Lecciones de Equilibrio General Computable con MatLab

Gonzalo Fernández de Córdoba Martos

Esta versión: 28 de julio de 2002

1 Lección primera

1.1 Objetivo del curso

Muchos de los modelos que se utilizan en la actualidad para plantear y resolver problemas macroeconómicos son modelos de equilibrio general computables. Estos modelos deben ser resueltos con la ayuda de un ordenador puesto que salvo en raras excepciones estos modelos tienen soluciones analíticas. Por tanto, en la mayoría de las ocasiones nos encontraremos con complicados modelos en los que el lector no "ve" cómo han sido resueltos, es decir, estos modelos tienen una caja negra en las que se introducen un conjunto de condiciones y de las que sale un conjunto de soluciones. El objetivo fundamental de este curso es el de abrir la caja y ver qué sucede en su interior. Un ejemplo de modelo de equilibrio general dinámico computable y determinista con una enorme caja negra es el de Fernández de Córdoba y Kehoe (1999). En las siguientes lecciones empezaremos a construir y resolver modelos sencillos que vayan aproximándose al modelo de FdCK introduciendo y explicando los elemento que dieron lugar a este modelo.

Este curso de computación de modelos de equilibrio general dinámico deterministas está dirigido a estudiantes de doctorado que quieren aprender a modelizar y a resolver con la ayuda de un ordenador modelos computables. No se supone ningún conocimiento previo de computación y por ello las primeras lecciones se dedican a mostrar la lógica de los algoritmos de optimización basados en el método de Newton explicando cómo se confeccionan los programas partiendo de los comandos más básicos. Todos los códigos que a continuación se presentan están escritos en MatLab®. La razón es que MatLab es un lenguaje suficientemente flexible y fácil de aprender, permite construir nuestras propias funciones con sencillez y el soporte gráfico es excelente. Además son muchos los programas ya construidos en MatLab por economistas.

1.2 La lógica del algoritmo de Newton

Cualquier modelo que planteemos a lo largo del curso se reducirá después de unas transformaciones a un sistema de ecuaciones que estará formado por las condiciones de primer orden del problema y por las restricciones de factibilidad. Podemos describir este sistema de n ecuaciones en n incógnitas como $F : R^n \longrightarrow R^n$. Nuestro problema será entonces encontrar un vector

$\widehat{x} = (\widehat{x}_1, \cdots, \widehat{x}_n)$ de R^n tal que su imagen por $F : R^n \longrightarrow R^n$ sea $F(\widehat{x}) = 0$. El algoritmo basado en el método de Newton-Raphson pretende encontrar una solución a este sistema de la forma $F(\widehat{x}) = 0$. Para encontrar el $\widehat{x}$ solución del sistema se aproxima la función a través de la primera expansión de Taylor a la función F.

$$F(x) = F(\overline{x}) + J(\overline{x})(x - \overline{x}), \tag{1}$$

donde $J(\overline{x})$ es la matriz jacobiana de F evaluada en $\overline{x}$, es decir,

$$J(\overline{x}) = \begin{bmatrix} F_{11}(\overline{x}) & F_{12}(\overline{x}) & F_{1n}(\overline{x}) \\ F_{21}(\overline{x}) & F_{22}(\overline{x}) & F_{2n}(\overline{x}) \\ \\ F_{n1}(\overline{x}) & F_{n2}(\overline{x}) & F_{nn}(\overline{x}) \end{bmatrix},$$

donde $F_{ij}(\overline{x}) = \frac{\partial F_i(\overline{x})}{\partial x_j}$.

Dado que estamos buscando un cero de la ecuación $F(x)$, la ecuación 1 podemos evaluarla en $\widehat{x}$ y escribirla como,

$$\widehat{x} = \overline{x} - J(\overline{x})^{-1} F(\overline{x}).$$

El algoritmo funcionaría de la siguiente manera:

1. Proponemos una semilla x_1 y evaluamos la función $F(x_1)$ y $J^{-1}(x_1)$, para calcular

$$x_2 = x_1 - J(x_1)^{-1} F(x_1)$$

2. Fijamos un nivel de tolerancia ε y calculamos alguna distancia entre x_2 y x_1. Si la distancia es inferior a ε, entonces nos quedamos con x_2 como solución. En caso contrario volvemos al paso 1. y evaluamos la función $F(x_2)$ y $J^{-1}(x_2)$, para calcular

$$x_3 = x_2 - J(x_2)^{-1} F(x_2)$$

Realizamos estas operaciones tantas veces como sea necesario hasta que encontremos un x_t y x_{t+1} tales que la distancia sea menor que el nivel de tolerancia.

1.2.1 Ejemplo 1

Veamos ahora cómo funciona con un ejemplo sencillo. Queremos encontrar los ceros de la función

$$y = (x - 4)(x + 4).$$

La simple inspección visual muestra que los ceros de la función se encontrarán en $x = 4$ y $x = -4$. Ahora vamos a construir nuestro primer programa de MatLab para encontrar esas soluciones.

```
%ej1.m
%Este programa resuelve la ecuación sencilla del ejemplo 1
%La ecuación es: y = (x - 4)(x + 4)
clear
%Punto inicial
x(1) = -10;
%Máximo número de iteraciones
maxit=1000;
   for s=1:maxit
            J(s) = 2*x(s);
            x(s+1) = x(s)-J(s)^(-1)*(x(s)-4)*(x(s)+4);
            if abs(x(s+1)-x(s))<1e-20; break; end
   end
if s>=maxit
sprintf('Atención: Número máximo de %g iteraciones alcanzado', maxit)
end
   sprintf('La solución es %g', x(s))
```

El programa ha empezado con una semilla $x_1 = -10$, y por tanto la solución que el programa nos dará es $x^* = -4$. Si hubiéramos introducido como semilla el valor $x_1 = 10$, el programa habría dado como solución $x^* = 4$. En este simple ejemplo vemos algo esencial del algoritmo de Newton: las semillas son de una importancia capital para encontrar la solución a un problema. La localización de la semilla en relación a la solución tiene un impacto en la respuesta del programa. Cuanto más cerca esté la semilla a la solución que buscamos tanto mejor. En este sencillo problema de hallar los ceros de una función de R en R hemos podido computar el jacobiano sin ningún problema, pero si la función hubiera sido de R^n en R^n con n muy grande entonces computar el jacobiano se puede convertir en un problema tan complicado como el de hallar los ceros del sistema. Para resolver este problema podemos computar las derivadas numéricas de la función. La definición de derivada nos dice que

$$\begin{aligned}
J_1(x) &= \lim_{h_1 \to 0} \frac{F(x) - F(x_1 + h, x_2, \cdots, x_n)}{h_1}, \\
J_2(x) &= \lim_{h_2 \to 0} \frac{F(x) - F(x_1, x_2 + h, \cdots, x_n)}{h_2}, \\
\vdots &= \qquad \vdots \\
J_n(x) &= \lim_{h_n \to 0} \frac{F(x) - F(x_1, x_2, \cdots, x_n + h)}{h_n},
\end{aligned}$$

donde $J_i(x)$ es el vector columna con las n derivadas parciales con respecto a x_i. Por tanto $J(x) = [J_1(x), J_2(x), \cdots, J_n(x)]$. Podemos fijar en un ordenador un incremento h suficientemente pequeño como para poder escribir

$$\begin{aligned}
J_1(x) &\approx \frac{F(x) - F(x_1 + h_1, x_2, \cdots, x_n)}{h_1}, \\
J_2(x) &\approx \frac{F(x) - F(x_1, x_2 + h_2, \cdots, x_n)}{h_2}, \\
\vdots &\approx \qquad \vdots \\
J_n(x) &\approx \frac{F(x) - F(x_1, x_2, \cdots, x_n + h_n)}{h_n},
\end{aligned}$$

y de esta manera aproximar las derivadas parciales.

1.2.2 Ejemplo 2

Ahora queremos encontrar los ceros de la función del Ejemplo 1, pero haciendo uso de una aproximación numerica a la derivada de y con respecto a x. Para ello construimos el siguiente programa en el que definimos un cociente incremental para el jacobiano y un incremento que fijamos en **h = 1e-8**.

```
%ej2.m
%Este programa resuelve por el método de Newton-Raphson la
%ecuación trivial y = (x + 4) * (x - 4) haciendo uso de una
%aproximación numérica a la derivada de y con respecto a x



clear
%Punto inicial
x0 = -10;
%Número máximo de iteraciones permitido
maxit = 1000;
%Incremento
h = 1e-8;
x(1) = x0;
    for s=1:maxit
    J(s) = 1/(2*h)*((x(s)+h+4)*(x(s)+h-4)-(x(s)-h+4)*(x(s)-h-4));
    x(s+1) = x(s)-J(s)^(-1)*(x(s)-4)*(x(s)+4);
    if abs(x(s+1)-x(s))<1e-20; break; end
    end
if s>=maxit
sprintf('Atención: Numero máximo de %g iteraciones alcanzado', maxit)
end
    sprintf('La solución es %g', x(s))
```