NOTAS SOBRE PROGRAMACIÓN EN MATLAB

por

Ángel Manuel Ramos del Olmo
Departamento de Matemática Aplicada
Universidad Complutense de Madrid

26 de septiembre de 2002

Índice

1 Introducción.

En estas notas, daremos unas ideas básicas sobre programación. Estas ideas pueden aplicarse a la mayoría de los lenguajes de programación, pero nosotros las desarrollaremos en MATLAB. Este lenguaje está pensado para el cálculo numérico, por lo que está diseñado para desarrollar programas con este objetivo de manera rápida y sencilla, dejando de lado alguna de las características de otros lenguajes de programación que no son de mucha importancia en el cálculo numérico.

Un *lenguaje de programación* es un lenguaje que permite la comunicación entre el programador y la computadora. Como en todo lenguaje, exitirá una sintaxis preestablecida que el programador debe seguir a la hora de escribir las *instrucciones* que quiere que la computadora ejecute.

Un *programa* es un conjunto de instrucciones, escritas en un determinado lenguaje de programación. En MATLAB se pueden escribir estas instrucciones una a una de manera interactiva en la ventana de MATLAB o escribirlar en un fichero con extensión $*$.m y posteriormente llamar a este fichero (sin la extensión) desde la ventana interactiva de MATLAB.

Programar es la acción de crear programas en un determinado lenguaje de programación.

La Programación es el área de la Tecnología que se ocupa del estudio y desarrollo del *arte* de programar.

En MATLAB los elementos básicos de programación son los vectores y matrices que hemos estudiado previamente. Estos elementos básicos se combinarán mediante las operaciones con vectores y matrices estudiadas previamente para crear instrucciones.

Como regla general, durante la ejecución de un programa, las instru-

cciones se van realizando de forma consecutiva en el orden que las hemos escrito. Obviamente, si esta regla no pudiera romperse, la cantidad de instrucciones que deberiamos escribir en un programa para pedir a la computadora que realizara alguna tarea con, por ejemplo, muchas operaciones aritméticas, sería tan grande que no nos merecería la pena escribir ese programa pues acabariamos antes utilizando la calculadora (piensesé por ejemplo en un programa para sumar los 1000 primeros naturales).

La secuencia consecutiva de ejecución de ordenes de un programa puede romperse de diversas maneras: las distintas posibilidades que existen las podemos agrupar en dos estructuras algorítmicas básicas:

- *Composición alternativa*: Se comparan dos valores y, según el resultado, la secuencia del programa sigue por un camino u otro.
- *Composiciones iterativas o Bucles*: Se repiten una serie de instrucciones un número (determinado a priori o no) de veces.

2 Estructuras algorítmicas básicas.

En esta sección estudiamos las estructuras algorítmicas básicas. Presentamos, en primer lugar, la sintaxis de estas estructuras en seudocódigo (veasé la sección 5.2), válidas para la mayoría de lenguajes de programación, y su "traducción" a código MATLAB. A continuación mostramos diversos ejemplos en seudocódigo y en código MATLAB.

2.1 Composición alternativa. El bloque IF.

Sintaxis 1:

- Seudocódigo:

Si condicion
  instrucciones
fsi

- Código MATLAB:

`if` *condicion*
  instrucciones
`end`

Ejemplo 1: Hallar el máximo de dos números.

- Seudocódigo:

leer a y b
si a<b
  a=b
fsi
Escribir El máximo es: a

- Código MATLAB:

```
a=input('Introduzca un número:  ');
b=input('Introduzca otro número:  ');
if a<b
   a=b;
end
disp('El máximo es:  ')
disp(a)
```

Sintaxis 2:

- Seudocódigo:

Si condicion
  instrucciones
si no
  instrucciones
fsi

- Código MATLAB:

`if` *condicion*
  instrucciones
`else`
  instrucciones
`end`

Ejemplo 2: Hallar el máximo de dos números y sumarle dos veces el mínimo.

- Seudocódigo:

leer a y b
Si a<b
  M=b
  m=a
si no
  M=a
  m=b
fsi
Escribir Resultado = M+2m

- Código MATLAB:

```
a=input('Introduzca un número:  ');
b=input('Introduzca otro número:  ');
if a<b
   M=b;
   m=a;
else
   M=a;
   m=b;
end
disp('El máximo más dos veces el mínimo es:  ')
disp(M+2*m)
```

Sintaxis 3:

- Seudocódigo:

Si condicion
    instrucciones
si no, si condicion
    instrucciones
si no, si condicion
    instrucciones
etc...
si no
    instrucciones
fsi

- Código MATLAB:

```
if condicion
   instrucciones
elseif condicion
   instrucciones
elseif condicion
   instrucciones
etc...
else
   instrucciones
end
```

Ejemplo 3: Hallar el valor de

$$f(x) = \begin{cases} -x & \text{si } x < 1 \\ 1 & \text{si } -1 \leq x \leq 1 \\ x & \text{si } x > 1 \end{cases}$$

para cualquier $x \in \mathbb{R}$.

- Seudocódigo:

```
leer x
Si x<1
   f=-x
si no, si x>1
   f=x
si no
   f=1
fsi
Escribir f(x) = f
```

- Código MATLAB:

```
x=input('Introduzca un número x:  ');
if x<-1
   f=-x;
elseif x>1;
   f=x;
else
   f=1
end
disp('f(x)= ')
disp(f)
```

Ejercicio 1 Escribir en seudocódigo y en código MATLAB un programa que pida por pantalla tres números enteros y muestre cuál es el menor, el mediano y el mayor.

Ejercicio 2 Escribir en seudocódigo y en código MATLAB un programa que calcule el área de algunas figuras geométricas: Triángulo, cuadrado, rectángulo o circunferencia. Para ello:

1. Pide el tipo de figura de la que se va a calcular su área, escribiendo mensajes claros para el usuario.
2. Pide los datos necesarios según el tipo de figura.
3. Muestra el resultado por pantalla.

2.2 Bucles.

El bloque FOR.

Sintaxis:

- Seudocódigo:

  Para variable=expresión
  instrucciones
  fpara

  donde *expresión* es del tipo $r, \cdots, s$ y denota que la variable en la primera iteración tiene el valor r y en cada iteración se incrementa en 1 unidad (esto se puede variar), hasta que llega al valor de s.

- Código MATLAB:

  `for` *variable=expresión*
  instrucciones
  `end`

  donde *expresión* es del tipo `r:i:s` y denota que la variable en la primera iteración tiene el valor `r` y en cada iteración se incrementa en `i` unidades, hasta que llega al valor de `s`. Si se omite en la expresión anterior la parte `i:`, MATLAB utiliza por defecto `i=1`.

Ejemplo: Calcular la suma de los n primeros términos de la sucesión $1, 2x, 3x^2, 4x^3, \ ...$

- Seudocódigo:

leer x n
suma =1
Para $i = 2, \cdots, n$
  suma=suma +$ix^{(i-1)}$
fpara
Escribir Resultado = suma

- Código MATLAB:

```
n=input('Cuántos términos quieres sumar?  ');
x=input('Dame el valor del numero x ');
suma=1;
for i=2:n
   suma=suma+i*x^(i-1);
end
disp('El valor pedido es ')
disp(suma)
```

El bloque WHILE.

Sintaxis:

- Seudocódigo:

Mientras condición
  instrucciones
fmientras

- Código MATLAB:

```
while condición
   instrucciones
end
```

Ejemplo: Escribir un número natural en una base dada (menor que diez).

- Seudocódigo:

leer n base
i=1
Mientras n>0
c(i)= resto de n/base
n=parte entera de n/base
i=i+1
fmientras
Escribir Resultado = c(i-1)c(i-2)$\cdots$c(1)

- Código MATLAB:

```
n=input('Dame el número a cambiar de base ');
base=input('¿En qué base quieres expresarlo?  ');
i=1;
while n>0
   c(i)=rem(n,base); % Resto de n/base
   n=fix(n/base); % Parte entera de n/base
   i=i+1;
end
disp('La expresión en la base dada es:')
i=i-1;
disp(c(i:-1:1))
```

Ejercicio 3 Escribir en seudocódigo y en código MATLAB un programa que

1. Lea una secuencia de números enteros cuyo final se señála con un 0.

2. Muestre por pantalla los ordinales de los lugares que ocupa el número 10 en dicha sucesión.

3. Si el número 10 no aparece, escriba un mensaje advirtiendo de ello.

4. Muestre por pantalla la suma de todos los números introducidos.

3 Entrada-Salida de datos.

3.1 Salida de datos.

Hay dos tipos principales de salida de datos: por pantalla o en ficheros.

Ya hemos visto en los ejemplos anteriores algunas de las maneras de sacar datos por pantalla. El programador de MATLAB va descubriendo poco a poco (con la práctica) las diversas opciones de salida de datos y elije en cada caso la que le parece más conveniente.

Para guardar datos en un fichero se utiliza la siguiente instrucción:

```
save [datos] variable_1 variable_2, ...  [-ascii]
```

Al ejecutar la instrucción anterior, se guardarán los datos de las variables `variable_1, variable_2, ...` en el fichero datos.mat en forma binaria, si no se incluye la opción `-ascii`, o en el fichero datos en formato ascii, si se incluye la opción `-ascii`. Obviamente los nombres `datos, variable_1, variable_2, ..` son a elegir por el programador en cada caso. Si se omite el nombre del fichero, MATLAB toma por defecto el fichero matlab.mat

3.2 Entrada de datos.

Hay dos tipos principales de entrada de datos: por teclado o por ficheros.

Ya hemos visto en los ejemplos anteriores algunas de las maneras de introducir datos por teclado. El programador de MATLAB va descubriendo poco a poco (con la práctica) las diversas opciones de entrada de datos y elije en cada caso la que le parece más conveniente.

Para leer datos en un fichero se utiliza la siguiente instrucción:

```
load [datos[.xyz]] [variable_1 variable_2, ...]  [-ascii]
```

Al ejecutar la instrucción anterior, se cargarán los datos de las variables `variable_1, variable_2, ...` guardados en el fichero datos.mat en forma binaria, si no se incluye la opción `-ascii` o el nombre del fichero

no tiene extensión, o en el fichero datos en formato ascii, si se incluye la opción `-ascii`, o en el fichero datos.xyz en formato ascii si se incluye una extensión arbitraria xyz distinta de mat. Si no se incluyen las variables que se quieren leer del fichero se leeran, por defecto, todas las variables disponibles. Obviamente los nombres `datos[.xyz]`, `variable_1, variable_2, ...` son a elegir por el programador en cada caso. Si se omite el nombre del fichero, MATLAB toma por defecto el fichero matlab.mat

4 Funciones.

Al igual que MATLAB tiene funciones predefinidas que podemos utilizar (como `sin`, `cos`, etc.), nosotros podemos construir nuestras propias funciones. Si queremos construir una función que se llame, por ejemplo, `fun`, creamos un fichero en el que escribimos las instrucciones de esta función. La primera instrucción de ese fichero debe ser:

`function [`*argumentos de salida*`]=fun(`*argumentos de entrada*`)`

Es conveniente que el fichero que contenga la función se llame como ella. De este modo, la función anterior debería guardarse en el fichero `fun.m`. Por ejemplo, si se desea programar una función `media` que calcule la media aritmética de dos números reales, basta escribir un fichero `media.m` cuyo contenido sea:

```
function m=media(a,b)
% Cálculo de la media aritmética de dos números reales
m=(a+b)/2;
```

Ejercicio 4 Hacer una función MATLAB que, dados tres números a, b, c halle el valor de $b^2 - 4ac$.

Ejercicio 5 Hacer una función MATLAB que, dados tres números a, b, c, devuelva la media aritmética del mayor y el menor.

5 Diseño de programas.

En cuanto intentemos escribir programas de mayor envergadura de los ejemplos que hemos visto hasta ahora nos daremos cuenta que, en general, no se puede (o no se debe) escribir directamente el programa en el ordenador si se quiere obtener un resultado satisfactorio. La razón es que, el programa seguirá caminos distintos según los valores que obtengan sus variables, etc.., por lo que hay que pensar de antemano todas las posibilidades y estructurar bien el programa. Es la misma razón por la cual, cuando queremos hacer un viaje en coche a un lugar al que nunca hemos ido, conviene antes de salir, echar un vistazo al mapa y preparar la ruta a seguir. Además, tener bien estructurada la idea general del programa nos ayudará en el futuro a una posible revisión o ampliación de éste.

En general, a la hora de preparar un programa se deberían seguir los siguientes pasos:

1. Tener claro el problema que se quiere resolver o la tarea que se quiere llevar a cabo.

2. Descripción de los pasos a seguir para resolver el problema o llevar a cabo la tarea. El resultado de este proceso es un *algoritmo*.

3. Transformación del algoritmo en un programa, escrito en cierto lenguaje de programación, que el ordenador es capaz de interpretar.

Aunque parezca una cosa trivial, el primero de los puntos es de gran importancia, pues es donde nos damos cuenta si el problema que nos planteamos es resoluble a través de un programa o no lo es.

Hay que señalar que para resolver un problema o llevar a cabo una cierta tarea con la computadora existen infinidad de algoritmos. Sin embargo, la labor del programador es construir un buen algoritmo. Es en este punto donde se distinguen los programadores buenos de los menos buenos.

Una vez que se tiene el algoritmo, la tarea de transformarlo en un programa no suele presentar dificultad, una vez que se tiene una cierta soltura y habilidad en el lenguaje de programación empleado.

Para la preparación de algoritmos hay diversas técnicas que los programadores utilizan de manera independiente o de forma combinada. Las técnicas más utilizadas son la preparación de organigramas y los seudocódigos.

5.1 Organigramas.

Un organigrama es el dibujo mediante símbolos de un diagrama de flujo, que representa, en mayor o menor grado, las posibles rutas en la ejecución de un programa. Es decir el organigrama es el *mapa* de nuestro programa. De la misma manera que un mapa puede tener distintas escalas, un organigrama puede estar más o menos detallado. En este sentido, un exceso de detalle puede hacerlo demasiado complicado de seguir, perdiendo de este modo su interés y principal ventaja. De hecho, la tendencia al excesivo detalle en la elaboración de un organigrama es lo que ha hecho perder su popularidad entre gran parte de los programadores, por lo que nosotros tampoco lo utilizaremos. Lo mejor es representar simplemente una idea general de las principales tareas

del programa, sin entrar en los detalles de estas.

Los símbolos más comunmente usados en el diseño de organigramas son los representados en la Figura 1

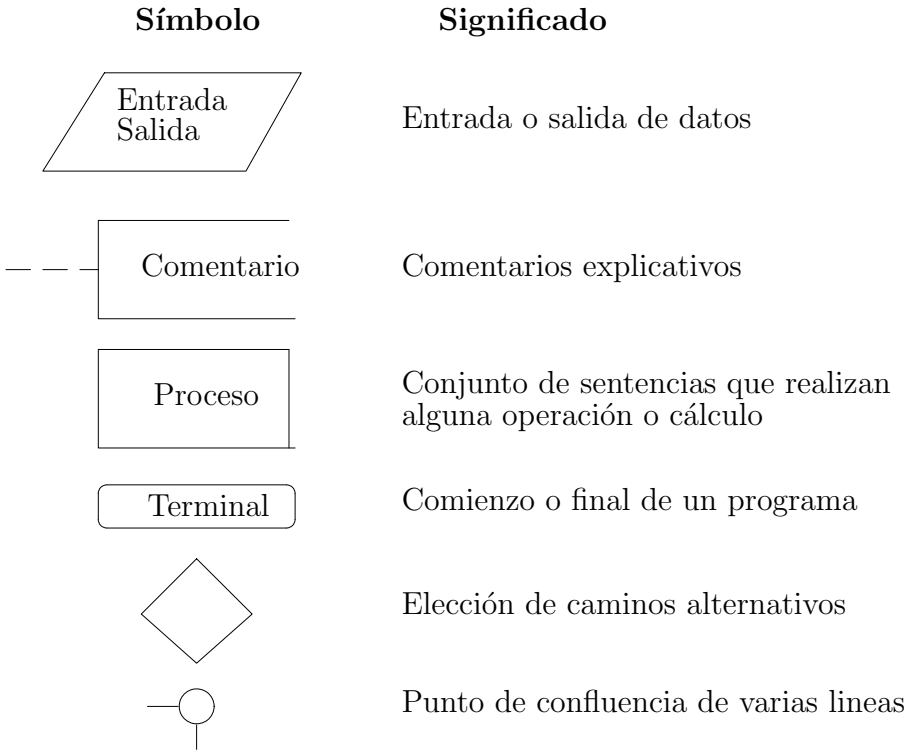

Figura 1: Símbolos utilizados en los organigramas.

Ejemplo 6 Un organigrama para el cálculo del m.c.d. de dos números, por el Algoritmo de Euclides, podría ser el presentado en la Figura 2

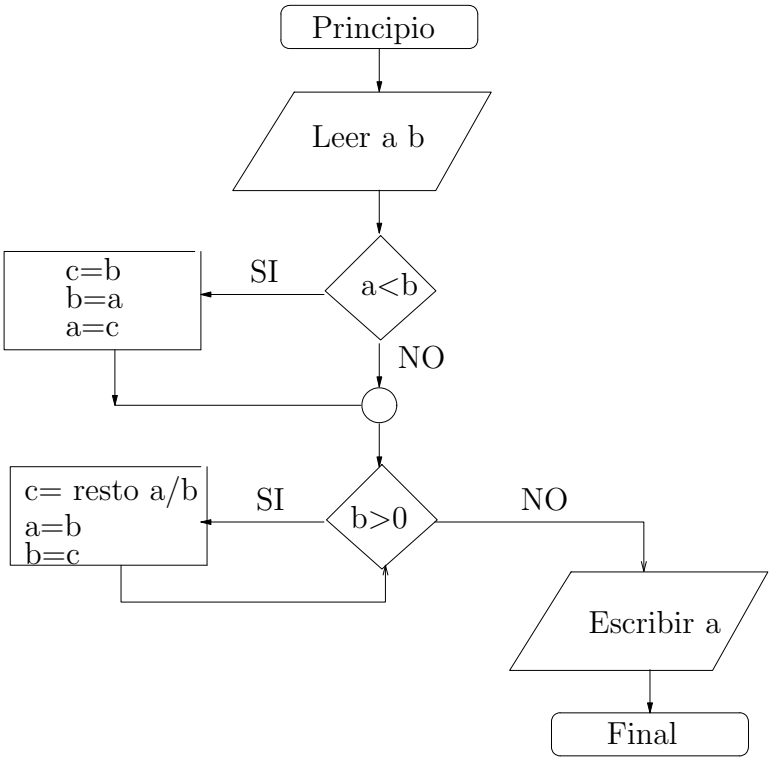

Figura 2: Organigrama del Algoritmo de Euclides.

5.2 Seudocódigo de un algoritmo.

En este método no se requiere hacer ningun dibujo. Consiste en escribir de forma más o menos detallada en un papel, los pasos a seguir por el programa. No existen reglas claras sobre la manera de hacerlos y cada programador adopta sus propios métodos. En general se adopta terminología parecida al lenguaje de programación utilizado y, al igual que lo dicho en los organigramas, normalmente se especifica tan solo las principales tareas del programa, sin entrar en el detalle de cada una de estas tareas.

Ejemplo 7 Un seudocódigo para el cálculo del m.c.d. de dos números, por el Algoritmo de Euclides, podría ser:

leer a b
Si a<b
    c=b
    b=a
    a=c
fsi
Mientras b>0
    c=resto de a/b
    a=b
    b=c
fmientras
Escribir m.c.d = a

Comparese este seudocódigo con el organigrama de la Figura 2.

Ejercicio 8 Escribir una función en código MATLAB que implemente el algoritmo de Euclides desarrollado en el organigrama de la Figura 2 y en el seudocódigo anterior.

Solución.

```
function m=euclides(a,b)
% Cálculo del máximo común divisor de dos números
% naturales mediante el algoritmo de Euclides
if a<b
   c=b;
   b=a;
   a=c;
end
while b>0
c=rem(a,b);
   a=b;
   b=c;
end
m=a;
```

5.3 Diseño de programas de forma modular.

Cuando un programa es muy grande no tiene sentido escribirlo en un solo fichero. Lo más razonable es, ayudandonos del organigrama o el seudocódigo del algoritmo utilizado, dividir el programa en subtareas, que programaremos como funciones o subprogramas independientes, a los que "llamaremos" posteriormente desde nuestro programa. Esto, además de facilitar la revisión y la localización de errores de programación, nos permitirá utilizar las funciones y subprogramas construidos, como parte de otros programas, sin necesidad de reescribir de nuevo el código. En este sentido, es importante en programación la idea de programar *de lo general a lo particular*, con el objetivo de poder utilizar las funciones y subprogramas preparados (de manera general), en otros

programas (con objetivo más particular).

Ejercicio 9 Hacer un programa que calcule las raices de una ecuación de segundo grado, utilizando la función construida en el Ejercicio 5. Además, si las raices son reales, que especifique cuál es la de menor y mayor valor absoluto. Para ello diseñar previamente el algoritmo a través de un seudocódigo.