejemplo y el resultado se presenta en la Figura B.13.

Figura B.12: Comando plot con tres argumentos

```
1 x = 0:0.1:2*pi;          % ingreso del vector x
2 y = sin(x);
3 figure(1);
4 plot(x,y,'r*')           % grafico de y en color rojo con *
```

Fuente: Elaboración propia.

Figura B.13: Trazada con asteriscos rojos



Fuente: Elaboración propia.

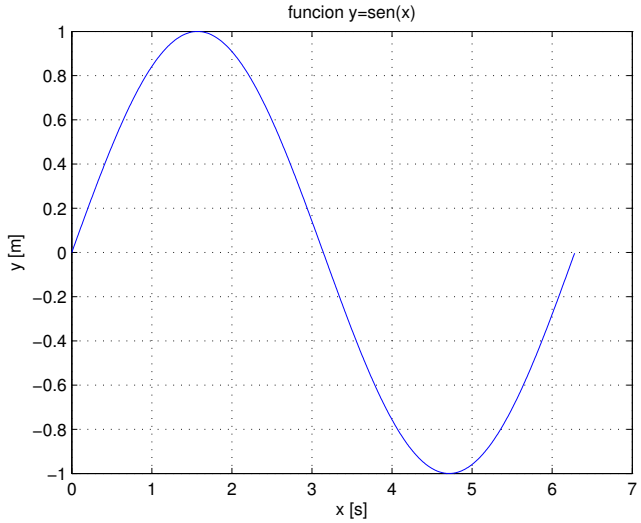
El siguiente grupo de comandos permiten agregar a la gráfica anterior una rejilla, un texto a los ejes coordenados x y y y un título, mediante los siguientes comandos: ***grid on***, ***xlabel***, ***ylabel*** y ***title***. En la Figura B.14 se presenta un ejemplo. El resultado de los comandos de la Figura B.14 aparecen en la Figura B.15.

Figura B.14: Comandos grid, xlabel, ylabel y title

```
1 x = 0:0.01:2*pi;              % ingreso del vector x
2 y = sin(x);
3 figure(1);
4 plot(x,y)                     % grafico de y
5 grid on                       % activa la rejilla
6 xlabel('x [s]')               % agrega texto al eje de abscisas
7 ylabel('y [m]')               % agrega texto al eje de ordenadas
8 title('funcion y=sen(x)')     % agrega titulo a la grafica
```

Fuente: Elaboración propia.

Figura B.15: Función $y = sen(x)$ con cuadrícula, título y etiquetas en los ejes



Fuente: Elaboración propia.

El comando ***area*** permite mostrar el área bajo la curva de una función. Las Figuras B.16 y B.17 presentan el código y la gráfica de la función:

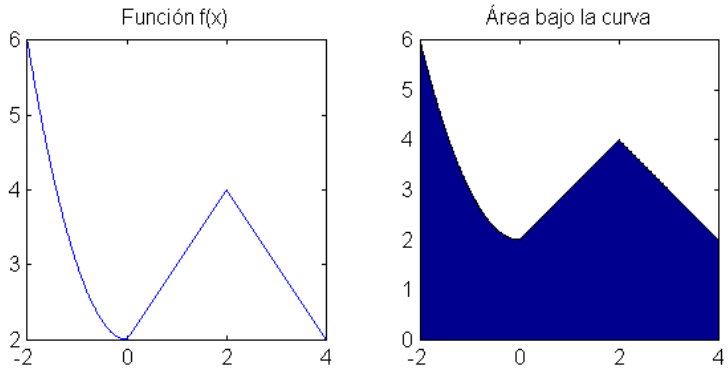
$$f(n) = \begin{cases} x^2 + 4 & , x \leq 0 \\ x + 2 & , 0 < x \leq 2 \\ -x + 6 & , x > 2 \end{cases}$$

Figura B.16: Comando ***area*** para presentar el área bajo la curva

```
1 x = -2:0.01:4;                                                    %Dominio
2 y = (x.^2+2).*(x<=0) + (x+2).*(x>0 & x<=2) + (-x+6).*(x>2);       %Rango
3
4 subplot(1,2,1), plot(x,y), title('Funcion f(x)'), axis square;
5 subplot(1,2,2), area(x,y), title('Area bajo la curva'), axis square;
```

Fuente: Elaboración propia.

Figura B.17: Área bajo la curva de una función



Fuente: Elaboración propia.

B.3.2. Comandos para Gráficos 3D

En MATLAB se pueden obtener gráficos de curvas y superficies en 3D

Comando plot3 Para obtener gráficos de curvas se usa el comando ***plot3***, éste es muy similar al comando ***plot***, excepto que en el argumento de ***plot3*** deben aparecer tres vectores de igual dimensión, en la Figura B.18 se presenta un ejemplo. El resultado de los comandos de la Figura B.18 aparecen en la Figura B.19

.

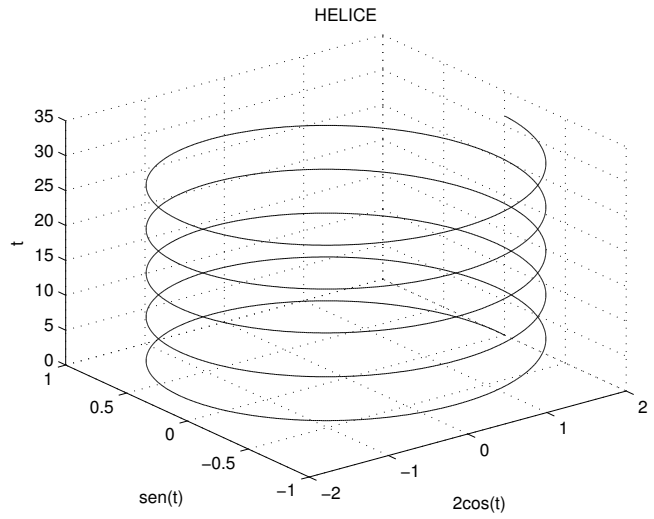
Figura B.18: Uso del comando plot3

```
1  t = 0:0.01:10*pi;          % ingreso del vector t
2  x = 2*cos(t);              % componente en x
3  y = sin(t);                % componente en y
4  z = t;                     % componente en z
5  plot3(x,y,z,'k')           % grafico de la Helice en color negro
6  grid on                    % activa la rejilla
7  xlabel('2cos(t)')          % agrega texto al eje x
8  ylabel('sen(t)')           % agrega texto al eje y
9  zlabel('t')                % agrega texto al eje z
10 title('HELICE')            % agrega titulo a la grafica
```

Fuente: Elaboración propia.

Comandos meshgrip y mesh Para dibujar en MATLAB gráficos de superficies expresadas como 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 y y, estos vectores contienen las coordendas de una rejilla (grid) sobre la que se va a dibujar la función $z = f(x, y)$. Luego, es necesario crear dos

Figura B.19: Gráfica de una Hélice

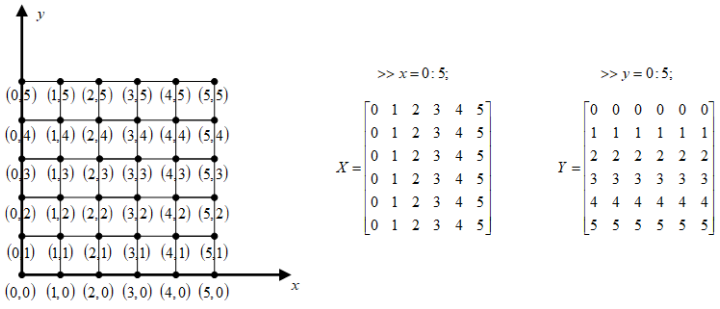


Fuente: Elaboración propia.

matrices, una es X cuyas filas son copias de x y la otra es Y cuyas columnas son copias de y. Estas matrices se crean con el comando **meshgrid**.

Por ejemplo el conjunto de puntos que forman la rejilla en el plano xy correspondientes al dominio $0 \leq x \leq 5$ y $0 \leq y \leq 5$ de la función se presenta en la Figura B.20 junto con las matrices X y Y.

Figura B.20: Rejilla en el plano xy para el dominio $0 \leq x \leq 5$ y $0 \leq y \leq 5$



Fuente: Elaboración propia.

Una vez generada las matrices X y Y se define la función a graficar en la que los argumentos son las matrices X y Y $[Z = f(X, Y)]$, para finalmente usar el comando ***mesh***.

Un ejemplo del uso de los comandos ***meshgrip*** y ***mesh*** se presenta en la Figura B.21

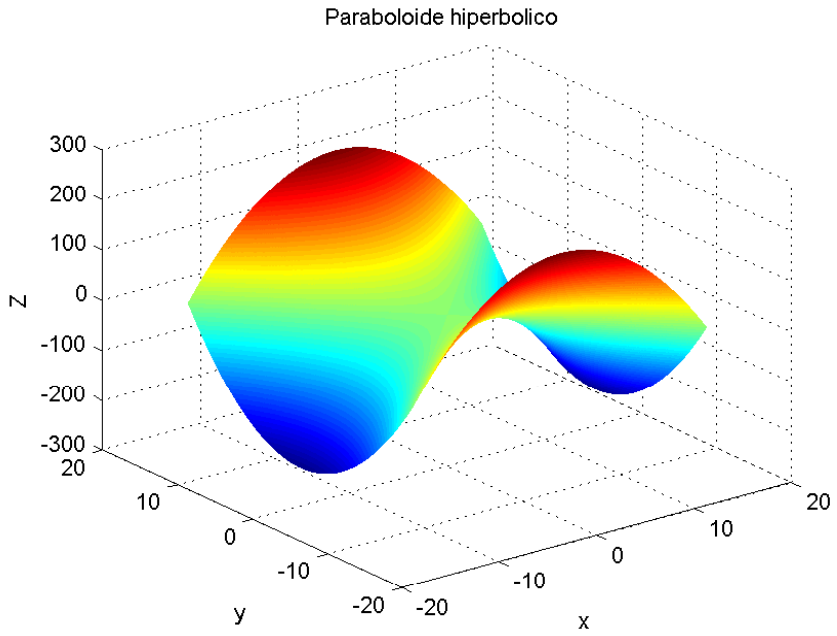
Figura B.21: Uso de los comandos meshgrid y mesh

```
1  x = -15:0.01:15;          % ingreso del vector x
2  y = -15:0.01:15;          % ingreso del vector y
3
4  [X,Y]=meshgrid(x,y);      % generacion matrices X y Y
5  Z=Y.^2-X.^2;              % definicion de la funcion Z
6  mesh(X,Y,Z)               % graficacion de Z
7
8  xlabel('x')               % agrega texto al eje x
9  ylabel('y')               % agrega texto al eje y
10 zlabel('Z')               % agrega texto al eje z
11 title('Paraboloide hiperbolico') % agrega titulo a la grafica
```

Fuente: Elaboración propia.

La gráfica que corresponde a la Figura B.21 se presenta en la Figura B.22

Figura B.22: Gráfica de una superficie en 3D



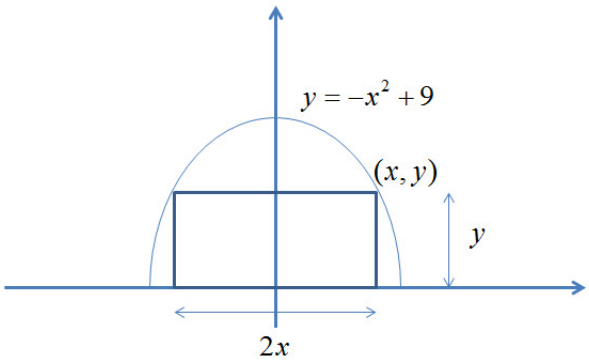
Fuente: Elaboración propia.

B.4. Programación con MATLAB

B.4.1. Programación Secuencial

Encontrar el área del rectángulo más grande dado por la ecuación $A(x, y) = 2xy$, cuya base es $2x$ y altura y, que se puede inscribir en la ecuación de la parábola $y(x) = -Bx^2 + C$, como se muestra en la Figura B.23. Para ello, el usuario debe ingresar los valores de B y C. Nota: Este es un problema que se resuelve optimizando la función del área del rectángulo para determinar las raíces de la derivada del área.

Figura B.23: Rectángulo inscrito en una parábola



Fuente: Elaboración propia.

Desarrollo:

Si reemplazamos la ecuación $y(x) = -Bx^2 + C$ en la la ecuación del área, se obtiene la expresión $A(x) = -2Bx^3 + 2Cx$. La derivada de esta expresión con respecto a x es $\frac{dA(x)}{dx} = -6Bx^2 + 2C$. Si planteamos que $\frac{dA(x)}{dx} = 0$, se desprende que $x = \sqrt{\frac{C}{3B}}$. La implementación de este algoritmo se observa en la Figura B.24 y su ejecución en la Figura B.25

Figura B.24: Cálculo del área máxima de un rectángulo inscrito en una parábola

```
1 disp('Ingrese los coeficientes de la parabola y = -Bx^2 + C')
2 B=input('Ingrese el coeficiente B: ');
3 C=input('Ingrese el coeficiente C: ');
4 x = sqrt(C/(3*B));       % Coordenada en x
5 y = -B*x^2+C;            % Coordenada en y
6 S = 2*x*y;               % Superficie
7 disp('La superficie del rectangulo es: ');
8 disp(S)
```

Fuente: Elaboración propia.

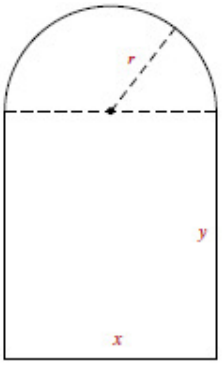
Figura B.25: Ejecución del Programa de la Figura B.24

```
Command Window
  Ingrese los coeficientes de la parabola y = -Bx^2 + C
  Ingrese el coeficiente B: 3
  Ingrese el coeficiente C: 81
  La superficie del rectangulo es:
fx    324
```

Fuente: Elaboración propia.

El programa debe mostrar los valores de las dimensiones de la ventana: x, y; que presenta la forma de un rectángulo coronado por un semicírculo, como se muestra en la figura B.26. Se debe maximizar el área A y considerar que el usuario conoce el valor del perímetro P.

Figura B.26: Forma de la ventana



Fuente: Elaboración propia.

Desarrollo:

Primero, plantemos las ecuaciones para el perímetro P y el área A de la Figura B.26, es decir; $P = x + 2y + \frac{\pi x}{2}$ y $A = \frac{\pi x^2}{8} + xy$. De la ecuación del perímetro, despejamos $y = \frac{1}{2}\left(P - x - \frac{\pi x}{2}\right)$, y reemplazamos en la ecuación del área: $A(x) = \frac{Px}{2} - \frac{(4+\pi)x^2}{8}$. Si planteamos que $\frac{dA(x)}{dx} = 0$, se desprende que $x = 2P/(\pi + 4)$. La implementación de este algoritmo se observa en la Figura B.27 y su ejecución en la Figura B.28

Figura B.27: Cálculo de las dimensiones de la ventana de la Figura B.26

```
P = input('Ingrese el valor del perimetro: ');
x = 2*P/(pi+4);              % Dimension en x
y = (P-x-pi*x/2)/2;          % Dimension en y
disp('El valor de la dimension x es: ');
disp(x);
disp('El valor de la dimension y es: ');
disp(y);
```

Fuente: Elaboración propia.

Figura B.28: Ejecución del Programa B.27

```
Command Window
  Ingrese el valor del perimetro: 10
  El valor de la dimension x es:
      2.8005

  El valor de la dimension y es:
fx    1.4002
```

Fuente: Elaboración propia.

B.4.2. Estructuras Condicionales

Lea un número de tres cifras. Determinar si la suma de las tres cifras es un número par o impar, muestre un mensaje al usuario.

Desarrollo:

La implementación de este algoritmo se observa en la Figura B.29 y su ejecución en la Figura B.30

Figura B.29: Descomposición de un número de 3 cifras

```
x = input('Ingrese el valor de tres cifras: ');
c = floor(x/100);           % Cifra de la centena
r = mod(x,100);             % Residuo de dos cifras
d = floor(r/10);            % Cifra de la decena
u = mod(x,10);              % Cifra de la unidad
y = c+d+u;
if x > 99 & x < 999         % Condicion para valor de 3 cifras
    if mod(y,2)==0;         % Condicion para valor par
        disp('La suma resulta un valor par')
    else
        disp('La suma resulta un valor impar')
    end
else
    disp('El valor ingresado no tiene 3 cifras.')
end
```

Fuente: Elaboración propia.

Figura B.30: Ejecución del Programa de la Figura B.29

```
Command Window
Ingrese el valor de tres cifras: 753
La suma resulta un valor impar
Ingrese el valor de tres cifras: 754
La suma resulta un valor par
```

Fuente: Elaboración propia.

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

Desarrollo:

La implementación de este algoritmo se observa en la Figura B.31 y su ejecución en la Figura B.32

Figura B.31: Clasificación de triángulos basado en las dimensiones de sus lados

```
a = input('Ingrese el primer lado del triangulo: ');
b = input('Ingrese el segundo lado del triangulo: ');
c = input('Ingrese el tercer lado del triangulo: ');

if a+b > c & a+c > b & b+c > a   % Desigualdad del triangulo
    if a == b & b == c
        disp('Es un triangulo equilatero');
    elseif a == b | b == c | a == c
        disp('Es un triangulo isoceles');
    else
        disp('Es un triangulo escaleno');
    end
else
    disp('Las dimensiones de los lados no forman un triangulo.')
end
```

Fuente: Elaboración propia.

Figura B.32: Ejecución del Programa de la Figura B.31

```
Command Window
  Ingrese el primer lado del triangulo: 5
  Ingrese el segundo lado del triangulo: 5
  Ingrese el tercer lado del triangulo: 5
  Es un triangulo equilatero

  Ingrese el primer lado del triangulo: 7
  Ingrese el segundo lado del triangulo: 7
  Ingrese el tercer lado del triangulo: 2
  Es un triangulo isoceles

  Ingrese el primer lado del triangulo: 5
  Ingrese el segundo lado del triangulo: 6
  Ingrese el tercer lado del triangulo: 8
fx Es un triangulo escaleno
```

Fuente: Elaboración propia.

Considere 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 Farenheit F a grados Celsius C con la fórmula $C = \frac{5}{9}(T - 32)$. Si el código es 2 convierta la temperatura t de grados Celsius C a grados Farenheit F con la fórmula $F = 32 + \frac{9}{5}T$.

Desarrollo:

La implementación de este algoritmo se observa en la Figura B.33 y su ejecución en la Figura B.34

Figura B.33: Conversion entre temperaturas Celsius y Farenheit

```
disp('Este programa realiza conversiones de temperatura.')
disp('El codigo 1 indica conversion de Farenheit a Celsius.')
disp('El codigo 2 indica conversion de Celsius a Farenheit.')
T = input('Ingrese la temperatura: ');
p = input('Ingrese el codigo de conversion (1 o 2): ');
switch p
    case 1
        C = 5/9*(T-32);      % Farenheit a Celsius
        disp('La Temperatura en Celsius es:')
        disp(C)
    case 2
        F = 32 + 9/5*T;      % Celsius a Farenheit
        disp('La Temperatura en Farenheit es:')
        disp(F)
    otherwise
        disp('No se ingreso el codigo correctamente')
end
```

Fuente: Elaboración propia.

Figura B.34: Ejecución del Programa de la Figura B.33

```
Command Window
Este programa realiza conversiones de temperatura.
El codigo 1 indica conversion de Farenheit a Celsius.
El codigo 2 indica conversion de Celsius a Farenheit.
Ingrese la temperatura: 93
Ingrese el codigo de conversion (1 o 2): 1
La Temperatura en Celsius es:
   33.8889

Este programa realiza conversiones de temperatura.
El codigo 1 indica conversion de Farenheit a Celsius.
El codigo 2 indica conversion de Celsius a Farenheit.
Ingrese la temperatura: 27
Ingrese el codigo de conversion (1 o 2): 2
La Temperatura en Farenheit es:
   80.6000
fx
```

Fuente: Elaboración propia.

B.4.3. Estructura Repetitivas

Dado un número entero positivo n, se debe descomponerlo en sus factores primos.

Desarrollo:

La implementación de este algoritmo se observa en la Figura B.35 y su ejecución en la Figura B.36

Figura B.35: Descomposición de factores primos

```
n = input('Ingrese un valor positivo n: ');
x = 2;

if n > 0
    disp('Sus factores primos son: ');
    while x ≤ n
        if mod(n,x) == 0     % Condicion de divisibilidad
            n = n/x;         % Descompone el valor
            disp(x);         % Se presenta el factor
        else
            x = x+1;         % Se busca el nuevo factor
        end
    end
else
    disp('El valor ingresado es menor o igual a 0.');
end
```

Fuente: Elaboración propia.

Figura B.36: Ejecución del Programa de la Figura B.35

```
Command Window
Ingrese un valor positivo n: 96
Sus factores primos son:

     2

     2

     2

     2

     2

     3
fx
```

Fuente: Elaboración propia.

Realizar el conteo del número de veces que se hace el lanzamiento de un dado hasta obtener el valor de 5.

Desarrollo:

La implementación de este algoritmo se observa en la Figura B.37 y su ejecución en la Figura B.38

Figura B.37: Conteo de lanzamientos de un dado hasta obtener un 5

```
valor = 0;
cont = 0;        % contador de lanzamientos

while valor ≠ 5
    x = rand();          % Genera valor aleatorio entre 0 y 1
    y = 5*x + 1;         % Normaliza el valor entre 1 y 6
    valor = round(y);    % Discretiza el valor
    disp(valor);         % Muestra el valor
    cont = cont + 1;     % Incrementa el contador
end

disp('El dado se ha lanzado: ');
disp(cont);
disp('veces');
```

Fuente: Elaboración propia.

Figura B.38: Ejecución del Programa de la Figura B.37

```
Command Window

     4

     3

     6

     1

     3

     3

     5

El dado se ha lanzado:
     7
fx veces
```

Fuente: Elaboración propia.

Dado un número entero, n, calcular la suma de los n primeros números pares, terminando el programa cuando aparezca un número divisible para 5

Desarrollo:

La implementación de este algoritmo se observa en la Figura B.39 y su ejecución en la Figura B.40

Figura B.39: Sumatoria den números pares hasta un número divisible para 5

```
n = input('Ingrese un valor entero: ');
suma = 0;            % acumula la suma de valores pares

for i = 1:n
    npar = 2*i;          % crea valores pares
    if mod(npar,5) == 0  % condicion de divisibilidad
        break            % interrumpe la iteracion
    else
        disp(npar);
        suma = suma + npar;  % calcula la suma
    end
end

disp('La suma de los n valores pares es: ');
disp(suma);
```

Fuente: Elaboración propia.

Figura B.40: Ejecución del Programa de la Figura B.39

```
Command Window
  Ingrese un valor entero: 20

       2

       4

       6

       8

  La suma de los n valores pares es:

fx     20
```

Fuente: Elaboración propia.

B.5. Funciones

Desarrolle una función que calcula las raíces de un polinomio de la forma $ax^2 + bx + c$

Desarrollo:

La implementación de esta función se observa en la Figura B.41 y su ejecución en la Figura B.42

Figura B.41: Función que calcula las raíces de un polinomio de la forma ax^2+bx+c

```
1  function r = raices(a,b,c)
2  % Esta funcion calcula la  reices de un polinomio de grado 2
3  % de la forma ax^2 + bx + c
4  % Variables de entrada
5  %   a = coeficiente de x^2
6  %   b = coeficiente de x
7  %   c = coeficiente independiente
8  % Variables de salida
9  %   r = vector que contiene las raices
10
11 x1 = (-b + sqrt(b^2 - 4*a*c))/(2*a);     % raiz 1
12 x2 = (-b - sqrt(b^2 - 4*a*c))/(2*a);     % raiz 2
13
14 r = [x1 x2];                 % salida de mi funcion
```

Fuente: Elaboración propia.

Figura B.42: Aplicación del comando ***help*** para la función ***raíces*** y su invocación desde la Ventana de Comandos.

```
Command Window
>> help raices
  Esta funcion calcula la  reices de un polinomio de grado 2
  de la forma ax^2 + bx + c
  Variables de entrada
    a = coeficiente de x^2
    b = coeficiente de x
    c = coeficiente independiente
  Variables de salida
    r = vector que contiene las raices

>> r = raices(1,-5,6)

r =

     3     2

fx
```

Fuente: Elaboración propia.

Desarrolle una función que calcula el área y el volumen de un cilindro de altura h y radio r.

Desarrollo:

La implementación de esta función se observa en la Figura B.43, su invocación se presenta en la Figura B.44 y la gráfica generada está en la Figura B.45.

Figura B.43: Función que calcula calcula el área y el volumen de un cilindro

```
1  function [V,S] = calculacilindro(r,h)
2  % [V,S] = calculacilindro(r,h)
3  % Esta funcion calcula el volumen y la superficie del
4  % cilindro de radio r y altura h.
5  %   Variables de entrada
6  %   r = radio
7  %   h = altura
8  %   Variables de salida
9  %   V = Volumen
10 %   S = Superficie
11 V = pi*r^2*h;                    % Volumen
12 S = 2*pi*r*h + 2*pi*r^2;         % Superficie
13 [x,y,z] = cylinder(r,100);       % cilindro con radio r con n
14                                  % puntos igualmente espaciados
15 surf(x,y,z*h,'FaceAlpha',0.2, 'EdgeColor', 'none');
```

Fuente: Elaboración propia.

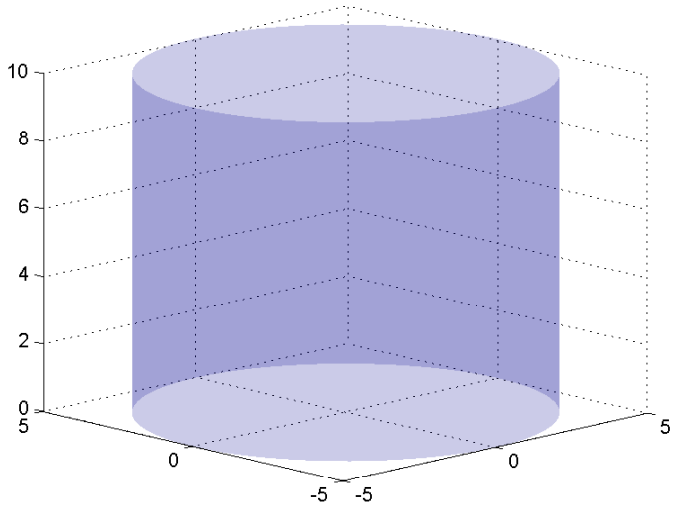
Figura B.44: Aplicación del comando ***help*** para la función ***calcula_cilindro*** y su invocación desde la Ventana de Comandos.

```
Command Window
>> help calcula_cilindro
  [V,S] = calculacilindro(r,h)
  Esta funcion calcula el volumen y la superficie del
  cilindro de radio r y altura h.
    Variables de entrada
    r = radio
    h = altura
    Variables de salida
    V = Volumen
    S = Superficie

>> [V,S] = calcula_cilindro(5,10)}
V =
   785.3982
S =
   471.2389
fx
```

Fuente: Elaboración propia.

Figura B.45: Cilindro de radio $r = 5$ y altura $h = 10$



Fuente: Elaboración propia.

Se puede apreciar que el comando ***help*** acompañado del nombre de la función o comando, despliega la ayuda del archivo .m, que es una breve descripción junto a la sintaxis de la función o de la instrucción, en la Ventana de Comandos. Del mismo modo que otros programas, MATLAB posee una enorme documentación de todas sus librerías y comandos, que puede ser accedida después de presionar la tecla ***F1*** en la Ventana de Comandos.

Métodos Numéricos para el Análisis Matemático con Matlab

Edición digital 2017-2018.
www.utmachala.edu.ec

Redes

Redes es la materialización del diálogo académico y propositivo entre investigadores de la UTMACH y de otras universidades iberoamericanas, que busca ofrecer respuestas glocalizadas a los requerimientos sociales y científicos. Los diversos textos de esta colección, tienen un espíritu crítico, constructivo y colaborativo. Ellos plasman alternativas novedosas para resignificar la pertinencia de nuestra investigación. Desde las ciencias experimentales hasta las artes y humanidades, Redes sintetiza policromías conceptuales que nos recuerdan, de forma empeñosa, la complejidad de los objetos construidos y la creatividad de sus autores para tratar temas de acalorada actualidad y de demanda creciente; por ello, cada interrogante y respuesta que se encierra en estas líneas, forman una trama que, sin lugar a dudas, inervará su sistema cognitivo, convirtiéndolo en un nodo de esta urdimbre de saberes.

UNIVERSIDAD TÉCNICA DE MACHALA
Editorial UTMACH
Km. 5 1/2 Vía Machala Pasaje
www.investigacion.utmachala.edu.ec / www.utmachala.edu.ec

ISBN: 978-9942-24-104-7