

Dpto. de Economía Cuantitativa
Universidad Complutense de Madrid
GNU Octave y Matlab

Scripts, controles de flujo y funciones

Marcos Bujosa

© 2005–2007 Marcos Bujosa marcos.bujosa@ccee.ucm.es
Actualizado el: 21 de junio de 2007

Versión 1.2



Algunos derechos reservados. Esta obra está bajo una licencia Reconocimiento-CompartirIgual de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/2.5/es/> o envíe una carta a Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Puede encontrar la última versión de este material en <http://www.ucm.es/info/ecocuan/mbb/index.html#octave>

Índice

Índice	1
1. Guiones (scripts) en GNU Octave (o Matlab)	1
2. Controles de flujo	2
2.1. Bucle <code>for</code>	2
2.2. Bucle <code>while</code>	3
2.3. Condiciones <code>if-then-else</code>	4
3. Funciones	4

1. Guiones (scripts) en GNU Octave (o Matlab)

Si teclea `pwd`, GNU Octave le indicará cual es el directorio de trabajo donde está operando GNU Octave. Con el comando `cd` puede variar dicho directorio de trabajo. Por ejemplo `cd c:\`.

GNU Octave puede ejecutar consecutivamente las órdenes escritas en un fichero de texto (ASCII) con extensión `.m`

El fichero debe estar situado en directorio de trabajo donde esté operando GNU Octave. O bien en el `path` de GNU Octave (o Matlab). El `path` es el conjunto de directorios donde GNU Octave (o Matlab) buscan programas o guiones (scripts) para ser ejecutados.

Teclee `path` para comprobar que directorios están en el `path`.

El comando `addpath` añade los directorios que se especifiquen al `path`; por ejemplo `addpath('E:\loquesea\loquesea\')`. Nótese las comillas. Si quisiéramos eliminar dicho directorio del `path` deberíamos teclear `rmpath('E:\loquesea\loquesea\')`.

Edite sus guiones (scripts) con un editor que respete el formato ASCII y la extensión `*.m` (por ejemplo, si trabaja en el entorno windows, el editor `wordpad`; o mejor aún `WinEdit`).

```

↑ directorio de trabajo 1
  ■ pwd
  ■ cd
  ■ path
  ■ addpath
    addpath('E:\loquesea\loquesea\')
    (Octave en Windows)
    addpath('/cygdrive/E/loquesea/loquesea/')
  ■ rmpath
    rmpath('E:\loquesea\loquesea\')
    (Octave en Windows)
    rmpath('/cygdrive/E/loquesea/loquesea/')

```

A continuación veamos un ejemplo de guión con instrucciones para GNU Octave.

```

↑ Guiones 2
Codigo guion1.m octave
1 n = 0:pi/100:2*pi; % crea un vector
2 y = cos(2*pi*n); % crea el vector y
3 plot(n,y); % representa el vector y frente a n

tecleando octave> guion1...

```

2. Controles de flujo

Los controles de flujo permiten regular la ejecución de órdenes. La regulación se basa en imponer condiciones para la ejecución de un conjunto de órdenes.

Estos controles son tremendamente útiles, aunque pueden “ralentizar” la ejecución del programa. Por ello veremos más tarde que es recomendable evitar bucles siempre que esto sea posible.

2.1. Bucle for

Los bucles for permiten repetir un conjunto de órdenes un número fijo de veces.

```

↑ Bucle for 3
Codigo EjemploFor1.m octave
1 for x = matriz
2     conjunto de ordenes
3 end

1. x toma como valor la primera columna de matriz
2. Se ejecuta el conjunto_de_ordenes
3. x toma como valor la siguiente columna de matriz
4. Se ejecuta el conjunto_de_ordenes
5. Vuelve al punto 3
6. ... sigue iterando hasta que ...
7. x toma como valor la última columna de matriz
8. Se ejecuta por última vez el conjunto_de_ordenes
9. finaliza el bucle

```

Por tanto, el `conjunto_de_ordenes` se ejecuta tantas veces como número de columnas tiene la variable `matriz`.

↑ Bucle for: Ejemplo contador 4

Codigo `EjemploFor2.m` octave

```

1 contador=0
2 for x = [0 -1 1;4 5 6]
3     contador=contador+1
4 end

```

¿Qué hace este bucle?...

octave> `EjemploFor2`

↑ Bucle for: Ejemplo contador 5

Codigo `EjemploFor3.m` octave

```

1 for i = 1:3
2     i
3 end

```

¿Y qué hace este otro bucle?

EJERCICIO 1.

- Genere una matriz de dimensiones 2×3 y con distribución Uniforme; y cree un bucle que muestre consecutivamente cada una de sus **columnas**.
- Genere una matriz de dimensiones 3×4 y con distribución Uniforme; y cree un bucle que muestre consecutivamente cada una de sus **filas**.
- Genere una matriz de dimensiones 3×4 y con distribución Uniforme; y cree un bucle que muestre consecutivamente las columnas **en el siguiente orden**: $[2 \ 4 \ 1]$ (por tanto sin mostrar la tercera columna).
- Genere un bucle que sume los elementos de un vector, por ejemplo del vector fila $v = [1 \ 5 \ 4 \ 3 \ 2]$

Los bucles se pueden anidar unos dentro de otros. El siguiente ejemplo calcula la suma de los elementos de una matriz.

(`[f,c]=size(x)` asigna a `f` en número de filas de `x`; y a `c` el número de columnas).

(`m=max(f,c)` asigna a `m` el mayor de los números `f`, `c`)

↑ Bucle for: Ejemplo suma 6

Codigo `EjemploFor4.m` octave

```

1 M=[1 2 3;-2 -4 -6;2 3 1] % creamos la matriz
2 [f,c]=size(M)           % f #filas; c #columnas
3 s=0;                    % suma inicial
4 for i = 1:f              % recorre filas
5     for j=1:c             % recorre columnas (en cada fila)
6         s=s+M(i,j); % agrega cada elemento a s
7     end
8 end
9 s                        % muestra suma total

```

2.2. Bucle while

Los bucles `while` permiten repetir un conjunto de órdenes mientras una condición fijada se cumpla. Cuando dicha condición sea falsa (cuando se deje de cumplir), el conjunto de órdenes deja de ser ejecutado (el bucle se para).

↑ Bucle while 7

Codigo `EjemploWhile1.m` octave

```

1 while condicion
2     conjunto de ordenes
3 end

```

1. Comprueba si `condicion` se cumple
2. Se ejecuta el `conjunto_de_ordenes`
3. Vuelve al punto 1
4. ...sigue iterando hasta que ...
5. Cuando `condicion` se incumple finaliza el bucle (**¡Ojo!**)

↑ Bucle while: Ejemplo registro de suceso 8

Codigo `EjemploWhile2.m` octave

```

1 x=0;
2 contador=0;
3 while x<=0.95
4     contador=contador+1;
5     x=rand(1)
6 end
7 contador

```

octave> `EjemploWhile2`

2.3. Condiciones if-then-else

Las condiciones `if-then-else` ejecutan un conjunto de órdenes una vez (por tanto no es un bucle) siempre que la condición fijada se cumpla. Cuando dicha condición sea falsa, el conjunto de órdenes no se ejecuta.

↑ Ejemplo if else 9

Codigo `EjemploIfElse1.m` octave

```

1 x=randn(1)
2 if x<-1
3     disp('menor que -1')
4 elseif x>1
5     disp('mayor que 1')
6 else
7     disp('entre -1 y 1')
8 end

```

octave> `EjemploIfElse1`

3. Funciones

Podemos convertir los guiones (scripts) en funciones de GNU Octave y Matlab. La estructura de una función es como sigue:

↑
Ejemplo de función
10

Codigo EjemploFun.m
octave

```

1 function [out1, out2, ...] = EjemploFun (in1, in2, ...)
2
3     conjunto de ordenes que calculan
4     out1, out2,...
5     empleando in1, in2,...

```

donde

- los argumentos de salida entre [] y separados por ,
- el nombre de la función foo debe grabarse en fichero foo.m
- los argumentos de entrada entre () y separados por ,

↑
Ejemplo de función: suma
11

Codigo SumaMatriz.m
octave

```

1 function [s] = SumaMatriz (M)
2
3 [f,c]=size(M)           % f #filas; c #columnas
4 s=0;                   % suma inicial
5 for i = 1:f             % recorre filas
6     for j=1:c           % recorre columnas (en cada fila)
7         s=s+M(i,j);    % agrega cada elemento a s
8     end
9 end

```

EJERCICIO 2. Suponga que tiene un vector v de números. Programe, mediante un bucle `for` y una condición `if`, un algoritmo que busque el valor máximo de los elementos de v (sin emplear la función `max`)

EJERCICIO 3. Suponga que tiene un vector fila v de números. Programe mediante un bucle `for` un algoritmo que:

- busque el número n en el vector fila v
- muestre la **última** posición en que aparece el número n en el vector v (si es que está en v).

EJERCICIO 4. Suponga que tiene un vector fila v de números. Programe mediante un bucle `while` un algoritmo que:

- busque un número n en el vector fila v
- muestre la **primera** posición en que aparece el número n en el vector v (si es que está en v).

EJERCICIO 5. Suponga que tiene un vector fila v de números. Programe un algoritmo que:

- busque el número n en el vector fila v
- muestre **todas** las posiciones donde aparece el número n en el vector v (si es que está en v).

EJERCICIO 6. Suponga que v está ordenado (de menor a mayor). Programe un nuevo un buscador que indique si número n está en el vector. Emplee un bucle `while`, y la siguiente estrategia:

1. La posición más baja posible **Min** es inicialmente 1
2. La posición más alta posible **Top** es inicialmente `length(v)`
3. Busque n en una posición intermedia **pos** (aprox. $(\text{Top}-\text{Min})/2$)
4. Si el número está ahí: solución es **pos**
5. Si el número NO está ahí:

- a) Si número en `pos >n` entonces `Top=pos`
 - b) Si número en `pos <n` entonces `Min=pos`
 - c) repetir punto 3
6. Si finalmente `Top=Min` entonces `n` no está en `v`

Algunas funciones de utilidad para elaborar el ejercicio anterior:

`round`. redondeo La orden `round(n)` redondea al entero más próximo a `n`.

`floor`. redondeo La orden `floor(n)` redondea al entero más próximo pero menor que `n`.

`ceil`. redondeo La orden `ceil(n)` redondea al entero más próximo pero mayor que `n`.

`isempty`. La orden `isempty(n)` es verdadera si `n` está vacía.

`disp`. La orden `disp('Hola mundo')` muestra el mensaje entre comillas `Hola mundo`.

EJERCICIO 7. Suponga que tiene una sucesión de números y considere el algoritmo de la “burbuja” como procedimiento de ordenación:

El algoritmo de la “burbuja” ordena de menor a mayor una sucesión de números realizando comparaciones entre pares de números adyacentes, y permutando su posición si es necesario.

Considere una sucesión de números $a_1, a_2, a_3, \dots, a_N$ en las posiciones iniciales $1^a, 2^a, 3^a, \dots, n$ -ésima. Compare los números que se encuentran en la primera y segunda posición (a_1 y a_2). No altere el orden si $a_1 < a_2$, pero intercambie las posiciones en caso contrario. En el paso siguiente compare los números colocados en las posiciones segunda y tercera, y proceda de manera análoga (si el número que ocupa la posición segunda es menor que el que ocupa la tercera posición no altere su orden, pero intercambie las posiciones en caso contrario). Repita hasta comparar el último par de números (es decir, el último con el penúltimo). Este procedimiento desliza el número más grande de la sucesión hasta la última posición (¿tras cuantas comparaciones?). Repita el proceso anterior hasta ordenar de menor a mayor toda la secuencia (deslizando el segundo número mayor, luego el tercero, etc...).

Programa una función que implemente este algoritmo, y úsela para ordenar diez números enteros con valores entre 0 y 1000.

Pista. use `rand` y `round` para generar los diez números enteros.

↑
definición de funciones
12

Definición de una función en matlab:

- mediante un fichero *.m:


```
function f=funF(x)

    f=x-cos(x);
```
- directamente:


```
funF=@(x) x-cos(x)
```

`feval(funF,1)` evalúa la función en 1, es decir, calcula

$$f(1) = 1 - \cos(1)$$

EJERCICIO 8. Programe el método Newton-Raphson para búsqueda de “ceros” de una función.

Pista. Aproximando la función por el desarrollo de Taylor tenemos:

$$f(x_0 + h) = f(x_0) + hf'(x_0) + R,$$

donde R contiene todos los términos del desarrollo que no aparecen explícitamente en la expresión.

Buscamos el punto $x = x_0 + h$, donde la función es cero, y por tanto

$$0 \approx f(x_0) + hf'(x_0) \Rightarrow h = -\frac{f(x_0)}{f'(x_0)}$$

(suponiendo R pequeño); así pues, un valor tentativo es

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Al nuevo x lo llamamos x_0 y vuelta a empezar...

Además, recuérdese que

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

debería añadir ejercicio de “pivotaje” con anterioridad, ayudaría a programar Gauss

EJERCICIO 9. Programe una función que de las soluciones de un sistema de ecuaciones triangular, es decir, análogo al siguiente

$$\begin{pmatrix} 1 & 1 & 1 & 1 & : & 1.5 \\ 0 & 1 & 4 & 12 & : & 0.8 \\ 0 & 0 & -2 & -9 & : & -1 \\ 0 & 0 & 0 & -0.5 & : & -1.6 \end{pmatrix}$$

EJERCICIO 10. Programe una función que resuelva un sistema de ecuaciones por el método de Gauss, por ejemplo

$$\begin{pmatrix} 0 & 1 & 2 & 3 & : & -0.2 \\ 0 & 1 & 4 & 12 & : & 0.8 \\ 1 & 1 & 1 & 1 & : & 1.5 \\ 1 & 2 & 4 & 8 & : & 1.2 \end{pmatrix}$$

EJERCICIO 11. Programe el método de Gauss-Jordan

EJERCICIO 12. Programe media aritmética recursivamente, es decir, con un programe que se “llame a si mismo”

Pista. Si llamamos \bar{x}_n a la media aritmética de un vector con $n > 1$ datos,

$$\bar{x}_n = \left(\sum_{i=1}^n x_i \right) / n = \left(\sum_{i=1}^{n-1} x_i + x_n \right) / n = \left((n-1)\bar{x}_{n-1} + x_n \right) / n$$

se puede añadir un solucionador de sudokus recursivo

Lista de Transparencias

- 1 directorio de trabajo
- 2 Guiones
- 3 Bucle `for`
- 4 Bucle `for`: Ejemplo contador
- 5 Bucle `for`: Ejemplo contador
- 6 Bucle `for`: Ejemplo suma
- 7 Bucle `while`
- 8 Bucle `while`: Ejemplo registro de suceso
- 9 Ejemplo `if else`
- 10 Ejemplo de función
- 11 Ejemplo de función: suma
- 12 definición de funciones