Lecciones de Equilibrio General Computable con MatLab

Gonzalo Fernández de Córdoba Martos

Esta versión 28 de julio de 2002

1 Lección segunda

Una vez que hemos entendido la lógica del algoritmo de Newton-Raphson y la aproximación numérica a las derivadas parciales nos gustaría disponer de un programa más general que resolviera estos problemas con sólo especificar el sistema de ecuaciones que queremos resolver. Esto es posible hacerlo gracias a que disponemos de rutinas muy eficientes que nos permiten resolver el problema de encontrar las soluciones a un sistema de ecuaciones llamando a esas rutinas. A continuación vamos a presentar dos de esas rutinas Urrutia (1998), una para el método de Newton, llamada **newton.m** y otra para el método de aproximación numérica a las derivadas parciales llamada **secant.m**. No son las únicas y de hecho MatLab posee en el "toolbox optim" una rutina llamada **fsolve.m** que permite resolver sistemas de ecuaciones no lineales y que incorpora tanto al método de Newton como al de aproximación numérica de las derivadas, además de poder especificar criterios de tolerancia, máximo número de iteraciones etc. La razón por la que usaremos fundamentalmente el programa **secant.m** se debe a que es fácil intervenir en este programa para obtener información sobre el proceso de optimización. Pero para entender esto, lo mejor es presentar y discutir brevemente estos programas**.** La diferencia entre el programa **newton.m** y el programa **secant.m** es simplemente que en el primero nosotros especificamos una función en la que está definido el jacobiano del sistema y en el otro hacemos una aproximación numérica a las derivadas parciales, con lo que sólo es necesario especificar en el programa el sistema de ecuaciones. Normalmente utilizaremos el programa **secant.m** dado que a medida que los problemas que nos planteemos sean más complejos no dispondremos de información sobre el jacobiano o éste será demasiado engorroso como para introducirlo dentro del programa.

```
function x = newton(func, x0, param, crit, maxit)
%newton.m      Programa para resolver un sistema de ecuaciones
%simultáneas.
%x=newton(func, x0, param, crit, maxit) usa el método de
%Newton-Raphson para resolver el siguiente sistema:
%              f1(z1, z2, ···, zn)=0
%              f2(z1, z2, ···, zn)=0
%              ⋮                              (*)
%              fn(z1, z2, ···, zn)=0
%en donde x=[z1, z2, ···, zn]' es el vector que resuelve (*).
%Para usar esta función, debe especificarse 'func' que es un "string"

%con el nombre del archivo .m que contiene la función f y su matriz
%jacobiana J. En caso necesario, puede usarse el vector
%param para incluir parámetros adicionales de dicha función.
%Los argumentos x0, crit, y maxit son la semilla inicial para x,
%el criterio de convergencia (tolerancia) y el número máximo de
%iteraciones permitidas respectivamente.
%Programa de Carlos Urrutia
for i=1:maxit
        [f, J]=feval(func, x0, param);
        x=x0-inv(J)*f;
        if norm(x-x0)<crit;      break;      end
        x0=x;
end
if i>=maxit
sprintf('Advertencia: Número máximo de %g iteraciones ...
alcanzado', maxit)
end
```

Lo primero que observamos al mirar al archivo **newton.m** es que comienza con el comando **function.** En MatLab hay dos tipos de archivo que tienen extensión **.m,** los primeros que hemos visto en los ejemplos 1 y 2, son archivos del tipo "script" y contienen una colección de comandos de MatLab que son ejecutados con sólo declarar el nombre del archivo sobre la ventana de comandos de MatLab tras el símbolo >>. Los segundos son los que estamos viendo ahora, son del tipo "function" y también contienen comandos de MatLab pero necesariamente tienen que empezar con el comando **function**. Su utilidad consiste en que dentro de ellos podemos definir nuevas funciones y lo que haremos será definir en su interior el sistema de ecuaciones (funciones) que queremos resolver. Para ver cómo funciona plantearemos un ejercicio sencillo.

1.0.1 Ejemplo3

Queremos encontrar el par (x^*, y^*) que resuelve el sistema de ecuaciones

$$\begin{aligned} ay - (x-b)(x+c) &= 0 \\ yx - d &= 0. \end{aligned}$$

Para resolver este problema usando la función **newton.m** debemos calcular el jacobiano del sistema.

$$J = \begin{bmatrix} -2x + (b-c) & a \\ y & x \end{bmatrix}$$

A continuación escribimos un programa similar a los que ya hemos escrito para resolver los ejemplos 1 y 2, con la salvedad de que ahora vamos a introducir el sistema de ecuaciones y el jacobiano en un archivo a parte y vamos a llamar a la rutina **newton.m** para que se encargue del asunto. Los pasos son los siguientes: primero construimos un archivo llamado **sistema.m** donde definimos nuestro problema, fijamos la semilla con la que el algoritmo debe empezar, fijamos el número máximo de iteraciones, fijamos el criterio de tolerancia, fijamos el valor de los parámetros y llamamos al programa **newton.m** indicando el nombre del archivo sobre el que tiene que operar y que vamos a llamar **aqui.m,** puesto que aquí es donde vamos a escribir el valor de la función y el valor del jacobiano.

```
%Archivo sistema.m
%Este programa resuelve por el método de Newton-Raphson
%el sistema de ecuaciones general
%
%ay-(x+b)*(x-c) = 0
%xy-d = 0
%
%para el caso particular
%
%y-(x+4)*(x-4) = 0
%xy-1 = 0
%
%Para ello hacemos uso de la función newton.m
clear
%Punto inicial
%x0 = [-0.001; 0.001];
%x0 = [-0.001; -2];
x0 = [-0.001; -0.001];
%Número máximo de iteraciones permitido
maxit = 1000;
%Criterio de convergencia (tolerancia)
crit = 1e-3;
%Vector de parámetros del sistema
param = [1 4 4 1];
%Llamamos a newton.m y especificamos el archivo sobre el que
debe actuar
sol = newton('aqui', x0, param, crit, maxit);
sprintf('x= %g', sol(1))
sprintf('y= %g', sol(2))
```

Ahora mostramos el contenido del archivo **aqui.m** que como podéis ver es un archivo del tipo function puesto que en él se define una función que debe ser evaluada.

```
function [f, J]=aqui(z, p)
%archivo aqui.m
x = z(1);
y = z(2);
a = p(1);
b = p(2);
c = p(3);
d = p(4);
f = [a*y-(x-b)*(x+c); y*x-d];
J = [-2*x+b-c a; y x];
```

El comando **function [f,J]=aqui(z,p)** dice que el resultado de los cálculos realizados son las matrices **f** (de dimensión 2x1) y **J** de dimensión (4x4), y que el archivo **aqui.m** tiene dos argumentos, el primero es **z** que se identifica con el valor de **x0** del archivo **sistema.m** y el segundo es **p,** que se identifica con el vector **param** del mismo archivo. Las líneas sucesivas asignan valores a las variables y a los parámetros.
Ahora podemos ver cómo funciona el programa **secant.m** en el mismo ejemplo.

1.0.2 Ejemplo 4

Las dos únicas diferencias con respecto a los programas del ejemplo 3 son que naturalmente en el programa **sistema.m** ya no vamos a llamar al programa **newton.m** sino al programa **secant.m**, y la otra diferencia es que el programa **aqui.m** ya no necesita tener especificado el jacobiano del sistema de ecuaciones puesto que la rutina **secant.m** hace una aproximación numérica al jacobiano. El programa **aqui.m** quedaría así:

```
function f=aqui(z, p)
%Programa aqui.m sin el jacobiano
x = z(1);
```

```
y = z(2);
a = p(1);
b = p(2);
c = p(3);
d = p(4);
f = [a*y-(x-b)*(x+c); y*x-d];
```

Como vemos, el valor del jacobiano no es computado en esta versión de **aqui.m**, ya que en la rutina **secant.m** que mostramos a continuación se realiza una aproximación numérica al jacobiano.

```
function x=secant(func, x0, param, crit, maxit)
%secant.m Programa para resolver un sistema de ecuaciones
%simultaneas.
% x=secant(func, x0, param, crit, maxit) usa el método de secante
% para resolver el siguiente sistema:
%
% f1(z1,z2, ...,zn)=0
% f2(z1,z2, ...,zn)=0 (*)
% : :
% fn(z1,z2, ...,zn)=0
%
% donde x=[z1,z2, ...,zn]' es el vector que resuelve (*).
% Para usar esta función, debe especificarse 'func' que es un string
% con el nombre de un archivo .m que contiene la función f.
%En caso necesario, puede usarse el vector 'param' para incluir
%parámetros adicionales de dicha función.
% Los argumentos x0, crit y maxit son la semilla inicial para x,
% el criterio de convergencia (tolerancia) y el número máximo de
% iteraciones permitidas.
% Programa de Carlos Urrutia
del = diag(max(abs(x0)*1e-4, 1e-8));
n = length(x0);
          for i=1:maxit
           f=feval(func,x0,param);
            for j=1:n
             J(:,j)=(f-feval(func,x0-del(:,j),param))/del(j,j);
          end
            x=x0-inv(J)*f;
            if norm(x-x0)<crit;       break;       end
            x0=x;
           end
if i>=maxit
sprintf('Advertencia: Número máximo de %g iteraciones ...
alcanzado', maxit)
end
```

1.0.3 Ejemplo 5

Antes de entrar en modelos dinámicos vamos a realizar un ejercico sobre un modelo de equilibrio general computable estático en el que un consumidor representativo consumirá dos bienes producidos.

Consumidores Existe un número muy grande de consumidores con idénticas preferencias y dotaciones de tiempo y capital, de modo que podemos agregarlos en un único agente representativo que resuelve el problema:

$$\underset{\{c_1,c_2\}}{Max}\ \varepsilon c_1^{\rho} + (1-\varepsilon)c_2^{\rho}$$

$$\begin{aligned} s.a.\ p_1c_1 + p_2c_2 &\leq w(l_1+l_2) + r(k_1+k_2) \\ l_1 + l_2 &\leq \overline{l} \\ k_1 + k_2 &\leq \overline{k} \\ & \overline{l}, \overline{k} \text{ dados} \end{aligned}$$

Hay que notar dos cosas importantes. La primera es que las condiciones de factibilidad sobre las dotaciones las podemos incorporar en la primera restricción, de modo que podemos plantear un problema más sencillo con sólo una restricción. La segunda cuestión importante es que sabemos que las ofertas totales de factores serán exactamente $\overline{l}$ y $\overline{k}$. La razón es que ambos factores son ofertados inelásticamente. Por tanto el problema es:

$$\underset{\{c_1,c_2\}}{Max}\ \varepsilon c_1^{\rho} + (1-\varepsilon)c_2^{\rho}$$

$$\begin{aligned} s.a.\ p_1c_1 + p_2c_2 &\leq w\overline{l} + r\overline{k} \\ & \overline{l}, \overline{k} \text{ dados} \end{aligned}$$

De este problema obtenemos las demanda de las dos mercancías una vez que hemos calculado las condiciones de primer orden del problema y tenemos en cuenta la restricción presupuestaria, que son:

$$\begin{aligned} \varepsilon\rho c_1^{\rho-1} - \lambda p_1 &= 0 \quad (1) \\ (1-\varepsilon)\rho c_2^{\rho-1} - \lambda p_2 &= 0 \quad (2) \\ p_1c_1 + p_2c_2 - w\overline{l} - r\overline{k} &= 0 \quad (3) \end{aligned}$$

Empresa 1 La primera empresa tiene una tecnología con la que produce el bien y_1 a partir de trabajo l_1 contratado y capital k_1 alquilado. La tecnología a su disposición es

$$y_1 = A_1 k_1^{\alpha_1} l_1^{1-\alpha_1}$$

que es una tecnología del tipo Coob-Douglas con parámetro de productividad agregada A_1 y parámetro distributivo α_1 (donde α_1 representa a la participación de las rentas del capital en el total de la renta generada por el primer sector, y $(1-\alpha_1)$ representa la participación de las rentas laborales en la renta generada por el primer sector). El problema de la empresa propiedad del trabajador y consumidor y capitalista representativo es:

$$\underset{\{l_1,k_1\}}{Min} \; wl_1 + rk_1$$

$$s.a.\; y_1 \leq A_1 k_1^{\alpha_1} l_1^{1-\alpha_1},$$

y que

$$p_1 y_1 - wl_1 - rk_1 = 0. \tag{4}$$

De la resolución de este problema obtenemos las siguientes condiciones de primer orden:

$$w = p_1 A_1 (1-\alpha_1) \left(\frac{k_1}{l_1}\right)^{\alpha_1}, \tag{5}$$

$$r = p_1 A_1 \alpha_1 \left(\frac{k_1}{l_1}\right)^{\alpha_1 - 1}. \tag{6}$$

Como vemos, la función de producción al ser homogénea de grado uno nos permite escribir las productividades marginales (o sea, los precios de los factores) como función de los ratios de capital por trabajador.

Empresa 2 La segunda empresa tiene una tecnología con la que produce el bien y_2 a partir de trabajo l_2 contratado y capital k_2 alquilado. La tecnología a su disposición es

$$y_2 = A_2 k_2^{\alpha_2} l_2^{1-\alpha_2}$$

que es una tecnología del tipo Coob-Douglas con parámetro de productividad agregada A_2 y parámetro distributivo α_2. El problema de la empresa es:

$$\underset{\{l_2,k_2\}}{Min} \; wl_2 + rk_2$$

$$s.a.\ y_2 \leq A_2 k_2^{\alpha_2} l_2^{1-\alpha_2},$$

y que

$$p_2 y_2 - w l_2 - r k_2 = 0. \tag{7}$$

De la resolución de este problema obtenemos las siguientes condiciones de primer orden:

$$w = p_2 A_2 (1-\alpha_2) \left(\frac{k_2}{l_2}\right)^{\alpha_2}, \tag{8}$$

$$r = p_2 A_2 \alpha_2 \left(\frac{k_2}{l_2}\right)^{\alpha_2 - 1}. \tag{9}$$

Condiciones de factibilidad Además de las condiciones de primer orden y de las restricciones presupuestaria y tecnológicas tenemos un conjunto de condiciones de factibilidad que nos dicen que en los mercados no pueden entrar en transacción más cantidades de factores que aquellas disponibles a través de las dotaciones. Esto es decir:

$$l_1 + l_2 \leq \overline{l} \tag{10}$$

$$k_1 + k_2 \leq \overline{k} \tag{11}$$

Definición de equilibrio competitivo Una vez que tenemos a todos nuestros agentes caracterizados a través de las funciones que definen su comportamiento, y una vez que tenemos explícitamente escritas todas las restricciones presupuestarias, tecnológicas y de factibilidad, estamos en condiciones de definir un equilibrio competitivo para esta economía.

Definición Un equilibrio competitivo para esta economía artificial es un vector de precios (p_1, p_2, r, w), una asignación de consumos (c_1, c_2) para el consumidor, y una utilización de factores productivos para las empresas (l_1, k_1, l_2, k_2) tal que:

- Dados los precios de las mercancías (p_1, p_2) y los precios de los factores de producción (r, w), las cantidades de consumo (c_1, c_2) resuelven el problema de maximización de los consumidores.

- Dado el precio de la mercancía p_1 y dados los precios de los factores (r, w) la asignación de factores de producción (l_1, k_1) resuelve el problema de minimización de costes, con beneficios cero, de la primera empresa.
- Dado el precio de la mercancía p_2 y dados los precios de los factores (r, w) la asignación de factores de producción (l_2, k_2) resuelve el problema de minimización de costes, con beneficios cero, de la segunda empresa.
- Todos los mercados vacían y la asignación es factible.

Que todos los mercados vacían significa que las ofertas y las demandas en los mercados de los bienes c_1 y c_2 coinciden con las cantidades producidas y_1, e y_2. Lo mismo sucede en los mercados de factores de producción donde debe suceder que $l_i^s = l_i^d$ y $k_i^s = k_i^d$, $i = \{1, 2\}$. Además de vaciarse los mercados, las condiciones de factibilidad 10 y 11 deben cumplirse.

Solución del modelo Resolver este modelo es tan sencillo como resolver un sistema de ecuaciones. Para hacerlo primero tenemos que tener claro qué ecuaciones son parte de la determinación de la solución y qué ecuaciones no lo son. Una vez que tengamos determinado un número de ecuaciones que son linealmente independientes sabremos para cuántas variables podemos resolver el modelo. Así que contamos ecuaciones:

$$\begin{aligned}
\varepsilon\rho c_1^{\rho-1} - \lambda p_1 &= 0 \\
(1-\varepsilon)\rho c_2^{\rho-1} - \lambda p_2 &= 0 \\
p_1 c_1 + p_2 c_2 - w\bar{l} - r\bar{k} &= 0 \\
p_1 A_1 (1-\alpha_1)\left(\frac{k_1}{l_1}\right)^{\alpha_1} - w &= 0 \\
p_1 A_1 \alpha_1 \left(\frac{k_1}{l_1}\right)^{\alpha_1 - 1} - r &= 0 \\
p_2 A_2 (1-\alpha_2)\left(\frac{k_2}{l_2}\right)^{\alpha_2} - w &= 0 \\
p_2 A_2 \alpha_2 \left(\frac{k_2}{l_2}\right)^{\alpha_2 - 1} - r &= 0 \\
l_1 + l_2 - \bar{l} &= 0
\end{aligned}$$

$$
\begin{aligned}
k_1 + k_2 - \overline{k} &= 0 \\
A_1 k_1^{\alpha_1} l_1^{1-\alpha_1} - c_1 &= 0 \\
A_2 k_2^{\alpha_2} l_2^{1-\alpha_2} - c_2 &= 0
\end{aligned}
$$

Aquí tenemos el listado completo. Hay que notar que algunas condiciones ya han sido incluidas en la forma en la que las ecuaciones estan escritas. Por ejemplo, no hemos escrito ahora variables como l_i^d, o k_i^s, para referirnos a las ofertas y demandas de factores. Simplemente hemos escrito l_i y k_i para referirnos al trabajo y al capital de equilibrio. Es decir, la condición de equilibrio en el mercados de factores está incluida y como $l_i^s = l_i^d = l_i$, entonces nosotros simplemente escribimos la variable sin hacer referencia a ofertas y demandas. En el mercado de factores sí hemos escrito más explícitamente que la condición de equlibrio es $A_i k_i^{\alpha_i} l_i^{1-\alpha_i} - c_i = 0$, $(i = \{1, 2\})$. El sistema de más arriba lo podemos arreglar marginalmente eliminando una variable y deshaciéndonos de una ecuación. La variable sería el multiplicador de Lagrange del problema del consumidor y la ecuación que perdemos es una de las dos condiciones de primer orden del problema del consumidor. Dividiendo una por la otra nos quedaríamos al final con el siguiente sistema:

$$
\begin{aligned}
\frac{\varepsilon}{(1-\varepsilon)} \left(\frac{c_1}{c_2}\right)^{\rho-1} - \frac{p_1}{p_2} &= 0 && (12) \\
p_1 c_1 + p_2 c_2 - w\overline{l} - r\overline{k} &= 0 && (13) \\
p_1 A_1 (1-\alpha_1) \left(\frac{k_1}{l_1}\right)^{\alpha_1} - w &= 0 && (14) \\
p_1 A_1 \alpha_1 \left(\frac{k_1}{l_1}\right)^{\alpha_1 - 1} - r &= 0 && (15) \\
p_2 A_2 (1-\alpha_2) \left(\frac{k_2}{l_2}\right)^{\alpha_2} - w &= 0 && (16) \\
p_2 A_2 \alpha_2 \left(\frac{k_2}{l_2}\right)^{\alpha_2 - 1} - r &= 0 && (17) \\
l_1 + l_2 - \overline{l} &= 0 && (18) \\
k_1 + k_2 - \overline{k} &= 0 && (19) \\
A_1 k_1^{\alpha_1} l_1^{1-\alpha_1} - c_1 &= 0 && (20) \\
A_2 k_2^{\alpha_2} l_2^{1-\alpha_2} - c_2 &= 0 && (21)
\end{aligned}
$$

Este es el sistema más pequeño que podemos obtener después de haber hecho uso de algunas relaciones triviales como $c_1 = y_1$, o $l_1^s = l_1^d$.

Pasamos a contar ecuaciones para descubrir que tenemos exactamente 10. Hacemos asimismo un recuento de las incógnitas y nos salen $(c_1, c_2, l_1, l_2, k_1, k_2, p_1, p_2, w, r)$ que son también 10. Podríamos pensar erróneamente que éste es un sistema con tantas ecuaciones como incógnitas y que por tanto podemos encontrar el valor de todas y cada una de ellas. Esto sería un error porque en realidad tenemos sólo 9 ecuaciones linealmente independientes. Para verlo escribamos

$$\begin{aligned} p_1c_1+p_2c_2 = \; & w\bar{l} + r\bar{k} \\ = \; & (wl_1 + rk_1) + (wl_2 + rk_2) \\ = \; & p_1y_1 + p_2y_2 \quad = p_1c_1 + p_2c_2 \end{aligned}$$

Es decir, tenemos una ecuación que puede ser encontrada como combinación lineal de otras ecuaciones del modelo, y por tanto no es una ecuación que nos pueda dar información que no tuviéramos ya. Esto significa que una ecuación sobra $[p_1c_1+p_2c_2-w\bar{l}-r\bar{k} = 0]$ y por tanto tenemos que eliminar una variable. Elegimos, por ejemplo $p_1 = 1$ (a la primera mercancía la consideramos como numerario), y ya tenemos el problema resuelto. Este problema no es más que una consecuencia de la Ley de Walras, que dice que una economía con n mercados de los cuales $n-1$ están en equilibrio, entonces tiene todos los mercados en equilibrio.

Tenemos, por tanto, 9 ecuaciones linealmente independientes y 9 incógnitas cuyo valor tenemos que determinar. De acuerdo con lo que hemos visto en los anteriores ejercicios bastaría hacer un programa en el que proponemos al algoritmo de Newton-Raphson una solución en este espacio de dimensión 9, dejamos que el algoritmo itere y esperamos la solución. En tanto que esta lógica es completamente correcta es posible hacerlo, sin embargo podemos ayudar al algoritmo de Newton-Raphson disminuyendo la dimensionalidad del problema. Como vamos a ver ahora mismo será suficiente con proponer al algoritmo una semilla que en vez de tener dimensión 9 tendrá solo dimensión 2. Vamos a verlo:
Imaginemos que conocemos (o proponemos como semilla) el valor de l_1 y k_1. Podemos seguir a través de la siguiente tabla el razonamiento sin olvidar que $p_1 = 1$:

De la ecuación	Obtenemos	Y sabemos
		l_1, k_1
18	l_2	l_1, k_1, l_2
19	k_2	l_1, k_1, l_2, k_2
20	c_1	l_1, k_1, l_2, k_2, c_1
21	c_2	$l_1, k_1, l_2, k_2, c_1, c_2$
14	w	$l_1, k_1, l_2, k_2, c_1, c_2, w$
15	r	$l_1, k_1, l_2, k_2, c_1, c_2, w, r$
12	p_2	$l_1, k_1, l_2, k_2, c_1, c_2, w, r, p_2$

Hemos obtenido todos los valores de todas las variables del sistema salvo de dos de cuyo valor no estamos seguros pues son una semilla. Tampoco hemos verificado si dos de las ecuaciones son correctas, a saber, la ecuación 16 y la ecuación 17. Por tanto podemos hacer correr el algoritmo de Newton-Raphson sobre estas dos ecuaciones que no hemos comprobado. Si el algoritmo se detiene en un punto del espacio bidimensional (l_1, k_1), entonces sabremos que esa es la solución ya que todas las demás ecuaciones se verán verificadas.

Para finalizar el ejercicio concluimos escribiendo un programa en MatLab que resuelve exáctamente este problema de dos sectores.

Programa en MatLab A continuación voy a escribir y comentar los programas de MatLab que nos ayudarían a resolver numéricamente este problema de Equilibrio General Competitivo. Para ello vamos a necesitar tres archivos de MatLab, dos de ellos serán del tipo *function* y uno será del tipo *script*. En el archivo de tipo *script* vamos a definir los parámetros del modelo y vamos a hacer una llamada a las funciones que contienen al algoritmo de Newton-Raphson y a las condiciones de primer orden.

Run4m.m

```
%Programa run4m.m, funciona conjuntamente con
%secant.m y cpo4m.m
%Este programa resuelve un modelo de equilibrio
%general computable
%para una economía con dos sectores y dos factores de
%producción ofertados inelásticamente por un
%consumidor representativo.
%CONSUMIDORES
```

```
%Max epsilon*c1^rho+(1-epsilon)*c2^rho
%c1,c2
%s.a. p1*c1+p2*c2=w*lbar+r*kbar
%
%EMPRESA 1
%Min w*l1+r*k1
%k1,l1
%s.a. y1=A1*k1^alpha1*l1^(1-alpha1)
%
%p1*y1-w*l1-r*k1=0
%
%EMPRESA 2
%Min w*l2+r*k2
%k2,l2
%s.a. y2=A2*k2^alpha2*l2^(1-alpha2)
%
%p2*y2-w*l2-r*k2=0
%
%lbar y kbar dados
clear
%Definición de parámetros del modelo
 A1      =  10;
 A2      =  10;
 alpha1  =  0.35;
 alpha2  =  0.4;
 epsilon =  0.2;
 rho     =  -1;
 lbar    =  2;
 kbar    =  20;
 param   =  [A1 A2 alpha1 alpha2 epsilon rho lbar kbar];
%Definición de parámetros de programa
 crit    =  1e-4;
 maxit   =  1000;
 x0      =  [1.0687; 9.6207];
 sol     =  secant('cpo4m', x0, param, crit, maxit);
%Resultados
```

```
l1   =  sol(1);
k1   =  sol(2);
l2   =  lbar-l1;
k2   =  kbar-k1;
c1   =  A1*k1^alpha1*l1^(1-alpha1);
c2   =  A2*k2^alpha2*l2^(1-alpha2);
p2   =  ((1-epsilon)/epsilon)*(c2/c1)^(rho-1);
r    =  A1*alpha1*(k1/l1)^(alpha1-1);
w    =  A1*(1-alpha1)*(k1/l1)^alpha1;
```

Este programa **run4m.m** comienza con unos comentarios y el comando **clear**. Después pasa a la definición de parámetros del modelo y a continuación se definien parámetros que tienen que ver con el funcionamiento del programa, que son una semilla (**x0**) que contiene el valor del par (l_1, k_1), un nivel de tolerancia (**crit**) y un número máximo de iteraciones (**maxit**) para el algoritmo de Newton-Raphson.

El programa **cpo4m.m** es un archivo del tipo *function* en el que tenemos todas las condiciones de primer oden y las condiciones de factibilidad del problema.

Cpo4m.m

```
function f=cpo4m(z, param)
 %Asignación de variables
  l1   =  z(1);
  k1   =  z(2);
 %Asignación de parámetros
  A1       =  param(1);
  A2       =  param(2);
  alpha1   =  param(3);
  alpha2   =  param(4);
  eosilon  =  param(5);
  rho      =  param(6);
  lbar     =  param(7);
  kbar     =  param(8);
 %Condiciones de factibilidad y vaciado
```

```
 l2   =   lbar-l1;
 k2   =   kbar-k1;
 c1   =   A1*k1^alpha1*l1^(1-alpha1);
 c2   =   A2*k2^alpha2*l2^(1-alpha2);
%Condiciones de primer orden
 p2   =   ((1-epsilon)/epsilon)*(c2/c1)^(rho-1);
 r    =   A1*alpha1*(k1/l1)^(alpha1-1);
 w    =   A1*(1-alpha1)*(k1/l1)^alpha1;
%Condiciones por comprobar
 f(1)   =   r-p2*A2*alpha2*(k2/l2)^(alpha2-1);
 f(2)   =   w-p2*A2*(1-alpha2)*(k2/l2)^alpha2;
f=f';
```

Finalmente la función **secant.m** que escribo aquí pero que ya tenéis más arriba.

Secant.m

```
function x=secant(func, x0, param, crit, maxit)
 %secant.m Programa para resolver un sistema de ecuaciones simultaneas.
 % x=secant(func, x0, param, crit, maxit) usa el metodo de secante
 % para resolver el siguiente sistema:
 %
 % f1(z1,z2, ...,zn)=0
 % f2(z1,z2, ...,zn)=0 (*)
 % : :
 % fn(z1,z2, ...,zn)=0
 %
 % donde x=[z1,z2, ...,zn]' es el vector que resuelve (*).
 % Para usar esta función, debe especificarse 'func' que es un string
 % con el nombre de un archivo .m que contiene la función f. En caso
 % necesario, puede usarse el vector 'param' para incluir parámetros
 % adicionales de dicha función.
 % Los argumentos x0, crit y maxit son la semilla inicial para x,
 % el criterio de convergencia (tolerancia) y el número máximo de
 % iteraciones permitidas.
```

```
% Programa de Carlos Urrutia
del = diag(max(abs(x0)*1e-4, 1e-8));
n = length(x0);
for i=1:maxit
f=feval(func,x0,param);
for j=1:n
J(:,j)=(f-feval(func,x0-del(:,j),param))/del(j,j);
end
x=x0-inv(J)*f;
if norm(x-x0)<crit
break
end
x0=x;
end
if i>=maxit
sprintf('Advertencia: Número máximo de %g iteraciones alcanzado', maxit)
end
```

Con estos programas ya podéis calcular el equilibrio general competitivo para una economía con dos mercados de factores y dos mercados de bienes.