

 UNIVERSIDAD DE OVIEDO DEPARTAMENTO DE MATEMÁTICAS	Asignatura	Métodos Numéricos	Página 1 de 4
	Tema	Sistemas Lineales y No Lineales: Punto fijo y Newton	
	Práctica	3.3	
	Autor	César Menéndez Fernández	

1 Métodos iterativos y de gradiente

En esta práctica se va a estudiar las instrucciones de Matlab para resolver sistemas no lineales. Se utilizarán tanto los métodos iterativos como los obtenidos en la minimización no restringida de funciones de varias variables.

1.1 Resolución analítica: solve

En aquellos casos en que sea posible obtener una solución analítica, se puede utilizar la instrucción `solve`. Por ejemplo, el siguiente sistema

$$F(X) = 0 \rightarrow \begin{pmatrix} f_1(x, y) \\ f_2(x, y) \end{pmatrix} = \begin{pmatrix} 5x^2 - y^2 \\ y - \frac{1}{4}(\sin x + \cos y) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ con } D = \{-2 \leq x, y \leq 2\}$$

Se resuelve mediante las instrucciones

```
» syms x y real
» F=[5*x^2-y^2; y-(sin(x)+cos(y))/4];
» S=solve(F(1),F(2)) % Solución doble
```

Se pueden obtener los puntos de corte de forma gráfica mediante

```
» t=linspace(-2,2); [X,Y]=meshgrid(t,t);
» Z1=subs(F(1),{x,y},{X,Y}); Z2=subs(F(2),{x,y},{X,Y});
» figure(1);
» hold off; contour(X,Y,Z1,[0,0]); hold on;
» contour(X,Y,Z2,[0,0]); grid on; axis('equal');
```

Sin embargo, esta instrucción no siempre funciona correctamente.

Se pide:

1. Verificar el funcionamiento de `solve` cuando se intenta resolver el sistema.

$$\begin{aligned} 4x^2 + y^2 - 4 &= 0 \\ e^x + y - 1 &= 0 \end{aligned} \text{ con } D = \{-2 \leq x, y \leq 2\}$$

1.2 Métodos Iterativos

MATLAB implementa la función ***fsolve*** para resolver (obtener las raíces) sistemas de ecuaciones no lineales. A continuación resumimos las diferentes sintaxis.

<code>X=fsolve(fun,x0)</code>	Intenta resolver el sistema no lineal $\text{fun}(X)=0$ partiendo del valor $X=x_0$
<code>X=fsolve(fun,x0,options)</code>	Resuelve el sistema con las opciones seleccionadas mediante la instrucción <code>optimset</code>
<code>[X,F]=fsolve(fun,x0)</code>	Devuelve también el valor de la función
<code>[X,F,finflag]=fsolve(...)</code>	Devuelve también la condición que ha motivado el final del programa.
	1 La función converge a la solución
	2 El cambio de x fue menor que la tolerancia especificada

	3	El cambio del residuo fue menor que la tolerancia especificada
	4	La magnitud de la dirección de búsqueda fue menor que la tolerancia especificada
	5	El número de iteraciones supera el valor <code>MaxIter</code> o el número de evaluaciones de la función supera <code>FunEvals</code> .
	-1	El algoritmo fue finalizado por la función de salida.
	-2	El algoritmo parece converger a un punto que no es la raíz.
	-3	El radio de confianza es demasiado pequeño.
	-4	La dirección de búsqueda no disminuye el valor del residual.
<code>[X,F,finflag,info]=fsolve(...)</code>	Devuelve también una estructura con información sobre el proceso. Sus campos son:	
	<code>iterations</code>	Número de iteraciones realizadas
	<code>count</code>	Número de evaluaciones de la función
	<code>algorithm</code>	Algoritmo usado
	<code>cgiterations</code>	Número de iteraciones PCG (solo con algoritmos de gran escala)
	<code>stepsize</code>	Tamaño final del paso (solo con algoritmos de mediana escala)
<code>[X,F,finflag,info,jaco]=fsolve(...)</code>	Devuelve también el valor del jacobiano	

La función viene descrita en un m-archivo, pudiendo tener uno o dos valores de salida, el valor de la función en el punto y el del jacobiano. Para usar el valor del jacobiano, es necesario que se haya activado previamente mediante

```
» options = optimset('Jacobian','on')
```

1.2.1 Opciones

Las opciones se definen con la instrucción `optimset` y estan asociadas al tipo de algoritmo usado en la resolución así como al tamaño del problema (media o gran escala). La opción '`LargeScale`' '`on`' indica que se prefiere utilizar un algoritmo de gran escala; esto es simplemente una preferencia, puesto que depende de que la función verifique las condiciones exigidas en tales algoritmos. En todo caso el número de ecuaciones debe coincidir o superar el número de incógnitas (no puede ser indeterminado).

1.2.1.1 Opciones comunes

<code>DerivativeCheck</code>	Compara las derivadas calculadas mediante el Jacobiano (dadas por la función) con las calculadas mediante diferencias finitas.
<code>Diagnostics</code>	Muestra información diagnóstica sobre la función a resolver.
<code>DiffMaxChange</code>	Máximo cambio de las variables cuando se usan diferencias finitas.
<code>DiffMinChange</code>	Mínimo cambio de las variables cuando se usan diferencias finitas.
<code>Display</code>	Nivel de información en pantalla: ' <code>off</code> ' ninguna; ' <code>iter</code> ' muestra cada iteración; ' <code>final</code> ' (defecto) sólo muestra la última.
<code>FunValCheck</code>	Verifica la validez de los valores de la función objetivo. ' <code>on</code> ' muestra un mensaje de error cuando la función objetivo devuelve un valor complejo, Inf, o NaN. ' <code>off</code> ' (defecto) no muestra los errors.
<code>Jacobian</code>	Si ' <code>on</code> ', utiliza el Jacobiano definido por la función, en otro caso lo aproxima mediante diferencias finitas.
<code>MaxFunEvals</code>	Máximo número de evaluaciones de la función permitido.
<code>MaxIter</code>	Máximo número de iteraciones permitido.
<code>PlotFcns</code>	Dibuja varias medidas de progreso del algoritmo durante su ejecución; puede ser una función definida por el usuario o utilizar alguna de las opciones de MATLAB, tales como <code>@optimplotx</code> (punto actual), <code>@optimplotfunccount</code> (contador de la función), <code>@optimplotfval</code> (valor de la función), <code>@optimplotresnorm</code> (norma del residuo), <code>@optimplotstepsize</code> (paso actual).

TolFun Tolerancia admisible del valor de la función.

TolX Tolerancia admisible de la variación de x.

1.2.1.2 Opciones de los algoritmos de mediana escala

NonlEqnAlgorithm especifica el algoritmo de resolución del sistema no lineal: 'dogleg' (defecto, newton con regiones de confianza), 'lm' (Levenberg-Marquardt), 'gn' (Gauss-Newton).

LineSearchType Elección del algoritmo de búsqueda, sólo aplicable a Levenberg-Marquardt y Gauss-Newton. Puede ser 'cubicpoly' o 'quadcubic' (defecto).

1.2.2 Ejemplos

Ejemplo 1: Resolver, comenzando en $x_0 = [-5, -5]$, el siguiente sistema no lineal del apartado anterior

$$F(X) = 0 \rightarrow \begin{pmatrix} f_1(x, y) \\ f_2(x, y) \end{pmatrix} = \begin{pmatrix} 5x^2 - y^2 \\ y - \frac{1}{4}(\sin x + \cos y) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ con } D = \{-2 \leq x, y \leq 2\}$$

Se comienza creando el archivo que define esa función:

```
function F = mifun1(x)
F = [5*x(1).^2 - x(2).^2; x(2) - 0.25*(sin(x(1))+cos(x(2)))];
```

A continuación definimos el punto de inicio y seleccionamos las opciones del método

```
>> x0 = [-5; -5];
>> options=optimset('Display','iter');
>> [x,fval] = fsolve(@mifun1,x0,options)
```

Para comprobar si existe alguna variación cuando se utiliza el jacobiano es necesario modificar la definición de la función

```
function [F,JF] = mifun1(x)
...
if nargin==1;return;end % Definicion del Jacobiano
JF(1,:)= [10*x(1) , -2*x(2); -0.25*cos(x(1)), 1+0.25*sin(x(2))];
```

Para ejecutar *fsolve* tras cambiar las opciones

```
>> options = optimset('Jacobian','on')
>> [x,fval] = fsolve(@mifun1,x0,options)
```

Ejemplo 2. Comenzando con los valores $x_0 = [1, 1; 1, 1]$ encontrar la matriz que satisface:

$$X^3 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Como en el caso anterior, se comienza por definir el m-archivo de la función, se da valor al punto inicial, se definen las opciones y se ejecuta la instrucción *fsolve*.

```
function F = mifun2(x)
F = x*x*x-[1,2;3,4];
>> x0 = ones(2);
>> options=optimset('Display','off');
>> [x,fval,finflag] = fsolve(@mifun2,x0,options)
```

Se pide:

2. Resolver el siguiente sistema no lineal:

$$\begin{aligned} 4x^2 + y^2 - 4 &= 0 \\ e^x + y - 1 &= 0 \end{aligned} \quad \text{con } D = \{-2 \leq x, y \leq 2\}$$

- Utilizando las opciones por defecto y partiendo de los puntos: [1,-1],[-1,1].
- Repetir partiendo del origen y representando las iteraciones.
- Repetir el caso anterior pero utilizando el jacobiano y otros métodos.

Para todos los casos se puede utilizar el siguiente código en la definición de la función:

```
function [F,JF]=mifun3(X)
if nargin~=1;error('numero de argumetos no valido');end
[n,nc]=size(X);if nc~=1;error('X no es un vector columna'),end
%
% Definicion de la funcion
F=zeros(n,1);
F(1)=4*X(1).^2+X(2).^2-4;
F(2)=exp(X(1))+X(2)-1;
if nargout==1;return;end % Definicion del Jacobiano
JF=zeros(n,n);
JF(1,:)=[4*X(1) , 2*X(2)];
JF(2,:)=[exp(X(1)),1];
```

1.3 Minimización no restringida

MATLAB implementa la función **lsqnonlin** para este caso. A continuación resumimos las diferentes sintaxis.

$X = \text{lsqnonlin}(\text{fun}, x_0)$	Intenta obtener el $\min_x \ F(X)\ _2^2 = \min_x \sum_i f_i(X)^2$ partiendo de $X=x_0$
$X = \text{lsqnonlin}(\text{fun}, x_0, \text{lb}, \text{ub}, \text{options})$	Intenta obtener el mínimo con los valores acotados por $\text{lb} \leq X \leq \text{ub}$ y las opciones seleccionadas mediante la instrucción <i>optimset</i>
$[X, \text{resn}, \text{resid}] = \text{lsqnonlin}(\dots)$	Devuelve también la norma dos del residuo y el valor del residual

Para resolver el ejemplo del ejercicio3 anterior mediante esta instrucción partiendo del origen, esto es.

$$\min_x \|F(X)\|_2^2 = \min_x \sum_i f_i(X)^2 = \min_x \left((4x^2 + y^2 - 4)^2 + (e^x + y - 1)^2 \right)$$

se tendría

```
» x0 = [0; 0];
» options=optimset('Display','iter');
» [x,resnorm,residue] = lsqnonlin(@mifun3,x0,options)
```