1 Condicionales y Bucles

En esta práctica mostraremos las diferentes estructuras de programación de un lenguaje estructurado.

1.1 Condicionales Simples

Permite seleccionar entre dos conjuntos alternativos de instrucciones dependiendo de que se verifique una condición lógica (cuyo resultado es cierto o falso). Su sintaxis es de la forma:

if condición
    Instrucciones que deben ejecutarse si la condición 1 es cierta.
else
    Instrucciones a ejecutar si no se verifica la condición anterior
end

Cuando no hay instrucciones que ejecutar si la condición no se cumple, la sintaxis anterior se reduce a

if condición
    Instrucciones que deben ejecutarse
end

Por el contrario, cuando se encadenan varios bloques alternativos, la sintaxis queda como:

if condición_1
    Instrucciones a ejecutar cuando se verifica la condición 1
elseif condición_2
    Instrucciones a ejecutar cuando no se verifica la condición 1 y sí la condición_2
elseif condición_3
…
else
    Instrucciones a ejecutar cuando no se verifican las condiciones anteriores
end

Las siguientes instrucciones crean una función que da el signo de un número *x*.

```
function s=signo(x)
disp('Inicio de la funcion');
if x>0; s='positivo';
  elseif x<0; s='negativo';
  else; s='nulo';
end
disp('Fin de la funcion');
```

Para evaluar dicha función en el punto 2, se utiliza cualquiera de las siguientes alternativas

```
» x=2;signo(x)
```
o bien
```
» signo(2)
```

Cuando se quiere evaluar una función en varios puntos, se pueden usar dos alternativas

```
» [sqrt(2),sqrt(3),sqrt(4)]
```
o bien
```
» sqrt([2,3,4])
```

Comprobar que sucede con la función signo al evaluarla, de ambas formas, en [-2,0,1].

Se pueden imponer varias condiciones, o condiciones complejas, utilizando los operadores relacionales (condiciones cuyo resultado es cierto o falso) combinados con operadores lógicos (sirven como nexo entre varios relacionales).

Símbolo	Operador Lógico
<	Menor (para complejos sólo a las partes reales)
<=	Menor o igual (sólo afecta a partes reales)
>	Mayor (sólo afecta a partes reales)
>=	Mayor o igual (sólo afecta a partes reales)
x==y	Igualdad (afecta a los números complejos)
x~=y	Desigualdad (afecta a los números complejos)

Símbolo	Operador Relacional
-A	Negación lógica o complementario (NO)
A&B	Conjunción lógica o intersección (Y)
A | B	Disyunción lógica o unión de A y B (O)
xor(A,B)	O exclusivo
any	Algún elemento cumple la condición
all	Todos los elementos cumplen la condición

Ejercicios recomendados:

1. Programar y representar la función

$$f(x)\begin{cases} -2x-1 & x \le -1 \\ x^2 & -1 < x < 1 \\ 2x-1 & x \ge 1 \end{cases}$$

1.2 Condicionales complejos

Permite seleccionar entre varios conjuntos alternativos de instrucciones dependiendo del valor de una expresión. Cada alternativa se denomina "caso" y consta de la instrucción "case", uno o más valores de la expresión y una o varias instrucciones. Su funcionamiento es similar al condicional, excepto que solo evalúa valores. Sintaxis:

switch expresión
  case valor1_de_la_expresión
    instrucciones
  case { valor2_de_la_expresión, valor3_de_la_expresión ,...}
    instrucciones
...
  otherwise
    instrucciones
end

El siguiente ejemplo clasifica una figura cerrada dependiendo del número de lados.

```
function clasifica(numlados)
switch numlados
  case {0,1,2}; disp.('No es un polígono cerrado');
  case 3;disp.('Triángulo');
  case 4;disp.('Cuadrilatero');
  case 5;disp.('Pentágono');
  otherwise; disp.('Poligono')
end
```

1.3 Bucles Simples

Permite repetir un número determinado de veces un conjunto de instrucciones. Su sintaxis es la siguiente:

for var = vector
    Instrucciones que deben ejecutarse
end

El argumento *vector* puede ser efectivamente un vector, en cuyo caso la variable va tomando los valores de las componentes del vector, o una estructura de la forma *inicio : incremento : fin*, en cuyo caso la variable va tomando valores desde *inicio* hasta *fin* con un determinado *incremento*. Si no se indica el valor del incremento, este se toma como unidad. El número de veces que se repite el bucle viene dado por la dimensión del vector o por la expresión

$$max\left(0, \frac{fin - inicio}{incremento} + 1\right).$$

La ejecución del bucle se interrumpe en cualquier momento mediante la instrucción **break**.

Veamos algunos ejemplos, como pueden ser el cálculo de una serie o del factorial.

$\sum_{i=1}^{10} i^2$

```
» serie=0;for i=1:10; serie=serie+i*i;end
» i=1:10;serie=i*i'  % alternativa sin bucles
» i=1:10;serie=sum(i.^2)  % otra alternativa sin bucles
```

$10!$

```
» factorial=1; for i=2:10;factorial=factorial*i; end
» factorial=prod(1:10) % alternativa sin bucles
```

Se pide:

2. Crear el programa primo que comprueba si un valor x es primo

Paso 1: y='cierto';

Paso 2: Para i desde 2 hasta la parte entera de sqrt(x) repetir el siguiente pasoObtener el

Paso 3: si el resto de x entre i es nulo, entonces y='falso'

Con estos datos, la primera línea del archivo debe ser: `function y = primo(x)`

3. Crear la función armónica `function y=armonica(n)` que evalúa $\sum_{k=1}^{n} \frac{1}{k}$ y evaluarla en n=12.

1.4 Bucles condicionales

Permite repetir un conjunto de instrucciones, en tanto se satisfaga una condición lógica. Su sintaxis es la siguiente:

while condición
    Instrucciones que deben ejecutarse mientras la condición sea cierta.
end

El siguiente ejemplo muestra el uso de esta instrucción para sumar términos de la serie armónica mientras su diferencia sea menor que *error*.

```
function y=armonica2(error)
y=0; k=1;
while (1/k)>error;y=y+1/k;k=k+1;end
```

Esta instrucción puede recrearse combinando un bucle *for* y un condicional *if*.

```
function y=armonica3(error)
y=0;
for k=1:10000;
y=y+1/k; if (1/k)<error;break;end
end
```

Se pide:

4. Modificar la función armónica2 anterior para obtener también el número de términos empleados.
5. Crear una función que calcule la norma p de un vector, definida mediante $\|v\|_p = \sqrt[p]{\sum_{k=1}^{n} |v_i|^p}$.

   Comparar los resultados con los obtenidos mediante la función norm.
6. Crear el programa horner[1] que evalúa el valor del polinomio P(x)=p1xn-1 + p2xn-2 + ... + pn-1x + pn en el punto x mediante el algoritmo de Horner (Regla de Rufini). Los coeficientes del polinomio se almacenan en un vector p = [p1, p2, ... , pn-1, pn].

Entrada: vector p = [p_1, p_2, ... , p_{n-1}, p_n] y punto x en que se evalúa

Salida: q, valor de P(x)

Paso 4: Verificar que p es un vector (1 fila o 1 columna)

Paso 5: Obtener el número de elementos de p

Paso 6: Asignar q= p_1

Paso 7: Para k desde 2 hasta n repetir paso 5

Paso 8: q = (q * x) + p_k

Con estos datos, la primera línea del archivo debe ser: `function q = horner(p,x).` En el paso 1 y paso 2 se debe emplear la función `size(p)`, además de los condicionales vistos en la práctica anterior. Comparar los resultados con los obtenidos al usar `polyval(p,x)`

[1] Los pasos para entrar en el editor y escribir la función son: Archivo→New→M-File.

La sintaxis de la función es **function** [variables_salida]= nombre_función (variables_entrada)

2 Anexo: Instrucciones necesarias

2.1 Introducciones de manejo de vectores y matrices

Repasamos aquí las instrucciones ya introducidas cursos anteriores.

vector=[a, b, c, d, …m] Define un vector fila, cuyos elementos son los valores dados.

vector=[a; b; c;, d; …m] Idem con un un vector columna.

variable=[primer_elemento:último_elemento] Define el vector cuyos primeros y último elementos son los especificados, y los elementos intermedios se diferencian en una unidad.

variable=[primer_elemento:incremento:último_elemento] Define el vector cuyos primeros y último elementos son los especificados, y los elementos intermedios se diferencian en la cantidad especificada por el incremento

variable=linspace(primer_elemento,último_elemento,n) Define el vector cuyos primeros y último elementos son los especificados, y que tiene en total n elementos uniformemente espaciados.

variable=logspace(primer_elemento,último_elemento,n) Define el vector cuyos primeros y último elementos son los especificados, y que tiene en total n elementos en escala logarítmica uniformemente espaciados entre sí.

Para definir una matriz en Matlab, basta con introducir entre corchetes todos sus vectores fila separados por punto y coma. Los vectores se pueden introducir separando sus componentes por espacios en blanco o por comas.

2.2 Selección de los elementos de un vector x o matriz A

x(n)	Devuelve el n-ésimo elemento del vector x
x([n,m,p])	Devuelve los elementos del vector x situados en las posiciones n-ésima, m-ésima y p-ésima.
x(n:m)	Devuelve los elementos del vector x situados entre el n-ésimo y el m-ésimo, ambos inclusive
x(n:p:m)	Devuelve los elementos del vector x situados entre el n-ésimo y el m-ésimo, ambos inclusive pero separados de p en p unidades
A(m,n)	Devuelve el elemento (m,n) de la matriz A (fila m y columna n)
A([m, n],[p, q])	Devuelve la submatriz de A formada por la intersección de las filas n-ésima y m-ésima y las columnas p-ésima y q-ésima.
A(n,:)	Devuelve la fila n-ésima de la matriz A
A(:,p)	Devuelve la columna p-ésima de la matriz A
A(:)	Devuelve un vector columna cuyos elementos son las columnas de A situadas por orden
A(:,:)	Devuelve toda la matriz A
[A,B,C]	Devuelve la matriz formada por las submatrices A,B,C,...

2.3 Funciones trigonométricas e hiperbólicas

Trigonométrica	sin(Z)	cos(Z)	tan(Z)	sec(Z)	csc(Z)	cot(Z)
Trig. Inversa	asin(Z)	acos(Z)	atan(Z), atan2(Z)	asec(Z)	acsc(Z)	acot(Z)
Hiperbólica	sinh(Z)	cosh(Z)	tanh(Z)	sech(Z)	csch(Z)	coth(Z)
Hip. Inversa	asinh(Z)	acosh(Z)	atanh(Z)	asech(Z)	acsch(Z)	acoth(Z)

2.4 Funciones exponenciales

exp(Z)	Función exponencial de base e	log(Z)	Función Logaritmo neperiano
sqrt(Z)	Función Raíz cuadrada	log10(Z)	Función Logaritmo decimal

2.5 Funciones específicas de variable numérica

abs(Z)	Módulo o valor absoluto	rem(a,b)	Da el resto de la división entre los reales a y b
angle(Z)	Argumento	fix(x)	Elimina la parte decimal del real x
conj(Z)	Complejo conjugado	floor(x)	Redondea los decimales al menor entero más cercano
imag(Z)	Parte imaginaria	ceil(x)	Redondea los decimales al mayor entero más cercano
real(Z)	Parte real	round(x)	El entero más próximo al real x
sign(x)	Signo del real x (1 si x>0, -1 si x<0)		

2.6 Funciones Matriciales

max(x)	Máximo de los números los elementos de x	gcd(x)	Máximo común divisor de los elementos de x
min(x)	Mínimo de los números los elementos de x	lcm(x)	Mínimo común múltiplo de los elementos de x
size(A)	Filas y columnas de la matriz A	length(x)	Longitud del vector x

2.7 Operaciones de matrices

transpose(A)	Traspuesta de la matriz A	det(A)	Determinante de la matriz A
A'	Traspuesta de la matriz A	trace(A)	Suma de los elementos de la diagonal de A
inv(A)	Inversa de la matriz A (A^{-1})	norm(A)	Norma de A (mayor valor singular de la matriz A)
zeros(m,n)	Crea una matriz con m filas y n columnas cuyos elementos son ceros	ones(m,n)	Crea una matriz con m filas y n columnas cuyos elementos son unos
eye(n)	Crea la matriz identidad de orden n	diag(v)	Crea una matriz diagonal con los elementos de v

2.8 Funciones lógicas vectoriales

isempty	Cierto si el vector es vacío	find(A)	Cierto si encuentra índices de elementos no nulos de A
isequal	Cierto si ambos son iguales	any(A)	Cierto si algún elemento de A es no nulo
isfinite	Cierto si es un valor finito	all(A)	Cierto si todos los elementos de A son no nulos

2.9 Variable especiales

En MATLAb existen variables de uso común, cuyo valor viene ya preasignado.

pi	3'1415926535897...	NaN	Indeterminación (Not a Number, por ejemplo 0/0)
i ó j	Unidad imaginaria (raíz cuadrada de-1)	realmin	El menor número real positivo utilizable
inf	Infinito, por ejemplo 1/0	realmax	El mayor número real positivo utilizable
eps	Menor valor positivo que sumado a la unidad tiene representación diferente a 1.	ans	Variable creada automáticamente para representar el último resultado

2.10 Gráficos bidimensionales (2-D)

plot(X)	Representa los puntos (k,X_k). Si X es una matriz, hace lo mismo para cada columna de la matriz. Si X es un vector complejo, representa Real(X) frente a IMAG(X)..
plot(X,Y)	Representa el conjunto de puntos (X,Y). Si X o Y son matrices, representa por filas o columnas los datos de X frente a los datos de Y, dependiendo si el otro vector es fila o columna. Para valores complejos de X e Y, se ignoran las partes imaginaria.
grid	Sitúa rejillas en los ejes de un gráfico 2-D o ·3-D. La opción grid on coloca las rejillas y greed off las elimina. La opción grid permuta entre on y off
hold	Permite mantener el gráfico existente con todas sus propiedades, de modo que el siguiente gráfico que se realice se sitúe sobre los mismos ejes y se superponga al existente. La opción hold on activa la opción y hold off la elimina. La opción hold permuta entre on y off. Válido para 2-D y 3-D
subplot(m,n,p)	Divide la ventana gráfica en mxn subventanas y coloca el gráfico corriente en la ventana p-ésima, empezando a contar por la parte superior izquierda y de izquierda a derecha hasta acabar la línea, para pasar a la siguiente
plot(X,Y,S)	Gráfica de plot(X,Y) con la opciones definidas en S. Usualmente, S se compone de dos caracteres entre comillas simples, el primero de los cuales fija el color de la línea del gráfico, y el segundo fija el carácter a usar en el graficado. Los valores posibles de colores y caracteres son, respectivamente, los siguientes:

y	amarillo	g	verde	.	puntos	-	sólido
m	magenta	b	azul	o	círculos	:	a puntos
c	cyan	w	blanco	x	x-marcas	-.	guiones y puntos
r	rojo	k	negro	+	signo más	--	semisólidos
				*	estrellas		

2.10.1 Titulos, etiquetas, mallas y textos

title('texto')	Añade el texto como título del gráfico en la parte superior del mismo	text(x,y,'texto')	Sitúa el texto en el punto (x,y)
xlabel('texto')	Sitúa el texto al lado del eje x	ylabel('texto')	Sitúa el texto al lado del eje y
gtex('texto')	Permite situar el texto en un punto seleccionado con el ratón	ginput(n)	Permite recuperar las coordenadas de n puntos de un grafico mediante ratón