3 MÓDULO SOBRE PROGRAMACIÓN MATLAB

3 PROPOSICIÓN for ... end

OBJETIVOS

Al terminar éste módulo el lector estará en condiciones de:

- Afirmar sí es posible construir un ciclo usando proposiciones Standard.
- Especificar los tres valores para construir una estructura **for ...end.**
- Afirmar sí números negativos pueden asignársele al valor inicial, al incremento y al valor límite en una estructura **for ... end** .
- Utilizar una estructura **for ...end,** para ejecutar sentencias sucesivas de expresiones aritméticas.
- Utilizar una estructura **for ... end**, usando decremento para realizar cálculos sucesivos de expresiones aritméticas hasta que se cumpla una condición específica.
- Comparar los ciclos **if lógica** con los ciclos **for lógico.**
- Utilizar una sentencia **for ... end** forma anidados.
- Utilizar las estructuras **while … end**, **switch … case … end y try ... catch ... end,** que evalúan un grupo de expresiones matemáticas un número indefinido de veces hasta que cierta condición lógica sea verdadera.

INTRODUCCIÓN

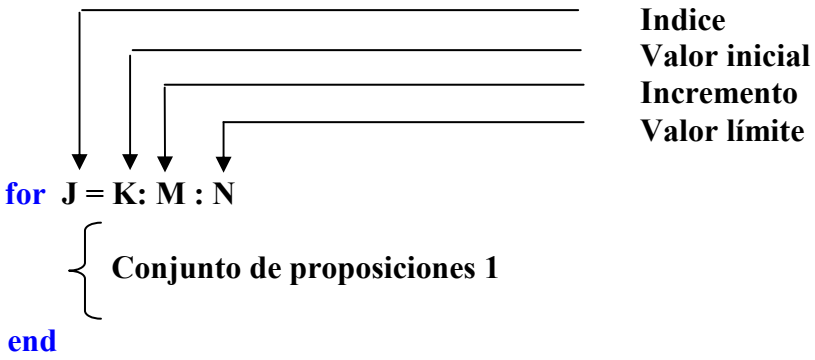
En el módulo 2 hemos aprendido a utilizar la estructuras de decisión **if ...end , if ...else ...end, if ... elseif ... else ...end en forma anidadas,** para seleccionar que sentencias debemos ejecutar en un programa. En éste módulo aprenderemos a ejecutar un conjunto de proposiciones una y otra vez mediante el uso de un ciclo. Utilizaremos un ciclo **for ... end** para ejecutar un número repetido de veces.

Un ciclo **for .... end** es en realidad una forma de escribir una larga lista de sentencias de programa. Como cada grupo de sentencias de la lista haría esencialmente lo mismo, Matlab nos permite definir un grupo de sentencias y que dichas sentencias se ejecuten tantas veces como queramos.

LA PROPOSICION for ... end explícita

La proposición **for ... end** explícita es una proposición ejecutable y provoca que una parte del programa se repita un número especifico de veces, a éste procedimiento se denomina **iterar.** Las proposiciones repetidas durante el procedimiento de iteración forman el ciclo **for ... end**, el cual se inicia y controla a través de la proposición **for** a la que debe seguir de manera inmediata una proposición ejecutable, y terminar con una proposición ejecutable permitida, referenciada por una etiqueta en la proposición **for.** Todas las proposiciones siguientes a la proposición **for ,** incluyendo la proposición terminal , se dicen que están

dentro del rango del **for.** El número de repeticiones del ciclo se establece con los valores inicial y límite de una variable índice entera con incremento automático.
La forma general de la proposición **for ... end** es :



Donde: J indica el nombre de una variable entera llamada **índice.**
K y N representa, respectivamente, los valores enteros inicial y límite que se asignan al índice J.
M señala el incremento (o decremento) del índice (diferente de cero)

El **índice J** siempre es un nombre de variable entera. Los valores inicial y límite K y N pueden constantes enteras, nombres de variables enteras o expresiones aritméticas enteras cuyos variables pueden ser positivos, negativos o ceros. El incremento M puede ser una constante entera, un nombre de variable o una expresión aritmética entera cuyo valor puede ser positivo o negativo, pero no cero.
Cuando el valor inicial K, el incremento M o el límite N son nombres de variables, a estas debe asignársele valor previamente. Sí K, M o N son expresiones aritméticas, todas las variables en las expresiones deben definirse con anterioridad. Los nombres de variables K, M, N que son parámetros de la proposición **for ... end;** sin embargo, estos nombres de variables no deben utilizarse en el ciclo en cualquier forma que pueda cambiar sus valores, por ejemplo:

```
 for  J= K:M:N
        J= 15;
        J= A;
  K=B;
        input(′ K= ′)
 M=10;
    N= D;
end
```

Es esencial que el programador sepa cuantas veces se repetirá un ciclo. En todo proceso iterativo que implique incrementos, el número de repeticiones es uno más que el número que el número de incrementos. El número de incrementos es la diferencia entre el valor límite N y el valor inicial K dividida entre el incremento M, truncado a la forma entera. Por tanto:

$$NR = \frac{N - K}{M} + \quad 1 \qquad \text{y} \qquad M=(N-K)/(NR-1)$$

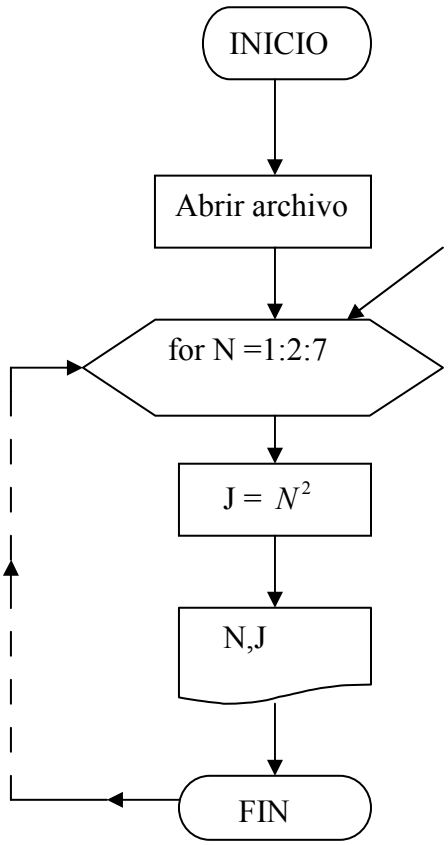
Donde:
NR es el número de repeticiones(o iteraciones) de la proposición **for**
K es el valor inicial que se especifica en la proposición **for**

N es el valor límite que se indica en la proposición **for**
M es el incremento que se especifica en la proposición **for.**

El siguiente programa Ejemplo 3.1 contiene los principales elementos de la proposición **for ... end** y muestra la manera en que se representa el ciclo **for ... end.** Nótese que el símbolo PREPARACION inicia el ciclo **for** y las líneas punteadas definen su rango. El rango del **for** incluye las proposiciones ejecutables J= N^2 y la proposición TERMINAL **disp**([N,J]). Inicialmente al índice N se le asigna un valor de 1. Este valor de N se compara con 7, valor límite del índice, y puesto que N es menor que 7 se inicia el ciclo. El índice N puede utilizarse como una variable entera dentro del ciclo con el valor de N controlado por la proposición **for ... end.** El valor de N^2 se asigna a J, y los valores de N^2 y J (1 y 1) se imprimen de acuerdo con la especificación de la proposición **disp** que puede estar dentro o fuera del ciclo, pero no puede colocarse inmediatamente después de la proposición **for ... end**. Luego el valor de N se incrementa en 2 y se le asigna a N el valor de 3. El valor de N se compara en forma automática con el valor límite 7 que se especificó en la proposición **for** . El valor de N es menor que 7, así que el ciclo se repite. El valor de N^2 (9) se asigna a J y se imprimen los valores de N y J (5 y 25). De nuevo, el índice se incrementa en 2, por lo cual N = 7. Puesto que N es igual pero no mayor que el valor límite, el ciclo se repite otra vez. Cuando el índice se incrementa nuevamente, el valor asignado a N es 9, que es mayor que el valor límite; se satisface el ciclo **for** y la ejecución del programa continúa con la próxima proposición ejecutable después de la proposición terminal del ciclo.

EJEMPLO 3.1 USO DE LA PROPOSICIÓN for ... end

Determine el cuadrado de los números del 1 al 7 con incremento de 2




```
disp( ' N  J  ' )
for  N = 1:2:7
      J= N^2;
disp( [ N,J])
end
```

El salida del programa es:

N	J
1	1
3	9
5	25
7	49

INCREMENTO CON UN CICLO for … end

Al índice de la proposición **for** puede o no asignarse el valor límite a causa de la relación entre el incremento y los valores inicial y límite. Por ejemplo, la proposición:

for I = 1:2:10

Asigna solo valores impares al índice I. El valor límite del índice, el número par 10, nunca se asigna a I. Sí bien este ejemplo de incremento puede parecer diferente de los anteriores, el principio es el mismo. El índice tendrá un valor inicial de 1, el cual es menor que el valor límite, permitiendo el inicio del ciclo. Después de ejecutar las proposiciones en el rango del ciclo, el índice se incrementa a 3, y se compara con el valor límite. Ya que es menor que el valor límite 10, las proposiciones en el rango del ciclo se ejecutan por segunda ocasión. Este procedimiento continúa hasta que el índice pasa del valor 9 al 11. Puesto que 11 rebasa el valor límite 10, se satisface el ciclo **for** y el control se transfiere a la siguiente proposición ejecutable después de la proposición terminal del ciclo. El valor del índice cuando se satisface el ciclo **for** es 11, valor que estará disponible para usos posteriores fuera del rango **for**

Con frecuencia la proposición **for** especifica incrementos de 1 para índice. Cuando el incremento es 1, no es necesario declararlo en forma explícita; la falta de un incremento en la proposición **for** implica un aumento de +1. Por ejemplo:

for J=1:10

Aquí el ciclo se repite para J = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Cuando J se convierte en 11, el programa continúa con la siguiente proposición ejecutable luego de la proposición **end**

Para una proposición **for** N=1:2:10, cuando los valores asignados se sustituyen por los nombres de variables. El valor de N es 1, que se compara con el valor límite 10, y ya que N es menor, se calcula el valor de NCUAD y se imprimen los valores de N y NCUAD (1 y 1). El valor de N se incrementa (N =1+2=3). Puesto que éste valor no excede el valor límite 10, el ciclo se repite y se imprimen los nuevos valores de N y NCUAD (3 y 9). Esta secuencia se repite hasta que el último incremento a N genera un valor de 11 (N = 9+2). Como éste valor es mayor que el valor límite 10, se satisface la proposición **for** y continúa el programa con la primera proposición ejecutable que sigue a la proposición **end**. Esta proposición imprime el valor de N, imprime 11. Esto demuestra que cuando se satisface el ciclo **for,** el último valor asignado al índice, N =11 en éste Ejemplo, está disponible para usos posteriores y es un incremento mayor que el valor límite posible dentro del rango del

ciclo. El programa ejemplo 2.2 muestra cómo pueden obtenerse los resultados con ciclos **for** con incrementos positivos.

EJEMPLO 2.2 INCREMENTO DEL CICLO for ... end

```
 % valor inicial
 INIC = 1;
%  valor del incremento
 INCRE = 2;
% valor limite
LIMITE = 10;
disp(' N   NCUAD ')
       for  N=INIC:INCRE:LIMITE
           NCUAD = N^2;
       disp ([ N,NCUAD])
    end
```

Se visualiza en pantalla:

N	*NCUAD*
1	*1*
3	*9*
5	*25*
7	*49*
9	*81*
11	

DECREMENTO CON UN CICLO for … end

También es posible emplear la proposición **for** con decrementos. Considérese el caso donde una variable debe disminuir de un valor inicial de +7 a un valor límite de –3 en incrementos de –2. Como ilustra el programa Ejemplo 3, esto puede realizarse mediante parámetros negativos en la proposición **for**. Nótese que de nuevo se imprime el valor del índice después de que se completó el ciclo y que es un incremento menor que el valor límite posible dentro del rango del ciclo. Es permisible ejecutar proposiciones dentro de un ciclo **for** antes de que éste quede satisfecho:

for K = 1:15

if X(K) ~=0

dx(K) = x(K)/100;

```
 else
dx(K) = 1/100;
 end
end
```

En éste caso, el ciclo **for** se realiza 15 veces. El ciclo **for** se iterará normalmente mientras que la expresión lógica **if** sea falsa o verdadera ejecutaran estas expresiones aritméticas hasta que termine el ciclo **for.**

EJEMPLO 3.3 DECREMENTO CON UN CICLO for ... end

```
disp(' N   NCUAD ' )
for  N = 7:-2 : -3
     NCUAD = N^2;
    fprinf ( ' %7.3f  %7.3f \n',N,NCUAD)
```

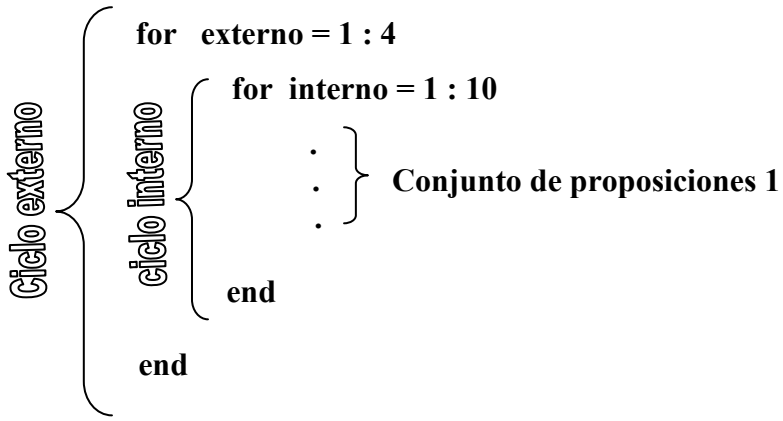
El programa visualiza:

N	NCUAD
7	49
5	25
3	9
1	1
-1	1
-3	9
-5	

CICLOS for ... end ANIDADOS

Al igual que sucede con todas las demás estructuras, pueden anidarse dos o más ciclos **for ... end**, uno dentro de otro. En cualquier momento que su programa necesite repetir un ciclo más de una vez, utilice uno anidado. Considere el ciclo interno como uno que se repetirá más veces que el ciclo externo. Sí el valor límite del ciclo externo es 4 y el valor límite del ciclo interno es 10 , el ciclo interno se ejecutará cuarenta veces, porque la variable interno va de 1 al 10 y la externo va del 1 al 4. Es cómo multiplicar 10x4. El ciclo externo determina cuantas veces se ejecuta el ciclo interno.

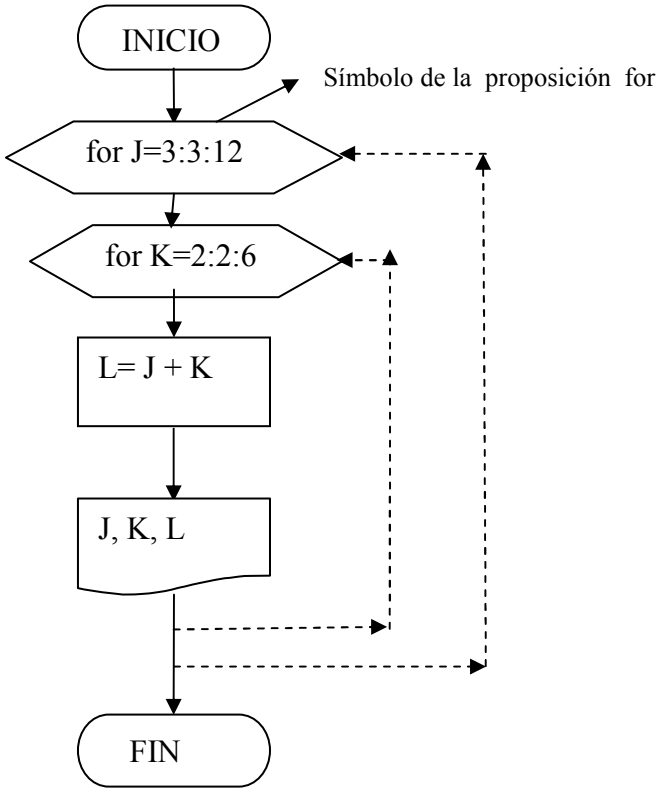
La forma general de un ciclo **for ... end anidado** es:



El siguiente programa ejemplo 3.4 ilustra el uso del ciclo **for .. end** anidado.

EJEMPLO 3.4 USO DEL CICLO for ... end ANIDADO

El ciclo for J= 3:3:12, se llama el ciclo for externo; el ciclo for K=2:2:6, se llama el ciclo **for** interno. Cuando al índice J del for externo se le asigna su valor inicial 3, el ciclo interno se ejecuta asignando al índice K los valores 2, 4, 6. Luego J se incrementa a 6 y el ciclo interno se efectúa una vez más con K = 2, 4, 6 . Esta repetición continúa hasta que se satisfacen los dos ciclos. La proposición de asignación L = J + K sólo indica cómo puede utilizarse la variable índice de una proposición **for ... end** cómo una variable de trabajo dentro del ciclo.




```
disp(' J  K  L' )
for J=3:3:12
      for K=2:2:6
        L= J + K;
        disp([ J  K  L ])
      end
end
```

El programa visualiza:

J	K	L
3	2	5
3	4	7
3	6	9
6	2	8
6	4	10
6	6	12
9	2	11
9	4	13
9	6	15
12	2	14
12	4	16
12	6	18

REGLAS PARA LOS CICLOS for … end

1.- La primera proposición en el rango de un **for ... end** (la proposición que sigue de manera inmediata a la proposición for)debe ser una instrucción ejecutable. Proposición de especificación, cómo por ejemplo la proposición **disp**, no puede estar inmediatamente después de una proposición **for**.

2.- La última proposición en el rango de un **for ... end,** debe ser ejecutable, pero no puede ser un **if** aritmético, un **if** lógico, pero sí otra proposición for ... end.

3.- No se recomienda que ninguna proposición del rango for asigne de alguna manera un valor al índice, a los valores inicial o limite o al incremento . Puede utilizarse el índice en el ciclo en alguna expresión lógica o aritmética, en una proposición de salida o para asignar su valor a otro nombre de variable, mientras que su valor no se altere (el asignado en la proposición for).

4.- En ciclos **for** anidados, el rango del ciclo interno debe estar por completo dentro del rango del ciclo externo. Los ciclos **for** anidados, deben usar una proposición **end** diferente.

5.- Con ciclos for anidados, pueden realizarse transferencias desde un ciclo interno a una proposición dentro del ciclo externo. Siempre se permiten traspasos entre proposiciones dentro del rango de un ciclo.

6.- No se permiten transferencia desde fuera de un ciclo for a una proposición dentro del ciclo. El acceso a un ciclo debe ser siempre a través de la proposición for. Los siguientes programas ejemplos ilustran la aplicación de estas reglas:

EJEMPLO 3.5 CÁLCULO DE LA RAIZ CUADRADA, CÚBICA Y CUARTA DE UN NÚMERO

```
% Ciclos for anidados
clear,clc
disp(   '------------------------------' )
disp(   '      Raiz      Raiz     Raiz  ' )
disp(   ' No  cuadrada  cubica   cuarta ' )
disp(   '------------------------------' )
for N=1:10
     fprintf(' \n %10.0f ',N)
    for R=2:4
        raiz(N)=N^(1/R);
     fprintf(' %8.5f %8.5f %8.5f\n ',raiz(N))
    end
end
```

El programa visualiza:

```
----------------------------------------
      Raíz       Raíz      Raíz
 No  cuadrada  cúbica    cuarta
----------------------------------------

  1   1.00000   1.00000   1.00000
  2   1.41421   1.25992   1.18921
  3   1.73205   1.44225   1.31607
  4   2.00000   1.58740   1.41421
  5   2.23607   1.70998   1.49535
  6   2.44949   1.81712   1.56508
  7   2.64575   1.91293   1.62658
  8   2.82843   2.00000   1.68179
  9   3.00000   2.08008   1.73205
 10   3.16228   2.15443   1.77828
-----------------------------------------
```

EJEMPLO 3.6 ALIMENTACIÓN DE DATOS CON ARREGLO MATRICIAL

El siguiente programa calcula el perímetro y el área de un círculo, cuyo radio, se suministra en forma de un arreglo matricial.

```
clear all,clc
% Ciclo for con arreglo matricial
Radio=[1 4.75 10.33 25.468]';
disp('  ----------------------------------' )
disp('  RESULTADOS FINALES    ' )
disp ('  ----------------------------------' )
disp('    Circun       Area  ')
for J=1:4
   Circun=pi.*Radio(J).*2;
   Area=pi.*Radio(J).^2;
   fprintf('\n %10.3f %10.3f ',Circun,Area)
end
```

La salida del programa es:

```
----------------------------------------
    RESULTADOS FINALES
----------------------------------------
     Circun         Area

      6.283         3.142
     29.845        70.882
     64.905       335.236
    160.020      2037.697
```

EJEMPLO 3.7 USO DEL CICLO for ... end con un incremento H.

```
clear all,clc
Rmin=1;H=0.1;Radio=Rmin+H;N=5;
disp('   i   Radio    Area   Circunferencia ')
for i=1:N
    fprintf('\n %3i',i)
    R(i)=Radio;
    Area(i)=pi*R(i).^2;
    Circun(i)=2*pi*R(i);
    Radio=Radio + H;
    fprintf(' %6.2f  %6.2f  %6.2f \n',R(i),Area(i),Circun(i))
end
```

La salida del programa es:

```
   i    Radio    Area     Circunferencia

   1    1.10     3.80     6.91

   2    1.20     4.52     7.54

   3    1.30     5.31     8.17

   4    1.40     6.16     8.80

   5    1.50     7.07     9.42
```

EJEMPLO 3.8 ALIMENTACIÓN DE DATOS CON VECTOR COLUMNA.

```
clear all,clc
% Este programa calcula el perímetro y el área
% de un circulo para diferentes valores sucesivos
% del diámetro
N=1;
Diam=[1:0.5:3]';
Circun=pi*Diam;
Area=pi*(Diam/2).^2;
N= N+1;
disp( '      Resultados finales    ')
disp('    Diam      Circun    Area ')
disp([Diam Circun Area])
```

La salida del programa es:

```
      Resultados finales
    Diam      Circun    Area
    1.0000    3.1416    0.7854
    1.5000    4.7124    1.7671
    2.0000    6.2832    3.1416
    2.5000    7.8540    4.9087
    3.0000    9.4248    7.0686
```

PROPOSICIÓN while … end

Opuesto al ciclo **for … end**, que evalúa un grupo de proposiciones para un número fijo de veces, un ciclo **while … end**, evalúa un grupo de proposiciones para un número indefinido de veces hasta que satisface alguna condición establecida por el usuario o hasta que una

cierta expresión lógica tome el valor verdadero. La forma genérica de la proposición while … end es:

```
while  condición lógica
        .
        .
        .   } Conjunto de proposiciones
        .
end
```

El conjunto de proposiciones entre while y end son ejecutadas tan pronto cómo todos los elementos en la condición lógica sean verdaderos.
Los siguientes programas ejemplos ilustran la aplicación del ciclo **while … end**:

EJEMPLO 3.9 USO DEL CICLO while … end

```
clear all,clc
i = 1;
  while i<5
     disp( ' Hola ' )
    i= 1+i;
end
```

La salida del programa es:

```
Hola
Hola
Hola
```

EJEMPLO 3.10 USO DEL CICLO while … end

El siguiente programa calcula la longitud de la circunferencia y el área del círculo para diferentes valores del radio.

```
clear all,clc
N=1;
Radio=[10 20 32 40 75]';
  While N <= 5
   Circun= 2*pi*Radio;
   Area= pi*radio.^2;
   fprintf('Cuando el radio =%3.0f',Radio(N))--*ç+
**
fprintf('La circunferencia es %7.3f',Circun(N))
fprintf('y el area es %7.3f\n',Area(N))
N= N+1;
end
```

La salida del programa es:

```
Cuando el radio=10 La circunferencia es 62.832 y el area es 314.159
Cuando el radio=20 La circunferencia es 125.664 y el area es 1256.637
Cuando el radio=32 La circunferencia es 201.062 y el area es 3216.991
Cuando el radio=40 La circunferencia es 251.327 y el area es 5026.548
Cuando el radio=75 La circunferencia es 471.239 y el area es 17671.459
```

EJEMPLO 3.11 USO DEL CICLO while … end

El siguiente programa calcula el ángulo del valor de la función tangente en cada uno de los cuatro cuadrantes.

```
 clear all,clc
A=1.0;B=0.27735;N=1;i=0;
Angulo=atan(A/B)*180/pi;
while N < 5
    i=Angulo+ i;
    fprintf(' Este ángulo está en el cuadrante   %i\n',N)
    fprintf(' y el ángulo =                    %3.4f grados\n',i)
    N= N + 1;
    disp(' ')
end
```

La salida del programa es:

```
 Este ángulo está en el cuadrante   1
 y el ángulo =                    74.4986 grados

 Este ángulo está en el cuadrante   2
 y el ángulo =                    148.9973 grados

 Este ángulo está en el cuadrante   3
 y el ángulo =                    223.4959 grados

 Este ángulo está en el cuadrante   4
 y el ángulo =                    297.9946 grados
```

EJEMPLO 3.12 USO DEL CICLO while … end

El siguiente programa calcula los valores de la funciones seno, coseno y tangente en cada uno de los cuatro cuadrantes.

```
clear all,clc
A=1.0;B=0.27735;N=1;i=0;
Angulo=atan(A/B)*180/pi;
while N < 5
    i=Angulo+ i;
    Seno=sin(i*pi/180);
    Coseno=cos(i*pi/180);
    Tangente=tan(i*pi/180);
    fprintf('  Este ángulo está en el cuadrante   %i\n',N)
    fprintf('  Angulo    Seno     Coseno   Tangente ')
    fprintf('\n  %3.4f  %3.4f  %3.4f  %3.4f \n',i,Seno,Coseno,Tangente)
    N= N + 1;
    disp(' ')
end
```

La salida del programa es:

```
  Este ángulo está en el cuadrante   1
  Angulo    Seno     Coseno   Tangente
  74.4986  0.9636  0.2673  3.6056

  Este ángulo está en el cuadrante   2
  Angulo    Seno     Coseno   Tangente
 148.9973  0.5151  -0.8571  -0.6009
  Este ángulo está en el cuadrante   3
  Angulo    Seno     Coseno   Tangente
  223.4959  -0.6883  -0.7254  0.9488

  Este ángulo está en el cuadrante   4
  Angulo    Seno     Coseno   Tangente
  297.9946  -0.8830  0.4694  -1.8812
```

Además de la finalización normal de un ciclo por la proposición **end,** existen comandos adicionales para interrumpir los cálculos. Estos comandos son listados en la tabla 3.1

Tabla 3.1 Comandos para interrumpir cálculos.

Comandos	Descripción
break	Termina la ejecución de los ciclos **for … end** y **while … end** de Matlab. En los ciclos anidados, break termina sólo el ciclo más interno en que está localizado
return	Usado en funciones de Matlab, return causa un retorno normal de una función desde el punto en el cual la proposición **return** se ejecuta.
error(`'text'`)	Termina la ejecución y visualiza el mensaje conteniendo el texto en la pantalla. Observe que el texto debe encerrarse entre apóstrofos.

CONTRUCCIÓN switch … case … end

La construcción **switch … case … end** proporciona una alternativa para usar las proposiciones if, else y elseif. Algo programado con proposiciones if, también puede ser programado usando construcciones switch … case … end. Se utiliza para comparar si una expresión es igual a una de varios valores posibles. La ventaja de la construcción switch … case … end, es que en algunas situaciones produce el código que es más leíble.
La forma genérica de la construcción **switch … case … end** es:

```
switch  expresión
case  expresión_prueba 1
        .
        .   } Conjunto de proposiciones 1
        .
case {expresión_prueba 2,expresión_prueba 3, …}
        .
        .   } Conjunto de proposiciones 2
        .
Otherwise
        .
        .   } Conjunto de proposiciones 3
        .
end
```

El resultado de **expresión** se compara a su vez con el resultado de cada una de las expresiones de **case.** Si son iguales entonces los comandos siguientes al comando **case** son ejecutados y continúa procesando los comandos siguientes hasta la proposición **end.** Si **expresión** es una cadena de caracteres, entonces se hace una comparación de cadenas con la expresión_prueba de case. Múltiples expresión_prueba pueden listarse separadas por comas y encerradas entre {}. Sólo el primer case se ejecuta. Sí no ocurre ningún **case**, la siguiente proposición del comando otherwise es ejecutada. Sin embargo, el comando otherwise es opcional. Si está ausente, continúa la ejecución con el siguiente comando hasta la proposición end. Sí no hay más casos en cada comando case, la expresión-prueba debe estar en una línea.

EJEMPLO 3.13 USO DE LA ESTRUCTURA switch … case … end

Este programa realiza la operación de **case** de acuerdo a la respuesta de la condición lógica de **switch.**

```
clear all,clc
    paso=2;
    switch paso < 1
          case 1
          DeltaX=0.1;
          case 0
    end
    Deltas
```

La salida del programa es:

```
DeltaX =
    0.1000
```

EJEMPLO 3.14 USO DE LA ESTRUCTURA switch … case … end

Este programa realiza la conversión de unidades de pies a metros para un valor de 6.1.

```
x= 6.1;
unidades = 'pie';
% conversión de pies a metros
     switch unidades
         case{'pulg','inch'}
             y= x*0.0254;
         case{'pie','ft '}
             y=x*0.3048;
         case{'metro','m'}
             y=x/100;
    otherwise
             disp ('unidades desconocidas')
             y=NaN;
end
y
```

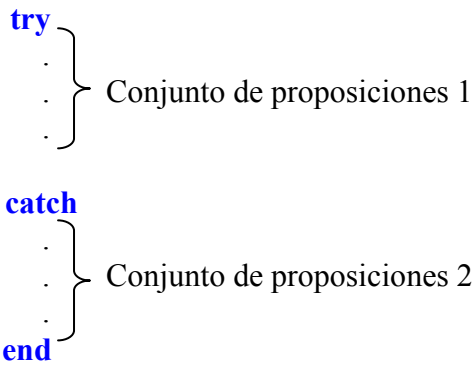
La salida del programa es:

```
y =
     1.8593
```

CONSTRUCCIÓN try … catch … end

Un bloque **try … catch … end** suministra capacidades al usuario para atrapar un error controlado. Con un bloque try … catch, los errores encontrados por Matlab son capturados, permitiéndole al usuario la habilidad para controlar el camino y responda a los errores.

La forma genérica de try … catch … end es:



Cuando el conjunto de proposiciones 1 son ejecutadas, si no se genera error, el control pasa a la proposición **end**. Sin embargo, si aparece un error mientras se ejecuta el conjunto de proposiciones 1, el control pasa inmediatamente a la proposición **catch** y al conjunto de proposiciones 2.

Los siguientes programas ejemplos ilustran su aplicación:

EJEMPLO 3.15 USO DEL BLOQUE try … catch … end

```
clear all,clc
x=ones(4,2);
y=4*eye(2);
   try
        z=x*y;
   catch
        z= NaN;
    disp('x and y no son compatibles')
end
z
```

La salida del programa es:

```
z=
       4     4
       4     4
       4     4
       4     4
```

EJEMPLO 3.16 USO DEL BLOQUE try … catch … end

```
clear all,clc
x= ones(4,2);
y=4*eye(3);
   try
z=x*y;
   catch
       disp(' x and y no son compatibles')
       z= NaN;
end
z
```

La salida del programa es:

```
x and y no son compatibles
z=
   NaN
```

En éste caso, sólo el código después de la proposición **catch** se ejecuta.