Capítulo 1

Introducción

1.1 ¿Qué es la Toolbox de Optimización?

La *Optimization Toolbox* es una colección de funciones que extienden la capacidad de MATLAB. Esta *toolbox* incluye rutinas para diversos tipos de optimización incluyendo:

- Minimización no lineal sin restricciones.
- Minimización no lineal con restricciones.
- Programación cuadrática y lineal.
- Ajuste de curvas por mínimos cuadrados.
- Resolución de sistemas de ecuaciones no lineales.
- Mínimos cuadrados con restricciones.

Todas las funciones de la *toolbox*, están realizadas en mediante código de MATLAB, puedes ver el código de esas funciones mediante la instrucción

```
type function_name
```

Obviamente, se pueden extender las capacidades de la toolbox escribiendo código propio mediante ficheros M, o utilizando la toolbox en combinación con otras toolboxes, o con MATLAB o Simulink.La Optimización está orientada a la minimización o maximización de funciones. La Toolbox de optimización consiste en un conjunto de funciones que realizan la minimización (o maximización) de funciones no lineales en general. También se proporcionan funciones para resolver problemas de ecuaciones no lineales y ajuste de datos (mínimos cuadrados). Esta introducción incluye las siguientes secciones:

- Problemas abarcados por la Toolbox
- Uso de las funciones de optimización.

1.2 Problemas tratados

Las tablas siguientes resumen las funciones disponibles para minimización, resolución de ecuaciones y problemas de mínimos cuadrados.

Observación 1 *La siguiente tabla indica los tipos de problemas en orden creciente de complejidad:*

Tipo	Notación	Función
Minimización Univariante	$\begin{cases} \min f(x) \\ x \in]a,b[\end{cases}$	*fminbnd*
Minimización sin restricciones	$\begin{cases} \min f(\mathbf{x}) \\ \mathbf{x} \in \Omega \subseteq \mathbb{R}^n \end{cases}$	*fminunc, fminsearch*
Programación Lineal	$\begin{cases} \min \mathbf{c}^T\mathbf{x} \\ \mathbf{Ax} \leq \mathbf{b} \\ \mathbf{A}_e\mathbf{x} = \mathbf{b}_e \\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{cases}$	*linprog*
Programación Cuadrática	$\begin{cases} \min \frac{1}{2}\mathbf{x}^T\mathbf{Hx} + \mathbf{c}^T\mathbf{x} \\ \mathbf{Ax} \leq \mathbf{b} \\ \mathbf{A}_e\mathbf{x} = \mathbf{b}_e \\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{cases}$	*quadprog*
Minimización con restricciones	$\begin{cases} \min f(\mathbf{x}) \\ \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ \mathbf{h}(x) = \mathbf{0} \\ \mathbf{Ax} \leq \mathbf{b} \\ \mathbf{A}_e\mathbf{x} = \mathbf{b}_e \\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{cases}$	*fmincon*

Tabla 1.1: Tabla 1-1: Minimización

Tipo	Notación	Función
Ecuaciones Lineales	$\begin{cases} Ax = b \\ n \text{ ecuaciones} \\ n \text{ variables} \end{cases}$	\ *(Barra invertida)*
Ecuación lineal de una variable	$\begin{cases} f(x) = 0 \end{cases}$	*fzero*
Sistemas de ecuaciones no lineales	$\begin{cases} \mathbf{F}(\mathbf{x}) = 0 \end{cases}$	*fsolve*

Tabla 1.2: Tabla 1-2: Resolución de ecuaciones

Tipo	Notación	Función
Mínimos cuadrados lineales	$\begin{cases} \min \lVert\mathbf{Ax} - \mathbf{b}\rVert_2^2 \\ m \text{ ecuaciones} \\ n \text{ variables} \end{cases}$	\ *(Barra invertida)*
Minimos cuadrados lineales no negativos	$\begin{cases} \min \lVert\mathbf{Ax} - \mathbf{b}\rVert_2^2 \\ \mathbf{x} \geq \mathbf{0} \\ m \text{ ecuaciones} \\ n \text{ variables} \end{cases}$	*lsqnonneg*
Mínimos cuadrados lineales con restricciones	$\begin{cases} \min \lVert\mathbf{Ax} - \mathbf{b}\rVert_2^2 \\ \mathbf{Ax} \leq \mathbf{b} \\ \mathbf{A}_e\mathbf{x} = \mathbf{b}_e \\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \\ m \text{ ecuaciones} \\ n \text{ variables} \end{cases}$	*lsqlin*
Mínimos cuadrados no lineales	$\begin{cases} \min \lVert\mathbf{F}(\mathbf{x})\rVert_2^2 = \frac{1}{2}\sum_i F_i(\mathbf{x}) \\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{cases}$	*lsqnonlin*
Ajuste de curvas no lineal	$\begin{cases} \min \lVert\mathbf{F}(\mathbf{x},\mathbf{y}) - \mathbf{y}\rVert_2^2 \\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{cases}$	*lsqcurvefit*

Tabla 1.3: Tabla 1-3: Ajuste de curvas

La mayoría de estas rutinas de optimización necesitan la definición de un fichero de tipo *M*, que incluya la definición de la función que hay que minimizar, i.e., la función objetivo. De forma alternativa, se puede crear un objeto *inline* creado a partir de una expresión de MATLAB. La maximización se obtiene indicando la rutina con $-f$, donde f es la función que hay que optimizar.

Las opciones de optimización que se pueden indicar en las rutinas cambian los parámetros de optimización. Los parámetros de optimización utilizados por defectos son los más usuales, pero pueden cambiarse a través de una estructura de opciones.

El gradiente se calcula mediante un método de diferencias finito adaptativo, a menos que se proporcione a través de una función. También se pueden incluir parámentros directamente a las funciones, eliminando la necesidad de utilizar variables globales.

Se distinguen dos tipos de algoritmos: algoritmos de gran escala y algoritmos de escala media. Este no es un término estándar, y se utiliza solamente para diferenciarse de los algoritmos de gran escala, que se han diseñado explícitamente para manejar problemas de gran escala eficientemente.

En este curso solamente utilizaremos las opciones y algoritmos de escala media. Las funciones de la *Toolbox de Optimización* continen diferentes de algoritmos multivariante, así como diferentes estrategias de búsquedad lineal. El algoritmo principal para minimización con restricciones son el método simplex de Nelder-Mead, y el método cuasi-Newton de BFGS (Broyden, Fletcher, Goldfarb, y Shanno).

Para la minimización con restricciones se utilizan variaciones de un algoritmo de programación cuadrática secuencial. Mientras que se utilizarán los algoritmos de Gauss-Newton y Levenberg-Marquardt para problemas de mínimos cuadrados no lineales.

Además se da la opción de elegir la estrategia de búsqueda lineal para minimización sin restricciones y para problemas de mínimos cuadrados no lineales. Las estrategias de búsqueda lineal utilizan métodos de interpolación cúbicos y cuadráticos con control (safeguarded methods).

1.3 Ejemplos

En esta sección se presentan los algoritmos estándar a través de un manual. El manual trata las funciones *fminunc* y *fmincon* con detalle. El resto de rutinas de optimización se utilizan de la misma forma, solamente con diferencias en los problemas que pueden tratarse y en los criterios de terminación. Esta sección incluye los siguientes ejemplos:

- Ejemplo sin restricciones.
- Ejemplo con restricciones no lineales de desigualdad.
- Ejemplo con restricciones y cotas en las variables.
- Ejemplo con restricciones, utilizando el gradiente.
- Comprobación del gradiente: Analítico versus numérico

También se discute acerca de:

- Maximización.
- Restricciones de mayor o igual que cero.
- Argumentos adicionales: prescindiendo de Variables Globales.

1.3.1 Problema sin restricciones

Consideremos el problema de encontrar un conjunto de valores $[x_1^*, x_2^*]$ que resuelva el problema

$$\text{Minimizar} \quad f(\mathbf{x}) = e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right) \tag{1.1}$$

Para resolver este problema dos-dimensional, escribimos un fichero de tipo *M (M-file)*, que devuelva el valor de la función. Después, se utilizará la rutina de minimización sin restricciones *fminunc.*

Paso 1: Write an M-file objfun.m

```
function f=objfun(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
```

Paso 2: Utilizar una función de optimización sin restricciones

```
x0 = [-1,1]; % Starting guess
options = optimset('LargeScale','off');
[x,fval,exitflag,output] = fminunc(@objfun,x0,options);
```

Después de 40 evaluaciones de la función, se obtiene como solución

```
x =
0.5000 -1.0000
```

El valor de la función en la solución se almacena en la variable **fval.**

```
fval =
1.3030e-10
```

La variable **exitflag** indica si el algoritmo converge. Un valor de **exitflag** > 0 indica que se ha localizado un mínimo local.

```
exitflag =
1
```

La estructura **output** proporciona más detalles respecto al proceso de optimización. En el caso de *fmiunc,* incluye el número de iteraciones del proceso en **iterations,** el número de evaluaciones de la función en **funCount,** el tamaño de paso final en **step-size,** la norma infinita del gradiente en la solución en **firstorderopt,** y el tipo de algoritmo utilizado en **algorithm.**

```
output =
iterations: 7
funcCount: 40
stepsize: 1
firstorderopt: 9.2801e-004
algorithm: 'medium-scale: Quasi-Newton line search'
```

Cuando la función tiene más de un mínimo local, el valor inicial para el vector $[x_1, x_2]$, afecta al número de evaluaciones de la función y al valor del punto solución. En el ejemplo anterior, x_0 se ha inicializado al valor $[-1, 1]$. La variable **options** puede utilizarse para cambiar las características del algoritmo de optimización, como en

```
x = fminunc(@objfun,x0,options);
```

options es una estructura que contiene valores para el error permitido y la elección del algoritmo. Se puede generar una estructura de este tipo mediante la función *optimset.*

```
options = optimset('LargeScale','off');
```

En este ejemplo hemos desactivado la selección de algoritmo a gran escala y por tanto se ha utilizado el algoritmo estándar (escala media). Otras opciones de optimización incluyen el control sobre la cantidad de información presentada durante cada iteración del proceso, los criterios de tolerancia o error permitido, si el usuario va a introducir el gradiente o el Jacobiano, y el máximo número de iteraciones o evaluaciones de la función, posteriormente se verá algún parámetro más de optimización y más información al respecto.

1.3.2 Problema con restricciones de desigualdad

Supongamos que añadimos restricciones de desigualdad en el ejemplo dado en la ecuación 1.1, en ese caso el problema resultante puede resolverse mediante la función *fmincon.* Por ejemplo, Encontrar la $\mathbf{x}$ que resuelva el problema

$$\begin{array}{cc} \text{Minimizar} & f(\mathbf{x}) = e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right) \\ \text{Sujeto a} & x_1x_2 - x_1 - x_2 \leq -1.5 \\ & x_1x_2 \geq -10 \end{array} \tag{1.2}$$

Puesto que ninguna de las restricciones es lineal, no se pueden pasar las restricciones a la función *fmincon* directamente en la línea de comando. En este caso tenemos que crear un segundo fichero *M-file,* que llamaremos *confun.m* que devuelva en una variable, el valor de ambas restricciones en el valor actual de $\mathbf{x}$. Después se utiliza la función de optimización con restricciones, *fmincon.* Como *fmincon* espera que las restricciones estén escritas en la forma

$$g(x) \leq 0$$

habrá que re-escribir las restricciones de la siguiente forma:

$$\begin{array}{c} x_1x_2 - x_1 - x_2 + 1.5 \leq 0 \\ -x_1x_2 - 10 \leq 0 \end{array} \tag{1.3}$$

Paso 1: Write an M-file confun.m para las restricciones

```
function [c, ceq] = confun(x)
% Restricciones de desigualdad
c = [1.5 + x(1)*x(2) - x(1) - x(2);
-x(1)*x(2) - 10];
% Restricciones de igualdad
ceq = [];
```

Paso 2: Utilizar una función de optimización coin restricciones

```
x0 = [-1,1]; % Starting guess
options = optimset('LargeScale','off');
[x, fval] = ...
fmincon(@objfun,x0,[],[],[],[],[],[],@confun,options);
```

Después de 38 evaluaciones de la función, la solución $\mathbf{x}^*$, proporcionada y su valor objetivo son:

```
x =
-9.5474 1.0474
fval =
0.0236
```

Podemos evaluar las restricciones en la solución de la forma:

```
[c,ceq] = confun(x)

c=
1.0e-15 *
-0.8882
0
ceq =
[]
```

Notar que en ambas restricciones el valor es menor o igual que cero, es decir, x satisface

$$\mathbf{c}(x) \leq \mathbf{0}$$

1.3.3 Ejemplo de restricciones con cotas en las variables

Las variables del vector **x** pueden estar restringidas a ciertos valores especificando restricciones de tipo cota. Para *fmincon,* el comando

```
x = fmincon(@objfun,x0,[],[],[],[],lb,ub,@confun,options);
```

indica que la variable **x** está acotada dentro del rango $\mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub}$.

Para indicar en el ejemplo anterior (ecuación 1.2), que las variables deben ser mayores o iguales que cero (i.e. $x_1 \geq 0$, $x_2 \geq 0$), utilizamos el los siguientes comandos:

```
x0 = [-1,1]; % Valor inicial
lb = [0,0]; % Indica las cotas inferiores
ub = [ ]; % No hay cotas superiores
options = optimset('LargeScale','off');
[x,fval = ...
fmincon(@objfun,x0,[],[],[],[],lb,ub,@confun,options)
[c, ceq] = confun(x)
```

Notar que para pasar los límites inferiores como el séptimo argumento de la función *fmincon,* hay que especificar valores para los argumentos tercero al sexto. En este ejemplo, se utiliza la matriz vacía [], como valor de esos argumentos puesto que no hay ni desigualdades, ni igualdades lineales.

Después de 13 evaluaciones de la función, la solución obtenida:

```
x =
0 1.5000

fval =
8.5000

c =
0
-10

ceq =
[]
```

Cuando **lb** o **ub** contienen menos elementos que **x**, solamente los primeros elementos correspondientes en esta variable estarán acotados. De la misma forma, si solamente hay algunas variables acotadas, podemos utilizar *-inf* en **lb** para variables no acotadas inferiormente e *inf* en **ub** para variables no acotadas superiormente. Por ejemplo

```
lb = [-inf 0];
ub = [10 inf];
```

produce $x_1 \leq 10$ y $0 \leq x_2$ (x_1 no tiene límites inferiores y x_2 no tiene límites usperiores). Utilizar *inf* y *-inf* proporciona resultados numéricos mejores que utilizar números positivos o un números negativos muy grande para indicar la ausencia de cota en las variables.

Notar que el número de evaluaciones de la función necesarias para encontrar la solución se ha reducido, puesto que se ha restringido el espacio de búsqueda. Se utilizan menos evaluaciones de la función cuando el problema tiene más restricciones y cotas en las variables puesto que la optimización realiza mejores decisiones que para el caso sin restricciones. Es por tanto, una buena práctica acotar y condicionar los problemas, cuando sea posible, para producir una rápida convergencia a la solución.

1.3.4 Ejemplo de problema con restricciones utilizando el gradiente

Normamente los algoritmos estándar de las rutinas de minimización utilizan gradientes numéricos calculado mediante aproximaciones en diferencias finitas. Este procedimiento perturba sistemáticamente cada variable para calcular las derivadas parciales de la función objetivo y de las restricciones. Alternativamente, se puede proporcionar una función para el cálculo de las derivadas de forma analítica. En este caso, el problema se resuelve de forma más precisa y eficiente.

Para resolver el problema 1.2, utilizando el gradiente de forma analítica, hay que hacer lo siguiente:

Paso 1: Escribe un fichero M para la función objetivo y el gradiente

```
function [f,G] = objfungrad(x)

f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);

% Gradiente de la f. objectivo
t = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
G = [ t + exp(x(1)) * (8*x(1) + 4*x(2)),
exp(x(1))*(4*x(1)+4*x(2)+2)];
```

Paso 2: Escribe un fichero M, para las restricciones no lineales y sus gradientes

```
function [c,ceq,DC,DCeq] = confungrad(x)
c(1) = 1.5 + x(1) * x(2) - x(1) - x(2); % Restricciones de desigualdad
c(2) = -x(1) * x(2)-10;
% Gradiente de las restricciones
DC= [x(2)-1, -x(2);
x(1)-1, -x(1)];
% Restricciones de igualdad no lineales
ceq=[];
DCeq = [ ];
```

G, contiene la derivada parcial de la función objetivo, *f,* y calculada en la función *objfungrad(***x***),* con respecto a cada elemento de **x**

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right) + e^{x_1}\left(8x_1 + 4x_2\right) \\ \\ e^{x_1}\left(4x_1 + 4x_2 + 2\right) \end{bmatrix} \tag{1.4}$$

Las columnas de DC contienen las derivadas parciales para cada restricción (i.e., la i–ésima columna de DC es la derivada parcial de la i–ésima restricción respecto a $\mathbf{x}$). Por tanto en el ejemplo anterior, DC es:

$$\begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_2}{\partial x_1} \\ \frac{\partial c_1}{\partial x_2} & \frac{\partial c_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} x_2 - 1 & -x_2 \\ x_1 - 1 & -x_1 \end{bmatrix} \tag{1.5}$$

Puesto que se ha proporcionado el gradiente de la función objetivo en *objfungrad.m* y el gradiente de las restricciones en *confungrad.m*, hay que indicarle a la función *fmincon* que esos ficheros M, contienen esa información adicional. Utilizaremos la función *optimset* para cambiar el valor de los parámetros *GradObj* y *GradConstr* por *'on',* en la estructura *options* del ejemplo

```
options = optimset(options,'GradObj','on','GradConstr','on');
```

Si no se cambia el valor de estos parámetros por *'on'* la función *fmincon* no utilizará los gradientes analíticos.

Los argumentos **lb** y **ub**, indican las cotas inferiores y superiores de la variable independiente **x**. En este ejemplo no tenemos de este tipo de restricciones, de modo que se utiliza la opción []

Paso 3: Llamar a la rutina de optimización con restricciones

```
x0 = [-1,1]; % Valor inicial
options = optimset('LargeScale','off');
options = optimset(options,'GradObj','on','GradConstr','on');
lb = [ ]; ub = [ ]; % No hay cotas
[x,fval] = fmincon(@objfungrad,x0,[],[],[],[],lb,ub,...
@confungrad,options)
[c,ceq] = confungrad(x) % Valor de las restricciones en x
```

Después de 20 evaluaciones de la función la solución obtenida es:

```
x =
-9.5474 1.0474

fval =
0.0236

c =
1.0e-14 *
0.1110
-0.1776

ceq =
[]
```

1.3.5 Verificación del gradiente: Analítico vs. Numérico

Cuando se determinan los gradientes de forma analítica, podemos compararlos sus valores en un punto con los correspondientes valores obtenidos mediante evaluación de diferencias finitas. Esta propiedad es bastante útil para detectar errores en la formulación bien de la función objetivo o bien de los gradientes.

Si queremos que esta comprobación se efectúe, entonces hay que modificar el valor del parámetro *DerivativeCheck* por *'on'*, utilizando la función *optimset.*

```
options = optimset(options,'DerivativeCheck','on');
```

En el primer ciclo de la optimización se comprueba el valor de los gradiente obtenidos de forma analítica (de la función objetivo, y si existen, de las restricciones no lineales). Si existe una discrepancia muy grande entre ambos valores, dentro de una tolerancia dada, aparece un mensaje de advertencia indicando la discrepancia y dando la oportunidad de parar el proceso de optimización.

1.3.6 Ejemplo de restricciones de igualdad

Para las rutinas que permiten restricciones de igualdad, las restricciones no lineales de este tipo deben calcularse en el fichero M, junto con las restricciones de desigualdad. Para igualdades lineales, solamente se necesitan los coeficientes que se pasan a la función a través de una matriz Aeq, y de un vector de términos independientes beq.

Por ejemplo, si tenemos la restricción de igualdad no lineal

$$x_1^2 + x_1^2 = 1$$

y la restricción no lineal de desigualdad

$$x_1 x_2 \geq -10$$

se reescriben en primer lugar como

$$x_1^2 + x_1^2 - 1 = 0$$

$$-x_1 x_2 - 10 \leq 0$$

y resolvemos el problema mediante los siguientes pasos:

Paso 1: Escribimos un fichero M, **llamado** *objfun.m*

```
function f = objfun(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
```

Paso 2:Escribimos un fichero M, **para las restricciones no lineales** *confuneq.m*

```
function [c, ceq] = confuneq(x)
% Restricciones de desigualdad no lineales
c = -x(1)*x(2) - 10;
% Restricciones de igualdad lineales
ceq = x(1)^2 + x(2) - 1;
```

Paso 3: Utilizamos la rutina de optimización con restricciones

```
x0 = [-1,1]; % Valor inicial
options = optimset('LargeScale','off');
[x,fval] = fmincon(@objfun,x0,[],[],[],[],[],[],...
@confuneq,options)
[c,ceq] = confuneq(x) % Comprobamos el valor de las restricciones en x
```

Después de 21 evaluaciones de la función, la solución obtenida es:

```
x =
-0.7529 0.4332

fval =
1.5093

c =
-9.6739

ceq =
4.0684e-010
```

Notar que *ceq* es igual que 0 dentro del margen de tolerancia de las restricciones que es de $1.0e-006$, y que c es menor o igual que cero como se requería.

1.3.7 Maximización

Las funciones de optimización *fminbnd, fminsearch, fminunc, fmincon, lsqcurvefit y lsqnonlin* realizan la minimización de la función objetivo, $f(\mathbf{x})$. La maximización se puede conseguir mediante una función M, que devuelva $-f(\mathbf{x})$. De forma similar, para alcanzar la maximización para *quadprog* se utiliza $-H$ y $-f$, y para *linprog* $-f$.

1.3.8 Restricciones de mayor o igual

La toolbox de Optimziación asume que las restricciones no lineales de desigualdad son del tipo

$$g_i(\mathbf{x}) \leq 0$$

para utilizar restricciones del tipo

$$g_i(\mathbf{x}) \geq 0$$

debe multiplicarse por -1, para conseguir una restricción del tipo menor o igual. Por ejemplo, una restricción del tipo $g_i(\mathbf{x}) \geq 0$, es equivalente a la restricción $-g_i(\mathbf{x}) \leq 0$; una restricción de la forma $g_i(\mathbf{x}) \geq b$, es equivalente a la restricción $-g_i(\mathbf{x}) - b \leq 0$.

1.3.9 Argumentos adicionales: variables globales

Se pueden indicar parámetros adicionales en las rutinas de optimización para aplicarse a funciones definida a través de parámetros y sin necesidad de emplear variables globales. Estos parámetros deben indicarse al final de la llamada a la rutina de optimización.

Por ejemplo, la introducción de variables al final de la llamada a la rutina *fsolve*

```
[x,fval] = fsolve(@objfun,x0,options,P1,P2,...)
```

incluye los argumentos directamente en la función *objfun*

```
F = objfun(x,P1,P2, ... )
```

Consideremos, por ejemplo, la búsqueda de los ceros de la función *ellipj(u,m).* La función necesita un parámetro m, así como una entrada u. Si queremos buscar un cero cerca del valor $u = 3$, para $m = 0.5$, utilizaremos

```
m = 0.5;
options = optimset('Display','off'); % Sin salidas
x = fsolve(@ellipj,3,options,m)
```

que devuelve

```
x =
3.7081
```

Y en este caso, el valor de la función *ellipj*

```
f = ellipj(x,m)
f =
-2.9925e-008
```

La utilización de la función *optimset* se hace para obtener todos los parámetros predeterminados que la función *fsolve* tiene. Esta llamada implica, además que se utilizan los errores de tolerancia por defecto y que no se utilizarán los gradientes analíticos.

1.4 Cambios en los parámetros

La estructura *options* contiene parámetros utilizados en las rutinas de optimización. Si, en la primera llamada a una rutina de optimización, no se proporciona la estrutura *options,* o es una matriz vacía, se genera entonces, un conjunto de parámetros predeterminados. Algunas de las opciones por defecto de los parámetros se calculan en base a factores relacionados con la dimensión del problema, tales como *MaxFunEvals.* Algunos parámetros dependen de la rutina de optimziación elegida, y su descripción se proporcionará posteriormente en la referencia a la función correspondiente.

1.4.1 Cambios en los parámetros predeterminados

La función *optimset* crea o actualiza una variable *options* que puede incluirse en las funciones de optimización. Los argumentos de esta función son pares de nombres de parámetros y valores de parámetros, tales como *TolX* y $1e-4$. Cualquier propiedad no especificada utilizará el valor por defecto. Tampoco se necesita escribir el nombre completo del parámetro, solamente una cantidad suficiente para determinar el nombre de forma inequívoca. El tipo de letra, mayúscula o minúscula, se ignora para los nombres de los parámetros. Sin embargo para los valores de los parámetros que son cadenas de caracteres, si hay distinción y es necesario incluir la palabra exacta.

Se puede utilizar la instrucción *help optimset* para extraer información de los diferentes parámetros utilizados y cómo utilizar la función.

A continuación se inlcuyen algunos ejemplos del uso de *optimset.*

Mostrar todos los parámetros

La función *optimset* devuelve todos los parámetros que pueden modificarse, junto con los valores típicos y los valores por defecto.

Determinación de los parámetros utilizados por una función

La estructura *options* determina los parámetros que pueden ser utilizados por las funciones de la toolbox. Como todas las funciones no utilizan todos los parámetros, puede ser bastante útil encontrar qué parámetros son utilizados por una función particular.

Para determinar que campos de la estructura *options* son utilizados por una función particular, incluimos el nombre de la función a *optimset,* por ejemplo, para *fmincon*

```
optimset('fmincon')
```

o bien

```
optimset fmincon
```

Esta instrucción devuelve una estructura. Los campos no utilizados por la función están vacíos ([]); mientras que los campos utilizados por la función contienen los valores por defecto para ella.

Presentación de resultados intermedios

Para obtener resultados intermedios en cada iteración, se utiliza

```
options = optimset('Display', 'iter');
```

Este comando modifica el campo *Display* de *options* por *'iter'*, que provoca que la toolbox presente información del proceso de optimización en cada iteración. También se puede desactivar con *'off'*, solamente obtener los datos al final del proceso con *'final'*, o proporcionar información solamente cuando en el problema no se produzca la convergencia con *'notify'*.

Optimización estándar

Aunque se puede emplear la toolbox de optimización de MATLAB, para resolver problemas especiales de los llamados *Large-Scale,* solamente utilizaremos algoritmos estándar para minimización de problemas usuales; puesto que la opción por defecto de MATLAB es la resolución de problemas a gran escala (*Large-Scale Problems*), es necesario cambiar esta opción mediante la siguiente instrucción:

```
options = optimset('LargeScale', 'off');
```

Inclusión de más de un parámetro

Se pueden modificar múltiples parámetros con una sola llamada a *optimset.* Por ejemplo, para modificar la salida y la tolerancia en el valor de $\mathbf{x}$, conjuntamente podemos utilizar la instrucción:

```
options = optimset('Display', 'iter', 'TolX', 1e-6);
```

Actualización de una estructura options

Para actualizar una estructura *options* ya existente, se puede incluir esta estructura como primer argumento de *optimset:*

```
options = optimset(options, 'Display', 'iter', 'TolX', 1e-6);
```

Obtención de los valores de los parámetros

Podemos obtener los valores de cada parámetro en la estructura *options,* mediante la función *optimget.* Por ejemplo, para obtener la opción actual de salida, introducimos

```
verbosity = optimget(options, 'Display');
```

La función *optimset* devolverá el valor del campo 'Display' y lo asignará a la variable 'verbosity'.

1.5 Optimización de objetos inline

Las funciones de la Toolbox de Optimización también realizan la optimización de objetos *inline*, de esta forma podemos evitar la creación de un fichero *M*, para definir las funciones.

Para representar una función matemática en la línea de comandos, crearemos un objeto *inline* a partir de una expresión alfanumérica. Por ejemplo, podemos crear un objeto *inline* de la función *humps* (utilizar el comando *humps* para conocer a esta función).

```
f = inline('1./((x-0.3).^2 + 0.01) + 1./((x-0.9).^2 + 0.04)-6');
```

Podemos ahora evaluar la función f en el punto 2.0

```
f(2.0)
ans =
-4.8552
```

Y se puede utilizar la función f con una rutina de optimización para obtener su mínimo:

```
x = fminbnd(f, 3, 4)
```

También es posible crear funciones de más de un argumento mediante *inline,* indicando los nombres de los argumentos de entrada dentro de la expresión alfanumérica. Por ejemplo, para utilizar *lsqcurvefit,* se necesita una función que tenga dos argumentos de entrada, x y $xdata$

```
f= inline('sin(x).*xdata +(x.^2).*cos(xdata)','x','xdata')
x = pi; xdata = pi*[4;2;3];
f(x, xdata)
ans =
9.8696e+000
9.8696e+000
-9.8696e+000
```

y después llamar a la función *lsqcurvefit*

```
% Si asumimos que la variable ydata existe
x = lsqcurvefit(f,x,xdata,ydata)
```

Aquí se presentan otro ejemplo que utiliza la misma técnica:

- Una ecuación matricial

```
x = fsolve(inline('x*x*x-[1,2;3,4]'),ones(2,2))
```

- Un problema de mínimos cuadrados

```
x = lsqnonlin(inline('x*x-[3 5;9 10]'),eye(2,2))
```

Capítulo 2

Referencia de funciones

A continuación se describen algunas funciones de la toobox de optimización y que son las utilizadas durante el curso. Se ha incluido alguna función adicional para que el lector pueda practicas con la toolbox.

2.1 Parámetros de Optimset

En esta primera sección se indican los diferentes parámetros empleados en el proceso de optimización y que pueden modificarse mediante la función *optimset*.

Parámetro	Descripción	Funciones que lo utilizan
DerivativeCheck	Compara el gradiente analítico con el numérico, calculado mediante diferencias finitas.	*fmincon, fminunc, fsolve lsqcurvefit, lsqnonlin.*
Diagnostics	Proporciona información sobre la función objetivo.	Todas salvo: *fminbnd, fminsearch, fzero y lsqnonneg.*
DiffMaxChange	Máximo cambio permitido para el cálculo de derivadas por diferencias finitas.	*fmincon, fminunc, fsolve, lsqcurvefit, lsqnonlin.*
DiffMinChange	Mínimo cambio permitido para el cálculo de derivadas por diferencias finitas.	*fmincon, fminunc, fsolve, lsqcurvefit, lsqnonlin.*
Display	Nivel de información: *'off"* no hay información; *'iter'* información en cada iteración; *'final'* iteraciónfinal; *'display'* información en caso de no convergencia.	Todas
GradConstr	Indica si se utilizan los gradientes analíticos de las restricciones. Hay que definirlos.	*fmincon.*
GradObj	Indica si se utiliza el gradiente analítico de la función objetivo. Hay que definirlo.	*fmincon, fminunc.*
HessUpdate	Tipo de algoritmo multivariante: *'bfgs'* Algoritmo BFGS; *'dfp'* Algortimo DFP; *'steepdesc'* Algoritmo del Gradiente.	*fminunc.*

Parámetro	Descripción	Funciones que lo utilizan
LargeScale	*'on':* Utiliza algoritmos large-scale. *'off':* Utiliza algoritmos estándar.	*fmincon, fminunc, fsolve, linprog, lsqcurvefit, lsqlin, lsqnonlin, quadprog.*
LevenbergMarquardt	*'on'* Algoritmo de Levenberg-Marquardt. *'off'* Algoritmo de Gauss-Newton.	*fsolve, lsqcurvefit, lsqnonlin.*
LineSearchType	*'quadcubic'* Búsqueda lineal cuadrático-cúbica. *'cubicpoly'* Búsqueda lineal cúbica.	*fminunc, fsolve, lsqcurvefit, lsqnonlin.*
MaxFunEvals	Máximo número de evaluaciones de la función permitido.	*fminbnd, fmincon, fminsearch, fminunc,fsolve, lsqcurvefit, lsqnonlin.*
MaxIter	Máximo número de iteraciones permitido.	Todas salvo *fzero* y *lsqnonneg.*
TolCon	Máximo error permitido en el incumplimiento de las restricciones.	*fmincon.*
TolFun	Criterio de terminación para la función objetivo.	*fmincon, fminsearch, fminunc, fsolve, lsqcurvefit, lsqnonlin.*
TolX	Criterio de terminación en $\mathbf{x}$.	Todas salvo: *linprog, lsqlin* y *quadprog.*

2.2 Descripción de las funciones utilizadas

En esta sección se describe particularmente cada función de la toolbox, utilizada en el curso.

2.2.1 fminbnd

- **Objetivo:** Encontrar el mínimo de una función de una variable dentro de un intervalo prefijado, es decir, busca soluciones al problema.

$$\begin{array}{rc} \text{Minimizar} & f(x) \\ & x_1 < x < x_2 \end{array}$$

  donde x, x_1 y x_2 son reales y $f(x)$ es una función que devuelve un número real.

- **Sintáxis:**

  Forma simple: *sol = fminbnd(fun,x1,x2)*

  Forma Completa: *[sol,fsol,exitflag,output] = fminbnd(fun,x1,x2,opciones,P1,P2,...)*

- **Argumentos:**

  Argumentos de Entrada:

  fun función objetivo del problema, es una función que acepta un valor real x y devuelve otro valor f, que es el valor de la función en x. Puede ser un fichero *.m* que defina la función, en este caso se utiliza el código

  x=fminbnd(@fun,....)

  o puede ser un objeto *inline.*

Vemos a continuación un ejemplo de fichero .m

function f=nombre($\mathbf{x}$*)*
% Definimos la función $x^2 + 2x + 1$

f = x^2 +2*x+1*;*

y la misma función definida mediante un objeto *inline*

*fun = inline('*x^2 +2* x+1*')*

Para un objeto *inline* solamente es necesario el nombre de ese objeto

x=fminbnd(fun,....)

x1,x2 son los extremos del intervalo.

opciones Estructura que contiene los parámetros del proceso de optimización. Las opciones aceptadas por *fminbnd* son: *Display, MaxFunEvals, MaxIter* y *TolX.*.

P1,P2,... son argumentos que pueden pasarse a la función.

Argumentos de Salida:

sol es la solución del problema, es decir, es el mínimo de $f(x)$ en el intervalo $]x_1, x_2[$ (siempre que haya convergencia).

fsol es el valor de $f(x)$ en *sol.*

exitflag es un código de salida de la función que puede tomar los siguientes valores:

1. 1 el algoritmo converge.
2. 0 Se han superado el número de evaluaciones de $f(x)$ permitidas.
3. -1 el algoritmo diverge

output estructura que contiene información respecto al proceso de optimización.

- **Observaciones:** El algoritmo utilizado por *fminbnd* está basado en el método de la sección áurea y en la interpolación parabólica. La función objetivo debe ser continua. *fminbnd* solamente encuentra mínimos locales.

2.2.2 fmincon

- **Objetivo:** Encontrar el mínimo de una función de varias variables con restricciones. Busca soluciones al problema.

$$\begin{array}{cc} \text{Minimizar} & f(\mathbf{x}) \\ \text{Sujeto a} & \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_e\mathbf{x} = \mathbf{b}_e \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \end{array}$$

donde $\mathbf{x}$, $\mathbf{b}$, $\mathbf{b}_e$, $\mathbf{lb}$ y $\mathbf{ub}$ son vectores, $\mathbf{A}$ y $\mathbf{A}_e$ son matrices, $\mathbf{g}(\mathbf{x})$ y $\mathbf{h}(\mathbf{x})$ son funciones vectoriales, de dimensiones p y m, respectivamente y $f(\mathbf{x})$ es una función real multivariante. Las funciones $f(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$ y $\mathbf{h}(\mathbf{x})$ pueden ser funciones no lineales.

- **Sintáxis:**

Forma simple: *sol = fmincon(fun,x0,A,b)*

Forma Completa: *[sol,fsol,exitflag,output,lambda,grad,hess] = ...*
fmincon(fun,x0,A,b,A_e,b_e,lb,ub,nonlcon,opciones,P1,P2,...)

- **Argumentos:**

Argumentos de Entrada:

fun función objetivo del problema, es una función que acepta un vector real **x** y devuelve otro valor f, que es el valor de la función en **x**. Puede ser un fichero *.m* que defina la función, en este caso se utiliza el código

x=fmincon(@fun,....)

Vemos a continuación un ejemplo de fichero .m

*function f=nombre(***x***)*
% Definimos la función $x^2 + y^2$

f = x(1)^2 +x(2)^2+1*;*

y la misma función definida mediante un objeto *inline*

*fun = inline('*x(1)^2 + x(2)^2*')*

de nuevo en este caso solamente es necesaria el nombre del objeto

x=fmincon(fun,....)

Si el gradiente de *fun* puede calcularse, y la opción *GradObj* se activa mediante la instrucción

opciones = *optimset('GradObj','on')*

entonces la función *fun,* debe suministrar, en el segundo argumento de salida, el valor del gradiente *gf*, en el vector **x**. Hay que observar que una comprobación del número de argumentos que devuelve la función, valor que se obtiene en la variable *nargout,* podemos evitar el cálculo de este gradiente cuando la función *fun* se utiliza con un sólo argumento de salida (cuando el algoritmo solamente necesite el valor de *f,* pero no el de *gf.* Por ejemplo

*function [f,gf]=nombre(***x***)*

% Definimos la función $x^2 + y^2$
f = x(1)^2 +x(2)^2+1*;*

% Definimos el gradiente $[2x, 2y]$,
% solamente cuando sea necesario:
if nargout>1
*gf = [2*x(1), 2*x(2)];*
end

x0 es la aproximación inicial de la solución o punto de partida del algoritmo.

A,b Matriz de coeficientes y vector de términos independientes asociados a las restricciones lineales de $\leq$.

A_e,b_e Matriz de coeficientes y vector de términos independientes asociados a las restricciones lineales de $=$.

lb,ub vector de cotas inferiores y superiores asociadas a las variables del problema de forma que

$$lb(i) \leq x(i) \leq ub(i)$$

estos valores pueden ser $-\inf$ y $+\inf$, cuando la componente correspondiente no está acotada inferior o superiormente respectivamente.

nonlcon es la función que calcula las restricciones no lineales del problema en un punto. Es una función que acepta un vector real $\mathbf{x}$ y devuelve dos vectores $[rd, ri]$, que son los valores de las restricciones de desigualdad, $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$, e igualdad, $\mathbf{h}(\mathbf{x}) = \mathbf{0}$, no lineales del problema en $\mathbf{x}$. El vector rd contiene las funciones de desigualdad evaluadas en $\mathbf{x}$, será por tanto un vector p–dimensional; mientras que ri, contiene el valor de las igualdades evaluadas en $\mathbf{x}$, y será por tanto un vector m–dimensional

Es un fichero *.m* que define las restricciones y en este caso se utiliza el código

x=fmincon(....,@nonlcon,...)

Por ejemplo

```
function [rd,ri]=restricc(x)

% Definimos las restricciones de desigualdad
% x^2 + yx ≤ 1
% x^3 + y ≤ 2

rd = [ x(1)^2 + x(1)*x(2) - 1, x(1)^3+x(2) -2]

% Notar el paso del término independiente al primer miembro
% En caso de no existir restricciones de este tipo
% hay que utilizar una matriz vacía []
% rd = []

% Definimos las restricciones de igualdad
% x^2 + y^2 = 9
%
ri = x(1)^2 + x(2)^2-9;

% De nuevo en caso de no existir restricciones de este tipo
% hay que utilizar una matriz vacía []
% ri = []
```

Si el gradiente de las restricciones puede calcularse, y la opción *GradConstr* se activa mediante la instrucción

opciones = *optimset('GradConstr','on')*

entonces la función *restricc,* debe suministrar, en el tercer y cuarto argumentos de salida, el valor de los gradientes *grd* y *gri*, en el vector $\mathbf{x}$. Hay que observar que una comprobación del número de argumentos que devuelve la función, valor que se obtiene en la variable *nargout,* podemos evitar el cálculo de estos gradientes cuando la función *restricc* se utiliza con dos argumentos de salida (cuando el algoritmo solamente necesite el valor de las restricciones, pero no de sus gradientes). Hay que tener en cuenta que tanto *Grd* como *Gri* son matrices cuyas componentes se definen como

Grd(i,j) es la derivada respecto a x_i *de la función* $g_j(x)$

Gri(i,j) es la derivada respecto a x_i *de la función* $h_j(x)$

Por ejemplo

```
function [rd,ri,Grd,Gri]=restricc(x)

% Definimos las restricciones de desigualdad
% x^2 + yx ≤ 1
% x^3 + y ≤ 2

rd = [ x(1)^2 + x(1)*x(2) - 1, x(1)^3+x(2) -2]

% Definimos las restricciones de igualdad
% x^2 + y^2 = 9
%
ri = x(1)^2 + x(2)^2-9;

% Definimos los gradientes
% solamente cuando sea necesario:
if nargout>2
Grd = [2*x(1) + x(2), 3*x(1)^2
x(1), 1];

Gri = [2*x(1)
 2*x(2)];

end
% Notar que en cada columna se encuentra el
% gradiente de la restricción correspondiente
```

opciones Es una estructura que contiene los parámetros del proceso de optimización. Las opciones que se utilizarán para *fmincon* son: *DerivativeCheck, Diagnostics, DiffMaxChange, DiffMinChange, Display, GradObj, GradConstr, LargeScale, MaxFunEvals, MaxIter, TolCon, TolFun, TolX.*

P1,P2,... son argumentos que pueden pasarse a la función.

Argumentos de Salida:

sol es la solución del problema obtenida a partir del punto inicial *x0*. Es un mínimo local de $f(\mathbf{x})$, siempre que haya convergencia.

fsol es el valor de $f(\mathbf{x})$ en *sol.*

exitflag es un código de salida de la función que puede tomar los siguientes valores:

1. 1 el algoritmo converge.
2. 0 Se han superado el número de evaluaciones de $f(\mathbf{x})$ permitidas.
3. -1 el algoritmo diverge

output estructura que contiene información respecto al proceso de optimización.

lambda Es una estructura que contiene los valores de los multiplicadores de Lagrange y Karush-Kuhn-Tucker de cada restricción. Los campos de esta estructura son:

1. *lower* Asociados al vector *lb.*
2. *upper* Asociados al vector *ub.*
3. *ineqlin* Asociadaos a las restricciones lineales de desigualdad.
4. *eqlin* Asociados a las restricciones lineales de igualdad.
5. *ineqnonlin* Asociados a las restricciones no lineales de desigualdad.
6. *eqnonlin* Asociados a las restricciones no lineales de igualdad.

grad Variable que contiene el valor del gradiente de $f(\mathbf{x})$ en *sol.*

hess Variable que contiene el valor del hessiano de $f(\mathbf{x})$ en *sol.*

2.2.3 fminsearch

- **Objetivo:** Encontrar el mínimo de una función de varias variables a partir de una aproximación inicial, es decir, busca soluciones al problema.

$$\begin{array}{c}\text{Minimizar} \\ \mathbf{x} \in \mathbb{R}^n\end{array} \quad f(\mathbf{x})$$

- **Sintáxis:**

  Forma simple: *sol = fminsearch(fun,x0)*

  Forma Completa: *[sol,fsol,exitflag,output] = fminsearch(fun,x0,opciones,P1,P2,...)*

- **Argumentos:**

  Argumentos de Entrada:

  fun función objetivo del problema, es una función que acepta un vector $\mathbf{x}$ y devuelve otro valor f, que es el valor de la función en $\mathbf{x}$. Puede ser un fichero *.m* que defina la función, en este caso se utiliza el código

  x=fminsearch(@fun,....)

  o puede ser un objeto *inline.*

  Vemos a continuación un ejemplo de fichero .m

  function f=fun($\mathbf{x}$*)*
  % Definimos la función $x^2 + y^2$

  fun = x(1)^2 + x(2)^2*;*

  y la misma función definida mediante un objeto *inline*

  *fun = inline('*x(1)^2 + x(2)^2*')*

  Para un objeto *inline* solamente es necesario el nombre de ese objeto

  x=fminsearch(fun,....)

  x0 es la aproximación inicial o punto de partida del algoritmo.

  opciones Estructura que contiene los parámetros del proceso de optimización. Las opciones aceptadas por *fminbnd* son: *Display, MaxFunEvals, MaxIter, TolFun* y *TolX.*

  P1,P2,... son argumentos que pueden pasarse a la función.

  Argumentos de Salida:

  sol es la solución del problema obtenida a partir del punto inicial *x0.* Es un mínimo local de $f(\mathbf{x})$, siempre que haya convergencia.

  fsol es el valor de $f(\mathbf{x})$ en *sol.*

  exitflag es un código de salida de la función que puede tomar los siguientes valores:

  1. 1 el algoritmo converge.
  2. 0 Se han superado el número de evaluaciones de $f(\mathbf{x})$ permitidas.
  3. -1 el algoritmo diverge

  output estructura que contiene información respecto al proceso de optimización.

- **Observaciones:** El algoritmo utilizado por *fminsearch* está basado en el método SIMPLEX o S^2, en la versión de Nelder-Mead, que utiliza simplex no regulares en cada iteración (con expansiones y contracciones). La función objetivo debe ser continua. *fminbnd* solamente encuentra mínimos locales. Es menos efectiva que *fminunc* para problemas de orden mayor que dos, pero funciona mejor para funciones con discontinuidades.

2.2.4 fminunc

- **Objetivo:** Encontrar el mínimo de una función de varias variables a partir de una aproximación inicial, es decir, busca soluciones al problema.

$$\begin{array}{c}\text{Minimizar} \quad f(\mathbf{x}) \\ \mathbf{x} \in \mathbb{R}^n\end{array}$$

- **Sintáxis:**

  Forma simple: *sol = fminunc(fun,x0)*

  Forma Completa: *[sol,fsol,exitflag,output,grad,hessian] = fminunc(fun,x0,opciones,P1,P2,...)*

- **Argumentos:**

  Argumentos de Entrada:

  fun función objetivo del problema, es una función que acepta un vector **x** y devuelve otro valor f, que es el valor de la función en **x**. Puede ser un fichero *.m* que defina la función, en este caso se utiliza el código

  x=fminunc(@fun,....)

  o puede ser un objeto *inline.*

  Vemos a continuación un ejemplo de fichero .m

  *function f=fun(***x***)*
  % Definimos la función $x^2 + y^2$

  fun = x(1)^2 + x(2)^2*;*

  y la misma función definida mediante un objeto *inline*

  *fun = inline('*x(1)^2 + x(2)^2*')*

  Para un objeto *inline* solamente es necesario el nombre de ese objeto

  x=fminunc(fun,....)

  Si el gradiente de *fun* puede calcularse, y la opción *GradObj* se activa mediante la instrucción

  opciones = *optimset('GradObj','on')*

  entonces la función *fun,* debe suministrar, en el segundo argumento de salida, el valor del gradiente *gf*, en el vector **x**. Hay que observar que una comprobación del número de argumentos que devuelve la función, valor que se obtiene en la variable *nargout,* podemos evitar el cálculo de este gradiente cuando la función *fun* se utiliza con un sólo argumento de salida (cuando el algoritmo solamente necesite el valor de *f,* pero no el de *gf.* Por ejemplo

  *function [f,gf]=nombre(***x***)*

  % Definimos la función $x^2 + y^2$
  f = x(1)^2 +x(2)^2+1*;*

  % Definimos el gradiente $[2x, 2y]$,
  % solamente cuando sea necesario:
  if nargout>1
  *gf = [2*x(1), 2*x(2)];*
  end

x0 es la aproximación inicial o punto de partida del algoritmo.

opciones Estructura que contiene los parámetros del proceso de optimización. Las opciones aceptadas por *fminunc* son: *DerivativeCheck, Diagnostics, DiffMaxChange, DiffMinChange, LineSearchType, Display, GradObj, HessUpdate, LargeScale, MaxFunEvals, MaxIter, TolFun* y *TolX.*

P1,P2,... son argumentos que pueden pasarse a la función.

Argumentos de Salida:

sol es la solución del problema obtenida a partir del punto inicial *x0.* Es un mínimo local de $f(\mathbf{x})$, siempre que haya convergencia.

fsol es el valor de $f(\mathbf{x})$ en *sol.*

exitflag es un código de salida de la función que puede tomar los siguientes valores:

1. 1 el algoritmo converge.
2. 0 Se han superado el número de evaluaciones de $f(\mathbf{x})$ permitidas.
3. -1 el algoritmo diverge

output estructura que contiene información respecto al proceso de optimización.

grad gradiente de la función objetivo $f(\mathbf{x})$ en el punto solución *sol.*

hessian matriz hessiana de la función objetivo $f(\mathbf{x})$ en el punto solución *sol.*

- **Observaciones:** La función objetivo debe ser continua. *fminunc* solamente encuentra mínimos locales. Utilizar *fminsearch* para funciones con discontinuidades.

2.2.5 linprog

- **Objetivo:** Encontrar el mínimo de un problema lineal. Busca soluciones al problema.

$$\begin{array}{rc} \text{Minimizar} & \mathbf{c}^T\mathbf{x} \\ \text{Sujeto a} & \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_e\mathbf{x} = \mathbf{b}_e \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \end{array}$$

donde **c, x**, **b**, $\mathbf{b}_e$, **lb** y **ub** son vectores, **A** y $\mathbf{A}_e$ son matrices.

- **Sintáxis:**

  Forma simple: *sol = linprog(c,A,b,A_e,b_e)*

  Forma Completa: *[sol,fsol,exitflag,output,lambda] = linprog(c„A,b,A_e,b_e,lb,ub,x0,opciones)*

- **Argumentos:**

  Argumentos de Entrada:

  c vector de coeficientes de la función objetivo del problema.

  A,b Matriz de coeficientes y vector de términos independientes asociados a las restricciones lineales de $\leq$.

  A_e,b_e Matriz de coeficientes y vector de términos independientes asociados a las restricciones lineales de $=$.

  lb,ub vector de cotas inferiores y superiores asociadas a las variables del problema de forma que

  $$lb(i) \leq x(i) \leq ub(i)$$

  estos valores pueden ser $-\inf$ y $+\inf$, cuando la componente correspondiente no está acotada inferior o superiormente respectivamente.

x0 es la aproximación inicial de la solución o punto de partida del algoritmo.

opciones Es una estructura que contiene los parámetros del proceso de optimización. Las opciones que se utilizarán para *linprog* son: *Diagnostics, Display, LargeScale, MaxIter.*

Argumentos de Salida:

sol es una solución del problema, es decir, es un mínimo global de $\mathbf{c}^T\mathbf{x}$ (siempre que el problema tenga solución, es decir sea factible y acotado)

fsol es el valor óptimo del problema lineal

exitflag es un código de salida de la función que puede tomar los siguientes valores:

1. 1 el algoritmo converge.
2. 0 Se han superado el número de evaluaciones de $f(\mathbf{x})$ permitidas.
3. -1 el algoritmo diverge

output estructura que contiene información respecto al proceso de optimización.

lambda Es una estructura que contiene los valores de los multiplicadores de Lagrange y Karush-Kuhn-Tucker de cada restricción. Los campos de esta estructura son:

1. *lower* Asociados al vector *lb.*
2. *upper* Asociados al vector *ub.*
3. *ineqlin* Asociadaos a las restricciones lineales de desigualdad.
4. *eqlin* Asociados a las restricciones lineales de igualdad.

2.2.6 lsqnonlin

- **Objetivo:** Encontrar el mínimo de una función de varias variables sin restricciones del tipo suma de cuadrados, es decir, *lsqnonlin* se utiliza para resolver problemas del tipo

$$\begin{array}{ll} \text{Minimizar} & F(\mathbf{x}) = f_1^2(\mathbf{x}) + f_2^2(\mathbf{x}) + \cdots + f_m^2(\mathbf{x}) \\ \text{Sujeto a} & \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \end{array}$$

donde $\mathbf{x}$, $\mathbf{lb}$ y $\mathbf{ub}$ son vectores, $f_j(\mathbf{x})$ son funciones reales multivariantes.

- **Sintáxis:**

  Forma simple: *sol = lsqnonlin(fun,x0)*

  Forma Completa: *[sol,resnorm,residuos,exitflag,output,lambda,jacobiano] = ... lsqnonlin(fun,x0,lb,ub,opciones,P1,P2,...)*

- **Argumentos:**

  Argumentos de Entrada:

  fun En lugar de calcular el valor de $F(\mathbf{x})$ en el vector $\mathbf{x}$ (la suma de los cuadrados), la función *lsqnonlin* utiliza como función definida por el usuario un fichero que devuelva el vector formado por las funciones $f_j(\mathbf{x})$ que definen a $F(\mathbf{x})$, es decir, la función *fun* debe suministrar un vector de la forma

$$[f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})]$$

  La función *fun* puede definirse mediante un fichero y usada posteriormente mediante la sintáxis

x = lsqnonlin(@fun,....)

  por ejemplo si

$$F(\mathbf{x}) = x^2 + y^2$$

la función *fun* puede definirse como

function f=fun($\mathbf{x}$*)*
% Definimos la función $x^2 + y^2$

f = [x(1), x(2)]*;*

y la misma función definida mediante un objeto *inline*

fun = inline('x')

en este caso solamente es necesario el nombre del objeto

x=fmincon(fun,....)

Si podemos calcular el Jacobiano de *fun*, y la opción *Jacobian* se activa mediante la instrucción

opciones = *optimset('Jacobian','on')*

entonces la función *fun,* debe suministrar, en el segundo argumento de salida, el valor de este jacobiano *Jf*, en el vector $\mathbf{x}$. Hay que observar que una comprobación del número de argumentos que devuelve la función, valor que se obtiene en la variable *nargout,* podemos evitar el cálculo de este jacobiano cuando la función *fun* se utiliza con un sólo argumento de salida (cuando el algoritmo solamente necesite el valor de *f,* pero no el de *Jf.* Si *fun* devuelve un vector de m componentes y $\mathbf{x}$ es de dimensión n, entonces el Jacobiano *Jf,* es una matriz $m \times n$, donde *Jf(i,j)* es la derivada parcial de $f_i(\mathbf{x})$ respecto a x_j. Por ejemplo

function [f,Jf]=fun($\mathbf{x}$*)*

% Definimos la función $x^2 + y^2$
f = [x(1), x(2)]*;*

% Definimos el Jacobiando $[1, 0; 0, 1]$,
% solamente cuando sea necesario:
if nargout>1
Jf = [1, 0; 0,1];
end

x0 es la aproximación inicial de la solución o punto de partida del algoritmo.

lb,ub vector de cotas inferiores y superiores asociadas a las variables del problema de forma que

$$lb(i) \leq x(i) \leq ub(i)$$

estos valores pueden ser $-\inf$ y $+\inf$, cuando la componente correspondiente no está acotada inferior o superiormente respectivamente. Estos vectores solamente pueden utilizarse con la opción *LargeScale* activada.

opciones Es una estructura que contiene los parámetros del proceso de optimización. Las opciones que se utilizarán para *fmincon* son: *DerivativeCheck, Diagnostics, DiffMaxChange, DiffMinChange, Display, Jacobian, LargeScale, LevenbergMarquardt, LineSearchType, MaxFunEvals, MaxIter, TolFun, TolX.*

P1,P2,... son argumentos que pueden pasarse a la función.

Argumentos de Salida:

sol es la solución del problema obtenida a partir del punto inicial *x0.* Es un mínimo local de $F(\mathbf{x})$, siempre que haya convergencia.

resnorm es el valor de la función $F(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})^2$en el punto *sol.*

residuos es un vector con los valores de $f_j(\mathbf{x})$ para cada j.

exitflag es un código de salida de la función que puede tomar los siguientes valores:

1. 1 el algoritmo converge.
2. 0 Se han superado el número de evaluaciones de $f(\mathbf{x})$ permitidas.
3. -1 el algoritmo diverge

output estructura que contiene información respecto al proceso de optimización.

lambda Es una estructura que contiene los valores de los multiplicadores de Lagrange y Karush-Kuhn-Tucker de cada restricción. Los campos de esta estructura son:

1. *lower* Asociados al vector *lb.*
2. *upper* Asociados al vector *ub.*

jacobiano Variable que contiene el valor del Jacobiano de $F(\mathbf{x})$ en *sol.*

- **Observaciones:** La función objetivo debe ser continua. *lsqnonlin* solamente encuentra mínimos locales. Utilizar *fminsearch* para funciones con discontinuidades.

2.2.7 optimget

- **Objetivo:**Función utilizada para extraer los valores de los parámetros en una estructura de opciones de optimización.

- **Sintáxis:**

  Forma simple: *val = optimget(opciones,'param')*

  Forma Completa: *val = optimget(opciones,'param', default)*

- **Argumentos:**

  Argumentos de Entrada:

  opciones: La estructura con las opciones utilizadas en el proceso de optimización.

  'param': Es el parámetro cuyo valor queremos conocer.

  default: En caso de que el parámetro *'param'* no tenga asignado un valor dentro de la estructura *opciones,* la función *optimget* devolverá su valor predeterminado.

  Argumentos de Salida:

  val: Valor del parámetro *'param'* dentro de la estructura *opciones.*

- **Ejemplos:** Con la siguiente instrucciones *optimget* devuelve el valor de la opción de minimización *Display* dentro de la estructura de optimización *mis_ opciones*

  val = optimget(mis_ opciones, 'Display')

  Mientras que con esta otra instrucción, la función *optimget* devuelve el valor de la opción *'Display'* dentro de la estructura de optimización *mis_ opciones,* (como en el ejemplo anterior), pero si el parámetro *Display,* no está definido, devuelve el valor *'final'.*

  val = optimget(mis_ opciones, 'Display','final')

2.2.8 optimset

- **Objetivo:**Función utilizada para crear o modificar los valores de los parámetros en una estructura de opciones de optimización.
- **Sintáxis y argumentos:**

opciones = optimset('param1',valor1,'param2',valor2,...)
Crea la estructura de optimización *opciones,* en la que los parámetros especificados (*parma1, parma2,...*) toman los valores indicados (*valor1, valor2,...*). Cualquier parámetro no especificado se igualará a *[]*, este valor indica que la función de optimización empleada, usará el valor predeterminado para ese parámetro. No es necesario escribir el nombre completo del parámetro, es suficiente con especificar solamente, el número de caracteres necesarios para distinguir al parámetro de forma unívoca. Además no existe diferencia entre mayúsculas y minúsculas.

optimset
Muestra la lista completa de parámetros de optimización y sus valores válidos.

opciones = optimset
Crea la estructura de optimización *opciones,* en la que todos los parámetros toman el valor *[].*

opciones = optimset(optimfun)
Crea la estructura de optimización *opciones,* con todos los parámetros y valores predeterminados relevantes para la función de optimización *optimfun.*

opciones = optimset(opciones_antiguas,'param_1',valor_2, 'param_2','valor_2'...)
Crea una copia de la estructura *opciones_antiguas* y modifica los parámetros *'param_k'* con los valores indicados en *'valor_k'*

opciones = optimset(opciones_antiguas,opciones_nuevas)
Combina una estructua de opciones existente, *opciones_antiguas,* con una estructura nueva *'opciones_nuevas'.* Cualquier parámetro en *opciones_nuevas* con valor no vacío sobrescribe el correspondiente parámetro en *opciones_antiguas.*

2.2.9 quadprog

- **Objetivo:** Encontrar el mínimo de un problema cuadrático. Busca soluciones al problema.

$$\begin{array}{rc} \text{Minimizar} & \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{Sujeto a} & \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{A}_e\mathbf{x} = \mathbf{b}_e \\ & \mathbf{lb} \leq \mathbf{x} \leq \mathbf{ub} \end{array}$$

donde **c, x**, **b**, $\mathbf{b}_e$, **lb** y **ub** son vectores, **A** y $\mathbf{A}_e$ son matrices y **H** es una matriz cuadrada.

- **Sintáxis:**

Forma simple: *sol = quadprog(H,c,A,b)*

Forma Completa: *[sol,fsol,exitflag,output,lambda] = quadprog(c„A,b,A_e,b_e,lb,ub,x0,opciones,P1,P2,..)*

- **Argumentos:**

Argumentos de Entrada:

H matriz cuadrada y simétrica, coincide con el hessiano de la función.

c vector de coeficientes de la función objetivo del problema.

A,b Matriz de coeficientes y vector de términos independientes asociados a las restricciones lineales de $\leq$.

A_e,b_e Matriz de coeficientes y vector de términos independientes asociados a las restricciones lineales de =.

lb,ub vector de cotas inferiores y superiores asociadas a las variables del problema de forma que

$$lb(i) \leq x(i) \leq ub(i)$$

estos valores pueden ser $-\inf$ y $+\inf$, cuando la componente correspondiente no está acotada inferior o superiormente respectivamente.

x0 es la aproximación inicial de la solución o punto de partida del algoritmo.

opciones Es una estructura que contiene los parámetros del proceso de optimización. Las opciones que se utilizarán para *linprog* son: *Diagnostics, Display, LargeScale, MaxIter.*

P1,P2,... son argumentos que pueden pasarse a la función.

Argumentos de Salida:

sol es la solución del problema, es decir, es el mínimo de $\frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x}$ (siempre que haya convergencia)

fsol es el valor óptimo del problema lineal

exitflag es un código de salida de la función que puede tomar los siguientes valores:

1. 1 el algoritmo converge.
2. 0 Se han superado el número de evaluaciones de $f(\mathbf{x})$ permitidas.
3. -1 el algoritmo diverge

output estructura que contiene información respecto al proceso de optimización.

lambda Es una estructura que contiene los valores de los multiplicadores de Lagrange y Karush-Kuhn-Tucker de cada restricción. Los campos de esta estructura son:

1. *lower* Asociados al vector *lb.*
2. *upper* Asociados al vector *ub.*
3. *ineqlin* Asociados a las restricciones lineales de desigualdad.
4. *eqlin* Asociados a las restricciones lineales de igualdad.

- **Ejemplo:**

Encuentra los valores de $\mathbf{x}$ que resuelven el siguiente problema

$$\begin{array}{rc} \text{Minimizar} & \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2 \\ \text{Sujeto a} & \\ & x_1 + x_2 \leq 2 \\ & -x_1 + 2x_2 \leq 2 \\ & 2x_1 + x_2 \leq 3 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{array}$$

en notación matricial tendremos

$$\begin{array}{rc} \text{Minimizar} & \frac{1}{2}(x_1, x_2)\begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [-2, -6]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \text{Sujeto a} & \\ & x_1 + x_2 \leq 2 \\ & -x_1 + 2x_2 \leq 2 \\ & 2x_1 + x_2 \leq 3 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{array}$$

Definimos los vectores correspondientes:

H = [1-1;-12]
c = [-2;-6]
A = [1 1; -1 2; 2 1]
b = [2; 2; 3]
lb = zeros(2,1)

a continuación utilizamos la función *quadprog*, mediante la istrucción

[x,fval,exitflag,output,lambda]=quadprog(H,f,A,b,[],[],lb)

que conduce a la solución

x = [0.6667, 1.3333]
fval = -8.2222
exitflag = 1
output: iterations:3;
algorithms: 'medium-scale: active-set'
firstorderopt : []
cgiterations : []
lambda.ineqlin = [3.1111, 0.4444, 0]
lambda.lower = [0,0]