MatLab Básico

Alexandra Pazmiño A.
Jairo Jácome T.
Luis A. Zabala A.
Javier J. Gavilanes C.

ESPOCH
2018

MATLAB BÁSICO

MATLAB BÁSICO

Pazmiño A. Alexandra
Jácome T. Jairo
Zabala A. Luis a.
Gavilanes C. Javier J.

DIRECCIÓN DE PUBLICACIONES

MATLAB BÁSICO

Panamericana Sur, kilómetro 1 $^{1}/_{2}$
Dirección de Publicaciones Científicas
Riobamba, Ecuador
Teléfono: (593 3) 2998200
Código Postal: EC060155

Aval Espoch

Este libro se sometió a arbitraje bajo el sistema de doble ciego (*peer review*).

Corrección y diseño:

La Caracola Editores

Impreso en Ecuador

CDU: 004 + 004.7
Riobamba: Escuela Superior Politécnica de Chimborazo
Dirección de Publicaciones, año 2018
185 pp. vol: 17 x 24 cm
ISBN: 978-9942-35-348-1
1. Informática
2. *Software*
3. MatLab

CONTENIDO GENERAL

ÍNDICE DE FUNCIONES

Funciones trigonométricas

Funciones logarítmicas y exponenciales

Funciones específicas de números reales

Funciones específicas para la parte real e imaginaria

Generación rápida de vectores

Funciones elementales que admiten como argumento un vector real o complejo

Funciones para determinar las características de una matriz

Generación y manejo de matrices especiales

Polinomios

Solución de sistemas lineales

Gráficos

ÍNDICE DE FIGURAS

En el libro de *Matlab básico* se exponen las principales características de Matlab, así como también la forma correcta de manejar, almacenar y recuperar variables. A crear vectores y matrices, las operaciones y funciones que se pueden aplicar a cada uno de ellos. Cómo se deben ingresar ecuaciones, polinomios y sistemas de ecuaciones, de manera que se resuelvan rápidamente. Además de ello se presentan las principales funciones para realizar gráficas en dos dimensiones, manejo de gráficas y subgráficas. En cada uno de los capítulos se resuelven ejercicios relacionados a los temas explicados.

Alexandra Orfelina Pazmiño Armijos, ingeniera electrónica en Computación y tecnóloga en Informática Aplicada, especialista en redes de comunicación de datos, magíster en Informática Empresarial, docente facultad de Mecánica de la Escuela Superior Politécnica de Chimborazo, miembro del grupo de Investigación y Estudios en Bioingeniería.

Jairo René Jácome Tinoco, ingeniero electrónico en Computación, tecnólogo en Informática Aplicada, magíster en Sistemas de Telecomunicaciones, técnico docente de la Facultad de Mecánica de la Escuela Superior Politécnica de Chimborazo, miembro del Grupo de Investigación y Estudios en Bioingeniería.

Luis Alberto Zabala Aguiar, ingeniero en Electrónica, Control y Redes Industriales, magíster en Sistemas de Telecomunicaciones, ha desarrollado proyectos de control industrial y en el sector petrolero, docente en las asignaturas Sistemas de Control, Instrumentación y sensores y Electrónica de potencia.

Gavilanes C. Javier J. ingeniero en Electrónica, Control Automático y Redes Industriales. máster universitario en Automática y Robótica. Adjudicatario Beca de Estudios Cuarto Nivel. Docente de la Facultad de Mecánica Escuela de Ingeniería Automotriz. Miembro del Grupo de Investigación y Estudios en Bioingeniería.

CAPÍTULO I

MATLAB es un lenguaje de programación desarrollado por MathWorks.Su nombre proviene de la abreviatura Matrix Laboratory que significa laboratorio de matrices.

MATLAB no siempre es la mejor herramienta para usar en tareas de programación como desarrollo de sistemas, manejo de bases de datos, etc. Este programa se destaca en análisis numérico, cálculo matricial, procesamiento de señales, gráficos, etc., es un programa similar a cualquier lenguaje de alto nivel, óptimo en el manejo de matrices; por lo tanto, si un problema se resuelve con una solución matricial, este lo ejecuta mucho más rápido.

1.1 AMBIENTE DE MATLAB

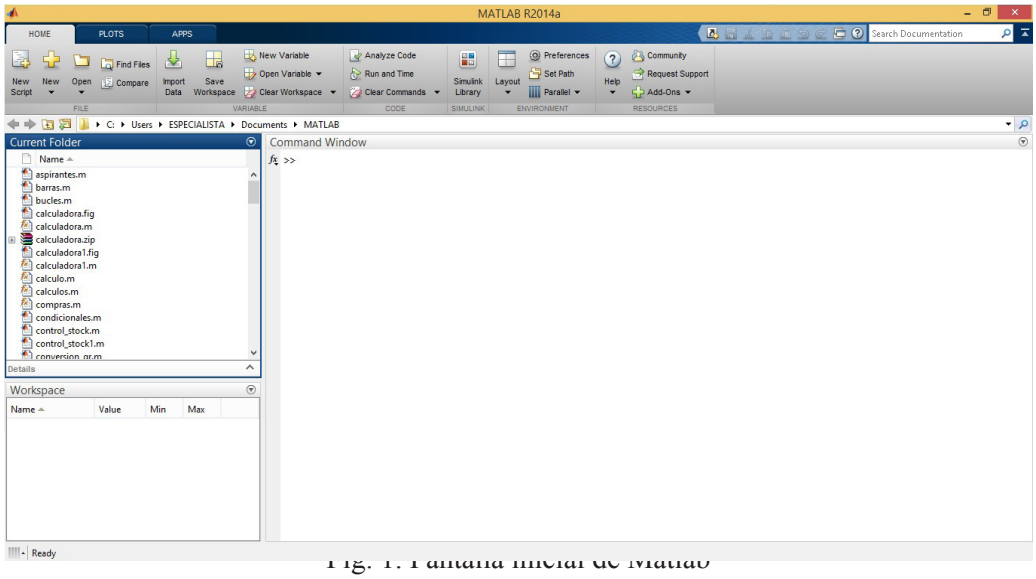


Fig. 1. Pantalla inicial de Matlab

Matlab está formado por varias ventanas, de las cuales algunas se encuentran visibles. A continuación, se menciona el funcionamiento de cada una.

1.1.1 *Command window*

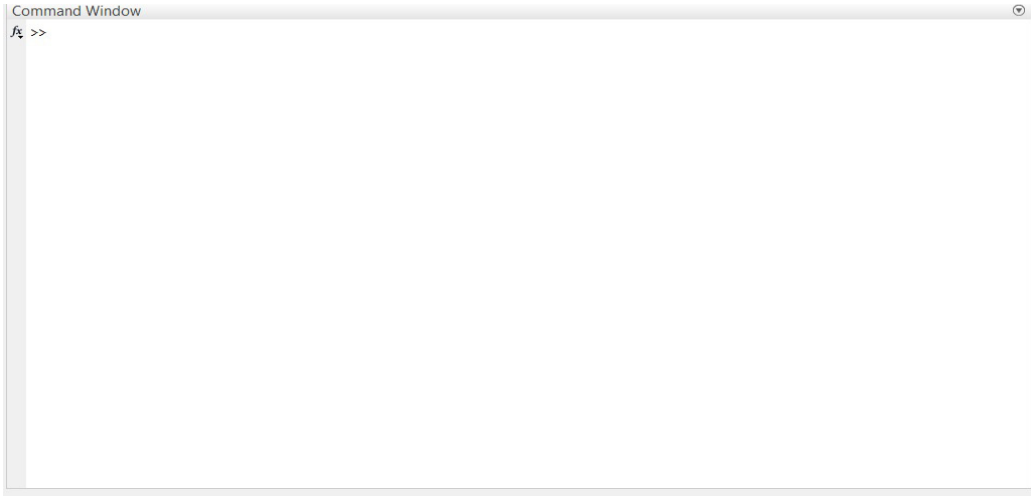


Fig. 2. Command windows (ventana de comandos)

La ventana de comandos (*command window*) se encuentra en el centro de la ventana de Matlab; en esta se ejecutan las instrucciones y nos da el resultado a continuación; es la ventana más importante del entorno de Matlab.

En esta ventana, se visualiza el *prompt* de inicio de Matlab, el cual indica que se espera una instrucción. Si este no aparece quiere decir que el programa se encuentra esperando o realizando alguna tarea. Se puede detener la ejecución de cualquier instrucción con la combinación de las teclas Ctrl + C.

1.1.2 *Workspace*

En la parte inferior de la pantalla de Matlab, a la izquierda, aparece la ventana de espacio de trabajo (*workspace*), en la que se almacena la información de todas las variables que se crean en la ventana de comandos.

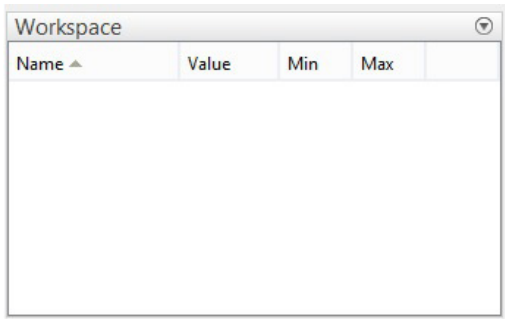
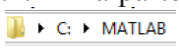


Fig. 3. Workspace (espacio de trabajo)

1.1.3 *Current Folder*

A la izquierda, en la parte superior, está la ventana carpeta actual (*current folder*), en este lugar,se almacenan todos los archivos, y es desde aquí que se ejecutan. En la parte superior de esta ventana aparece la dirección de la carpeta actual 

la cual se puede cambiar dando clic en el icono  , subir un nivel entre las carpetas  o navegar entre las carpetas 

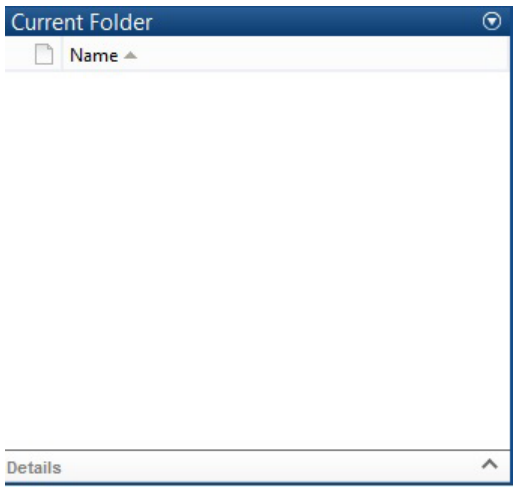


Fig. 4. *Current folder* (carpeta actual)

1.1.4 *Command history*

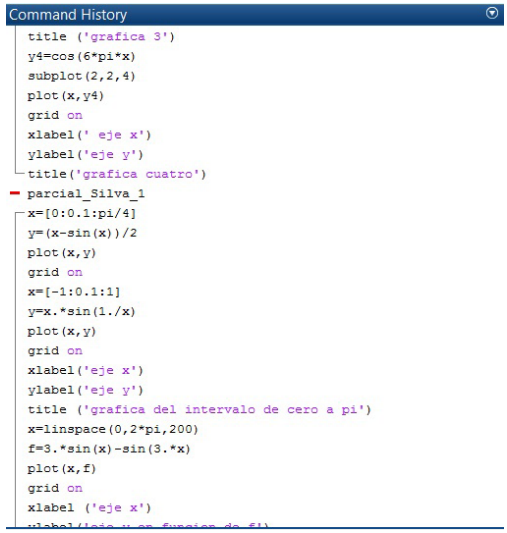


Fig. 5. *Command history* (historia de comandos)

Las instrucciones que se han ingresado en la ventana de comandos se almacenan en la ventana de historia de comandos (*command history*).Esta ventana puede o no estar visible en Matlab. Desde aquí se puede hacer uso de los comandos ejecutados anteriormente dando clic sobre ellos.

Con las flechas del cursor (↑ ↓), se pueden recuperar las órdenes anteriores, sin volver a escribirlas. Las flechas (← →) permiten presentar el desplazamiento horizontal en la línea de comandos. Estas flechas son de mucha utilidad en el caso de una equivocación o cuando se quiere volver a ejecutar un mismo comando o hacerle una pequeña modificación.

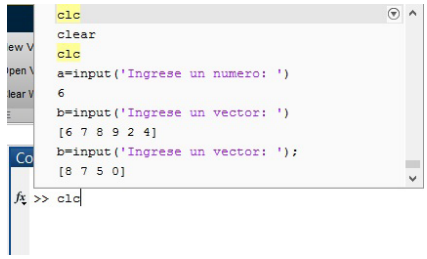


Fig. 6. *Pop up* historia de comandos

1.1.5 Editor

El editor es la ventana donde se escribirán todos los programas (*scripts*) que se ejecutarán en Matlab.

Fig. 7. Editor de Matlab

1.1.6 Ventana de gráficos

En esta ventana, se visualizarán todos los gráficos de Matlab en 2D y 3D

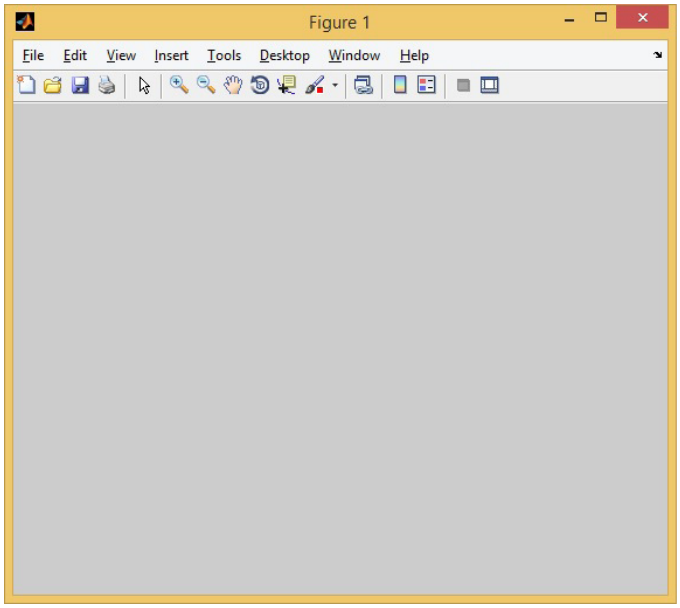


Fig. 8. Ventana de graficación

1.1.7 Ayuda de Matlab

Se puede consultar una función determinada o información al digitar en la ventana de comandos *help* <comando a consultar>, o simplemente *help* y se abre la ventana de ayuda o con la tecla F1. Una vez abierta, se busca por contenidos, palabras concretas, y otros. Cada vez que se visualiza la ayuda de un determinado comando se encontrará más comandos relacionados.

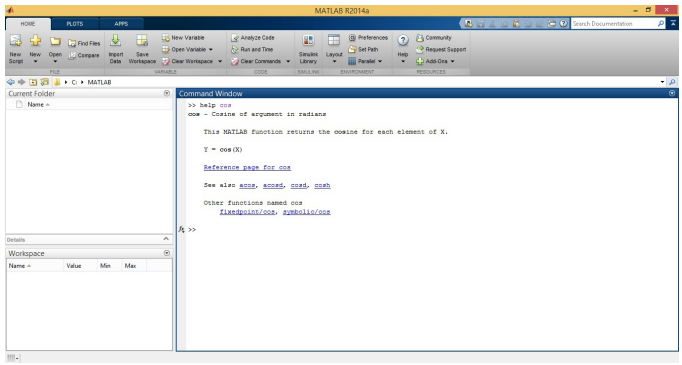


Fig. 9. Ayuda de Matlab

1.2 Trabajo en Matlab

Para comenzar a trabajar con Matlab, se tecleará la orden que se desee ejecutar en la ventana de comandos después del símbolo del sistema >>. Matlab ejecuta la instrucción y almacena el resultado en la variable indicada o crea la variable *ans* por defecto.

Las funciones introducidas, llamadas también "entradas" se ejecutan pulsando la tecla *enter*. Se debe tener en cuenta que al escribir los nombres de las funciones o de los comandos, Matlab distingue entre mayúsculas y minúsculas (por lo general, las funciones se escriben en minúsculas).

Si se ingresa una operación sin indicar en donde se va a almacenar el resultado; Matlab crea y guarda la variable *ans* con el resultado de la operación.

A continuación un ejemplo en el que se pide a Matlab que ejecute una operación sencilla:

```
« 9+3
ans  =
  13
```

Como se puede observar a continuación, en la ventana del *workspace* ,se crea esta variable con su contenido.

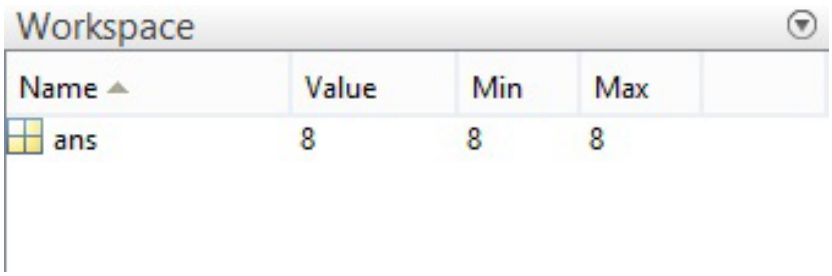


Fig. 10. Variable ans en el *workspace*

Declaración de variables

No hace falta declarar variables en Matlab con algún tipo de comando o instrucción; simplemente se crea la variable con una asignación directa del valor. Una vez creada la variable se la puede utilizar posteriormente en los cálculos.

```
» a = 9
  a=
     9
» b = 3
  b=
     3
» c=a+b
  c=
    12
```

A continuación, una lista de comandos útiles en la línea de comandos:

- clc: Borra el contenido de la pantalla y coloca el cursor en la primera línea.
- Esc: Borra la línea.
- %: Todo lo que aparece detrás del símbolo % y en la misma línea se considera un comentario.
- [ctrl]+[c]: Detiene la ejecución de cualquier comando o función.

Los cálculos realizados en la ventana de comandos suelen ser largos y ocupan mucho espacio, por lo que a veces es necesario que no se muestren en la pantalla, se puede utilizar el operador; al final de la instrucción.

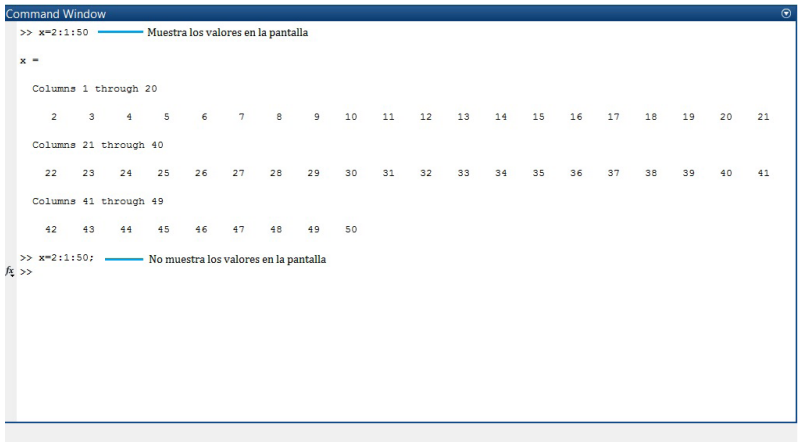


Fig. 11. Operador ; en la ventana de comandos

1.2.1 Números reales y números complejos

Reales: el programa trabaja con números racionales e irracionales positivos, negativos, e incluso con el cero.

Las operaciones habituales con números enteros se realizan de forma exacta, sin importar el tamaño que tenga el resultado. Se puede especificar el número de cifras que aparece en la pantalla.

Función vpa (variable de precisión aritmética): especifica el número de dígitos decimales de precisión de un valor.

Sintaxis:

```
vpa A
vpa A d
```

Ejemplo:

```
>> vpa 6^400

ans =

1.821797716821872825139468712408937126733897152817 5e311

>> vpa '6^400' 50

ans =

1.82179771682187282513946871240893712673389715281 75e311
```

Complejos: estos números están representados como la suma de un número real y un número imaginario.

Este campo está igualmente implementado en Matlab. Se representa con la letra minúscula i o j que representa el número complejo √(-1), sobre estos números se pueden aplicar los operadores habituales, además de algunas otras funciones específicas.

Ejemplo:

```
>> i

ans =

  0.0000 + 1.0000i

>> j

ans =

  0.0000 + 1.0000i

>> 5+7i

ans =

  5.0000 + 7.0000i

>> 9-8i

ans =

  9.0000 - 8.0000i

>> 3-5j

ans =

  3.0000 - 5.0000i
```

Se puede utilizar para representar el número complejo la i o la j indistintamente.

1.2.2 Operaciones aritméticas

En Matlab, para realizar operaciones aritméticas, se utilizan los siguientes operadores:

- Sumar: +
- Restar: -
- Producto: *
- División derecha: /
- División izquierda: \
- Potenciación: ^

```
>> 6+9

ans =

     15
```

Los enteros se imprimen sin punto decimal

```
>> 98-67

ans =

     31

>> 3+9-8+(-2)

ans =

     2

>> 3.5*8

ans =

     28

>> 4^3

ans =

     64
```

```
>> -2^4

ans =

  -16

>> 9/6

ans =

    1.5000

>> 9\6

ans =

    0.6667

>> 4/5^3

ans =

    0.0320
```

1.2.3 Formatos de despliegue de números

Para mostrar números punto flotante Matlab utiliza notación científica; es decir, expresa un valor como un número entre 1 y 10 multiplicado por una potencia de 10. En Matlab, se designan con una e entre el número decimal y el exponente.

```
>> 0.786e9

ans =

  786000000
```

No debe existir espacios en blanco entre el número decimal y el exponente.

Correcto

```
>> a=9.768e23

a =

  9.7680e+23
```

Incorrecto

```
>> b=9.768 e23
 b=9.768 e23
          |
Error: Unexpected MATLAB expression.
```

Ejemplo:

```
>> d=8.765e34

d =

  8.7650e+34

>> f=3.456e-23

f =

  3.4560e-23
```

Matlab usa, en sus cálculos, números punto flotante.De cuántos dígitos se usen ,depende de su cálculo. Los números enteros se muestra sin punto decimal, y los números decimales se muestran en formato corto (*format short*) que, por defecto, es cuatro dígitos decimales.

```
>> c=9.8

c =
        9.8000
```

Se pueden especificar otros formatos para mostrar más o menos dígitos decimales. Para cambiar entre un formato y otro se especifica la palabra *format* y. A continuación, el nombre del formato.

format long: Muestra en un formato decimal con 14 dígitos decimales.

```
>> format long
>> 8.763

ans =

  8.763000000000000
```

format long e: muestra los números en notación científica con 14 dígitos decimales.

```
>> format long e
>> 0.0000000003488

ans =

        3.488000000000000e-10
```

format short: muestra en formato decimal con 4 dígitos decimales.

```
>> format short
>> 8.763
ans =
        8.7630
```

format short e: muestra los números en notación científica con cuatro dígitos decimales.

```
>> format short e
>> 0.0000000003488

ans =

  3.4880e-10
```

format bank: (formato banco): se despliega dos dígitos decimales.

```
>> format bank
>> 7.8873873

ans =

        7.89
```

format +: muestra el signo más o menos solamente, sea de un número ingresado, de los elementos de un vector o de una matriz.

```
>> format +
>> v=[3 -2 5.4 3 -1 -3]

v =

+-++--
```

format rat: muestra los números ingresados en forma fraccionaria.

```
>> format rat
>> s=[0.45 0.78 2.5 3.8 -1.9]
s =

        9/20          39/50          5/2          19/5          -19/10
```

Ejemplo comando	Descripción	Ejemplo
format short	4 dígitos decimales	>> format short >> 787.98768 ans = 787.9877
format short e	4 dígitos decimales con notación científica	>> format short e >> 787.98768 ans = 7.8799e+02
format long	14 dígitos decimales	>> format long >> 787.98768 ans = 787.9876800000000e +02
format long e	14 dígitos decimales con notación científica	>> format long e >> 787.98768 ans = 787.9876800000000e +02
format bank	2 dígitos decimales	>> format bank >> 787.98768 ans = 787.99
format +	+ , -	>> format + >> 787.98768 ans = +
format rat	forma fraccionaria	>> format rat >> 787.98768 ans = 63827/81

1.2.4 Números especiales

Existe un grupo de números que, por su uso, tiene un trato especial.

Función /Definición	Comando en Matlab

Pi	`>> pi`
	`ans =`
	`3.1416`

Exp
Neper
	`>> exp(1)`
	`ans =`
	`2.7183`

inf
Infinito Ejemplo: 1/0
	`>> 1/0`
	`ans =`
	`Inf`

NaN
Indeterminación Ejemplo: 0/0
	`>> 0/0`
	`ans =`
	`NaN`

Realmin
Menor número real positivo utilizable
	`>> realmin`
	`ans =`
	`2.2251e-30`

Realmax
Mayor número real positivo utilizable
	`>> realmax`
	`ans =`
	`1.7977e+30`

1.2.5 Funciones con números enteros

Función rem: determina el residuo de la división entre dos números.

Sintaxis: rem(n,m)

Ejemplo:

```
>> rem(9,2)

ans =

     1
```

Función sign: determina el signo de un número.

Devuelve 1 si el número es mayor a cero

Devuelve 0 si el número es igual a cero

Devuelve -1 si el número es menor a cero

Sintaxis: sign(n)

Ejemplo:

```
>> sign(-9)

ans =

    -1

>> sign(67)

ans =

     1
```

```
>> sign(0)

ans =

     0
```

Función max: determina el valor máximo de dos números.

Sintaxis: max(n1,n2)

Ejemplo:

```
>> max (5,8)

ans =

     8
```

Función min: determina el valor mínimo de dos números.

Sintaxis: min(n1,n2)

Ejemplo:

```
>> min(6,9)

ans =

     6
```

Función gcd: determina el máximo común divisor de dos números.

Sintaxis: gcd(n1,n2)

Ejemplo:

```
>> gcd(25,35)
```

```
ans =

     5
```

Función lcm: determina el mínimo común múltiplo de dos números.

Sintaxis: lcm(n1,n2)

Ejemplo:

```
>> lcm(7,15)

ans =

  105
```

Función factorial: determina el factorial de un número.

Sintaxis: factorial(n)

Ejemplo:

```
>> factorial(5)

ans =

  120
```

Función factor: descompone un número en sus factores primos.

Sintaxis: factor(n)

Ejemplo:

```
>> factor(45)

ans =

     3     3     5
```

1.2.6 Funciones con argumento real y complejo

Funciones trigonométricas

Función sin y asin: determina el seno y la inversa del seno correspondientemente de un argumento (real o complejo) en radianes.

Sintaxis: sin(x) sin(z)

Sintaxis: asin(x) asin(z)

Ejemplo:

```
>> sin(34.5)

ans =

        0.0575

>> asin(4.53)

ans =

  1.5708 - 2.1915i     >> sin(5-i)

ans =

 -1.4797 - 0.3334i

>> asin(2+3i)

ans =

  0.5707 + 1.9834i
```

Función cos y acos: determina el coseno y la inversa del coseno correspondientemente de un argumento (real o complejo) en radianes.

Sintaxis: cos(x) cos(z)

Sintaxis: acos(x) acos(z)

Ejemplo:

```
>> cos(90)

ans =

  -0.4481

>> acos(90)

ans =

  0.0000 + 5.1929i   >> cos(3-5i)

ans =

 -73.4673 +10.4716i

>> acos(4-9i)

ans =

  1.1545 + 2.9822i
```

Función tan y atan: determina la tangente y la inversa de la tangente correspondientemente de un argumento (real o complejo) en radianes.

Sintaxis: tan(x) tan(z)

Sintaxis: atan(x) atan(z)

Ejemplo:

```
>> tan(60)

ans =

        0.3200

>> atan(45)

ans =

        1.5486>> tan(9+5i)

ans =

 -0.0001 + 0.9999i

>> atan(7-8i)

ans =

  1.5086 - 0.0706i
```

Función cot y acot: determina el cotangente y la inversa del cotangente correspondientemente de un argumento (real o complejo) en radianes.

Sintaxis: cot(x) cot(z)

Sintaxis: acot(x) acot(z)

Ejemplo:

```
>> cot(34.5)

ans =

      -17.3663
```

```
>> acot(9.45)

ans =

        0.1054>> cot(5+8i)

ans =

 -0.0000 - 1.0000i

>> acot(6-8i)

ans =

  0.0603 + 0.0799i
```

Función sec y asec: determina la secante y la inversa de la secante correspondientemente de un argumento (real o complejo) en radianes.

Sintaxis: sec(x) sec(z)

Sintaxis: asec(x) asec(z)

Ejemplo:

```
>> sec(56)

ans =

   1.1720

>> asec(8.1)

ans =

   1.4470        >> sec(5-i)

ans =
```

```
  0.2995 + 0.7710i

>> asec(4-7i)

ans =

  1.5096 - 0.1077i
```

Función csc y acsc: determina la cosecante y la inversa de la cosecante correspondientemente de un argumento (real o complejo) en radianes.

Sintaxis: csc(x) csc(z)

Sintaxis: acsc(x) acsc(z)

Ejemplo:

```
>> csc(67)

ans =

  -1.1689

>> acsc(92)

ans =

   0.0109        >> csc(4-7i)

ans =

 -0.0014 - 0.0012i

>> acsc(4+9i)

ans =

  0.0411 - 0.0927i
```

Funciones logarítmicas y exponenciales

Función exp: devuelve el exponencial (e^x) de un argumento (real o complejo).
Sintaxis: exp(x) exp(z)

Ejemplo:

```
>> exp(3)
ans =
  20.0855
>> exp(4-5i)
ans =
15.4874 +52.3555i
```

Función log: devuelve el logaritmo natural de un argumento (real o complejo).
Sintaxis: log(x) log(z)

Ejemplo:

```
>> log(5)
ans =
   1.6094
>> log(4-6i)
ans =
  1.9756 - 0.9828i
```

Función log10: devuelve el logaritmo en base 10 de un argumento (real o complejo)

Sintaxis: log10(x) log10(z)

Ejemplo:

```
>> log10(3)

ans =

   0.4771

>> log10(5+4i)

ans =

   0.8064 + 0.2930i
```

Función log2: devuelve el logaritmo en base 2 de un argumento (real o complejo)

Sintaxis: log2(x) log2(z)

Ejemplo:

```
>> log2(86)

ans =

   6.4263

>> log2(5-7i)

ans =

   3.1047 - 1.3713i
```

Función sqrt: devuelve la raíz cuadrada de un argumento (real o complejo).

Sintaxis: sqrt(x) sqrt(z)

Ejemplo:

```
>> sqrt(36)
ans =
   6
>> sqrt(4+16i)
ans =
  3.2010 + 2.4992i
```

1.2.7 Funciones específicas de números reales

Función abs: devuelve el valor absoluto de un argumento.

Sintaxis: abs(x)

Ejemplo:

```
>> abs(-98.6)
ans =
  98.6000
```

Función floor: redondea un número al entero más cercano hacia el infinito negativo.

Sintaxis: floor(x)

Ejemplo:

```
>> floor(9.8)

ans =
```

Función ceil: redondea un número al entero más cercano hacia el infinito positivo.

Sintaxis: ceil(x)

Ejemplo:

```
>> ceil(9.8)

ans =

   10
```

Función round: redondea un número al entero más cercano.

Sintaxis: round(x)

Ejemplo:

```
>> round(8.4)

ans =

    8

>> round(8.9)
ans
9
```

Función fix: redondea o trunca un número al entero más cercano hacia cero.

Sintaxis: fix(x)

Ejemplo:

```
>> fix(9.4)

ans =

   9

>> fix(9.9)

ans =

   9
```

1.2.8 Funciones específicas para la parte real e imaginaria

Función floor: redondea la parte real y la parte imaginaria al entero más cercano hacia el infinito negativo.

Sintaxis: floor(z)

Ejemplo:

```
>> floor(7.8+3.6i)

ans =
7.0000 + 3.0000i
```

Función ceil: redondea la parte real y la parte imaginaria al entero más cercano hacia el infinito positivo.

Sintaxis: ceil(z)

Ejemplo:

```
>> ceil(4.2-5.4i)

ans =

  5.0000 - 5.0000i
```

Función round: redondea la parte real y la parte imaginaria al entero más cercano.

Sintaxis: round(z)

Ejemplo:

```
>> round(9.2-5.7i)

ans =

  9.0000 - 6.0000i
```

Función fix: redondea o trunca la parte real y la parte imaginaria al entero más cercano hacia cero.

Sintaxis: fix(z)

Ejemplo:

```
>> fix(3.4+7.1i)
```

```
ans =

  3.0000 + 7.0000i
```

Función abs: devuelve el módulo de un número complejo.
Sintaxis: abs(z)

Ejemplo:

```
>> abs(4-7i)

ans =

  8.0623
```

Función angle: devuelve el ángulo de un número complejo.
Sintaxis: angle(z)

Ejemplo:

```
>> angle(8-6i)

ans =

  -0.6435
```

Función conj: devuelve la conjugada de un número complejo.
Sintaxis: conj(z)

Ejemplo:

```
>> conj(3-7i)
ans =
  3.0000 + 7.0000i
```

Función real: devuelve la parte real de un número complejo.

Sintaxis: real (z)

Ejemplo:

```
>> real(67-8.9i)
ans =
   67
```

Función imag: devuelve la parte imaginaria de un número complejo.

Sintaxis: imag(z)

Ejemplo:

```
>> imag(67-8.9i)
ans =
  -8.9000
```

1.2.9 Manejo de variables

1.2.9.1 Guardar variables

Cuando se guardan variables del área de trabajo (*Workspace*) se almacena en un archivo binario con extensión .mat, solo las variables, no las instrucciones que se ejecutaron para crear las variables.

Para guardar variables se puede utilizar el icono 

del menú Home e ingresar el nombre del archivo en la ventana del explorador.

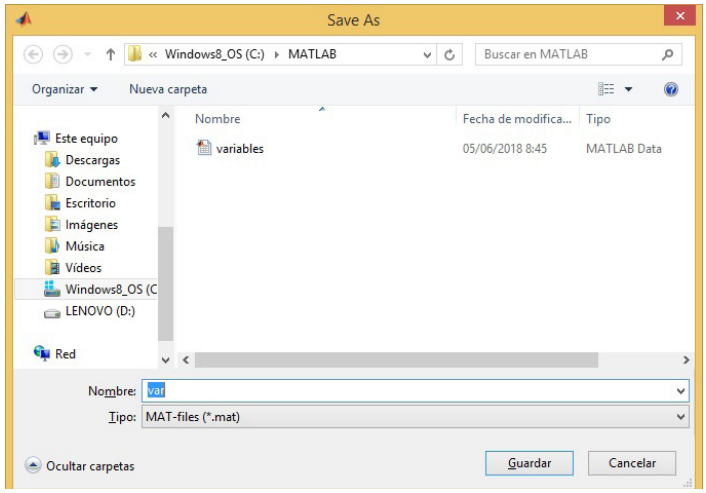


Fig. 12. Guardar variables

También se pueden almacenar las variables mediante la ventana de comandos con la instrucción save <nombre archivo>

Ejemplo:

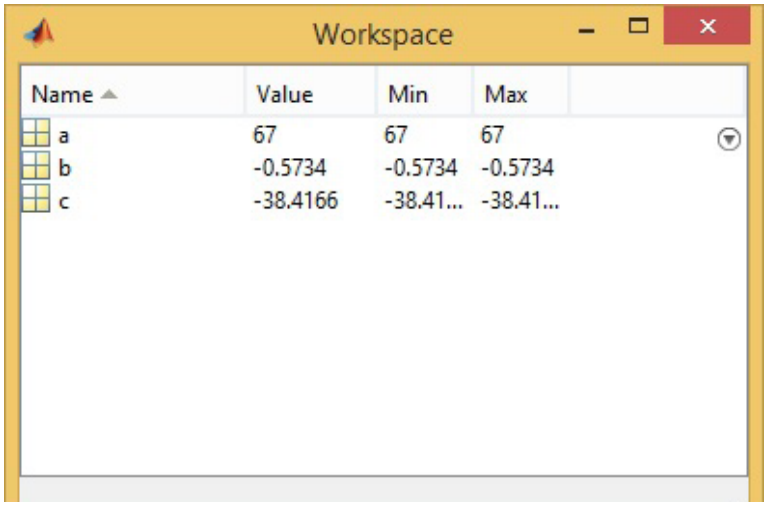


Fig. 13. Ventana del *workspace*

Guarda todas las variables creadas en el *workspace* en un archivo llamado variable en el directorio actual.

También se pueden almacenar variables individuales o listas de variables en el directorio actual con el comando save <nombre archivo><lista de variables>

Ejemplo:

```
>> save variable
```

Guarda todas las variables creadas en el *workspace* en un archivo llamado variable en el directorio actual.

También se pueden almacenar variables individuales o listas de variables en el directorio actual con el comando

save <nombre archivo><lista de variables>

Ejemplo:

```
>> save variables a b
```

1.2.9.2 Cargar variables

Para restaurar el área de trabajo o *workspace*, se pulsa el icono 

y se busca el archivo en la ubicación almacenada.

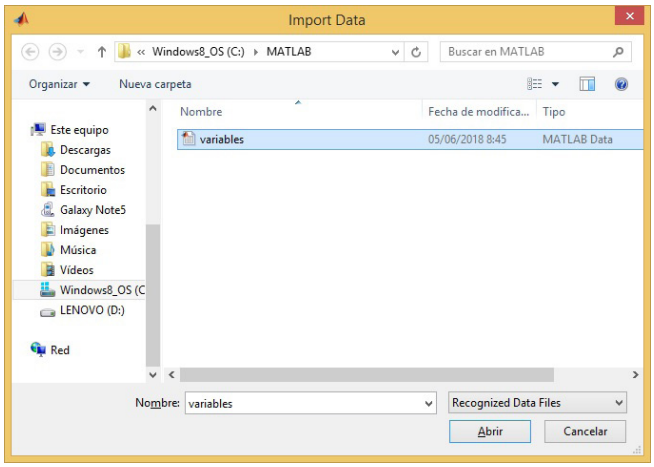


Fig. 14. Restaurar variables en el Workspace

Si se lo hace mediante la ventana de comandos se escribe el comando load <nombre del archivo>

```
>> load variables
```

1.2.10 Archivos *m-script*

Se pueden crear, en Matlab *scripts* que contengan el código que se vaya a ejecutar; al guardarlo, se crean archivos con la extensión .m

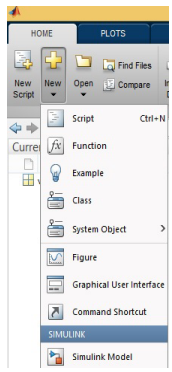


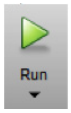
Fig. 15. Crear un nuevo archivo *script*

Al guardar un *script*, este se almacena en la carpeta actual, tomando en cuenta que para nombrar el archivo, se debe considerar lo siguiente:

- Un nombre que empiece con una letra.
- Solo puede contener números, letras y el guion bajo
- No se permiten espacios en blanco

1.2.10.1 Ejecución del *script*

Para ejecutar un *script* creado en el editor, puede hacerse desde del icono



o se puede escribir el nombre del archivo en la ventana de comandos.

1.2.10.2 Comentarios en un *script*

Cuando se tiene un código extenso que forma parte de un *script*, es necesario muchas veces utilizar comentarios que permiten describir ciertas instrucciones; para ello se utiliza el signo de porcentaje. Al ejecutar el *script,* estas líneas que contienen este signo no se ejecutan ni devuelve como error.

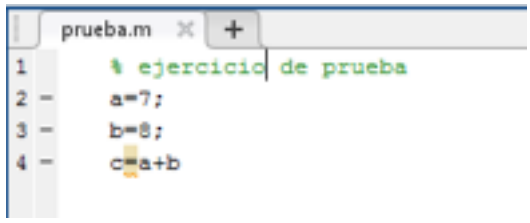


Fig. 16. Ventana del editor con script

1.2.11 Ejercicios resueltos

Ingresar en Matlab los siguientes ejercicios:

1. 5+3
9-1

```
>> (5+3)/(9-1)
ans =
   1
```

Se debe utilizar paréntesis para agrupar términos y que la división se realice correctamente.

2. $2^3 - \frac{4}{5+3}$

```
>> (2^3)-(4/(5+3))

ans =

   7.5000
```

Se utiliza paréntesis para la potenciación y la suma del dividendo; de esta manera evitamos errores en la jerarquía de las operaciones.

3. $\frac{5^2+1}{4-1}$

```
>> (5^(2+1))/(4-1)

ans =

  41.6667
```

El exponente de la potencia se debe colocar entre paréntesis y a la vez agrupar el dividendo y el divisor utilizando paréntesis.

4. $4\frac{1}{2} * 5\frac{2}{3}$

```
>> format rat
>> (4+1/2)*(5+2/3)

ans =
51/2
```

Para ingresar números enteros con parte fraccionaria utilizamos el signo más y agrupamos lo términos. Además, en el ejercicio, utilizamos el comando format para visualizar el resultado en forma de fracción.

5. $\dfrac{5 + 6 * \frac{7}{3} - 2^2}{\frac{2}{3} * \frac{3}{3x6}}$

```
>> (5+6*(7/3)-(2^2)/((2/3)*(3/(3*6))))

ans =

-17
```

Como se puede observar, se debe utilizar el número de paréntesis que sea necesario para agrupar los términos de manera correcta.

6. $\sqrt{\sqrt{144} - \sqrt{(1/16)^{-1/4}}}$

```
>> sqrt(sqrt(144)-sqrt((1/16)^(-1/4)))

ans =

  2271/698
```

Se puede anidar el uso de funciones como es el caso de la raíz cuadrada usando correctamente signos de agrupación.

7. $\frac{(5-i)^2}{3i} + \sqrt{2-i}$

```
>> (((5-i)^2)/3i)+sqrt(2-i)

ans =

  -2201/1172  - 1360/163i
```

En el caso de los números complejos también se utiliza paréntesis para agrupar los términos y que se realice correctamente la potencia y las divisiones de este ejercicio.

8. (MCD (8^2, 45)) !

```
>> factorial(gcd(5^2,45))

ans =

   120
```

En el ejercicio, se debe tomar en cuenta que operaciones se realiza primero, como, en este caso, el máximo común divisor y posteriormente el factorial de la respuesta anterior.

9. $\frac{tg\left(\frac{\pi}{3}\right)cos(\frac{\pi}{4})}{Min(\pi,e)}$

```
>> (tan(pi/3)*cos(pi/4))/min(pi,exp(1))

ans =

   565/1254
```

En este ejercicio, se utiliza valores definidos en Matlab como el valor de pi y el neper. Como se puede observar, se los usa directamente dentro de las otras funciones como el mínimo, coseno, tangente, etc.

10. $$(imag(3+2i))^{2i} - \frac{sen(6i)}{cos(2-4i)}$$

```
>> ((imag(3+2i))^2i)-((sin(6i)/cos(2-4i)))

ans =

  -3536/541   + 1750/431i
```

Se utilizan signos de agrupación para los números imaginarios y a la vez para la potenciación y las otras funciones trigonométricas utilizadas en el ejercicio.

1.3 VECTORES

Todas las variables en MATLAB son arreglos (*arrays*), lo cual significa que cada variable puede contener múltiples elementos. Un número solo, llamado escalar, es de hecho un arreglo de 1x1, lo que significando que contiene una fila y una columna.

1.3.1 Definir vectores

Para definir un vector no hace falta definir su tamaño (de hecho, puede cambiar si es necesario ingresando más elementos o eliminando elementos). Simplemente, se ingresan los valores de los elementos que van a componer el vector

entre corchetes, separados por espacios o una coma, en el caso de vectores fila, o por el carácter punto y coma (;) , en el caso de vectores columna.

Ejemplo de vectores fila

```
>> c=[2 3 4 5 6]

c =

    2     3     4     5     6

>> d=[1,2,3,4,5]

d =

    1     2     3     4     5
```

Ejemplo de vector columna

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

a =

    1
    2
    3
    4
```

Se utiliza un índice que es un número entero para acceder a los elementos de un vector. El índice se inicializa en uno para identificar desde el primer elemento del vector.

```
>> v=[5;3;7;1;9;4]
v =

     5
     3
     7
     1
     9
     4

>> v(3)

ans =

     7
```

1.3.2 Generación rápida de vectores

- Operador (:)

Para vectores largos, ingresar los números uno por uno no es práctico, un método abreviado para crear vectores uniformemente espaciados es el uso del operador : (dos puntos) y especificar solamente los puntos inicial y final.

Se pueden generar vectores en los que no necesariamente se debe escribir cada uno de los elementos del vector

variable = [vini : vfin] Para definir el vector se pueden utilizar corchetes, paréntesis o no utilizarlos. El primer y último elemento se definen en vini y vfin respectivamente, separados los componentes entre el primer y último elemento de uno en uno.

Ejemplo:

```
>> r=1:10

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

variable = [vin : incr : vfin] El operador : usa un intervalo o incremento predeterminado de 1; sin embargo se puede especificar un propio incremento. Define el vector cuyo primer y último elemento son los especificados por vin y vfin, estando los componentes intermedios separados por incr. Está permitido no utilizar los corchetes o sustituirlos por paréntesis.

Ejemplo:

```
>> v=1:2:10

v =

   1    3    5    7    9
```

- Función linspace

variable = linspace (x1,x2,n) Genera un vector siendo el primer elemento x1 y el último elemento x2 y n el número de elementos del vector. Si se conoce el número de elementos que debe tener el vector en lugar del intervalo entre cada elemento se utiliza la función linspace.

Ejemplo:

```
>> p=linspace(2,20,7)

p =

   2    5    8   11   14   17   20
```

Tanto linspace como el operador : crean vectores fila. Sin embargo, se puede convertir un vector fila en un vector columna usando el operador de transposición (').

Ejemplo:

```
>> x=linspace(5,40,5)

x =

   5.0000   13.7500   22.5000   31.2500   40.0000

>> x=x'

x =

   5.0000
  13.7500
  22.5000
  31.2500
  40.0000
```

1.3.3 Seleccionar un elemento o un subconjunto de elementos

- x(n): selecciona el elemento n-ésimo del vector.

Ejemplo:

```
>> x=linspace(3,25,8)

x =

 3.0000   6.1429   9.2857  12.4286  15.5714  18.7143  21.8571  25.0000

>> x(4)
```

```
ans =
        12.4286
```

► x(a:b): selecciona del vector los elementos que se encuentran entre la posición a y la posición b.

Ejemplo:

```
>> x(2:5)

ans =

   6.1429   9.2857   12.4286   15.5714
```

► x(a:in:b): selecciona del vector los elementos que se encuentran entre la posición a y la posición b con el incremento o decremento indicado en la variable in.

Ejemplo:

```
>> x(3:2:8)

ans =

   9.2857   15.5714   21.8571

>> x(7:-3:1)

ans =

  21.8571   12.4286   3.0000
```

► x([a b c]): Selecciona del vector los elementos que se encuentran en las posiciones a , b, c, etc.

Ejemplo:

```
>> x([1 4 7 8])
ans =
   3.0000   12.4286   21.8571   25.0000
```

1.3.4 Operaciones con vectores

Se pueden realizar operaciones entre un vector, un valor escalar y entre vectores.

Operaciones entre un vector y un valor escalar

x+n: suma de cada elemento del vector x con un valor escalar n o viceversa.

Ejemplo:

```
>> p=1:3:15
p =
    1    4    7   10   13
>> p+5
ans =
    6    9   12   15   18
```

- x-n: resta de cada elemento del vector x con un valor escalar o viceversa.

Ejemplo:

```
>> p
p =
   1    4    7   10   13
>> 9-p
ans =
   8    5    2   -1   -4
```

► x*n: multiplica cada elemento del vector x por un valor escalar o viceversa.

Ejemplo:

```
>> x
x =
  3.0000   6.1429   9.2857  12.4286  15.5714  18.7143  21.8571  25.0000
>> x*4
ans =
 12.0000  24.5714  37.1429  49.7143  62.2857  74.8571  87.4286 100.0000
```

► x/n: divide cada elemento del vector c para un valor escalar n o viceversa.

Ejemplo:

```
>> x

x =

  3.0000   6.1429   9.2857  12.4286  15.5714  18.7143  21.8571  25.0000

>> x/5

ans =

  0.6000   1.2286   1.8571   2.4857   3.1143   3.7429   4.3714   5.0000
```

Operaciones entre vectores (elemento a elemento)

► x+y: suma cada elemento del vector x con cada elemento del vector y, es decir posición a posición.

Ejemplo:

```
>> x=1:2:10

x =

   1   3   5   7   9

>> y=3:3:15

y =

   3   6   9  12  15

>> x+y

ans =
   4   9  14  19  24
```

► x-y: resta cada elemento del vector x con cada elemento de vector y, es decir posición a posición.

Ejemplo:

```
>> x

x =

   1    3    5    7    9

>> y

y =

   3    6    9   12   15

>> y-x

ans =

   2    3    4    5    6
```

► x.*y: multiplica cada elemento del vector x con cada elemento de vector y, es decir posición a posición. Para este tipo de operación se debe utilizar el operador punto (.) que permite realizar la multiplicación elemento a elemento.

Si no se utiliza el operador punto (.) los vectores deben cumplir la condición para multiplicación de matrices; es decir, el número de columnas de la primera matriz debe ser igual al número de filas de la segunda matriz.

Ejemplo:

```
>> x

x =

   1    3    5    7    9

>> y

y =

   3    6    9   12   15

>> x*y
Error using  *
Inner matrix dimensions must agree.
```

Si no se utiliza el operador punto (.) nos da error, debido a que no se cumple la condición para multiplicar matrices.

```
>> x.*y

ans =

   3   18   45   84  135
```

- x./y: divide cada elemento del vector x con cada elemento de vector y, es decir posición a posición. Para este tipo de operación se debe utilizar el operador punto (.) que permite realizar la división elemento a elemento.

Ejemplo:

```
>> x

x =
```

```
1    3    5    7    9

>> y

y =

    3    6    9   12   15

>> x./y

ans =

   0.3333   0.5000   0.5556   0.5833   0.6000
```

► x.\y: divide cada elemento del vector y con cada elemento de vector x, es decir posición a posición. Para este tipo de operación se debe utilizar el operador punto (.), que permite realizar la división elemento a elemento.

Ejemplo:

```
>> x

x =

    1    3    5    7    9

>> y

y =

    3    6    9   12   15

>> x.\y
ans =

   3.0000   2.0000   1.8000   1.7143   1.6667
```

► x.^y: eleva cada elemento del vector x con cada elemento del vector y, es decir posición a posición. Para este tipo de operación se debe utilizar el operador punto (.) que permite realizar la potenciación elemento a elemento.

Ejemplo:

```
>> x=1:2:10

x =

   1    3    5    7    9

>> y=linspace(1,5,5)

y =

   1    2    3    4    5

>> x.^y

ans =

        1        9      125     2401    59049
```

1.3.5 Funciones elementales que admiten como argumento un vector real o complejo

► Función max: mayor elemento de un vector, para números complejos max(abs(v)) devuelve el modulo del número complejo.

Sintaxis: max(v)

max(abs(v))

Ejemplo:

```
>> v=[9 6 8 2 4 1 3]
v =
     9     6     8     2     4     1     3
>> max(v)
ans =
     9
>> z=[4+5i 9-6i 3-3i 2-9i]
z =
  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i
>> max(abs(z))
ans =
  10.8167
```

► Función min: menor elemento de un vector, para números complejos min(abs(v)) devuelve el modulo del número complejo.

Sintaxis: min(v)
min(abs(v))

Ejemplo:

```
>> v
v =
```

```
     9     6     8     2     4     1     3

>> min(v)

ans =

     1

>> z
z =

  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i

>> min(abs(z))

ans =

    4.2426
```

- Función mean: promedio de los elementos de un vector.

Sintaxis: mean(v)

Ejemplo:

```
>> v

v =

     9     6     8     2     4     1     3

>> mean(v)

ans =

    4.7143
```

```
>> z
z =
  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i
>> mean(z)
ans =
  4.5000 - 3.2500i
```

- Función median: mediana de los elementos de un vector.

Sintaxis: median(v)

Ejemplo:

```
>> v
v =
     9     6     8     2     4     1     3
>> median(v)
ans =
     4
>> z
z =
  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i
>> median(z)
```

```
ans =

  3.0000 - 2.0000i
```

► Función std: desviación típica de los elementos de un vector.

Sintaxis: std(v)

Ejemplo:

```
>> v

v =

     9     6     8     2     4     1     3

>> std(v)

ans =

    3.0394

>> z

z =

  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i

>> std(z)

ans =

    6.7762
```

► Función sort: ordena en forma ascendente los elementos de un vector.

Para números complejos, ordena según los valores absolutos.

Sintaxis: sort(v)

Ejemplo:

```
>> v
v =
     9     6     8     2     4     1     3
>> sort(v)
ans =
     1     2     3     4     6     8     9
>> z
z =
  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i
>> sort(z)
ans =
  3.0000 - 3.0000i   4.0000 + 5.0000i   2.0000 - 9.0000i   9.0000 - 6.0000i
```

- Función sum: suma los elementos de un vector.

Sintaxis: sum(v)

Ejemplo:

```
>> v
v =
     9     6     8     2     4     1     3
>> sum(v)
ans =
    33
>> z
z =
  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i
>> sum(z)
ans =
 18.0000 -13.0000i
```

- Función prod: multiplica los elementos de un vector.

Sintaxis: prod(v)

Ejemplo:

```
>> v
v =
```

```
     9     6     8     2     4     1     3
>> prod(v)
ans =
     10368
>> z
z =
  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i
>> prod(z)
ans =
 -6.9300e+02 - 2.6190e+03i
```

- Función cumsum: devuelve el vector de sumas acumuladas.

Sintaxis: cumsum(v)

Ejemplo:

```
>> v
v =
     9     6     8     2     4     1     3
>> cumsum(v)
```

```
ans =

     9    15    23    25    29    30    33

>> z

z =

  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i

>> cumsum(z)

ans =

  4.0000 + 5.0000i  13.0000 - 1.0000i  16.0000 - 4.0000i  18.0000 -13.0000i
```

- Función cumprod: devuelve el vector de productos acumulados.

Sintaxis: cumprod(v)

Ejemplo:

```
>> v

v =

     9     6     8     2     4     1     3

>> cumprod(v)

ans =

     9    54   432   864  3456  3456 10368

>> z
```

```
z =

  4.0000 + 5.0000i   9.0000 - 6.0000i   3.0000 - 3.0000i   2.0000 - 9.0000i

>> cumprod(z)

ans =

  1.0e+03 *

  0.0040 + 0.0050i   0.0660 + 0.0210i   0.2610 - 0.1350i  -0.6930 - 2.6190i
```

1.3.6 Ejercicios resueltos de vectores

1. Crear un vector fila llamado x que contenga los valores 4, 6 y 8, en ese orden.

```
>> x=[4  6  8]

x =

     4     6     8
```

2. Crear el vector fila llamado x con valores 4, 6 y 8, pero esta vez usando el operador :

```
>> x=4:2:8

x =

     4     6     8
```

3. Crear un vector fila llamado x que comienza en 2, termina en 6, y cada elemento está separado por 0,7.

```
>> x=2:0.7:6

x =

    2.00002.70003.40004.10004.80005.5000
```

4. Crear un vector c compuesto de 30 elementos espaciados en 4 unidades con elemento inicial el 6.

```
>> c=6:4:122

c =

 Columns 1 through 21

     6    10    14    18    22    26    30    34    38    42
46   50    54    58    62    66    70    74    78    82    86

 Columns 22 through 30

    90    94    98   102   106   110   114   118   122
```

5. Crear un vector fila llamado x que comienza en 5, termina en 50, y contiene 6 elementos.

```
>> x=linspace(5,50,6)

x =

     5    14    23    32    41    50
```

6. En un solo comando, crea un vector columna llamado x que comienza en 3, termina en 12 y tiene elementos que están espaciados por 4.

```
>> x=[3:4:12]'

x =

        3
        7
        11
```

Se utiliza el apóstrofe (‘) para obtener la matriz transpuesta de la matriz obtenida con el operador dos puntos :

7. Crear un vector d con los elementos del vector c (ejercicio 4) que van desde la posición 25 hasta la 15 en decrementos de 5, y desde la 20 a la 30 con incrementos de 6.

```
>> d=[c(25:-5:15),c(20:6:30)]

d =

  102  82      62      82  106
```

8. Crear un vector que tenga como elemento inicial el 5 hasta el 18 en incrementos de 3, mostrar los elementos 4, 1, 5 en un vector en el orden mencionado.

```
>> p=5:3:18

p =

        5       8       11      14      17

>> q=[p(4),p(1),p(5)]

q =
        14      5       17
```

9. Crear un vector del 0 al 20 en incrementos de 5. Elevar a la cuarta cada elemento. Sumar de forma acumulada los elementos. Determinar el producto de los elementos del vector.

```
>> a=0:5:20

a =

     0     5    10    15    20

>> b=a.^4

b =

     0   625 10000 50625 160000

>> c=cumsum(a)

c =

     0     5    15    30    50

>> d=prod(a)

d =

     0
```

10. Crear un vector x que va desde 2 al 20 en incrementos de 4.

Crear otro vector y con los mismo elementos de x, de forma que la suma de los dos vectores nos dé un vector con todos los elementos iguales.

```
>> x=2:4:20

x =

     2     6    10    14    18
```

```
>> y=x(5:-1:1)

y =

    18    14    10     6     2

>> z=x+y

z =

    20    20    20    20    20
```

1.4 Matrices en Matlab

1.4.1 Definir matrices

De acuerdo a los elementos que se ingresen en la matriz, se determina el número de filas y columnas y, por lo tanto ,el tamaño de la misma; no es necesario establecer su tamaño inicialmente.

Los elementos de las filas se separan con espacios o comas que forman las columnas, mientras que, para formar una nueva fila, se utiliza el punto y coma (;).

Ejemplo:

La siguiente instrucción crea una matriz A y una matriz B de dimensión (3x3), en la primera se utilizan espacios y en la segunda la coma para definir los elementos de cada fila.

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

A =
```

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

>> B=[3,5,4;1,7,3;5,9,2]

B =

     3     5     4
     1     7     3
     5     9     2
```

Se pueden definir matrices especificando una fila en cada línea.

```
>> C=  [  5  2  6  9
       2  8  5  1
       6 4  3  0]

C =

     5     2     6     9
     2     8     5     1
     6     4     3     0
```

Se puede utilizar el operador dos puntos (:) o las funciones linspace o logspace para generar los elementos de las filas sin tener que escribirlos uno a uno, como se revisó en los vectores.

Ejemplo:

```
>> A=[1:3:15;5:-1:1;linspace(3,30,5)]

A =

    1.0000 4.0000 7.0000   10.0000   13.0000
    5.0000 4.0000 3.0000 2.0000 1.0000
    3.0000 9.7500   16.5000   23.2500   30.0000
```

1.4.2 Definición de matrices en función de otras matrices

Se puede definir una matriz en Matlab en términos de otro vector o de otra matriz que ya se haya creado antes.

```
>> B=[4 7 1]

B =

   4    7    1

>> A=[2 9 B]

A =

   2    9    4    7    1

>> C=[9 2 3 8 1;A]

C =

        9    2    3    8    1
        2    9    4    7    1
```

Nota: se debe tomar en cuenta la dimensión de las matrices.

Para acceder a las componentes de una matriz se utilizan unos enteros llamados índices. Estos se enumeran de columna en columna cuando utilizamos un solo índice.

```
>> C=[9 2 3 8 1;A]

C =

   9   2   3   8   1
   2   9   4   7   1

>> C(5)

ans =

   3
```

O podemos hacerlo nombrando el número de la fila y la columna.

```
>> C

C =

   9   2   3   8   1
   2   9   4   7   1

>> C(2,4)

ans =

   7
```

Se pueden cambiar los valores en una matriz, especificando su ubicación dentro de la matriz.

```
>> C

C =

   9   2   3   8   1
   2   9   4   7   1

>> C(1,4)=5

C =

   9   2   3   5   1
   2   9   4   7   1
```

1.4.3 Uso del operador dos puntos

Se puede definir una nueva matriz o modificar una existente utilizando el operador dos puntos, también se utiliza para extraer datos de las matrices con los cuales se pueden realizar operaciones, cálculos y análisis de datos.

Cuando se extrae información de una matriz al utilizar dos índices, se menciona siempre primero la o las filas, y luego la o las columnas.

Se utiliza el operador dos puntos (:) para hacer referencia a todas las filas o a todas las columnas.

Se utiliza el operador dos puntos (:) para hacer referencia un intervalo.

- A(:,a) e xtrae de la matriz A los elementos de todas filas (:) que se encuentran en la columna a.

Ejemplos:

```
>> C

C =
```

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

```
>> a=C (:,3)

a =

   3
   4
   8
   4
```

Almacena en la matriz a los elementos de todas las filas de la columna 3.

► A(b,:) Extrae de la matriz A los elementos de la fila b de todas las columnas (:).

Ejemplos:

```
>> b=C(2,:)

b =

   2   9   4   7   6
```

Almacena en la matriz b los elementos de la fila 2 de todas las columnas.

► A(a:b,c:d) Extrae de la matriz A los elementos comprendidos entre la fila a y b y la columna c y d.

Ejemplo:

```
>> C

C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> v=C(2:3,3:5)

v =

   4   7   6
   8   9   1
```

Almacena en la matriz v los elementos de la matriz C de las filas 2 hasta la 3, de las columnas 3 a la 5.

- A(a:p:b,c:q:d) Extrae de la matriz A los elementos de las comprendidos entre a y b con intervalo de p y de las columnas entre c y d en intervalos de q.

Ejemplo:

```
C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> f=C(2:2:5,1:3:4)
```

```
f =

    2    7
    3    7
```

Almacena en una matriz f los elementos de la matriz C que se encuentran en las filas 2 hasta la 5 en intervalos de dos en dos, de las columnas 1 hasta la 4 en intervalos de tres en tres.

- A(:,c:d) Extrae de la matriz A los elementos de todas las filas (:) que se encuentran entre las columnas c y d.

Ejemplo:

```
C =

    9    2    3    8    1
    2    9    4    7    6
    4    7    8    9    1
    3    2    4    7    2

>> g=C(:,5:-2:1)

g =

    1    3    9
    6    4    2
    1    8    4
    2    4    3
```

Almacena en una matriz g los elementos de la matriz C de todas las filas (:) que se encuentran de la columna 5 hasta la columna 1 en intervalos de menos uno.

```
C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> t=C(:,2:4)

t =

   2   3   8
   9   4   7
   7   8   9
   2   4   7
```

Almacena en la matriz t los elementos de la matriz C de todas las filas (:) que se encuentren en las columnas 2 hasta la 4.

- A(a:b,:) Extrae de la matriz A los elementos comprendidos entre la fila a y b de todas las columnas (:).

Ejemplo:

```
C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> v=C(1:2:4,:)

v =

   9   2   3   8   1
   4   7   8   9   1
```

Almacena en la matriz v los elementos de la matriz C de la fila 1 hasta la 4 en intervalos de dos en dos de todas las columnas (:).

► A(:,[c d e …) Extrae de la matriz A los elementos de todas las filas que se encuentran en las columnas c, d, e,…

Ejemplo:

```
C =

     9     2     3     8     1
     2     9     4     7     6
     4     7     8     9     1
     3     2     4     7     2

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

f =

     9     2     1
     2     9     6
     4     7     1
     3     2     2
```

Almacena en la matriz f los elementos de la matriz C de todas las filas (:) que se encuentran en las columnas 1, 2 y 5 (las columnas no necesariamente deben ser seguidas, ni con un intervalo).

► A([a b c …], :) Extrae de la matriz A los elementos de las fila a, b, c… de todas las columnas (:).

Ejemplo:

```
C =

    9    2    3    8    1
    2    9    4    7    6
    4    7    8    9    1
    3    2    4    7    2

>> g=C([1 4],:)

g =

    9    2    3    8    1
    3    2    4    7    2
```

Almacena en la matriz g los elementos de la matriz C de las filas 1 y 4 de todas las columnas. (Las filas se pueden enlistar sin un orden ni con intervalo).

► A([a b],[c d]) Extrae de la matriz A los elementos que se encuentran entre la fila a y b y los que se encuentran en la intersección de las filas a, b y de las columnas c y d.

Ejemplo:

```
C =

    9    2    3    8    1
    2    9    4    7    6
    4    7    8    9    1
    3    2    4    7    2

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

a =

    2    1
    7    1
```

Almacena en la matriz a los elementos de la matriz C comprendidos en la intersección de las filas 1 y 3 y de las columnas 2 y 5.

- A(a,end) Extrae de la matriz A el elemento que se encuentra al final de la fila a.

```
C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> y=C(2,end)

y =

   6
```

Extrae de la matriz C el último elemento de la fila 2 y lo almacena en y.

- A(end,c) Extrae de la matriz A el último elemento de la columna c.

Ejemplo:

```
C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> b=C(end,3)

   b =

   4
```

Extrae de la matriz C el último elemento de la columna 3 y lo almacena en b.

► A(end,end) o A(end) Extrae de la matriz A el elemento de la última fila y columna.

Ejemplo:

```
C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> h=C(end)

h =

   2
```

Extrae de la matriz C el último elemento de la matriz y lo almacena en h.

► A(:) Transforma la matriz en un vector tipo columna.

Ejemplo:

```
C =

   9   2   3   8   1
   2   9   4   7   6
   4   7   8   9   1
   3   2   4   7   2

>> p=C(:)
```

```
p =

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

Transforma la matriz C en un vector tipo columna.

1.4.4 Operaciones

Operaciones entre matrices

► Suma de matrices A+B: suma cada elemento de la matriz A con cada elemento de la matriz B, posición a posición.

Ejemplo:

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

```
A =

   1   6   7   5
   9   2   4   1
   1   7   9   5

>> B=[7 8 1 5;8 9 3 1;3 7 1 8]

B =

   7   8   1   5
   8   9   3   1
   3   7   1   8

>> C=A+B

C =

   8  14   8  10
  17  11   7   2
   4  14  10  13
```

► Resta de matrices A-B: resta cada elemento de la matriz A con cada elemento de la matriz B, posición a posición.

Ejemplo:

```
>> A
A =

   1   6   7   5
   9   2   4   1
   1   7   9   5

>> B

B =
```

```
   7    8    1    5
   8    9    3    1
   3    7    1    8

>> D=A-B

D =

  -6   -2    6    0
   1   -7    1    0
  -2    0    8   -3
```

► Multiplicación de un escalar por una matriz c*A: multiplica cada elemento de la matriz A por un escalar.

Ejemplo:

```
>> A

A =

   1    6    7    5
   9    2    4    1
   1    7    9    5

>> E=3*A

E =

   3   18   21   15
  27    6   12    3
   3   21   27   15
```

► Multiplicación de matrices A*B: realiza la multiplicación de matrices. (El número de columnas de la matriz A debe ser igual al número de filas de B para que se pueda llevar a cabo la multiplicación).

Ejemplo:

```
>> A

A =

   1   6   7   5
   9   2   4   1
   1   7   9   5

>> B=[2 9 1;3 4 5;9 1 5;4 8 2]

B =

   2   9   1
   3   4   5
   9   1   5
   4   8   2

>> A*B

ans =

  103   80   76
   64  101   41
  124   86   91
```

► Potenciación de una matriz A^n: eleva una matriz a una potencia n (valor escalar).

Ejemplo:

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

C =

   4   5   6
```

```
   9    1    8
   3    6    2

>> D=C^3

D =

     1093      912     1114
     1356      907     1322
      795      813      836
```

► División derecha A/B: divide cada elemento de la matriz A para una matriz B, posición a posición.

Ejemplo:

```
>> A

A =

   3    5
   7    1

>> B

B =

   3    5
   9    2

>> A/B

ans =

   1.0000         0
  -0.1282    0.8205
```

► División izquierda A\B: divide cada elemento de la matriz B para una matriz A, posición a posición.

Ejemplo:

```
>> A

A =

   3   5
   7   1

>> B

B =

   3   5
   9   2

>> A\B

ans =

   1.3125   0.1563
  -0.1875   0.9063
```

Operaciones elemento a elemento

► Multiplicación A.*B: multiplica cada elemento de la primera matriz con cada elemento de la segunda matriz (no necesita cumplir con las característica para realizar multiplicación de matrices).

Ejemplo:

```
>> C=[5 2 7;6 8 4]

C =

   5   2   7
   6   8   4

>> D=[1 5 3;9 3 6]

D =

   1   5   3
   9   3   6

>> E=C.*D

E =

   5  10  21
  54  24  24
```

- Potenciación A.^n: eleva cada elemento de la matriz a una potencia.

Ejemplo:

```
C =

   5   2   7
   6   8   4

>> F=C.^3

F =
```

```
 125   8  343
 216  512   64
```

► A.^B: eleva cada elemento de la primera matriz a una potencia con cada elemento de la segunda matriz.

Ejemplo:

```
>> C

C =

   5   2   7
   6   8   4

>> D

D =

   1   5   3
   9   3   6

>> G=C.^D

G =

        5       32      343
 10077696      512     4096
```

► División derecha A./B: divide cada elemento de la primera matriz con cada elemento de la segunda matriz (división derecha).

Ejemplo:

```
>> C

C =

   5   2   7
   6   8   4

>> D

D =

   1   5   3
   9   3   6

>> H=C./D

H =

   5.0000   0.4000   2.3333
   0.6667   2.6667   0.6667
```

► División izquierda A.\B: divide cada elemento de la matriz B para cada elemento de la matriz A.

Ejemplo:

```
>> C

C =

   5   2   7
   6   8   4
```

```
>> D

D =

   1   5   3
   9   3   6

>> Q=C.\D

Q =

   0.2000   2.5000   0.4286
   1.5000   0.3750   1.5000
```

1.4.5 Características de una matriz

► función size: [f c]=size(A) Determina la cantidad de filas y la cantidad de columnas que tiene la matriz y lo almacena en las variables f y c.

Ejemplo:

```
>> E

E =

   3   18   21   15
  27    6   12    3
   3   21   27   15

>> [f,c]=size(E)

f =

   3

   c =

    4
```

- Función det(A) calcula el determinante de la matriz

Ejemplo:

```
>> C

C =

   4   5   6
   9   1   8
   3   6   2

>> a=det(C)

a =

  152
```

- Función rank(A) calcula el rango de la matriz

Ejemplo:

```
>> C

C =

   4   5   6
   9   1   8
   3   6   2
>> b=rank(C)

b =

   3
```

► Función trace(A) calcula la traza de la matriz

Ejemplo:

```
>> D

D =

   5   4   3
   8   7   6
   1   2   3

>> t=trace(D)

t =

  15
```

1.4.6 Generación y manejo de matrices especiales

Existe una serie de funciones para crear matrices especiales que son utilizadas en algunos cálculos.

► Función eye(n) Esta función devuelve una matriz identidad de tamaño n.

Ejemplo:

```
>> eye(4)

ans =

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

Matriz identidad de tamaño 4x4.

► Función zeros(n) zeros(n,m) Esta función devuelve una matriz de ceros. Genera una matriz cuadrada cuando se usa la función con un solo argumento escalar de entrada. Mientras que si se indica el número de filas y columnas que va a tener la matriz de ceros, se deben ingresar en la función zeros dos argumentos de entrada.

Ejemplo:

```
>> zeros(3)

ans =

   0   0   0
   0   0   0
   0   0   0

>> zeros(3,4)

ans =

   0   0   0   0
   0   0   0   0
   0   0   0   0
```

► Función rand(n) - rand(n,m) Esta función devuelve una matriz de números aleatorios entre 0 y 1. Genera una matriz cuadrada cuando se usa la función con un solo argumento escalar de entrada. Si se utiliza la función rand con dos argumentos de entrada se está indicando el número de filas y columnas que va a tener la matriz.

Ejemplos:

```
>> rand(3)
```

```
ans =

   0.8147   0.9134   0.2785
   0.9058   0.6324   0.5469
   0.1270   0.0975   0.9575

>> rand(3,4)

ans =

   0.9649   0.9572   0.1419   0.7922
   0.1576   0.4854   0.4218   0.9595
   0.9706   0.8003   0.9157   0.6557
```

► Función ones(n) – ones(n,m) Esta función devuelve una matriz de unos. Cuando se usa la función con un solo argumento escalar de entrada, se genera una matriz cuadrada. Si se utiliza la función ones con dos argumentos de entrada se está indicando el número de filas y columnas que va a tener la matriz.

Ejemplos:

```
>> ones(4)

ans =

   1   1   1   1
   1   1   1   1
   1   1   1   1
   1   1   1   1

>> ones(2,4)

ans =

   1   1   1   1
   1   1   1   1
```

► Función diag(A) Devuelve los elementos de la diagonal principal de una matriz en forma de vector.

```
>> B

B =

   2   6   4   8
   9   5   1   8
   2   6   4   9
   5   8   3   8

>> p=diag(B)

p =

   2
   5
   4
   8
```

► diag(v) Crea una matriz diagonal con los elementos del vector v en la diagonal principal.

```
>> v=[9 7 4 8]

v =

   9   7   4   8

>> A=diag(v)

A =

   9   0   0   0
   0   7   0   0
   0   0   4   0
   0   0   0   8
```

► Función tril(A) Esta función devuelve la parte triangular inferior de una matriz.

Ejemplo:

```
B =

   2   6   4   8
   9   5   1   8
   2   6   4   9
   5   8   3   8

>> tril(B)

ans =

   2   0   0   0
   9   5   0   0
   2   6   4   0
   5   8   3   8
```

► Función triu(A) Esta función devuelve la parte triangular superior de una matriz.

Ejemplo:

```
B =

   2   6   4   8
   9   5   1   8
   2   6   4   9
   5   8   3   8

>> triu(B)

ans =
```

```
2   6   4   8
0   5   1   8
0   0   4   9
0   0   0   8
```

► Función flipud(A) Esta función devuelve la matriz invertida de abajo hacia arriba.

Ejemplo:

```
B =

   2   6   4   8
   9   5   1   8
   2   6   4   9
   5   8   3   8

>> flipud(B)

ans =

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

► Función fliplr(A) Esta función devuelve una matriz invertida de derecha a izquierda.

Ejemplo:

```
>> B
```

```
B =

   2   6   4   8
   9   5   1   8
   2   6   4   9
   5   8   3   8

>> fliplr(B)

ans =

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

- Función inv(A) Esta función devuelve la matriz inversa.

Ejemplo:

```
B =

   2   6   4   8
   9   5   1   8
   2   6   4   9
   5   8   3   8

>> inv(B)

ans =

   0.9000   0.2000  -0.8000  -0.2000
  -0.5429  -0.2000   0.2286   0.4857
   2.6143   0.2000  -1.9429  -0.6286
  -1.0000        0   1.0000        0
```

► Transpuesta de una matriz A’ Mediante este comando podemos obtener la transpuesta de una matriz. Es decir, cambia las filas por columnas y viceversa.

Ejemplo:

```
B =

   2   6   4   8
   9   5   1   8
   2   6   4   9
   5   8   3   8

>> B'

ans =

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

► Función magic(n) Matlab posee esta función la cual genera una matriz a la que se le llama mágica de tamaño n x n. La característica de estas matrices es que la suma de filas, columnas y las diagonales principal y secundaria es la misma, estas matrices no son utilizadas para cálculos numéricos sino más bien como entretenimiento o diversión.

Ejemplo:

```
>> magic(3)

ans =
```

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

>> magic(4)

ans =

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

Ejemplo:

Comprobar que la sumatoria de todas las columnas, filas y diagonales es igual.

```
>> d=magic(3)

d =

   8    1    6
   3    5    7
   4    9    2

>> sum(d)

ans =

  15   15   15

>> sum(d')

ans =

  15   15   15
```

```
>> sum(diag(d))

ans =

   15

>> v=fliplr(d)

v =

   6   1   8
   7   5   3
   2   9   4

>> sum(diag(v))

ans =

   15
```

1.4.7 Operaciones elementales fila y columna de matrices

Se pueden realizar diversas operaciones que nos lleven a la solución de problemas utilizando matrices.

A continuación algunas de las operaciones elementales FILA en una matriz.

1. Sumar — Una fila — Con un escala
   Restar
   Multiplicar
   Dividir
2. Intercambio de fila

3. Sumar Una fila A otra fila
   Restar
   Multiplicar
   Dividir

Ejemplos:

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

A =

   3   4   5   6
   8   9   1   3
   4   5   3   9

>> A(1,:)=2*A(1,:)

A =

   6   8  10  12
   8   9   1   3
   4   5   3   9
```

Multiplica todos los elementos de la fila 1 por 2 y lo almacena en la misma fila.

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

A =

   3   4   5   6
   8   9   1   3
   4   5   3   9

>> aux=A(3,:);
```

```
>> A(3,:)=A(1,:);
>> A(1,:)=aux;
>> A

A =

   4   5   3   9
   8   9   1   3
   3   4   5   6
```

Intercambia los elementos de la fila 1 con los elementos de la fila 3. Para ello se utiliza una variable auxiliar para almacenar temporalmente una de las filas y no perder valores.

Ejemplo:

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

A =

   3   4   5   6
   8   9   1   3
   4   5   3   9

>> A(2,:)=A(2,:)+4*A(3,:)

A =

   3   4   5   6
  24  29  13  39
   4   5   3   9
```

Almacena en la fila 2 la suma de la fila 2 con la multiplicación de la fila 3 por 4.

A continuación algunas de las operaciones elementales COLUMNA en una matriz.

1. Sumar Una columna Con un escalar
   Restar
   Multiplicar
   Dividir
2. Intercambio de columnas
3. Sumar Una columna A otra columna
   Restar
   Multiplicar
   Dividir

Ejemplos:

```
>> B=[-4 3 2 -1;5 6 -8 1;4 -2 -5 9]

B =

   -4    3    2   -1
    5    6   -8    1
    4   -2   -5    9

>> B(:,2)=B(:,2)*0.2

B =

  -4.0000    0.6000    2.0000   -1.0000
   5.0000    1.2000   -8.0000    1.0000
   4.0000   -0.4000   -5.0000    9.0000
```

Multiplica la columna 2 por 0.2 y lo almacena en la misma columna.

```
>> B=[-4 3 2 -1;5 6 -8 1;4 -2 -5 9]
B =
```

```
   -4    3    2   -1
    5    6   -8    1
    4   -2   -5    9

>> p=B(:,3);
>> B(:,3)=B(:,1);
>> B(:,1)=p;
>> B

B =

    2    3   -4   -1
   -8    6    5    1
   -5   -2    4    9
```

Intercambia los elementos de la columna 1 con los elementos de la columna 3.Para ello se utiliza la variable p para almacenar temporalmente una de las columnas.

```
>> B=[-4 3 2 -1;5 6 -8 1;4 -2 -5 9]

B =

   -4    3    2   -1
    5    6   -8    1
    4   -2   -5    9

>> B(:,2)=B(:,2)-B(:,4)/2

B =

   -4.0000    3.5000    2.0000   -1.0000
    5.0000    5.5000   -8.0000    1.0000
    4.0000   -6.5000   -5.0000    9.0000
```

Almacena en la columna 2 la resta de la columna 2 con la división para 2 de la columna 4.

1.4.8 Ejercicios resueltos de matrices

1. Crear una variable llamada d que contenga la segunda columna de la matriz llamada A.

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

A =

   5   6   2
   1   7   3
   5   9   2
   8   1   7

>> d=A(:,2)

d =

   6
   7
   9
   1
```

Al utilizar el operador dos puntos (:) como primer índice de la matriz A quiere decir que seleccione todas las filas y el 2 indica que es de la segunda columna.

2. Crear una variable v que contenga las últimas dos columnas de la matriz A.

```
>> A

A =
```

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

>> v=A(:,2:3)

v =

   6   2
   7   3
   9   2
   1   7
```

Selecciona todas las filas de la matriz desde la columna dos hasta la tres.

3. Sumar 1 a cada elemento de un vector v1 y almacenar el resultado en una variable llamada r.

```
>> v1=[3;7;6;9;4;2]

v1 =

   3
   7
   6
   9
   4
   2

>> r=v1+1

r =

   4
   8
   7
  10
   5
   3
```

Devuelve una vector columna de la misma dimensión sumando 1 a cada elemento del vector ingresado. Con el operador suma (+) no hace falta utilizar el punto (.) para que la operación se realice elemento a elemento.

4. Crear una matriz vs que sea la suma de las matrices v1 y v2.

```
>> v1=[4 5;7 1]

v1 =

   4   5
   7   1

>> v2=[9 2;5 7]

v2 =

   9   2
   5   7

>> vs=v1+v2

vs =

  13   7
  12   8
```

Suma cada elemento de la matriz v1 con cada elemento de la matriz v2, y almacena su resultado en una matriz vs de la misma dimensión.

5. Crear una variable va que contenga el valor de la matriz vs del ejercicio anterior dividido para 3.

```
>> va=vs/3

va =4.3333   2.3333
    4.0000   2.6667
```

Divide cada elemento de la matriz vs para 3, cuando la operación es entre una matriz y un escalar no es necesario utilizar el operador punto (.)

6. Crear una variable llamada ma que contenga la multiplicación de elemento por elemento de la matriz.

$$d = \begin{bmatrix} 3 & 5 & 1 \\ 7 & 2 & 9 \end{bmatrix} \quad \text{y la matriz} \quad v = \begin{bmatrix} 2 & 1 & 6 \\ 4 & 8 & 7 \end{bmatrix}$$

```
>> d=[3 5 1;7 2 9]

d =

   3   5   1
   7   2   9

>> v=[2 1 6;4 8 7]

v =

   2   1   6
   4   8   7

>> ma=d.*v

ma =

   6   5   6
  28  16  63
```

Cuando la operación es entre dos matrices y se desea realizar elemento a elemento se debe utilizar el operador punto (.), de lo contrario debe cumplir con la condición que el número de columnas de la primera matriz sea igual al número de filas de la segunda matriz para realizar la multiplicación en este ejercicio.

Dada la matriz $A = \begin{bmatrix} 3 & 0 & -2 \\ -1 & 8 & 4 \\ 6 & 5 & 0 \end{bmatrix}$

- Calcular la traza
- Calcular la inversa
- Calcular el determinante

```
>> A=[3 0 -2;-1 8 4;6 5 0]

A =

   3   0  -2
  -1   8   4
   6   5   0

>> tr=trace(A)

tr =

  11

>> inversa=inv(A)

inversa =

  -0.4348  -0.2174   0.3478
   0.5217   0.2609  -0.2174
  -1.1522  -0.3261   0.5217

>> deter=det(A)

deter =

  46
```

Para realizar estos cálculos, se utilizan las funciones predefinidas en Matlab.

A partir del vector $v = [\ 3 \quad -5 \quad 1\ ]$

Construya una matriz diagonal

Cambiar los elementos de la segunda fila por [-7 2 -4]

Luego, cambiar los elementos de la primera columna por $\begin{bmatrix} -2 \\ 1 \\ 9 \end{bmatrix}$

```
>> v=[3 -5 1]

v =

   3   -5    1

>> A=diag(v)

A =

   3    0    0
   0   -5    0
   0    0    1

>> A(2,:)=[-7 2 -4]

A =

   3    0    0
  -7    2   -4
   0    0    1

>> A(:,1)=[-2;1;9]

A =

  -2    0    0
   1    2   -4
   9    0    1
```

Para cambiar los elementos de la segunda fila, primero indicamos el lugar a ubicar los elementos es decir en la fila 2 en todas la columnas (:), de la misma manera para cambiar la columna indicamos todas las filas (:) de la columna 1.

9. Para la matriz $A = \begin{bmatrix} -2 & 3 & -1 \\ 6 & 9 & -4 \end{bmatrix}$

» Muestre la cantidad de filas y columnas de A

» Elevar cada elemento de la matriz al cubo y el resultado sumarlo con el triple del valor original de A.

```
>> A=[-2 3 -1;6 9 -4]

A =

   -2    3   -1
    6    9   -4

>> [fil,col]=size(A)

fil =

    2


col =

    3

>> D=A.^3

D =

   -8   27   -1
  216  729  -64

>> E=D+3*A

E =

  -14   36   -4
  234  756  -76
```

Utilizamos el operador punto (.) en el caso de la potenciación para elevar cada elemento de la matriz al cubo, y en el caso de la multiplicación no es necesario debido que es una operación entre un escalar y una matriz.

10. Crear las matrices A y B de orden 3

$$A = \begin{bmatrix} 1 & 6 & 2 \\ -4 & 5 & 9 \\ -1 & 6 & 8 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 8 & 7 \\ -7 & 2 & 1 \\ 3 & -6 & 9 \end{bmatrix}$$

» Juntarlas en una sola matriz $C = \begin{bmatrix} A \\ B \end{bmatrix}$, luego eliminar las filas impares
» A la matriz resultante D, multiplicar con A (E=DA)

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

A =

   1   6   2
  -4   5   9
  -1   6   8

>> B=[5 8 7;-7 2 1;3 -6 9]

B =

   5   8   7
  -7   2   1
   3  -6   9

>> C=[A;B]

C =

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

Primer forma de eliminar las filas impares

```
>> D=C(2:2:6,:)

D =

   -4    5    9
    5    8    7
    3   -6    9
```

Seleccionamos de la matriz C la fila hasta la fila en intervalos de 2, todas las columnas (:).

Segunda forma de eliminar las filas impares

```
>> C(1:2:6,:)=[]

C =

   -4    5    9
    5    8    7
    3   -6    9

>> D=C

D =

   -4    5    9
    5    8    7
    3   -6    9
```

Al utilizar los corchetes [] indicamos que se eliminen todas las filas que se mencionan en la extracción de la matriz la fila 1 hasta la 6 en intervalos de 2, y devuelve las demás filas es decir las filas pares.

```
>> E=D*A

E =

  -33   55  109
  -34  112  138
   18   42   24
```

CAPÍTULO II

2.1 Polinomios

Un polinomio puede ser introducido en Matlab mediante un vector cuyos elementos son los coeficientes reales o complejos del polinomio completo y ordenado (los términos se escriben de mayor a menor grado y se encuentran todos los términos completando con cero si fuese necesario).

Ejemplos:

Definir el siguiente polinomio $P(x)=4x^3-2x^2+6$

```
>> p=[4 -2 0 6]

p =

   4   -2    0    6
```

Definir el polinomio $S(x)= x^4 -2ix^2 + x-4 + i$

```
>> s=[1 0 -2i 1 -4+i]

s =

  1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 2.0000i   1.0000 + 0.0000i
-4.0000 + 1.0000i
```

2.1.1 Evaluando en la variable

- **Función polyval:** evalúa un polinomio en un valor.

Sintaxis:

polyval(p,x).

p-> polinomio
x-> valor para evaluar en el polinomio puede ser real o complejo
Ejemplo:

```
>> s=polyval(p,3)

s =

   96

>> t=polyval(p,2-i)

t =

  8.0000 -36.0000i
```

2.1.2 Raíces del polinomio

Tanto para los polinomios con coeficientes reales P(x) como también para polinomios con coeficientes complejos S(z), Matlab nos permite encontrar sus raíces con el uso del comando roots().

- **Función roots**

Sintaxis:

roots(p).

p-> polinomio con coeficientes reales P(x) o polinomios con coeficientes complejos S(x)

Ejemplo:

```
>> p

p =

    4    -2     0     6

>> raices_p=roots(p)

raices_p =

   0.7500 + 0.9682i
   0.7500 - 0.9682i
  -1.0000 + 0.0000i
```

Raíces del polinomio p con números reales.

```
>> s=[1 0 -2i 1 -4+i]

s =

   1.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 2.0000i   1.0000 + 0.0000i
-4.0000 + 1.0000i

>> raices_p=roots(s)

raices_p =

  -1.4841 - 0.2798i
  -0.3216 - 1.3889i
   0.5966 + 1.4362i
   1.2092 + 0.2325i
```

Raíces del polinomio s con coeficientes complejos.

2.1.3 Producto polinomial

► **Función conv:** multiplicación de polinomios

Sintaxis:

conv(p,t)
p-> polinomio
t->polinomio

Ejemplo:

Definir el polinomio $p(x)=x^3+3x^2-4x+2$ y el polinomio $t(x)=2x^2-3x+1$

```
>> p=[1 3 -4 2]

p =

   1    3   -4    2

>> t=[2 -3 1]

t =

   2   -3    1

>> s=conv(p,t)

s =

   2    3  -16   19  -10    2
```

Matlab devuelve los coeficientes del polinomio resultante $s(x)=2x^5+3x^4-16x^3+19x^2-10x+2$

2.1.4 División polinomial

► **Función deconv:** para dividir dos polinomios P(x) y T(x) utilizamos el comando [q,r]=deconv(p,t) y nos devolverá el cociente q(x) y residuo r(x).

Sintaxis:

```
[q,r]=deconv(p,t)

q->almacena el cociente de la división
r-> almacena le residuo de la división
p->polinomio
t->polinomio
```

Ejemplo:

```
>> p

p =

   1    3   -4    2

>> t

t =

   2   -3    1

>> [q,r]=deconv(p,t)

q =

  0.5000    2.2500


r =
            0         0    2.2500   -0.2500
```

Los polinomios resultantes de la división son:

Cociente: $q(x)=0.5x+2.25$

Residuo: $r(x)=2.25x-0.25$

2.1.5 Derivada de un polinomio

► **Función polyder(p):** calcula la derivada del polinomio

Ejemplo:

Calcular la derivada del siguiente polinomio $p(x)=x^3-4x^2+4$

```
>> p=[1 -4 0 4]

p =

   1   -4    0    4

>> d=polyder(p)

d =

   3   -8    0
```

El polinomio resultante es $d(x)=3x^2-8x$

2.1.6 Integración de un polinomio

► **Función polyint:** calcula la integral de un polinomio.

Sintaxis:

```
polyint(p)
p->polinomio
```

Ejemplo:

Calcular la integral del siguiente polinomio p(x)=□2x□^3+3x^2-2

```
>> p=[2 3 0 -2]

p =

   2    3    0   -2

>> polyint(p)

ans =

  0.5000    1.0000         0   -2.0000         0
```

Nos da como resultado: $\int p(x) = 0.5x^4+x^3-2x$

2.1.7 Polinomio interpolador

► **Función polyfit:** ajusta una curva polinómica a partir de unos puntos, el comando devuelve los coeficientes de un polinomio P(x) de grado n que tiene un mejor ajuste para los valores en (x,y).

Sintaxis:

```
polyfit(x,y,n)
```

x ,y->puntos de ajuste

n->grado del polinomio

Obtener el polinomio interpolar a partir de los siguientes puntos x=0 a 20 y=cos(x).

```
>> x=linspace(0,20,15);
>> y=cos(x);
>> p=polyfit(x,y,4)

p =

   0.0003   -0.0103    0.1376   -0.6703    0.8154
```

El polinomio resultante es: $p(x)=0.0003x^4-0.0103x^3+0.1376x^2-0.6703x+0.8154$

2.1.8 Ejercicios resueltos

1. Dado el polinomio $P(x)=x^4-2x^2+4x+6$

Evaluar $p(\pi^2)$

Determinar las raíces del polinomio

Calcular la derivada

```
>> p=[1 0 -2 4 6]

p =

     1     0    -2     4     6

>> eval=polyval(p,pi^2)
```

```
eval =

                9.3392e+03

>> raices=roots(p)

raices =

  1.3345 + 1.2370i
  1.3345 - 1.2370i
 -1.3345 + 0.1772i
 -1.3345 - 0.1772i

>> derivada=polyder(p)

derivada =

                4    0   -4    4
```

2. Dado el polinomio $q(x) = - x^3+ix^4+5x-2$

Evaluar q (3-i)

Determinar las raíces

Multiplicarlo con el polinomio $s(x)=3x^{2-4}$

```
>> q=[i -1 0 5 -2]

q =

  0.0000 + 1.0000i  -1.0000 + 0.0000i   0.0000 + 0.0000i   5.0000 + 0.0000i
-2.0000 + 0.0000i

>> val=polyval(q,3-i)

val =

 91.0000 +49.0000i
```

```
>> raices=roots(q)

raices =

  -0.1008 - 2.1270i
  -1.5595 + 0.5790i
   1.2465 + 0.5546i
   0.4138 - 0.0065i

>> s=[3 0 -4]

s =

    3     0    -4

>> mult=conv(q,s)

mult =

  Columns 1 through 6

   0.0000 + 3.0000i  -3.0000 + 0.0000i   0.0000 - 4.0000i  19.0000 + 0.0000i
-6.0000 + 0.0000i -20.0000 + 0.0000i

  Column 7

   8.0000 + 0.0000i
```

3. Crear un polinomio p de grado cuatro, con coeficientes iguales a los primeros términos de la sucesión de Fibonacci, el último término es el término independiente; es decir, 1.

Crear un vector d, con elementos iguales a los del polinomio anterior pero en orden inverso; evaluar el polinomio p en cada uno de los términos del vector d.

Hallar las raíces de p.

Mostrar la derivada e integral del polinomio p.

```
>> p=[8 5 3 2 1]

p =

   8    5    3    2    1

>> d=p(5:-1:1)

d =

   1    2    3    5    8

>> eva_p=polyval(p,d)

eva_p =

       19       185       817      5711     35537

>> raices=roots(p)

raices =

  0.1720 + 0.5886i
  0.1720 - 0.5886i
 -0.4845 + 0.3125i
 -0.4845 - 0.3125i

>> derivada=polyder(p)
derivada =

   32   15    6    2

>> integral=polyint(p)

integral =

   1.6000   1.2500   1.0000   1.0000   1.0000        0
```

2.2 Solución de sistemas lineales

Para resolver un sistema de ecuaciones lineales se pueden aplicar diferentes métodos. A continuación, algunos de ellos.

2.2.1 Solución con el uso de la matriz inversa

Para resolver un sistema de ecuaciones lineales podemos representar el mismo en forma matricial, para los cual consideramos lo siguiente:

$$ax+by+c=o$$
$$dx+ey+fz=p$$
$$gx+hy-iz=q$$

$$A=\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad X=\begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad B=\begin{bmatrix} o \\ p \\ q \end{bmatrix}$$

$$AX=B$$

En donde se sabe que

$$A^{-1}\,A=1$$

Entonces

$$A^{-1}\,A = A^{-1}\,B$$
$$x = A^{-1}\,B$$

Ejemplo:

Resolver el siguiente sistema de ecuaciones:

$$3x+2y+z=1$$
$$5x+3y+4z=2$$
$$x+y-z=1$$

El sistema de ecuaciones representado en forma matricial

$$A = \begin{bmatrix} 3 & 2 & 1 \\ 5 & 3 & 4 \\ 1 & 1 & -1 \end{bmatrix} \quad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

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

A =

   3   2   1
   5   3   4
   1   1  -1

>> B=[1;2;1]

B =

   1
   2
   1

>> x=inv(A)*B

x =

  -4.0000
   6.0000
   1.0000
```

La solución al sistema de ecuaciones es:

x=-4
y=6
z=1

2.2.2 Solución con división izquierda de matriz

En Matlab se puede usar división izquierda para resolver el problema por eliminación gaussiana.

X=A\B

Ejemplo:

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

A =

   3    2    1
   5    3    4
   1    1   -1

>> B=[1;2;1]

B =

   1
   2
   1

>> x=A\B

x =

  -4.0000
   6.0000
   1.0000
```

2.2.3 Solución utilizando la función rref

Esta función es útil para resolver sistemas de ecuaciones, se utiliza la matriz ampliada Ab del sistema.

Ejemplo:

```
>> A

A =

   3   2   1
   5   3   4
   1   1  -1

>> B

B =

   1
   2
   1

>> Ab=[A B]

Ab =

   3   2   1   1
   5   3   4   2
   1   1  -1   1

>> sol=rref(Ab)

sol =

   1   0   0  -4
   0   1   0   6
   0   0   1   1
```

Devuelve una matriz en la que la última columna esta la solución al sistema de ecuaciones.

2.2.4 Ejercicios resueltos

Resolver el siguiente sistema de ecuaciones utilizando los tres métodos.

```
5x+2y=1
-3x+3y=5

        >> A=[5 2;-3 3]

A =

   5    2
  -3    3

>> B=[1;5]

B =

   1
   5

>> format rat
>> x=inv(A)*B

x =

    -1/3
     4/3

>> A\B

ans =

    -1/3
```

4/3

```
>> Ab=[A B]

Ab =

     5     2     1
    -3     3     5

>> rref(Ab)

ans =

     1     0  -1/3
     0     1   4/3
```

Se mezclan dos tipos de líquido, el primero cuesta $0.94 dólar/litro y el segundo de $ 0.86 por litro, con lo que se obtiene 40 litros de mezcla a $ 0.89 por litro. ¿Cuántos litros se utilizó de cada tipo?

	Tipo 1	**Tipo 1**	**Mezcla**
N° Litros	x	y	40
Precio/litro	0,94	0,86	0,89
Precio total	0,94x	0,86y	35,6

$$x+y=40$$
$$0.94x+0.86y=35.6$$

```
>> A=[1 1;0.94 0.86]

A =

   1.0000    1.0000
```

```
   0.9400    0.8600

>> B=[40;35.6]

B =

   40.0000
   35.6000

>> sol=inv(A)*B

sol =

   15.0000
   25.0000

>> sol1=A\B

sol1=

   15.0000
   25.0000

>> Ab=[A B]

Ab =

    1.0000    1.0000   40.0000
    0.9400    0.8600   35.6000


>> sol2=rref(Ab)

sol2 =

     1     0    15
     0     1    25
```

CAPÍTULO III

3.1 Funciones para graficar en 2D

Matlab, en cualquier versión, dispone de varias funciones que permiten graficar vectores, matrices o funciones en dos dimensiones.

3.1.1 Función plot

Esta función permite graficar en 2D vectores o matrices en una escala lineal es decir la misma distancia entre punto y punto.

Permite graficar en el plano x y y los valores de los vectores X y Y siempre y cuando la longitud de estos sea igual. De igual manera si son matrices deben ser de la misma dimensión.

Si X o Y es un vector y el otro es una matriz, de la misma forma una de sus dimensiones debe ser igual a la longitud del vector.

Si el número de filas de la matriz es igual a la longitud del vector, entonces se grafica el vector *versus* cada columna. Y de igual forma si el número de columnas es igual a la longitud del vector, el comando traza cada fila con el vector como se observa en la siguiente figura.

```
>> a=[1 2 3 4 5];
>>b=[2 3;3 4;4 5;5 6;6 7];
>> plot(a,b)
```

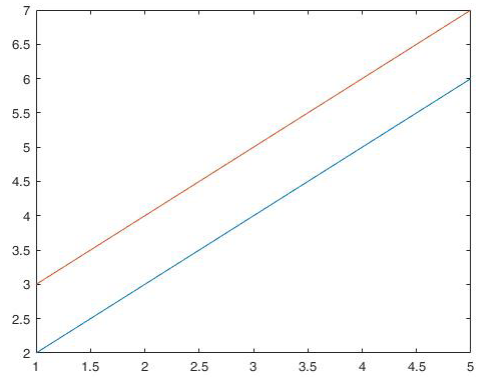


Fig. 17. Función plot

El comando plot grafica también valores reales como imaginarios tanto para los ejes de abscisas y ordenadas.

```
>> compl=[3 -i 2+i 3*i 4];
>> plot(compl)
```

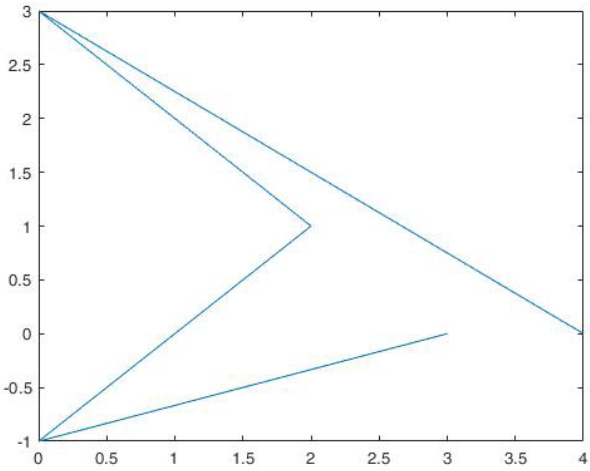


Fig. 18. Gráfica de números imaginarios

Si X y Y son escalares, la función plot grafica puntos discretos, sin embargo,para poder identificarlos es necesario agregar un atributo de marcador que se verá más adelante.

3 .1.2 Función plot(X,Y,m)

A la función anterior se le pueden añadir estilos para diferenciar los gráficos en color, tipo de línea, y marcadores. El atributo m es de tipo string compuesto de tres caracteres que definen estos atributos.

Primer carácter: color

y Yellow

m Magenta

c Cyan

r Red
g Green
b Blue
w White
k Black

Segundo carácter: tipo de línea

- Línea continua
: Línea a puntos
-. Línea a barra-punto
-- Línea a trazos

Tercer carácter: marcador

. Puntos
o Círculos
x Marcas con x
+ Marcas con +
* Marcas con *
s Marcas cuadradas (square)
d Marcas en diamantes (diamond)
^ Triángulo apuntando arriba
v Triángulo apuntando abajo
> Triángulo apuntando a la derecha
< Triángulo apuntando a la izquierda
p estrella de cinco puntas
h estrella de seis puntas

Ejemplo:

Trazar la función seno entre 0 y π y agregar los tres tipos de estilo:

```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> plot(x,y,'bo--')
```

Nota: Los atributos pueden especificarse en cualquier orden.

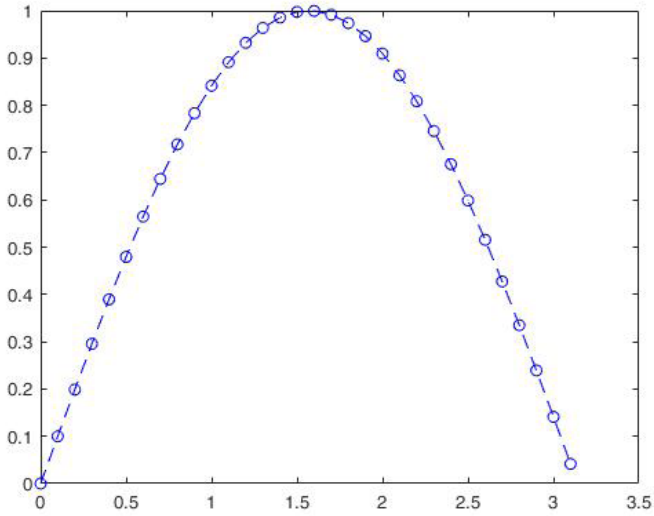


Fig. 19. Características de una gráfica

Se pueden también graficar varias funciones en una misma ventana y se diferencian usando diferentes características para cada una de ellas.

3.1.3 Función plot(X1,Y1,m1,…,Xn,Yn,mn)

Ejemplo:

Graficar la función seno y coseno entre 0 y 2π usando características diferentes:

```
>> x=0:0.1:2*pi;
>> y1=sin(x);
>> y2=cos(x);
>> plot(x,y1,'rx',x,y2,'bo')
```

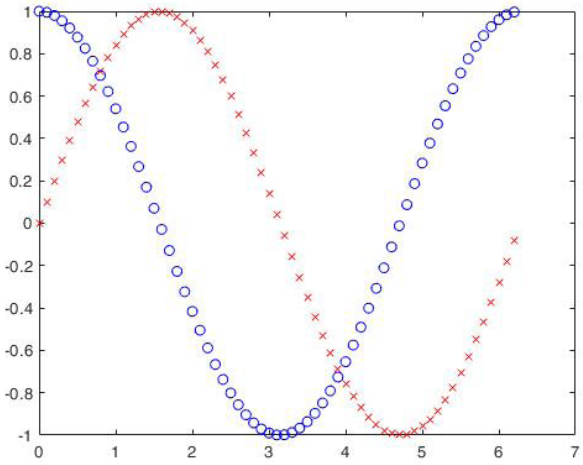


Fig. 20. Grafica de la función seno y coseno en una sola figura

3.1.4 Función plotyy()

Esta función grafica funciones que requieran dos escalas diferentes en el eje de las ordenadas.

Ejemplo:

```
>> x1=0:0.1:2*pi;
>> y1=sin(x);
>> y2=3*cos(x);
>> plotyy(x,y1,x,y2)
```

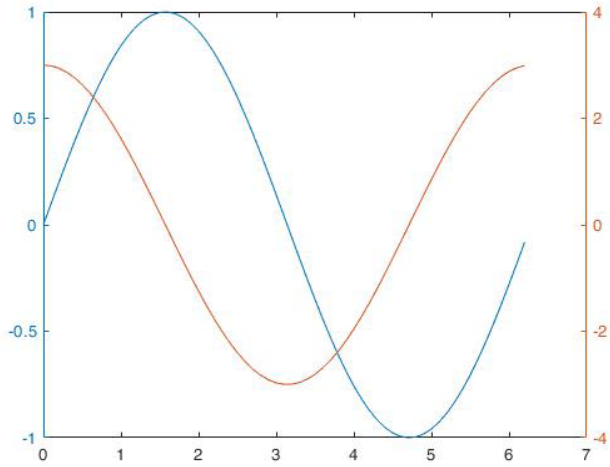


Fig. 21. Funciones con escala diferente en el eje y

3.1.5 Función loglog()

Grafica de igual forma que la función plot pero en escala logarítmica para los dos ejes.

Ejemplo:

```
>> x=0:1:100;
>> y=2+3*x;
>> loglog(x,y)
```

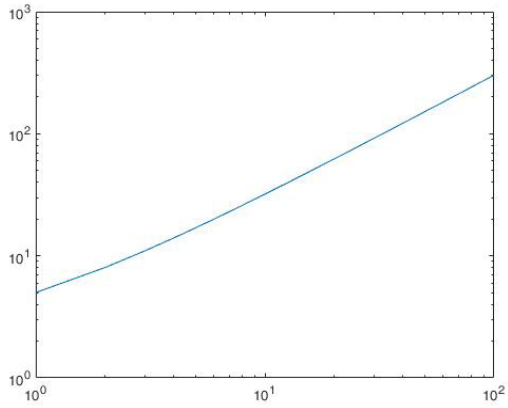


Fig. 22. Grafica de una función en escala logarítmica

3.1.6 Función semilogx()

Esta función es similar a plot y su nombre se debe a que la gráfica en el eje de las abscisas es en escala logarítmica y en el eje de las ordenadas es de escala lineal, es decir, normal.

Ejemplo:

```
>> x=0:1:100;
>> y=2+3*x;
>> semilogx(x,y)
```

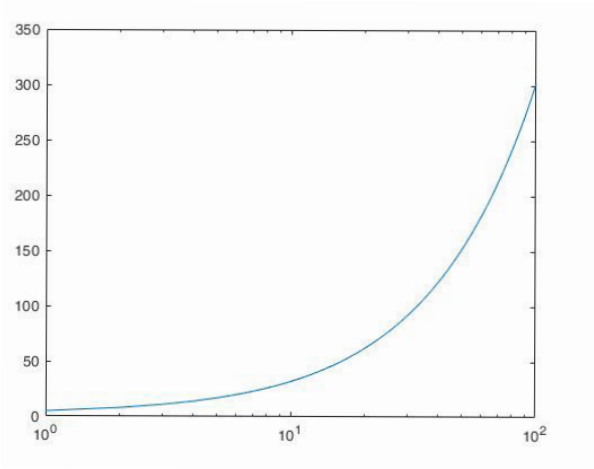


Fig. 23. Gráfica semilogarítmica eje x

Se puede realizar gráfica semilogarítmica solo en el eje y utilizando el comando semilogy()

Ejemplo:

```
>> x=0:1:100;
>> y=2+3*x;
>> semilogy(x,y)
```

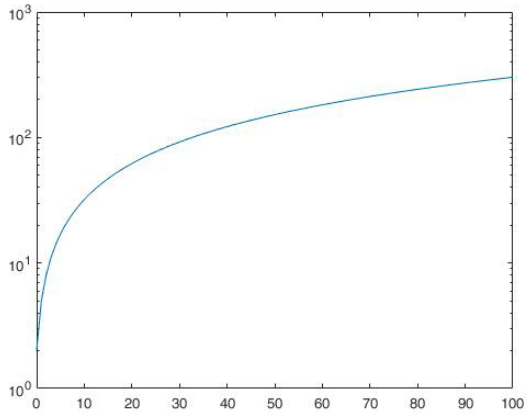


Fig. 24. Gráfica semilogaritmica eje y

3.1.7 Función fplot()

fplot (fun,[xmin, xmax]) grafica la función en el intervalo de variación de x definido.

fplot (fun,[xmin, xmax],m) similar al anterior añadiendo las opciones de color y caracteres que se especifican en m.

fplot fun, [xmin, xmax,ymin,ymax],m) grafica la función en los intervalos de x y y definidos, con las opciones de color y caracteres dadas por m.

fun indica la función que se va a graficar, se debe ingresar entre apóstrofes.

Ejemplos:

```
>> fplot('x^3',[-2 12],'rx')
```

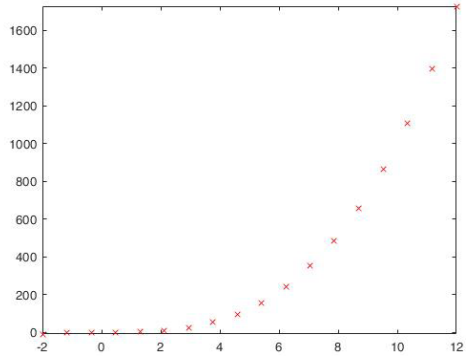


Fig. 25. Gráfica utilizando la función fplot con intervalo en el eje x y características

```
>> fplot('x^3',[-2 5 -5 5],'rx')
```

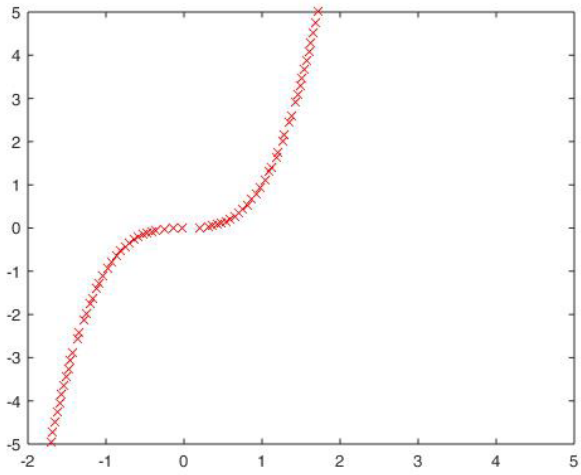


Fig. 26. Gráfica utilizando la función fplot con intervalo en el eje x - y y características

3.1.8 Función ezplot()

ezplot (fun) grafica la expresión fun(x) sobre el rango definido -2π<x<2π, donde fun(x) es una función explícita de una sola variable x.

ezplot (fun, [xmin, xmax]) representa la fun(x) sobre el rango: xmin<x<xmax.

Para una función implícita, fun2(x,y):

ezplot (‘fun2’) grafica fun2(x,y)=0 sobre el rango definido -2π<x<2π, -2π<y<2π

ezplot (‘fun2’, [a, b]) grafica fun2(x,y)=0 sobre a<x<b y a<y<b.

ezplot (‘fun2’, [xmin, xmax, ymin, ymax]) grafica fun2(x,y)=0 sobre xmin<x<xmax y ymin<y<ymax.

Ejemplo:

Graficar la función implícita x^2-y^2=0

```
>> ezplot('x^2-y^2')
```

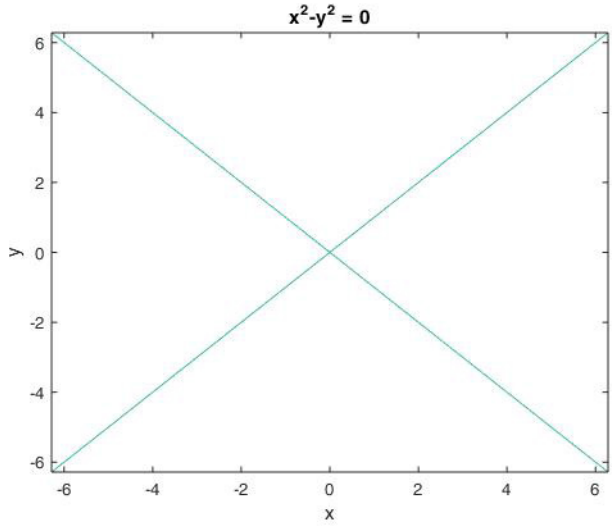


Fig. 27. Gráfica utilizando la función ezplot()

3.2 Subdivisión de Ventanas

Una ventana gráfica puede ser dividida en varias subventanas como se desee, dentro de la misma figura. Esto es que se pueden tener n filas por m columnas, los cuales manejan sus propios ejes y características.

Se utiliza el comando subplot(n,m,k)

n -> es el número de filas en que se divide la figura

m -> el número de columnas en que se divide la figura

k -> indica la posición activa en la cual se desea graficar.

Ejemplo:

Graficar para el intervalo 0<x<2π, las funciones seno, coseno, tangente y f(x)=x^2, respectivamente, en una sola figura.

```
>> x=0:0.1:2*pi;
>> y1=sin(x);
>> y2=cos(x);
>> y3=tan(x);
>> y4=x.^2;
>> subplot(2,2,1),
>> plot(x,y1)
>> subplot(2,2,2),
>> plot(x,y2)
>> subplot(2,2,3),
>> plot(x,y3)
>> subplot(2,2,4),
>> plot(x,y4)
```

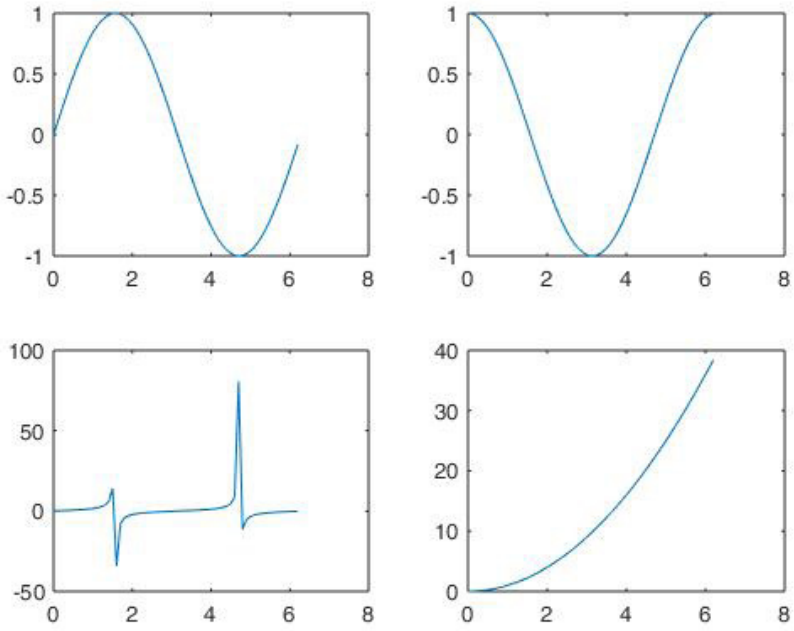


Fig. 28. Uso del comando subplot.

3.3 Control de ejes

Por defecto, Matlab escala el gráfico a los valores máximos y mínimos para el control de sus ejes. Sin embargo, en ocasiones es necesario aumentar o disminuir la escala de los ejes dependiendo de las necesidades.

Función axis([xmin, xmax, ymin, ymax])

xmin -> valor mínimo para el eje de las x

xmax -> valor máximo para el eje de las x

ymin -> valor mínimo para el eje de las y

ymax -> valor máximo para el eje de las y

axis('auto')

Esto permite volver al escalado por defecto de los ejes.

v=axis

Devuelve los valores mínimos y máximos de los ejes en forma de vector.

axis('ij')

Utiliza ejes de pantalla, con el origen en la esquina superior izquierda y el eje j en dirección vertical descendente.

axis('xy')

Utiliza ejes cartesianos normales, con el origen en la esquina inferior izquierda y el eje y vertical ascendente.

axis('image')

La ventana tendrá las proporciones de la imagen que se desea representar en ella (por ejemplo la de una imagen bitmap que se desee importar) y el escalado de los ejes será coherente con dicha imagen.

axis('equal')

Ambos ejes tienen la misma escala.

axis('square')

Devuelve una escala en la cual es cuadrada la ventana.

axis('normal')

Devuelve los ejes a su forma original, eliminado comandos ejecutados antes.

axis('off')

Desactiva los ejes, etiquetas, y los números de la figura.

axis('on')

Activa los ejes, las etiquetas y números de la figura.

Ejemplos:

>> axis([-2,7,-2,2])

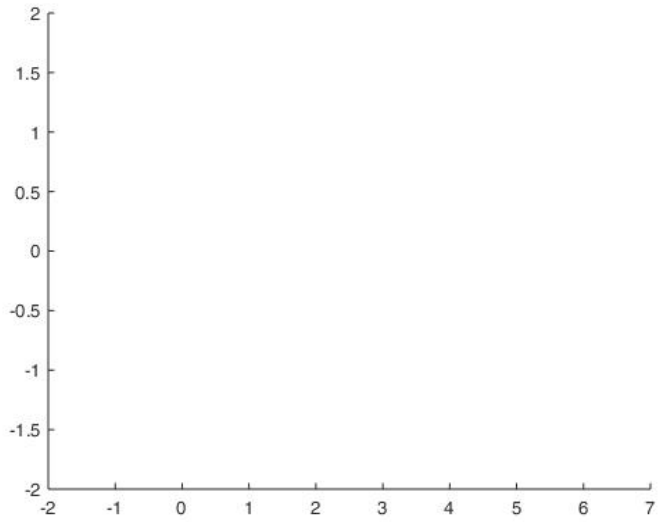


Fig. 29. Escalamiento en el eje x y y

>> axis('ij')

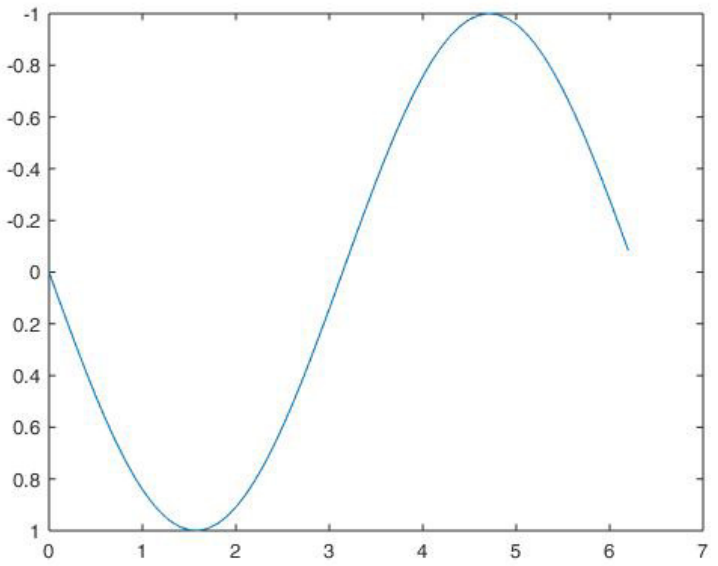


Fig. 30. Gráfica utilizando axis('ij')

```
>> axis('xy')
```

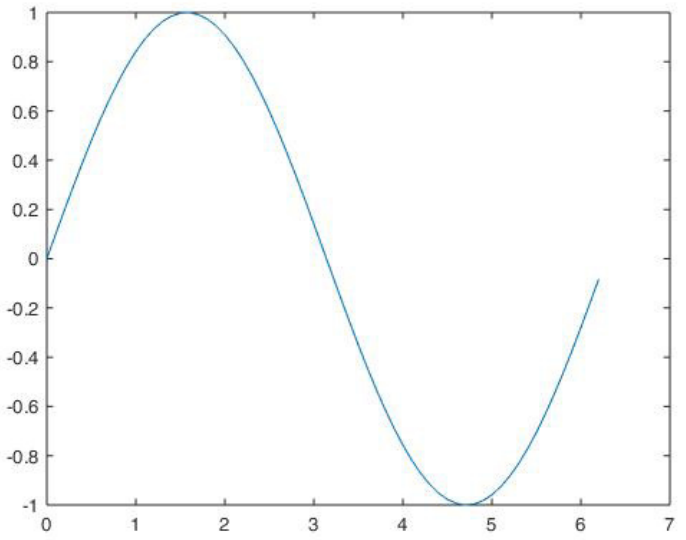


Fig. 31. Gráfica utilizando axis('xy')

```
>> axis('image')
```

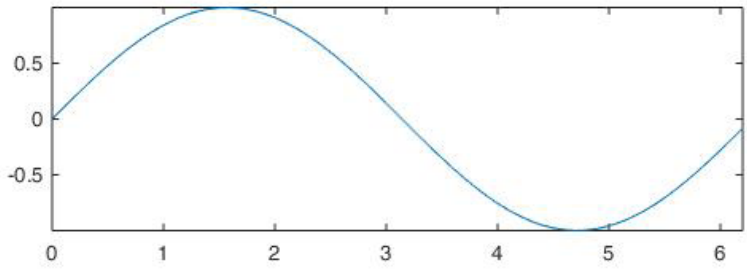


Fig. 32. Gráfica utilizando axis(‘image’)

```
>> axis('equal')
```

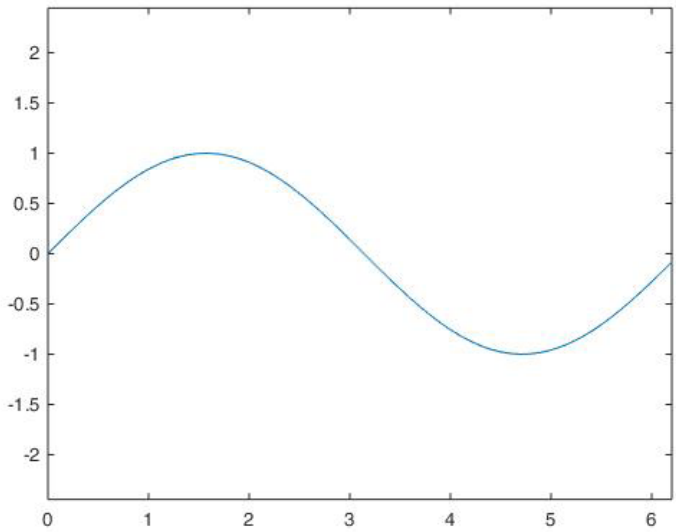


Fig. 33. Gráfica utilizando axis(‘equal’)

```
>> axis('square')
```

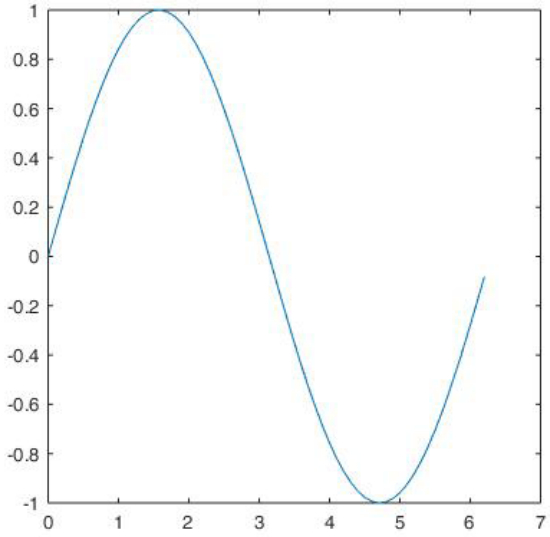


Fig. 34. Gráfica utilizando axis(‘square’)

```
>> axis('normal')
```

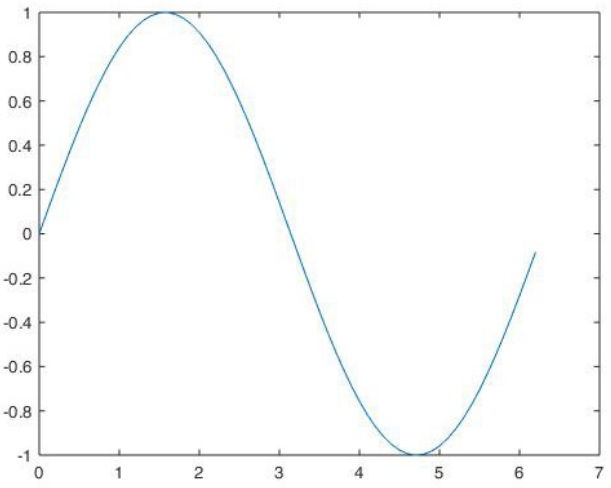


Fig. 35. Gráfica utilizando axis('normal')

```
>> axis('off')
```

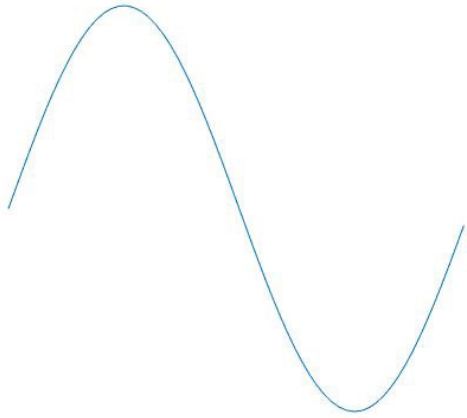


Fig. 36. Gráfica utilizando axis('off')

```
>> axis('on')
```

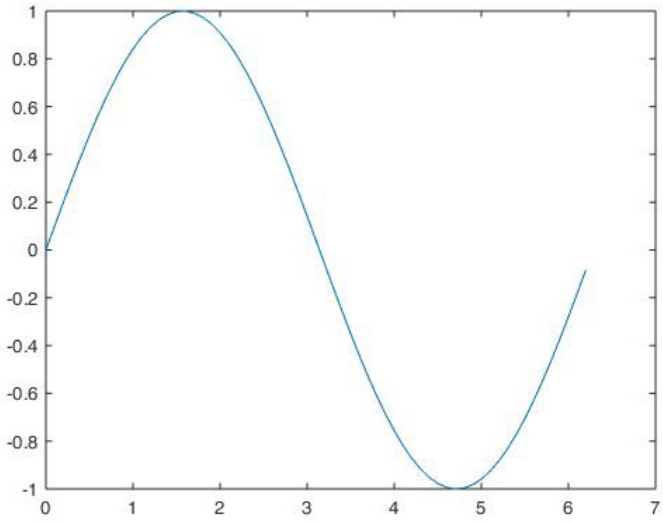


Fig. 37. Gráfica utilizando axis('on')

3.4 Títulos y etiquetas

Matlab permite colocar títulos a los gráficos, nombres a los ejes, leyendas descriptivas y texto en cualquier punto dentro del gráfico. Para esto se usan los siguientes comandos de la tabla.

► **Función title:** añade el texto como título del gráfico en la parte superior del mismo.

Sintaxis: title (‘texto’)

Ejemplo:

```
>> title('Funcion Seno')
```

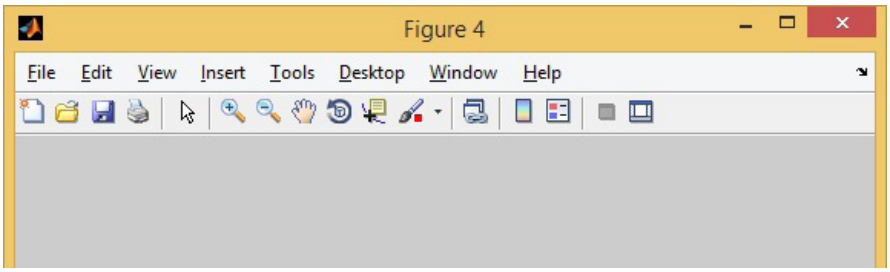


Fig. 38. Uso del comando *title*

► **Función xlabel, ylabel:** etiquetas del eje x y del eje y respectivamente.

Sintaxis: xlabel(‘texto’)
ylabel(‘texto’)

Ejemplo:

```
>> xlabel('eje x'),
>> ylabel('eje y')
```

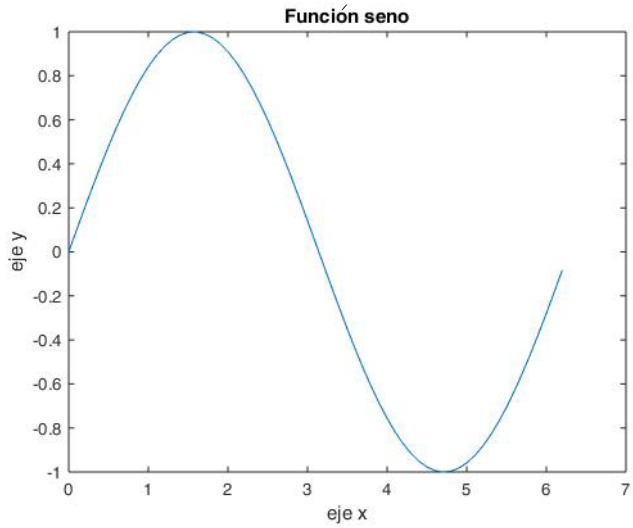


Fig. 39. Etiquetas a los ejes x y y

► **Función legend** (‘cadena1’, ’cadena2’,…): sitúa las leyendas especificadas por las cadenas en n gráficos consecutivos.

Sintaxis: legend(‘cadena1’, ’cadena2’,…)

legend(‘off’) Elimina las leyendas de los ejes actuales.

Ejemplo:

```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> z=cos(x);
>> plot(x,y,x,z)
>> legend('Seno', 'Coseno')
```

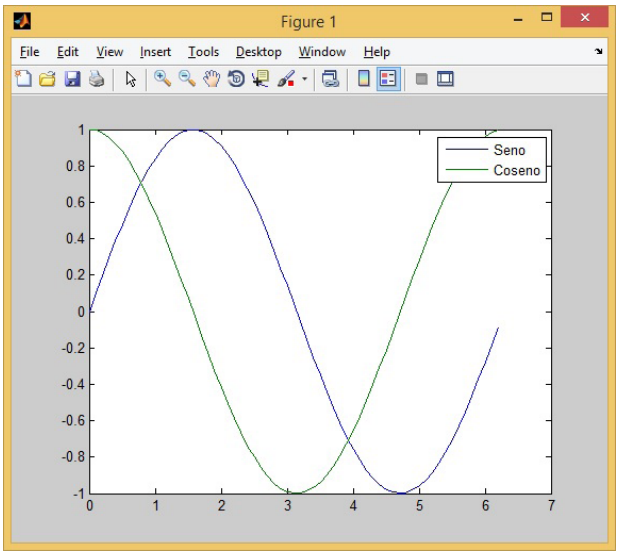


Fig. 40. Uso de la función legend

► **Función text** (x, y, 'texto') Sitúa el texto en el punto (x,y) dentro del grafico 2D.

Sintaxis: text(x,y,'texto')

Ejemplo:

```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> plot(x,y)
>> text(3,0,'Funcion Seno')
```

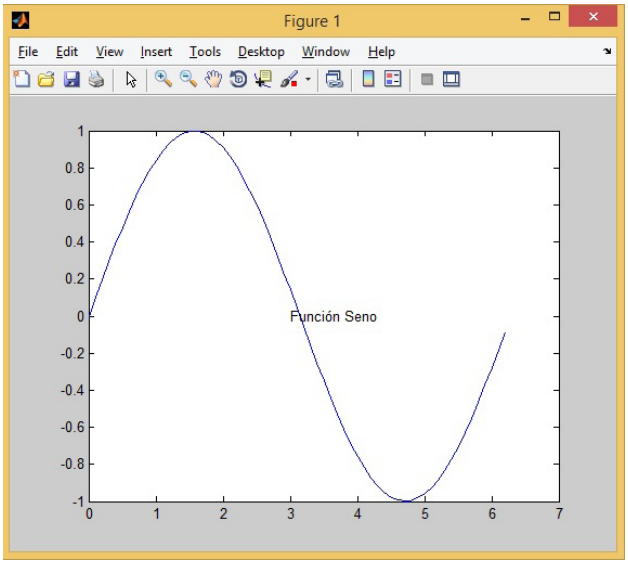


Fig. 41. Uso de la función text

► **Función gtext:** permite situar el texto en un punto seleccionado con el ratón dentro de un gráfico 2D.

Sintaxis: gtext('texto')

Ejemplo:

```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> plot(x,y)
>> gtext(3,0,'Funcion Seno')
```

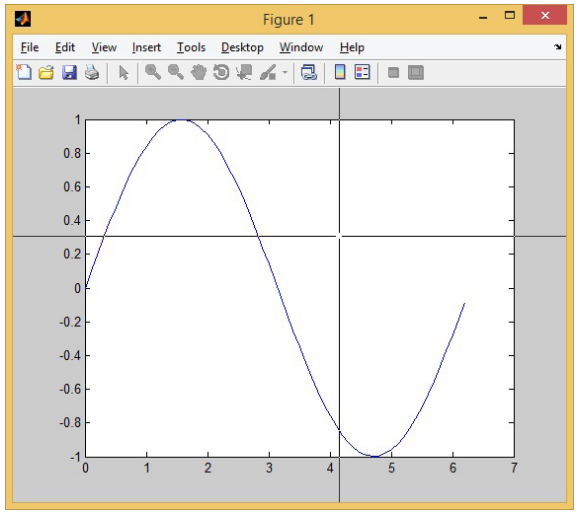


Fig. 42. Uso de la función gtext

► **Función grid:** sitúa rejillas en los ejes de un gráfico; grid on mantiene activa las rejillas y grid off las elimina.

Sintaxis: grid on

grid off

Ejemplo:

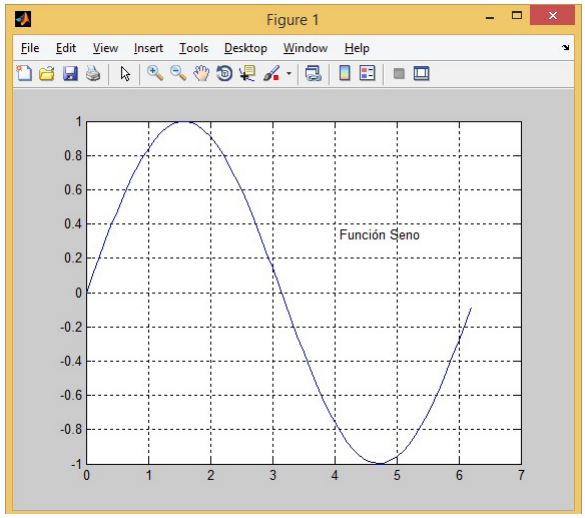


Fig. 43. Rejillas en la gráfica

► **Función hold:** activa y desactiva a superposición de gráficos en los mismos ejes (congela una gráfica).

Sintaxis: hold on

hold off

```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> z=cos(x);
>> plot(x,y)
>> hold on
>> plot(x,z)
```

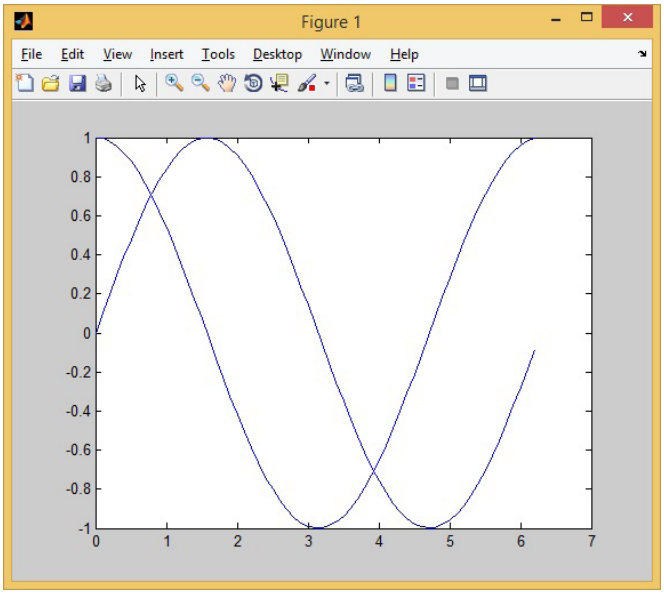


Fig. 44. Uso del comando hold on

3.5 Control de ventanas graficas

► Función figure: En Matlab se puede referir a una ventana gráfica a través del comando figure(n), donde n es el número de ventana que se desea activar. Si solo se desea activar una ventana grafica sin importar el número de orden, el comando se reduce a figure().

Ejemplo:

```
>> figure(4)
```

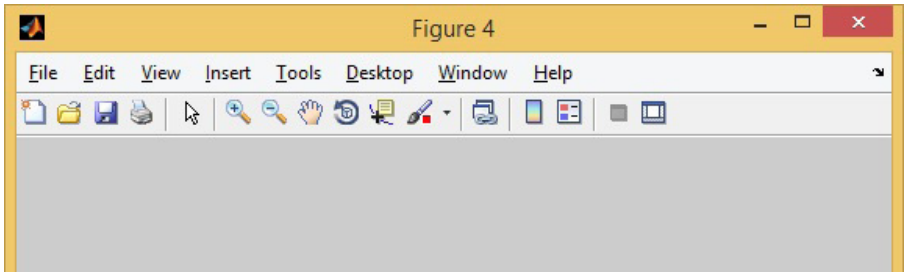


Fig. 45. Crea una nueva figura

Para desactivar la ventana gráfica, el comando close es utilizado o análogamente close(n), cierra el número de ventana correspondiente a n.

El comando clf elimina el contenido de la figura activa, es decir, la deja abierta pero vacía.

La función gcf devuelve el número de la figura activa en ese momento.

Ejemplo:

```
>> gcf

ans =
     4
```

3.6 Ejercicios resueltos

Graficar la función sen(x)+cos(2x) en el intervalo –π a π.

```
>> x=-pi:0.1:pi;
>> y=sin(x)+cos(2*x);
>> plot(x,y)
```

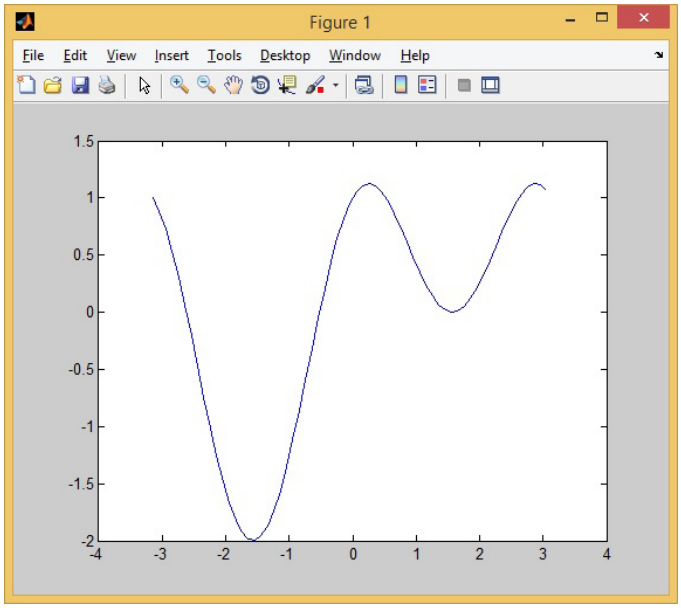


Fig. 46. Gráfica ejercicio 1

Primero se define el vector para el eje de las ordenadas en este caso el eje x desde –pi a pi, para luego calcular las funciones que se van a dibujar en el eje y, finalmente utilizamos el comando plot para dibujar los dos vectores en sistema de coordenadas.

Graficar las siguientes funciones en la misma ventana gráfica. Añada también título, ejes y cuadrícula:

$$f(x) = \begin{cases} \ln(2x-1) & (1 \le x \le 5) \\ x^2 - 1 & (-5 \le x \le 1) \end{cases}$$

```
>> x=1:0.1:5;
>> y=log(2*x+1);
>> u=-5:0.1:1;
>> v=u.^2-1;
>> plot(x,y,u,v);
>> title('Funciones a intervalos');
>> xlabel('eje x');
>> ylabel('eje y');
>> grid on;
```

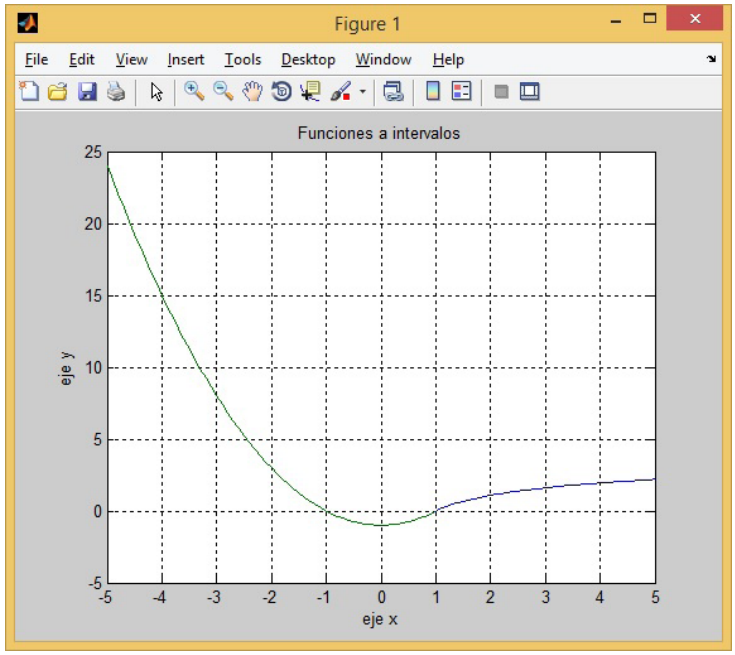


Fig. 47. Gráfica ejercicio 2

De igual manera se definen los vectores para el eje de las x y luego los vectores para el eje de las y, al utilizar los cuatro vectores en el mismo comando permite dibujar las dos funciones en la misma figura.

3. Crear un vector de 100 elementos con valores enteros aleatorios entre 0 y 100 y graficarlo con línea de color rojo y anchura 2. Incluir títulos, ejes, rejilla y leyenda.

```
>> x=randi([0 100],1,100);
>> plot(x,'r', 'linewidth',2)
>> title('Vector aleatorio')
>> xlabel('eje x'),
>> ylabel('eje y')
>> legend('vector'),
>> grid on
```

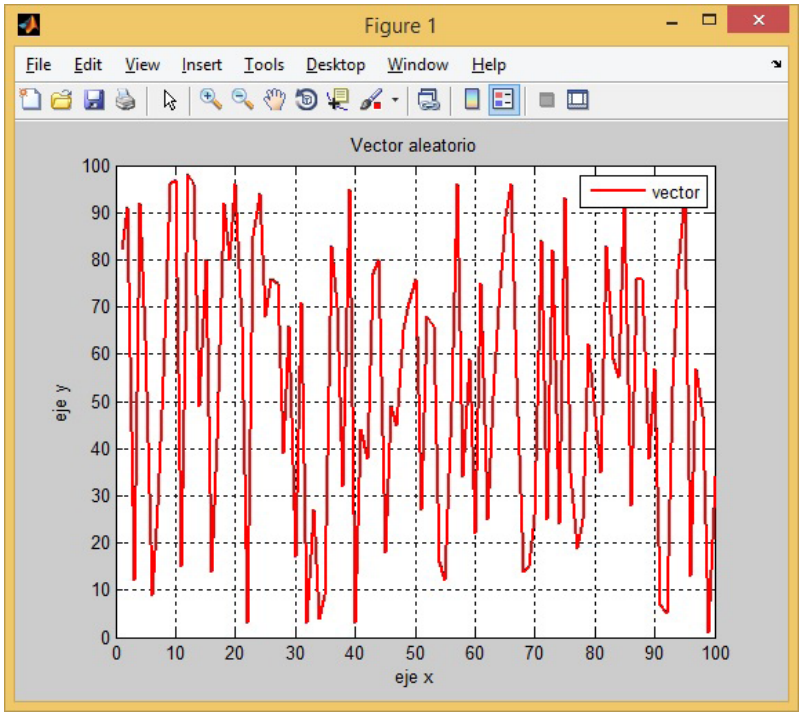


Fig. 48. Gráfica ejercicio 3

Utilizamos, en el comando plot, solo un vector. Por lo tanto, en la gráfica se dibuja el posición del elemento dentro del vector *versus* el valor del elemento, y con el comando linewidth, indicamos el ancho de la línea.

4. Con una sola instrucción utilizando el comando fplot, grafique $f(x)=x^2+2x+1$ para el intervalo $-5 \leq x \leq 5$. El gráfico debe ser con la marca + en color negro y estilo de línea a trazos.

```
>> fplot('x^2+2*x+1',[-5 5],'k+--')
>> grid
```

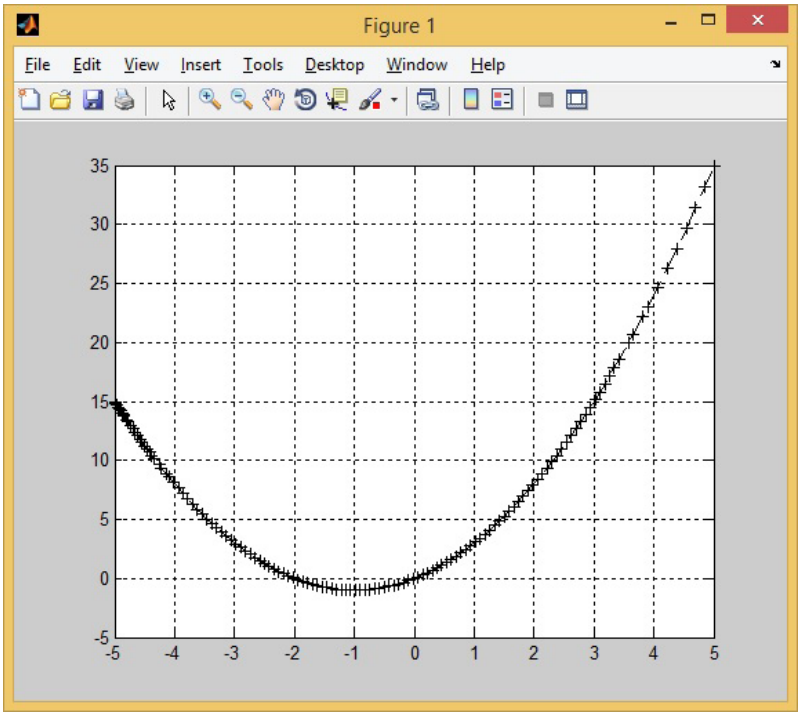


Fig. 49. Gráfica ejercicio 4

Con el comando fplot se puede dibujar directamente la función, el intervalo y las características de la misma.

5. Graficar la siguiente función $x^2+y^2=64$ en los intervalos $-10 \leq x \leq 12$ e $-12 \leq y \leq 10$. Agregar nombres a los ejes.

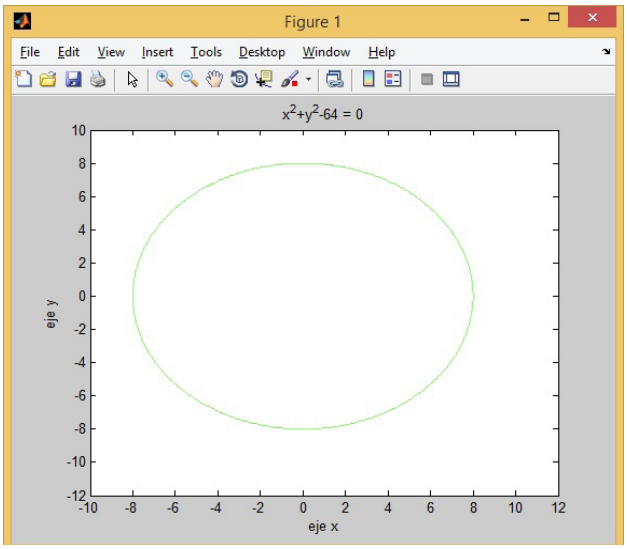


Fig. 50. Gráfica ejercicio 5

6. Graficar las siguientes funciones en los intervalos indicados:

$$f(x)\begin{cases}\tan(x) & -\frac{\pi}{4} \leq x \leq \frac{\pi}{4} \\ \cos\left(x-\frac{\pi}{4}\right) & \frac{\pi}{4} \leq x \leq \frac{\pi}{2} \\ e^x & \frac{\pi}{2} \leq x \leq 3\end{cases}$$

```
>> x1=-pi/4:0.02:pi/4;
>> y1=tan(x1);
>> x2=pi/4:0.02:pi/2;
>> y2=cos(x2-pi/4);
>> x3=linspace(pi/2,3);
>> y3=exp(x3);
>> x=[x1,x2,x3];
>> y=[y1,y2,y3];
>> plot(x,y)
>> title('Grafica de funciones')
>> xlabel('eje x'),
>> ylabel('eje y'),
>> grid on
```

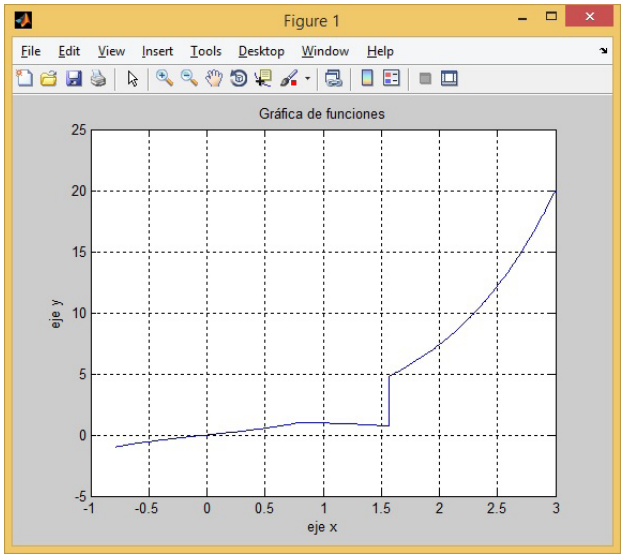


Fig. 51. Gráfica ejercicio 6

En este ejercicio creamos los diferentes vectores para cada una de las funciones tanto para el eje de las x (x1,x2,x3) como para el eje de las y (y,y2,y3), para luego agrupar estos vectores en una sola matriz lo que permitirá dibujar, mediante el comando plot cada fila de la matriz x con cada fila de la matriz y.

Graficar las siguientes funciones f1(x)=2sen(2x), f2(x)=5cos (3x), en la misma ventana. Aplicar el escalamiento independiente para el eje Y, mostrar el resultado utilizando subventanas. Observar la diferencia.

```
>> x=-2*pi:0.1:2*pi;
>> y=2*sin(2*x);
>> z=5*cos(3*x);
>> subplot(1,2,1)
>> plot(x,y,x,z)
>> title('Grafica sin escalamiento independiente del eje Y')
>> xlabel('eje X'),
>> ylabel('eje Y'),
>> grid on
>> legend('2sin(2x)','5cos(3x)')
>> subplot(1,2,2)
>> plotyy(x,y,x,z)
>> title('Grafica con escalamiento independiente del eje Y')
```

```
>> xlabel('eje X'),
>> ylabel('eje Y'),
>> grid on
>> legend('2sin(2x)','5cos(3x)')
```

Para cada una de las subgráficas, se debe primero indicar el lugar en donde se va a dibujar utilizando el subplot. En este caso, se divide la pantalla en una fila con dos columnas. En la primera se realiza la gráfica sin escalamiento y, en la segunda, con escalamiento en el eje y utilizando el comando plotyy.

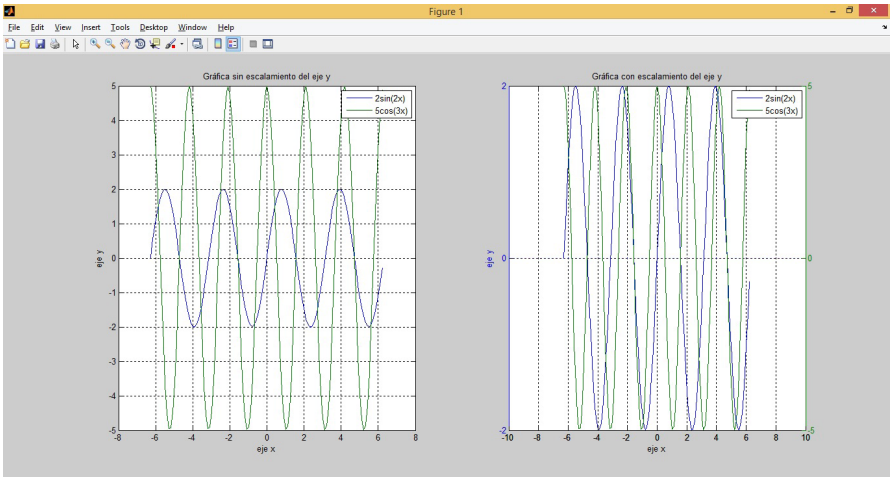


Fig. 52. Gráfica ejercicio 7

8. Graficar la función f(x)=e-x^2 con 200 puntos entre -10 y 10.

```
>> x=linspace(-10,10,200);
>> y=exp(-x.^2);
>> plot(x,y)
>> title('Funcion exponencial')
>> xlabel('eje X'),ylabel('eje Y')
>> legend('exp(-x.^2)')
```

Para definir el vector x, se utiliza el comando linspace que permite obtener 200 puntos entre -10 y 10 y, para el eje y, se dibuja la función exponencial.

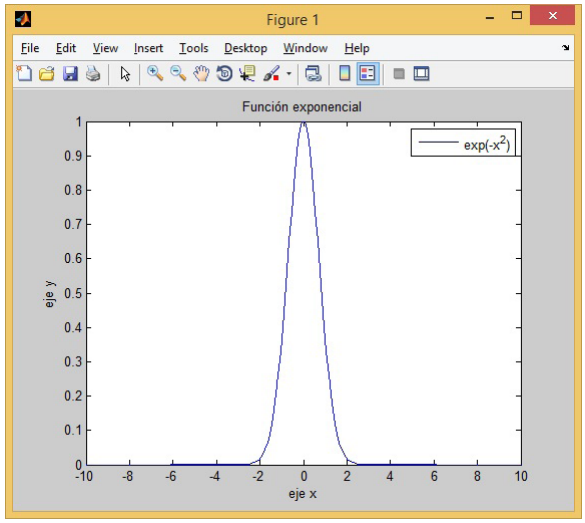


Fig. 53. Gráfica ejercicio 8

BIBLIOGRAFÍA

Arnáez, E. (2014). *Enfoque práctico del control moderno con aplicaciones en Matlab.* Lima Universidad Peruana de Ciencias Aplicadas (UPC).

Etter, D. (1998). *Solución de problemas de ingeniería con MATLAB*. México: 2.ª ed. Prentice Hall.

Gil, Rodríguez M. (2003). *Introducción rápida a Matlab y Simulink para ciencia e ingeniería.* Madrid: Díaz de Santos.

Gilat, A. (2006). *Matlab: una introducción con ejemplos prácticos*. Barcelona: 2.ª ed Reverté.

Herrera, J. (2011). *Métodos matriciales para ingenieros con MATLAB*. Cali: Sello Editorial Javeriano.

MathWorks, I. (2017). Matlab Academy. Obtenido de https://matlabacademy.mathworks.com/es

Moore, H. (2007). *MATLAB para ingenieros.* México: Pearson.

Pinto E., (2010). *Fundamentos de control con matlab.*Madrid: Pearson.

GLOSARIO DE TÉRMINOS

A

Abscisas: cordenada de dirección horizontal que aparece en un plano cartesiano rectangular.....148

Apóstrofes: forma de puntuación; son pequeños, como comas en el aire.154

Archivo binario: es un archivo informático que contiene información de cualquier tipo codificada en binario para el propósito de almacenamiento y procesamiento en ordenadores.....49

Argumento: es una variable utilizada para recibir valores de entrada o dar valores de salida.....36

Arreglos: grupo o colección finita, homogénea y ordenada de elementos..20

Asignación: cambia el valor de la variable que está a la izquierda por un literal o el resultado de la expresión que se encuentra a la derecha.....22

C

Código: conjunto de líneas de texto con los pasos que debe seguir la computadora para ejecutar un programa.....51

Conjugada: de un número complejo; se obtiene cambiando el signo de su componente imaginaria.....47

D

Desviación típica: se define como la raíz cuadrada de la varianza de la variable.....72

Directorio: es un contenedor virtual en el que se almacenan una agrupación de archi- vos informáticos y otros subdirectorios.....50

E

Entorno de Matlab: son las ventanas o diferentes secciones en las que se divide Matlab.....17

F

G

I

L

N

O

P

R

S

T

En el libro de Matlab básico se exponen las principales características de Matlab, así como también la forma correcta de manejar, almacenar y recuperar variables. A crear vectores y matrices, las operaciones y funciones que se pueden aplicar a cada uno de ellos. Cómo se deben ingresar ecuaciones, polinomios y sistemas de ecuaciones, de manera que se resuelvan rápidamente. Además de ello se presentan las principales funciones para realizar gráficas en dos dimensiones, manejo de gráficas y subgráficas. En cada uno de los capítulos se resuelven ejercicios relacionados a los temas explicados.

Alexandra Orfelina Pazmiño Armijos, ingeniera electrónica en Computación y tecnóloga en Informática Aplicada, especialista en redes de comunicación de datos, magíster en Informática Empresarial, docente facultad de Mecánica de la Escuela Superior Politécnica de Chimborazo, miembro del grupo de Investigación y Estudios en Bioingeniería.

Jairo René Jácome Tinoco, ingeniero electrónico en Computación, tecnólogo en Informática Aplicada, magíster en Sistemas de Telecomunicaciones, técnico docente de la Facultad de Mecánica de la Escuela Superior Politécnica de Chimborazo, miembro del Grupo de Investigación y Estudios en Bioingeniería.

Luis Alberto Zabala Aguiar, ingeniero en Electrónica, Control y Redes Industriales, magíster en Sistemas de Telecomunicaciones, ha desarrollado proyectos de control industrial y en el sector petrolero, docente en las asignaturas Sistemas de Control, Instrumentación y sensores y Electrónica de potencia.

Javier J. Gavilanes C., ingeniero en Electrónica, Control Automático y Redes Industriales. máster universitario en Automática y Robótica. Adjudicatario Beca de Estudios Cuarto Nivel. Docente de la Facultad de Mecánica Escuela de Ingeniería Automotriz. Miembro del Grupo de Investigación y Estudios en Bioingeniería.

ISBN: 978-9942-35-348-1