

Python para ingenieros

P.J. De Los Santos

23 de agosto de 2018

Índice general

1. Fundamentos del lenguaje	1
1.1. ¿Qué es MATLAB?	1
1.2. Descripción del entorno de desarrollo	1
1.3. Comandos básicos y generalidades	3
1.3.1. Consultar ayuda de MATLAB	3
1.3.2. Limpiar ventana de comandos y variables del workspace	4
1.3.3. Líneas de comentarios	4
1.3.4. Valores especiales	4
1.4. Tipos de datos y operadores	4
1.4.1. Tipo logical	5
1.4.2. Tipo char	5
1.4.3. Tipo numeric	5
1.4.4. Tipo cell	6
1.4.5. Tipo struct	6
1.4.6. Referencias de función (function handle)	6
1.4.7. Identificar tipos de datos	6
1.4.8. Conversiones entre tipos de datos	7
1.4.9. Operadores aritméticos, relacionales y lógicos	8
1.5. Un mini tutorial de introducción	10
1.6. Ficheros de comandos (scripts)	14
1.6.1. Ejecutando scripts	14
1.6.2. Modificando el Path de MATLAB	15
1.7. Entradas y salidas en el Command Window	15
1.7.1. La función input	16
1.7.2. Salida sin formato: la función disp	16
1.7.3. La función fprintf	16
1.8. Funciones	17
1.8.1. Funciones, una introducción	17
1.8.2. Verificar argumentos de entrada y salida	18
1.8.3. Sub-funciones	18
1.8.4. Argumentos variables	18

1.8.5. Ayuda de una función	19
1.9. Bifurcaciones y bucles	19
1.9.1. Sentencia if-elseif-else	19
1.9.2. Sentencia switch	20
1.9.3. Bucle for	20
1.9.4. Bucle while	20
1.10.Fecha y hora	21
2. Cálculo diferencial	25
3. Cálculo integral	26
4. Cinemática de la partícula	27
5. Cinemática del sólido rígido	28

Versión incompleta

Esta es una versión incompleta del libro **Programación en MATLAB, fundamentos y aplicaciones**, te agradezco por ser un *early-reader*, cualquier comentario u observación puedes hacérmela saber a través de los siguientes medios de contacto.

- E-mail: delossantosmfq@gmail.com
- Twitter: <https://twitter.com/numython>
- Github: <https://github.com/JorgeDeLosSantos>

O bien a través de la plataforma de LeanPub en el siguiente link:

https://leanpub.com/programacionmatlab/email_author/new

Introducción

Del contenido

El por qué...

P.J. De Los Santos
Celaya, Guanajuato, México.

Fundamentos del lenguaje

1.1 ¿Qué es MATLAB?

MATLAB es un lenguaje de programación de alto nivel y entorno de desarrollo interactivo, utilizado para numerosas aplicaciones de carácter técnico y científicas. MATLAB permite realizar adquisición y análisis de datos, desarrollo de algoritmos computacionales, creación y simulación de modelos físicos y la visualización gráfica de procesos determinados. Entre los campos de uso de MATLAB se incluyen el procesamiento digital de señales, audio, imágenes y vídeo, sistemas de control, finanzas computacionales, biología computacional, redes neuronales, etc.

Características del lenguaje:

- **Interpretado:** Esta característica le convierte en un lenguaje no muy apto para aplicaciones donde la rapidez de ejecución sea crítica, pero esto mismo facilita la depuración de errores y permite un tiempo de desarrollo reducido en comparación a los lenguajes compilados tradicionales como C/C++.
- **Tipado dinámico:** No es necesario declarar el tipo de variable a utilizar, MATLAB reconoce de forma automática el tipo de dato con el que trabajará, aunque claro que es posible declarar un tipo de dato de forma explícita utilizando las funciones de conversión adecuadas.
- **Multiplataforma:** MATLAB está disponible para las plataformas más comunes: Unix, Windows, GNU/Linux y Mac OS.
- **Multiparadigma:** Soporta programación imperativa, funcional y orientada a objetos.

1.2 Descripción del entorno de desarrollo

El entorno de MATLAB mostrado en la figura 1.1 pertenece a la versión 2012b, si dispone de otra versión quizá encontrará cambios significativos en la interfaz, pero los componentes más importantes permanecen invariables.

Como puede observarse en la figura 1.1, se distinguen cuatro componentes en el escritorio del entorno MATLAB, los cuáles son:

Command Window

Ventana de comandos interactiva en la cual deberán introducirse las instrucciones de MATLAB, el prompt >> le indica que está listo para recibir instrucciones.

i ¿Qué es el prompt?

En la jerga informática, se denomina prompt al símbolo o caracter que aparece en una terminal o consola, cuando esta se encuentra en disposición de aceptar un comando de entrada.

Current Folder

Carpeta en la que se está situado, y en la que MATLAB buscará y guardará (por defecto) los archivos generados durante la sesión.

Workspace

Ventana que muestra las variables creadas por el usuario durante la sesión, indicando el nombre, valor y tipo de la misma.

Command History

Permite buscar comandos introducidos con anterioridad en la ventana de comandos y ejecutarlos nuevamente o copiarlos.



Figura 1.1 Pantalla de MATLAB R2012b

1.3 Comandos básicos y generalidades

1.3.1 Consultar ayuda de MATLAB

Uno de los puntos fuertes de MATLAB es la extensa documentación que viene adjunta al software, la cual contiene múltiples ejemplos y recomendaciones para la mayoría de las funciones. Puede acceder a la ayuda ubicando el ícono característico de ayuda, o bien tecleando la instrucción `doc` en la línea de comandos.

Si requiere una referencia rápida acerca de un comando o función puede utilizar el comando `help` seguido por el nombre la función a consultar, lo anterior le mostrará en la ventana de comandos una descripción breve referente a la función consultada, por ejemplo la siguiente línea le permite consultar ayuda rápida acerca del comando `clc`:

```
help clc clc Clear command window. clc clears the command window and homes the cursor. See also home. Reference page in Help browser doc clc
```


1.3.2 Limpiar ventana de comandos y variables del workspace

Generalmente se considera una buena práctica de programación en MATLAB iniciar los programas con instrucciones de limpiar la consola (Command Window) y borrar las variables de la memoria. Lo anterior se logra utilizando las instrucciones `clc` para limpiar la ventana de comandos y `clear` para borrar las variables del workspace. Suele acompañarse a la instrucción `clear` con el argumento adicional `all`, que permite borrar incluso variables globales, es decir conjuntamente: `clear all`.

1.3.3 Líneas de comentarios

Los comentarios de una sola línea en MATLAB deben comenzar con el símbolo de porcentaje%, todo aquello escrito después de este símbolo será ignorado por el intérprete y reconocido como comentario, asignándosele un color verde característico de forma automática.

Para hacer bloques de comentarios (o comentarios multilínea) MATLAB dispone de una sintaxis específica que se muestra enseguida:

Esto es un comentario de múltiples líneas en MATLAB, delimitado por llaves conjuntas con el signo

1.3.4 Valores especiales

En la siguiente tabla se resumen algunos valores especiales *devueltos* por funciones predefinidas en MATLAB:

Función	Descripción
<code>ans</code>	Guarda el ultimo valor no asignado a una variable
<code>eps</code>	Tolerancia que MATLAB soporta en los cálculos
<code>intmax</code>	Máximo valor entero que puede utilizarse
<code>intmin</code>	Mínimo valor entero que puede utilizarse
<code>realmax</code>	Valor de coma flotante máximo que puede representarse
<code>realmin</code>	Valor de coma flotante mínimo que puede representarse
<code>pi</code>	Constante matemática (3.14159265...)
<code>inf</code>	Valor asignado a un número demasiado grande respecto a la capacidad de cálculo del software.
<code>NaN</code>	Iniciales de “Not a Number”, tal cual traducción literal hace referencia a un valor numérico inválido.
<code>computer</code>	Devuelve el tipo de computadora que se está utilizando
<code>version</code>	Devuelve la versión de MATLAB

Tabla 1.1 Valores especiales

1.4 Tipos de datos y operadores

El siguiente listado resume los tipos de datos más comunes en MATLAB:

- `logical` (tipo booleano o lógico)
- `char` (cadenas de caracteres)

- `numeric` (datos tipo numérico)
 - `int8`, `int16`, `int32`, `int64` (tipos entero)
 - `uint8`, `uint16`, `uint32`, `uint64` (enteros sin signo)
 - `single` (flotantes de precisión simple)
 - `double` (flotantes de precisión doble)
- `cell` (arreglos de celdas)
- `struct` (estructuras)

1.4.1 Tipo logical

El tipo de dato lógico o booleano es en computación aquel que puede representar valores de lógica binaria, esto es 2 valores, valores que normalmente representan falso o verdadero.

En MATLAB las variables de tipo lógico permiten, evidentemente, dos valores, que pueden ser `true` o `false`. Una forma de declarar una variable de tipo lógico sería:

```
a = true a = 1
```

Otra manera que resulta en lo mismo es la siguiente:

```
a=logical(1) a = 1
```

Las líneas anteriores crean una variable `a` de tipo lógico con un valor `true`.

La utilidad de los variables lógicas se hace evidente cuando es necesario tomar decisiones respecto al estatus o valor de una variable o subrutina.

1.4.2 Tipo char

Son cadenas de caracteres, que pueden contener valores alfanuméricos e incluso símbolos especiales. Para declararlas no hace falta especificar que son variables tipo `char`, dado que MATLAB es de tipado dinámico y reconoce como tal aquellas cuyo valor asignado se encuentre delimitado por comillas simples, un ejemplo muy clásico es el siguiente:

```
txt = 'Hola Mundo' txt = Hola Mundo
```

El manejo de cadenas de caracteres o strings es una cuestión fundamental en la programación de computadoras. En MATLAB las cadenas de caracteres tienen muchas utilidades, entre las cuales pueden citarse: sirven como argumentos de entrada para funciones (como todo tipo de dato), identificadores para campos de estructuras, anotaciones y etiquetas en las gráficas, lectura y escritura de datos no uniformes, entre muchas otras.

En el capítulo 3 se aborda el manejo de strings de una manera muy completa, desde operaciones básicas como las comparaciones y concatenación, hasta el uso de expresiones regulares.

1.4.3 Tipo numeric

Normalmente cuando en MATLAB tecleamos un valor numérico o bien lo asignamos a una determinada variable, esta será de tipo `double`, a menos que se haga una conversión explícita a otro tipo de dato. Por ejemplo, si insertamos en MATLAB lo siguiente:

```
num = 10 num = 10
```

Y posteriormente tecleamos la instrucción `whos` para verificar el tipo o clase de dicha variable:

```
whos Name Size Bytes Class Attributes num 1x1 8 double
```

Si se requiere utilizar un dato de tipo entero habrá de realizarse la conversión como sigue:

```
numInt = int8(23) numInt = 23 whos Name Size Bytes Class Attributes numInt 1x1 1 int8
```

1.4.4 Tipo cell

Un cell array es un tipo de dato característico del lenguaje MATLAB que consiste en un arreglo multidimensional de celdas que pueden contener cualquier tipo de dato, inclusive otro cell array. Un ejemplo muy sencillo de cell array se muestra enseguida:

```
C=10,'MATLAB','5',[1 1] C = [10] 'MATLAB' '5' [1x2 double]
```

1.4.5 Tipo struct

Las estructuras son arreglos de datos que, de forma similar a los cell arrays, pueden almacenar variables de diversos tipos. Para la organización de los datos se utilizan campos que pueden contener sólo un tipo de dato. A continuación se muestra un ejemplo de cómo crear una estructura:

```
Alumno.Nombre='Jorge'; Alumno.Apellido='De Los Santos'; Alumno.Cursos='Programación','Cálculo','Métodos Numéricos'; Alumno.Notas=[10 9 10]; Alumno Alumno = Nombre: 'Jorge' Apellido: 'De Los Santos' Cursos: 'Programación' 'Cálculo' 'Métodos Numéricos' Notas: [10 9 10]
```

En el capítulo 3 se tratan con más detenimiento las estructuras y su utilidad en la programación en MATLAB.

1.4.6 Referencias de función (function handle)

Las *function handle* son referencias asociadas a una función nativa de MATLAB o bien a una función anónima creada por el usuario.

El siguiente ejemplo muestra la definición de una función anónima y su posterior uso mediante su referencia:

```
f=@(x) x+cos(x) f = @(x)x+cos(x) whos Name Size Bytes Class Attributes f 1x1 32 function_handle >> fzero(f,0)ans = -0.7391 >> f(pi/2)ans = 1.5708
```

1.4.7 Identificar tipos de datos

Para identificar tipos de datos en MATLAB se cuentan con diversos comandos que nos facilitan esta tarea. El comando `whos` nos proporciona información acerca de las variables existentes en el workspace, tales como el nombre, tamaño y tipo. A manera de ejemplo crearemos las siguientes variables e introducimos la instrucción `whos` para verificar el tipo de información que nos imprime en la consola:

```
n=10; val=false; s='MATLAB'; C=1,2,3; ST.Nombre='Anna'; whos Name Size Bytes Class Attributes C 1x3 360 cell ST 1x1 184 struct n 1x1 8 double s 1x6 12 char val 1x1 1 logical
```

Además del comando `whos`, puede utilizarse la función `class` para determinar el tipo de dato de una variable pasada como argumento, por ejemplo:

```
a=3; class(a) ans = double
```

1.4.8 Conversiones entre tipos de datos

Las conversiones entre tipos de datos son muy utilizadas en la programación en cualquier lenguaje, puesto que permiten controlar la precisión de los cálculos, mejorar la presentación de los datos o bien evitar errores en la ejecución.

Entre tipos numéricos

Cuando se crea una variable de tipo numérico en MATLAB por defecto será de tipo `double`, por ejemplo, creamos una variable llamada `num`:

```
num=2; class(num) ans = double
```

Las conversiones entre tipos numéricos son de sintaxis muy sencilla, solo habrá que especificar el tipo de dato al cual se convertirá, siendo permitidos los especificados en la tabla siguiente:

Tipo de dato	Sintaxis de conversión	Rango
Precisión doble	<code>double(num)</code>	2.2251e-308 a 1.7977e+308
Precisión simple	<code>single(num)</code>	1.1755e-38 a 3.4028e+38
Entero de 8 bits	<code>int8(num)</code>	-128 a 127
Entero de 16 bits	<code>int16(num)</code>	-32768 a 32767
Entero de 32 bits	<code>int32(num)</code>	-231 a 231-1
Entero de 64 bits	<code>int64(num)</code>	-263 a 263-1
Entero sin signo de 8 bits	<code>uint8(num)</code>	0 a 255
Entero sin signo de 16 bits	<code>uint16(num)</code>	0 a 65535
Entero sin signo de 32 bits	<code>uint32(num)</code>	0 a 4294967295
Entero sin signo de 64 bits	<code>uint64(num)</code>	0 a 18446744073709551615

Tabla 1.2 Conversiones entre tipos numéricos

Así, podemos convertir la variable `num`, creada con anterioridad, a otro tipo de dato numérico, por ejemplo a un entero de 8 bits:

```
num=int8(num); class(num) ans = int8
```

Es necesario poner especial atención en los rangos que pueden manipularse con cada tipo numérico, debido a que por ejemplo si se realiza la siguiente conversión:

```
num=int8(653) num = 127
```

El valor que ha sido pasado como argumento de conversión excede el rango para un entero de 8 bits, por lo cual simplemente se le asigna el máximo valor permitido para una variable de este tipo.

Si requiere verificar por usted mismo los valores máximos y mínimos permitidos para cada tipo de dato, puede usar las funciones `realmin` y `realmax` para los tipos de coma flotante, y las correspondientes `intmin` e `intmax` para tipos enteros.

De string a tipo numérico

Para este tipo de conversiones MATLAB dispone de las funciones `str2double` y `str2num`, en algunos casos no notará la diferencia en los resultados, salvo en la rapidez de ejecución. Pese a lo anterior, es necesario tomar en cuenta cómo trabaja cada función y cual le resulta de utilidad; con `str2double` se convierte una variable tipo string en un valor de tipo `double`, la función `str2num` también realiza conversión a tipo `double` pero además realiza conversiones a otros tipos de datos numéricos si se

especifica de manera explícita, de hecho esta tiene una funcionalidad muy similar a la de la función `eval`. Los siguientes ejemplos muestran las diferencias y utilidades de las funciones descritas.

```
a=str2double('1237') a = 1237 b=str2num('1237') b = 1237 whos
Name Size Bytes Class Attributes
a 1x1 8 double b 1x1 8 double
```

1.4.9 Operadores aritméticos, relacionales y lógicos

Los operadores son símbolos especiales fundamentales en un lenguaje de programación, se utilizan para *operar* sobre variables (operandos) y, de manera general, pueden clasificarse en tres grupos:

- Operadores aritméticos
- Operadores relacionales
- Operadores lógicos

Operadores aritméticos

Los operadores aritméticos toman valores numéricos como entrada y devuelven un valor resultante de aplicar la operación correspondiente sobre los operandos. Por ejemplo, vea la siguiente expresión que corresponde a una suma de escalares:

```
1+2 ans = 3
```

En este caso 1 y 2 son los operandos o valores sobre los cuales se aplica el operador +, y 3 es, evidentemente, el resultado de ejecutar la operación.

En la siguiente tabla se listan los principales operadores aritméticos disponibles en MATLAB.

Operador	Descripción
+	Operador suma
-	Operador resta
*	Operador multiplicación (escalares)
/	Operador división
./	División elemento a elemento (matrices)
.*	Multiplicación elemento a elemento (matrices)

Tabla 1.3 Operadores aritméticos

Note que además de los operadores correspondientes a las operaciones aritméticas básicas (suma, resta, multiplicación, división, potenciación), se tiene operadores que realizan operaciones elemento a elemento, que se utilizan para operar sobre arreglos matriciales.

Por ejemplo, sean A y B dos matrices de 3x3 definidas como

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad A = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

Si realizamos una suma o resta con estas matrices no habrá mayor complicación, dado que tanto la suma y resta matricial se realizan elemento a elemento:

```
A=[1,2;3,4]; B=[5,6;7,8]; A+B ans = 6 8 10 12 A-B ans = -4 -4 -4 -4
```

Luego, note las diferencias de utilizar la multiplicación elemento a elemento (.*) y la ordinaria (*):
 $A*B$ ans = 19 22 43 50 $A.*B$ ans = 5 12 21 32

Sí, efectivamente los resultados son completamente diferentes: en el primer caso se realiza una multiplicación matricial, siguiendo las reglas dictadas por el álgebra de matrices, en el segundo caso lo que se hace es una multiplicación elemento a elemento, es decir, cada elemento en la posición (i, j) de A se multiplica con el elemento ubicado en la misma posición de B .

Operadores relacionales

Los operadores relacionales se utilizan para comparar dos valores, devolviendo un valor lógico. Normalmente se utilizan en conjunto con las estructuras de control para la toma de decisiones sobre el procedimiento o flujo de un programa.

La siguiente tabla resume los operadores relacionales: su notación y descripción.

Operador	Descripción
==	Igual a
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
~=	Diferente de

Tabla 1.4 Operadores relacionales

Puede ver que la notación difiere de la que ordinariamente utilizamos cuando escribimos en papel, por ejemplo el símbolo del menor o igual que \leq se escribe como $<=$. Es importante notar que la comparación “igual que” se realiza con un doble signo igual ($==$), puesto que el uso de un único signo corresponde a la asignación, a continuación se muestra lo que ocurre cuando intentamos hacer comparaciones utilizando sólo un signo $=$:

$1==1$ ans = 1 $1=1$ 1=1 | Error: The expression to the left of the equals sign is not a valid target for an assignment.

En el primer caso (doble signo) la comparación se hace devolviendo un true, pero, en el segundo nos manda un error de sintaxis, indicando que el 1 ubicado a la izquierda del signo igual no es un carácter válido para realizar una asignación.

Operadores lógicos

Operador	Descripción
&	Operador lógico and
	Operador lógico or
~	Operador lógico not

Tabla 1.5 Operadores lógicos

1.5 Un mini tutorial de introducción

Una vez conocidos los tipos de datos y los operadores, podemos comenzar con una breve introducción al uso de MATLAB como una poderosa calculadora muy fácil de utilizar, además vamos a ver algunas características interesantes. Si hay algo en esta sección que te parece difícil de asimilar, no debes preocuparte, lo subsiguiente se abordará en capítulos posteriores de manera más *detenida*.

Como se ha descrito en secciones anteriores, el command window o ventana de comandos es la parte del entorno MATLAB que nos permite interactuar de forma dinámica, si tecleamos una instrucción automáticamente nos devolverá un resultado y se crearán variables en las cuales se almacenen los diversos valores de salida. Por ejemplo, vamos a teclear una simple suma aritmética:

```
3+2 ans = 5
```

Puede verificar que en el workspace ahora aparece una variable llamada `ans` con valor de 5, en `ans` se guarda por defecto el último resultado no asignado a una variable, podríamos asignar el resultado de la suma a una variable específica:

```
suma=3+2 suma = 5
```

Podemos también asignar valores a determinadas variables y enseguida utilizarlas para ejecutar alguna operación, por ejemplo:

```
a=5; b=7; a*b ans = 35 a-b ans = -2 a/b ans = 0.7143
```

Note que el colocar un punto y coma (;) al final de una instrucción evita que se muestre un resultado de salida, lo cual no afecta en el almacenamiento de los valores correspondientes, pero podría resultar de mucha ayuda al momento de seleccionar los valores que se quieren mostrar en la ventana de comandos.

MATLAB también tiene disponible diversas funciones matemáticas predefinidas, que pueden ser aplicadas sobre un número o sobre una matriz o arreglo de números. Algunas funciones trigonométricas:

```
sin(pi/2) ans = 1 cos(pi/4) ans = 0.7071 tan(pi/3) ans = 1.7321
```

Note que el valor de la constante π está predefinida en MATLAB mediante la cadena `pi`:

```
pi ans = 3.1416
```

MATLAB devuelve un valor de 3.1416, lo cual es un valor *redondeado* de π , pero esto es cuestión solamente de la representación, normalmente se utiliza el formato short (4 dígitos después del punto decimal) para la representación de valores numéricos, internamente MATLAB utiliza más dígitos para *manejar* y operar con el valor de π . Si queremos obtener más dígitos en la salida por consola podemos cambiar el formato de salida:

```
format long pi ans = 3.141592653589793
```

El formato largo permite representar una cantidad con 16 decimales. Incluso es posible *forzar* a que se muestre una representación en forma racional:

```
format rat pi ans = 355/113 0.1+0.123 ans = 223/1000 0.125 ans = 1/8
```

Se puede crear una lista o arreglo de valores numéricos encerrando estos entre corchetes, y separando cada elemento por comas o espacios.

```
A=[5,8,10,2,7] A = 5 8 10 2 7 B=[3 7 1 0 -2] B = 3 7 1 0 -2
```

Se puede obtener el valor máximo y mínimo de un arreglo numérico utilizando las funciones `max` y `min` respectivamente.

```
max(A) ans = 10 min(A) ans = 2
```

También podemos calcular el promedio de los valores utilizando la función `mean`:

```
mean(A) ans = 6.4000
```

Obtener la cantidad de elementos que componen lista con `length` o `numel`:

```
length(A) ans = 5  numel(A) ans = 5
```

Incluso podemos representar gráficamente cada uno de los elementos de un vector mediante la función `plot`:

```
x=[1,2,1,3,4,2,0,1]; plot(x);
```



Figura 1.2 Gráfica de un vector de puntos


De manera similar puede evaluar una función matemática en un intervalo determinado y trazar su gráfica:

```
x=0:0.01:10; y=sin(x); plot(x,y)
```

En lo anterior, se crea un vector `x` en el intervalo $[0,10]$, con incrementos de 0.01, es decir, el vector contiene los puntos:

$$x = [0, 0.01, 0.02, 0.03, \dots, 9.99, 10]$$

Luego, al aplicar la función `sin` sobre ese vector, MATLAB evalúa la función seno en cada uno de los puntos o valores contenidos en el arreglo `x` y los guarda en `y`. El capítulo 2 (Vectores y matrices) trata con mayor profundidad la notación de dos puntos utilizada y el manejo correcto de estructuras matriciales.




src/img/ch1/img_1_3.png

Figura 1.3 Gráfica de la función $f(x) = \sin x$

¿Bastante interesante, verdad?. Bueno, incluso es posible trazar gráficas de superficies tridimensionales con unas cuantas líneas de código:

```
[X,Y]=meshgrid(0:0.1:10, 0:0.1:10); Z = sin(X)+cos(Y); surf(X,Y,Z)
```

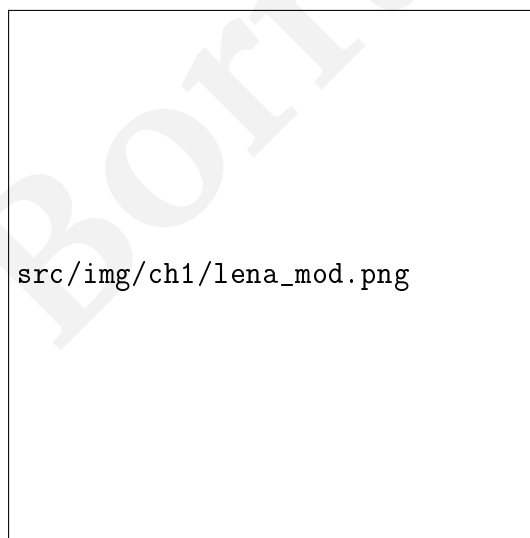


src/img/ch1/img_1_4.png

Figura 1.4 Gráfica de una función $f(x, y)$

Lo mismo podemos leer una imagen y hacerle algunos cambios (restauración, segmentación, etc,...) utilizando el *Image Processing Toolbox* (que es una colección de códigos MATLAB que facilitan esta tarea). Vea el siguiente ejemplo:

```
img = imread('lena_td.tif'); >> img = rgb2gray(img); >> filtro = [111; 1 - 81; 111]; >>  
img_mod = imfilter(img, filtro); >> imshow(255 - img_mod)
```

**Figura 1.5** Imagen de Lena original**Figura 1.6** Imagen de Lena modificada

Todo esto muestra un poco de lo que puede hacer MATLAB, pero, lo cierto es que es un entorno muy completo con una *infinidad* de opciones que facilitan el desarrollo de algoritmos para aplicaciones en múltiples disciplinas científicas. En los capítulos posteriores de este texto se abordan algunas características y herramientas proporcionadas por MATLAB para algunos campos específicos.

1.6 Ficheros de comandos (scripts)

Los ficheros de comandos, conocidos también como *scripts*, son archivos de texto sin formato (ASCII) con la extensión característica de los archivos de MATLAB (*.m), se utilizan para almacenar una serie de comandos o instrucciones que se ejecutan sucesivamente y que habrán de realizar una tarea específica. Los scripts de MATLAB pueden editarse utilizando cualquier editor de texto sin formato (Bloc de Notas, Notepad++, Sublime Text, etc. . .), aunque es más recomendable utilizar el editor de MATLAB, puesto que proporciona herramientas que facilitan la corrección de errores, el control sobre la ejecución del código y la capacidad de autocompletado y sugerencias cuando se utilizan funciones nativas de MATLAB.

Para crear un nuevo script puede pulsar la combinación **Ctrl + N** (bajo SO Windows), o buscar en la interfaz de MATLAB la opción **New** y enseguida seleccionar **Script**; si prefiere hacerlo desde la ventana de comandos puede introducir el comando `edit` que le abrirá un nuevo script.

Para guardar un fichero de comandos utilice la opción **Save** de la barra de herramientas o bien mediante la combinación de teclas **Ctrl + S** en Windows. Debe tomarse en cuenta que al guardar un script se le proporcione un nombre que no entre en conflicto con las funciones nativas de MATLAB o las palabras reservadas del lenguaje. Algunas recomendaciones que deben seguirse para nombrar un script son:

- El nombre deberá contener sólo letras, números o guiones bajos.
- No deberá comenzar con un carácter diferente a una letra (Por ejemplo: 102metodo.m, es un nombre inválido dado que comienza con un número).
- Evite utilizar nombres de funciones nativas de MATLAB o palabras reservadas del lenguaje que podrían ocasionar conflictos.

Para saber cuáles son las palabras reservadas del lenguaje puede teclear `iskeyword` en la ventana de comandos y MATLAB le devolverá un cell array de strings:

```
iskeyword ans = 'break' 'case' 'catch' 'classdef' 'continue' 'else' 'elseif' 'end' 'for' 'function' 'global' 'if' 'otherwise' 'parfor' 'persistent' 'return' 'spmd' 'switch' 'try' 'while'
```

Además, puede verificar si el nombre de un fichero existe utilizando la función `exist`:

```
exist('size') ans = 5
```

Si devuelve un resultado diferente de cero, entonces ese nombre está siendo utilizado en una de las funciones/scripts incluidas en el path de MATLAB.

1.6.1 Ejecutando scripts

La utilidad de los scripts radica en la posibilidad de almacenar comandos de manera estructurada y poderlos ejecutar posteriormente, para hacerlo puede ir a la opción **Run** de la interfaz principal de MATLAB y entonces se ejecutarán todas las instrucciones que conforman el script.

Otra forma es ubicarse en la carpeta del script y teclear el nombre del fichero en el Command Window. Claro que si el fichero no se encuentra en el *Current Folder* este no se ejecutará, exceptuando aquellos que sean agregados al *Path* de MATLAB.

i De los ficheros de funciones

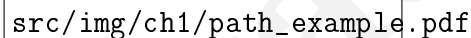
Para ejecutar los ficheros que contienen definiciones de funciones no se procede como se ha descrito anteriormente, puesto que, normalmente, estos necesitan *información extra* o argumentos de entrada que deben pasarse utilizando la sintaxis de *llamada de funciones*, misma que será objeto de estudio en secciones posteriores.

1.6.2 Modificando el Path de MATLAB

Primero, ¿qué es el path de MATLAB?, en resumen, son directorios o carpetas en los cuales MATLAB busca las funciones, clases y/o ficheros en general que el usuario demanda durante una sesión.

Si teclea el comando `path`, este imprimirá en pantalla una lista de directorios, ordenados de manera jerárquica, en los cuales MATLAB busca las sentencias introducidas.

Por ejemplo, vamos a suponer que en nuestro directorio actual tenemos un fichero llamado `principal.m` y que tenemos también una carpeta `utils` en la cual tenemos algunos códigos necesarios (`codigo1.m` y `codigo2.m`) para que nuestro código principal funcione:



src/img/ch1/path_example.pdf

Una solución evidente (pero muy *tosca*) es colocar los ficheros `codigo1.m` y `codigo2.m` en el mismo directorio, pero claro, eso implicaría tener muchos ficheros en una misma carpeta, lo cual no suele ser buena idea.

Y la otra solución consiste en agregar la carpeta `utils` al path de MATLAB, lo cual es tan sencillo como ejecutar:

```
path(path, 'utils');
```

Con esto podrá llamar los ficheros `codigo1.m` y `codigo2.m` desde `principal.m` sin necesidad de colocarlos en la misma carpeta.

1.7 Entradas y salidas en el Command Window

En la sección ?? se describió al Command Window (ventana de comandos) y se hizo referencia a este como la parte del escritorio de MATLAB que permite interactuar tecleando instrucciones y devolviendo al instante un resultado. En esta sección veremos cómo utilizar funciones que permitan introducir y mostrar ciertos valores de manera controlada por el usuario.

1.7.1 La función input

La función input permite *pedir* un valor al usuario utilizando una cadena de caracteres como prompt, la sintaxis es muy sencilla:

```
var=input('Introduzca un valor: ');
```

En la variable var se guarda el valor que el usuario introduzca, los valores aceptados por la función input pueden ser de tipo numérico, cell arrays, e inclusive tipo char. Aunque para introducir cadenas de texto la función input dispone de un modificador que hará que la entrada se evalúe como una variable tipo char o cadena de texto, la sintaxis para esto es la siguiente:

```
var=input('Introduzca una cadena de texto: ', 's');
```

La letra s entre comillas simples le indica a MATLAB que deberá evaluar la entrada como tipo string.

1.7.2 Salida sin formato: la función disp

La función disp muestra en pantalla el valor de una determinada variable que se pasa como argumento, por ejemplo:

```
a=3; disp(a) 3
```

Para el caso anterior se pasa como argumento la variable a que ha sido declarada previamente y simplemente se muestra el valor correspondiente a esta. Las variables a mostrar pueden ser de cualquier tipo, incluyendo cadenas de texto, matrices, cell arrays y estructuras, véanse los siguientes ejemplos:

```
disp(magic(3)) 8 1 6 3 5 7 4 9 2 disp(1,0,2,-2) [1] [0] [2] [-2] disp('Hola Mundo') Hola Mundo
```

Con disp también es posible mostrar enlaces a un sitio web, utilizando la sintaxis HTML para un enlace dentro de la función disp, por ejemplo:

```
>> disp('<a href="http://matlab-typ.blogspot.mx">MATLAB TYP</a>');  
MATLAB TYP
```

1.7.3 La función fprintf

Con fprintf es posible dar formato a la salida que se quiere imprimir en pantalla, por ejemplo, es posible especificar el número de decimales que se mostrarán o bien si se quiere mostrar como un entero o quizá como una cadena de texto. La sintaxis de la función fprintf es como sigue:

```
fprintf('Especificaciones de formato',a1,...,an);
```

Donde las especificaciones de formato incluyen uno o más de los identificadores de un mismo tipo o combinados que se muestran en la siguiente tabla:

Identificador	Formato de salida
%d	Tipo entero
%f	Tipo coma flotante
%g	Tipo coma flotante compacta.
%u	Tipo entero sin signo
%e	Tipo coma flotante, notación exponencial
%s	Tipo char, cadena de texto
%c	Tipo char, carácter a carácter.

Tabla 1.6 Opciones de formato para fprintf

Véase el siguiente ejemplo:

```
fprintf('3.141593e+00
```

Observe que se imprime el valor de π en este caso, pero el prompt de la ventana de comandos queda situado justo después del valor de salida en la misma línea, para evitar lo anterior puede utilizar la secuencia de escape `||` después del valor a imprimir, lo cual le indica a MATLAB que debe comenzar en una nueva línea. Modificamos y vemos el resultado que produce:

```
fprintf('3.141593e+00
```

Ahora observe lo que se imprime utilizando otros identificadores:

```
fprintf('3.141593 fprintf('3.14159 fprintf('3.141593e+00 fprintf('3.141593e+00
```

Para las salidas de coma flotante puede especificar el número de decimales que tendrá la salida, por ejemplo si desea mostrar solamente dos decimales del número π :

```
fprintf('3.14
```

1.8 Funciones

1.8.1 Funciones, una introducción

Las funciones son porciones de código que por lo general aceptan argumentos o valores de entrada y devuelven un valor de salida. Una función es una herramienta muy útil en la programación, dado que permite la reutilización de código para procedimientos que lo requieran, así como una facilidad significativa para mantener el código, lo cual se traduce en una mayor productividad. MATLAB, de hecho, está compuesto por una multitud de funciones agrupadas en *toolboxes*, cada una de ellas pensada para resolver una situación concreta.

Una función debe definirse en un fichero único, es decir, por cada función creada debemos utilizar un archivo `*.m`, mismo que tendrá el nombre dado a la función.

La estructura básica de una función contiene los siguientes elementos:

- La palabra reservada `function`
- Los valores de salida
- El nombre de la función
- Los argumentos de entrada
- Cuerpo de la función

Para una mejor comprensión de cada uno de esos elementos, refiérase a las siguientes líneas de código:

```
function res = suma(a,b) res = a+b; end
```

La función anterior llamada `suma`, recibe como argumentos de entrada dos valores numéricos `a` y `b`, y devuelve un resultado guardado en `res` que equivale a la suma aritmética de las variables de entrada. Note que el valor de retorno debe asignarse a la variable indicada en la línea de definición.

Si ejecutamos la función en la ventana de comandos obtenemos algo similar a esto:

```
s=suma(3,2) s = 5
```

Si no hace una asignación el resultado devuelto se guarda en la variable `ans`.

1.8.2 Verificar argumentos de entrada y salida

Cuando se crea una función es recomendable verificar si la cantidad de argumentos de entrada corresponden a los soportados, o bien, si el tipo de dato que se ha introducido es el adecuado para proceder con el resto de la programación; MATLAB proporciona los comandos `nargin` y `nargout` que sirven para *contar* el número de argumentos de entrada y salida respectivamente.

Utilizando como ejemplo la función `suma` creada con anterioridad, podemos verificar que el número de argumentos sean exactamente dos para poder proceder y en caso contrario enviar al usuario un mensaje de error en la ventana de comandos, el código implicado sería similar al siguiente:

```
function res = suma(a,b) if nargin==2 res=a+b; else error('Introduzca dos argumentos de entrada'); end end
```

Si ejecutamos la función pasándole solamente un argumento de entrada nos devolverá un mensaje de error:

```
s=suma(7) Error using suma (line 5) Introduzca dos argumentos de entrada
```

1.8.3 Sub-funciones

Las sub-funciones son funciones definidas dentro del espacio de otra función principal. Se utilizan como funciones auxiliares con la finalidad de hacer más legible el código y facilitar la depuración de errores. Enseguida se muestra el ejemplo de una sub-función:

```
function r=isfibo(num) ff=fibonacci(num); if any(ff==num) r=true; else r=false; end function F=fibonacci(n) F(1:2)=1; i=3; while 1 F=[F F(i-1)+F(i-2)]; if F(end) >= n,break,end; i=i+1; end end end
```

La función anterior `isfibo` determina si el entero pasado como argumento de entrada pertenece a la sucesión de Fibonacci, para ello utiliza como una función auxiliar a la sub-función `fibonacci` que se encarga de generar la sucesión de Fibonacci en un intervalo dado y guardarlo en un vector de salida. Una sub-función puede ser llamada solamente por la función principal que la contiene.

1.8.4 Argumentos variables

En la introducción a las funciones se ha mencionado que estas por lo general tienen un número específico de argumentos de entrada y salida, no obstante se presentan situaciones en donde los argumentos de entrada o salida de una función no son fijos o bien los argumentos pueden ser demasiados de tal modo que resulte incómodo definirlos en la línea correspondiente. Para solucionar lo anterior MATLAB permite el uso de `varargin` y `varargout` como argumentos de entrada y salida respectivamente. Para tener una idea más práctica de lo anterior véase el ejemplo siguiente:

```
function m=max2(varargin) if nargin==1 v=varargin1; m=v(1); for i=2:length(v) if v(i)>m m=v(i); end end elseif nargin==2 a=varargin1; b=varargin2; if a>b m=a; else m=b; end end end
```

La función anterior `max2` emula a la función nativa `max`, puede recibir como argumento de entrada un vector o bien dos valores escalares. Si observa el código anterior notará que `varargin` es un cell array que guarda todos los argumentos de entrada pasados a la función, como se verá en el capítulo 3 la manera de acceder a los elementos de un cell array es utilizando la sintaxis: `var{k}`, donde `var` es la variable en la que está almacenada el cell array y `k` es el *k*-ésimo elemento contenido en el cell array.

1.8.5 Ayuda de una función

Como parte de las buenas prácticas de programación es recomendable incluir comentarios dentro de una función que indiquen el propósito de esta, así como una descripción breve de los argumentos de entrada y salida e incluso un ejemplo concreto de la misma.

Por convención estos comentarios deben colocarse justamente después de la definición de la función y antes de todo el código restante, además de que esto servirá como referencia al resto de usuarios también le permitirá a MATLAB interpretarlo como las líneas de ayuda cuando se le solicite expresamente mediante la función `help`. Véase el siguiente ejemplo:

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

```
x1=(1/(2*a))*(-b+sqrt(b^2-4*a*c)); x2=(1/(2*a))*(-b-sqrt(b^2-4*a*c)); end
```

Podemos teclear `help ecuat` en la ventana de comandos y verificar lo que MATLAB nos devuelve como ayuda de la función:

```
help ecuat Resuelve una ecuación cuadrática de la forma: a*x^2+b*x+c = 0Argumentosdeentrada :  
a - Coeficientecuadraticob - Coeficientelinealc - CoeficienteconstanteArgumentosdesalida :  
x1,x2 - RacesdelaecuacincuadraticaEjemplo :>> [r1,r2] = ecuat(-1,2,1);
```

Es común agregar a la ayuda de una función algunas referencias hacia otras funciones similares, para ello en los comentarios debe agregar una línea que comience con las palabras `SEE ALSO` (Ver también), seguidas de las funciones similares separadas por comas, véase el ejemplo a continuación:

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

```
x1=(1/(2*a))*(-b+sqrt(b^2-4*a*c)); x2=(1/(2*a))*(-b-sqrt(b^2-4*a*c)); end
```

```
help ecuat Resuelve una ecuación cuadrática de la forma: a*x^2+b*x+c = 0Argumentosdeentrada :  
a - Coeficientecuadraticob - Coeficientelinealc - CoeficienteconstanteArgumentosdesalida :  
x1,x2 - RacesdelaecuacincuadraticaEjemplo :>> [r1,r2] = ecuat(-1,2,1);
```

```
SEE ALSO roots,solve,fzero
```

1.9 Bifurcaciones y bucles

1.9.1 Sentencia if-elseif-else

La sentencia `if` se utiliza como bifurcación simple por sí sola, es decir, en aquellas situaciones en las cuales se requiera evaluar solamente una condición, por ejemplo, suponga que tiene dos números a y b y necesita comprobar si son iguales y ejecutar una acción, para ello bastaría con una sentencia `if` simple:

```
if a==b disp('a es igual a b'); end
```

A diferencia del caso anterior hay situaciones que requieren la ejecución de una acción cuando la condición se cumpla y de otra en caso contrario, entonces puede utilizarse una bifurcación doble formada por las sentencias `if-else`. Retomando el ejemplo para la bifurcación `if` simple, podríamos modificarlo de tal manera que envíe también un mensaje (ejecute una acción) para cuando la condición no se cumple:

```
if a==b disp('a es igual a b'); else disp('a es diferente de b'); end
```

Ahora imagine que para los ejemplos anteriores se necesita especificar si $a=b$, si $a > b$ o bien si $a < b$, lo cual implicaría tener una sentencia de selección múltiple `if-elseif-else` que permite escoger entre varias opciones, evaluándose en orden descendente, por ejemplo refiérase a la siguiente estructura:


```
if cond1 elseif cond2 elseif cond3 . . . elseif condN else end
```

MATLAB evalúa primeramente la condición 1 contenida en la sentencia `if (cond1)` y en el caso de no cumplirse evalúa la siguiente condición de forma sucesiva (`cond2`, `cond3`, ...); finalmente y en el caso de que ninguna de las opciones evaluadas se cumpla, se ejecuta la instrucción contenida en la sentencia `else`. A continuación se muestra el ejemplo de una bifurcación múltiple para la situación descrita al principio:

```
if a==b disp('a es igual que b'); elseif a>b disp('a es mayor que b'); elseif a<b disp('a es menor que b'); end
```

1.9.2 Sentencia switch

La sentencia `switch` es una bifurcación múltiple que permite seleccionar entre varias opciones o casos la acción a ejecutar. La sintaxis general es:

```
switch var case opc1 case opc2 . . . otherwise end
```

Siendo `var` la variable que servirá como criterio de selección. Después de la palabra reservada `case`, se coloca el valor de `var` para el cual se ejecutarán esas instrucciones, y en `otherwise` se insertan las instrucciones que MATLAB deberá ejecutar por defecto en caso de no cumplirse ninguno de los casos especificados.

Enseguida se muestran dos ejemplos correspondientes a la sentencia de selección `switch`:

```
X=input('Inserte 0 o 1: '); switch X case 0 disp('Insertó cero'); case 1 disp('Insertó uno'); otherwise warning('Valor incorrecto, verifique'); end
letra=input('Inserte una letra: ','s'); switch letra case 'a','e','i','o','u' disp('Es una vocal'); otherwise disp('Es una consonante'); end
```

1.9.3 Bucle for

La sintaxis general de un bucle `for` se muestra enseguida:

```
for i=inicio:incremento:fin end
```

El valor `inicio` es a partir del cual se ejecutará el ciclo, el `incremento` es la cantidad que varía en cada paso de ejecución, y el valor de `final` establece el último valor que tomará el ciclo.

El siguiente código muestra un ciclo `for` muy básico, el cual simplemente muestra en consola el valor actual adquirido por la variable.

```
for i=1:10 fprintf('Valor actual: end
```

Cuando no se especifica el `incremento`, como el caso anterior, MATLAB asume que es unitario.

Es posible utilizar ciclos `for` anidados, por ejemplo para cuando se requiere recorrer una matriz en sus dos dimensiones y ejecutar operaciones elemento por elemento. Véase el siguiente ejemplo:

```
A=round(rand(5)*10); for i=1:5 for j=1:5 disp(A(i,j)); end end
```

1.9.4 Bucle while

El bucle `while` se utiliza, por lo general, cuando no se tiene un rango definido sobre el cual se realice la ejecución del ciclo o bien cuando la terminación del mismo viene dada por una condición. La sintaxis más común es:

```
while cond end
```

Donde `cond` es la condición que determina la finalización de ejecución.

Enseguida se muestra un ejemplo muy básico que muestra en pantalla el valor de una variable utilizada como referencia:

```
k=1; while k<10 disp(k); k=k+1; end
```

Lo anterior muestra en consola el valor de `k` mientras esta sea menor a 10, es decir muestra todos los valores enteros en el intervalo $[1, 9]$, es importante notar que la variable `k` debe incrementarse en cada ciclo para que en un momento determinado la condición de finalización se cumpla, de lo contrario se convertiría en un bucle infinito.

Ahora, veamos un ejemplo más práctico. La aproximación de una raíz cuadrada por el método babilónico implica realizar n iteraciones mediante la siguiente expresión:

$$r_n(x) = \frac{1}{2} \left(\frac{x}{r_{n-1}} + r_{n-1} \right)$$

Donde x es el número del cual se calcula la raíz cuadrada. A continuación se muestra el código implementado en MATLAB utilizando un bucle `while`:

```
x=input('Introduzca un número positivo: '); r=x; ra=0; while ra==r ra=r; r=(1/2)*(x/r+r); end  
fprintf('Íz cuadrada de
```

Como se observa, en la variable `ra` se guarda la raíz aproximada calculada en una iteración anterior, de manera que esta sirva como comparación respecto a la nueva raíz calculada, el bucle termina cuando la diferencia entre el valor actual y el anterior es inferior a la tolerancia numérica (`eps`) soportada por MATLAB y por ende pasan a considerarse como valores iguales.

i While-if-break

Es común utilizar el ciclo `while` poniendo un valor verdadero como condición, y usar la sentencia combinada `if-break` como punto de parada, por ejemplo:

```
while true a = randi(10); if a>5 break; end end
```

1.10 Fecha y hora

Primeramente es importante mencionar que MATLAB maneja tres formatos de fechas y hora, a saber:

- Un vector de seis elementos los cuales son: [año, mes, día, hora, minuto, segundo].
- Un valor escalar de coma flotante (tipo `double`), en el cual la parte entera representa la cantidad de días que han transcurrido desde el año cero (calendario gregoriano) y la parte decimal representa la fracción del día transcurrido.
- Una cadena de texto con la forma `'dd-mmm-aaa HH:MM:SS'`.

Para obtener la fecha actual MATLAB proporciona el comando `now`:

```
now ans = 7.3575e+05
```

Lo anterior podría resultar útil para efectos de cálculo pero no es tan significativo para el usuario que está acostumbrado a visualizar la fecha y hora mediante los formatos convencionales; podemos

convertir el valor numérico anterior a una cadena de texto que nos proporcione mayor información a primer vista, para ello se utiliza la función `datestr` como sigue:

```
datestr(now) ans = 03-Jun-2014 17:09:36
```

Además de las anteriores MATLAB dispone de las funciones `datevec` y `clock`, la primera convierte una determinada fecha pasada como argumento en formato string o numérico a un vector de seis elementos como se describió anteriormente, y `clock` devuelve la fecha y hora actual tal como la hace `now` pero como un vector de seis elementos.

Ejemplo 1.1 Calculando área y perímetro de un círculo

El área y perímetro (circunferencia) de un círculo vienen dados por las siguientes ecuaciones:

$$A = \pi r^2$$

$$P = 2\pi r$$

Escriba un programa que reciba como dato de entrada el radio del círculo y que imprima en consola el área y perímetro.

Solución

```
r = input('Radio: '); A = pi*r.^2; P = 2*pi.*r; fprintf('rea :  
Si ejecutamos el programa obtendremos algo como lo siguiente:
```

```
Radio: 5
```

```
Área: 78.5398
```

```
Perímetro: 31.4159
```

Ejemplo 1.2 Clasificando flujos

El número de Reynolds es un parámetro adimensional utilizado en mecánica de fluidos para caracterizar el movimiento de un fluido, usualmente para un flujo interno en tuberías circulares se define como:

$$Re = \frac{vD}{\nu}$$

Donde v es la velocidad del fluido, D el diámetro de la tubería a través de la cual circula el fluido y ν la viscosidad cinemática. La teoría subyacente del número de Reynolds se establece conforme a varias características propias y externas al fluido, pero en este caso vamos a limitarlo al flujo interno en tuberías circulares; siendo así, el número de Reynolds permite caracterizar si un flujo es laminar o turbulento dependiendo de ciertos intervalos establecidos de manera experimental, enseguida se muestran los intervalos de valores y el tipo de flujo en cada caso:

- $Re < 2100$ Flujo laminar
- $2100 \leq Re \leq 3000$ Flujo transitorio
- $Re > 3000$ Flujo turbulento

Basado en lo anterior, escriba un programa cuyos valores de entrada sean la velocidad del fluido, el diámetro de la tubería y la viscosidad cinemática, y que devuelva como variable de salida el tipo de flujo.

Solución:

Vamos a implementar una solución utilizando la estructura de control if-elseif-else, en la cual nuestra variable o dato de comprobación será el número Reynolds.

```
velocidad = input('Velocidad: '); diametro = input('Diámetro: '); viscosidad = input('Viscosidad: ');
```

```
Re = (velocidad*diametro)/(viscosidad);
```

```
if Re<2100 disp('Flujo laminar'); elseif Re >= 2100 Re <= 3000 disp('Flujo transitorio'); else disp('Flujo turbulento'); end
```

¿Parece lo anterior una buena solución?

Está muy bien, pero, y si alguien tiene la brillante idea de colocar cantidades negativas; el programa nunca se "quejará", pero desde luego estaremos entrando en una situación un tanto extraña, seguramente el programa nos mandará que estamos en el caso de un flujo turbulento, aún cuando los datos ingresados sean incorrectos. Luego, lo que se quiere mostrar es que necesitamos trabajar más aún sobre ese código, asegurarnos que los datos ingresados sean los adecuados para realizar el cálculo. Claro está que en un código que necesitamos para nuestros deberes académicos pueda sonar un poco absurdo, pero en el mundo real estas cosas nunca, nunca están de más.

Problemas

1.1 ¿Qué tipo de dato devuelve cada una de las siguientes instrucciones? (Puede verificar utilizando la función `class`).

```
3; true; 3==2; 1,2,3;
```

1.2 ¿Es posible realizar las siguientes operaciones?

```
3+int8(2); true+5; int8(10)+int16(5); 1,2,3+0,1,0; [5,1,-2]+[2 3 0];
```

1.3 Desarrolle un script que le solicite su nombre (utilice la función `input`) y que devuelva un saludo más el nombre ingresado, por ejemplo: Hola Jorge, bienvenido.

1.4 Identifique el error en las siguientes líneas de código:

```
edad=input('Introduzca su edad: ','s'); if edad >= 18 disp('Mayor de edad'); else disp('Menor de edad'); end
```

1.5 Utilizando la escala de calificación del 0 a 10 y siendo 6 la calificación mínima aprobatoria, cree un programa en el cual ingrese una calificación y este le devuelva un mensaje de APROBADO o NO APROBADO en el caso que corresponda.

1.6 El siguiente es un problema clásico en los cursos básicos de programación: escriba un programa que determine si un número ingresado es par o impar.

1.7 Escriba una función llamada `sfibonacci`, que reciba como argumento un entero positivo `n`, y que devuelva un vector con los primeros `n` términos de la sucesión de Fibonacci.

1.8 Desarrolle una función llamada `bucle_test`, que reciba como argumento de entrada un número entero N . Esta función debe recorrer todos los valores enteros en el rango de 1 a N mostrando “ n es divisible por 2”, “ n es divisible por 3”, “ n es divisible por 2 y 3” o “ n no es divisible por 2 o 3” (donde n será el valor actual). Use un bucle `for` para recorrer los valores, la función `rem` para verificar la divisibilidad y `num2str` para convertir cada número en un string y mostrarlo en pantalla. Para verificar cada caso puede utilizar una bifurcación múltiple `if-elseif-else`.¹

1.9 Escriba una función que le permita determinar si un número entero pasado como argumento es primo, en caso de serlo devolverá un valor lógico `true` y un valor `false` en caso contrario. (La función `isprime` de MATLAB realiza la misma operación, pero claro, evite utilizarla en este caso).

¹Danilo Šćepanović. 6.094 Introduction to MATLAB, January IAP 2010. (Massachusetts Institute of Technology: MIT OpenCourseWare), <http://ocw.mit.edu> (Accessed).

Borrador

Borrador

Borrador

Cinemática del sólido rígido

Borrador