UNIVERSIDAD DE IBAGUÉ - CORUNIVERSITARIA

PROGRAMA DE INGENIERÍA MECÁNICA

Guía de MATLAB para Ing. Mecánica

Ing. Alfonso Cubillos Varela
Ing. Mecánico
Ibagué, 2007
(Ver 0.5)

Índice general

Introducción

Los grandes avances de la ingeniería moderna se han apoyado enormemente en el uso de las herramientas computacionales, estas facilitan y agilizan los procesos iterativos de cálculo, permiten realizar simulaciones y el manejo óptimo de gran cantidad de información. Aunque comercialmente se encuentran paquetes computacionales que realizan labores muy especializadas en ingeniería (AutoCAD, Solid Edge, Working Model, etc.) es necesaria la formación del ingeniero en el manejo de las *estructuras básicas* de programación. Estas le permiten solucionar problemas específicos y generar sus propios programas o algoritmos de solución.

Las herramientas programables, también son muy útiles en el proceso de formación del ingeniero, por que se reduce el tiempo de cálculos manuales, permitiendo un mejor análisis y comprensión de los problemas. Es normal cometer errores mientras se realiza cualquier calculo, por lo tanto es necesario revisar de nuevo el problema, y esto implica realizar de nuevo todas las operaciones del mismo, duplicando el tiempo necesario para realizar el ejercicio o resolver el problema. De esta forma, la aplicación de algoritmos computacionales permite identificar fácil y rápidamente errores obtenidos en el proceso de análisis; y en los procesos de diseño, permiten realizar iteraciones en busca de la mejor opción (optimización).

Tomando estas consideraciones se presenta este manual, el cual se utiliza como base en el curso de Lenguaje de Programación, pero puede servir como guía en cualquier semestre de la carrera como libro de referencia. El autor selecciono como software de trabajo MATLAB, aunque los conceptos tratados pueden ser aplicados sobre la mayoría de los lenguajes de programación.

MATLAB es un software para realizar cálculos en ingeniería, ciencias y matemáticas aplicadas. Este ofrece un potente lenguaje de programación, excelentes gráficas y un amplio rango de aplicaciones científicas especializadas. MATLAB es publicado y una marca registrada de The MatWorks, Inc.

El foco de MATLAB esta en los *cálculos*, no en la matemática. Ahora que las expresiones simbólicas y su manipulación no son posibles (excepto a través de una interface con Maple). Todos los resultados son solamente numéricos y por lo tanto inexactos, gracias al error de redondeo inherente a la aritmética computacional. La limitación del calculo computacional puede ser visto como un inconveniente, pero MATLAB la evita al correr algunos ciclos sobre Maple y Mathematica, y luego él los convierte a soluciones numéricas.

Por otro lado, comparándolo con otros lenguajes como C++ y FORTRAN, MATLAB es mucho mas fácil de usar y viene con una gran librería de funciones estándar. La única y mas grande desventaja en comparación, es un problema en la velocidad de ejecución. Este vacío puede no ser tan dramático como muchos lo creen, pues puede ser disminuido o eliminado con un buen programa en MATLAB, aún así es necesario aceptar que MATLAB no es la mejor herramienta para cálculos de alta eficiencia.

Este documento fue realizado en LaTeX.

Capítulo 1

Visión Informática

Este capitulo presenta los comandos básicos del programa, los tipos de datos, el manejo de variables y operadores, así como el uso de funciones elementales. Debido a que esta es la información básica, es necesario que el estudiante posea, al final de capítulo, la comprensión clara de cada uno de los temas tratados.

1.1. Generalidades y fundamentos de MATLAB

MATLAB básicamente se puede tomar como una calculadora, aunque es esto y mucho más. En la ventana inicial de MATLAB aparece el *prompt* (aviso) característico del programa `(>> )`. Esto quiere decir que el programa esta *listo para recibir instrucciones*, y es sobre este donde se digitan y ejecutan los comandos en MATLAB.

Si se escribe una expresión válida y presiona Enter, MATLAB la ejecuta inmediatamente y devuelve el resultado.

```
>> 2+2              >>  4 * 2
ans =               ans =
    4                     8
```

Algunos problemas en ingeniería se solucionan únicamente con operaciones simples como sumas y restas, pero la gran mayoría requieren el uso de otros datos y operadores, por ejemplo operaciones vectoriales o matriciales, el manejo de complejos, obtener la potencia o la raíz de un valor, etc.

1.2. Tipos de datos

Se denomina *Dato* a una parte de la información. Llevan a una conclusión o información. Se pueden declarar comúnmente con los siguientes tipos.

1.2.1. Datos numéricos

Son los que contienen números únicamente. Es obvio entonces que están compuestos de dígitos 0 al 9. MATLAB trabaja siempre en *doble precisión*, es decir guardando cada dato en 8 bytes, con unas 15 cifras decimales exactas.

Enteros : No tienen parte decimal, solo tienen parte real y pueden ser positivos o negativos.

Real : Tienen parte decimal y parte entera. Este valor puede ser positivo o negativo. Para definir un número real en MATLAB, se separa la parte entera de la decimal con un punto (.) como se muestra:

```
>> -3.56
```

1.2.2. Datos String o Alfanuméricos

Son datos cuyo contenido puede ser un caracter o una cadena (sucesión) de caracteres.

Caracter : Cualquier letra (de la `A` a la `Z` ó `a` a la `z`), número (0 al 9) o el signo *underline* (_)

Cadena de caracteres : Se define como una sucesión de varios caracteres.

Para que el computador reconozca los datos los datos alfanuméricos como string, estos se deben encerrar en comillas sencillas (`' '`). Los espacios en blanco que se encuentran dentro de una cadena de caracteres también son contados como caracteres.

```
>> 'Hola'
ans =
       Hola

>> '2+3'
ans =
       2+3
```

1.2.3. Datos lógicos o booleanos

Son datos que solo pueden tomar dos valores: verdadero (true ó 1) o falso (false ó 0).

1.2.4. Imaginarios o complejos

En muchos cálculos matriciales los datos y/o los resultados no son reales sino *complejos*, con *parte real* y *parte imaginaria*. MATLAB trabaja sin ninguna dificultad con números complejos. Para ver como se representan por defecto los números complejos, ejecútense los siguientes comandos:

```
>> sqrt(-4)  % Respuesta compleja
ans =
       0   +   2.0000i

>> 3 + 4j    % Variable compleja definida
ans =
   3.0000  +   4.0000i
```

En la entrada de datos de MATLAB se pueden utilizar indistintamente la **i** y la **j** para representar el *número imaginario unidad* (en la salida, sin embargo, puede verse que siempre aparece la **i**).

Si la **i** o la **j** no están definidas como variables, puede intercalarse el signo (*). Esto no es posible en el caso de que sí estén definidas, porque entonces se utiliza el valor de la variable. En general, cuando se está trabajando con números complejos, conviene no utilizar la **i** como variable ordinaria, pues puede dar lugar a errores y confusiones. Por ejemplo, obsérvense los siguientes resultados:

```
>> i = 2;
>> 2 + 3i
ans =
   2.0000   +   3.0000i
```

```
>> 2 + 3*i
ans =
       8

>> 2 + 3*j
ans =
   2.0000  +   3.0000i
```

MATLAB dispone asimismo de la función complex, que crea un número complejo a partir de dos argumentos que representan la parte real e imaginaria `complex(1,2)`.

1.3. Representación de datos

En programación los datos se representan con un nombre, el tamaño y la forma de escribir el nombre depende del lenguaje. El nombre que representa el dato comúnmente se clasifica como una *variable* o como una *constante*.

1.3.1. Constante

Es el nombre que se le da a un campo cuyo contenido no cambia o varia a través del proceso. MATLAB tiene definidas por defecto algunas constantes útiles en ciertos cálculos, es el caso de `pi` para π e `i` para $\sqrt{-1}$.

```
>> sin(pi/2)
ans =
    1
```

```
>> exp(i*pi)
 ans =
      -1.0000 + 0.0000i
```

MATLAB mantiene una forma especial para los *números muy grandes* (más grandes que los que es capaz de representar), que son considerados como *infinito*. Por ejemplo, obsérvese como responde el programa al ejecutar el siguiente comando:

```
>> 1.0 / 0.0
Warning: Divide by zero
ans =
       Inf
```

Así pues, para MATLAB el *infinito* se representa como `inf` o `Inf`. MATLAB tiene también una representación especial para los resultados que no están definidos como números. Por ejemplo, ejecútencen los siguientes comandos y obsérvensen las respuestas obtenidas:

```
>> 0/0
Warning: Divide by zero
ans =
       NaN

>> inf/inf
ans =
       NaN
```

En ambos casos la respuesta es `NaN`, que es la abreviatura de *Not a Number*. Este tipo de respuesta, así como la de `Inf`, son enormemente importantes en MATLAB, pues permiten controlar la fiabilidad de los resultados de los cálculos matriciales. Los `NaN` se propagan al realizar con ellos cualquier operación aritmética, en el sentido de que, por ejemplo, cualquier número sumado a un `NaN` da otro `NaN`. Algo parecido sucede con

los `Inf`.

MATLAB dispone de tres constantes útiles relacionadas con las operaciones de coma flotante. Estas funciones, que no tienen argumentos, son las siguientes:

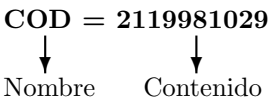
`eps`	devuelve la diferencia entre 1.0 y el número de coma flotante inmediatamente superior. Da una idea de la precisión o número de cifras almacenadas. En un PC, eps vale 2.2204e-016.
`realmin`	devuelve el número más pequeño con que se puede trabajar (2.2251e-308)
`realmax`	devuelve el número más grande con que se puede trabajar (1.7977e+308)

1.3.2. Variable

Una variable es el nombre que se le asigna a un campo, cuyo contenido puede cambiar o variar a través de un proceso. El *nombre* y el *contenido* son las partes de una variable.

El **nombre de una variable** es el conjunto de caracteres (letras, números y caracteres especiales) aceptados, con los cuales se identifica el contenido de un campo en un momento determinado.

El **contenido de una variable** es el dato (Sección 1.2) que una variables representa o tiene asociada en un determinado momento.



Normas para definir una variable : Son las normas que se sugieren para la asignación de nombres de variables, estas son:

1. El nombre debe comenzar por una letra y no por un número
2. No deben existir espacios entre las letras del nombre.
3. No deben poseer caracteres especiales, excepto el underline (_) la cual se toma como *sub*.
4. A semejanza de C, *MATLAB distingue entre mayúsculas y minúsculas* en los nombres de variables.
5. El nombre debe ser mnemotécnico, es decir que tenga relación con su contenido.
6. Se sugiere que el nombre no exceda 8 caracteres pero puede contener hasta 21.
7. No se deben utilizar palabras o funciones reservadas del programa, por ejemplo, si usa `>> pi = 8`, el nuevo valor de la constante π será 8 y no el definido por el programa.

Las reglas de la 1 a la 4 son obligatorias, las otras son sugerencias que facilitan el uso del programa.

Asignación del contenido de una variable Algunos programas difieren en la forma como asignan el contenido de una variable. En MATLAB se utiliza la estructura mostrada en la figura superior, es decir de la forma:

```
>> COD = 2119981029
COD =
       2119981029

>> a = 2 + 3
a =
     5
```

MATLAB toma el valor (u operación) de la derecha y lo asigna a la variable de la izquierda. En el caso de ser un string o un complejo, se siguen las estructuras ya definidas:

```
>> x = sqrt(3)
x =
   1.7321

>> z  = sqrt(-8)
z =
     0 + 2.8284i

>> y = 4 + 5j
y =
    4 + 5i

>> mensaje = 'Hola'
mensaje =
           Hola

>> 3* x
ans =
       5.1961

>> 3*b
??? Undefined function or variable 'b'.
```

Cuando una expresión devuelve un único resultado que no es asignado a ninguna variable, este resultado es asignado a la variable `ans` (*answer*), la cual puede ser usadas luego para otra operación como una variable más.

Observe que *las variables deben poseer valores antes de ser usadas*. Cuando la variable a utilizar (b) aún no ha sido definida, se genera el mensaje de error `??? Undefined function or variable 'b'.` Este es el error más común, por lo tanto, lea el mensaje de error y verifique que la variable ha sido definida de forma correcta.

Cuando no se recuerda el valor de una variable, se le puede preguntar a MATLAB:

```
>> COD
COD =
       2119981029
```

1.3.3. Expresión Aritmética o Algebraica

Es el conjunto de variables y/o constantes unidas por operadores aritméticos:

$$Y = X^3 + X^2 + 6$$

Vrble Vrble Cnste

1.4. Operadores

El *operador* es un símbolo o palabra que permite formular operaciones, establecer relaciones y hacer comparaciones de tipo lógico matemático entre datos. Significan que se ha de realizar cierta acción u operación con base en algún o algunos valores.

Los operadores se dividen principalmente en tres tipos:

1. Operadores Aritméticos
2. Operadores Relacionales
3. Operadores Lógicos

1.4.1. Operadores Aritméticos

La tabla 1.1 resume algunos de los operadores aritméticos posibles en MATLAB.

Tabla 1.1: Operadores Aritméticos

Operador	Operación
+	suma
-	resta
*	multiplicación
/	división derecha
\	división izquierda
^	potenciación

El operador separa los dos datos entre los cuales se realiza la operación, y el programa devuelve el resultado de la operación:

```
>> 3 + 4
ans =
      7

>> 3 - 4
ans =
     -1

>> 3 * 4
ans =
      12

>> 3 / 4
ans =
      0.7500

>> 3 ^ 4
ans =
      81
```

1.4.2. Operadores Relacionales

Los operadores de relación que se utilizan en MATLAB, los resume la tabla 1.2.

Tabla 1.2: Operadores Relacionales

Operador	Operación
==	igual que
~=	no igual
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que

La solución en este tipo de operaciones, es una repuesta de tipo lógico, es decir, se utiliza el valor de 1 como *verdadero* y 0 como *falso*.

```
>> 5 == 5          >> 2 > 3        >> 2 <= 3
ans =               ans =           ans =
       1                   0               1
```

1.4.3. Operadores Lógicos

Tabla 1.3: Operadores Lógicos

Operador	Función	Operación
&	and(x,y)	and lógico
|	or(x,y)	or lógico
~	not(x,y)	not lógico
	xor(x,y)	or exclusivo

Para las operaciones lógicas, se utilizan los operadores o funciones que muestra la tabla 1.3. Es necesario utilizar valores lógicos en las operaciones, y de la misma forma, el resultado también es un valor lógico. Estas operaciones se basan en las tablas de verdad vistas en el curso de lógica.

```
>> 1&0             >> 1 | 0        >>~1
ans =               ans =           ans =
       0                   1               0
```

1.5. Orden de las operaciones

Al realizar cálculos con los operadores aritméticos, lógicos y/o relacionales, es necesario tener en cuenta el orden en que el computador realiza las operaciones, pues esto influye directamente en el resultado obtenido.

1. Operaciones de Izquierda a derecha

Cuando en una expresión aritmética se utilizan varios datos y operadores, estas se realizan de izquierda a derecha:

$$Y = \underbrace{4 * 5}_{\downarrow} * 6 \quad \downarrow$$
$$Y = \underbrace{20 * 6}_{\downarrow}$$
$$Y = 120$$

```
>> 4*5*6
ans =
     120
```

Debido a la propiedad conmutativa de la mayoría de los operadores aritméticos, esto no es mayor problema, pero veamos otro ejemplo:

$$X = \frac{3}{4} * 5$$

$$X = \underbrace{3/4}_{\downarrow} * 5 \quad \downarrow$$
$$X = \underbrace{0{,}75 * 5}_{\downarrow}$$
$$X = 3{,}75$$

```
>> X = 3 / 4 * 5
X =
    3.7500
```

En este caso, se realiza primero la división, y luego el resultado es multiplicado por 5. Note que este resultado es muy diferente al de multiplicar primero $4 * 5 = 20$ y luego dividir $3/20 = 0{,}15$ el cual es, desde luego, un resultado erróneo.

2. Jerarquía de las operaciones

Los operadores aritméticos poseen una jerarquía, y esta modifica el orden en que se realizan las operaciones. La jerarquía natural de los operadores aritméticos es la siguiente:

`^`	alta jerarquía
`*,/`	media jerarquía
`-,+`	baja jerarquía

El significado de la jerarquía es que las operaciones de más alta jerarquía se realizan primero:

$X = 4+6*15$

$X = 4 + \underbrace{6 * 15}$

$X = \underbrace{4 \ + \ 90}$

$X = \quad 94$

```
>> X = 4 + 6 * 15
X =
     94
```

y las operaciones de igual jerarquía se evalúan en el orden que aparecen de izquierda a derecha:

$Y = 5+7*3+2*4$

$Y = 5 + \underbrace{7 * 3} + \underbrace{2 * 4}$

$Y = \underbrace{5 \ + \ 21 \ + \ 8}$

$Y = \quad 34$

```
>> Y = 5+7*3+2*4
Y =
     34
```

Analice las siguientes ecuaciones, y determine cuál es el orden de las operaciones debido a la jerarquía de las mismas:

$-4 * 7 + \frac{2^3}{4} - 5 \quad \longrightarrow \quad$ `-4 * 7 + 2^3 / 4 -5 = -31`

$x^2 + \frac{2y}{3} + c - 3d \quad \longrightarrow \quad$ `x^2+2*y/3+c-3*d`

3. Uso del paréntesis

Cuando una expresión aritmética posee paréntesis, se deben tener en cuenta los siguientes parámetros:

- El computador ejecuta primero las operaciones que estén dentro del paréntesis ( ).
- Si existen varios pares de paréntesis, comienza a realizar el más interno hasta llegar al externo.
- Dentro del paréntesis se sigue le jerarquía normal de operaciones.

$Z = \frac{3}{4*5}$

$Z = 3 \ / \underbrace{(4 * 5)}$

$Z = \underbrace{3 \ / \ 20}$

$Z = \quad 0{,}15$

```
>> Z = 3 / (4 * 5)
Z =
     0.1500
```

De la misma forma,

$$X = \frac{4+6}{20} \qquad X = \underbrace{(4+6)}_{\downarrow}\ /20 \qquad X = \underbrace{10\ /\ 20}_{\downarrow} \qquad X = 0{,}5$$

```
>> X = (4 + 6) / 20
Z =
    0.1500
```

Analice la siguiente ecuación, y determine cuál es el orden de las operaciones debido a la jerarquía de las mismas y el uso del paréntesis:

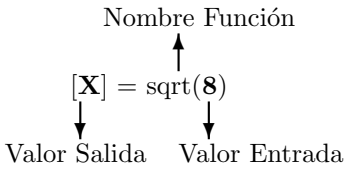
$\frac{x^2+2y}{3+c} - 3d \quad \longrightarrow \quad$ `(x^2+2*y)/(3+c)-3*d`

Cuando se realizan divisiones, se recomienda primero colocar los dos pares de paréntesis separados por el operador de división `( )/( )`. En el primer par de paréntesis se ubica el numerador y en el segundo el denominador. Esto garantiza que la división se realiza correctamente especialmente cuando la operación es muy larga o compleja.

1.6. Uso de funciones matemáticas

Hasta el momento hemos visto y realizado operaciones sencillas, como sumas, restas, multiplicaciones y similares, pero ¿cómo se realizan operaciones trigonométricas, logarítmicas, hiperbólicas, etc.? Este tipo de operaciones se pueden realizar gracias a las *funciones*.

Las *funciones* poseen tres elementos principales: El (los) *valor(es) de salida*, el *nombre de la función* y el (los) *valor(es) de entrada*



El nombre de la función es la palabra reservada que se utiliza para 'invocar' la función y realizar las operaciones para las cuales se ha creado la misma. El nombre de la función tiene mucho que ver con la operación que esta realiza, con el fin de facilitar la memorización (nemotécnica).

La función posee un valor (o valores) de entrada, el cual toma para realizar los cálculos u operaciones necesarias, estos cálculos generan un resultado y son 'devueltos' como variable(s) de salida de la función.

El valor(es) de entrada es el dato(s) que toma la función para realizar los cálculos necesarios y se debe especificar al lado derecho del nombre de la función, definidos por *paréntesis*. El valor de entrada puede ser un dato (numérico, string, booleano o complejo) ó el nombre de una variable que contiene un dato. En algunas ocasiones es necesario especificar varios valores de entrada, en este caso, cada uno de los valores de entrada se separan por una coma (,).

El valor(es) de salida es la respuesta o dato resultado que la función ofrece; se ubica al lado izquierdo del signo igual (=) y debe ir encerrado entre corchetes cuadrados [ ]. En este caso se define el nombre(s) de la(s) variable(s) en el cual(es) se van a guardar los datos de salida o respuestas de la función. Cuando el valor de salida de la función es un solo valor, no es necesario encerrar la respuesta en corchetes cuadrados, pero cuando la función 'devuelve' varios valores, es necesario encerrarlos en los corchetes y separar el nombre de las variables por comas (,).

La tabla 1.4, muestra algunas de las funciones matemáticas especiales que posee MATLAB, junto a la descripción de la misma. Utilice el formato especificado para cada una.

Tabla 1.4: Funciones Matemáticas Especiales

Funciones Trigonométricas	
`sin`(x)	seno en radianes
`cos`(x)	coseno en radianes
`tan`(x)	tangente
`sec`(x)	secante
`csc`(x)	cosecante
`cot`(x)	cotangente
`asin`(x)	inversa del seno
`acos`(x)	inversa del coseno
`atan`(x)	inversa del tangente
`sinh`(x)	seno-hyperbólico
Funciones de conversión Decimal-Entero	
`round`(x)	redondea hacia el entero más próximo
`ceil`(x)	redondea hacia el infinito
`fix`(x)	redondea hacia cero
`floor`(x)	redondea hacia menos infinito

Funciones elementales	
`abs`(x)	valor absoluto $\|x\|$
`sign`(x)	devuelve el signo del argumento
`exp`(x)	exponencial e^x
`log`(x)	logaritmo natural $log_e(x)$
`log10`(x)	logaritmo decimal $log_{10}(x)$
`sqrt`(x)	raíz cuadrada $\sqrt{x}$
Funciones de manipulación de complejos	
`z =` `complex`(a,b)	forma un número complejo
`abs`(z)	magnitud de un número complejo
`angle`(z)	ángulo de un número complejo
`conj`(z)	complejo conjugado
`imag`(z)	parte imaginaria del complejo
`real`(z)	parte real de un número complejo

1.7. Espacio de trabajo

El espacio de trabajo de MATLAB (*workspace*) es el conjunto de variables y funciones de usuario que en un momento determinado están definidas en la memoria del programa. Para obtener información sobre el workspace se pueden utilizar los comandos `who` y `whos`. El segundo proporciona información más detallada que el primero.

```
>> who

Your variables are:

ans s t x
```

Comando *clear*

El comando `clear` se utiliza para borrar variables del espacio de trabajo, y presenta las siguientes variantes:

`clear`	sin argumentos, elimina todas las variables
`clear s,t`	borra las variables indicadas
`clear functions`	borra las funciones
`clear all`	borra todas las variables y funciones

1.8. Uso del *help*

MATLAB posee diferentes fuentes de ayuda, la primera y mas sencilla es cuando no se recuerda o no se conoce el nombre exacto de la función a utilizar, el comando `lookfor` seguido por una palabra clave, busca todas las funciones que mencionan la palabra clave especificada:

```
>> lookfor variables
CLEAR  Clear variables and functions from memory.
LOAD   Load workspace variables from disk.
SAVE   Save workspace variables to disk.
WHO    List current variables.
```

```
WHOS   List current variables, long form
```

y para obtener detalles de la función se utiliza el comando `help`:

```
>> help whos
 WHOS List current variables, long form.
   WHOS is a long form of WHO.  It lists all the variables in
   the current workspace, together with information about their
   size, bytes, class, etc.
   ...
```

Además de los anteriores comandos en linea, se pueden utilizar los manuales de usuario en formato PDF que acompañan el CD de instalación de MATLAB. Tutoriales y páginas de internet son del mismo modo una buena fuente de información, y de funciones ya realizadas para ser evaluadas y aprender nuevos trucos.

1.9. Scripts

Un *M-file* es un archivo de texto regular que contiene comandos de MATLAB, el cual se guarda con la extensión **.m*. Hay dos clases de M-files: *scripts* y *funciones*. La importancia de estos *ficheros-M* es que al teclear su nombre en la linea de comandos y pulsar **Enter**, se ejecutan uno tras otro todos los comandos contenidos en dicho fichero. Guardar instrucciones y grandes matrices en un fichero, permite ahorrar mucho trabajo tecleando.

Un script es mayormente útil como "seguidor" para múltiples tareas. Los comandos en un script son interpretados literalmente como si ellos fueran digitados en el prompt (línea por línea).

MATLAB viene con un muy buen editor el cual se puede integrar al espacio de trabajo, sin embargo se puede usar cualquier otro editor de texto. El editor de MATLAB muestra con diferentes colores los diferentes tipos o elementos constitutivos de los comandos (en *verde* los comentarios, el *rojo* las cadenas de caracteres, etc.). El editor también ayuda a determinar donde cierran los paréntesis y comillas para evitar errores.

Un archivo *script* es usualmente utilizado en los casos donde:

1. El programa contendrá más de algunas líneas de código, y es necesario revisar la secuencia de comandos.
2. El programa y sus cálculos serán usados después.
3. Se desea un registro permanente de los cálculos.
4. Se espera que la actualización ocasional sea requerida.
5. Se requiere la depuración profunda del programa.
6. El programa o cálculos serán transferidos a otra persona u organización.

Adicionalmente, un script o función usualmente contiene los siguientes elementos:

1. *Documentación,* contiene la información principal del programa para su clasificación o recuperación. Permite que otras personas conozcan que hace el script, o recordarlo fácilmente. La documentación debe indicar al menos:
   a) El objetivo u operaciones que realiza el script o función
   b) El nombre y código de los programadores
   c) Fecha de creación
   d) Las variables de entrada: nombre, descripción y tipo.

e) Las variables de salida: nombre, descripción y tipo.

2. *Definición de constantes,* son aquellos datos que no van a cambiar durante la ejecución del programa.

3. *Inicialización de Variables,* permite definir claramente cuales son las variables con las cuales se pueden realizar todos los cálculos, y claro, asignarles un valor. Esto permite modificar rápidamente el valor de cualquier variable si se hace necesario.

4. *Cálculos,* donde se realizan las operaciones necesarias.

5. *Variables de salida,* donde los resultados son presentados como gráficas y/o valores numéricos.

Un M-file debe estar guardado en el *path* actual para ser ejecutado. El path es solo una lista de directorios (carpetas) en las cuales MATLAB busca los archivos. Usando `editpath` o los menus, se puede ver o cambiar el path actual.

No es necesario compilar los archivos de tipo M-file. Simplemente digite el nombre del archivo (sin extensión) para ejecutarlo. El primer bloque de comentarios en un M-file, sirve como documentación para el archivo y puede ser visto en el prompt si el comando `help` es usado con el nombre del archivo.

Por ejemplo, generar un programa que calcule el volumen de un liquido que esta contenido dentro de un recipiente cilíndrico de radio `2 cm`; el nivel de liquido puede cambiar, pero actualmente se encuentra a `10 cm`.

Solución : Primero, se crea sobre el path actual un archivo con el nombre `VolumenLiquido.m`, y el código del programa podría ser algo como:

```
 % Calcular Volumen liquido en recipiente cilindrico
 % Ing. Alfonso Cubillos V.
 % Agosto 2006
 % Vbles entrada: h (nivel de liquido)
 % Vbles salida: V (Volumen del liquido)

% Constantes
r = 2;  % Radio del recipiente [cm]

% Variables
h = 10; % Altura del liquido [cm]

% Calculos
A = pi*r^2; % Area base del cilindro [cm^2]
V = A*h;    % Volumen de liquido    [cm^3]

% Variable de salida o solucion
V
```

Luego, en el prompt se puede verificar el comentario con el comando `help` y el comando `lookfor` busca la palabra clave en la primera linea del comentario del script.

1.10. Tips Especiales

Antes de continuar, es posible recuperar comandos anteriores de MATLAB y moverse por dichos comandos con la teclas-flechas ↑ y ↓. Al pulsar la primera de dichas flechas aparecerá el comando que se había introducido

inmediatamente antes. De modo análogo es posible moverse sobre la linea de los comandos con las teclas ← y →, ir al principio de la linea con *inicio*, al final con *fin* y borrar toda la línea con *Esc*.

Para borrar todas las salidas anteriores de MATLAB y dejar limpia la ventana principal se pueden utilizar las funciones `clc` y `home`. La función `clc` (*Clear Console*) elimina todas las salidas anteriores, mientras que `home` las mantiene, pero lleva el *prompt* `(>> )` a la primera linea de la ventana.

Aquí se presentan algunos comandos comunes y se explicaran con detalle más adelante.

```
>> % Esto es un comentario
>> x = rand(100,100); % El ; significa ‘‘no muestre el resultado’’
>> t = 1 + 2 + 3 + ...
4 +5 + 6              % ... continua la linea

t =
   21
```

Comentarios (%)

Cuando se desea agregar información adicional acerca del programa u operaciones realizadas, se agrega un *comentario*. El comentario debe ir precedido por el símbolo de porcentaje (`%`). Cualquier texto en la misma linea después del símbolo de porcentaje es ignorado por MATLAB (A menos que `%` aparezca como parte de un texto(string) en comillas).

Mostrar o No mostrar resultado (;)

MATLAB muestra la respuesta a la operación pedida en el momento de realizado el cálculo. Para evitar que el programa muestre en la pantalla el resultado, se utiliza al final de la operación, el símbolo *punto y coma* (;).

Operación de más de una línea (. . .)

Se presentan momentos en los cuales la operación es muy larga y no se puede realizar en la misma línea de código. Para esto se utiliza el operador *tres puntos* (. . .) que le indican al programa que la operación continua en la siguiente línea.

1.11. Ejercicios

Entregue un *script* que realice los siguiente ejercicios, recuerde documentar muy buen el script. Este archivo se debe llamar *Ejer01Gr#.m*

1. Dados los valores `a=2; b=3; c=4; d=5;` evalúe las siguientes expresiones para obtener el valor de e y compárela con la respuesta real. Escriba al lado de la ecuación el código necesario para obtener la solución:

$$a)\ e = a^a + b - \frac{c}{d} =$$
$$b)\ e = \frac{a^{a+b} - c}{d} =$$
$$c)\ e = a^a + \frac{b - c}{d} =$$
$$d)\ e = a^a * b - \frac{c * c}{d} + \frac{a}{b} * c =$$
$$e)\ e = a + \frac{b}{c + \frac{a}{d}} =$$
$$f)\ e = \sqrt{\frac{\sqrt{a+b}}{c^d * a}} =$$
$$g)\ e = \text{sen}^2\, a + \cos^2 a =$$
$$h)\ e = \text{sen}\, a^2 + \cos a^2 =$$

2. Escriba un script que resuelva el siguiente sistema con dos ecuaciones y 2 incógnitas

$$a * x + b * y = c$$
$$d * x + e * y = f$$

Comprobar la solución con los siguientes valores:

a = 2;	b = 4;	c = 10;	d = 1;	e = -1;	f = 5;	x = 5	y = 0
a = 1;	b = 5;	c = 3;	d = 4;	e = 9;	f = 2;	x =	y =
a = 1;	b = 5;	c = 3;	d = 4;	e = 8;	f = 2;	x =	y =

3. La nomenclatura de los engranajes rectos se indica en la figura 1.1. La *circunferencia de paso* es un círculo teórico en el que generalmente se basan todos los cálculos; su diámetro es el *diámetro de paso.*

El *paso circular* p es la distancia, medida sobre la circunferencia de paso, entre determinado punto de un diente y el correspondiente de uno inmediato. De manera que el paso circular es igual a la suma del *grueso del diente* y el *ancho del espacio* entre dos consecutivos.

El *módulo* m es la relación o razón del diámetro de paso al número de dientes. La unidad de longitud que habitualmente se utiliza es el milímetro. El módulo es el índice del tamaño de los dientes en el sistema SI.

El *adendo* a es la distancia radial entre el tope del diente (o la circunferencia de adendo) y la circunferencia de paso. El *dedendo* b es la distancia radial entre el fondo del espacio (o la circunferencia del dedendo) y la circunferencia de paso. La *altura total* h_t de un diente es la suma del adendo y el dedendo.

La siguiente tabla resume las principales variables de la geometría de un diente:

d	diámetro de paso	r	radio de paso
p	paso circular	m	módulo
r_b	radio base	a	adendo
b	dedendo	N	número de dientes
t	grueso del diente	ϕ	ángulo de presión (20 grados)

Las ecuaciones que relacionan estas variables son:

$$p = \pi m \qquad r_b = r\cos\phi$$
$$a = m \qquad m = d/N$$
$$b = 1{,}25m \qquad t = p/2$$

Escriba un script para obtener todos los valores geométricos del diente a partir del módulo ($m = 5\ mm$) y el número de dientes ($N = 20$).

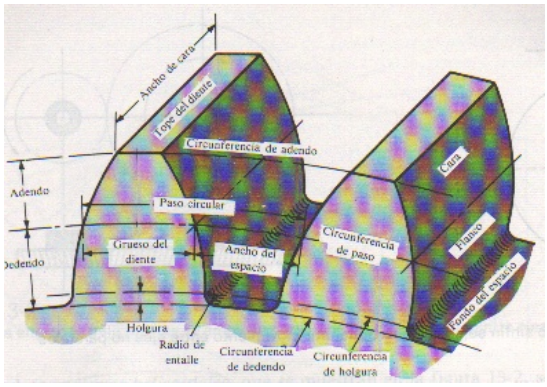


Figura 1.1: Geometría del diente de un engrane recto

4. El flujo de caudal Q en m^3/s en un canal abierto de sección transversal circular como se muestra en la figura 1.2 es dada por

$$Q = \frac{2^{3/2}\ D_c^{5/2}\sqrt{g}\ (\theta - 0{,}5\sin(2\theta))^{3/2}}{8\ \sqrt{\sin\theta}\ (1-\cos\theta)^{5/2}}$$

donde $g = 9{,}81\ m/s^2$ es la constante gravitacional y D_c esta dado por

$$D_c = \frac{d}{2}(1-\cos\theta)$$

Si se asume que $d = 2\ m$ y $\theta = 60°$. Determine el valor del caudal Q.

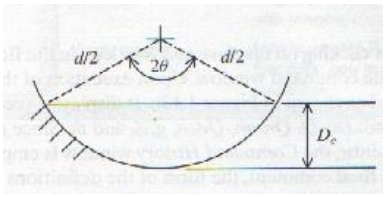


Figura 1.2: Canal de sección circular

5. El momento de Inercia de una sección semi–circular es

$$I_{y'} = \left(\frac{\pi}{8} - \frac{8}{9\pi}\right) R^4$$

donde R es el radio del circulo. Determine $I_{y'}$ cuando $R = 2{,}5\ cm$.

6. El momento de inercia en x de una sección sector circular es

$$I_x = \frac{1}{4}\left(\alpha - \frac{1}{2}\sin(2\alpha)\right) R^4$$

donde R es el radio del circulo y 2α es el ángulo interno del sector circular. Determine I_x cuando $R = 3\ cm$ y $\alpha = 10°$.

7. El momento de inercia en y de una sección sector circular es

$$I_y = \frac{1}{4}\left(\alpha + \frac{1}{2}\sin(2\alpha)\right)R^4$$

donde R es el radio del circulo y 2α es el ángulo interno del sector circular. Determine I_y cuando $R = 3\ cm$ y $\alpha = 10°$.

8. La corrección para la curvatura de un resorte helicoidal en comprensión es

$$K = \frac{4c-1}{4c-4} + \frac{0{,}615}{c}$$

donde $c = D/d$, D es el diámetro de la bobina del resorte, y d es el diámetro del alambre que forma la bobina. Determine el valor de K cuando $c = 5$.

9. El factor de forma para la deflección de un resorte plano es

$$K = \frac{3}{(1-B)^3}(0{,}5 - 2B + B(1{,}5 - \ln B))$$

donde $B < 1$ es la razón del final del trapezoide. Determine K cuando $B = 0{,}6$

10. La longitud L de una correa que transfiere potencia entre dos poleas, una de radio R y otra de radio r, cuyos centros se encuentran a una distancia S, esta dada por

$$L = 2S\cos\theta + \pi(R+r) + 2\theta(R-r)$$

donde

$$\theta = \sin^{-1}\left(\frac{R-r}{S}\right)$$

Determine L cuando $R = 30$ cm, $r = 12$ cm, y $S = 50$ cm.

11. El torque T en un freno de bloqueo esta dado por

$$T = \frac{4fF_n r\sin(\theta/2)}{\theta + \sin(\theta)}$$

donde θ es el ángulo de contacto en radianes, f es el coeficiente de fricción, r es el radio del tambor, y F_n es la fuerza normal que actúa sobre tambor. Determine T cuando $F_n = 250$ N, $f = 0{,}35$, $r = 0{,}4$ m, y $\theta = 60°$.

12. El flujo de aire en un ducto rectangular con lados de longitud A y B tiene la resistencia de flujo equivalente a un ducto circular de diámetro D, es dado por la ecuación

$$D = 1{,}265\left(\frac{(AB)^3}{A+B}\right)^{1/5}$$

Determine D cuando $A = 1{,}7$ m, y $B = 1{,}2$ m.

13. La aceleración angular máxima de la rueda de Ginebra que contiene n ranuras es

$$a_G = \omega^2\frac{M(1-M^2)\sin\alpha}{(1+M^2-2M\cos\alpha)^2}$$

donde

$$\cos\alpha = \sqrt{\left(\frac{1+M^2}{4M}\right)^2 + 2} - \left(\frac{1+M^2}{4M}\right)$$

$$M = \frac{1}{\sin(\pi/n)}$$

Determine a_G/ω^2 cuando $n = 6$.

14. La caída de presión de aire a condiciones estándar de flujo a través de un tubo de acero es

$$\Delta p = \frac{0{,}03L}{d^{1{,}24}}\left(\frac{V}{1000}\right)^{1{,}84}$$

donde L es la longitud del tubo en metros, V es la velocidad del aire en metros por minuto, y d es el diámetro del tubo en milímetros. Determine Δp cuando $L = 3000$ m, $d = 45$ mm, y $V = 1600$ m/min.

15. La siguiente expresión describe el esfuerzo principal de contacto la dirección $-x$, $-y$, y $-z$, respectivamente, cuando dos esferas son presionadas juntas con una fuerza F:

$$\sigma_x = \sigma_y = -p_{max}\left(\left(1 - \frac{z}{a}\tan^{-1}\left(\frac{a}{z}\right)\right)(1-\nu_1) - 0{,}5\left(1+\frac{z^2}{a^2}\right)^{-1}\right)$$

$$\sigma_z = \frac{-p_{max}}{1+z^2/a^2}$$

donde

$$a = \sqrt[3]{\frac{3F}{8}\frac{(1-\nu_1^2)/E_1 + (1-\nu_2^2)/E_2}{1/d_1 + 1/d_2}}$$

$$p_{max} = \frac{3F}{2\pi a^2}$$

y ν_j, E_j y d_j (donde $j = 1{,}2$) son la relación de Poisson, el módulo de Young, y el diámetro, respectivamente de cada una de las esferas.

Determine el esfuerzo principal cuando $\nu_1 = \nu_2 = 0{,}3$. $E_1 = E_2 = 3\text{x}10^7$ psi, $d_1 = 1{,}5$ in; $d_2 = 2{,}75$ in; $F = 100$ lb; y $z = 0{,}01$ in.

16. La siguiente expresión describe el esfuerzo principal de contacto las direcciones $-x$, $-y$, y $-z$ respectivamente, cuando dos cilindros , cuyos ejes son paralelos, son presionados juntos con una fuerza F:

$$\sigma_x = -2\nu_2 p_{max}\left(\sqrt{1+\frac{z^2}{b^2}} - \frac{z}{b}\right)$$

$$\sigma_y = -p_{max}\left(\left(2 - \left(1+\frac{z^2}{b^2}\right)^{-1}\right)\sqrt{1+\frac{z^2}{b^2}} - 2\frac{z}{b}\right)$$

$$\sigma_z = \frac{-p_{max}}{\sqrt{1+z^2/b^2}}$$

donde

$$p_{max} = \frac{2F}{\pi b L}$$

$$b = \sqrt{\frac{2F}{\pi L}\frac{(1-\nu_1^2)/E_1 + (1-\nu_2^2)/E_2}{1/d_1 + 1/d_2}}$$

y ν_j, E_j y d_j (donde $j = 1{,}2$) son la relación de Poisson, el módulo de Young, y el diámetro, respectivamente de cada uno de los cilindros.

Determine los esfuerzos principales cuando $\nu_1 = \nu_2 = 0{,}3$. $E_1 = E_2 = 3\text{x}10^7$ psi, $d_1 = 1{,}5$ in; $d_2 = 2{,}75$ in; $F = 100$ lb; $L = 2$ in; y $z = 0{,}001$ in.

17. El modulo de carga de un rodamiento hidrodinámico esta dado por

$$N_L = \frac{\pi\varepsilon\sqrt{\pi^2(1-\varepsilon^2)+16\varepsilon^2}}{(1-\varepsilon^2)^2}$$

donde ε es la relación de excentricidad. Determine el valor de N_L cuando $\varepsilon = 0{,}8$

18. Considere la rosca de un tornillo de altura h y cuyo material posee un modulo de Young E. La rigidez k del tornillo cuando esta en un agujero de diámetro d_0 puede ser estimada a partir de:

$$k = \frac{\pi E d_0 \tan 30^\circ}{\ln\left(\frac{(d_2-d_0)(d_1+d_0)}{(d_2+d_0)(d_1-d_0)}\right)}$$

donde d_1 es el diámetro de la arandela bajo el tornillo y

$$d_2 = d_1 + h\tan 30^\circ$$

Determine el valor de k cuando $h = 1{,}25$ in; $d_0 = 0{,}25$ in; $d_1 = 0{,}625$ in; y $E = 3\text{x}10^7$ psi. Recuerde que los argumentos en las funciones trigonométricas deben estar en radianes.

19. El esfuerzo radial y tangencial a lo largo de un tubo a temperatura T_a en su superficie interna de radio a, y temperatura T_b en la superficie externa de radio b, es respectivamente:

$$\sigma_r = \frac{\alpha E(T_a - T_b)}{2(1-\nu)\ln(b/a)}\left(\frac{a^2}{b^2-a^2}\left(\frac{b^2}{r^2}-1\right)\ln\left(\frac{b}{a}\right) - \ln\left(\frac{b}{r}\right)\right)$$

$$\sigma_t = \frac{\alpha E(T_a - T_b)}{2(1-\nu)\ln(b/a)}\left(1 - \frac{a^2}{b^2-a^2}\left(\frac{b^2}{r^2}+1\right)\ln\left(\frac{b}{a}\right) - \ln\left(\frac{b}{r}\right)\right)$$

donde r es la coordenada radial sobre el tubo, E es el modulo de Young del material del tubo, y α es el coeficiente de condución térmica. La distribución temperatura sobre la pared del tubo en la dirección radial es

$$T = T_b + \frac{(T_a - T_b)\ln(b/r)}{\ln(b/a)}$$

Determine los esfuerzos y la temperatura T cuando $\alpha = 1{,}2\text{x}10^{-5}$ in/in/°F; $E = 3\text{x}10^7$ psi; $\nu = 0{,}3$; $T_a = 500$°F; $T_b = 300$°F; $a = 0{,}25$ in; $b = 0{,}5$ in; $r = 0{,}375$ in.

20. El flujo másico de gas que escapa de un tanque de presión p_0 y bajo condiciones adiabáticas irreversibles es proporcional a

$$\psi = \sqrt{\frac{k}{k-1}}\sqrt{\left(\frac{p_e}{p_0}\right)^{2/k} - \left(\frac{p_e}{p_0}\right)^{(k+1)/k}}$$

donde p_e es la presión externa al tanque, y k es la constante adiabática irreversible del gas.
Determine ψ cuando $k = 1{,}4$ y $p_e/p_0 = 0{,}3$.

21. El factor de descarga de flujo a través de un canal abierto de sección parabólica es

$$K = \frac{1{,}2}{x}\left(\sqrt{16x^2+1} + \frac{1}{4x}\ln(\sqrt{16x^2+1}+4x)\right)^{-2/3}$$

donde x es la relación entre la máxima profundidad de agua y el ancho del canal en la parte superior del canal.
Determine K cuando $x = 0{,}45$

22. El momento de torsión de inercia ejercido por las partes móviles sobre el cigüeñal (teniendo en cuenta α) de un motor de combustión interna de 18 cilindros y 5,2 $MWatts$ es

$$T_{21}'' = -m_B \cdot r^2 \cdot \alpha \sin(\omega_t) \left(\sin(\omega_t) + \frac{r}{L}\sin(2\omega_t)\right) - \frac{m_B \cdot r^2 \cdot \omega^2}{2}\left(\frac{r}{2L}\sin(\omega_t) + \sin(2\omega_t) + \frac{3r}{2L}\sin(3\omega_t)\right)$$

donde m_B es la masa del piston, r es la distancia entre el eje del cigüeñal y el eje del muñón de la biela, L es la longitud de la biela, α y ω son la aceleración y velocidad angular respectivamente, y ω_t es el ángulo de la manivela. Determine el valor del momento de torsión T_{21}'' en $N \cdot m$ en un motor que posee $m_B = 45\ kg$, $L = 700\ mm$, $r = 240\ mm$, $\alpha = 12{,}5\ rpm/s$, $\omega = 50\ rpm$ y $\omega_t = 45°$

23. La fuerza de inercia total producida por las partes móviles de un motor de combustión interna de 18 cilindros y 5,2 $MWatts$ esta dado por

$$F^x = (m_A + m_B)(r \cdot \alpha \sin(\omega_t) + r \cdot \omega^2 \cos(\omega_t)) + \frac{m_B \cdot r^2}{L}\left(\frac{\alpha}{2}\sin(2\omega_t) + \omega^2 \cos(2\omega_t)\right)$$

$$F^y = -m_A \cdot r \cdot \alpha \cos(\omega_t) + m_A \cdot r \cdot \omega^2 \sin(\omega_t)$$

donde m_A y m_B es la masa de la biela y del piston respectivamente, r es la distancia entre el eje del cigüeñal y el eje del muñón de la biela, L es la longitud de la biela, α y ω son la aceleración y velocidad angular respectivamente, y ω_t es el ángulo de la manivela. Determine el valor de las fuerzas de inercia F^x y F^y en $Newton$ en un motor que posee $m_A = 245\ kg$ $m_B = 77\ kg$, $L = 700\ mm$, $r = 240\ mm$, $\alpha = 12{,}5\ rpm/s$, $\omega = 50\ rpm$ y $\omega_t = 45°$

24. Dados P, Q y R como datos booleanos, realizar las siguientes operaciones

    a) $\sim (\sim P \vee \sim Q)$
    b) $\sim (\sim P \wedge \sim Q)$
    c) $P \wedge (P \vee Q)$
    d) $P \wedge (P \wedge Q)$
    e) $(P \ \wedge Q) \vee (\sim P \ \wedge Q) \vee (P \wedge \sim Q) \vee (\sim P \wedge \sim Q)$
    f) $(\sim P \wedge (\sim Q \wedge R)) \vee (Q \wedge R) \vee (P \wedge R)$
    g) $(\sim P \vee (\sim Q \wedge R)) \wedge (Q \wedge R) \wedge (P \vee R)$

    cuando

    - P es verdadero *si* $4 = 3$
    - Q es falso *si* $8 - 2 > 10$
    - R es verdadero *si* $5 * 3 < 25$

1.12. Solución a problemas propuestos

1. El momento de torsión de inercia ejercido por las partes móviles sobre el cigüeñal (teniendo en cuenta α) de un motor de combustión interna de 18 cilindros y 5,2 $MWatts$ es

$$T_{21}'' = -m_B \cdot r^2 \cdot \alpha \sin(\omega_t) \left(\sin(\omega_t) + \frac{r}{L}\sin(2\omega_t)\right) - \frac{m_B \cdot r^2 \cdot \omega^2}{2}\left(\frac{r}{2L}\sin(\omega_t) + \sin(2\omega_t) + \frac{3r}{2L}\sin(3\omega_t)\right)$$

donde m_B es la masa del piston, r es la distancia entre el eje del cigüeñal y el eje del muñón de la biela, L es la longitud de la biela, α y ω son la aceleración y velocidad angular respectivamente, y ω_t es el ángulo de la manivela. Determine el valor del momento de torsión T_{21}'' en $N \cdot m$ en un motor que posee $m_B = 45\ kg$, $L = 700\ mm$, $r = 240\ mm$, $\alpha = 12{,}5\ rpm/s$, $\omega = 50\ rpm$ y $\omega_t = 45°$

a) Realice las operaciones paso por paso, teniendo en cuenta la jerarquía de los operadores.

b) Escriba el código en MATLAB para solucionar el problema, incluyendo los respectivos comentarios (documentación).

Solución

Primero es necesario realizar las conversiones a unidades consistentes:

Variable	m_B	L	r	α	ω
Unid. Base	45 Kg	700 mm	240 mm	12.5 rpm/s	50 rpm
Unid. Consistente	45 Kg	0.7 m	0.24 m	1.30 rad/s^2	5.236 rad/s

$$T''_{21} = -m_B \cdot r^2 \cdot \alpha \sin(\omega_t)\left(\sin(\omega_t) + \frac{r}{L}\sin(2\omega_t)\right) - \frac{m_B \cdot r^2 \cdot \omega^2}{2}\left(\frac{r}{2L}\sin(\omega_t) + \sin(2\omega_t) + \frac{3r}{2L}\sin(3\omega_t)\right)$$

$$T''_{21} = -2{,}39 * 1{,}05 - 35{,}53 * 1{,}4849$$

$$T''_{21} = -2{,}51 - 52{,}75$$

$$T''_{21} = -55{,}27$$

El código para desarrollar, puede ser de la siguiente forma:

```
% Solucion problema motor combustion interna
% Examen de Lenguaje de Programacion

mb=45;  % Masa del piston [Kg]
r=0.240;  % Radio ciguenal y eje biela [m]
L=0.700;  % Longitud Biela [m]
alpha=12.5*2*pi/60;  % Aceleracion angular [rad/s^2]
w=50*2*pi/60;        % Velocidad angular [rad/s]
Wt = 45*pi/180; % Angulo manivela [rad]

T21 = -mb*r^2*alpha*sin(Wt)*(sin(Wt)+r/L*sin(2*Wt))...
     -mb*r^2*w^2/2*(r/(2*L)*sin(Wt)+sin(2*Wt)+(3*r)/(2*L)*sin(3*Wt));
```

2. Llene los espacios con la función que considere correcta:

a) Si $x = 2{,}7$, que función de las funciones de conversión decimal-entero se debe usar si se desea obtener:

y = _____(x) = 2 y = _____(x) = 3

b) Dado el número complejo $z = 3 + 4i$ qué función se debe utilizar para obtener

z = _____(3,4) = 3+4i

y = _____(z) = 4 y = _____(z) = 3

y = _____(z) = 5 y = _____(z) = 3-4i

Solución

Utilizando la tabla de funciones:

a) Si $x = 2{,}7$, que función de las funciones de conversión decimal-entero se debe usar si se desea obtener:

y = fix floor(x) = 2 y = round ceil(x) = 3

b) Dado el número complejo $z = 3 + 4i$ qué función se debe utilizar para obtener

z = complex(3,4) = 3+4i
y = imag (z) = 4 y = real(z) = 3
y = abs (z) = 5 y = conj(z) = 3-4i

3. Explique la causa del siguiente mensaje de error, y la posible solución:

```
??? Undefined function or variable 'a'.
```

Solución

La variable no ha sido definida y se corrige dándole valor a la variable `a`.

4. Se presenta la definición de algunas variables en MATLAB. Corrija el error.

a) `>> Nombre = Carlos A Vargas`

b) `>> Fecha de Nacimiento = 22031998`

c) `>> 2_Nota_Programacion = 2.5`

Solución

Cada uno de los errores se puede corregir de la siguiente forma:

a) `>> Nombre = 'Carlos A Vargas' % String o Texto`

b) `>> Fecha_de_Nacimiento = 22031998 % El nombre de la vble no debe llevar espacios`

c) `>> Nota_Programacion_2 = 2.5 % El nombre de la vble no puede empezar en numero`

Capítulo 2

Arrays, Vectores y Matrices

El corazón y alma de MATLAB es el algebra lineal. En efecto, MATLAB es originalmente una contracción de "*Mat*rix *Lab*oratory". Mas que otros lenguajes, MATLAB se especializa en el uso y las operaciones entre arrays, vectores y matrices. En este sentido MATLAB sería como una potente calculadora matricial, aunque en realidad es esto, y mucho mas.

2.1. Definiciones

Array: Un *array* es una colección de números, caracteres, ecuaciones, etc. llamados *elementos* o *entradas*, *referenciadas* por uno o más índices que se mueven dentro de un set de índices. En MATLAB, el set de índices son siempre números enteros secuenciales que empiezan con 1. La *dimensión* de un array es el número de índices necesarios para especificar un elemento. El *tamaño* de un array es el tamaño del set de indices necesarios para definir todos los elementos.

De esta forma, un *array bidimensional* es una tabla de $m \times n$ elementos dispuestos en m filas y n columnas. Se suelen representar por letras mayúsculas $\mathbf{A}$, $\mathbf{B}$, ... etc. y a sus *elementos* de la forma a_{ij} donde el primer subíndice indica la fila, y el segundo, la columna en la que se encuentra dicho elemento.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Matriz: Una *matiz* es un array con reglas especiales para la adición, multiplicación y otras operaciones, descritas por el Algebra Lineal.

Vector: Un *vector* es una matriz en cual uno de los indices de dimensión es solamente 1. Se denomina *vector fila* a aquella matriz que consta de una única fila.

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}$$

y *vector columna* a aquella matriz que consta de una única columna.

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$$

Para la asignación de variables, se sugiere que se utilicen letras minúsculas para vectores y escalares (MATLAB no exige esto, pero puede ser útil).

Aunque un array es mucho mas general y menos matemático que una matriz, los términos se pueden intercambiar. Esto permite que una variable se pueda comportar como un array, una matriz, un vector o un escalar, dependiendo las operaciones solicitadas.

2.2. Construyendo arrays

Definamos las matrices que se utilizan como ejemplo:

$$\mathbf{A} = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 0{,}5 & 0{,}6 & 0{,}7 & 0{,}8 \\ 3 & 2{,}75 & 2{,}5 & 2{,}25 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 21 \\ 17 \\ 13 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 0{,}5 & 0{,}6 & 0{,}7 & 0{,}8 \\ 3 & 2{,}75 & 2{,}5 & 2{,}25 \\ 8 & 7 & 9 & 6 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 8 & 7 & 9 & 6 \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 21 & 11 & 12 & 13 & 14 \\ 17 & 0{,}5 & 0{,}6 & 0{,}7 & 0{,}8 \\ 13 & 3 & 2{,}75 & 2{,}5 & 2{,}25 \end{bmatrix}$$

La manera más simple de construir un array o matriz es encerrando los elementos en corchetes cuadrados.

```
>> A = [11 12 13 14 ; 0.5 0.6 0.7 0.8 ; 3 2.75 2.5 2.25]
 A =
   11   12    13   14
   0.5  0.6   0.7  0.8
   3    2.75  2.5  2.25

>> b = [8  7  9  6]
b =
   8  7  9  6

>> c = [21 ; 17 ; 13]
c =
    21
    17
    13
```

Las **columnas** se separan por *espacios* o *comas*, y las **filas** por *punto y coma* o *una nueva linea*. La información sobre el tamaño y la dimensión es almacenada con el array. Como se puede apreciar, no es necesario definir de antemano las variables o el tamaño de la misma, de hecho, puede ser modificado posteriormente.

A partir de este momento la matriz **A** y el vector **b** y **c**, están disponibles para realizar cualquier tipo de operación, o aplicar una función.

Los arrays pueden ser construidos a partir de otros arrays, siempre que las dimensiones sean compatibles.

```
>> B = [A ; b]
B =
   11   12    13   14
   0.5  0.60   0.7  0.8
   3.0  2.75  2.5  2.25
   8.0    7     9    6

>> D = [c A]
D =
    21  11   12    13   14
    17  0.5  0.6   0.7  0.8
    13  3    2.75  2.5  2.25
```

```
>> [A b]
??? Error using ==> vertcat
All rows in the bracketed expression
must have the same number of columns.
```

La construcción de arrays con corchetes cuadrados es útil para construir matrices pequeñas. Para las más grandes, hay muchas funciones, algunas son mostradas en la tabla 2.1.

Tabla 2.1: Funciones Matriciales

Función	Descripción
eye	genera matriz identidad
zeros	matriz de elementos ceros
ones	matriz de elementos unos
diag	extrae la diagonal
toeplitz	constantes en cada diagonal
triu	triangular superior
tril	triangular inferior
rand, randn	matriz de valores aleatorios
ndims	determina la dimensión
size	número de filas y columnas
length	número de elementos de un vector
'	transpuesta conjugada de la matriz
.'	transpuesta NO conjugada de la matriz
:	(a) Separador en la creación de vectores de la forma `x = a:b:c` (b) Para la matriz `z`, indica "todas las filas" cuando `z(:,k)` o "todas las columnas" cuando `z(k,:)`.

La *matriz identidad* es aquella donde los elementos de sus diagonal principal son uno.

$$\mathbf{I1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{I2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

las cuales se pueden definir en MATLAB de la forma

```
>> I1 = eye(4)   % Matriz identidad cuadrada de 4 x 4
>> I2 = eye(3,4) % Matriz identidad de 3 filas - 4 columnas
```

También se pueden definir matrices llenas de unos o de una valor diferente, como

$$\mathbf{U1} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad \mathbf{U2} = \begin{bmatrix} 7 & 7 \\ 7 & 7 \\ 7 & 7 \\ 7 & 7 \end{bmatrix}$$

que se definen por medio de los comandos

```
>> U1 = ones(5)    % Matriz cuadrada de unos de 5 x 5
>> U2 = ones(4,2)*7  % Matriz de sietes de 4 filas - 2 columnas
```

La matriz *diagonal principal* esta constituida por los elementos de la diagonal principal de la matriz especificada.

```
>> Bd = diag(B)  % Elementos de la diagonal principal
Bd =
  11.00
```

```
   0.60
   2.50
   6.00
```

Otras funciones, permiten obtener información sobre las matrices que están definidas

```
>> size(A) % Determina el numero de filas y columnas de la matriz
ans =
     3   4

>> size(c) % Determina el numero de filas y columnas del vector
ans =
     3   1

 >> length(B)  % Numero de elementos de la matriz B
ans =
      16

>> length(b)
ans =
     4

>> size(D) % Determina el numero de filas y columnas de la matriz
ans =
     3   5
```

Operador Dos Puntos (:)

Una forma especial de construcción es el operador *dos puntos*. Este operador permite crear vectores a partir de una regla definida de construcción, es decir, si se conoce el *primer* elemento, el *último* y el intervalo o *pasos* entre ellos, se crea el vector a partir del formato `primero:paso:ultimo`. Donde el resultado es siempre un vector fila.

```
>> 1:8
ans =
     1   2   3   4   5   6   7   8

>> 0:2:10
ans =
     0   2   4   6   8   10

>> 1:-0.5:-1
ans =
    1.0000  0.5000  0   -0.5000  -10000

>> c = [21 : -4 : 13]';
>> A = [11:14 ; 0.5:0.1:0.8 ; 3:-0.25:2.25];
```

2.3. Referenciando elementos

Es frecuentemente necesario acceder a uno o mas elementos de una matriz. El resultado es un *bloque* extraído de la matriz. Dependiendo de la dimensión del array, se debe utilizar el mismo número de índices. Por ejemplo, un vector es un array de una dimensión, y por lo tanto solo es necesario un índice para especificar el valor a extraer

```
>> b(2)     % b es un vector
ans =
      7

>> c(3)     % c es un vector
ans =
      13
```

En el caso de una matriz bidimensional (como **A**), se deben utilizar dos indices para especificar el valor a obtener. Cada uno de los índices se separan por **coma** (,) donde el primero indica la *fila* y el segundo indica la *columna*.

```
>> A(2,3)          >> A(1,2)          >> A(2,1)
ans =              ans =              ans =
      0.7                  12                0.5
```

Cuando es necesario referenciar multiples elementos de la matriz o vector, se utiliza un *vector de índices*

```
>> b([1 3]) % multiples elementos
ans =
      8  9

>> A(1:2,2:3)   % a una submatriz
ans =
    12    13
    0.6   0.7

>> B(1,2:end)   % comando especial -- end : ultimo
ans =
    12   13   14

>> B(:,3)   % : -- comando de incluya todo
ans =
    13
    0.7
    2.5
    9.0

>> A(2 , :)      % Los elementos de la fila '2' y 'todas' las columnas
ans =
   0.5   0.6    0.7    0.8

>> A(2 , 2:3)  % los elementos de la fila '2' y las columnas 'de la 2 a la 3'
ans =
    0.6   0.7
```

Los dos puntos (:) es una forma sencilla de de construir los índices, y en este caso, se interpretan como *hasta*, o como la forma corta de decir `1:end`. Otra sintaxis especial, el comando `end` significa el índice del último elemento, todo en esa dimensión.

Los vectores pueden ser especificados por un solo subíndice. En efecto, un array puede ser especificado también por un solo subíndice. Los arrays multidimensionales son almacenados en memoria linealmente, cambiando de la primera dimensión a la segunda y así sucesivamente. Es como si pusieran las columnas de un vector una sobre la otra. En este sentido, el array es equivalente a un vector, y solo subíndice puede ser usado en este contexto.

```
>> A       % Recordar los valores de la matriz A
A =
  11.0000   12.0000   13.0000   14.0000
   0.5000    0.6000    0.7000    0.8000
   3.0000    2.7500    2.5000    2.2500

>> A(2)
ans =
       0.5

>> A([1 2 3 4])
ans =
     11    0.5    3   12

>> A(1:4)      % O de la misma forma...
ans =
     11    0.5    3   12

>> A(1:2:end)   % Los elementos del '1' hasta el 'final' de '2 en 2'
ans =
      11    3    0.6   13    2.5    0.8

>> A([1;2;3;4])  % Los mismos elementos pero en vector columna
ans =
   11.00
   0.500
   3.000
   12.00
```

El potencialmente ambiguo `A(:)` es siempre un vector columna con todos elementos del array.

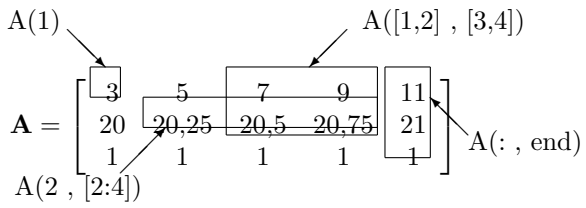
El subíndice puede ser usado en cualquiera de los dos lados de la asignación:

```
>> A(1,3) = 9;   % Guardar en la posicion (1,3) de 'A' el valor 9

>> A(2,[2 3]) = [-0.3 -0.5];   % Guardar en la fila '2', columna '2' y '3' de 'A'
>>    % los valores -0.3 y -0.5 respectivamente

>> A(end,3) = B(3,end);  % Guardar en la 'ultima' fila, columna '3'de 'A'
>>    %  el valor de la fila '3', 'ultima' columna de 'B'

>> % Guardar en la fila '1' y 'todas' las columnas de la matriz 'A'
>> % los elementos de la 'ultima' fila y 'todas' las columnas de la matriz 'B'
>> A(1,:) = B(end,:)
A  =
     8.00    7.00    9.00    6.00
     0.50   -0.30   -0.50    0.80
     3.00    2.75    2.25    2.25

>> F = rand(2,5)  % Vector de elementos aleatorios de 2 filas, 5 columnas
F =
   0.8125  0.4054  0.4909  0.5943
   0.2176  0.5699  0.1294  0.3020
```

```
>> F(:,4) = [ ]  % Borra los elementos de 'todas' las filas, columna '4'
F =
   0.8125  0.4054  0.4909
   0.2176  0.5699  0.1294

>> % Colocar el escalar '0' dentro de la matriz F en las posiciones
>> %   fila '2', 'todas' las columnas
>> F(2,:) = 0
F =
   0.8125  0.4054  0.4909  0.5943
        0       0       0       0

>> F(3,1) = 3   % Crea una nueva fila para crear el espacio necesario
F =
   0.8125  0.4054  0.4909  0.5943
        0       0       0       0
   3.0000       0       0       0
```

Un array es redimensionado automáticamente si se borran elementos o se hace una asignación por fuera de tamaño actual, con lo cual los elementos no especificados se llenan de ceros. Esto puede ser claramente conveniente, pero puede hacerlo mas difícil de encontrar errores.

Hagamos un repaso, dada la matiz **A** se escribe junto a la flecha indicadora, la línea de código necesaria para obtener los valores señalados



2.4. Operaciones con Matrices

MATLAB puede operar con matrices por medio de *operadores aritméticos* (Cap 1.4.1) y por medio de *funciones* (Cap 2.4.4).

2.4.1. Suma y Resta de Matrices

Para sumar matrices se utiliza el operador aritmético +, pero es necesario tener presente que solo es posible sumar matrices del mismo tamaño

$$\mathbf{A}+\mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1m}+b_{1m} \\ a_{21}+b_{21} & a_{22}+b_{12} & \dots & a_{2m}+b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nm}+b_{nm} \end{bmatrix}$$

de igual forma, se puede utilizar el operador −, para obtener la diferencia entre dos matrices. Con el uso del operador aritmético, MATLAB calcula el resultado del calculo:

```
>> A + B
>> A - B
```

2.4.2. Multiplicación con un escalar

El producto de una matriz por un escalar, es otra matriz en la cual cada elemento se multiplica por el escalar

$$k\mathbf{A} = \begin{bmatrix} k\ a_{11} & k\ a_{12} & \dots & k\ a_{1m} \\ k\ a_{21} & k\ a_{22} & \dots & k\ a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ k\ a_{n1} & k\ a_{n2} & \dots & k\ a_{nm} \end{bmatrix}$$

Para realizarlo en MATLAB de la forma

```
>> k * A
>> A * k   % por que la operacion es conmutativa
```

2.4.3. Multiplicación de Matrices

Para realizar la multiplicación entre dos matrices es muy importante tener en cuenta: *el número de columnas de la primera matriz,* **debe** *ser igual, al número de la filas de la segunda*

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1r} \\ b_{21} & b_{22} & \dots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mr} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1r} \\ c_{21} & c_{22} & \dots & c_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nr} \end{bmatrix}$$

donde

$$c_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk}$$

De ahí que, la multiplicación de una matriz $(n \times m)$ por una matriz $(m \times r)$, genera como resultado una matriz $(n \times r)$.

```
>> A * B
```

En general la multiplicación de matrices no es conmutativa

$$\mathbf{A} * \mathbf{B} \neq \mathbf{B} * \mathbf{A}$$

Potencia de matrices

La potencia de matrices, se efectúa realizando las multiplicaciones necesarias para obtener el resultado

$$\mathbf{A}^2 = \mathbf{A} * \mathbf{A}$$

por lo tanto, la matriz *debe* ser una matriz cuadrada. Y se utiliza, de la misma forma, el operador para realizar el calculo:

```
>> A^2
```

2.4.4. Funciones Matriciales

Además de las operaciones comunes con matrices, MATLAB posee funciones matriciales especiales.

Conjugada de una Matriz

La conjugada de una matriz **A** es una matriz en la cual cada elemento es el *complejo conjugado* del correspondiente elemento de **A**. La sintaxis utilizada es:

```
>> conj(A)
```

Transpuesta

Si las filas y columnas de una matriz **A** de $n \times m$ se intercambian, entonces la matriz resultante $m \times n$ se denomina *la transpuesta de* **A**. La transpuesta de **A** se denota por $\mathbf{A}^t$.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \Longrightarrow \quad \mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \dots & a_{nm} \end{bmatrix}$$

Para obtener la transpuesta de un vector o matriz, se utiliza el operador *comilla sencilla* '.

```
>> A'
ans =
      8.00   0.50   3.00
      7.00  -0.30   2.75
      9.00  -0.50   2.25
      6.00   0.80   2.25

>> b'
ans =
      8
      7
      9
      6
```

Determinante de una matriz

Los determinantes proporcionan un método para el calculo de la matriz inversa de una dada (en caso de existir) y un criterio para estudiar si una matriz es o no invertible. Sus aplicaciones son múltiples en todas las ramas de las ciencias que tratan problemas lineales en los que necesariamente aparecen matrices y por tanto, determinantes.

Para una matriz cuadrada **A** existe un número que se denomina como *determinante* de **A** y se presenta por $det(A)$ o $|A|$. El determinante de una matriz cuadrada de $n \times n$ es un polinomio cuyos sumandos son todos los posibles productos de n factores (factores que son elementos de **A**), de tal forma que en todos los productos exista uno y solamente un factor de cada fila de **A** y uno y solamente un factor de cada columna.

Para obtener el determinante de una matriz cuadrada, en MATLAB se puede utilizar la expresión

```
>> det(A)
```

Inversa de una matriz cuadrada

La inversa de una matriz cuadrada **A** es una matriz que cumple

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

siempre que **A** sea *no singular*, es decir, que su determinante sea diferente de cero ($|\mathbf{A}| \neq 0$). La expresión para obtener la inversa de la matriz **A** es

```
>> inv(A)
```

2.5. Operaciones con Arrays

Además del algebra matricial clásica, MATLAB permite realizar operaciones especiales con arrays. La tabla 2.2 resume algunas de las operaciones con más comunes entre escalares y arrays.

Tabla 2.2: Operaciones con Arrays

Datos ilustrativos	
$a = [\ a_1 \quad a_2 \quad \dots \quad a_n\ ]$, $b = [\ b_1 \quad b_2 \quad \dots \quad b_n\ ]$ y c es un escalar	
Operación	Formato y evaluación MATLAB
suma escalar	`a+c` = $[\ a_1 + c \quad a_2 + c \quad \dots \quad a_n + c\ ]$
mult. escalar	`a*c` = $[\ a_1 * c \quad a_2 * c \quad \dots \quad a_n * c\ ]$
div. escalar	`c./a` = $[\ c/a_1 \quad c/a_2 \quad \dots \quad c/a_n\ ]$
mult. array	`a.*b` = $[\ a_1 * b_1 \quad a_2 * b_1 \quad \dots \quad a_n * b_n\ ]$
divi. array	`a./b` = $[\ a_1/b_1 \quad a_2/b_1 \quad \dots \quad a_n/b_n\ ]$
pot. array	`a.^c` = $[\ {a_1}^c \quad {a_2}^c \quad \dots \quad {a_n}^c\ ]$
	`c.^a` = $[\ c^{a_1} \quad c^{a_2} \quad \dots \quad c^{a_n}\ ]$
	`a.^b` = $[\ {a_1}^{b_1} \quad {a_2}^{b_1} \quad \dots \quad {a_n}^{b_n}\ ]$

Se definen nuevas matrices para realizar los ejemplos:

```
>> A = [1:3 ; 4:6 ; 7:9]
>> b = [5 : 3 : 11]
>> c = 4
```

De esta forma entonces se realizan las siguientes operaciones, verifique el resultado de cada una de ellas:

```
 >> A.*A     % Multiplicacion elemento a elemento
ans =
   1   4   9
  16  25  36
  49  64  81

>> A.^2      % Que es equivalente a la anterior

>> (A.*A)-1  % En este caso el escalar (1) es expandido como matriz
ans =
   0   3   8
  15  24  35
  48  63  80

>> 2.^A % 2 elevado a cada uno de los valores de la matriz
ans =
    2     4     8
   16    32    64
  128   256   512

>> A./A  % Division entre cada uno de los elementos de la matiz
ans =
   1   1   1
   1   1   1
   1   1   1
```

```
>> 2./A    % Divide el escalar a cada valor de la matriz
ans =
   2.00    1.00    0.66
   0.50    0.40    0.33
   0.28    0.25    0.22
```

La mayoría de las funciones elementales, como *sin*, *exp*, etc., operan elemento por elemento.

```
>> cos(pi*b)  % Coseno de los valores de 'b' multiplicados por 'pi'
ans =
      -1     1    -1

>> exp(A)      % Cada uno de los valores de 'A' en exponente
ans =
   1.0e+03 *
    0.0027     0.0074      0.0201
    0.0546     0.1484      0.4034
    1.0966     2.9810      8.1031
```

2.6. Operadores Relacionales

En MATLAB los operadores relacionales pueden aplicarse a vectores y matrices, y eso hace que tengan un significado especial. El resultado de aplicar los operadores relacionales, es un *array lógico* cuyos elementos son 0 y 1 con la interpretación de "falso" y "verdadero". Usando un array lógico como indices, este devuelve aquellos valores en los cuales el índice es 1.

```
>> B > 3 % Los valores de 'B' que son mayores a 3
ans =
      1     1     1     1
      0     0     0     0
      0     0     0     0
      1     1     1     1

>> % En 'ans' se guardan las posiciones que cumplen la anterior condicion
>> B(ans)'
ans =
      2.50  7.00  9.00  2.75  14.00  13.00  0.80  0.6
```

La principal diferencia con C está en que cuando los operadores relacionales de MATLAB se aplican a dos matrices o vectores del mismo tamaño, la comparación se realiza elemento a elemento, y el resultado es otra matriz de unos y ceros del mismo tamaño, que recoge el resultado de cada comparación entre elementos.

```
>> D = [1 2;0 3];  E = [4 2;1 5]; % Definicion de las matrices
>> D==E % Posiciones en las cuales 'D' es igual a 'E'
ans =
       0   1
       0   0

>> B~=E % Posiciones en las cuales 'B' es diferente a 'E'
ans =
       1   1
       1   0
```

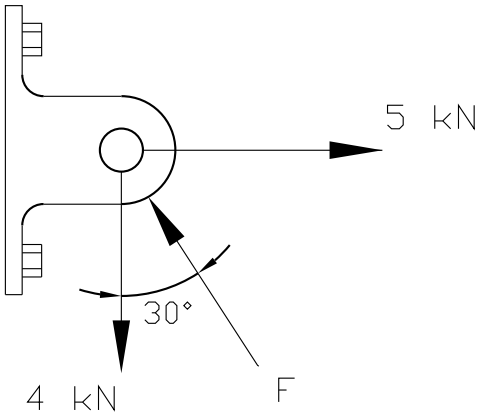
2.7. Aplicaciones en Ingeniería

El rápido desarrollo de la ingeniería en los últimos años, ha sido promovida principalmente por los sistemas programables. Son incontables las aplicaciones en ingeniería de los conceptos de lenguaje de programación. A continuación, se presentan solo algunos de los casos más sencillos, aplicados principalmente para el desarrollo de problemas de física mecánica.

2.7.1. Operaciones de Array para optimización

La forma mas sencilla (pero más larga) de optimización, se realiza, aplicando sobre las ecuaciones del sistemas, *todas* las posibles opciones. De esta forma, todas los soluciones son comparadas buscando los mínimos o máximos resultados (dependiendo la aplicación).

La Figura 2.1 describe el problema, que se puede resolver por medio del método de optimización. A continuación, se muestra una posible solución al problema, y la Figura 6.16, muestra de forma gráfica el punto donde se encuentra el valor mínimo.



Determine la magnitud de la fuerza **F** de tal forma que la magnitud de la resultante F_R de las tres fuerzas tenga un valor tan pequeño como sea posible. ¿Cuál será la magnitud mínima de F_R?

Figura 2.1: Ejemplo de optimización

```
% Solucion Ejemplo de Optimización Estatica
% Ing. Alfonso Cubillos V
% Agosto 2006

% Definicion de constantes
F1 = [5 0];    % Fuerza horizontal como vector cartesiano
F2 = [0 -4];   % Fuerza Vertical como vector cartesiano
tetha = 120*pi/180;    % Angulo fuerza F [rad]

% Definicion de variables
F = (0:0.5:10)';   % Array de valores de F

% Operaciones
F3 = [F*cos(tetha) F*sin(tetha)];  % Fuerza F como vector cartesiano

Fr_x = F1(1) + F2(1) + F3(:,1);    % Sumatorias de fuerzas en x
Fr_y = F1(2) + F2(2) + F3(:,2);    % Sumatorias de fuerzas en y
```

```
[th,Fr] = cart2pol(Fr_x,Fr_y)      % Fuerzas Resultantes polares
[Fr_min,i] = min(Fr)               % Magnitud y posicion del menor valor de Fr
F_min = F(i)                       % Magnitud de F, para el valor minimo

% Solucion  y salida del problema
Fr_min
F_min
```

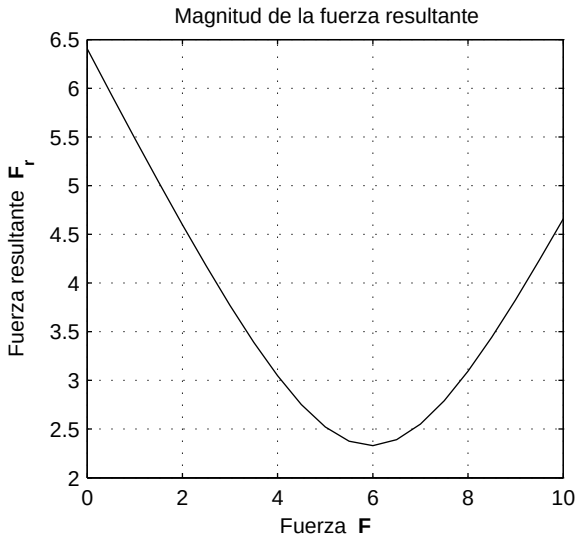


Figura 2.2: Grafica ejemplo

2.7.2. Arrays como vector

El array como vector (fila o columna) posee diferentes ventajas en la solución de problemas físicos, debido a que muchas de las variables físicas se determinan a partir de vectores (fuerza, aceleración, momentos, flujos magnéticos, corriente eléctrica, etc). Los vectores fuerza como

$$T = -2i + 4{,}3j \quad [Lb]$$

$$F = 3i + 2j - 6k \quad [N]$$

o las posiciones de un punto en el espacio

$$s = 9{,}4i + 4{,}23j \quad [ft]$$

$$r = 3{,}4i - 2{,}8j - 3{,}4k \quad [m]$$

se introducen en MATLAB:

```
>> T = [-2  4.3];          % Vector fuerza bidimensional
>> F = [3  2  -6];         % Vector fuerza tridimensional
>> s = [9.4  4.23];        % Vector posicion bidimensional
>> r = [3.4  -2.8  -3.4];  % Vector posicion tridimensional
```

Con estos vectores se pueden utilizar la funciones que muestra la tabla 2.3.

Tabla 2.3: Operaciones Vectoriales

Función	Descripción
cart2pol	transforma coordenadas cartesianas a polares
cart2sph	transforma coordenadas cartesianas a esféricas
pol2cart	transforma coordenadas polares a cartesianas
sph2cart	transforma coordenadas esféricas a cartesianas
cross	producto cruz vectorial
dot	producto punto vectorial
norm	magnitud de un vector

2.7.3. Array como polinomio

Los polinomios también se pueden trabajar como arrays. Tomando los polinomios

$$p_1 = x^2 + 5x - 3$$

$$p_2 = 7{,}6t^5 - 4{,}9t^3 + 5{,}1t^2 - 3{,}1t$$

se introducen en MATLAB de la forma

```
>> p1 = [1  5  -3]; % Primer polinomio
>> p2 = [7.6  0  -4.9  5.1  -3.1  0];   % Segundo polinomio
```

Como polinomios, también se pueden realizar diferentes tipos de operaciones:

Tabla 2.4: Operaciones con Polinomios

Función	Descripción
conv	multiplicación de polinomios
deconv	división de polinomios
roots	raíces de polinomios - método de la matriz compañera
roots1	raíces de polinomios - método de Laguerre
poly	construye un polinomio con unas raíces especificas
polyder	derivada de un polinomio
polyeig	resuelve el problema de los autovalores de un polinomio
polyfit	ajuste de un polinomio a unos datos
polyval	evalúa un polinomio
polyvalm	evalúa un polinomio con una matriz como argumento
residue	desarrollo en fracciones parciales (residuos)

2.7.4. Soluciones de sistemas de ecuaciones Lineales

Uno de los problemas fundamentales del Algebra Lineal es la resolución simultánea de ecuaciones lineales, siendo el caso más simple aquel en el que el número de incógnitas coincide con el número de ecuaciones. Estos sistemas lineales admiten una sencilla representación matricial. Así se puede denotar $\mathbf{Ax} = \mathbf{b}$ siendo

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

donde la matriz **A** se denomina *Matriz de coeficientes del sistema*, el vector **x** como *Variables del sistema* y el vector **b** son los *términos independientes del sistema.*

De esta forma, se puede utilizar la notación matricial para la solución del sistema de ecuaciones:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 336 \\ 2x_1 + 5x_2 + 6x_3 &= 806 \\ 7x_1 + 8x_2 &= 351 \end{aligned}$$

El objetivo es encontrar los valores de x_1, x_2 y x_3 que son las incógnitas del sistema, por medio de la ecuación estándar $Ax = b$, donde x y b son *vectores columna*, y A una matriz cuadrada invertible. La solución de este sistema se puede realizar de la siguiente forma $x = A^{-1} * b$. La solución en MATLAB es:

```
A = [1 2 3 ; 2 5 6 ; 7 8 0];  % Definicion de la matriz cuadrada A
b = [336 ; 806 ; 351];        % Definicion del vector columna b
x = inv(A)*b                  % Solucion del sistema
```

donde el vector columna `x`, es la solución del sistema.

2.8. Ejercicios

1. Por medio de la función de ayuda `help`, describa el objetivo y genere un ejemplo, de las siguientes funciones:
   - `min`
   - `max`
   - `sum`
   - `cumsum`
   - `mean`
2. La posición de un objeto sobre el eje x esta dado por la ecuación
   $$x = 2{,}3t^4 - 30t^2 - 45t + t$$
   a) Obtenga el valor de la posición del cuerpo para cada 0.1 segundos desde 0 hasta 5 segundos.
   b) Utilice los anteriores valores para graficar la posición del cuerpo con respecto al tiempo. ¿El objeto cruza de nuevo la posición $x = 0$? ¿En qué tiempo? ¿Cuál es la posición más negativa a la cual el objeto llega? ¿En qué tiempo?
   c) Obtenga el polinomio que representa la velocidad del cuerpo y grafique para los mismos valores de tiempo. ¿En qué momento la velocidad es cero?
   d) Obtenga un polinomio que describa la aceleración del cuerpo y grafiquela.
3. Una esfera de aluminio sometida a una corriente eléctrica constante, se mide la temperatura de la esfera a intervalos de tiempo definidos, la siguiente tabla muestra los resultados de la prueba

Tiempo	Temperatura	Tiempo	Temperatura	Tiempo	Temperatura
0	23	70	315.51	140	351.33
10	109.39	80	326.09	150	352.63
20	173.39	90	333.93	160	353.59
30	220.81	100	339.73	170	354.30
40	255.93	110	344.03	180	354.82
50	281.95	120	347.22	190	355.21
60	301.23	130	349.58	200	355.50

   Obtenga un polinomio de 4 orden que aproxime el comportamiento de la temperatura con respecto al tiempo.

4. Determine las raíces de los siguientes polinomios:

   a) $x^2 + 2x - 3 = 0$

   b) $2y - 5y^3 = 4y^2 - 3{,}4y^3 + 9y - 8$

   c) $3z^5 + 4z^3 - 9z + 5 = 0$

5. Dadas las raíces, determine los polinomios:

   a) $x_1 = 5$, $x_2 = 3{,}3$, $x_3 = 9{,}5$

   b) $y_1 = 5 + 2i$, $y_2 = 5 - 2i$, $x_3 = 5$, $x_4 = -3{,}2$

6. Un vector fuerza $\mathbf{U} = 20\mathbf{i} + 60\mathbf{j} - 90\mathbf{k}$ (N). Determine la magnitud de la fuerza.

7. Los cables A y B de la figura 2.3 están sometidos las fuerzas $\mathbf{F}_A$ y $\mathbf{F}_B$ sobre el gancho. La magnitud de $\mathbf{F}_A$ es 100lb y $\mathbf{F}_B$ es de 150lb. La longitud de la cuerda A es de 5in y del cable B es de 4in. ¿Cuál es la fuerza resultante sobre el gancho?

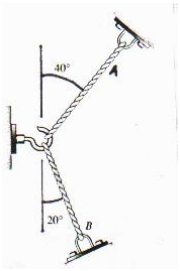


Figura 2.3:

8. Determine el producto punto $\mathbf{U} \cdot \mathbf{V}$ de los vectores $\mathbf{U} = 40\mathbf{i} + 20\mathbf{j} + 60\mathbf{k}$ y $\mathbf{V} = -30\mathbf{i} + 15\mathbf{k}$

9. Cuál es el producto cruz $\mathbf{r} \times \mathbf{F}$ del vector posición $\mathbf{r} = 4\mathbf{i} - 12\mathbf{j} - 3\mathbf{k}$ [ft] y la fuerza $\mathbf{F} = 20\mathbf{i} + 30\mathbf{j} - 10\mathbf{k}$ [lb]?

10. El cilindro de 1000lb de la figura 2.4, esta suspendido de cables atados a los puntos B, C y D. ¿Cuál es la tensión de los cables AB, AC, y AD?

11. *a*) Busque la ayuda de la función `diag` y úsela (tal vez más de una vez) para construir la matriz de 16 x 16

$$D = \begin{bmatrix} -2 & 1 & 0 & 0 & \dots & 0 & 1 \\ 1 & -2 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & \dots & 0 & 1 & -2 \end{bmatrix}$$

   b) Ahora lea acerca de la función `toeplitz` y úsela para construir la matriz D.

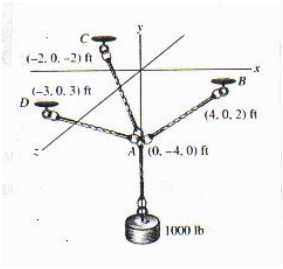


Figura 2.4:

c) Use `toeplitz` y lo que sea necesario para construir

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1 & 1/2 & 1/3 \\ 1/3 & 1/2 & 1 & 1/2 \\ 1/4 & 1/3 & 1/2 & 1 \end{bmatrix} \quad \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 2 & 1 & 2 \\ 2 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

No digite directamente los elementos, la solución debe funcionar para crear matrices de 100 x 100 si es necesario.

12. Dada una matriz de elementos aleatorios de 8 x 8. Encuentre los valores máximos en (a)cada columna, (b) en cada fila, y (c) en toda la matriz. Además (d) encuentre la fila y la columna de los elementos que sean mayores a 0.25, y (e) muéstrelos.

13. Una *matriz mágica* es una matriz de *n x n* en donde cada entero 1, 2, ..., n^2 aparece una vez y para cada columna, fila y diagonal, la suma es idéntica. MATLAB posee el comando `magic` que muestra la matriz mágica. Verifique la salida de la función para diferentes tamaños.

2.9. Solución a problemas propuestos

1. El momento de torsión de inercia ejercido por las partes móviles sobre el cigüeñal (teniendo en cuenta α) de un motor de combustión interna de 18 cilindros y 5,2 $MWatts$ es

$$T''_{21} = -m_B \cdot r^2 \cdot \alpha \sin(\omega_t) \left(\sin(\omega_t) + \frac{r}{L}\sin(2\omega_t)\right) - \frac{m_B \cdot r^2 \cdot \omega^2}{2}\left(\frac{r}{2L}\sin(\omega_t) + \sin(2\omega_t) + \frac{3r}{2L}\sin(3\omega_t)\right)$$

donde m_B es la masa del piston, r es la distancia entre el eje del cigüeñal y el eje del muñón de la biela, L es la longitud de la biela, α y ω son la aceleración y velocidad angular respectivamente, y ω_t es el ángulo de la manivela. Determine el valor del momento de torsión T''_{21} en $N \cdot m$ en un motor que posee $m_B = 45\ kg$, $L = 700\ mm$, $\alpha = 12{,}5\ rpm/s$, $\omega = 50\ rpm$, $\omega_t = 45°$ y $r = [200,\ 210,\ 220,\ \ldots 300]\ mm$

   a) Escriba el código en MATLAB para solucionar el problema, incluyendo los respectivos comentarios.
   b) Coloque los puntos donde sea necesario (no agregue más de los necesarios).

2. Dadas las siguientes matrices:

$$\mathbf{A} = \begin{bmatrix} 10 & 12 & 14 \\ 0{,}5 & 0{,}4 & 0{,}3 \\ 3 & 2{,}75 & 2{,}5 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} -5 \\ 0 \\ 5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 11 & 8 & 0{,}25 & 5 \\ 12 & 6 & 0{,}50 & 5 \\ 13 & 4 & 0{,}75 & 5 \\ 14 & 2 & 1{,}00 & 5 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 8 & 7 & 9 & 6 \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 11 & 8 & 0{,}25 & 5 & 8 \\ 12 & 6 & 0{,}50 & 5 & 7 \\ 13 & 4 & 0{,}75 & 5 & 9 \\ 14 & 2 & 1{,}00 & 5 & 6 \end{bmatrix} \qquad \mathbf{f} = 2$$

a) Cual es el código en MATLAB para generar las matrices. Escriba la forma más corta posible.

b) Marque con una × o con un ✓ si la operación es posible o no con MATLAB

A*B	A*c	b*B	B*b	B*b'	A.*B	B^f	B.^f	f.^A	f*c

c) Cuál es el resultado de las siguientes ordenes

1) `>> A(3,2)`
2) `>> B([3 2],[3 4])`
3) `>> C(:,end)`
4) `>> c([1 3])`
5) `>> b([1 3]')`

3. El extremo O de la barra mostrada en la figura, esta sometido a tres fuerzas coplanares concurrentes. Escriba un código en MATLAB para determinar la magnitud y la orientación de la fuerza resultante. (*As small, as beautiful*)

Solución

1. El código para desarrollar, puede ser de la siguiente forma:

```
% Solucion problema motor combustion interna
% Examen de Lenguaje de Programacion

mb=45;  % Masa del piston
r=[0.2:0.01:0.3];  % Radio ciguenal y eje biela
L=0.700;  % Longitud Biela
alpha=12.5*2*pi/60;  % Aceleracion angular
w=50*2*pi/60;        % Velocidad angular
Wt = 45*pi/180; % Angulo manivela

T21 = -mb*r.^2*alpha*sin(Wt).*(sin(Wt)+r/L*sin(2*Wt))...
     -mb*r.^2*w^2/2.*(r/(2*L)*sin(Wt)+sin(2*Wt)+(3*r)/(2*L)*sin(3*Wt));
```

2. La matrices se pueden generar con las siguientes lineas:

```
A = [10:2:14 ; 0.5:-0.1:0.3 ; 3:-0.25:2.5];
c = [-5:5:5]'
B = [(11:14)' (8:-2:2)' (0.25:0.25:1)' ones(4,1)*5]
b = [8 7 9 6]
D = [B b']
```

A*B	A*c	b*B	B*b	B*b'	A.*B	B^f	B.^f	f.^A	f*c
×	✓	✓	×	✓	×	✓	✓	✓	✓

3. El código de solución:

```
F1x = -400;
[F2x,F2y]=pol2cart(pi/4,250);
[F3x,F3y]=pol2cart(pi-acos(4/5),200);
Fx = F1x + F2x + F3x
Fy = F2y + F3y
[th,F]=cart2pol(Fx,Fy)
th = th*180/pi
```

La ayuda de las funciones:

```
CART2POL Transform Cartesian to polar coordinates.
    [TH,R] = CART2POL(X,Y) transforms corresponding elements of data
    stored in Cartesian coordinates X,Y to polar coordinates (angle TH
    and radius R).  The arrays X and Y must be the same size (or
    either can be scalar). TH is returned in radians.


 POL2CART Transform polar to Cartesian coordinates.
    [X,Y] = POL2CART(TH,R) transforms corresponding elements of data
    stored in polar coordinates (angle TH, radius R) to Cartesian
    coordinates X,Y.  The arrays TH and R must the same size (or
    either can be scalar).  TH must be in radians.
```

Capítulo 3

Entrada y Salida de datos

La interacción con el usuario, mejora la presentación y facilita, en algunos casos, la comprensión del programa. La interacción con otros programas, a partir de la generación o lectura de archivos de datos, aumenta la funcionalidad del mismo.

En este capitulo se explican los medios para obtener valores de variables desde el prompt, así como para mostrar los resultados. También, se estudia la forma de guardar datos en archivos planos (*.txt), y cómo cargar los mismos.

3.1. Por pantalla

Para mejorar la presentación de los programas, en algunas ocasiones es necesario que el usuario introduzca datos y también para presentarle al mismo la solución o respuesta del problema.

Para adquirir datos del *prompt* se utiliza la función `input`. Esta permite desplegar un mensaje solicitandole al usuario un valor definido, este valor luego es asignado a la variable determinada. Por ejemplo, se desea que el usuario introduzca el valor de la distancia entre dos poleas en centímetros, y que este valor sea guardado en un variable llamada S para los cálculos posteriores:

```
S=input('Digite el valor de la distancia entre las poleas [cm]: ');
```

Tomemos el ejercicio de la longitud de la correa dados los radios de las poleas y la distancia entre las mismas. Vamos a suponer que el radio mayor R y menor r no cambian, y que el usuario va a modificar la distancia entre poleas S. Para ello se utiliza el siguiente script, digitelo y guardelo con el nombre, `correa.m`:

```
% Determinar la longitud de la correa L dado el radio mayor R,
%  el radio menor r y la distancia entre poleas S.
% correa.m
% Ing. Alfonso Cubillos V.
% Marzo 2005
% Valores de entrada : R , r y S
% Valores de salida : L

% a) Datos de entrada
R = 30; % Radio mayor [cm]
r = 12; % Radio menor [cm]
% Pregunta al usuario el valor S :
 S=input('Digite el valor de la distancia entre las poleas [cm]: ');
```

```
% b) Calculos necesarios
th=asin((R-r)/S);   % Angulo de agarre [rad]
L=2*S*cos(th)+pi*(R+r)+2*th*(R-r); % Longitud de la correa [cm]
```

Guarde el programa en la carpeta seleccionada por el programa, es decir, no cambie de carpeta. Ahora en el prompt digite el nombre del archivo:

```
>> clear all % Borrar todas la variables
>> clc       % Borrar pantalla
>> correa    % Ejecutar el archivo 'correa.m'
```

De esta forma le aparece el mensaje:

```
Digite el valor de la distancia entre las poleas [cm]:
```

Escriba 50 para el valor de S. El programa termina y muestra el prompt en señal que esta listo para realizar más cálculos. Preguntemos la respuesta

```
>> L
L =
 238.4998
```

Pero hay una forma de que MATLAB muestre la respuesta deseada. Para esto agregue la siguiente linea al final del programa `correa.m` :

```
disp(['La longitud de la correa es ' num2str(L) ' cm'])
```

Guarde el programa y de nuevo desde el prompt escriba el nombre del mismo para ejecutarlo. Observe la diferencia.
La función `disp` muestra en pantalla el texto o datos de la variable seleccionada. En el primer caso, debido a que la función solo debe mostrar texto, se deben tener en cuenta 3 cosas que se ven en el ejemplo:

1. El mensaje de texto debe ir entre comillas sencillas `' '`
2. Los numeros se deben convertir de variables numéricas a texto o string con la función `num2str`
3. El texto y los números convertidos a texto deben ser una sola cadena o array y por lo tanto es necesario que todos estén entre corchetes cuadrados `[ ]`

Para mostrar solamente el valor o valores de una variable cambie la última línea de código por:

```
disp(L)
```

y verifique la diferencia.

3.2. Por Archivos

MATLAB también permite guardar y leer datos de archivos; estos archivos pueden ser de EXCEL o en el mejor de los casos, archivos planos texto de *Block de Notas* con extensión *.txt*.

3.2.1. Guardar Datos

Primero guardemos datos, para esto utilice el programa `correa4.m` generado en los ejercicios de este capitulo, obtenga el valor de L para los valores de S entre 50 y 100 con pasos de 2. Después de ejecutar el programa se puede preguntar a MATLAB las variables presentes en memoria,

```
>> whos
Name      Size                    Bytes  Class

 L         1x26                     208  double array
 R         1x1                        8  double array
 S         1x26                     208  double array
 r         1x1                        8  double array
 th        1x26                     208  double array

Grand total is 80 elements using 640 bytes
```

indica que las variables L, S y th son arrays o vectores fila de 26 columnas. Debido a que es más fácil observar los datos como vectores columna, entonces se aplica la transpuesta a los tres vectores, y al mismo tiempo generaremos una matriz llamada `datos` con estos tres vectores

```
>> datos=[L' th' S']
```

verifique el tamaño de la matriz `datos`, esta debe ser de 3 columnas y 26 filas.

Para guardar la matriz `datos` en un archivo plano se utiliza la sintaxis

```
>> save resultado.txt datos -ASCII
```

`save` es la función que se utiliza para guardar datos en archivo, puede mirar la ayuda de la función para ver más posibilidades. La matriz `datos` se guarda en el archivo `resultado.txt` y `-ASCII` define la forma en que se guardan los archivos.
Ahora abra el archivo `resultado.txt` para eso digite la siguiente orden sobre el prompt

```
>> open resultado.txt
```

3.2.2. Cargar datos

Ahora suponemos que tomamos un archivo de datos del banco de vibraciones mecánicas y necesitamos analizar los mismo. Generemos entonces primero el archivo block de notas con los datos manualmente, y vamos a tomar los siguientes datos como ejemplo:

Tiempo	**Velocidad**	**Aceleración**
0	2.3	0
0.1	2.5	0.02
0.2	2.7	0.03
0.3	2.6	0.07
0.4	2.1	-0.1
0.5	2.1	0.0

Entonces abra un archivo block de notas nuevo, para esto en el menú **Inicio** de Windows seleccione **Ejecutar**, y en la ventana que aparece escriba **notepad**. Ahora introduzca los datos de la tabla, pero tenga en cuenta que el cambio de columna se debe realizar por medio de la tecla **TAB**. Cuando termine de copiar los datos, guarde el archivo en la carpeta `C:\MATLAB6p5\work\` con el nombre `vibra`. El archivo se debe ver como lo muestra la Figura 3.2.2.

Ahora en el prompt de MATLAB vamos a cargar el archivo

```
>> load vibra.txt
```

la función `load` carga archivos externos, puede mirar las diferentes opciones con el comando `help` de la función. Ahora verifiquemos que el archivo se ha cargado satisfactoriamente

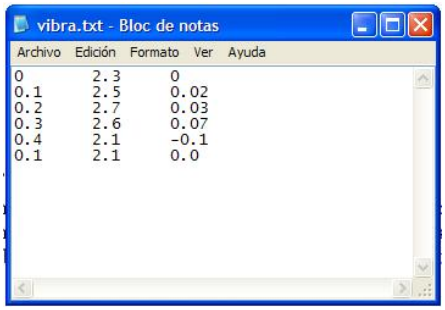


Figura 3.1: Archivo Block de Notas Original

```
>> whos
  Name         Size                    Bytes  Class

 vibra        6x3                       144  double array
```

se ha cargado entonces una variables llamada `vibra` de 6 filas y 3 columnas que son los datos del archivo, pero puede verificar si son los datos correctos. También puede ser importante agregar información al archivo block de notas, estos comentario se pueden agregar con `%` como lo muestra la Figura 3.2.2.

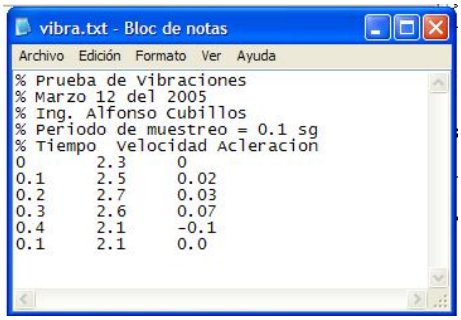


Figura 3.2: Archivo Block de Notas Modificado

al cargar el archivo, MATLAB ignora aquella líneas que empiezan con `%`.

3.3. Gestión de Archivos

Cuando se trabaja con archivos es importante tener en cuenta el *path* o *camino* que toma MATLAB para buscar, abrir o guardar los archivos. Para conocer el path actual se utiliza la función `cd`, por ejemplo:

```
>> cd
    C:\MATLAB6p5\work
```

esto quiere decir que todos los archivos que abra, guarde o cargue MATLAB deben estar dentro de esta carpeta, si el archivo que se quiere cargar no esta en esta carpeta, MATLAB no podrá encontrarlo y generará un error. Para verificar si el archivo que quiero cargar esta en el path actual se puede utilizar la función `dir`, esta muestra todos los archivos que actualmente están en el path. Utilice la función para verificar que el archivo `vibra.txt` y `resultado.txt` se encuentran en el path actual.
Se puede cambiar el path actual con la función `cd`. Por ejemplo cambiemos el path a la carpeta `C:\MATLAB` con la siguiente línea de código:

```
>> cd c:\MATLAB6p5
```

verifique el nuevo path, mire los archivos que están en esta carpeta con el comando `dir`, y regrese a la carpeta `C:\MATLAB6p5\work`.

3.4. Ejercicios

3.4.1. Ejercicio Input y disp

1. Genere un archivo llamado `correa2.m` que pregunte por cada una de las variables del problema: R, r y S, es decir que el usuario pueda definir todos estos valores. Y como salida de nuevo el valor L.

2. Genere un archivo llamado `correa3.m` que tenga como entrada de nuevo todos valores, pero que el usuario los introduzca en forma de array, es decir que el usuario digite [40 20 50] y el programa tiene que tomar este array y realizar los cálculos necesarios.

3. Genere un archivo llamado `correa4.m` que tenga como entrada solo el valor de S pero como array, es decir para diferentes valores de S. El programa debe mostrar como solución dos columnas, la primera con los valores de S y la segunda con L.

3.4.2. Ejercicio Gestión de archivos

1. El proceso de diseño de sistemas termodinámicos, implica la obtención de propiedades termodinámicas principalmente del agua (H_2O). Los cálculos son sencillos pero extensos, y el proceso se dificulta al tener que buscar y algunas veces interpolar valores de las tablas. El objetivo del ejercicio es obtener una función que genere estos valores de forma automática, para esto es necesario generar los archivos con los datos de las tablas.

   Con las tablas de datos, genere un archivo plano (block de notas) que contenga la información de la tabla. Recuerde documentar muy bien los archivos con los nombres de los integrantes, la página asignada, la tabla asignada, la propiedad de cada una de las columnas y sus unidades. Recuerde también que las columnas se separan con la tecla **TAB**. Las páginas a digitalizar son 818, 819, 822-829.

2. El apéndice E del libro "Diseño de Maquinaria"de Robert Norton presenta los factores de concentración de esfuerzo por geometría (K_t) de 14 casos comunes. Los casos 1-6 y 9-12 poseen una tabla junto a la figura. Digitalice en un archivo block de notas la información de las tablas, y guarde el archivo con el formato `APP_E_XX.txt`, donde `XX` indica el número de la tabla. Utilice el siguiente ejemplo como base:

   ```
   % Archivo APP_E_03.txt
   % Factor de concentracion de esfuerzo teorico por geometria K_t
   % Caso E-03 : Barra circular con entalle circunferencial sometida a torsion
   % D/d     A       b
   2.00   0.86331 -0.23865
   1.33   0.84897 -0.23161
   1.20   0.83425 -0.21649
   1.09   0.90337 -0.12692
   ```

3. La Tabla A-5 del Apéndice del libro DISEÑO EN INGENIERIA MECANICA de Robert Shigley, muestra las constantes elásticas y físicas de materiales comunes. Genere un archivo Block de Notas que posea todos los valores numéricos de la tabla, conservando su orden.

Capítulo 4

Control de Flujo de Programa

Para controlar el orden o el camino en el cual se desarrollan las expresiones u operaciones de un programa, es necesario utilizar *estructuras de control de flujo*. Estas estructuras se dividen principalmente en dos tipos: las instrucciones de *decisión* y las instrucciones de *iteración*. Estos elementos están disponibles en MATLAB como en muchos otros lenguajes de programación. Para decisiones, se posee el `if` y el `switch`, y para iteraciones se cuenta con el `for` y `while`. Cada vez que aparezca uno de estos operadores, este debe ser precedido (en alguna parte del mismo programa) por el comando `end`. Cada una de las estructuras de control pueden aparecer las veces que sea necesario, tanto individualmente, como incluidas dentro de otras. Cuando esto ocurre, se denominan *estructuras anidadas*.

En las estructuras de control frecuentemente es necesario utilizar datos lógicos o booleanos (Sección 1.2.3) que se obtienen principalmente a partir de las operadores relacionales (Sección 1.4.2) y los operadores lógicos (Sección 1.4.3). MATLAB primero evalúa las operaciones relacionales y/o lógicas para obtener un resultado lógico, el cual es evaluado en la estructura de control para indicarle al programa las operaciones a realizar o el camino a tomar.

4.1. Instrucciones de Decisión

Las instrucciones de decisión, se trabajan con las estructuras de selección de instrucciones y sirven para la solución de casi todos los tipos de problemas cuando en el desarrollo de la solución es necesario tomar una decisión, con el fin de establecer qué camino debe seguir el programa.

4.1.1. Sentencia IF

En la estructuras de selección de instrucciones (**IF**), cuando la *condición* se cumple tomará el camino del **SI** o **verdadero** y ejecuta el *bloque* asignado en este camino, si la *condición* no se cumple se irá por el camino **NO** o **falso** y se ejecuta el *bloque* de dicho camino; en cualquiera de los dos casos, después de ejecutar las instrucciones que se encuentren en el camino, saldrá de la estructura e irá a ejecutar las instrucciones que se encuentren por fuera y a continuación de ella (donde se encuentra el `end`), hasta encontrar el fin del proceso.

La estructura más sencilla es la que muestra la figura 4.1, en ella solo se ejecuta un bloque de instrucciones cuando la condición se cumple.

Por ejemplo, dadas las notas de una asignatura, se debe calcular el promedio: **si** *la nota es mayor o igual a 3* **entonces** mostrar un mensaje que indique "pasó la signatura". Para solucionar este problema se puede utilizar el siguiente código en MATLAB:

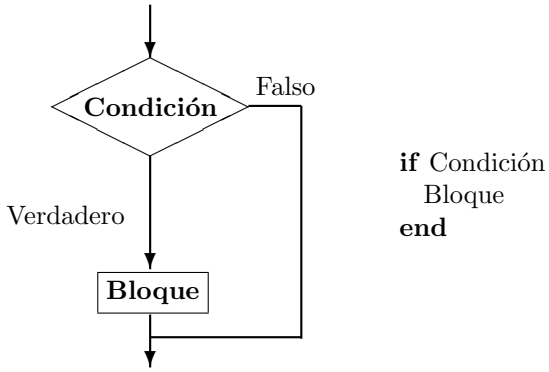


Figura 4.1: Estructura **IF** simple

```
% Calculo de promedio y selector de alumnos
clc
notas = [3.0 4.2 2.1];  % Notas durante el semestre
prom = mean(notas);     % Promedio de las notas
if prom >= 3    % Condicion
  disp(['Paso la asignatura con ' num2str(prom)]) % Bloque
end
```

En este caso, cuando el promedio o nota final es menor a 3, no se realiza ninguna acción, para solucionar este tipo de problemas se utiliza la estructura de la figura 4.2.

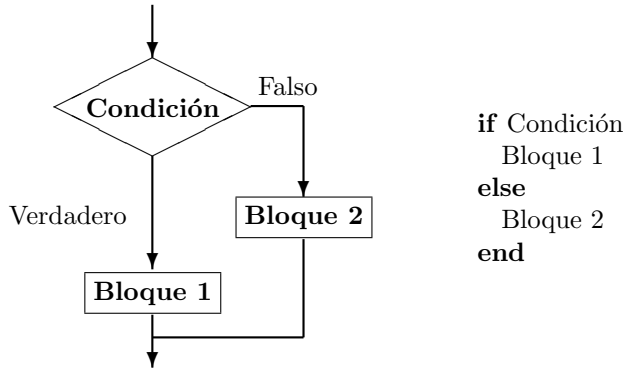


Figura 4.2: Estructura **IF** con doble opción

Como ejemplo: **si** la nota del estudiante es mayor o igual a 3, muestre el mensaje "Paso la asignatura", **si no** muestre el mensaje "Perdio la asignatura". El problema se soluciona con el código en MATLAB:

```
% Calculo de promedio y selector de alumnos
notas = [3.0 4.2 2.1];  % Notas durante el semestre
prom = mean(notas);     % Promedio de las notas
if prom >= 3    % Condicion
  disp(['Paso la asignatura con ' num2str(prom)]) % Bloque 1
else
  disp(['Perdio la asignatura con ' num2str(prom)]) % Bloque 2
end
```

Ahora bien, aumentando la complejidad del ejercicio: **si** la nota del estudiante es mayor o igual a 3, muestre el mensaje "Paso la asignatura": **si** la nota del estudiante NO es mayor o igual a 3.0 pero es mayor a 2.7 (2.7<nota<3.0) el esudiante puede habilitar y la habilitación se promedia de nuevo con la nota final; **si no** muestre el mensaje "Perdio la asignatura".

Este tipo de problemas se puede solucionar con la estructura de la figura 4.3, y el código que soluciona el problema es:

```
clc
notas = [3.0 3.2 2.0];  % Notas durante el semestre
prom = mean(notas);     % Promedio de las notas
if prom >= 3   % Condicion 1
  disp(['Paso la asignatura con ' num2str(prom)]) % Bloque 1
elseif prom > 2.7  % Condicion 2
   falta=3*2-prom;    % Bloque 2
   disp(['Habilita la asignatura con ' num2str(prom)])
   disp(['y le falta ' num2str(falta) ' para pasar'])
else
 disp(['Perdio la asignatura con ' num2str(prom)]) % Bloque 3
end
```

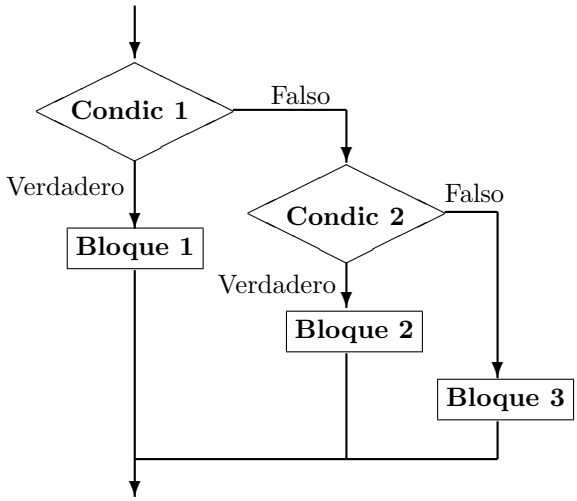


Figura 4.3: Estructura **IF** con doble condición

Verificando el concepto con otro ejemplo: Se desea determinar si un número cualquiera es mayor 0, muy sencillo en MATLAB:

```
clc
x = 8;  % Variable a modificar
if x > 0
   disp([num2str(x) ' es mayor a cero'])
end
```

ahora se desea saber si el número es mayor o menor a cero:

```
clc
x = 8;  % Variable a modificar
```

```
if x > 0
   disp([num2str(x) ' es mayor a cero'])
else
   disp([num2str(x) ' es menor a cero'])
end
```

bastante sencillo, ahora, si el numero es mayor a cero, se desea saber también si es múltiplo de 2. Para realizar esto se puede utilizar la función `rem`, esta devuelve el *resto* de la división de enteros, así por ejemplo, 3/2 es igual a 1 y sobra 1, esto quiere decir que 3 no es múltiplo de 2. Otro ejemplo, 6/2 es igual 3 y sobra 0, que dice que 6 múltiplo de 2. Sin más preámbulos el código:

```
clc
x = 8;  % Variable a modificar
if x > 0
   disp([num2str(x) ' es mayor a cero'])
   if rem(x,2)==0
     disp([num2str(x) ' es multiplo de 2'])
   end
else
   disp([num2str(x) ' es menor a cero'])
end
```

ahora, además de la anterior, determinar si el número es multiplo de 2 y 3 al mismo tiempo

```
clc
x = 8;  % Variable a modificar
if x > 0
   disp([num2str(x) ' es mayor a cero'])
   if rem(x,2)==0
     disp([num2str(x) ' es multiplo de 2'])
     if rem(x,3)==0
        disp([num2str(x) ' es multiplo de 3'])
     end
   end
else
   disp([num2str(x) ' es menor a cero'])
end
```

como se puede apreciar, los condicionales pueden ir *anidados* es decir, uno dentro de otro. Por último último, se desea saber *solamente* si el número es múltiplo de 2 y 3:

```
clc
x = 8;  % Variable a modificar
if rem(x,2) == 0 & rem(x,3) == 0
   disp([num2str(x) ' es multiplo de 2 y 3'])
end
```

4.1.2. Sentencia SWITCH

La sentencia **switch** realiza una función análoga a un conjunto de `if..elseif` concatenados. Su forma general es como la muestra la figura 4.4.

Al principio se evalúa la *Expresion switch*, cuyo resultado debe ser un número escalar o una cadena de caracteres. Este resultado se compara con *caso #*, y se ejecuta el bloque de sentencias que corresponda con ese resultado. Cuando se ejecuta el bloque asignado, pasa a la línea del `end`. Si ningún *caso #* es igual a *Expresion switch* se ejecutan las sentencias correspondientes a *otherwise* (en otro caso).

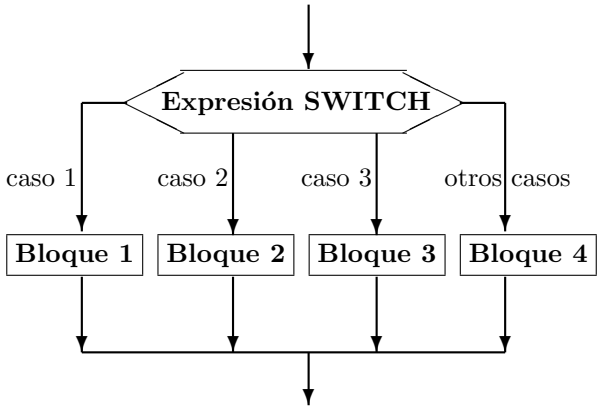


Figura 4.4: Estructura **SWITCH**

El siguiente código con la estructura `switch` se toma como ejemplo (el valor de la variable k debe ser asignada o calculada antes de utilizar el código).

```
a=3;
switch k
   case 1
       disp('Caso 1')
   case {2,3}
       disp('Caso 2 o 3')
   case a^2
       disp('Caso 9')
   otherwise
       disp('En otro caso ...')
end
```

Otro ejemplo, generar un conversor de unidades, transformar un valor en metros a otra unidad de longitud:

```
unid=input('Digite la unidad a llegar: mm, cm, ft, in : ','s');
valor=input('El valor a convertir: ');

switch unid
   case 'mm'
       valor_f=valor*1000; % Valor en milimetros
   case 'cm'
       valor_f=valor*100; % Valor en centimetros
   case 'ft'
       valor_f=valor*3.281; % Valor en pies
   case 'in'
       valor_f=valor*39.37; % Valor en pulgadas
   otherwise
       error('Valor no valido')
end

disp([num2str(valor) ' metros, son ' num2str(valor_f) ' ' unid])
```

4.2. Estructuras de Repetición o Iteraciones

Para realizar iteraciones o repetir un conjunto de instrucciones los lenguajes de programación contienen generalmente dos comandos especiales, cualquiera de ellos se puede utilizar según la facilidad o tarea a realizar. Sirven también para obtener varios valores o resultados, a diferencia como se venía trabajando en donde solo se generaba un valor o resultado.

En este tipo de estructuras, es necesario conocer unos conceptos adicionales:

Contador : Es una variable numérica de tipo entero (por lo general) que sirve para llevar una cuenta con incrementos o decrementos constantes.

```
M = M + 1;
```

- En este caso el incremento es 1, cada vez que se pase por dicha línea.
- Al lado y lado del igual se coloca o escribe la misma variable, esto con el fin de llevar la cuenta del valor anterior.
- También es necesario notar que `M` debe tener un valor inicial (inicializar la variable) para poder comenzar a ejecutarse.

Acumulador: Es una variable que sirve para guardar y acumular valores que pueden ser diferentes cada vez, es decir, es una variable en la cual se puede ir calculando una suma de los valores que toma otra variable dentro del algoritmo.

```
ACUM = ACUM + V
TOT = TOT + NUM
```

- El nombre del acumulador se escribe a lado y lado del signo igual por que es necesario que cada vez que se pase por dicha linea, se comience por el valor que había quedado antes.
- El valor que se suma o incrementa puede ser diferente cada vez.
- Para poder efectuar la operación de acumulación es necesario que tanto el acumulador como el valor que se este acumulando, hayan tenido antes un valor inicial (inicializar) con el cual se puede comenzar a trabajar.

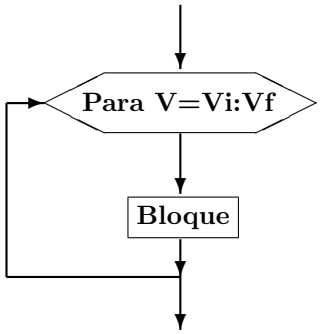
4.2.1. Sentencia FOR

La estructura repetitiva *Para ...* (FOR) es aquella en el que el número de iteraciones de bucle o ciclo es determinado directamente, por lo tanto se sabe en que valor debe comenzar y en que valor debe terminar el contador.

Este ciclo se utiliza cuando el programador conoce el principio y el fin del ciclo, es decir conoce los límites y por consiguiente es quien ordena la salida del ciclo.

Esta estructura como muestra la figura 4.5, ejecuta las instrucciones del ciclo un número determinado de veces y controla la forma automática el número de repeticiones o pasos que se tienen que llevar a cabo. Cuando termina de ejecutar las instrucciones del ciclo automáticamente sale de la estructura para continuar ejecutando el resto del programa. En esta estructura esta claramente definido en qué valor el contador debe empezar y terminar.

- Si el valor inicial es menor que el valor final, el incremento será positivo.
- Si el valor inicial es mayor que el final, el incremento al valor será negativo, es decir, decremento.
- El incremento del contador será siempre 1 automáticamente, a menos que se indique otra cosa.



for contador=inicio **:** final
Bloque
end

for contador=inicio **:** incremento **:** final
Bloque
end

for contador = VectorValores
Bloque
end

Figura 4.5: Estructura **FOR**

- Cuando el contador haya llegado a su valor final ejecuta por última vez las instrucciones que se encuentran dentro de la estructura y luego se sale automáticamente de esta.

Como ejemplo: calcular la sumatoria de 6 números digitados por el usuario. El programa entonces pregunta al usuario por cada uno de los 6 valores, y va calculando al mismo tiempo la suma de los mismos:

```
total=0; % Inicializacion del acumulador

for i = 1 : 6 % Contador
    valor=input(['Digite el valor ' num2str(i) ' : ']);
    total=total+valor;  % Acumulador
end

disp(['La suma total es igual a ' num2str(total)])
```

Este problema se puede realizar de otra forma, definiendo el vector del contador de antemano:

```
total=0; % Inicializacion del acumulador
vector = 1 : 6;

for i = vector  % El contador esta en el vector
    valor=input(['Digite el valor ' num2str(i) ' : ']);
    total = total + valor;  % Acumulador
end

disp(['La suma total es igual a ' num2str(total)])
```

Otro ejemplo, calcular la suma de los números pares entre 2 y 50. Para esto también se puede utilizar un ciclo **for**:

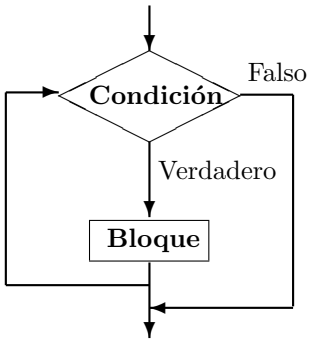
```
total=0; % Inicializacion del acumulador

for i = 2 : 2 : 50  % Inicia en 2, incrementos de 2 y termina en 50
    i               % Muestra el contador
    total=total+i;  % Acumulador
end

disp(['La suma total es igual a ' num2str(total)])
```

4.2.2. Sentencia WHILE

La estructura *WHILE*, se inicia y continúa mientras la condición se cumple (sea verdadera) y finaliza cuando la condición no se cumple (sea falsa). Se acostumbra a colocar antes del ciclo la inicialización de la variable de la condición, para que esta se cumpla al menos la primera vez; por ejemplo si tiene un ciclo mientras que una variable sea diferente a cero (0), la variable se inicializa con un valor diferente de 0 antes de empezar el ciclo.

La figura 4.6 muestra la estructura del ciclo, en esta, mientras se cumpla la condición, se ejecutarán las operaciones que se encuentran por el camino de **Verdadero** o **Si**, cuando la condición se deje de cumplir tomará el camino opuesto es decir el **Falso** o **No**, y realizará por dicho camino el resto de operaciones.

En el uso de la sentencia WHILE se pueden presentar bucles infinitos, estos se generan cuando la condición se cumple de forma indeterminada.



while Condición
  Bloque
end

Figura 4.6: Estructura **WHILE**

La **condición** puede ser una expresión vectorial o matricial y de esta forma el bloque se sigue ejecutando mientras haya elementos distintos de cero en el vector o matriz **condición**, es decir, mientras haya algún o algunos elementos **true**. El bucle se termina cuando *todos los elementos* de **condición** son **false** (cero).

Como ejemplo: crear un programa donde se lean números hasta que el usuario lo desee digitando cero (0). En este ejemplo, el programador no sabe cuantas veces tiene que preguntar por el valor, y por lo tanto lo mejor es utilizar la función **while**.

```
h = 1;
while h~=0
   h=input('Digite un valor ');
end
```

Aspectos importantes a tener en cuenta en el programa:

- La condición en este caso `h~=0` (h diferente de cero) garantiza que el ciclo se ejecute *mientras* la condición se cumpla, es decir, *mientras* el usuario digite un valor diferente de cero. Cuando el usuario digita cero, el valor es guardado en la variable `h` y se evalúa la condición de nuevo, como la condición en este caso es falsa, entonces no ejecuta las instrucciones y sale de la estructura.

- Para poder entrar la primera vez al ciclo es necesario *inicializar* la variable de la condición, y garantizar que se cumpla al menos una vez la condición, para este caso, la variable `h` inicializada puede tener cualquier valor diferente de cero.

- Dentro del ciclo, el valor digitado por el usuario es guardado en la variable de la condición `h`, esto garantiza que el ciclo tiene fin.

Otro ejemplo: se sabe que la serie infinita

$$\sum_{n=1}^{+\infty} \frac{1}{n(n+1)} = \frac{1}{2} + \frac{1}{6} + \frac{1}{12} + \frac{1}{20} + \ldots + \frac{1}{n(n+1)} \ldots$$

converge. Determine el valor al cual la serie converge.

Para determinar la convergencia de la serie, se puede utilizar la expresión

$$\sum_{n=1}^{m-1} \frac{1}{n(n+1)} - \sum_{n=1}^{m} \frac{1}{n(n+1)} \approx 0$$

es decir, cuando el valor de la sumatoria hasta m es aproximadamente igual a la sumatoria hasta $m-1$, la serie ha convergido. El código es entonces:

```
dif = 1; % Inicializacion de la diferencia
n = 2; % Inicializacion del contador
S(1) = 1/2; % Primer valor de la serie
suma(1)=S(1); % Primer valor del sumador

while dif > 1e-9
   S(n)=1/(n*(n+1)); % Evaluacion de la serie en n
   suma(n)=suma(n-1)+S(n); % Acumulador
   dif=abs(S(n-1)-S(n));  % Diferencia del actual con el anterior
   n=n+1;      % Contador
end

suma(end)   % Ultimo valor del acumulador
plot(suma)  % Grafica de convergencia
```

4.3. Ejercicios

1. *Conversor de unidades*: Realizar un programa que realice conversión de unidades de longitud, el usuario selecciona la unidad base (`m,cm,ft,in`), la unidad a la cual desea llegar (`m,cm,ft,in`) y el valor a convertir.

2. Leer 3 números enteros diferentes y determinar cual de ellos es mayor y verificar la respuesta con la función `max`. Realizar la prueba con los valores que aparecen en la tabla.

9	5	3	,	5	9	3	,	5	3	9
3	5	9	,	9	3	5	,	3	9	5

3. Se captura un número entero, mostrar un mensaje indicando si el número es múltiplo 2, 5 y 7 a la vez.

4. Considere las relaciones que gobiernan el factor de corrección usado para estimar el esfuerzo por fatiga en materiales metálicos:

Factor	**Rango**	**Corrección**
Tipo de carga	Voladizo	$C_{carga} = 1$
	Axial	$C_{carga} = 0{,}70$
Tamaño	$d \leq 8\ mm$	$C_{tam} = 1$
	$8 \leq d \leq 250\ mm$	$C_{tam} = 1{,}1189 d^{-0{,}097}$
Temperatura	$T < 450\ C$	$C_{temp} = 1$
	$450\ C \leq T$	$C_{temp} = 1 - 0{,}0032(T - 840)$

Después de calcular cada uno de los factores de corrección se calcula el esfuerzo real:

$$\sigma_{real} = \sigma * C_{carga} * C_{tam} * C_{temp}$$

En el código se debe definir de antemano σ (`Sigma`), `d` y `T` como valores numéricos, y el tipo de `carga` como un string. Y devolver al final del código σ_{real} (`Sigma_real`).

El programa también debe mostrar mensajes de error si el tipo de carga no es la adecuada y si el diámetro es negativo o mayor a 250 mm.

5. Un rey persa, al conocer el juego de ajedrez quiso pagarle a su creador por tan fantástico juego. Después de mucha insistencia, el creador del juego pidió como recompensa el total de granos de arroz, si en cada casilla del tablero, se pone el doble de granos de la casilla anterior, empezando en 1. Es decir, en la primera casilla se pone un grano, en la segunda dos granos, en la tercera cuatro granos, en la cuarta ocho granos, y así las 64 casillas del tablero. Calcule el total de granos necesarios para pagarle al creador del juego. Además, si 3 granos pesan 0.1 gramos, cual es el peso total del arroz a pagar.

6. La Tabla A-9 del Apéndice del libro DISEÑO EN INGENIERIA MECANICA de Robert Shigley, muestra la defección de vigas sometidas bajo diferentes modelos de carga. La tabla muestra un modelo del sistema con sus variables, así como las ecuaciones para el momento flector, esfuerzo cortante y desplazamiento de la viga en función de la posición x sobre la viga.
Escriba un código teniendo en cuenta las siguientes consideraciones:

   - Cada grupo de dos estudiantes toma 1 modelo de carga según la siguiente tabla:

Grupo	**Modelo de carga**	**Grupo**	**Modelo de carga**
1	1	6	6
2	2	7	7
3	3	8	8
4	4	9	9
5	5	10	10

   - El programa debe solicitar todas las variables geométricas y de carga incluidas en el análisis de cada modelo de carga.
   - Para el modulo de elasticidad E, el programa debe dar 4 opciones: 3 para escoger 3 materiales incluidos en la Tabla A-5 del mismo libro, la cuarta opción es para incluir el valor de la elasticidad como valor numérico.
   - La inercia I debe ser un valor digitado por el usuario.
   - Una posición x para calcular el momento flector M, el esfuerzo cortante V y la deflexión y en esa posición.
   - El programa debe hacer los respectivos cálculos y mostrar la delexión máxima, y los valores M, V y y para el valor x seleccionado.

7. Calcular e imprimir el factorial del cualquier número entero, comenzando en 1 y terminando en el número que el estudiante desee. Comparar el resultado con la función `factorial`.

8. Realizar todos los paso necesarios para calcular

$$1 + 1/2 + 1/3 + 1/4 + \ldots + 1/n$$

donde n es un valor conocido.

9. Se toman dos arrays del mismo tamaño A y B. Sumar el primer elemento de A con el último de B, el segundo de A con el penúltimo de B, y así sucesivamente.

10. Determinar el valor al cual converge la sumatoria

$$\sum_{n=1}^{+\infty} \left(\frac{3}{2^n} - \frac{2}{3^n} \right)$$

11. Se considera la serie definida por

$a_1 = 0$
$a_2 = 1$
$a_n = 3 * a_{n-1} + 2 * a_{n-2}$ Para $n > 3$

Se desea obtener el valor y el rango del primer término que sea mayor o igual a 100.

12. Se desea organizar un arreglo de `n` elementos ya sea de mayor a menor o de menor a mayor. Desarrolle un algoritmo que permita seleccionar el modo de organizar dichos elementos y arroje como resultado una lista con los números ya organizados de acuerdo con la solución tomada. Compare la solución con el comando `sort`.

13. Desarrolle un algoritmo que permita ubicar la posición de un elemento dentro de una matriz A de `n` filas por `m` columnas, conocido el valor del elemento. La analogía puede ser un vecindario con calles (filas) y avenidas (columnas) donde un cartero desea entregar una carta. El cartero conoce el apellido del destinatario (elemento dentro de la matriz) y posee un mapa donde puede ubicar el apellido (Matriz). La tarea es, dado el apellido y el mapa encontrar la dirección del destinatario.

14. Algunas calculadoras calculan la raíz cuadrada de un número positivo A usando un algoritmo iterativo (que se repite hasta alcanzar el valor deseado), dicho algoritmo es de la de la siguiente forma:

$$R(n) = \frac{1}{2} \left[R(n-1) + \frac{A}{R(n-1)} \right]$$

para `n = 2, 3, 4 ...`, donde $R(1)$ es un valor con el cual se inicia la iteración (estimación inicial de la raíz de A) luego para $n = 2$, se obtiene $R(2)$ conocido $R(1)$ y cada valor obtenido se vuelve a introducir en la ecuación principal hasta que $R(n) \approx R(n-1)$. El criterio para acabar la iteración, puede ser la diferencia entre el valor $R(n)$ y el último calculado $R(n-1)$.
Los parámetros de entrada al algoritmo deben ser:

- A: Número a calcular la raíz.
- P: Tolerancia (diferencia entre $R(n)$ y $R(n-1)$). El algoritmo debe finalizar cuando este valor de tolerancia se satisfaga.
- $R(1)$: Valor con el cual se inicia la estimación.

15. Solicitar al usuario un numero cada vez hasta que digite cero. Calcule la suma de los números dados por el usuario.

16. Solicitar al usuario un número cada vez hasta que la suma de los números dados sea mayor o igual a 100.

4.4. Solución a problemas propuestos

1. Dada la siguiente lista de materiales con su respectivo modulo de elasticidad:

Material	Modulo de Elasticidad
Aluminio	10.3
Cobre	17.2
Latón	15.4
Acero	30
Plomo	5.3

   Escriba un código que pregunte al usuario el material, y el programa debe mostrar el modulo de elasticidad del mismo.

2. Determinar el valor al cual converge la sumatoria

$$\sum_{n=1}^{+\infty} \left(\frac{2}{3}\right)^n$$

3. Se posee la matriz `datos` con los siguientes valores:

2	0
5	5
6	16
10	29
15	45
23	50
32	53
40	59
41	63

   Dado un valor de la primera columna, el programa debe buscar el compañero de la segunda columna y guardarlo en una variable llamada `soluc`.

4. Escribir el código necesario para determinar el valor de la sumatoria para cualquier valor m

$$\sum_{n=1}^{m} \left(\frac{3}{2^n} - \frac{2}{3^n}\right)$$

5. Solicitar al usuario un número cada vez hasta que la suma de los números dados sea mayor o igual a 100.

6. Se captura un número entero, mostrar un mensaje para conocer que el número es múltiplo 2, 3 y 5 a la vez, o que no lo es.

Solución

1. En este caso, es necesario hacer la selección en base al nombre del material, por lo tanto, lo mejor es utilizar una estructura de tipo *switch -case*.

```
% Pedir el nombre del material
mat = input('Especifique el material : ','s'); % Especificar el material

% Realizar la seleccion automatica
switch mat
   case 'Aluminio'
```

```
        E = 10.3;
    case 'Cobre'
        E = 17.2;
    case 'Laton'
        E = 15.4;
    case 'Acero'
        E = 30;
    case 'Plomo'
        E = 5.3;
    otherwise
        E = 0;
        error('El material no se encuentra')
end

% Responder a la pregunta
disp(['El modulo de elasticidad del ' mat ' es ' num2str(E)])
```

2. La sumatoria hasta el infinito de $(2/3)^n$. En este caso, no se conoce el numero de iteraciones necesarias, entonces se utiliza un ciclo WHILE, y como condición se utiliza la diferencia entre el valor actual y el anterior de la serie.

```
% Inicializacion de los contadores
dif = 1;    % Inicializacion de la diferencia
n = 2;      % Inicializador del contador
S(1) = 2/3;       % Primer valor de la serie
suma(1) = S(1);   % Primer valor del sumador

% Ciclo de iteraciones
while dif > 1e-9
    S(n) = (2/3)^n; % Evaluacion de la serie en n
    suma(n) = suma(n-1) + S(n);  % Acumulador
    dif = abs(S(n-1) - S(n));    % Diferencia del valor actual con el anterior
    n = n + 1;      % Incrementar en contador
end

% Respuestas
suma(end)  % Valor al cual converge la serie
plot(suma) % Grafica de la convergencia de la serie
```

3. Buscar el compañero del valor en la tabla. En este caso es necesario utilizar dos estructuras, una iteración para buscar en cada una de las posiciones de la tabla, y una estructura de decisión para comparar el valor buscado con cada una de los valores de la tabla.

```
% Tabla de datos
tabla = [2 5 6 10 15 23 32 40 41 ; 0 5 16 29 45 50 53 59 63]';
v1 = 15; % Valor de la primera columna

% Busqueda del valor en la tabla
for i = 1 : length(tabla)
    if tabla(i,1) == v1
        v2 = tabla(i,2);
```

```
    end
end

% Respuesta
v2
```

4. En el caso de la sumatoria se conoce el número de iteraciones, por lo tanto, lo mejor es utilizar un ciclo *FOR*.

```
% Inicializacion de vbles
m = 10;     % Numero de iteraciones
suma = 0;   % Inicializacion del acumulador

% Calculos Iterativos
for n = 1:m
    S(n) = 3/2^n - 2/3^n;   % Valor de la funcion
    suma = suma + S(n);     % Acumulacion
end

% Respuesta
suma
```

5. El programador no conoce los valores que el usuario especifique, por lo tanto no conoce el número de veces que tendrá que preguntar el número. Con este análisis, se concluye que lo mejor es utilizar un ciclo *WHILE*, y la condición, que la suma no sea mayor a 100.

```
suma = 0;   % Inicializacion del acumulador

% Iteraciones
while suma < 100
    valor = input('Digite un valor : ');
    suma = suma + valor;
end

% Respuesta
suma
```

6. Para averiguar si el valor es múltiplo de 2, 3 y 5, es necesario utilizar la función `rem` y el operador lógico `&`.

```
valor = 60;  % Valor a evaluar

% Evaluacion de las condiciones
if rem(valor,2) == 0 & rem(valor,3) == 0 & rem(valor,5) == 0
   disp([num2str(valor) ' es multiplo de 2, 3 y 5'])
else
   error('El valor no es multiplo de 2, 3 y 5')
end
```

Capítulo 5

Funciones

Las funciones son módulos o partes en que se divide un programa, permiten que este sea más fácil de entender, ubicar rápidamente errores, evitar la redundancia del código, en fin, da facilidad al programador. En programación se recomienda siempre trabajar modularmente, es decir, utilizando funciones, es por eso que por lo general un programa muy grande esta conformado por otros pequeños programas.

La función es un conjunto de instrucciones cuyo objetivo principal es retornar un valor (o varios valores) a partir de otro valor (o varios valores).

La creación de funciones y sus diferentes usos con MATLAB son descritos en este capitulo, además, son presentadas algunas funciones (ToolBox) de MATLAB que son frecuentemente usadas para obtener la solución de problemas en ingeniería.

5.1. Por qué usar funciones

Hay muchas razones para crear una función además de las funciones que incluye MATLAB:

1. Evitar código que se repite
2. Limitar el efecto de cambios, para secciones específicas de un programa.
3. Promover la reutilización de partes del programa en otros.
4. Aislar operaciones complejas.
5. Aumentar la portabilidad.
6. Hacer más fácil el desarrollo y aislar los errores.
7. Aumentar el desempeño, por que cada función puede ser "optimizada".

5.2. El nombre de las funciones

El nombre de la función debe ser escogida de forma tal que sea significativa e indique que hace la función. La longitud del nombre es normalmente entre 9 y 20 caracteres. No debe contener espacios y tampoco puede empezar por un número. Tampoco puede ser el nombre de una variable declarada, ni de otra función ya creada o de una palabra reservada como `if`, `switch`, `case`, etc

5.3. Crear una nueva función

Un archivo con extensión *.m* puede ser un *script* (sección 1.9) o una *función*. Una función es un script que posee sus propias variables, es decir que no afectan a otras funciones o al script que usa la función.

Una de las formas de generar una nueva función, es por medio de un archivo independiente que contenga el código de la misma, este archivo debe poseer al menos dos líneas, donde la primera es el *formato* requerido por MATLAB para las funciones (Interfase de la función).

El nombre del archivo debe ser el mismo de la función, solo que con la extensión *.m*.

El número, tipo y nombre de las variables de entrada y salida de la función, son controladas por la interfase de la misma, la cual es la primera linea no comentada del archivo de la función. En general el archivo de una función puede contener la línea de interfase, algunos comentarios y una o más expresiones como se muestra a continuación:

```
function [Vsal1,Vasl2] = NombreFuncion(Ventr1,Ventr2)
% Comentarios
Expresiones
```

Donde `Vsal1` y `Vsal2` son las variables de salida de la función y van separadas comas (,). `Ventr1` y `Ventr2` son las variables de entrada de la función y deben separadas por comas (,). `NombreFuncion` es el nombre de la función y del archivo que contiene la función, éstos tienen las restricciones ya descritas. La primera palabra `function` es una palabra reservada que debe ser usada sólo en este contexto.

Una de las importantes características de una función es el *espacio de trabajo local*. Cualquier argumento u otra variable creada mientras la función se ejecuta, están sólo disponibles para las declaraciones dentro de la misma función. De igual forma, la variables en el espacio de trabajo de la linea de comandos (llamada *espacio de trabajo base*) normalmente no están disponibles para las funciones. Si durante la ejecución de la función, mas funciones son llamadas, cada una de ellas posee a su vez su propio espacio de trabajo. Esta restricción es llamada *scoping*, y ella hace posible escribir programas complejos sin preocuparse por los conflictos de nombre de las variables. Los valores de los argumentos de entrada son copiados directamente de los datos originales, entonces cualquier cambio que se realice no afecta fuera de la función. En general, la única comunicación entre la función y el script que la llama, es a través de los argumentos de entrada y salida en la *interfase*.

Los comentarios que siguen a la interfase de la función, son usados por MATLAB para crear la ayuda de la función, esto es, cuando se escribe:

```
help NombreFuncion
```

Todos los comentarios iniciales aparecen en el **Command Window** o **prompt**. Cualquier comentario que aparezca antes del comando `function` no aparecerá en la ayuda del programa. La información de ayuda termina cuando no aparecen dos líneas de comentario seguidas, esto es, cuando una línea en blanco o una expresión ejecutable se encuentra.

Veamos un ejemplo, crear una función que calcule las raíces de un polinomio de segundo orden, por medio de la ecuación cuadrática.

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Como ve las variables de entrada (necesarias para realizar las operaciones) son a, b y c. Y las variables de salidas (las que el programa va a calcular) son x_1 y x_2. El nombre de la función va a ser `quadform`. Entonces la función queda de la forma:

```
function [x1,x2] = quadform(a,b,c)
% [x1,x2] = quadform(a,b,c)
% Calcula las raices del polinomio de tercer orden
```

```
%   a x^2 + b x + c = 0

% Esto no aparece

 d = sqrt(b^2-4*a*c);
 x1 = (-b + d) / (2*a);
 x2 = (-b - d) / (2*a);
```

Y desde MATLAB se puede llamar la función:

```
>> [r1,r2] = quadform(1,-2,1)
r1 =
    1
r2 =
    1
```

O se puede puede solicitar la información de la función:

```
 >> help quadform
     [x1,x2] = quadform(a,b,c)
     Calcula las raices del polinomio de tercer orden
     a x^2 + b x + c = 0
```

Un M-file simple puede contener mas de una función definida. Una nueva función en la cabecera del archivo es la *función primaria* y las siguientes son *subfunciones*. Como ejemplo simple considere:

```
function [x1,x2] = quadform(a,b,c)

 d = discrim(a,b,c);
 x1 = (-b + d) / (2*a);
 x2 = (-b - d) / (2*a);

 function D = discrim(a,b,c)
 D = sqrt(b^2 - 4*a*c);
```

Una subfunción tiene su propio espacio de trabajo; por lo tanto, los cambios realizados dentro de `discrim` no se pueden propagar dentro de `quadform`. Además, las subfunciones tienen limitado alcance. En el ejemplo la subfunción `discrim` esta disponible *solo* en la función primaria `quadform`, y no en la línea de comando.

Otra importante característica de las funciones de MATLAB, es que la mayoría de las mismas (excepto algunas especiales) son M-files los cuales pueden ser leídos o copiados. Este es un excelente via para aprender buenas prácticas de programación—y trucos sucios.

5.4. Grupos de funciones - TOOLBOX

La agrupación de funciones especiales para algún propósito específico se denomina *Toolbox*. Estos toolbox se generan y se pueden distribuir en la red, generando grupos de trabajo y desarrollo a través de éstos. MATLAB ha incluido en su cd de instalación algunos de estos toolbox, y trabaja en la generación de muchos más. Cada uno de estos ToolBox posee un costo de licencia, pero también hay otros que son generados por universidades y organizaciones, las cuales no poseen ánimo de lucro y los ToolBox se pueden descargar, utilizar y modificar sin ninguna restricción.

Se pueden ver que ToolBox están instalados en MATLAB aplicando el comando `ver`.

5.4.1. ToolBox Symbolic

Uno de los ToolBox incluidos por MATLAB, es el *Symbolic ToolBox* (ToolBox de Matemática Simbólica), y añade a MATLAB la capacidad de realizar cálculos simbólicos. Entre otros, los principales tipos de operaciones soportadas son la siguientes:

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

Lo primero que se debe hacer para manipular expresiones simbólicas, es definir las variables que esta posea, por medio del comando 'syms' seguido por las variables separadas por espacios:

```
>> syms a x % Definicion de las variables (a x) como simbolicas
```

y después se pueden definir las operaciones simbólicas:

```
>> f = a*cos(x)^2 + a*sin(x)^2
```

Ya definidas las variables y la función, es posibles realizar diferentes operaciones simbólicas:

```
>> diff(f) % Derivada de la funcion respecto a (x) (por defecto)
>> diff(f,a) % Derivada de la funcion respecto a (a)
>> int(f)  % Integral de la funcion f
>> int(f,0,1) % Integral de la funcion f (evaluada entre 0 y 1)
>> simplify(f)  % Simplifica la funcion
```

También se puede definir matrices de funciones:

```
>> R = [ cos(a*x) sin(a*x) ; -sin(a*x) cos(a*x) ]
>> pretty(R)        % Muestra las ecuaciones de forma 'bonita'
```

Una función simbólica muy útil en la solución de sistemas lineales, es la función `jacobian`, el jacobiano se define como: Si una matriz $f(x)$ es una función vectorial de un vector n-dimencional x, entonces

$$J = \frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Pero utilicemos un ejemplo para ver entender la ventaja de la función: Supongamos un ejercicio de estática el cual genera el siguiente set de ecuaciones:

$$F_{ac} \cdot \cos(30) - F_{ab} \cdot \cos(45) = 0$$
$$F_{ac} \cdot \sin(30) + F_{ab} \cdot \sin(45) + F_w = 0$$
$$F_e - F_{ac} \cdot \cos(30) = 0$$
$$F_d - F_{ac} \cdot \sin(30) = 0$$

Tenemos entonces un sistema de 4 ecuaciones con 4 incógnitas, y por lo tanto el sistema posee solución exacta, en este caso tomemos como incógnitas a Fac, Fab, Fe y Fd, y como valores de entrada a Fw.

El código en MATLAB queda entonces:

```
syms Fac Fab Fe Fd Fw t1 t2    % Define las variables simbolicas
G = [ Fac*cos(t1) - Fab*cos(t2) ; ...
   Fac*sin(t1) + Fab*sin(t2) - Fw ; ...
   Fe - Fac*cos(t1); ...
   Fd - Fac*sin(t1) ]      % Define la matriz de ecuaciones
In = [ Fac ; Fab ; Fe ; Fd  ]  % vector de incognitas
J = jacobian(G,In)     % Obtiene matriz de vbles dependientes
F = J*In - G       % Obtiene matriz de vbles independientes
Fs = inv(J)*F      % Soluciona el sistema
% Define valores numericos
t1 = 30*pi/180;   t2 = 45*pi/180;
Fw = 10;
eval(Fs)   % evalua la matriz para obtener una solucion
```

Obtener las ecuaciones Fs en variables que solucionan el sistema tiene la ventaja de generar múltiples soluciones sin necesidad de solucionar de nuevo todas las ecuaciones, por ejemplo si se desea obtener una gráfica de la tensión de todas las cuerdas cuando el ángulo $t2$ cambia desde 10º hasta 60º en intervalos de 1º.

Ahora bien suponga que se define un límite para la tensión soportada por alguna de las cuerdas y se desea obtener la carga máxima que puede soportar el sistema, entonces lo único que hay que hacer es modificar el vector de incógnitas, y listo, se obtiene la nueva solución.

Hay que advertir que se esta solucionando un sistema lineal exacto, esto implica que el ángulo no se puede generar como incógnita, para eso existen otro método, que se verá más adelante.

El toolbox también permite solucionar expresiones simbólicas:

```
syms a b c x
S = a*x^2 + b*x +c  % Ecuacion cuadratica
solve(S)            % genera la solucion de la cuadratica
```

Cuando no se especifica cuál es la variable a solucionar, el programa toma por defecto la variable 'x'. Pero la ecuación se puede resolver para otra variable:

```
b = solve(S,b)          % despeja b de la ecuacion S
```

También se puede obtener la solución de un conjunto de ecuaciones. Tomemos las 4 ecuaciones del ejemplo previo, pero ahora se van a definir de una forma diferente:

```
syms Fac Fab Fe Fd Fw t1 t2 % Define las variables simbolicas
equ = ' Fac*cos(t1) - Fab*cos(t2) , Fac*sin(t1) + Fab*sin(t2) - Fw , ...
     Fe -    Fac*cos(t1), Fd - Fac*sin(t1) '     % Define las ecuaciones
S = solve(equ,'Fac,Fab,Fe,Fd')      % Solucion del sistema
% Y para evaluar
t1 = 30*pi/180;   t2 = 45*pi/180;
```

```
Fw = 10;
Fab = eval(S.Fab)
Fac = eval(S.Fac)
Fd = eval(S.Fd)
Fe = eval(S.Fe)
```

El objeto S, contiene entonces la solución al sistema, que debe ser igual a la obtenida en el ejemplo anterior. Este método, tiene la ventaja de poder solucionar el sistema para cualquier set de variables, incluso los ángulos t1 y t2:

```
S = solve(equ,'Fac,Fab,t1,Fd')      % Solucion del sistema
```

5.5. Ejercicios

1. Genere una función que calcule el factor de concentrador de esfuerzo por geometría (K_t).
2. Genere una función que calcule todos los parámetros geométricos del diente de un engrane a partir del Número de dientes y módulo.
3. Genere una función para calcular el concentrador de esfuerzo de esfuerzo por fatiga.
4. Genere una función que muestre el modulo de elasticidad E, de un material especificado por la Tabla A-5.
5. Resolver por medio de los dos métodos descritos en la sección 5.4.1 (usando la función `jacobian` y `solve`) los ejercicios de la estática de Hibbelert (Séptima Edidición) dados por la tabla:

Grupo	**Ejercicio**
1	3-35
2	3-34
3	3-47
4	3-55
5	3-50
6	3-48
7	3-65
8	3-35
9	3-34
10	3-47

6. Obtenga la integral de las siguientes funciones:

$$1.\int_0^1 \frac{u\,du}{a+bu} \qquad 2.\int_0^1 \frac{u^2\,du}{a+bu} \qquad 3.\int_0^1 \frac{u\,du}{(a+bu)^2}$$

$$4.\int_0^1 \frac{u\,du}{(a+bu)^3} \qquad 5.\int_0^1 \frac{du}{u(a+bu)} \qquad 6.\int_0^1 \frac{du}{u^2(a+bu)}$$

$$7.\int_0^1 u\sqrt{a+bu}\,du \qquad 8.\int_0^1 u^2\sqrt{a+bu}\,du \qquad 9.\int_0^1 \frac{u\,du}{\sqrt{a+bu}}$$

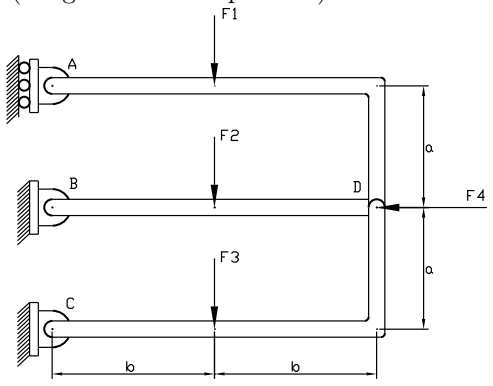
$$10.\int_0^1 \frac{du}{\sqrt{a^2+u^2}} \qquad 11.\int_0^1 \frac{du}{\sqrt{a^2-u^2}} \qquad 12.\int_0^1 \frac{du}{\sqrt{u^2-a^2}}$$

7. Calcule la derivada $\frac{\delta f}{\delta u}$ de las siguientes funciones:

   1. $f(u) = \frac{1}{b^2}\left[\frac{a}{2(a+bu)^2} - \frac{1}{a+bu}\right]$
   2. $f(u) = \frac{2}{15b^3}(3bu-2a)(a+bu)^{3/2}$
   3. $f(u) = \frac{2}{105b^3}(15b^2u^2-12abu+8a^2)(a+bu)^{3/2}$
   4. $f(u) = \frac{2}{3b^2}(bu-2a)\sqrt{a+bu}$
   5. $f(u) = \frac{2}{15b^3}(3b^2u^2-4abu+8a^2)\sqrt{a+bu}$
   6. $f(u) = \frac{u}{2}\sqrt{a^2-u^2} + \frac{a^2}{2}\sin^{-1}\left(\frac{u}{a}\right)$

5.6. Solución a problemas propuestos

1. La Figura 5.1 muestra una estructura que consta de tres elementos (AD, BD y CD). El sistema esta cargado con las fuerza F_1 , F_2 , F_3 y F_4. La figura 5.2 presenta cada elemento con sus reacciones (Diagrama de cuerpo libre).

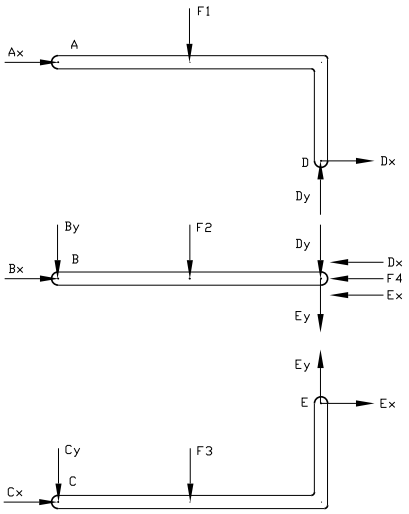


$$\begin{aligned}\sum F_x &= A_x + D_x = 0\\ \sum F_y &= D_y - F_1 = 0\\ \sum M_d &= a\cdot A_x - b\cdot F_1 = 0\end{aligned}$$

$$\begin{aligned}\sum F_x &= B_x - D_x - E_x - F_4 = 0\\ \sum F_y &= B_y - F_2 - E_y - D_y = 0\\ \sum M_d &= 2b\cdot B_y - F_2\cdot b = 0\end{aligned}$$

$$\begin{aligned}\sum F_x &= C_x + E_x = 0\\ \sum F_y &= C_y - F_3 - E_y = 0\\ \sum M_d &= b\cdot F_3 + a\cdot E_x - 2b\cdot E_y = 0\end{aligned}$$

Figura 5.1: Estructura Principal y las ecuaciones del sistema



Si se conocen todas la fuerzas externas y las distancias a y b.

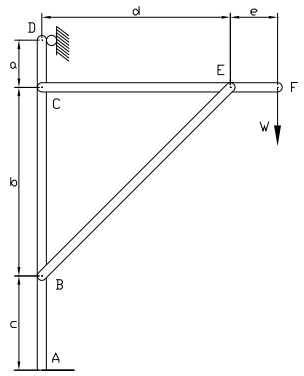
- Determine las reacciones en cada apoyo por medio del método de `jacobian` y `solve`.
- Al problema asigne los siguientes valores:

a	b	F_1	F_2	F_3	F_4
0.3	0.4	120	180	240	300

  Escriba la solución numérica.

Figura 5.2: Ensamble con las reacciones

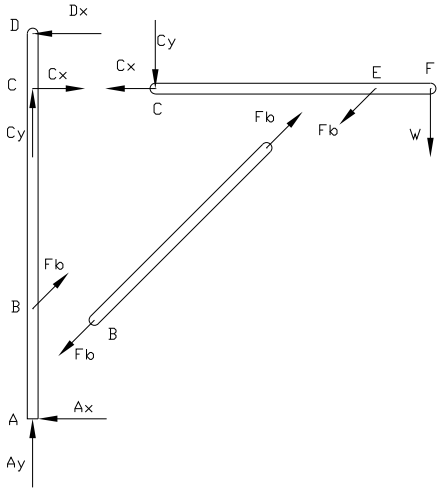
2. La figura 5.3 muestra una estructura que consta de tres elementos (AD, CF y BE), y las ecuaciones que rigen la estructura. El sistema esta cargado con la fuerza W. La figura 5.4 presenta cada elemento con sus reacciones, y la formulación del problema.



$$\begin{aligned}\sum F_x &= -D_x + C_x - A_x - F_b \cdot \cos(\theta) = 0\\ \sum F_y &= C_y - F_b \cdot \text{sen}(\theta) + A_y = 0\\ \sum M_B &= -A_x \cdot c - C_x \cdot b + D_x(a+b) = 0\end{aligned}$$

$$\begin{aligned}\sum F_x &= -C_x + F_b \cdot \cos(\theta) = 0\\ \sum F_y &= -C_y + F_b \cdot \sin(\theta) - W = 0\\ \sum M_d &= C_y \cdot d - W \cdot e = 0\end{aligned}$$

Figura 5.3: Estructura Principal y las ecuaciones del sistema



Si se conoce la fuerza D_x y todas las distancias (incluyendo el ángulo θ).

- Determine las reacciones en cada apoyo y el peso W por medio del método de `jacobian` y `solve`.
- Al problema asigne los siguientes valores:

a	b	c	d	e	D_x	θ
0.4	1.6	0.8	b	a	700	45°

Escriba la solución numérica.

Figura 5.4: Ensamble con las reacciones

3. Resuelva los siguientes problemas simbólicos y escriba en la parte posterior de la hoja la solución más simple.

$$\int u\sqrt{a+bu}\; du$$

$$\int \frac{\sqrt{u^2-a^2}}{u}\; du$$

$$\frac{d}{du}\left(\frac{2}{3b^2}(bu-2a)\sqrt{a+bu}\right)$$

$$\frac{d}{du}\left(\frac{2}{15b^3}(3b^2u^2-4abu+8a^2)\sqrt{a+bu}\right)$$

4. Realice un función que dado el radio de un circulo, devuelva el area e I_x. Se tendrá en cuenta el nombre de la función y la ayuda de la misma.

$$Area = \pi r^2, \quad I_x = \tfrac{1}{4}\pi r^4$$

5. Realice un función que dados la base y la altura de un rectángulo, ella devuelva el area, I_x, I_y e I_{xy}. Se tendrá en cuenta el nombre de la función y la ayuda de la misma.

$$Area = bh$$

$$I_x = \tfrac{1}{12}bh^3, \qquad I_y = \tfrac{1}{4}hb^3, \qquad I_{xy} = \tfrac{1}{8}b^2h^2$$

Solución

1. El primer paso para solucionar el problema por cualquiera de los dos métodos es determinar las incógnitas o variables desconocidas del sistema, en este caso son: A_x, B_x, B_y, C_x, C_y, D_x, D_y, E_x y E_y. Lo que implica que se tienen 9 incógnitas, y como se poseen 9 ecuaciones, entonces el sistema tiene solución.

```
 % Metodo de Jacobiano
 syms F1 F2 F3  F4 Ax Bx By Dx Dy Cx Cy Ex Ey a b
% Define Matriz de Ecuaciones
G = [Ax+Dx ; Dy-F1 ; a*Ax-b*F1 ; ...
     Bx-Dx-Ex-F4; By-F2-Ey-Dy ; 2*b*By - F2*b ; ...
     Cx+Ex ; Cy-F3+Ey ; b*F3+a*Ex-2*b*Ey];
% Vector Columna de incognitas
In = [Ax ; Bx ; By ; Cx ; Cy ; Dx ; Dy ; Ex ; Ey];
J = jacobian(G,In); % Matriz de Vbles dependientes
F = J*In - G;   % Matriz de Vbles Independientes
Fs1 = inv(J)*F;  % Solucion del sistema
Fs1 = simple(Fs1); % Simplifica el sistema
```

```
 % Metodo Solve
 syms F1 F2 F3  F4 Ax Bx By Dx Dy Cx Cy Ex Ey a b
 % Define Matriz de Ecuaciones
 equ = 'Ax+Dx , Dy-F1 , a*Ax-b*F1 , Bx-Dx-Ex-F4 , By-F2-Ey-Dy ,
        2*b*By - F2*b , Cx+Ex , Cy-F3+Ey , b*F3+a*Ex-2*b*Ey';
 S = solve(equ,'Ax,Bx,By,Cx,Cy,Dx,Dy,Ex,Ey');
  Fs2(1,1)=S.Ax; Fs2(2,1)=S.Bx; Fs2(3,1)=S.By;
  Fs2(4,1)=S.Cx; Fs2(5,1)=S.Cy; Fs2(6,1)=S.Dx;
  Fs2(7,1)=S.Dy; Fs2(8,1)=S.Ex; Fs2(9,1)=S.Ey;
```

En este momento, la matriz $Fs1$ y $Fs2$ son matrices de ecuaciones que contienen la solución del problema. Si se comparan $Fs1$ y $Fs2$ se puede apreciar que las matrices son iguales:

```
Fs1 =

[                   1/a*b*F1]
[ (-3*b*F1+F4*a-F2*b-b*F3)/a]
[                     1/2*F2]
[           b*(2*F1+F2+F3)/a]
[               F1+1/2*F2+F3]
[                  -1/a*b*F1]
[                        F1]
[          -b*(2*F1+F2+F3)/a]
[                 -F1-1/2*F2]
```

Donde la solución de cada variable se toma en el mismo orden de la matriz $Fs1$ o $Fs2$. Para aplicar valores numéricos a la solución, se definen entonces los valores de variables conocidas:

```
a = 0.3; b = 0.4; F1 = 120;
F2 = 180; F3 = 240; F4 = 300;
sol = eval(Fs1)

sol =

  160.0000
 -740.0000
   90.0000
  880.0000
  450.0000
 -160.0000
  120.0000
 -880.0000
 -210.0000
```

Lo cual implica que la solución del problema es:

$$\begin{array}{lll} A_x = 160 & B_x = -740 & B_y = 90 \\ C_x = 880 & C_y = 450 & D_x = -160 \\ D_y = 120 & E_x = -880 & E_y = -210 \end{array}$$

2. El primer paso para solucionar el problema por cualquiera de los dos métodos es determinar las incógnitas o variables desconocidas del sistema, en este caso son: A_x, A_y, F_b, C_x, C_y, W. Lo que implica que se tienen 6 incógnitas, y como se poseen 6 ecuaciones, entonces el sistema tiene solución.

```
 % Metodo de Jacobiano
syms Ax Ay Fb Cx Cy Dx a b c d e W tetha
% Define Matriz de Ecuaciones
G = [-Dx+Cx-Ax-Fb*cos(tetha) ;...
      Cy-Fb*sin(tetha)+Ay; ...
     -Ax*c-Cx*b+Dx*(a+b);...
     -Cx+Fb*cos(tetha); ...
     -Cy+Fb*sin(tetha)-W ;...
     Cy*d-W*e];
% Vector Columna de incognitas
In = [Ax ; Ay ; Fb ; Cx ; Cy ; W];
J = jacobian(G,In); % Matriz de Vbles dependientes
F = J*In - G;   % Matriz de Vbles Independientes
Fs1 = inv(J)*F;  % Solucion del sistema
Fs1 = simple(Fs1); % Simplifica el sistema
```

```
 % Metodo Solve
 syms F1 F2 F3  F4 Ax Bx By Dx Dy Cx Cy Ex Ey a b
 % Define Matriz de Ecuaciones
 equ  = '-Dx+Cx-Ax-Fb*cos(tetha) , Cy-Fb*sin(tetha)+Ay ,
        -Ax*c-Cx*b+Dx*(a+b) , -Cx+Fb*cos(tetha) ,
        -Cy+Fb*sin(tetha)-W , Cy*d-W*e';
 S = solve(equ,'Ax,Ay,Fb,Cx,Cy,W');
  Fs2(1,1)=S.Ax; Fs2(2,1)=S.Ay; Fs2(3,1)=S.Fb;
  Fs2(4,1)=S.Cx; Fs2(5,1)=S.Cy; Fs2(6,1)=S.W;
```

En este momento, la matriz $Fs1$ y $Fs2$ son matrices de ecuaciones que contienen la solución del problema. Si se comparan $Fs1$ y $Fs2$ se puede apreciar que las matrices son iguales:

```
 Fs2 =

 [                                        -Dx]
 [ d*Dx*(c+a+b)*sin(tetha)/cos(tetha)/b/(e+d)]
 [                     Dx*(c+a+b)/cos(tetha)/b]
 [                                Dx*(c+a+b)/b]
 [ Dx*(c+a+b)*sin(tetha)*e/cos(tetha)/b/(e+d)]
 [ d*Dx*(c+a+b)*sin(tetha)/cos(tetha)/b/(e+d)]
```

Donde la solución de cada variable se toma en el mismo orden de la matriz $Fs1$ o $Fs2$. Para aplicar valores numéricos a la solución, se definen entonces los valores de variables conocidas:

```
a = 0.4; b = 1.6; c = 0.8; d=b; e=a;
Dx=700; tetha=45*pi/180;
sol = eval(Fs2)

sol =

 1.0e+003 *

  -0.7000
   0.9800
   1.7324
   1.2250
   0.2450
   0.9800
```

Lo cual implica que la solución del problema es:

$$A_x = -700 \qquad A_y = 980 \qquad F_b = 1732{,}4$$
$$C_x = 1225 \qquad C_y = 245 \qquad W = 980$$

3. La solución de los problemas es la siguiente

```
% La primera integral
syms u a b
f=u*sqrt(a+b*u); % Funcion
g=int(f,u); %Derivada respecto a u
g2=simple(g); % Simplificacion
pretty(g)
pretty(g2)
```

La variable f contiene la función a integrar, g es la integral de f y $g2$ es la simplificación de g. Como se pide la forma más simple (que es $g2$), entonces, la solución al problema es

$$g2 = -2/15\,\frac{(a+bu)^{3/2}\,(2\,a-3\,bu)}{b^2}$$

```
% La segunda integral
 syms u a b
 f=sqrt(u^2-a^2)/u; % Funcion
 g=int(f,u); %Integral respecto a u
 g2=simple(g); % Simplificacion
 pretty(g)
 pretty(g2)
```

La variable f contiene la función a integrar, g es la integral de f y $g2$ es la simplificación de g. Como se pide la forma más simple (que es g), entonces, la solución al problema es

$$g = \sqrt{u^2-a^2} - a^2\,arctanh\left(\frac{a^2}{\sqrt{-a^2}\sqrt{u^2-a^2}}\right)\frac{1}{\sqrt{-a^2}}$$

```
% La primera derivada
syms a b u
h = 2/(3*b^2)*(b*u-2*a)*sqrt(a+b*u);
z = diff(h,u);
z2 = simple(z);
pretty(z)
pretty(z2)
```

La variable h contiene la función a derivar, z es la derivada de h y $z2$ es la simplificación de z. Como se pide la forma más simple (que es $z2$), entonces, la solución al problema es

$$z2 = \frac{u}{\sqrt{a+bu}}$$

```
% La segunda derivada
syms a b u
h = (2/(15*b^3))*(3*(b*u)^2-4*a*b*u+8*a^2)
                  *sqrt(a+b*u);
z = diff(h,u);
z2 = simple(z);
pretty(z)
pretty(z2)
```

La variable h contiene la función a derivar, z es la derivada de h y $z2$ es la simplificación de z. Como se pide la forma más simple (que es $z2$), entonces, la solución al problema es

$$z2 = \frac{u^2}{\sqrt{a+bu}}$$

4. La función entonces posee como variable de entrada el radio del circulo (r), y como salida la Inercia I_x y el área del mismo A. El nombre de la función, para que sea acorde con su objetivo se llama *circ*. Entonces la interfase de la función es de la forma:

```
function [Ix,A] = circ(r)
```

Después sigue la ayuda de la función, la cual debe indicar la forma de utilizar la función:

```
% [Ix,A] = circ(r)  Calcula de un circulo de radio r,
% la inercia y el Area
```

Para por último pasar a los cálculo u operaciones:

```
A = pi * r^2;
Ix = 1/4*pi*r^4;
```

La función completa es de la forma:

```
function [Ix,A] = circ(r)
% [Ix,A] = circ(r)  Calcula de un circulo de radio r,
% la inercia y el Area

A = pi * r^2;
Ix = 1/4*pi*r^4;
```

Recuerde que el nombre del archivo debe poseer el mismo nombre de la función. Ahora bien, para probar la función, nos dirigimos al prompt de MATLAB y aplicamos la siguiente instrucción:

```
>> [InerciaX,Area]=circ(3)

InerciaX =
   63.6173

Area =
   28.2743
```

5. La función entonces posee como variables de entrada la base (b) y la altura (h) del rectangulo, y como salida la Inercia I_x , I_y, I_xy y el área del mismo A. El nombre de la función, para que sea acorde con su objetivo se llama *rect*. Entonces la interfase de la función es de la forma:

```
function [Ix,Iy,Ixy,A] = rect(b,h)
```

Después sigue la ayuda de la función, la cual debe indicar la forma de utilizar la función:

```
% function [Ix,Iy,Ixy,A] = rect(b,h) Calcula de un rectangulo de base b y
% altura h, las inercias Ix, Iy, Ixy y el area A.
```

Y por último pasar a los cálculos u operaciones:

```
A = h*b;
Ix = 1/12*b*h^3;
Iy = 1/4*h*b^3;
Ixy = 1/8*(b*h)^2;
```

La función completa es de la forma:

```
function [Ix,Iy,Ixy,A] = rect(b,h)
% [Ix,Iy,Ixy,A] = rect(b,h) Calcula de un rectangulo de base b y
% altura h, las inercias Ix, Iy, Ixy y el area A.

A = h*b;
Ix = 1/12*b*h^3;
Iy = 1/4*h*b^3;
Ixy = 1/8*(b*h)^2;
```

Recuerde que el nombre del archivo debe poseer el mismo nombre de la función. Ahora bien, para probar la función, nos dirigimos al prompt de MATLAB y aplicamos la siguiente instrucción:

```
>> [InerX,InerY,InerXY,Area] = rect(3,8)

InerX =                 InerXY =
   128                       72

InerY =                 Area =
    54                      24
```

Capítulo 6

Gráficas 2D

MATLAB posee una gran selección de funciones muy fáciles de usar para generar gráficas 2D y 3D. Las funciones gráficas se pueden agrupar en tres categorías: *administración de figuras*, *generación de curvas y superficies* y *anotaciones y características gráficas*. Las funciones gráficas de MATLAB se pueden resumir en la tabla 6.1.

Tabla 6.1: Resumen Funciones Graficas

Administración	Generación	Anotaciones y Características
`figure`	**2-D**	`xlabel`
`subplot`	`plot`	`ylabel`
`zoom`	`polar`	`text`
`hold`	`fill`	`title`
	`plotyy`	`legend`
3-D	`semilogx` , `semilogy`	`box`
`view`	`loglog`	`set`
`rotate3d`	`stairs`	`grid`
	`stem`	`axis`, `exis equal`, `axis off`
	`bar`	`clabel`
	3-D	**3-D**
	`plot3`	`text3`
	`surf`, `surfc`	`zlabel`
	`mesh`, `meshc`, `meshz`	`colorbar`
	`contour`, `contour3`,	`colormap`
	`contourf`	`shading`
	`waterfall`	
	`cilinder`	

Cuando se generan entidades gráficas, es necesario garantizar que en cada figura se encuentra la solución del problema de forma clara y específica, mostrando solo la información importante; la figura debe estar totalmente documentada con el nombre de cada eje, y el titulo de la figura, además, cada curva debe ser identificada fácilmente (si hay más de una), y los valores numéricos importantes mostrados. Sin embargo algunos mecanismos de para mejora las figuras, como el color, los tipos de líneas, símbolos y texto, deben se utilizados sin convertirse en distractores.

Por lo general, el set instrucciones para generar ua figura, empieza por algunas instrucciones de administración, para pasar luego a las instrucciones de generación de la figura, y por último, para mejorar la presentación de la figura, se utilizan algunas instrucciones de anotación y características. Dependiendo las

necesidades, uno u otras instrucción serán necesarias y otras no.

6.1. Funciones de Administración

Cada gráfica es creada en una ventana de dibujo, esta ventana es creada por MATLAB automáticamente cuando algunas de las funciones gráficas es ejecutada.Cuando un programa, sea una función o un script, une más de una función de *generación*, MATLAB crea una nueva ventana de dibujo. Sin embargo, una figura creada previamente es cerrada antes de crear la nueva ventana de dibujo. Para retener cada nueva gráfica en su propia ventana de figura, se debe usar la instrucción:

```
figure(n)
```

donde `n` es un entero. Si el argumento `n` de `figure` es omitido, MATLAB automáticamente genera la siguiente ventana de dibujo.

También se pueden crear varias gráficas en la misma ventana de dibujo con la función

```
subplot(i,j,k)
```

Los dos primeros argumentos dividen la ventana de dibujo en sectores (filas y columnas), y el tercero indica en qué sector va a ser aplicada la gráfica. El valor de 1 para este argumento indica la esquina superior izquierda. La suma de las filas y columnas indica el valor más alto que puede tomar este índice y corresponde a la posición inferior derecha de la ventana de dibujo. A medida que este número se va incrementando, indica la posición de izquierda a derecha comenzando en la fila superior. Cualquier función de generación o anotación que aparezca en el programa después la función `figure` y/o `subplot`, se aplica solamente a la figura indicada por el índice de `figure` o por el tercer índice de la función `subplot`. En cada una de las secciones se pueden utilizar todas las funciones de generación de gráficas 2D y 3D.(La figura 6.1 muestra algunos ejemplos de como usar la función `figure` y `subplot`). Si solo se va usar una figura, la función `figure` puede ser omitida.

Debido a que cada gráfica es creada en una nueva figura de dibujo, para dibujar una o más curvas, superficies o líneas (o combinaciones de éstas) sobre una gráfica dada, se debe usar la orden

```
hold on
```

la cual mantiene (*hold*) la ventana (o `subplot`) activo actualmente. Todas las figuras creadas pueden ser copiadas y pegadas en cualquier editor de texto (como Word), para esto seleccione la opción *Copy Figure* del menú *Edit* de cada una de las ventanas de dibujo.

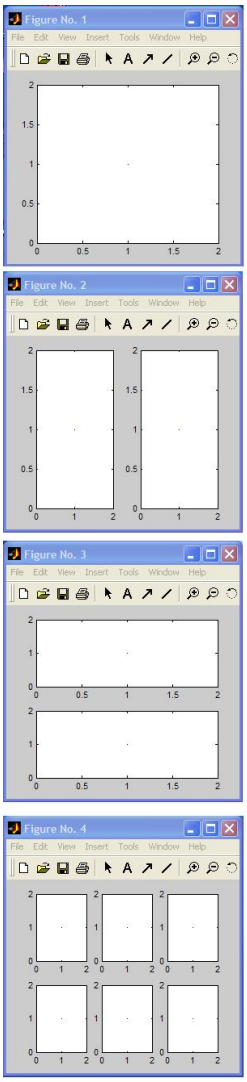
6.2. Comandos básicos para la generación de gráficas 2D

El principal comando de generación de gráficas es

```
plot(u1,v1,c1,u2,v2,c2,...)
```

donde u_j y v_j son las coordenadas x y y respectivamente, de un punto o una serie de puntos. Ellos son un par de números, vectores de la misma longitud, matrices del mismo orden, o expresiones que al ser evaluadas resulten en una de estas tres opciones. El termino c_j es una cadena de caracteres: Un caracter especifica el color de línea o punto, otro caracter especifica el tipo del punto si van a ser graficados puntos y el último caracter puede ser usado para especificar las características de la línea. Los diferentes tipos de líneas y puntos, así como los colores que se pueden usar se muestran en la tabla 6.2.

Cuando una serie de puntos van a ser dibujados, uno de los caracteres de c_j, puede ser por ejemplo un 's', para dibujar un cuadrados, o un asterisco (`*`) para dibujar asteriscos. Cuando no se desean mostrar los puntos, estos pueden ser conectados por líneas, utilizando los tipos de linea de la tabla. Cuando líneas y puntos van a ser graficadas al mismo tiempo y del mismo color, c_j contiene ambas descripciones. El orden



Scripts o Funciones

```
figure(1)
```
expresiones de dibujo
⋮

```
figure(2)
subplot(1,2,1)
```
expresiones de dibujo
⋮
```
subplot(1,2,2)
```
expresiones de dibujo
⋮

```
figure(3)
subplot(2,1,1)
```
expresiones de dibujo
⋮
```
subplot(2,1,2)
```
expresiones de dibujo
⋮

```
figure(4)
subplot(2,3,1)
```
expresiones de dibujo
⋮
```
subplot(2,3,2)
```
expresiones de dibujo
⋮
```
subplot(2,3,3)
```
expresiones de dibujo
⋮
```
subplot(2,3,6)
```
expresiones de dibujo
⋮
```
subplot(2,3,5)
```
expresiones de dibujo
⋮
```
subplot(2,3,4)
```
expresiones de dibujo
⋮

Figura 6.1: Ejemplo del uso de varias combinaciones de `figure` y `subplot`.

Tabla 6.2: Características de Líneas y Puntos

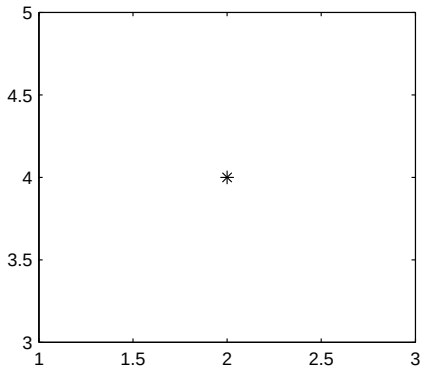
Tipo de linea		Color de Linea		Tipo de punto	
Símbolo	Descripción	Símbolo	Descripción	Símbolo	Descripción
`-`	Solida	`r`	Rojo	`+`	Signo más
`--`	Dashed	`g`	Verde	`o`	Circulo
`:`	Punteada	`b`	Azul	`*`	Asterisco
`_.`	Dashed-punto	`c`	Cian	`.`	Punto
		`m`	Magenta	`x`	Cruz
		`y`	Amarillo	`s`	Cuadrado
		`k`	Negro	`d`	Diamante
		`w`	Blanco	`>`	Trián. derecha
				`<`	Trián. izquierda
				`p`	Pentagono
				`h`	Hexagono

de los caracteres dentro de la camillas no es importante. Si c_j es omitido, entonces el programa utiliza los valores por defecto. Si se grafican varias líneas, el programa le asigna a cada una un color diferente también por defecto.

Ahora se describen algunas instrucciones para dibujar puntos, líneas, círculos, etc.

6.2.1. Puntos

Para poner un asterisco rojo en el punto $(2, 4)$, se utiliza como lo muestra la figura 6.2.




```
plot(2,4,'r*')
```

Figura 6.2: Dibujar un punto

6.2.2. Líneas

Para dibujar una linea que salga del punto $(0, 0)$ al $(1, 2)$ usando el tipo de línea (solida) y color (azul) por defecto, la instrucción es

```
plot([0,1],[0,2])
```

Note que los elementos del primer vector $[0, 1]$ representan los valores de la coordenadas en el eje-x y los elementos del segundo vector $[0, 2]$ son las coordenadas en el eje-y de los puntos de las líneas a graficar.

Suponga ahora que se desea dibujar un grupo de n líneas no conectadas cuyos puntos están dados por los pares (x_{1n}, y_{1n}) y (x_{2n}, y_{2n}). Entonces se deben generar cuatro vectores:

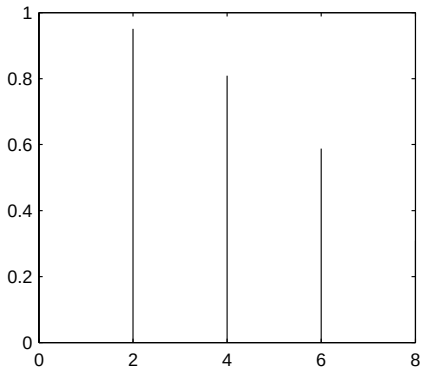
$$x_j = [x_{j1} \;\; x_{j2} \;\; \ldots \;\; x_{jn}]$$
$$y_j = [y_{j1} \;\; y_{j2} \;\; \ldots \;\; y_{jn}], \;\; j = 1, 2$$

Entonces la instrucción `plot` es de la forma:

```
plot([x1;x2];[y1;y2])
```

donde `[x1,x2]` y `[y1,y2]` son vectores de (2 × n) cada uno.

Para entender esta expresión, dibujar 5 líneas verticales desde $y = 0$ a $y = \cos(\pi x/20)$ cuando $x = 2, 4, 6, 8, 10$ como lo muestra la figura 6.3, y junto a ella el código necesario para generarla.




```
x1 = 0:2:8;
x2 = x1;
y1 = zeros(1,length(x1));
y2 = cos(pi*x1/20);
plot([x1;x2],[y1;y2],'k')
```

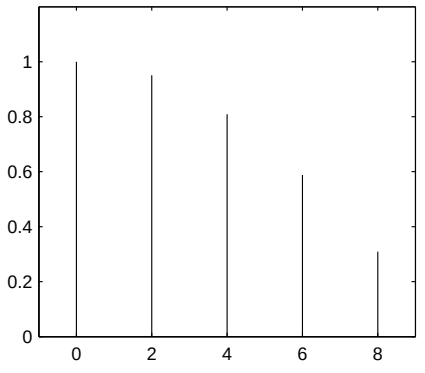
Figura 6.3: Líneas verticales paralelas

El color especificado (negro) se aplica a todas la líneas. La función `zeros` se utiliza para generar una matriz llena de ceros con un número de filas y columnas específicas. Desafortunadamente, debido a que el programa selecciona automáticamente los ejes, la primera y la última línea coinciden con el cuadro que encierra la figura. Si se ajustan los ejes, entonces será posible ver todas las líneas.

Este ajuste se puede realizar con:

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

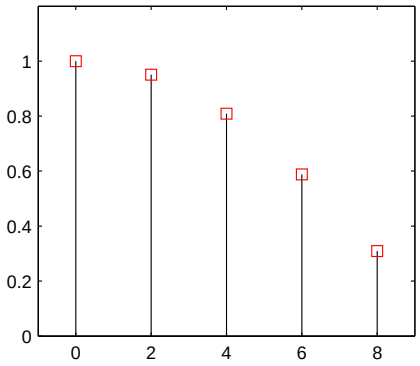
donde x_{min} , x_{min} , y_{max} y y_{max} son los valores máximos y mínimos de los ejes x y y, respectivamente. Entonces la figura y el script se muestran en la figura 6.4.




```
x1 = 0:2:8;
x2 = x1;
y1 = zeros(1,length(x1));
y2 = cos(pi*x1/20);
plot([x1;x2],[y1;y2],'k')
axis([-1 9 0 1.2])
```

Figura 6.4: Cambio de ejes de la figura

Ahora se desea poner un cuadrado rojo al final de cada una de líneas verticales, entonces es necesario adicionar otra tripleta de instrucciones al comando `plot`. Como lo muestra la figura 6.5.




```
x1 = 0:2:8;
x2 = x1;
y1 = zeros(1,length(x1));
y2 = cos(pi*x1/20);
plot([x1;x2],[y1;y2],'k',x1,y2,'sr')
axis([-1 9 0 1.2])
```

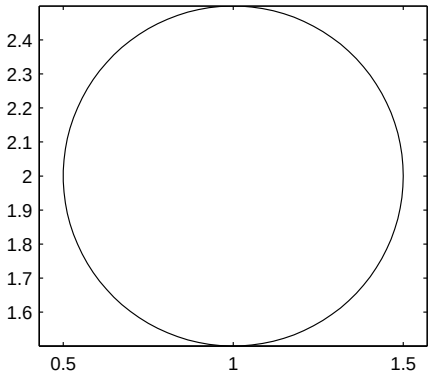
Figura 6.5: Cuadrados en la punta

6.2.3. Círculos

Para dibujar círculos de radio r y centro en el punto (a, b) en el sistema de coordenadas cartesianas, es necesario primero hacer la transformación de coordenadas radiales a coordenadas cartesianas:

$$x = a + r\ \cos(\theta)$$
$$y = b + r\ \text{sen}(\theta)$$

donde $0 \leq \theta \leq \theta_1 \leq 2\pi$. Si $\theta < 2\pi$ entonces se dibuja el arco de un circulo. Se asume ahora que $\theta_1 = 2\pi$, $a = 1$, $b = 2$ y $r = 0{,}5$. El circulo y el script se muestran en la figura 6.6.




```
theta = linspace(0,2*pi);
a = 1; b = 2; r = 0.5;
x = a + r*cos(theta);
y = b + r*sin(theta);
plot(x,y)
axis equal
```

Figura 6.6: Circulo

La función `axis equal` obliga a los ejes a ser iguales, evitando que el circulo luzca como una elipse.

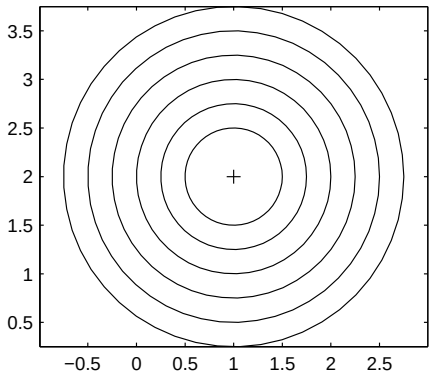
Ahora se desean dibujar 6 círculos concéntricos, con radios que comienzan en 0.5 hasta 1.75 y el centro debe estar indicado por un signo más.

6.2.4. Multiples funciones graficadas en una misma figura

Considere las tres funciones:

$$f_1(x) = 0{,}1x^2$$
$$f_2(y) = \cos^2 y$$
$$f_3(z) = e^{-0{,}3z}$$

donde $0 \leq x = y = z \leq 3{,}5$. Lo primero es generar los vectores con los valores de cada función evaluada en el rango especificado:

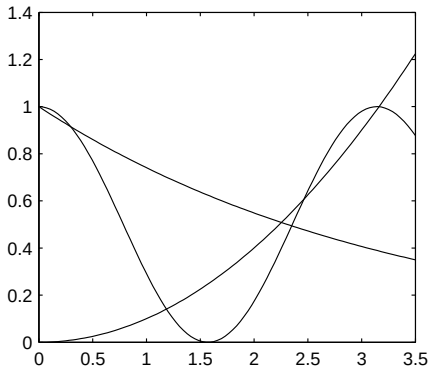



```
theta = linspace(0,2*pi,50);
a = 1; b = 2;
r = 0.5 : 0.25 : 1.75;
x = a + r'*cos(theta);
y = b + r'*sin(theta);
plot(x',y','k',a,b,'k+')
axis equal
```

Figura 6.7: Círculos Concéntricos

```
x = linspace(0,3.5);
g1 = 0.1*x.^2;
g2 = cos(x).^2;
g3 = exp(-0.3*x);
```

Ahora se puede generar la figura con las curvas necesarias, esta se puede realizar de tres formas como muestra la figura 6.8.




```
% Primera forma:
plot(x,[g1 ; g2 ; g3],'k')

% Segunda Forma
plot(x,g1,'r',x,g2,'c',x,g3,'b')

% Tercera forma
plot(x,g1,'r')
hold on
plot(x,g2,'c')
plot(x,g3,'b')
```

Figura 6.8: Varias curvas en la misma gráfica

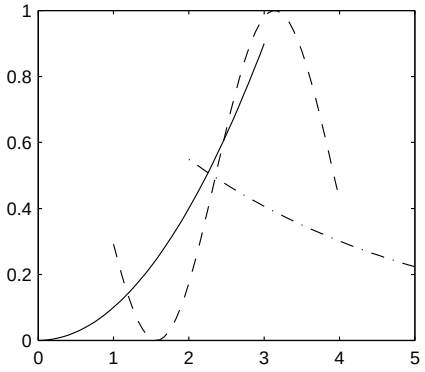
Por otro lado, si el rango de la variable independiente para cada una de estas funciones es diferente, solo el segundo y tercer script pueden ser usados. Por ejemplo, si $0 \leq x \leq 3$, $1 \leq y \leq 4$, y $2 \leq z \leq 5$, entonces se pueden generar primero los vectores con las instrucciones:

```
x = linspace(0,3,45);
y = linspace(1,4,55);
z = linspace(2,5,65);
g1 = 0.1*x.^2;
g2 = cos(y).^2;
g3 = exp(-0.3*z);
```

Y para generar la gráfica hay dos opciones como lo muestra la figura 6.9.

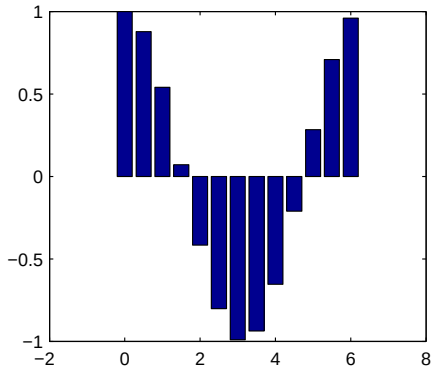
Hay que notar que la curva para cada función fue generada con diferente tipo de línea, color línea y diferentes número de datos.

Además de `plot` se pueden utilizar otras funciones para generar gráficas dependiendo de la necesidad como lo muestran las figuras 6.10, 6.11 y 6.12.



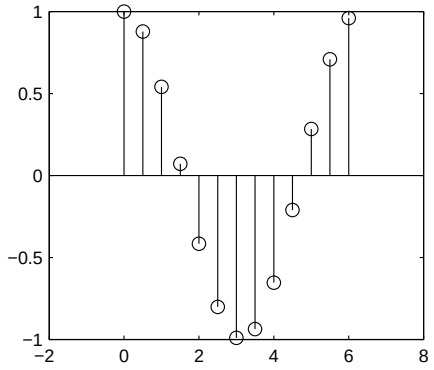
```
% Primera forma:
plot(x,g1,'r',y,g2,'c--',z,g3,'b-.')

% Segunda Forma
plot(x,g1,'r')
hold on
plot(y,g2,'c--')
plot(z,g3,'b-.')
```

Figura 6.9: Con diferente rango



```
x = 0 : 0.5 : 6;
y = cos(x);
bar(x,y)
```

Figura 6.10: Comando bar



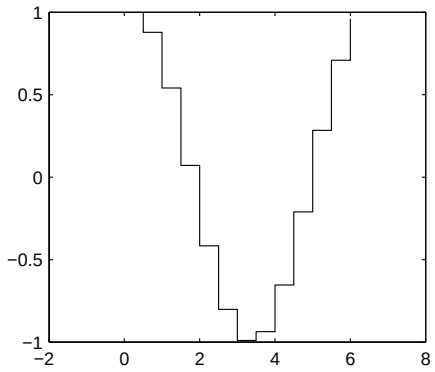
```
x = 0 : 0.5 : 6;
y = cos(x);
stem(x,y)
```

Figura 6.11: Comando stem

6.3. Anotaciones y Gráficas y mejoramiento visual

MATLAB posee una gran cantidad de funciones para mejorar la presentación de sus figuras. En esta sección se ven algunas funciones y ejemplos para obtener:

- Etiquetas para los ejes, titulo de la figura, leyendas, y texto dentro de la gráfica.
- Letras Griegas, símbolos matemáticos, sub-índice y super-índices.




```
x = 0 : 0.5 : 6;
y = cos(x);
stairs(x,y)
```

Figura 6.12: Comando stairs

6.3.1. Etiquetas para los ejes, titulo, leyenda y texto

Las funciones utilizadas para etiquetar el eje-x y eje-y y para asignar título a la gráfica son respectivamente:

```
xlabel(s)
ylabel(s)
title(s)
```

donde `s` es un string (o cadenas de caracteres entre comillas sencillas). La función que pone texto en cualquier parte de de la ventana de dibujo es:

```
text(x,y,s)
```

donde x y y son las coordenadas donde el texto dado por un string s, va a ser ubicado.

Dado una gráfica, etiquetas, título, y algunas relaciones entre la intersección de dos curvas, $cos(x)$ y $1/cosh(x)$, sobre el rango $0 \leq x \leq 6$. En este rango las figuras se cruzan en el punto $x = 4{,}73$. Se puede además dibujar una línea vertical a través del punto de intersección, y mostrar el valor de x cerca a la intersección.

```
x=linspace(0,6);
y1=cos(x);
y2=1./cosh(x);
plot(x,y1,'--',x,y2,'-',[4.73 4.73],[-1 1])
title('Visualizacion de la interseccion de dos curvas')
ylabel('Valor de la funcion')
xlabel('x')
text(4.8,-0.1,'x=4.73')
text(2.1,0.3,'1/cosh(x)')
text(1.2,-0.4,'cos(x)')
```

Este código genera la solución del problema al producir la figura 6.13. Las coordenadas para la posición de textos sobre la figura, son seleccionados después de aplicar la función `plot`, esto es, solo después de la primeras líneas que generan la figura. Luego, la función `text` es incluida.

Hay otra forma de identificar las curvas en la figura 6.13, esto es con la función `legend`, la cual posee la siguiente estructura:

```
legend(s1,s2,...,sn,p)
```

donde $s1$, $s2$, etc., son textos (*string* entre comillas) alfanuméricos que identifican cada una de la líneas que aparecen de la misma forma y color en el cuadro de leyenda en el mismo orden en que fueron generadas. `p`

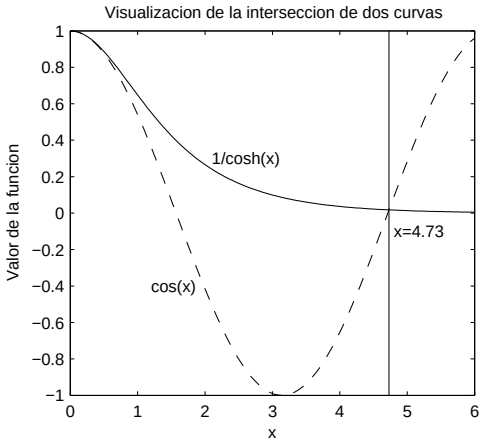


Figura 6.13: Comandos para generar anotaciones en la figura

Tabla 6.3: Opciones y Posiciones del Legend

Valor de p	Posición en la ventana
0	Posición automática (Menor conflicto con las curvas)
1	Esquina superior derecha (Por defecto)
2	Esquina superior izquierda
3	Esquina inferior izquierda
4	Esquina inferior derecha
-1	A la izquierda de la figura

es un número que indica la posición del cuadro de leyenda en la ventada de dibujo, y este puede tomar los valores que muestra la tabla 6.3.

La función `legend` debe ser aplicada después de generar la figura, es decir, después de ejecutar la función `plot`.

Para ilustrar el uso de la función `legend` y comparar el resultado con la figura 6.13, omitiendo los comandos `text` se realiza la figura 6.14 con el siguiente código:

```
x=linspace(0,6);
y1=cos(x);
y2=1./cosh(x);
plot(x,y1,'--',x,y2,'-',[4.73 4.73],[-1 1])
title('Visualizacion de la interseccion de dos curvas')
ylabel('Valor de la funcion')
xlabel('x')
text(4.8,-0.1,'x=4.73')
legend('cos(x)','1/cosh(x)',3)
```

La caja alrededor de la leyenda puede ser mostrada o ocultada con las funciones

```
legend('boxon') % Mostrar la caja
legend('boxoff') % Ocultar la caja
```

respectivamente.

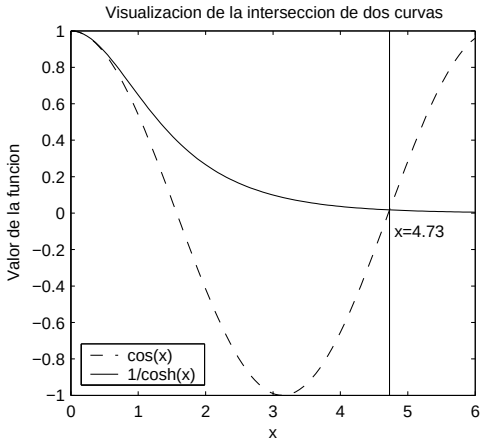


Figura 6.14: Uso de la función `legend`

6.3.2. Letras Griegas, Símbolos Matemáticos, Sub- y Super-índices

MATLAB también posee la capacidad de incluir en la figura letras griegas tanto en mayúscula como en minúscula, símbolos matemáticos, sub- y super-índices. Estas pueden ser utilizadas con la funciones `title`, `xlabel`, `ylabel`, `text` y `legend`. Ninguna de estas herramientas es aplicable al *command window* (prompt)– esto es, usando el comando `disp`.

Los sub-índices son creados con *underline* (línea pegada al suelo: `_`), y los super-índices se generan con el operador exponencial (`^`). La creación de las letras Griegas se realiza con el nombre de la letra precedido por *backslash* (\), como se muestra en la tabla 6.4.

Tabla 6.4: Caracteres Especiales de Anotación

Minúsculas				Mayúsculas	
Símbolo	Sintaxis	Símbolo	Sintaxis	Símbolo	Sintaxis
α	`\alpha`	ν	`\nu`	Γ	`\Gamma`
β	`\beta`	ξ	`\xi`	Δ	`\Delta`
γ	`\gamma`	o	`o`	Θ	`\Theta`
δ	`\delta`	π	`\pi`	Λ	`\Lambda`
ϵ	`\epsilon`	ρ	`\rho`	Ξ	`\Xi`
ζ	`\zeta`	σ	`\sigma`	Π	`\Pi`
η	`\eta`	τ	`\tau`	Σ	`\Sigma`
θ	`\theta`	υ	`\upsilon`	Υ	`\Upsilon`
ι	`\iota`	ϕ	`\phi`	Φ	`\Phi`
κ	`\kappa`	χ	`\chi`	Ψ	`\Psi`
λ	`\lambda`	ψ	`\psi`	Ω	`\Omega`
μ	`\mu`	ω	`\omega`		

Matemáticos			
Símbolo	Sintaxis	Símbolo	Sintaxis
$\leq$	`\leq`	$\circ$	`\circ`
$\geq$	`\geq`	$\ll$	`\ll`
$\neq$	`\neq`	$\gg$	`\gg`
$\pm$	`\pm`	$\prime$	`\prime`
$\times$	`\times`	$\Leftarrow$	`\Leftarrow`
∞	`\infty`	$\angle$	`\angle`
$\sum$	`\sum`	$\surd$	`\surd`
$\int$	`\int`	$\#$	`\#`
$\div$	`\div`	$	`\$`
$\sim$	`\sim`	%	`\%`
$\leftarrow$	`\leftarrow`	&	`\&`
$\uparrow$	`\uparrow`	{	`\{`

Además de los símbolos de las tablas, los caracteres alfanuméricos pueden ser mostrados en *negrilla* precediendo el texto con `\bf` , para que se muestre en *italica* `\it` , o para que los muestre de forma normal, se usa `\rm` .

La sintaxis general es colocar el texto entre apóstrofes o comillas sencillas (`' '`), y el set de instrucciones dentro de un par de corchetes (`{}`). Esto también aplica para los que van a ser puestos como sub- o super-índices.

Ahora un ejemplo, graficar la función

$$g_2 = \cos(4\pi x)e^{-(1+x^{\beta})}$$

para $\beta = 3$ y $1 \leq \Omega_1 \leq 2$ y luego etiquetar la figura razonablemente. La figura 6.15 se genera con el script: (Apreciar que también se usa la función `grid` para generar la grilla sobre la figura)

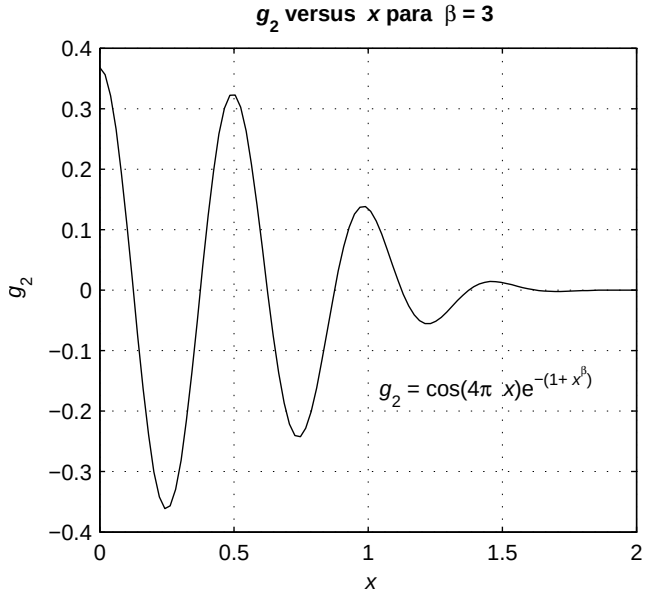


Figura 6.15: Anotaciones con sub-, super-indices y letras Griegas

```
x=linspace(0,2);
beta=3;
y=cos(4*pi*x).*exp(-(1+x.^beta));
plot(x,y,'k')
title('{\bf {\it g}_{2} versus {\it x} para {\it \beta} = 3}')
ylabel('{\it g}_{2}')
xlabel('{\it x}')
text(1.02,-0.16,
     '{\it g}_2 = cos(4\pi {\it x})e^{-(1+{\it x}^{\beta})}')
grid
```

6.4. Ejercicios

1. La principal función u objetivo de realizar una grafica, es analizar el comportamiento de la variable dependiente en función de la independiente en un rango dado. Esto permite identificar según el caso:

   - El punto mínimo (Valor mínimo)
   - El punto máximo (Valor máximo)
   - El cruce por cero de la función (Raíces)
   - El cruce de dos ecuaciones
   - etc.

   Graficando las ecuaciones, solucione el problema:

   a) El mínimo valor de la función $S(r) = \dfrac{120}{r} + 2\pi r^2$ para $0{,}1 \leq r \leq 5$

   b) La raíz de la función $f_1(t) = log(t) - 5 + t$ para $0 \leq t \leq 10$

   c) El máximo de la función $g_2(x) = \dfrac{sen(x-2)}{\sqrt{20x}}$ para $1 \leq x \leq 10$

   d) El cruce entre la función $y_1 = 1 - 0{,}5x$ y la función $y_2 = \dfrac{sen(3x)}{cos(3x)}$ para $0{,}01 \leq x \leq 0{,}4$

2. Escribir el código necesario para generar la figura 6.16.

3. Escriba un script para resolver el ejemplo dado a cada grupo:

Grupo	1	2	3	4	5	6	7	8	9	10
Ejemplo	10.10	6.11	3.8	11.6	3.18	5.13	4.22	4.22	3.8	6.11

4. Dibujar los casos de carga de la Shigley ya calculados en los ejercicios del capitulo 2.

5. El torque T aplicado por un freno en un tambor esta dado por,

$$T = \frac{4 \cdot f \cdot \mathbf{Fn} \cdot r \cdot \sin(\theta/2)}{\theta + \sin(\theta)}$$

   Donde θ es el ángulo de contacto en radianes, f es el coeficiente de fricción, r es el radio de el radio del tambor y **Fn** es la fuerza normal que actúa sobre el tambor. Dado $\theta = 60^o$ y $f = 0{,}35$.

   a) Dibuje el torque T como función del radio r, cuando r varia de 0,2 a 0,7 con incrementos de 0,01 y la fuerza F_n toma los valores de 250 y 500.

   b) Genere el código necesario para que las características de la gráfica luzcan como las mostradas en la figura 6.17.

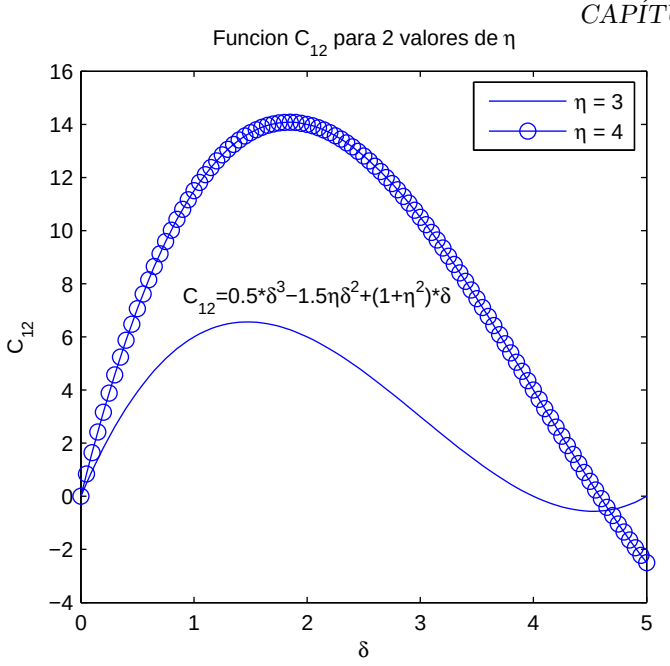


Figura 6.16: Grafica ejercicio 2

c) Dibuje sobre la figura 6.17, la gráfica obtenida.

d) Escriba TODO el código necesario para generar la figura 6.17 y sus propiedades.

e) Se sabe que el Torque necesario para detener el tambor es de 60, y que las fuerzas aplicadas F_n sólo pueden se 250 y 500. ¿Cuál debe ser el radio del tambor para cada una de las fuerzas que garanticen que el torque generado detiene el tambor?

f) Se posee un tambor de 0,45 ¿Qué torque se genera al aplicar una fuerza normal de 250?

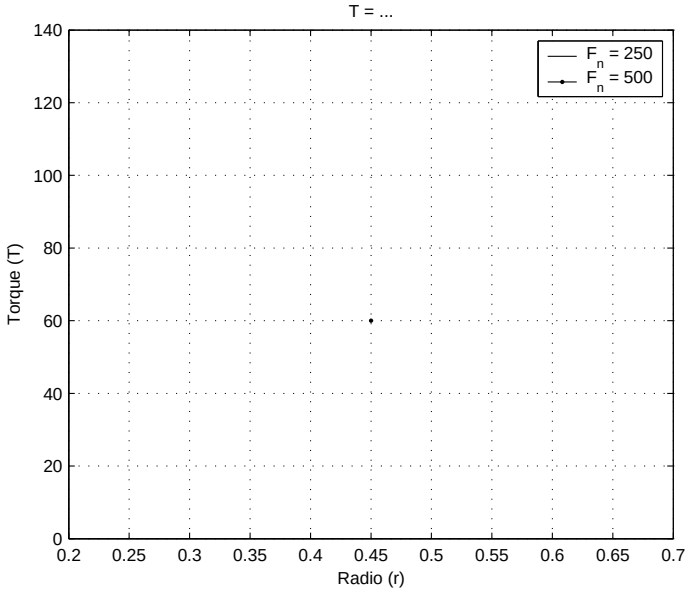


Figura 6.17: Figura obtenida

Respuestas

Capitulo 1

1.a 6,2
1.b 5,6
1.c 38
1.d 11,46
1.e 2,68
1.f 0,03
1.g 1
1.h $-1{,}41$

2.b $x = -1{,}54$
$y = 0{,}9$
2.c $x = -1{,}16$
$y = 0{,}83$

3. $d = 100\ mm$
$r = 50\ mm$
$p = 15{,}7\ mm$
$r_b = 46{,}98\ mm$
$a = 5\ mm$
$b = 6{,}25\ mm$
$t = 7{,}85\ mm$

4. $D_c = 0{,}5\ m$
$Q = 0{,}5728\ m^3/s$

5. $I_{y'} = 4{,}28\ cm^4$

6. $I_x = 0{,}071\ cm^4$

7. $I_y = 6{,}99\ cm^4$

8. $K = 1{,}31$

9. $K = 23{,}74$

10. $\theta = 0{,}368\ rad$
$L = 238{,}49\ cm$

11. $T = 36{,}58\ N \cdot m$

12. $D = 1{,}56\ m$

13. $M = 2$
$\alpha = 0{,}3997\ rad$
$a_G/\omega^2 = -1{,}34$

14. $\Delta p = 1{,}9\ Pa$

15. $a = 0{,}013\ in$
$p_{max} = 2{,}81 \times 10^5\ psi$
$\sigma_x = \sigma_y = 3 \times 10^5\ psi$
$\sigma_z = -1{,}77 \times 10^5\ psi$

16. $b = 0{,}014\ in$
$p_{max} = 23250{,}51\ psi$
$\sigma_x = -7085{,}7\ psi$
$\sigma_y = -4843{,}76\ psi$
$\sigma_z = -18775{,}2\ psi$

17. $N_L = 72{,}022$

18. $d_2 = 1{,}34\ in$
$k = 2{,}88 \times 10^7\ lb/in$

19. $T = 383{,}0075\ °C$
$\sigma_r = -8011{,}5\ psi$
$\sigma_t = 5231{,}9\ psi$

20. $\psi = 0{,}4271$

21. $K = 1{,}3394$

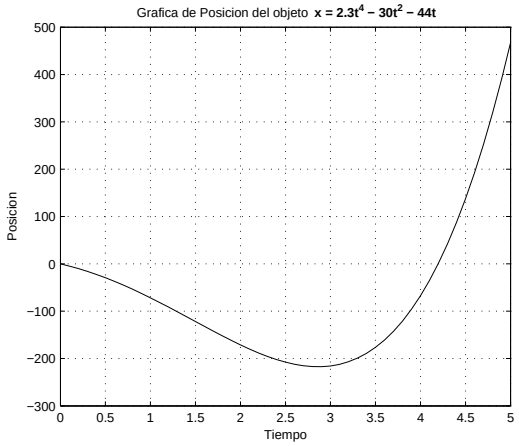
22. $F''_{21} = -55{,}27\ N \cdot m$

23. $F^x = 1573{,}8\ N$
$F^y = 1085{,}5\ N$

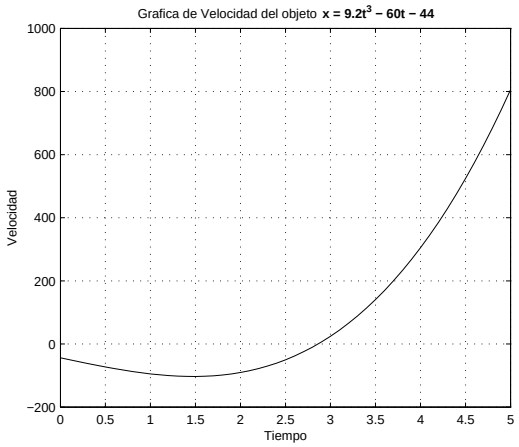
24. P = Falso
Q = Verdadero
R = Verdadero
24.a Falso
24.b Verdadero
24.c Falso

24.d Falso
24.e Verdadero
24.f Verdadero
24.g Verdadero

Capitulo 2



1.b El cuerpo cruza de nuevo el punto cero en $t = 4{,}19$
$x_{min} = -217$ en $t = 2{,}9$

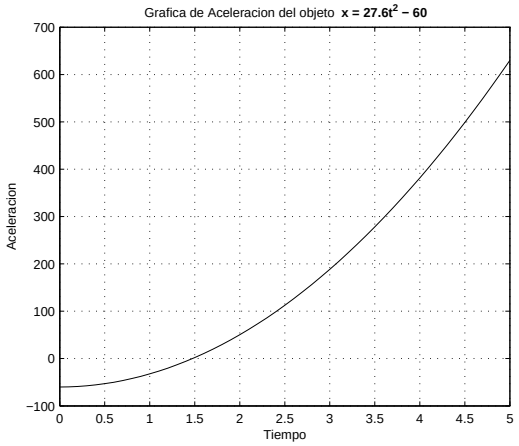


1.c $v = 9{,}2t^3 - 60t - 44$
$t_{v=0} = 2{,}86$
1.d $a = 27{,}6t^2 - 60$

2. $p = -8{,}6 \times 10^{-7}t^4 + 4{,}68 \times 10^{-4}t^3$
$-0{,}095t^2 + 8{,}83t + 27{,}37$

3.a $x_1 = -3$ y $x_2 = 1$

3.b $y_1 = -1{,}61 + 2{,}03i$
$y_2 = -1{,}61 - 2{,}03i$
$y_3 = 0{,}7388$
3.c $z_1 = -0{,}0857 + 1{,}6i$
$z_2 = -0{,}0857 - 1{,}6i$
$z_3 = -1{,}2341$
$z_4 = 0{,}7 - 0{,}17i$
$z_5 = 0{,}7 + 0{,}17i$



4.a $x^3 - 17{,}8x^2 + 95{,}35x - 156{,}75 = 0$
4.b $y^4 - 11{,}8y^3 + 31y^2 + 107{,}8y - 464 = 0$

5. $|\mathbf{U}| = 110\ N$

6. $\overrightarrow{\mathbf{F_r}} = 132{,}28\ Lb \angle -29{,}7°$

7. $\mathbf{U} \cdot \mathbf{V} = -300\ N$

8. $\mathbf{r} \times \mathbf{F} = 210\mathbf{i} - 20\mathbf{j} + 360\mathbf{k}\ [lb \cdot ft]$

9. $F_{AB} = 529{,}41\ lb$, $F_{AC} = 648{,}39\ lb$
$F_{AD} = 171{,}49\ lb$

Bibliografía

- *An Engineer's Guide to MATLAB* Edward B. Magrab. 2nd Edition. Ed. Prentice Hall. 2005
- *Advanced Math for Control Engineering* Msc. Luis Alfonso Muñoz Hernandez. Universidad de Ibagué. 2003
- *Aprenda MATLAB 5.3 como si estuviera en primero* Javier Garcia de Jalon. Escuela Técnica Superior de Ingenieros Industriales. Universidad Politécnica de Madrid. 2001.
- *Lógica de Programación y Algoritmos* Silvina Caro Pineda. Fundación Universitaria de Boyacá. Uni-Boyaca.
- *Aprenda fácilmente fundamentos para la programación de computadores* Lenin Valderrama Alvis y miguel Antonio Albornoz M. Universidad Antonio Nariño.
- *Algebra Lineal, Apuntes de Curso 2004 - 2005* Escuela Técnica Superior de Ingeniería Informática. Mexico D.F.
- *MATLAB, Edición estudiante. Versión 4 Guía de Usuario* The Math Works Inc. Ed. Prentice Hall, 1998
- *Introducción a MATLAB* Segunda Edición, Kermit Sigmon, Departamento de Matemáticas. Universidad de Florida
- *Crash course in MATLAB* Tobin A. Driscoll, Universidad de Delaware, Junio 2003.