Capítulo 1

Conceptos básicos en MATLAB. Ficheros .m

1.1. Operaciones básicas con números complejos

En MATLAB, por defecto, las letras i ó j representan la unidad imaginaria. Obsérvese cómo se introduce el complejo $1+i$

```
» z=1+i
» z=1+j
```

Las operaciones con complejos se realizan igual que con números reales:

Suma	Resta	Multiplicación	División	Potenciación
+	−	*	/	^

Ejemplo 1 *Calcule los siguientes complejos en forma binómica:*

$$(3+5i)(4-i), \qquad \frac{3-i}{4+5i}, \qquad (1+\sqrt{3}i)^3$$

Solución

```
» (3+5i)*(4-i)
» (3-i)/(4+5i)
```

Cuando la parte imaginaria del complejo es una función, como $\sqrt{3}$ o una operación con varios elementos, por ejemplo $1+\frac{1}{3}i$, debe ponerse el signo * entre la parte imaginaria y la unidad imaginaria:

```
» (1+sqrt(3)i)^3        %Devuelve un mensaje de error
??? (1+sqrt(3)i)^3
             |
Error: ")" expected, "identifier" found.
» (1+sqrt(3)*i)^3
```

También es posible trabajar en modo simbólico:

```
» z=sym((1+sqrt(3)*i)^3)
```

Algunas funciones útiles en complejos son las siguientes:

Orden	**Salida**
`real(z)`	Parte real de `z`
`imag(z)`	Parte imaginaria de `z`
`abs(z)`	Módulo de `z`
`conj(z)`	conjugado de `z`
`angle(z)`	devuelve el único argumento de `z` que pertenece a $]-\pi,\pi]$ si `z` es no nulo, y `0` si `z` es `0` (en radianes)

Ejemplo 2 *Calcule el módulo, el argumento, la parte real y la parte imaginaria de* $z = \dfrac{w-\bar{w}}{2i}$ *siendo* $w = 3+5i$.

Solución

```
» w=3+5i
» z=(w-conj(w))/(2i)
» abs(z)              %Módulo
» angle(z)            %Argumento en radianes en ]-pi,pi]
» angle(z)*180/pi     %Argumento en grados
» real(z)             %Parte real
» imag(z)             %Parte imaginaria
```

Si las funciones anteriores actúan sobre una matriz, la salida es otra matriz del mismo tipo que es el resultado de evaluar la función en cada elemento.

Ejemplo 3 *Calcule el módulo y el argumento en grados de cada elemento de la matriz* $A = \begin{pmatrix} 1+i & \sqrt{3}-i \\ -\sqrt{2}-\sqrt{2}i & -1 \end{pmatrix}$

Solución:

```
» A=[1+i sqrt(3)-i; -sqrt(2)-sqrt(2)*i -1]
» abs(A)
» angle(A) %Salida en radianes por defecto
» angle(A)*180/pi %Paso a grados
```

Todas estas funciones pueden actuar tanto sobre variables simbólicas como numéricas, excepto `angle`, que sólo puede actuar sobre variables numéricas.

1.2. Programación

Los problemas de este curso se van a ejecutar en modo *batch*, es decir, ejecutando secuencialmente el conjunto de órdenes que han sido escritas previamente en un fichero ASCII. Esos ficheros reciben el nombre de ficheros `m`, debido a que su nombre tiene extensión m.

Para crear un fichero `.m` se pincha con el ratón `File ->New ->M-File`, o bien se pincha el primer icono de la barra de herramientas. Los dos caminos nos llevan a un editor de texto en el que se escriben las instrucciones que se ejecutarán posteriormente en el área de trabajo.

El signo % permite añadir comentarios, MATLAB obviará todo lo que esté escrito a la derecha de dicho símbolo. Además, si las primeras líneas de un fichero `m` van precedidas de este símbolo, MATLAB considerará estas como la ayuda, y cuando en el área de trabajo tecleemos `help nombre_fichero` nos devolverá este comentario.

Una vez escrito el fichero, nos situamos en la opción `File` del menú del editor, se elige la opción `Save As` y aparece una ventana donde escribiremos el nombre del fichero `nombre_fichero.m`. Las reglas para dar nombre a un fichero son las mismas que las que rigen para las variables, excepto que no distingue entre mayúsculas y minúsculas.

Para ejecutar un fichero `.m` se escribe el nombre de dicho fichero sin extensión en el área de trabajo, y se pulsa $\boxed{\hookleftarrow}$.

Programando ficheros `.m` es frecuente que queramos presentar algún mensaje o dato en la pantalla, e incluso que el propio programa nos pida parte de los datos por pantalla. Esto puede hacerse con las órdenes siguientes:

`input`	Permite introducir un dato desde el teclado La ejecución del programa continúa al pulsar ↩
`input('Cadena de caracteres')`	Realiza la misma operación que la instrucción anterior, pero mostrando en pantalla el mensaje *Cadena de caracteres*, a la espera de recibir datos
`disp('Cadena de caracteres')`	Muestra en pantalla la `Cadena de caracteres`. No espera datos
`disp(x)`	Muestra en pantalla el valor de la variable x
`error('Mensaje de error')`	Muestra en pantalla el `'Mensaje de error'` y corta la ejecución del fichero

Ejemplo 4 *Construya un fichero* `m` *que pida por pantalla un complejo* `z` *y devuelva su argumento en el intervalo* $]-\pi,\pi]$ *expresado en radianes y en grados.*

Solución

```
% Calcula el único argumento del complejo z en el intervalo
% ]-pi+alfa,pi+alpha]

 z=input('Complejo del que quieres calcular el argumento: ');
 argum=angle(z);
 disp('El argumento en radianes es: ')
 disp(argum)
 disp('y en grados es: ')
 disp(argum*180/pi)
```

A menudo, según sean los datos que se utilizan, es necesario tomar una decisión sobre las órdenes a ejecutar, por lo que resultan de gran utilidad los operadores y bucles que se recuerdan a continuación.

1.2.1. Operadores relacionales

MATLAB utiliza los operadores relacionales que se describen en la tabla adjunta:

Operador	Descripción
$<$	Menor
$<=$	Menor o igual
$>$	Mayor
$>=$	Mayor o igual
$==$	Igual
$\sim=$	Distinto

El símbolo $\sim$ se obtiene pulsando simultáneamente Alt Gr + 4 + 4. El 4 debe pulsarse dos veces seguidas sin dejas de pulsar la tecla Alt Gr.

Es importante distinguir el símbolo = de asignación de un valor a una variable, del símbolo == que compara el valor de dos variables.

Los operadores relacionales permiten construir **expresiones lógicas** cuya estructura es

expresion1 OpR expresion2

donde OpR es un relacional y expresion1 y expresion2 son números, matrices (de igual dimensión) o cadenas de caracteres. La respuesta de estas expresiones lógicas es 1 si son verdaderas y 0 cuando son falsas.

```
» x=1+i
x =
1.0000 + 1.0000i
» x==1+i
ans =
    1
» x==1+2i
ans =
    0
```

En la primera línea se asigna a x el valor 1+i, en la siguiente se compara x con 1+i, y nos devuelve un 1 debido a que la proposición lógica es cierta. En la tercera línea se compara la variable x con 1+2i y nos devuelve un 0 debido a que la proposición lógica es falsa.

1.2.2. Estructuras `if-elseif-else-end`

En ocasiones se quiere ejecutar un conjunto de órdenes solo en el caso de que se verifique cierta condición. Esto se consigue con las combinaciones de órdenes `if-end`, `if-else-end` e `if-elseif-else-end`.

La estructura `if-elseif-else-end` se utiliza como sigue:

```
if expresión lógica
     conjunto de órdenes 1
elseif expresión lógica 2
     conjunto de órdenes 2
elseif expresión lógica 3
     conjunto de órdenes 3
                .
                .
                .
```

```
else
     conjunto de órdenes
end
```

El *conjunto de órdenes 1* se ejecuta si la *expresión lógica 1* es verdadera, el *conjunto de órdenes 2* se ejecuta si la *expresión lógica 2* es verdadera, etc. Cuando todas las expresiones lógicas son falsas, se ejecuta el *conjunto de órdenes* que sigue a `else`.

La orden `else` puede aparecer o no. También puede aparecer sólo la combinación `if-end` o la combinación `if-else-end`.

Ejemplo 5 *Escriba un fichero* `m` *que calcule la imagen de un complejo* $z = x + yi$ *por la función:*

$$f(x+yi) = \begin{cases} 0 & si \quad x = y = 0 \\ \arctan\left(\frac{y}{x}\right) & si \quad x > 0 \\ \frac{\pi}{2} & si \quad x = 0\, y > 0 \\ \pi + \arctan\left(\frac{y}{x}\right) & si \quad x < 0 \\ \frac{3\pi}{2} & si \quad x = 0\, y < 0 \end{cases}$$

```
z=input('Introduce el complejo del que quieres calcular la imagen ');
if (real(z)==0)&(imag(z)==0)
   imagen=0;
elseif (real(z)>0)&(imag(z)>=0)
   imagen=atan(imag(z)/real(z));
elseif (real(z)>0)&(imag(z)<=0)
   imagen=atan(imag(z)/real(z))+2*pi;
elseif (real(z)==0)&(imag(z)>0)
   imagen=pi/2;
elseif (real(z)==0)&(imag(z)<0)
   imagen=3*pi/2;
elseif real(z)<0
   imagen=pi+atan(imag(z)/real(z));
end
disp('La imagen en radianes es ')
disp(imagen)
disp('y en grados')
disp(imagen*180/pi)
```

1.2.3. La estructura `for-end`

Hasta ahora las órdenes se ejecutaban de forma secuencial, pero pueden existir procesos en los que un conjunto de órdenes se deban ejecutar varias veces, para ello existen los bucles `for-end`. La sintaxis es la siguiente:

```
for k=x
    conjunto de órdenes
end
```

donde `k` es una variable y `x` es un vector.

Al llegar el programa a la orden `for` la variable `k` toma como valor la primera coordenada del vector `x` y se ejecuta el `conjunto de órdenes`. A continuación `k` toma como valor la segunda coordenada de `x` y se vuelve a ejecutar `conjunto de órdenes`. El bucle se repite tantas veces como coordenadas tenga `x`. Cuando `k` ha recorrido todas las posiciones de `x` el programa seguirá con las órdenes posteriores a `end`. En el caso de que `x` sea una matriz, la variable `k` tomará como valor las distintas columnas de `x`.

1.2.4. Bucles `while-end`

Su sintaxis es la siguiente:

```
while expresión lógica
    conjunto de órdenes
end
```

Hacen que un conjunto de órdenes se ejecute mientras una expresión lógica sea verdadera.

Ejemplo 6 *Escriba un fichero `fibonacci.m` que calcule el primer complejo de la sucesión:*

$$z_n = f(n+1) + f(n)i \quad \text{con} \quad \mathrm{f(n+2) = f(n+1) + f(n)} \quad \mathrm{f(1) = f(0) = 1}$$

que tenga módulo mayor que un valor v que se introducirá por pantalla.

Solución

```
v=input('Introduce la cota inferior del módulo: ');
if v<=1
    error('v debe ser mayor o igual que 1')
end
f_n=1;  %f(n) iniciada para n=0
```

```
f_n1=1; %f(n+1) iniciada para n=0
while abs(f_n1+f_n*i)<v
   f=f_n1;
   f_n1=f_n1+f_n;
   f_n=f;
end
disp('El complejo pedido es: '), disp(f_n1+f_n*i)
```

1.2.5. Ficheros *m* de función

Para definir funciones en MATLAB se utilizar los llamados ficheros *m* de función. Éste tipo de ficheros también ejecutan las órdenes de forma secuencial, pero tienen la particularidad de que tienen argumentos de entrada y de salida. Así, si queremos construir una función del tipo:

$$\begin{array}{rccc} f: & \mathbb{R}^n & \longrightarrow & \mathbb{R}^m \\ & (x_1,\ldots,x_n) & \to & (y_1,\ldots,y_m) \end{array}$$

escribiríamos un fichero *m* cuyo nombre debe coincidir con el de la función definida (en este caso f.m) con las siguientes órdenes:

```
function [y1,y2,...,ym]=f(x1,x2,...,xn)
y1=f1(x1,x2,...,xn)
y2=f2(x1,x2,...,xn)
          .
          .
          .
ym=fm(x1,x2,...,xn)
```

Las variables auxiliares que sean definidas en la programación de la función, habitualmente llamadas variables locales, no pasan a formar parte del espacio de trabajo cuando la función es llamada. Cuando una variable local necesite ser conservada, debe declararse como variable global, esto puede hacerse con la orden `global`.

Ejemplo 7 *Construya una función* `girar.m` *cuyas variables de entrada sean un* $z=(a,b)\in\mathbb{R}^2$ *y un* $\alpha\in\mathbb{R}$ *y cuya salida sea el giro de* z *en sentido antihorario* α *grados.*

Solución:

Recordemos que el producto de dos complejos de módulo r_1 y r_2 y argumento α_1 y α_2 respectivamente es un número de módulo $r_1 r_2$ y argumento $\alpha_1+\alpha_2$. Lo que debemos calcular es el producto de z por un complejo unitario, para no modificar la longitud del vector z, de argumento α, o lo que es lo mismo, multiplicar z por $\cos(\alpha)+\text{sen}(\alpha)i$.

```
function w=girar(z,alfa)
```

```
% calcula un complejo de módulo 1 y argumento alfa grados y
% el producto de éste por z

alfa=alfa*pi/180;
u=cos(alfa)+i*sin(alfa);    % También u=exp(alfa*i)
w=z*u;
```

Ejecutamos en el área de trabajo:

```
» w=girar(1+i,45)
w =
0.0000 + 1.4142i
» whos
Name      Size         Bytes  Class
w         1x1             16  double array (complex)
Grand total is 1 elements using 16 bytes
```

Obsérvese que las variables `alfa`, `z` y `u` no están en memoria, se tratan de variables locales. Si este mismo programa lo realizamos mediante un fichero `m` podremos observar que, a diferencia de antes, todas las variables que se utilizan se guardan en memoria. Escribamos en el fichero `girar2.m` lo siguiente:

```
z=input('Complejo que quieres girar: ');
alfa=input('Ángulo que quieres girar (en grados): ');
alfa=alfa*pi/180;
u=exp(alfa*i);
w=z*u;
disp('El resultado al girarlo es: ')
disp(w)
```

al ejecutarlo

```
» girar2
Complejo que quieres girar: 1+i
Ángulo que quieres girar (en grados): 45
El resultado al girarlo es:
0.0000 + 1.4142i
» whos
Name       Size         Bytes  Class
alfa       1x1              8  double array
u          1x1             16  double array (complex)
w          1x1             16  double array (complex)
z          1x1             16  double array (complex)
Grand total is 4 elements using 56 bytes
```

1.3. Ejercicios

Ejercicio 1.1 *Construya una función* `a=argalfa(z,alfa)` *que calcule el único argumento de z que se encuentra en el intervalo* $]-\pi+\alpha,\pi+\alpha[$. *Si el complejo no tiene argumento en dicho intervalo debe devolver un mensaje de error. Aplica la función a los siguientes datos para comprobar si la has programado correctamente:*

1. $\alpha = 7\pi$, $z = 1+5i$
2. $\alpha = -7\pi$, $z = -3+i$
3. $\alpha = 7\pi$, $z = -3$

Ejercicio 1.2 *Construye una función* `[r,a]=raizpolar(z,n)` *en la que las variables de salida serán, en este orden,* `r`*, el módulo de las raíces n-ésimas de z, y* `a`*, un vector formado por los n argumentos principales de las n raíces n-ésimas de z. Si la entrada* `n` *no es un número natural mayor o igual que* 1*, la función debe devolver un mensaje de error y cortar la ejecución. Aplica la función a los siguientes datos:*

1. $z = -5i$, $n = 2$
2. $z = -16$, $n = 4$
3. $z = 1+i$, $n = 5.6$
4. $z = 4$, $n = 3-i$

Ejercicio 1.3 *Construye una función* `v=raices(z,n)` *que, utilizando la función del ejercicio anterior, devuelva las* `n` *raíces n-ésimas de* `z` *en forma binómica. Aplica la función a los siguientes datos:*

1. $z = -5i$, $n = 2$
2. $z = -16$, $n = 4$
3. $z = 1+i$, $n = 5$
4. $z = 4$, $n = 2$