CLASE 2 DE PL DEL CURSO 2020/21
GRUPO F, MÉTODOS NUMÉRICOS

PEDRO FORTUNY AYUSO

1. Introducción

En esta sesión se introducen los *archivos de función* (ó archivos de programa) de Matlab. Tienen algunas peculiaridades *importantes*:

(1) Cada archivo de función tiene que tener *el mismo nombre* que la función que define y el `.m` al final. P.ej. si se define una función `patata`, el archivo debe llamarse `patata.m`.
(2) Los archivos de función *deben estar* guardados siempre en el directorio `Documentos\MATLAB`. Si no, no se podrán ejectuar.
(3) En un archivo de función *solo se incluye* la definición. Cualquier cosa que no sea eso, *es un error*.
(4) Los archivos de función *no se ejecutan*: se usan las funciones definidas en ellos, como las demás funciones de Matlab.

Para aclarar estos cuatro puntos, ved primero de todo el vídeo sobre definición de funciones de Matlab.

Una vez hecho eso, intentad haced el programa que devuelva el mínimo y el máximo de una lista. Si se llama `minmax`, entonces el archivo ha de llamarse `minmax.m` y dentro de ese archivo debe ir *exclusivamente* este texto:

```
% minmax(L)
% devuelve dos elementos m, M
% m es el menor elemento de L
% M es el mayor elemento de L
function [m M] = minmax(L)
    m = min(L);
    M = max(L);
end
```

Listado 1. Contenido del archivo `minmax.m`.

Para utilizarlo, en el archivo `p12.m` podéis hacer ejemplos y correrlos, como siempre, seleccionando y presionando `F9`. Es mejor que vayáis copiando cada tres líneas y ejecutando, para aclararos mejor.

```
L1 = [-1:.01:1];
C1 = cos(L1); % lista de los cosenos
[a b] = minmax(C1) % sin punto y coma para verlos

L2 = linspace(0,2,1000);
C2 = exp(sin(L2));
[a b] = minmax(C2) % sin punto y coma
```

Fecha: 9 de febrero de 2021.

```
[c d] = minmax([-1 7 8 -33 42 7 -1 100])

% etc...
```

Una vez hecho esto, podéis ver el vídeo en que explico en detalle el programa `max2`.

Ahora leéis el Capítulo 1 de la prácticas de laboratorio de mi página web y vais intentando hacer los ejercicios 1,2 y 3. Están resueltos a continuación pero lo importante es que *saquéis lápiz y papel e intentéis hacerlos vosotros*: así es cómo realmente se aprende.

Finalmente, los ejercicios importantes son el 5 y el 6 y 7. También están resueltos al final de este documento.

Más que los ejercicios en sí, la clave de esta clase es que os acostumbréis a:

(1) Definir funciones de Matlab como archivos de función y utilizarlos *en el archivo de la PL*, o en la línea de comandos.
(2) Distinguir entre `for` y `while` y saber usar `if` y `elseif` (el `else` debería ser bien conocido). Importante: *todos* los bucles y los `if` hay que terminarlos con `end`.
(3) Saber obtener los valores de salida de una función que devuelve más de uno. Recuerdo:

```
[a b c] = FuncionConTresSalidas(x, y);
```

(4) Dar valores iniciales a las variables de retorno en el programa.

El documento de "estructura de un programa en Matlab" es, creo yo, bastante útil.

2. Soluciones a los ejercicios propuestos

Solución al ejercicio 1. El listado del programa es el siguiente: obsérvese que pide devolver *los tres elementos* (no una lista de los tres). Es prácticamente igual que `max2` pero con un `elseif` más:

```
% max3(L)
% Dada una lista L
% devuelve tres elementos: el máximo, el siguiente y el siguiente, en este
     orden
function [m1 m2 m3] = max3(L)
    % asignamos -inf a todos para que las 3 primeras iteraciones
    % siempre funcionen bien
    m1 = -inf;
    m2 = -inf;
    m3 = -inf;
    % hemos de recorrer toda la lista L: for
    for e = L % lo llamamos e ("elemento") porque "l" se confunde con 1
        if(e > m1)
            % por que estas asignaciones van en este orden?
            m3 = m2;
```

```
            m2 = m1;
            m1 = k;
        elseif(e > m2)
            m3 = m2;
            m2 = k;
        elseif(e > m3)
            m3 = k;
        end
    end
end
```

LISTADO 2. Contenido del archivo `max3.m`.

□

Solución al ejercicio 2. En este ejercicio, hay que *ir leyendo una lista mientras ocurra (o no ocurra) algo*: esto es claramente un `while`. En este caso nos dicen que si la lista cumple que cada elemento es menor o igual que el siguiente (no es decreciente), entonces se devuelve un 1. Si esto no ocurre (es decir, si en algún momento hay un elemento que es mayor que el siguiente), se devuelve un 0. Solo se devuelve un valor, que llamaremos T (de "test").

```
% es_creciente
% Dada una lista L
% devuelve un 1 si L(k) <= L(k+1) siempre
% y un 0 si no ocurre eso.
function [T] = es_creciente(L)
    % lo habitual es que no ocurra, asi que ponemos T=0 para empezar
    T = 0;
    % como hemos de recorrer la lista posicion a posicion (no por valores)
    % usamos un contador. Comenzamos en 1
    k = 1;
    % hacemos el test en cada posicion
    % IMPORTANTISIMO:
    % hemos de verificar que L(k+1) existe:
    % es decir, si hemos llegado al final ANTES de evaluar l(k+1)
    while (k < length(L) && L(k) <= L(k+1))
        k = k + 1;
    end
    % hemos recorrido toda la lista? si es asi, T = 1
    if (k == length(L))
        T = 1;
    end
end
```

LISTADO 3. Contenido del archivo `es_creciente.m`.

Aparte del `while`, lo más importante de este programa es verificar, *antes de mirarlo* que `L(k+1)` tiene sentido: es decir, que k es estrictamente menor que la longitud de la lista. Hágase el programa sin esa verificación y compruébese que da un error para listas ya ordenadas: `[1 2 3 4 5]`, por ejemplo. □

Solución al ejercicio 3. En este programa hay que devolver dos valores. Lo que hay que añadir al anterior es simplemente la variable de contador como variable de salida (pues el contador dice justamente lo que se busca: cuántos elementos hay que forman una sucesión creciente al principio de la lista). El programa queda, por tanto:

```
% es_creciente2
% Dada una lista L
% devuelve un 1 si L(k) <= L(k+1) siempre
% y un 0 si no ocurre eso.
function [T k] = es_creciente2(L)
    % lo habitual es que no ocurra, asi que ponemos T=0 para empezar
    T = 0;
    % como hemos de recorrer la lista posicion a posicion (no por valores)
    % usamos un contador. Comenzamos en 1
    k = 1;
    % hacemos el test en cada posicion
    % IMPORTANTISIMO:
    % hemos de verificar que L(k+1) existe:
    % es decir, si hemos llegado al final ANTES de evaluar l(k+1)
    while (k < length(L) && L(k) <= L(k+1))
        k = k + 1;
    end
    % hemos recorrido toda la lista? si es asi, T = 1
    if (k == length(L))
        T = 1;
    end
end
```

LISTADO 4. Contenido del archivo `es_creciente2.m`.

□

Solución al ejercicio 5. Hágase un dibujo. Esto es fundamental: sin él uno no será capaz de razonar correctamente. Por ejemplo, como el de la figura 1 (pero hágalo el alumno, claro, y razone por qué es así).

□

El programa no tiene más dificultad que hacer los cálculos que se muestran ya en el dibujo. Por tanto, es una mera asignación.

```
% Dada una funcion f, un punto x0 y una anchura epsilon
% devuelve la pendiente y la intercepcion de la recta secante que pasa
% por los puntos (x0-epsilon, f(x0-epsilon)) y (x0+epsilon, f(x0+epsilon))
function [a b] = secante(f, x0, e)
    % se asignan los valores y ya esta
    h = f(x0+e)-f(x0-e);
    a = h/(2*e); % OJO a los parentesis!!!!
    b = f(x0-e) - h/(2*e)*(x0-e); % OJO a los parentesis!!!!
end
```

LISTADO 5. Contenido del archivo `secante.m` del Ejercicio 5.

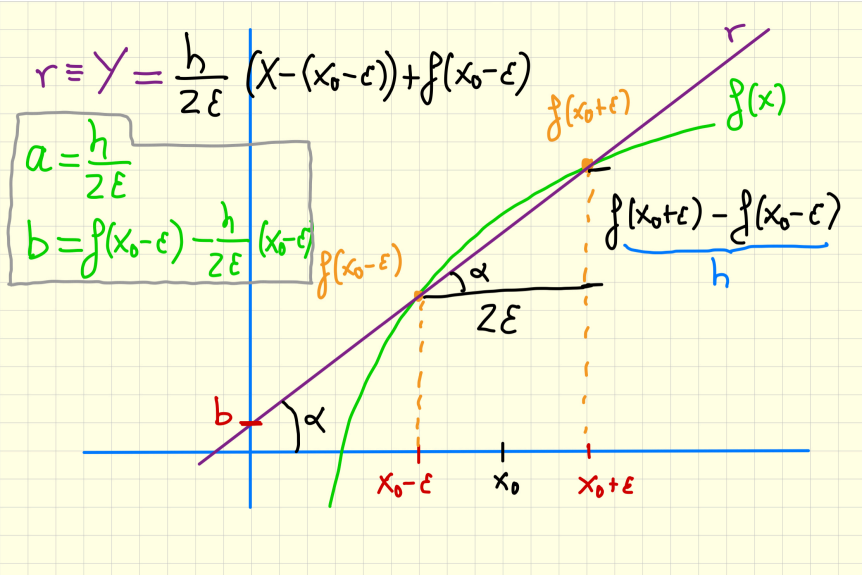

FIGURA 1. Figura para el ejercicio 5.

Para verificar que funciona con un ejemplo, se pueden dibujar una funcion y dicha secante. Tómese $f(x) = \cos(x)$, el punto $x_0 = 0$ y el valor $\epsilon = 0.1$. Ahora dibújense las dos gráficas: $f(x)$ y la secante $ax + b$ en un intervalo adecuado:

```
f = @(x) cos(x);
x0 = 0;
e = 0.1;
[a b] = secante(f, x0, e);
v = linspace(-.4,.4,1000);
plot(v, f(v));
hold on;
plot(v, a*v + b);
```

Solución al ejercicio 6. Este ejercicio sirve para saber cómo obtener elementos de una matriz (con paréntesis, dos puntos, etc...). Necesitamos crear una lista vacía para la salida `v` e ir llenándola con las filas de `A`, recorriendo estas filas de una en una. Es más sencillo de lo que parece.

```
% mav(A)
% devuelve un vector compuesto por las filas de A encadenadas de arriba
    abajo
function [v] = mav(A)
    [f c] = size(A); % f-> filas, c-> columnas
    v = [];
    for k=[1:f] % un contador para cada fila de A
        v = [v A(k,:)];
    end
```

```
end
```

LISTADO 6. Contenido del archivo `mav.m` del Ejercicio 6.

Comentario: la línea

```
v = [v A(k,:)];
```

hace lo siguiente: asigna a `v` (que es una lista) la lista (eso significan los corchetes) formada por el propio `v` y, a continuación, la expresión `A(k,:)`, que significa literalmente "de la fila `k` de `A`, tomar *todas* (eso es el `:`) las columnas: es decir, la fila `k` de `A`. Hágase la prueba con el ejemplo

```
A = [1 0 -3; 2 4 7; 3 2 1; -2 0 6];
v = mav(A)
```

□

Solución al ejercicio 7. En este ejercicio la entrada son 3 datos: el vector `v`, y las dimensiones de la matriz en que se quiere convertir, `n` y `m`. Hay que comprobar si `n`×`m` es mayor que la longitud de `v`, claro. Como se devuelve una matriz de 0 si no se puede conseguir, se comienza creándola por si acaso.

```
% vam(v, n, m)
% Convierte el vector v
% en una matriz de tamanyo n x m
% si n x m es mayor que la longitud, la matriz esta llena de 0
% si n x m es menor, los elementos sobrantes se olvidan
function [A] = vam(v, n, m)
    s = length(v);
    A = zeros(n, m); % y ahora vemos si se puede construir
    if(n*m > s)
        return;
    end
    for k=[1:n]
        A(k,:) = v((k-1)*m+1:k*m); % esta es la cuenta importante
    end
end
```

LISTADO 7. Contenido del archivo `vam.m` del ejercicio 7.

El punto clave es saber: ¿qué elementos de `v` son los que hay que poner en la fila `k` de la matriz `A`? Fijándose uno, para la primera fila (`k=1`) deben ser los que van desde el índice `1` hasta el índice `m` (esto se escribe `v(1:m)`). Para la segunda fila (`k=2`), son los que van desde `m+1` hasta `2*m` (que es `v(m+1:2*m)`. Y, cada vez que se cuenta una fila más, se añade uno a `k`:

```
1:m,  m+1:2*m,  2*m+1:3*m, ..., (k-1)*m+1:k*m
```

que es lo que se indica en la línea central del bucle `for`. Compruébese con el ejemplo.

```
v = [1 2 0 4 5 7 -2 -3 1 0 2 -6];
n = 3;
m = 4;
vam(v, n, m)
```

□

No hay que preocuparse mucho si uno "no sabe hacer bien" estos ejercicios, insisto. Lo importante es ir viendo cómo funciona la programación en Matlab, los bucles y las asignaciones. Según vayamos adelantando, todo irá encajando poco a poco y se verá que no hace falta una programación en mucho detalle.

Curso 2020/21, EPIG, Gijón. Universidad de Oviedo
Correo electrónico: fortunypedro@uniovi.es