Matlab
Toolbox de optimización
Aplicaciones en ciencias económicas

Iván Cabezas
Juan David Páez

Unidad de Informática y Comunicaciones
Facultad de Ciencias Económicas Universidad
Nacional de Colombia Bogotá DC, 2010-II

Índice general

Capítulo 1

Introducción

La inversión realizada en este software, por parte de la Facultad de Ciencias Económicas de la Universidad Nacional de Colombia, y la necesidad de dominar un programa como Matab, el cual permite el desarrollo y la difusión del conocimiento alrededor de temas de interés, justifican el hecho de hacer una investigación que permita presentar las herramientas básicas para la optimización de modelos matemáticos aplicados a la economía.

El presente documento es el resultado una investigación realizada acerca de las funciones provistas en el toolbox de optimización de Matlab, el cual pretende acercar al lector a un conjunto de aplicaciones funcionales de estas herramientas en economía. A lo largo del contenido, se presentarán los alcances del software mediante la resolución de problemas tipo[1]. Adicionalmente, los códigos utilizados en la resolución de los ejemplos presentados se adjuntan al presente trabajo, bajo un título representativo.

Se parte del hecho de que el lector maneja los temas, usados por las ciencias económicas (contaduría pública, administración de empresas y economía), que se presentan aquí y que posee bases sólidas sobre el uso básico de Matlab.

1.1. Notación

Cabe aclarar que en algunas secciones de este documento se expondrán ejemplos en los cuales se presentan las órdenes de los problemas a resolver, los cuales pueden escribirse de dos formas, en la línea de comandos o en el editor de ficheros del programa. Para distingir el primero del segundo, se anteponen un par de caracteres (`>>`) a las instrucciones, que simbolizan el prompt de la línea de comandos, siempre y cuando sea necesario.

[1] Algunos de éstos presentan la solución analítica

Capítulo 2

Matlab
Toolbox de optimización

Matlab es un lenguaje técnico computacional y un ambiente interactivo para el desarrollo de algoritmos, visualización y análisis de datos, y computación numérica. Es usado en una gran variedad de aplicaciones, como el procesamiento de señales e imágenes, comunicaciones, modelado y análisis financiero, entre otros. Se pueden añadir toolboxes, los cuales son colecciones de funciones para un propósito en especial, que extienden las funcionalidades de Matlab [1]. Uno de estos, es el toolbox de Optimización, el cual provee algoritmos para resolver diferentes tipos de problemas de optimización, con restricciones o sin ellas; los cuales incluyen programación lineal, programación cuadrática, minimización (maximización) de funciones de una o varias variables, solución de sistemas de ecuaciones no lineales, entre otros[2].

El toolbox de optimización de Matlab es un conjunto de funciones (almacenadas en m-files) que proveen algoritmos para solucionar diferentes tipos de problemas. Además, posee una interfaz gráfica que facilita a los usuarios menos experimentados usar fácilmente el toolbox, sin necesidad de hacer uso de código.

2.1. La interfaz gráfica

Para ejecutar el toolbox de optimización desde la interfaz gráfica se accede a través del menú *Start* → *Toolboxes* → *Optimization* → *Optimization tool (optimtool)*, o desde la línea de comandos `optimtool`. Desde el mismo menú es posible seleccionar la ayuda proporcionada por cada toolbox o ver ejemplos de uso del mismo. La siguiente imagen[3] muestra la ruta para acceder al entorno gráfico del toolbox de optimización.

Como se puede observar en la Fig. 2.1[4], la ventana del toolbox consta de tres partes principalmente: una para seleccionar el tipo de solver y sus parámetros principales, *Problem Setup and Results*, y donde posteriormente se mostraran los resultados del proceso de optimización; la

[1]Tomado de: http://www.mathworks.com/access/helpdesk/help/techdoc/learn_matlab/f0-14059.html
[2]Tomado de: http://www.mathworks.com/products/optimization/
[3]Tomada de: http://www.mathworks.com/help/toolbox/gads/launchoptimtool.png
[4]Tomada de: http://www.mathworks.com/help/toolbox/optim/ug/optimtool png1.png

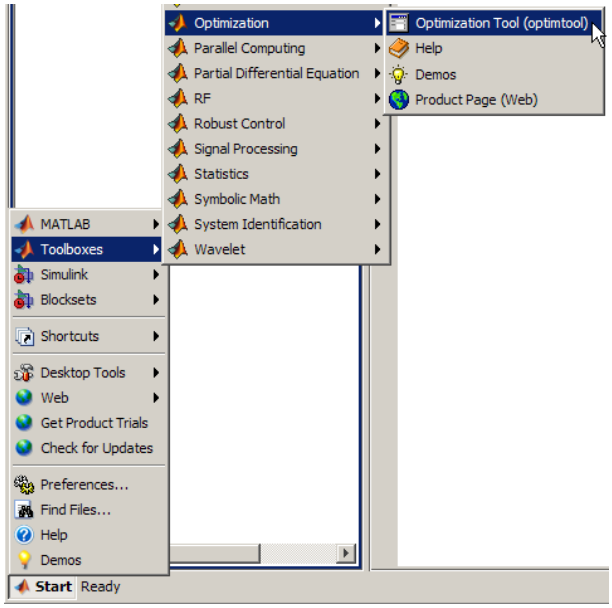

Figura 2.1: Ruta de acceso al toolbox de optimización

segunda sección, *Options*, contiene las opciones disponibles para usar en cada herramienta; y la tercera, *Quick Reference*, es un menú de ayuda.

2.2. La línea de comandos

También es posible hacer uso del toolbox desde la ventana de comandos de Matlab, llamando a las funciones de este directamente, como se hace con cualquier función o script almacenados en un m-file, pasando los respectivos parámetros y almacenando los datos de salida en los vectores correspondientes.

Sin importar el algoritmo que se va a usar siempre se debe declarar la función objetivo para que el toolbox realice su optimización. Para realizar esta tarea, existen varios métodos; uno de estos es declarar la función directamente en la línea de comandos, por ejemplo:

```
>> myfun = @[(x)] x. ^ 2 + x ;
```

Sin embargo, al declararla de esta forma, la función será borrada en futuras sesiones. Si la función objetivo es algo compleja o requiere ser usada en sesiones posteriores, es recomendable crear un script que la contenga y así llamarla desde la consola principal cuantas veces se necesite. Para hacer la llamada a esta función, se debe anteponer el caracter arroba en el lugar que ocuparía la función original. Un ejemplo de esto es:

```
function y = myfun(x)
   y = x. ^ 2 + x;        %Declaración de la función en un m-file
```

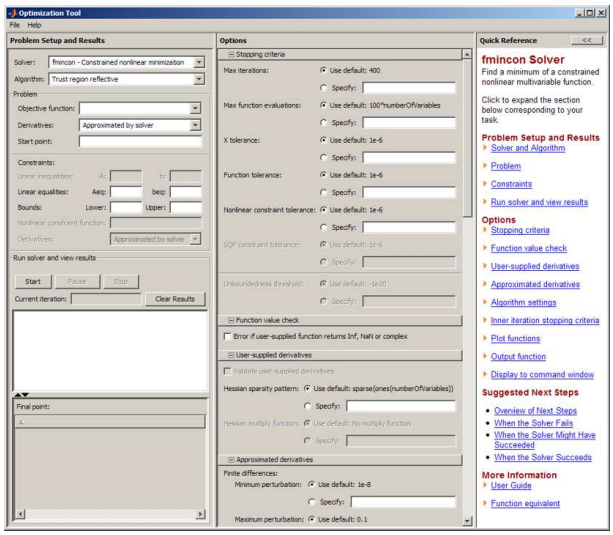

Figura 2.2: Ventana principal del toolbox de optimización

```
>> funcion(@myfun,...)    %Llamada de una funcion del solver que
                          %con la función 'myfun' como parámetro
```

La forma en la que se hace la llamada de una función del solver puede variar de acuerdo a los valores que ésta retorna. Cuando se llama como en el ejemplo anterior, el resultado es un vector que contiene la posición del punto óptimo. Si se desea conocer el valor de la función objetivo evaluada en dicho punto, la sintaxis es:

```
>> [x,fval] = funcion(@myfun,...)
```

De esta manera, el punto óptimo se almacena en la variable `x` y el valor de la función en éste, en `fval`. Dependiendo del tipo de función, ésta puede o no retornar valores adicionales, para lo cual deber escribirse:

```
>> [x,fval,a,b,c,...] = funcion(@myfun,...)
```

2.3. Aclaraciones para tener en cuenta

- Es importante tener en cuenta que el toolbox siempre busca minimizar funciones, por tanto si se requiere resolver un problema de maximización es necesario multiplicar la función objetivo por -1, de esta forma, los valores obtenidos en la minimización de esta nueva función son los equivalentes a los de la maximización de la función objetivo original.

- Cuando se requieran dar restricciones de desigualdad el toolbox siempre toma restricciones del tipo menor que ($c \leq b$). Si se requieren desigualdades de tipo mayor que $c \geq b$, se debe multiplicar por -1 así pues se deja en la forma menor que, para que pueda ser entendida por el toolbox. Por tanto tenemos que $c \geq b$ es equivalente a $-c \leq -b$.

- De ser necesario, es posible ingresar opciones adicionales; desde la interfaz gráfica del toolbox, se seleccionan las que se deseen; desde la línea decomandos, se usa la función `optimset`, especificándola en una variable o en la llamada de la función del toolbox.

```
>> options = optimset('parametro_1',valor_1,'parametro_2',valor_2,...)

>> [x,fval] = funcion(@myfun,...,optimset('Algorithm','active-set'))
```

Capítulo 3

Tipos de optimización

El toolbox de optimización consta de distintos solver que solucionan gran variedad de problemas. Con el propósito de ayudar al usuario a seleccionar el solver para un problema específico, se presentan la siguientes tablas[1] que proporcionan, a partir de los tipos de funciones objetivo y de restricciones, las funciones que deben usarse.

Constraint Type	Objective Type				
	Linear	Quadratic	Least Squares	Smooth nonlinear	Nonsmooth
None	n/a (*f* = const, or min = −∞)	quadprog, Theory, Examples	\, lsqcurvefit, lsqnonlin, Theory, Examples	fminsearch, fminunc, Theory, Examples	fminsearch, *
Bound	linprog, Theory, Examples	quadprog, Theory, Examples	lsqcurvefit, lsqlin, lsqnonlin, lsqnonneg, Theory, Examples	fminbnd, fmincon, fseminf, Theory, Examples	*
Linear	linprog, Theory, Examples	quadprog, Theory, Examples	lsqlin, Theory, Examples	fmincon, fseminf, Theory, Examples	*
General smooth	fmincon, Theory, Examples	fmincon, Theory, Examples	fmincon, Theory, Examples	fmincon, fseminf, Theory, Examples	*
Discrete	bintprog, Theory, Example				

Figura 3.1: Funciones a emplear según las características del problema

Type	Formulation	Solver
Scalar minimization	$\min_x f(x)$ such that $l < x < u$ (x is scalar)	fminbnd
Unconstrained minimization	$\min_x f(x)$	fminunc, fminsearch
Linear programming	$\min_x f^T x$ such that $A \cdot x \le b$, $Aeq \cdot x = beq$, $l \le x \le u$	linprog
Quadratic programming	$\min_x \frac{1}{2} x^T H x + c^T x$ such that $A \cdot x \le b$, $Aeq \cdot x = beq$, $l \le x \le u$	quadprog
Constrained minimization	$\min_x f(x)$ such that $c(x) \le 0$, $ceq(x) = 0$, $A \cdot x \le b$, $Aeq \cdot x = beq$, $l \le x \le u$	fmincon
Semi-infinite minimization	$\min_x f(x)$ such that $K(x,w) \le 0$ for all w, $c(x) \le 0$, $ceq(x) = 0$, $A \cdot x \le b$, $Aeq \cdot x = beq$, $l \le x \le u$	fseminf
Binary integer programming	$\min_x f^T x$ such that $A \cdot x \le b$, $Aeq \cdot x = beq$, x binary	bintprog

Figura 3.2: Funciones del toolbox clasificadas según el tipo de optimización que realizan

También se muestran un par de tablas que clasifican las funciones segúin el tipo de problema a solucionar, la primera (Fig. 3) de ellas corresponde a las funciones de minimización y la segunda

[1]Captura tomada de: http://www.mathworks.com/help/toolbox/optim/ug/brhkghv-18.html

Type	Formulation	Solver
Linear equations	$C \cdot x = d$, n equations, n variables	\ (matrix left division)
Nonlinear equation of one variable	$f(x) = 0$	fzero
Nonlinear equations	$F(x) = 0$, n equations, n variables	fsolve

Figura 3.3: Funciones del toolbox clasificadas según el tipo de optimización que realizan

(Fig. 3), solución de ecuaciones.

Las subsecciones posteriores se dedican al acercamiento de las funciones contenidas en el toolbox de optimización, de uso más frecuente, y a la solución de diversos tipos de problemas aplicados a la economía, haciendo uso de tales funciones[2].

3.1. Programación lineal

Los problemas de programación lineal consisten en optimizar una ecuación lineal que está sujeta a una serie de restricciones conformadas por desigualdades lineales. Para resolverlos el toolbox posee la función `linprog`, la cual posee tres algoritmos para su solución, el método de *larga escala*, el método *simplex* y el de *Active Set*.

La sintaxis para llamar esta función es la siguiente:

```
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

Donde:

`f`: es el vector de coeficientes de la función objetivo[3], organizado segun las variables[4].

`A,b`: corresponden a las restricciones de desigualdad, siendo el primero la matriz y el segundo el vector del lado derecho del sistema de inecuaciones $Ax \leq b$.

`Aeq,beq`: tienen el mismo tratamiento que `A` y `b`, respectivamente, teniendo en cuenta que los nuevos corresponden a un sistema de ecuaciones, en tanto que los antiguos constituían uno de inecuaciones.

`lb,ub`: son, respectivamente, los límites inferior y superior de la región donde se espera que se encuentre el punto óptimo.

`x0`: es el punto inicial para la iteración. Según el algoritmo usado es posible, o no, omitir este último.

[2]Es indispensable tener en cuenta las aclaraciones del capítulo anterior

[3]Se debe tener en cuenta que la función objetivo es lineal, de otro modo no habría forma de escribirla como vector.

[4]Por ejemplo, si la función objetivo es $3x_1 - 5x_2 + 12x_3$, el vector de coeficientes es [3 -5 12].

3.1.1. Ejercicios de aplicación

1. Un taller confecciona faldas, blusas, vestidos y abrigos para los que utiliza 2 horas, 3 horas y media, 4 horas y media y 5 horas de máquina de coser y 1, 2, 1 y 10 horas de cosido a mano para cada prenda, respectivamente. Si se dispone de 3000 horas de máquina y 2000 horas para coser a mano, y sabiendo que los beneficios obtenidos por unidad son de 6, 10, 13 y 30 u.m, respectivamente, calcular el número de prendas de cada tipo que deben confeccionarse para obtener el máximo beneficio.

2. Una empresa que se dedica a la producción de frascos de perfume, de agua de colonia y de champú utiliza tres factores productivos F_1, F_2 y F_3 disponiendo de 240, 460 y 430 unidades, respectivamente. Las cantidades de dichos factores utilizados en la producción de un frasco por cada producto se detallan en la siguiente tabla:

	Perfume	Agua de colonia	Champú
F_1	1	2	1
F_2	2	0	3
F_3	0	4	1

   Sabiendo que el precio unitario de venta del perfume es de 5 unidades monetarias, el del agua de colonia de 2 y el del champú de 3, y que se vende todo lo que se produce, calcular el beneficio máximo y el número de frascos de cada tipo que debe producir la empresa para obtenerlo.

Planteamiento general

Para poder utilizar la función `linprog`, mediante la interfaz gráfica o la línea de comandos, es necesario convertir las ecuaciones dadas a los vectores y matrices correspondientes, después de plantear todos los parámetros requeridos.

1. El planteamiento para el primer problema es:

$$\begin{aligned} \text{Maximizar:} \quad & 6x_1 + 10x_2 + 13x_3 + 30x_4 \\ \text{Sujeto a:} \quad & 2x_1 + 3{,}5x_2 + 4{,}5x_3 + 5x_4 = 3000 \\ & x_1 + 2x_2 + x_3 + 10x_4 = 2000 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

   Por lo tanto, cada argumento se define así:

   - Función objetivo[5]:

$$f = [-6 - 10 - 13 - 30]$$

   - Restricciones de desigualdad[6]:

$$A = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

[5] Obsérvese el que el vector de coeficientes de la función objetivo fue multiplicado por -1, ya que se busca maximizar la función objetivo, y las funciones del toolbox la minimizan por defecto.

[6] La matriz A ha sido multiplicada por -1 debido a las restricciones de desigualdad del tipo menor que.

- Restricciones de igualdad:

$$Aeq = \begin{bmatrix} 2 & 3{,}5 & 4{,}5 & 5 \\ 1 & 2 & 1 & 10 \end{bmatrix}, beq = \begin{bmatrix} 3000 \\ 2000 \end{bmatrix}$$

2. La formulación el segundo problema es:

$$\begin{aligned} \text{Maximizar:} \quad & 5x_1 + 2x_2 + 3x_3 \\ \text{Sujeto a:} \quad & F_1 \leq 240 \\ & F_2 \leq 460 \\ & F_3 \leq 430 \end{aligned}$$

Dado que las restricciones están en función de los factores de producción,se debe plantear el problema en función de las variables cuyos óptimos se desean encontrar, es decir, de las cantidades de frascos a producir para obtener el beneficio máximo. De esta forma se tienen las siguientes restricciones:

$$\begin{aligned} x_1 + 2x_2 + x_3 &\leq 240 \\ 2x_1 + 3x_3 &\leq 460 \\ 4x_2 + x_3 &\leq 430 \end{aligned}$$

Ahora, se plantea la función objetivo y sus restricciones, para utilizarlos en la función `linprog`:

- Función objetivo:

$$f = [-5 - 2 - 3]$$

- Restricciones de desigualdad:

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 3 \\ 0 & 4 & 1 \end{bmatrix}, b = \begin{bmatrix} 240 \\ 460 \\ 430 \end{bmatrix}$$

Planteamiento en Matlab

Es importante tener en cuenta que el software encuentra los valores óptimos que satisfacen los parámetros ingresados, los cuales son números reales; no obstante, se deben tomar precauciones cuando se require que los valores de respuesta sean positivos, para lo cual es necesario agregar restricciones adicionales o tomar como punto inicial de iteración, $x0$, unos valores que estén dentro de una región que satisfaga a plenitud tales restricciones.

1. Para definir todas las variables del primer problema, en Matlab, se debe escribir:

```
>> f = [-6 -10 -13 -30];
>> A = -eye(4);                        % matriz identidad de tamaño 4x4
>> b = [0 0 0 0];
>> Aeq = [2 3.5 4.5 5 ; 1 2 1 10];
>> beq = [3000 2000];
```

2. La asignación de los valores de las variables, correspondientes al segundo problema, se realiza de la siguiente manera:

```
>> f = [-5 -2 -3];
>> A = [1 2 1 ; 2 0 3 ; 0 4 1];
>> b = [240 460 430];
>> x0 = [0 0 0];
```

Solución usando interfaz gráfica

En *Problem Setup and Results* se introducen las variables que requiere el solver y en las opciones adicionales se conservan las que vienen por defecto. Al presionar el botón *Start* se inicia el proceso de optimización. En el visor de resultados, se pueden observar los valores óptimos de las variables y el valor máximo que alcanza la función objetivo.

1. La solución al primer ejemplo es confeccionar 500 vestidos, 150 abrigos y ninguna falda ni blusa, para que el beneficio máximo alcanzado sea de aproximadamente 11000 u.m.[7].

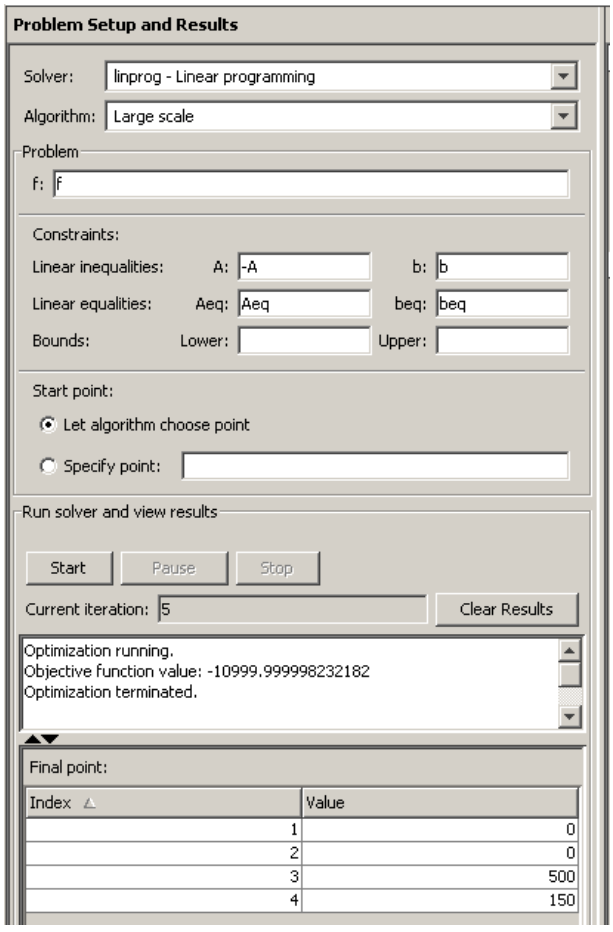

Figura 3.4: Solución al primer problema usando la interfaz gráfica

[7]El valor obtenido en el solver es negativo debido a que la función objetivo inicial fue multiplicada por -1 para lograr la maximización

2. Para que las cantidades obtenidas sean mayores que cero, se define como punto inicial de iteración $(0, 0, 0)$. Al correr el solver, se obtiene que deben producirse 230 frascos de perfume, 5 de agua de colonia y ninguno de champú y que el beneficio máximo obtenido, con estas cantidades, es de 1160 unidades monetarias.

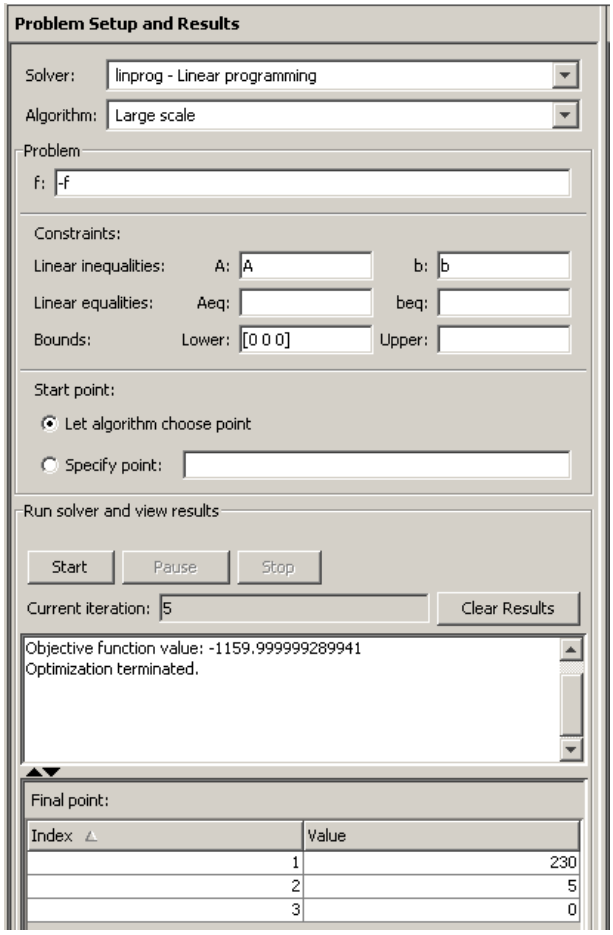

Figura 3.5: Solución al segundo problema usando la interfaz gráfica

Solución usando línea de comandos

Obviamente, se obtienen los mismos resultados solucionando el problema llamado la función `linprog` desde la línea de comandos. La ventaja de utilizar este mecanismo es que los resultados pueden utilizarse como parámetros de otras funciones o para la implementación de un programa más complejo.

1. Finalmete, se usa la sintáxis respectiva con las variables del primer problema, cargadas previamente, para obtener lo siguiente:

```
>> [x,fval] = linprog(f,A,b,Aeq,beq,lb,ub,x0)
Optimization Terminated.
```

```
x =

  0.0000
  0.0000
500.0000
150.0000

fval =

-1.1000e+004
```

2. Para el caso del segundo problema, se tiene:

```
>> [x,fval] = linprog(f,A,b,[],[],[0 0 0])
Optimization Terminated.

x =

230.0000
  5.0000
  0.0000

fval =

-1.1600e+003
```

3.2. Programación lineal binaria entera

Un problema de programación lineal binaria entera es similar a un problema de programación lineal, busca optimizar una función objetivo lineal sujeta a una serie de restricciones lineales. Sin embargo, las variables involucradas deben ser de tipo entero o binario, es decir su valor sólo puede ser uno o cero.

La sintaxis para hacer uso de la función `bintprog` es similar a la usada para la función `linprog`:

```
x = bintprog(f,A,b,Aeq,beq,x0,options)
```

Donde, de la misma manera que `linprog`:

`f`: es el vector de coeficientes de la función objetivo, organizado segun las variables.

`A,b`: corresponden a las restricciones de desigualdad, siendo el primero la matriz y el segundo el vector del lado derecho del sistema de inecuaciones $Ax \leq b$.

`Aeq,beq`: tienen el mismo tratamiento que `A` y `b`, respectivamente, teniendo en cuenta que los nuevos corresponden a un sistema de ecuaciones, en tanto que los antiguos constituyen uno de inecuaciones.

`x0`: es el punto inicial para la iteración. Según el algoritmo usado es posible, o no, omitir este último.

3.3. Funciones de una variable sin restricciones

El toolbox de optimización posee la función `fminbnd` para resolver esta clase de problemas, que encuentra el mínimo de una función de una sola variableen un intervalo dado. La sintaxis usada para llamar a esta función es:

```
x = fminbnd(fun,x1,x2,options)
```

Donde:

`fun`: es la función objetivo, declarada en un fichero *.m.

`x1,x2`: son los límites del intervalo en el que se desea encontrar el mínimo.

Esta función tiene limitaciones sobre las soluciones que retorna, ya que si la función objetivo contiene varios mínimos en la región evaluada ésta devuelve sólo uno de los mínimos; además, el solver es válido únicamente para valores en el dominio de los número reales. Esto se puede comprobar fácilmente al ingresar como funciones de prueba $sen(x)$ ó $-x^2 + 2x + 1$.

3.4. Función de impuesto inflacionario

Esta función está dada por:

$$I = xM_0e^{-\alpha x}$$

Donde:

I: es el impuesto inflacionario.

x: la tasa de inflación.

M_0: el monto inicial de dinero.

α: es la elasticidad de demanda de dinero de inflación.

Para optimizarla analíticamente, se deriva e iguala a cero y se despeja x, de donde se obtiene la tasa óptima de inflación:

$$x^* = \frac{1}{\alpha}$$

3.4.1. Ejercicio de aplicación

Se requiere encontrar la tasa de inflación óptima para un monto inicial unitario y una elasticidad de 0.2, usando el toolbox de optimización de Matlab.

Planteamiento en Matlab

Como en todas las funciones del toolbox, es necesario declarar la función objetivo para usarla en el solver. Para esto se crea un script que contiene lo siguiente:

```
function i = impuesto(x)
   alpha = 0.2;
   i = -x * exp(-alpha * x);
```

El intervalo de valores en el que se desea encontrar la tasa de inflación óptima es $[0, 100]$.

Solución usando interfaz gráfica

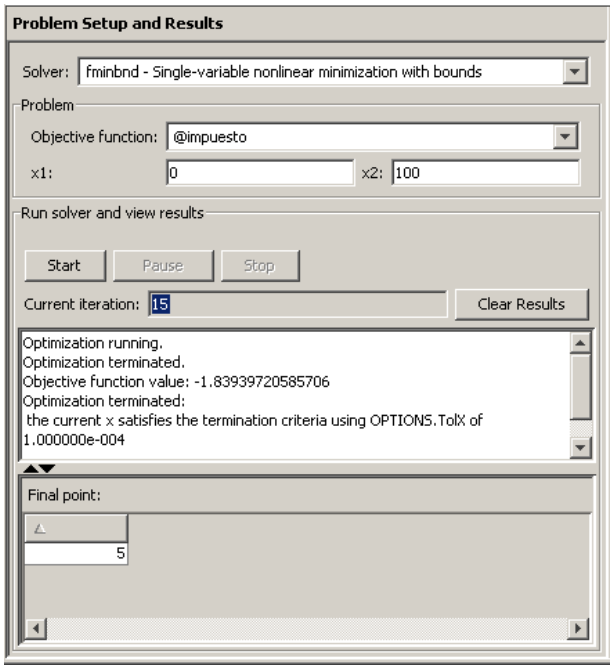

Figura 3.6: Solución al problema usando la interfaz gráfica

Como se observa, los resultados obtenidos reflejan que el valor óptimo de la tasa de inflación es 5. Adicionalmente, debido a la naturaleza del problema, es posible graficar la función, mediante la siguiente instrucción:

```
>> ezplot(@impuesto,[0 30]);
```

Solución usando línea de comandos

Introduciendo la función con sus argumentos respectivos, se obtiene:

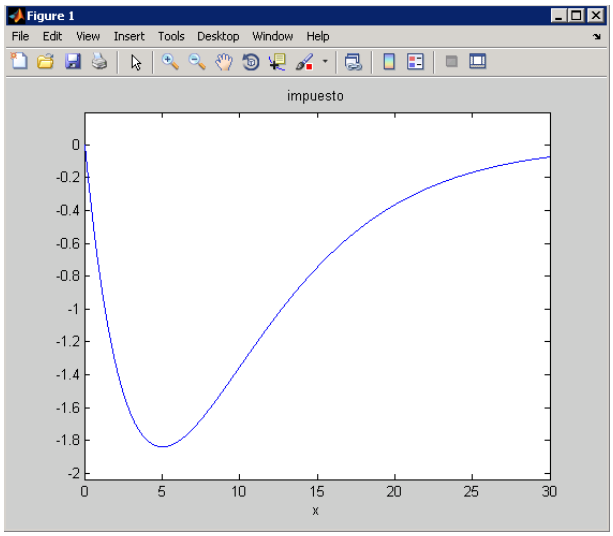

Figura 3.7: Gráfica de la función objetivo del problema

```
>> [x,fval] = fminbnd(@impuesto,0,100);

   x =

   5.0000

   fval =

   -1.8394
```

Los resultados, por ambos métodos, concuerdan con el valor óptimo que se espera tras la solución analítica:

$$x^* = \frac{1}{0{,}2} = 5$$

3.5. Modelo de crecimiento de Solow

El modelo de crecimiento de Solow se presenta como una función de producción $F(k) = Ak^{\alpha}$, la cual debe satisfacer dos condiciones:

$$\frac{dF}{dk} >= 0 \qquad \frac{d^2F}{dk^2} < 0$$

Donde:

A es el cambio técnico.

k es el capital per cápita.

α es la participación del capital en el producto.

Adicionalmente, se tiene define el cambio temporal del capital como:

$$\dot{k} = SF(k) - (n + \delta)k \tag{3.1}$$

Donde el primer término del lado derecho corresponde a la inversión de capital y el segundo a la reposición del mismo. Además:

S es la tasa de ahorro.

n es el crecimiento de la población.

δ es la tasa de depreciación.

Igualando a cero la ecuación (3.1), se puede establecer el capital óptimo en el punto donde se cumple:

$$SF(k^*) = (n + \delta)k^*$$

$$k^* = \frac{SF(k^*)}{n + \delta}$$

Finalmente, las funciones presentadas se relacionan a través de la función de consumo[8]:

$$C(k) = F(k) - (n + \delta)k \tag{3.2}$$

la cual se optimiza, derivando e igualando a cero:

$$\frac{dC(k)}{dk} = \frac{dF(k)}{dk} - (n + \delta)$$

Obteniendo una expresión conocida como *la regla de oro de Solium*:

$$\frac{dF(k)}{dk}^* = (n + \delta) \tag{3.3}$$

3.5.1. Ejercicio de aplicación

En una comunidad, la tasa de crecimiento de población es del 10 % y cuyos bienes pierden su valor a razón del 15 %. Tomando en la función de producción que se muestra acontinuación, calcular el capital promedio óptimo que minimiza el consumo en esa comunidad.

$$F(k) = k^{0,65}$$

Planteamiento en Matlab

De acuerdo a la definición de la función de consumo (Ec. 3.2), se define la función objetivo del problema como:

$$C(k) = k^{0,65} - 0{,}25k$$

A partir de esto, ésta se declara en un script para emplearla en el toolbox:

```
function c = consumo(x)
   c = x.^0.65 - 0.25 * x;
```

[8]Función que relaciona el nivel de consumo con el nivel de renta de una comunidad

Solución usando interfaz gráfica

Introduciendo los parámetros en los campos correspondientes, de forma similar al ejemplo anterior, se obtiene como capital óptimo 15,3332.

Solución usando línea de comandos

Las instrucciones respectivas para solucionar el problema dado, tomando como intervalo de búsqueda $[0, 50]$, son:

```
>> [x,fval] = fminbnd(@consumo,0,50);

   x =

   15.3333

   fval =

   2.0641
```

Nótese, que al aplicar la *regla de oro de Solium* (Ec. 3.3), se observa que la solución calculada, a través de la función del toolbox, es igual al obtenido analíticamente, obviando errores de aproximación.

3.6. Funciones multivariable sin restricciones

Para solucionar este tipo de problemas, el toolbox cuenta con dos funciones, `fminsearch` y `fminunc`. La llamada a estas funciones se realiza bajo la siguiente sintaxis:

```
x = fminsearch(fun,x0,options)
x = fminunc(fun,x0,options)
```

Donde, en forma similar a otras funciones del toolbox:

`fun`: es la función objetivo.

`x0`: es el punto inicial de iteración.

La diferencia entre `fminsearch` y `fminunc` es que la primera función usa un algoritmo que no necesita información sobre el gradiente de la función objetivo (matriz hessiana), mientras que la segunda usa uno que lo requiere, para lo cual puede especificarse manualmente o permitir que el software lo calcule por su cuenta.

3.6.1. Ejercicio de aplicación

La función de producción de una empresa es:

$$q(x_1, x_2) = 7x_1^2 + 7x_2^2 + 6x_1x_2$$

x_1 y x_2 son las cantidades utilizadas en los factores productivos y la función de costo es:

$$C(x_1, x_2) = 4x_1^3 + 4x_2^3$$

Si se sabe que el precio de venta unitario del bien producido es de 3 unidades monetarias, calcular las cantidades de los factores productivos necesarias para maximizar el beneficio de la empresa y encontrar el máximo beneficio.

3.7. Funciones multivariable con restricciones

La función `fmincon` del toolbox de optimización resuelve problemas que consisten en la minimización de una función multivariable sujeta a una serie de restricciones, ya sean de desigualdad o igualdad, lineales o no. La sintaxis usada para invocar ésta función desde la ventana de comandos o desde un fichero *.m, es la siguiente:

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options);
```

Donde:

`fun`: es un manejador (handle) de la función objetivo.

`x0`: es el punto inicial de iteración.

`A,b`: corresponden a las restricciones de desigualdad, siendo el primero la matriz y el segundo el vector del lado derecho del sistema de inecuaciones $Ax \leq b$.

`Aeq,beq`: tienen el mismo tratamiento que `A` y `b`, respectivamente, teniendo en cuenta que los nuevos corresponden a un sistema de ecuaciones, en tanto que los antiguos constituían uno de inecuaciones.

`lb,ub`: son, respectivamente, los límites inferior y superior de la región donde se espera que se encuentre el punto óptimo.

`nonlcon`: es el manejador de las restricciones no lineales, el cual retorna un vector con su valor.

3.8. Función de utilidad

Una función de utilidad es una función que mide el grado de satisfacción o utilidad que obtiene un individuo del uso de cierta cantidad de bienes y/o servicios. Se puede modelar de diferentes formas, una de ellas es la de *Cobb-Douglas*, que expresa la relación entre cantidad de productos y utilidad obtenida por un consumidor ideal, de la siguiente forma:

$$U(x, y) = x^{\alpha} y^{\beta} \tag{3.4}$$

Donde x, y y α, β representan las cantidades y el grado de utilidad obtenido de los productos x y y, respectivamente.

Generalmente, una función de utilidad está condicionada por una restricción presupuestaria, de tal forma que un consumidor no gaste más de lo asignado a bienes y servicios. Así pues, optimizar una función de utilidad consiste en encontrar las cantidades de productos que logren la máxima utilidad posible, sin gastar más de lo estimado. La formulación matemática de esto es:

$$\begin{aligned} \text{Maximizar:} \quad & U(x,y) = x^{\alpha}y^{1-\alpha} \\ \text{Sujeto a:} \quad & px + qy \leq m \end{aligned}$$

Donde x y y son las cantidades de los bienes, p y q son sus respectivos precios y m es el presupuesto del que se dispone.

Al resolver el problema planteado, usando el método de los *multiplicadores de Lagrange*, se obtiene:

$$L(x,y) = x^{\alpha}y^{1-\alpha} - \lambda(px + qy - m)$$

De donde se deriva parcialmente para obtener:

$$\alpha x^{\alpha-1}y^{1-\alpha} = \lambda p \qquad (1-\beta)x^{\alpha}y^{\alpha-1} = \lambda q \qquad px + qy - m = 0 \tag{3.5}$$

Dividiendo la primera expresión entre la segunda y reduciendo lo obtenido, se llega a:

$$\frac{\alpha}{(1-\alpha)}\frac{y}{x} = \frac{p}{q} \tag{3.6}$$

Resolviendo para cada una de las variables asociadas:

$$x = \frac{\alpha}{(1-\alpha)}\frac{q}{p}y \qquad y = \frac{(1-\alpha)}{\alpha}\frac{p}{q}x \tag{3.7}$$

Finalmente, reemplazando en la tercera expresión de (3.5), que resulta ser la restricción presupuestaria, se obtienen las cantidades óptimas (también llamadas demandas marshallianas)[9]:

$$x^{*} = \frac{\alpha}{p}m \qquad y^{*} = \frac{(1-\alpha)}{q}m \tag{3.8}$$

Ahora se pretende minimizar la restriccion presupuestal o gasto, sujeta a una utilidad dada, por lo que se recurre a un procedimiento idéntico al anterior:

$$\begin{aligned} \text{Minimizar:} \quad & m = px + qy \\ \text{Sujeto a:} \quad & u = x^{\alpha}y^{1-\alpha} \end{aligned}$$

Utilizando de nuevo el método de los *multiplicadores de Lagrange*, se obtiene:

$$p = \alpha\lambda x^{\alpha-1}y^{1-\alpha} \qquad q = \lambda(\alpha-1)x^{\alpha}y^{-\alpha} \tag{3.9}$$

Al dividir la primera y segunda ecuaciones de (3.9), se llega a:

$$\frac{p}{q} = \frac{\alpha}{(1-\alpha)}\frac{y}{x} \tag{3.10}$$

[9]Nótese la independencia mutua entre x^{*} y y^{*}.

De donde las cantidades son:

$$x = \frac{\alpha}{(1-\alpha)}\frac{q}{p}y \qquad y = \frac{(1-\alpha)}{\alpha}\frac{p}{q}x \tag{3.11}$$

Optimizándolas en forma similar, se tiene:

$$x^* = u\left(\frac{\alpha}{(1-\alpha)}\frac{q}{p}\right)^{1-\alpha} \qquad y^* = u\left(\frac{(1-\alpha)}{\alpha}\frac{p}{q}\right)^{\alpha} \tag{3.12}$$

Se han econtrado los valores óptimos que satisfacen las condiciones dadas (*demandas hicksianas*). Es posible demostrar que las demandas óptimas encontradas por ambos caminos mostrados son iguales, para un mismo problema, lo que permite demostrar que maximizar la utilidad es equivalente a minimizar el gasto.

3.8.1. Ejercicio de aplicación

Usando la siguiente función de utilidad y restricción presupuestal, encontrar las demandas óptimas.

$$\begin{aligned} \text{Maximizar:} \quad & x_1^{0,7}x_2^{0,15}x_3^{0,15} \\ \text{Sujeto a:} \quad & 3x_1 + 2x_2 + 2x_3 = 5 \end{aligned}$$

Planteamiento en Matlab

Colocando las siguientes líneas de código dentro de un fichero *.m, quedará definida la función objetivo:

```
function u = utilidad(x)
   u = -x(1) ^ 0.7 * x(2) ^ 0.15 * x(3) ^ 0.15;
```

Ahora, es el momento de definir los argumentos de la función `fmincon`:

```
>> x0 = [0 0 0];
>> Aeq = [3 2 2];
>> beq = 5;
```

Solución usando interfaz gráfica

De manera similar a los ejercicios realizados con anterioridad, se disponen las variables en los campos del formulario. Realizando el proceso de optimización, se obtienen los valores óptimos del problema, los cuales son 1.167, 0.375 y 0.375. El valor de la función, ignorando el signo negativo por razones expuestas en secciones precedentes, evaluada en el conjunto de puntos es 0.8299.

Solución usando línea de comandos

Al correr la instrucción siguiente, habiendo definido los argumentos presentados, se obtiene el siguiente resultado:

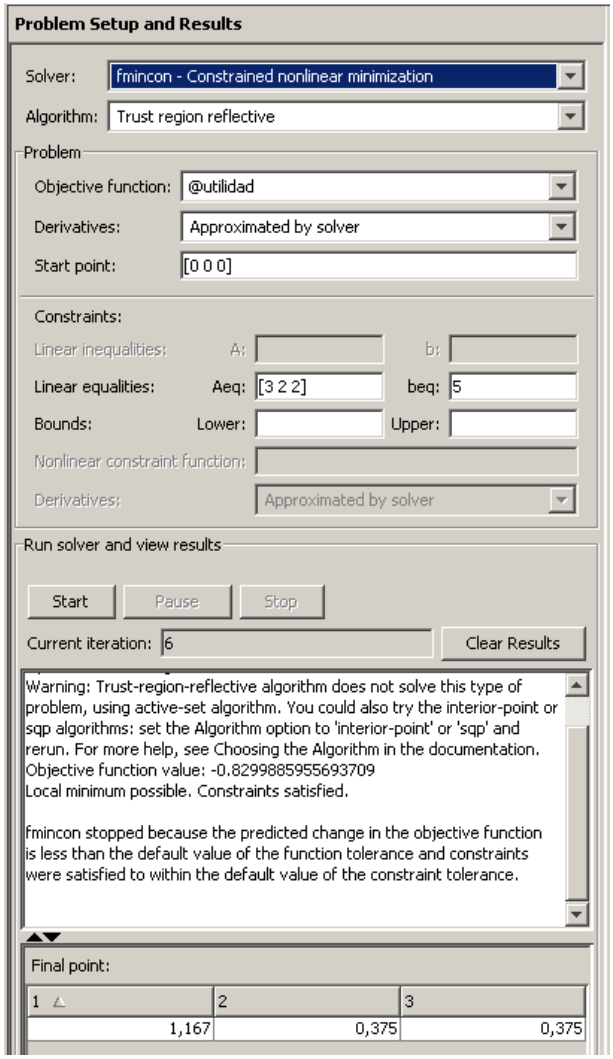

Figura 3.8: Gráfica de la función objetivo del problema

```
>> [x,fval] = fmincon(@utilidad,x0,[],[],Aeq,beq,[],[],[])
Optimization Terminated.

   x =

   1.1670
   0.3750
   0.3750

   fval =

   -0.8299
```

Si se comparan las respuestas alcanzadas por el toolbox con las calculadas analíticamente, se puede evidenciar que son exactamente iguales:

$$\begin{aligned} x_i &= \frac{\alpha m}{p_i} \\ x_1 &= 1{,}166 \\ x_{2,3} &= 0{,}375 \end{aligned}$$

Se ha construido un fichero *.m que contiene un script para realizar la optimización de funciones de utilidad[10].

[10]Se entiende que el lector está familiarizado con el uso de scripts en Matlab.