Análisis Económico con MatLab

Gonzalo Fernández de Córdoba Martos

Primero de Diciembre de dos mil

1 Introducción

Este curso de introducción a MatLab está orientado a estudiantes de economía con interés por la programación. Los conocimientos de economía requeridos no son demasiado altos ya que está pensado para estudiantes de todos los cursos. Las dos primeras lecciones servirán de repaso para los estudiantes de cursos avanzados y muy actuales para los estudiantes de los primeros cursos. Los conocimientos que hay que tener sobre ordenadores son los básicos. Este curso está pensado para dotar a los asistentes con un instrumento de proceso y cálculo. La razón fundamental por la que esto es importante en el análisis económico (al igual que en otras disciplinas) es que en muchas ocasiones las modernas teorías y los modelos en las que éstas se sustentan no producen soluciones analíticas y hay que buscar por tanto soluciones numéricas a los problemas que plantean. Otra razón de importancia es que incluso en el caso en que la teoría produzca soluciones analíticas a veces es conveniente realizar simulaciones de los modelos o presentar los resultados de un modo que requiere el proceso de datos para problemas muy particulares. Un lenguaje de programación como MatLab permite resolver todos los problemas que resuelven paquetes de programación diseñados para propósitos generales y además permite resolver también los problemas particulares que se nos planteen para los cuales no existen paquetes. MatLab es un lenguaje de programación tan flexible como Fortran y, aunque por ser un lenguaje interpretado sea más lento que Fortran o C, tiene unas capacidades gráficas y una sencillez de manejo que lo hacen perfecto para iniciarse en la programación. Además las últimas versiones de MatLab permiten el diseño de interfaces gráficas de usuario que nos permiten diseñar entornos gráficos muy fáciles de usar por aquellos a los que nuestros programas estén dirigidos.

En este curso vamos a ver los comandos básicos de programación e iremos amenizando el aprendizaje con ejercicios de modelos económicos sencillos, para que al tiempo que aprendemos a manejar el lenguaje vayamos viendo su utilidad con problemas concretos de economía. Una vez que tengamos los conocimientos básicos estaremos en condiciones de subir un nivel con la versión MatLab 5 en la que existen una serie de bibliotecas para una variedad sorprendente de problemas entre los que se encuentran el análisis de señales, matemáticas financieras, redes neuronales y un larguísimo etcétera. A mí no me cabe ninguna duda de que este instrumento de trabajo tiene valor en el mercado y por tanto es bueno que se acostumbren a manejarlo con soltura.

La notación que voy a utilizar en el texto indicará en **negrita** nombres

de comandos interpretables por MatLab como **disp** o nombres de archivos que MatLab puede leer como **programa.m**.

2 Arranque y espacio de trabajo

2.1 La ventana de comandos

Despues de pinchar sobre el icono de MatLab donde quiera que esté en vuestro ordenador se abrirá una pantalla que es la interfaz gráfica de comandos de MatLab, o ventana de comandos. En esta ventana de comandos encontraremos el símbolo » que es el que indica la linea de comandos. Tras el símbolo » podemos escribir cualquier comando o nombre de archivo que contenga comandos en MatLab. Así, por ejemplo, podemos escribir »**a=2** y obtener la respuesta de MatLab que será:

```
» a=2
a =
2
»
```

Como podemos ver MatLab ha replicado exactamente lo que le hemos dicho, y es que **a=2.** En realidad MatLab ha hecho algo más que repetir lo que le hemos dicho, de hecho ahora MatLab se acuerda de que existe un símbolo **a** que tiene el valor 2. Para verlo basta con escribir el comando **whos** para obtener un listado de todas las variables que tenemos guardadas, así que si escribimos:

```
»whos
  Name    Size      Elements   Bytes   Density   Complex
  a       1 by 1    1          8       Full      No
Grand total is 1 elements using 8 bytes
»
```

Como se puede ver MatLab ha asignado la letra **a** que ocupa 8 bytes de memoria a una matriz de dimensión 1 por 1, que naturalmente tiene un sólo elemento y que además no es complejo, o sea, es real. Sería sin embargo desesperante que cada vez que MatLab asigna un valor a una variable o realiza un operación replicara en la pantalla de comandos lo que acaba de hacer. Nosotros deseamos escribir programas en archivos que a veces serán muy largos, así que una replicación de cada línea de código haría a los programas muy lentos, ya que el resultado de cada operación aparecería en pantalla.

Para evitar esto basta con poner un **;** detrás de cada línea. De este modo si ahora queremos asignar a **b** el valor 3 escribiríamos en la ventana de comandos »**b=3;** y no obtendríamos ninguna respuesta por parte de MatLab aunque sí ejecutaría la instrucción.

```
» b=3;
»
```

Si ahora volvemos a escribir el comando **whos** sobre la ventana de comandos encontraremos que efectivamente existe **b** y que tiene asignados sus 8 bytes de memoria.

2.2 Espacio de trabajo

Nuestra finalidad como ya he dicho es la de escribir comandos para realizar tareas de computación. Una forma de hacerlo es ir escribiéndolos poco a poco sobre la ventana de comandos y otra es escribirlos ordenadamente en un archivo de texto sin formato, guardarlo con un nombre con extensión **m** (como **programa.m**) y luego escribir sobre la ventana de comandos »**mi_programa** (sin el **.m**) para que MatLab ejecute los comandos que hay dentro del archivo exactamente en el orden en el que lo hemos escrito. Es un hecho que no hay nadie suficientemente paciente y aventurero como para escribir los comandos de un programa sobre la línea de comandos, y esto es así porque es seguro que al escribir vamos a cometer errores y por tanto nos gustaría tener que corregir sólo los errores que hemos cometido sin necesidad de tener que escribirlo todo otra vez (y cometer nuevos errores). Además los programas que ya funcionan bien los podéiís guardar y transportar de un ordenador a otro o de un archivo a otro o utilizarlos más tarde. Ahora la pregunta es, ¿dónde se escriben los archivos?, y otra es ¿cómo accede MatLab a los archivos que hemos escrito?. Responder a estas dos preguntas es importante antes de comenzar a trabajar. La respuesta a la primera pregunta es donde nos de la gana. Un lugar posible en mi ordenador es c:\docs\user\gonzalo\cursos\matlab. En mi ordenador la unidad c:\ tiene sólo dos carpetas, una de paquetes de programas y otra de documentos, de modo que la trayectoria de acceso a un documento siempre debe ser c:\docs. Mi ordenador está siendo usado en la actualidad por dos personas, una es Alberto y la otra soy yo, de modo que para acceder a uno de mis documentos debo acudir a la subcarpeta user y una vez allí debo acudir a la subsubcarpeta c:\docs\user\gonzalo. Pero resulta que yo hago muchísimas cosas con el ordenador; tengo cartas; artículos; guiones de cine; cursos; etc. Como los

documentos que yo voy a escribir forman parte de un curso y este curso es de MatLab, voy a escribir mis programas en la subsubsubsubcarpeta llamada matlab. Quiero hacer notar dos cosas importantes. Aunque el sistema operativo de windows permite tener carpetas con nombres de longitud superior a 8 caracteres, MatLab 4 no es capaz de leerlos, así que a lo largo de toda la trayectoria de acceso al archivo cada subcarpeta debe tener 8 caracteres o menos. La otra cosa importante que quiero hacer notar es la estructura horizontal en la que deben estar organizados los directorios. Esto no tiene nada que ver con MatLab sino con el estilo. Un odernador del que cuelgan cientos de subcarpetas del directorio principal c:\ como si fuera un peine es visualmente desagradable, hace difícil el acceso a la información y por tanto disuade del trabajo.

Una vez que hemos decidido dónde queremos tener nuestros programas de MatLab debemos "transportar" la ventana de comandos al mismo directorio donde están los programas que queremos ejecutar. Si sobre la ventana de comandos escribimos »**cd**, MatLab responde:

```
» cd
C:\C1PRG\UTILES\MATLAB\BIN
»
```

Yo tengo instalado mi MatLab 4 en el directorio c:\c1prg\utiles\matlab, que es mi carpeta para programas, mi subcarpeta para lenguajes de programación y finalmente mi MatLab. El programa ejecutable de MatLab, el **matlab.exe** se encuentra en la subcarpeta de MatLab BIN. Por defecto MatLab abre su ventana de comandos en esa subcarpeta, y desde allí podría ejecutar nuestros programas si siempre los escribiéramos precisamente allí. Esto es algo que absolutamente no debéis hacer nunca ya que no sólo es un follón sino que además hace lenta la ejecución de programas por razones que explicaré más adelante. Lo que hay que hacer es aprender a "transportar" la ventana de comandos al directorio que nosotros queremos porque es muy sencillo. Lo único que tenemos que hacer es escribir sobre la ventana de comandos la trayectoria de destino con el comando **cd** (heredado del sistema DOS y que quiere decir *change directory*). Así, como yo he escrito mis programas en c:\docs\user\gonzalo\cursos\matlab tendría que escribir

```
» cd c:\docs\user\gonzalo\cursos\matlab
»
```

Para ver que MatLab a respondido correctamente basta con que volváis a introducir el comando »**cd** para ver:

```
» cd
```

C:\DOCS\USER\GONZALO\CURSOS\MATLAB
»

Ahora MatLab está listo para ejecutar los programas que hayamos escrito en esta subcarpeta.

La ventana de comandos tiene además otras cosas interesantes. La más espectacular es cuando escribimos el comando **help**. Este comando muestra los contenidos de las distintas cajas de herramientas que están incorporadas a MatLab y que se denominan *toolboxes.* Escribiendo »**help** obtenemos:

» help
HELP topics:
toolbox\local - Local function library.
matlab\datafun - Data analysis and Fourier transform functions.
matlab\elfun - Elementary math functions.
matlab\elmat - Elementary matrices and matrix manipulation.
matlab\funfun - Function functions - nonlinear numerical methods.
matlab\general - General purpose commands.
matlab\color - Color control and lighting model functions.
matlab\graphics - General purpose graphics functions.
matlab\iofun - Low-level file I/O functions.
matlab\lang - Language constructs and debugging.
matlab\matfun - Matrix functions - numerical linear algebra.
matlab\ops - Operators and special characters.
matlab\plotxy - Two dimensional graphics.
matlab\plotxyz - Three dimensional graphics.
matlab\polyfun - Polynomial and interpolation functions.
matlab\sounds - Sound processing functions.
matlab\sparfun - Sparse matrix functions.
matlab\specfun - Specialized math functions.
matlab\specmat - Specialized matrices.
matlab\strfun - Character string functions.
matlab\dde - DDE Toolbox.
matlab\demos - The MATLAB Expo and other demonstrations.
toolbox\stats - Statistics Toolbox.
toolbox\uitools - User Interface Utilities.
toolbox\ident - System Identification Toolbox.
toolbox\control - Control System Toolbox.
toolbox\ncd - Nonlinear Control Design Toolbox.
nnet\nnet - Neural Network Toolbox.

nnet\nndemos - Neural Network Demonstrations and Applications.
toolbox\mmle3 - MMLE3 Identification Toolbox.
toolbox\signal - Signal Processing Toolbox.
simulink\simulink - SIMULINK model analysis and construction functions.
simulink\simdemos - SIMULINK demonstrations and samples.
simulink\blocks - SIMULINK block library.
simulink\sb2sl - SystemBuild 3.0 model import into SIMULINK.
toolbox\optim - Optimization Toolbox.
nag\nag - NAG Foundation Toolbox - Numerical & Statistical Library
nag\examples - NAG Foundation Toolbox - Numerical & Statistical Library
For more help on directory/topic, type "help topic".

Como podemos ver, en la última línea encontramos una indicación de cómo obtener más ayuda sobre los comandos incorporados en el lenguaje. Si introducimos »**help elfun** nos encontramos con comandos básicos de MatLab. Lo que vemos es:

» help elfun
Elementary math functions.
Trigonometric.
sin - Sine.
sinh - Hyperbolic sine.
asin - Inverse sine.
asinh - Inverse hyperbolic sine.
cos - Cosine.
cosh - Hyperbolic cosine.
acos - Inverse cosine.
acosh - Inverse hyperbolic cosine.
tan - Tangent.
tanh - Hyperbolic tangent.
atan - Inverse tangent.
atan2 - Four quadrant inverse tangent.
atanh - Inverse hyperbolic tangent.
sec - Secant.
sech - Hyperbolic secant.
asec - Inverse secant.
asech - Inverse hyperbolic secant.

csc - Cosecant.
csch - Hyperbolic cosecant.
acsc - Inverse cosecant.
acsch - Inverse hyperbolic cosecant.
cot - Cotangent.
coth - Hyperbolic cotangent.
acot - Inverse cotangent.
acoth - Inverse hyperbolic cotangent.
Exponential.
exp - Exponential.
log - Natural logarithm.
log10 - Common logarithm.
sqrt - Square root.
Complex.
abs - Absolute value.
angle - Phase angle.
conj - Complex conjugate.
imag - Complex imaginary part.
real - Complex real part.
Numeric.
fix - Round towards zero.
floor - Round towards minus infinity.
ceil - Round towards plus infinity.
round - Round towards nearest integer.
rem - Remainder after division.
sign - Signum function.
»

Que son las funciones básicas con números. Podemos obtener aún más información escribiendo el comando **help** seguido del nombre de la función. Por ejemplo si escribimos »**help abs** tendremos en la misma ventana de comandos una breve descripción, de manual de referencia, de cómo se usa el comando en cuestión. Veámoslo:

» help abs
ABS Absolute value and string to numeric conversion.
ABS(X) is the absolute value of the elements of X. When
X is complex, ABS(X) is the complex modulus (magnitude) of
the elements of X.
See also ANGLE, UNWRAP.

ABS(S), where S is a MATLAB string variable, returns the
numeric values of the ASCII characters in the string.
It does not change the internal representation, only the
way it prints.
See also SETSTR.
»

De esta manera podemos entender muchos comandos sin necesidad de acudir a los manuales. La forma básica en la que **abs** funciona es transformando al valor absoluto el valor del argumento numérico introducido. Por ejemplo:

» a=-2;
» b=abs(a)
b =
2
»

Sería una buena idea que delante de la pantalla pasen un rato mirando los comandos que MatLab tiene construidos en el lenguaje tecleando **help** y el nombre de los comandos. En particular deben mirar los comandos **save** y **load**.

3 Creación de matrices y vectores

Para MatLab todo es una matriz. Una matriz es una ordenación de números. Un número es una matriz de 1 por 1. Un vector de dimensión n es una matriz de dimensión $n \times 1$ (o $1 \times n$) según sea un vector columna (o fila). Nosotros vamos a trabajar con números ordenados luego vamos a trabajar con matrices. Primero vamos a aprender a introducir matrices y luego vamos a aprender a hacer algunas operaciones con ellas.

3.1 Introducir matrices

Para que se vayan acostumbrando a manejar archivos propongo que creemos uno ahora mismo. El nombre será **ej1.m** y lo van a guardar en un directorio. Luego vamos a llevar la pantalla de comandos al directorio donde está el archivo **ej1.m**, y luego vamos a abrir ese archivo para escribir lo siguiente:

```
%Este programa introduce una matriz de dos formas distintas.
clear
A=[1 2 3; 4 5 6; 7 8 9];
```

B=[1, 2, 3; 4, 5, 6; 7, 8, 9]

Antes de ejecutar el programa en la ventana de comandos con la instrucción »**ej1** les comento dos cosas. La primera es que cualquier línea escrita que comience con el símbolo % no será ejecutada por MatLab, de modo que % es el comando que nos va a permitir comentar y documentar nuestros programas dentro del propio programa. Esto es algo de la mayor importancia para programar bien. Los programas tienen que estar debidamente documentados porque o bien en algún momento los va a leer alguien que no los ha hecho y por tanto no entiende de que va el programa o bien lo vamos a leer nosotros mismos un día después de haberlo escrito y ya no nos acordaremos de qué es lo que hacía exáctamente. El otro comando que aparece es **clear,** que significa borra. Lo que hace es liberar el espacio de memoria en el que residían las variables que existían en el espacio de trabajo destruyéndolas. Es muy importante que los programas comiencen con la memoria fresca y vacía porque de lo contrario es posible que MatLab confunda variables que ya existían con variables que están en proceso de ejecución y los resultados pueden ser incontrolables. Una vez aclarado el contenido del programa basta con que observemos cómo están escritas las matrices y los resultados. Las comas o los espacios separan elementos de una misma fila y el **;** abre una nueva fila. Ahora podemos ejecutar el programa escribiendo sobre la ventana de comandos »**ej1**. El resultado será:

```
» ej1
B =
 1   2   3
 4   5   6
 7   8   9
»
```

¿Por qué?. Porque si han seguido las instrucciones la línea que contiene a la matriz **A** tiene un **;** al final y por tanto no se reproduce en la ventana de comandos. Fíjense qué curioso, si escriben »**help ej1** obtienen la información que andaba detrás del %. De hecho cada vez que se escribe **help** seguido de un comando, MatLab simplemente reproduce la documentación que los programadores de MathWorks (la compañía que construyó a MatLab) pusieron en el encabezamiento de sus programas.

Ahora vamos a construir un par de vectores. Les propongo que abran un archivo llamado **ej2.m** y que escriban dentro de él lo siguiente:

```
%Este programa hace un par de vectores
clear
```

```
a=[1:20];
b=[1:0.1:20];
```

Al ejecutarlo no obtenemos ninguna respuesta de MatLab, aunque los vectores **a** y **b** están en la memoria. Para ver que efectivamente existen podemos hacer varias cosas. Una es ejecutar el comando **whos** sobre la ventana de comandos y obtener el siguiente resultado

```
» whos
  Name      Size            Elements    Bytes  Density  Complex
  a         1 by 20         20          160    Full     No
  b         1 by 191        191         1528   Full     No
Grand total is 211 elements using 1688 bytes
»
```

Al construir el vector **b** hemos especificado entre paréntesis cuadrados tres números separados por **:**, el primero es el valor inicial, el segundo es el incremento y el último es el valor final. En el vector **a** no hemos especificado el segundo valor y en ese caso MatLab asume que el incremento es de 1. Otra cosa que podemos hacer para ver que efectivamente existen los vectores **a** y **b** es simplemente declararlos en la ventana de comandos, así si escribimos »**a** obtenemos:

```
» a
a =
Columns 1 through 12
  1   2   3   4   5   6   7   8   9   10   11   12
Columns 13 through 20
  13   14   15   16   17   18   19   20
»
```

Finalmente le podemos pedir a MatLab en la ventana de comandos que nos de prueba de su existencia a través del comando **size** o del comando **length** que son comandos muy usados y que nos informan sobre el tamaño del vector que metamos como argumento (si quieren saber más sobre estos comandos ya saben que no hay más que usar el **help** seguido del nombre del comando).

3.2 Operaciones con matrices

Matlab hace operaciones con matrices de un modo muy eficiente. Para ver lo sencillo que es operar con matrices propongo hacer un tercer programa llamado **ej3.m** en el que vamos a escribir:

```
%Este programa hace operaciones estándar con matrices
```

```
clear
%Definición de las matrices
A=[1 2 1; 4 6 5; 7 2 7];
B=[1 1 2; 5 4 2; 2 4 3];
%Suma
C=A+B;
%Multiplicación
D=A*B;
%Potencia enésima (tenemos que definir el n)
n=3;
E=A^n;
%Suma de un escalar a todos los elementos de la matriz
F=B+n;
%Multiplicación de un escalar por todos los elementos de una matriz
G=n*A;
%Traspuesta
H=B';
%Inversa
I=inv(F);
%División izquierda (similar a inv(A)*B)
J=A\B;
%División derecha (similar a B*inv(A))
K=B/A;
%Dimensión de una matriz
L=size(A);
%Obtención de la primera fila de una matriz
M=A(1,:);
%Obtención de la segunda fila de una matriz
N=B(2,:);
%Obtención de la primera columna de una matriz
O=A(:,1);
%Obtención del elemento (2,1) (segunda fila primera columna) de una
matriz
P=B(2,1);
%Obtención de los valores propios de una matriz
K=eig(A);
```

Aquí hemos visto algunas de las operaciones estándar que podemos realizar con matrices, sin embargo MatLab permite hacer operaciones con matrices

que si las hicieran en un examen de matemáticas estarían suspensos con toda seguridad. Son operaciones llamadas de *array* y permiten tratar y operar sobre las matrices saltándonos reglas que resultan incómodas. Por ejemplo, imagine que tiene dos matrices y quiere multiplicar una por otra pero elemento a elemento, entonces tiene que hacer una operación *array*. Escribimos un programa llamado **ej4.m** con los siguientes comandos:

```
%Este programa hace operaciones array con matrices
clear
%Definición de las matrices
A=[1 2; 6 9];
B=[4 5; 3 2];
%Producto elemento a elemento
C=A.*B;
%Cociente elemento a elemento
D=A./B;
%Potencia enésima de los elementos de una matriz
n=3;
E=A.^n;
```

Como se ve no es necesario más que poner un puntito **.** antes de la operación a realizar para que MatLab entienda que tiene que operar como un *array*. Para ver los resultados de las anteriores operaciones basta con declarar el nombre de las variables y verlos en la ventana de comandos.

3.3 Operaciones con vectores

Si a y b son dos vectores de la misma dimensión pueden definirse las siguientes operaciones:

$a.*b = (a_1b_1, a_2b_2, \cdots a_nb_n)$ (producto elemento a elemento)

$a*b' = (a_1b_1 + a_2b_2 + \cdots a_nb_n)$ (producto escalar)

$a'*b = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} (b_1, \cdots, b_n) = \begin{pmatrix} a_1b_1 & \cdots & a_1b_n \\ \vdots & & \vdots \\ a_nb_1 & \cdots & a_nb_n \end{pmatrix}$ (producto de Kroneker)

Ejercicio: Hacer un pequeño programa llamado **ej5.m** con las operaciones con vectores indicadas arriba.

3.4 Matrices especiales

MatLab tiene construidas unas funciones que crean matrices especiales. Algunos ejemplos son los siguientes:

X1=1:20 (Número enteros entre 1 y 20)

X2=(1:20) (Números enteros entre 1 y 20)

X3=(1:0.1:5) (Números entre 1 y 5 a intervalos de una décima)

X4=randn(3,4) (Matriz de dimensión 3x4 con números aleatorios gaussianos)

X5=rand(2,7) (Matriz 2x7 con números aleatorios uniformes)

X7=eye(5) (Matriz identidad de dimensión 5x5)

X8=zeros(3) (Matriz de ceros de dimensión 3x3)

X9=zeros(1,10) (Vector de ceros de dimensión 10)

X10=ones(3) (Matriz de unos de dimensión 3x3)

X11=ones(1,10) (Vector de unos de dimensión 10)

Ejercicio: Escribir un programa llamado **ej6.m** que cree las matrices y vectores indicados más arriba.

4 Operaciones con números

En el momento de realizar operaciones con números tenemos que tener en cuenta que cuando MatLab se encuentra ante una operación ejecuta unas operaciones antes que otras, es decir, si en una expresión hay un $*$ y también hay un $+$ entonces MatLab primero ejecuta la multiplicación y después la suma. Para ejercitarse en las operaciones propongo como ejercicio que realicen:

$$1+\frac{5}{e}+\frac{3}{\pi^2}=3.1434$$

$$\frac{\sqrt{23}}{e}+\left|\sqrt{2}-\sqrt{5}\right|=2.5861$$

$$\frac{15}{\frac{1}{3}+\frac{e}{2\pi}+\frac{sen(\pi/8)}{4}+\ln(23)}=3.7527$$

$$\sqrt[3]{\left|\pi^2-\frac{120}{\pi-14}\right|+\frac{1}{\pi}}=2.7694$$

y que comprueben los resultados.

5 Control del flujo de programa

Una de las cosas que seguramente desearemos hacer al escribir un programa es repetir una operación o un conjunto de operaciones muchas veces hasta que una condición se satisfaga. Hay dos tipos de estructuras básicas que deben conocer ahora mismo. Una es el bucle **for** y la otra es la condición **if**. Para obtener información sobre estas estructuras sería suficiente con escribir sobre la ventana de comandos »**help for** y »**help if**, no obstante voy a explicar cómo funcionan.

5.1 for

El bucle

for [condición]
[secuencia de comandos]
end

realiza toda la secuencia de comandos contenidas entre las líneas **for** y **end** siempre que la condición no se vea satisfecha. Una vez que la condición se satisface entonces MatLab salta a ejecutar todas las líneas de código posteriores a **end**. Un ejemplo sería un código con las siguientes líneas:

```
%Este programa usa el comando for para generar una secuencia
%autorregresiva
clear
T = 50;
k0 = 0;
phi = 0.7;
k(1) = k0;
for t=2:T
k(t)=phi*k(t-1)+randn;
end
plot(k)
```

Este programa comienza con una breve documentación y el necesario comando **clear**. Después define una serie de parámetros que son la longitud de la secuencia **T**, el valor inicial de la serie **k** y un parámetro **phi**. Después asigna el valor de **k0** al primer valor de la secuencia para entrar en el bucle **for** donde se le dice a MatLab que ejecute en el interior del bucle unos comandos siempre que **t** sea inferior a **T**, es decir, la línea que dice **for t=2:T**, quiere decir "desde **t** igual a 2 hasta **T**" haz lo que hay escrito

hasta **end**. El comando **end** indica a MatLab hasta dónde tiene que hacer la evaluación para el valor de **t**. Una vez que llega a **end** MatLab acude a la línea **for,** suma una unidad a **t,** comprueba si **t** es **T** y, en caso de ser inferior, vuelve a ejecutar los comandos de dentro del bucle. Una vez que lo ha hecho tantas veces que **t=T** se verifica la condición y salta por debajo de **end**, encontrándose con el comando **plot** que indica a MatLab que debe sacar por pantalla un gráfico con la secuencia de **k**. Es muy importante hacer notar que las variables **t** y **T** son variables distintas porque MatLab, a diferencia de Fortran, distingue entre símbolos en mayúsculas y en minúsculas.

Los bucles **for** pueden estar anidados unos dentro de otros en cuyo caso los va resolviendo desde el más interior al más exterior. Por ejemplo en el siguiente programa construimos una matriz cuyos elementos a_{ij} dependen de la posición en la que el elemento se encuentra:

```
%Este programa ilustra otro uso del comando for
clear
n=10;
for i=1:n
        for j=1:n
        A(i,j)=i*j*abs(i-j);
        end
end
mesh(A)
AZ = 37.5; EL = 45;
view(AZ, EL)
```

Como se puede ver este programa genera una matriz cuyos valores a_{ij} son más pequeños a medida que nos acercamos a la diagonal (con 0 exactamente en la diagonal), y más grande a medida que nos alejamos de ella. Para hacer una gráfico de una matriz ejecutamos el comando **mesh** que lo que hace es asignar en el espacio (x, y) los índices (i, j) y la elevación viene dada por el valor a_{ij}. El comando **view** toma dos argumentos, que son el ázimut y la elevación y modifican la perspectiva con la que se ve el gráfico. Como pueden ver queda un corazoncito muy mono. El resultado de anidar dos bucles **for** es el siguiente: el programa llega por primera vez a **for i=1:n** asignando a **i** el valor 1, a continuación se encuentra con **for j=1:n** asignando a **j** el valor 1, y opera **A(i,j)=i*j*abs(i-j);** (que es igual a 0). LLega al **end** del bucle **j**, y comprueba que **j** todavía no es **n**, por tanto a **j** le suma una unidad y por tanto **j**=2. Opera la expresión **A(i,j)=i*j*abs(i-j);** (que es igual a 2). MatLab hará esto hasta que el valor de **j** sea igual a **n**, entonces se encuentra

con el **end** del bucle **i**. Compara el valor de **i** (que era igual a 1) con **n**, y al ver que **i** todavía no es **n**, le suma una unidad y se encuentra de nuevo (por segunda vez ahora) con el bucle **for j=1:n**. A **j** le asigna el valor 1 otra vez (pero ahora con **i** valiendo 2) y opera **A(i,j)=i*j*abs(i-j);** (que de nuevo es 2). Esto se hace hasta que **i** sea igual a **n**, provocando el salto hasta el **end** asociado al bucle de **i**. El siguiente comando es **mesh**, y el programa termina.

Vamos a hacer un último ejemplo del comando **for** que pone de manifiesto sólo un aspecto de las enormes capacidades gráficas de MatLab. Este programa será igual que el anterior pero con el corazón moviéndose en su ázimut.

```
%Este programa ilustra las posibilidades gráficas de MatLab
clear
n=10;
for i=1:n
        for j=1:n
        A(i,j)=i*j*abs(i-j);
        end
end
EL = 45;
for i=0:90
        mesh(A)
        AZ = i;
        view(AZ, EL)
        pause(0.1)
end
```

En el segundo bucle vamos cambiando el ázimut en cada iteración consiguiendo el efecto de una figura que se mueve. El comando **pause** (ejecutar »**help pause**) detiene la ejecución de cualquier programa hasta que se pulsa la tecla intro, pero si añadimos una cantidad en segundos como argumento, entonces MatLab espera esa cantidad y se pone en ejecución otra vez. En este programa he incluido el comando **pause(0.1)** para que a la tarjeta gráfica le de tiempo a repintar la figura, ya que si no lo pusiéramos no veríamos más que la última de las figuras.

5.2 if

La estructura

if [condición]
[secuencia de comandos]
else
[secuencia de comandos]
end

comprueba la primera condición y si el resultado de la condición es verdadero entonces ejecuta la secuencia de comandos que está entre los comandos **if** y **else**, en caso contrario (es falsa) ejecuta la secuencia de comandos que está entre **else** y **end**. Veamos algunos ejemplos de su funcionamiento:

```
%Este programa simula T lanzamientos de una moneda posiblemente
%desequilibrada.
clear
T = 1000;
equil = 0.5;
for i=1:T
a = rand;
        if a<=equil
                lanzamiento(i) = 0;
        else
                lanzamiento(i) = 1;
        end
end
hist(lanzamiento)
frecunos = sum(lanzamiento)/T;
```

En este programa vemos varias cosas interesantes. En primer lugar dentro del bucle **for** hemos definido una variable **a** que toma un valor aleatorio en una distribución uniforme sobre el intervalo $[0, 1]$. En la estructura **if** decimos que si el valor de **a** es menor o igual al parámetro **equil**, que hemos fijado en 0.5 para hacer la moneda equilibrada pero que podía haber tomado otro valor distinto, entonces asigne a la variable **lanzamiento** el valor 0, y en caso contrario (**a**>**equil**) asigne el valor 1. Finalmente el programa realiza un histograma a través del comando **hist** (ver **help hist**) donde vemos las frecuencias relativas en un gráfico. Finalmente, inventamos la variable **frecunos** que es la suma de los componentes del vector **lanzamiento** dividido por el número de lanzamientos. Como el vector **lanzamiento** consta de ceros y unos, al pedir la suma (ejecutar »**help sum**) de los elementos del vector en realidad encontramos exáctamente el número de ocurrencias del evento 1 y, al dividirlo entre el número total de elementos **T,** encontramos

la frecuencia con la que el uno apareció. Es conveniente notar dos cosas. La primera es que los nombres de las variables nos dicen algo sobre su naturaleza y sus nombres son largos. Imaginen el mismo programa en el que las variables en vez de llamarse **equil**, **lanzamiento** y **frecunos** se llamaran a, b, y c. El resultado es que los programas serían completamente ilegibles por un humano, cansarían al tener que estar recordando quién es quién y muestra un estilo de programación nefasto. Además pueden ver que los bucles anidados y las estructuras del programa están tabuladas, de modo que hay líneas de comandos que empiezan en la columna 1 de mi editor y otras que comienzan en la columna 7 y otras en la 14. Esto hace que los programas sean más fáciles de leer porque si además hacemos que un **for** comience en la misma columna que su correspondiente **end**, y hemos metido los comandos de dentro del bucle hacia adentro, entonces es muy fácil ver el alcance de los bucles **for** y de las estructuras condicionales **if**.

Otro ejemplo sobre el uso de la estructura **if** lo vemos aquí

```
%Este programa pone de manifiesto la conjetura de Slutsky sobre ciclos
%económicos. Él mostró que los ciclos económicos pueden ser el resultado
%de la acumulación de choques estocásticos.
clear
T = 50;
phi=0.9;
y(1) = 0;
for t=2:T
        if rand<=0.5
         alet(t) = 1;
        else
         alet(t) = -1;
        end
y(t) = phi*y(t-1)+alet(t);
end
plot(y)
```

Al ejecutar el programa se puede ver que se generan ciclos en la variable **y**. Como la variable aleatoria tiene sólo dos estados la serie temporal queda muy quebrada. Podemos mejorarlo introduciendo más estados.

```
%Este programa pone de manifiesto la conjetura de Slutsky sobre ciclos
%económicos. Este programa es igual al anterior pero tiene cuatro estados
clear
T = 100;
```

```
phi=0.9;
y(1) = 0;
for t=2:T
a=rand;
        if a<=0.25
                alet(t) = 1;
        elseif a<=0.5
                alet(t) = 0.5;
        elseif a<=0.75
                alet(t) = -0.5;
        else
                alet(t) = -1;
end
y(t) = phi*y(t-1)+alet(t);
end
plot(y, 'r')
```

El anterior programa hace uso de la estructura

if [condición]
[comandos]
elseif [condición]
[comandos]
elseif [condición]
[comandos]
⋮
else
[comandos]
end

que como podemos ver es muy útil cuando la condición tiene más de dos estados. Al final del programa pedimos que MatLab pinte la función en color rojo (ver **help plot**).

Con las estructuras que hemos visto ya podemos ponernos a trabajar y hacer todos los programas que queramos. Existen otras estructuras interesantes que las iremos introduciendo a medida que las necesitemos, pero lo cierto es que con estas dos se pueden hacer todos los programas que necesitaremos. Ahora que ya tenemos control del programa y sabemos algunos comandos esenciales podemos empezar a hacer algunos modelos económicos.

6 Oferta y demanda

Sabemos que la función de demanda es una representación sobre el espacio de precios y cantidades de los deseos, y en ocasiones de las necesidades, del hombre. Por otro lado sabemos que la oferta de bienes homogéneos, en entornos competitivos, representa en el mismo espacio la capacidad tecnológica para satisfacer esos deseos y necesidades. En un primer modelo sencillo vamos a representar la función de demanda inversa a través de una sencilla función lineal $p = a - bQ$, y la oferta a través de $p = c + dQ$. El programa que escribimos a continuación hace una gráfica de las dos.

```
%Este programa hace unos ejercicios sencillos con las fuciones
%de oferta y demanda p=c+dQ y p=a-bQ.
clear
%definición de los parámetros
a = 10;
b = 0.7;
c = 2;
d = 0.5;
%Definimos valores para el vector p
p = [0:0.1:10];
%Creamos las funciones
Qd = (a-p)/b;
Qs = (p-c)/d;
%Gráfica de ambas funciones
plot(Qd, p, 'r', Qs, p, 'b')
%axis([0 max(max(Qs), max(Qd)) 0 max(p)])
%axis('off')
grid
```

El programa comienza con las necesarias líneas de comentario seguidas del imprescindible comando **clear**. Luego viene la definición de los parámetros. A continuación creamos un vector de precios positivo y después creamos las cantidades demandadas de acuerdo con la ley de la demanda definida y las cantidades ofertadas para cada uno de los precios. El comando **plot** nos permite representar simultáneamente las dos funciones siempre y cuando los vectores tengan la misma dimensión como es el caso. El orden es importante, primero la abscisa de la primera función seguida de la ordenada y del color que deseamos, luego la abscisa de la segunda función, la ordenada, (que es la misma) y su color. Más tarde jugamos un poco con las escalas. El programa

tal y como está escrito en el texto tomará las escalas automáticas de MatLab. Ejecútenlo para ver el resultado.

Si quitamos la marca de comentario % de la línea
%axis([0 max(max(Qs), max(Qd)) 0 max(p)])
guardamos el programa y lo ejecutamos vemos que ahora el espacio en el que se representan es $\Re^2_+$ que es donde las funciones están definidas. Esto es así porque MatLab permite que especifiquemos los límites de la ventana para hacer la representación. La sintaxis del comando **axis** pide que se provean los valores del mínimo valor de la abscisa, el máximo valor de la abscisa, el mínimo valor de la ordenada y finalmente el máximo valor de la ordenada. Al hacerlo yo quería que el mínimo valor del eje x fuera 0, y el máximo valor del eje x fuera el máximo valor de, o bien la demanda Q_d o bien la oferta Q_s. Como quiera que tanto Q_d como Q_s son dos vectores, al escribir **max(max(Qs), max(Qd))** estamos buscando el máximo de dos números. El primero es el máximo de la oferta y el segundo es el máximo de la demanda, luego el número que sale al final es el máximo del máximo de la oferta o la demanda. Finalmente yo deseaba que el valor mínimo de la ordenada fuera 0 y el máximo valor de la ordenada fuera el máximo valor de p. Ejecútenlo para ver el resultado.

Si volvemos a comentar la línea anterior con la marca de comentarios % pero quitamos el comentario a la línea que dice axis('off') obtenemos una nueva representación al ejecutar el programa. Para ver más detalles ver »**help axis**. Con esto ya hemos hecho nuestro primer programa de economía en MatLab.

A este primer programa podríamos ir añadiendo más cosas hasta llegar a tener un programa que haga cosas verdaderamente interesantes. Lo primero que se me ocurre ahora es por supuesto calcular el par estrella, o sea, el par de equilibrio. Para ello necesitamos una condición de equilibrio. Nuestra condición de equilibrio dice que es un par de precios y cantidades (p^*, Q^*) tal que los mercados vacían, es decir, tal que los planes de los productores y de los consumidores son compatibles, no quedando nada en las estanterías de los comercios ni quedando consumidores insatisfechos. Escrito de otra manera estamos diciendo que un equilibrio es un par (p^*, Q^*) tal que dados los precios para productores y consumidores sucede que

$$Q_d(p^*) = Q_s(p^*) = Q^*.$$

Para calcular el equilibrio es suficiente con resolver el siguiente sistema de

ecuaciones

$$\begin{aligned} p &= a - bQ_d, \\ p &= c + dQ_s, \\ Q_d &= Q_s = Q. \end{aligned}$$

Que es un sistema de tres ecuaciones con tres incógnitas (p, Q_d, Q_s). La forma más sencilla de resolverlo es sustituir Q_s y Q_d por su equivalente en el equilibrio Q. Así, el sistema queda

$$\begin{aligned} p &= a - bQ, \\ p &= c + dQ. \end{aligned}$$

Igualando obtenemos que $a - bQ = c + dQ$, agrupando términos y sacando factores comunes obtenemos finalmente

$$Q^* = \frac{a-c}{b+d}.$$

Sustituyendo esta expresión de Q^* en la función de demanda (o en la de oferta) obtenemos

$$p^* = a - b\frac{a-c}{b+d} = \frac{a(b+d) - b(a-c)}{b+d} = \frac{ad+bc}{b+d}.$$

Para hacer que nuestro programa compute el equilibrio sólo tendríamos que añadir las siguientes líneas de comandos al programa que ya tenemos:

```
%Cálculo del equilibrio
Qstar = (a-c)/(b+d);
pstar = (a*d+b*c)/(b+d);
disp('El par de equilibrio (p_estrella, Q_estrella) es')
[pstar Qstar]
```

Las asignaciones a **Qstar** y a **pstar** están claras ya que se corresponden exactamente con nuestro resultado anterior. Después tenemos dos líneas de código que nos muestran el resultado. La primera es el comando **disp** que toma como argumento una cadena de texto encerrada entre **'**comillas**'**. La siguiente línea simplemente declara una matriz de una fila y dos columnas, que al no estar seguida de **;** hace que MatLab reproduzca el resultado. Una forma muy bonita de sacar los resultados es a través del comando **num2str** (*number to string*) y que asigna un número a una cadena de texto. Su sintaxis para el caso sería:

```
disp(['Precios de equilibrio= ', num2str(pstar)])
disp(['Cantidades de equilibrio= ', num2str(Qstar)])
```

6.1 Impuestos

Podemos continuar con nuestro ejercicio y suponer que hay un gobierno que interviene en el mercado de la mercancía Q poniendo un impuesto a las transacciones. En este caso las ecuaciones serían:

$$\begin{aligned} p + t &= a - bQ, \\ p &= c + dQ. \end{aligned}$$

Fíjense que ya he incorporado la condición de equilibrio eliminando los subíndices d y s de las cantidades en las funciones de oferta y demanda. Podemos volver a calcular el equilibrio y obtener:

$$Q(t) = \frac{a - (c + t)}{b + d}.$$

Al calcular los precios tenemos que tener en cuenta que el impuesto produce una separación entre el precio pagado por los consumidores y el precio percibido por los productores. Éstos son:

$$\begin{aligned} p_c &= \frac{ad + b(c + t)}{b + d}, \\ p_v &= \frac{d(a - t) + cb}{b + d}. \end{aligned}$$

Naturalmente esta separación es, como se puede probar, igual a $p_c - p_v = t$, y la recaudación del gobierno es $T = t * Q(t)$. Para incluir al gobierno dentro de nuestro programa podemos añadir las siguientes líneas.

```
%Gobierno
tasa = 1;
pc = (a*d+b*(c+tasa))/(b+d);
pv = ((a-tasa)*d+b*c)/(b+d);
Qt = (a-(c+tasa))/(b+d);
infis = tasa*Qt;
disp(['Precio del comprador = ', num2str(pc)])
disp(['Precio del productor = ', num2str(pv)])
disp(['Cantidad con impuesto = ', num2str(Qt)])
disp(['Ingreso fiscal = ', num2str(infis)])
```

Una vez más hemos calculado a mano unas expresiones que tienen sentido para nosotros y las hemos escrito en un código para obtener una respuesta de

MatLab. Podemos hacer lo mismo con otras magnitudes de interés como por ejemplo los excedentes de los productores y consumidores, así como la pérdida irrecuperable de eficiencia ocasionada por el impuesto. Podemos hacer un gráfico en el que podamos ver una representación de las funciones de oferta y demanda, de la solución de mercado sin impuestos, de la distorsión ocasionada por los impuestos y de los excedentes de productores, consumidores y del gobierno, así como de la pérdida irrecuperable de bienestar ocasionada por los impuestos. Cada una de las áreas se corresponde con una expresión analítica:

$$\begin{aligned} EE &= tQ(t) \\ EC &= (a - p_c)Q(t)/2 \\ EP &= (p_v - c)Q(t)/2 \\ PIE &= t(Q^* - Q(t))/2 \end{aligned}$$

Para calcular cada una de estas expresiones analíticas es suficiente con incluir las siguientes líneas de código en nuestro programa:

```
%Bienestar
EE = tasa*Qt;
EC = (a-pc)*Qt/2;
EP = (pv-c)*Qt/2;
PIE = tasa*(Qstar-Qt)/2;
ET = EE+EC+EP;
disp(['Excedente del Estado = ', num2str(EE)])
disp(['Excedente de los Consumidores = ', num2str(EC)])
disp(['Excedente de los Productores = ', num2str(EP)])
disp(['Pérdida Irrecuperable de Bienestar = ', num2str(PIE)])
disp(['Excedente Total = ', num2str(ET)])
```

6.1.1 La curva de Laffer

Hasta aquí lo único que hemos hecho ha sido calcular unas expresiones, escribirlas en un archivo interpretable por MatLab y obtener unos resultados. Pero podemos hacer más cosas. Las cosas que queremos hacer se relacionan con el hecho de que los ordenadores son muy rápidos haciendo cuentas pesadas y repetitivas. Nos gustaría aprovechar esa capacidad para obtener resultados nuevos que serían muy tediosos de conseguir si no fuera porque podemos pedirle al ordenador que repita las cuentas una y otra vez, y que

nos guarde los resultados de cada cuenta de un modo ordenado para que luego nosotros podamos ver esos resultados. Vamos a escribir un pequeño programa que resuelva el ejercicio anterior para toda una colección de impuestos y para ello vamos a hacer uso del bucle **for** y de la estructura **if**. El programa se llamará **laffer.m** y lo escribiremos a continuación.

```
%Este programa muestra la curva de Laffer para una
%colección de impuestos
%para ello repetimos una operación de cálculo del equilibrio variando
%en cada iteración la cantidad de impuestos. Cuando el impuesto
%sea tan alto
%que la cantidad de equilibrio con impuesto sea igual o menor que cero
%detenemos el programa y hacemos un gráfico con los ingresos fiscales.
clear
%definición de los parámetros
a = 10;
b = 0.1;
c = 2;
d = 0.5;
%Cálculo del equilibrio sin impuestos
Qstar = (a-c)/(b+d);
pstar = (a*d+b*c)/(b+d);
%Máximo número de iteraciones, incremento y tasa inicial
maxit = 1000;
inc = 0.1;
tasa(1) = 0;
for i=1:maxit
        pc(i) = (a*d+b*(c+tasa(i)))/(b+d);
        pv(i) = ((a-tasa(i))*d+b*c)/(b+d);
        Qt(i) = (a-(c+tasa(i)))/(b+d);
        EE(i) = tasa(i)*Qt(i);
        EC(i) = (a-pc(i))*Qt(i)/2;
        EP(i) = (pv(i)-c)*Qt(i)/2;
        PIE(i) = tasa(i)*(Qstar-Qt(i))/2;
        ET(i) = EE(i)+EC(i)+EP(i);
        if Qt(i)<=0; break; end
        tasa(i+1)= tasa(i)+inc;
end
plot(tasa, EE, 'r')
```

```
title('Curva de Laffer')
xlabel('Impuesto')
ylabel('Recaudación Fiscal')
grid
```

Algunas partes de este programa ya son conocidas y otras lo son menos. En primer lugar tenemos que observar que el límite del bucle **for** ha sido fijado en **maxit**, y que este es un número muy grande. En realidad nosotros no queremos que el bucle se ejecute **maxit** veces, sino sólo las necesarias. Para eso está la línea de código que dice

if Qt(i)<=0; break; end

Esta línea dice: si la cantidad de equilibrio calculada en la iteración i-ésima es menor o igual que 0, entonces abandona el bucle en el que estoy metido aunque la condición de salida del bucle **for** no se vea satisfecha. Eso es **break** (significa rompe y es interesante que hagan un »**help break**). Además, como en el caso de que **break** no se vea alcanzado no queremos que haga nada más que continuar, entonces no tiene demasiado sentido poner una condición **else** vacía y, por tanto, cerramos la estructura **if** con su correspondiente **end**. Como pueden ver en esa línea hemos usado una forma particular de la estructura **if** [condición] [comandos] **else** [comandos] **end** en la que el **else** no es necesario.

La línea de comandos

tasa(i+1)= tasa(i)+inc;

es un simple actualizador de valor para la variable **tasa**. El valor de **tasa** para la primera iteración ha sido definido en la línea que dice **tasa(1)=0**, y una vez que entra en el bucle se actualiza a través de otra variable a la que he llamado **inc** (que quiere ser una abreviación de incremento). Una de las particularidades que hace a MatLab muy interesante es que nosotros hemos sido capaces de definir vectores con sólo añadir un índice a cada una de las variables. Así, por ejemplo, cuando la iteración es la primera y MatLab alcanza la línea **pc(i) = (a*d+b*(c+tasa(i)))/(b+d);** crea una variable llamada **pc** a la que asigna 8 bytes de memoria. Cuando pasa a la segunda iteración ($i = 2$) MatLab simplemente añade al vector **pc** un nuevo valor y el asigna 16 bytes de memoria, y así sucesivamente hasta que termina. En otros lenguajes como Fortran, C, C++ o Java, al crear una variable el programador tiene que especificar desde el primer momento cuantos bytes de memoria tienen que ser reservados para esa variable. Fíjense que cantidad de trabajo de papel y lápiz nos quita MatLab al tener esa propiedad. Si nosotro queremos tener toda la colección de impuestos para los cuales el

problema tiene sentido económico (o sea, valores de $Q(t) > 0$) tendríamos que calcular la tasa para la cual $Q(t) \leq 0$, saber cuantos valores de **tasa** vamos a considerar (y esto depende de **inc**) y sólo cuando tuviéramos todo eso sabríamos cual va a ser el tamaño de todos los vectores que vamos a usar. Por ejemplo, si después de la ejecución de **laffer.m** tecléan »**i** para obtener el número de iteraciones realizadas ($i = 82$). Si cambian el código y hacen **inc=0.01**, guardan y ejecutan el nuevo programa obtienen $i = 802$. Es decir, la densidad de todos y cada uno de los vectores ha aumentado sin que hayan tenido que especificar un nuevo tamaño para los vectores creados. Al final del programa encontramos el comado **plot** ya conocido y otros nuevos que se encargan de añadir cosas al gráfico creado como **title**, **xlabel**, **ylabel** y **grid**. La ventana donde aparece el gráfico contiene un menú deplegable del que un elemento es Edit. Al pinchar sobre Edit se despliegan unas opciones entre las que se encuentra Copy Options. Una vez allí vemos que podemos elegir entre un gráfico con formato WMF (Windows Meta File) o BMP (Mapa de Bits o BitMaP) además de la posibilidad de invertir el color de fondo (que al ser negro puede hacer que nuestra impresora se agote y que el papel quede como una esponja bañada en tinta), y la de mantener el aspecto con el que se ve la ventana. Una vez realizada nuestra selección salimos y pinchamos de nuevo en Edit, pero ahora para seleccionar la opción Copy. Esta última tarea copia el gráfico sobre una memoria transitoria que es el ClipBoard. Una vez que tenemos el gráfico allí podemos acudir a nuestro editor favorito (Scientific WorkPlace, por ejemplo) y lo pegamos con el comando Paste o con el acelerador de teclado Crtl+ins.

6.1.2 Ricardo, Reagan y la curva de Laffer

El resultado del programa anterior es un gráfico en el que vemos la famosa curva de Laffer. Cuando el impuesto es 0 la recaudación fiscal es cero, y cuando es increíblemente alto la recaudación fiscal vuelve a ser 0 debido a que la demanda y la producción caen hasta 0. Como la curva de Laffer es continua entonces tiene que haber un máximo en algún sitio. Reagan razonaba que si los impuestos eran demasiado altos (a la derecha del máximo) un incremento de los impuestos llevaba a *i*) una disminución de la recaudación fiscal y *ii*) una caída en la oferta de trabajo y *iii*) una caída en la producción. El recíproco llevaría a un incremento de la recaudación, de la oferta de trabajo, de la producción y de los excedentes totales. David Ricardo sabía que los impuestos inducían una distorsión que generaba una pérdida de bienestar. También

sabía que la pérdida irrecuperable de bienestar dependía de la elasticidad de las funciones de oferta y demanda. Él proponía que los impuestos debían recaer sobre las rentas de la tierra ya que la oferta de tierra cultivable es tremendamente inelástica. Sabemos por la teoría económica que Ricardo tenía razón. En el caso límite en el que la función de oferta es perfectamente inelástica la distorsión causada por un impuesto es 0, y todo el excedente del productor pasa a manos del estado a través del impuesto sin que se produzcan pérdidas irrecuperables de eficiencia. Hemos visto con el programa de la sección anterior que la curva de Laffer tiene un máximo. El máximo lo podemos calcular sin ninguna dificultad, para ello basta con escribir la función de ingresos fiscales,

$$EE = t * Q(t) = t * \frac{a - (c + t)}{b + d}.$$

Vemos que es una función cuadrática muy simple y que podemos escribir como

$$EE = \frac{1}{b + d}\left((a - c)t - t^2\right).$$

Derivando EE con respecto a t e igualando a 0, llegamos a que $t^* = (a-c)/2$. Que es el nivel de impuestos para el que los ingresos fiscales son máximos[1]. La idea de Ricardo expresada en términos más formales es que si la función de oferta se hace más y más inelástica entonces la diferencia $(a - c)$ se hace cada vez más grande, y en consecuencia la tasa a partir de la cual los ingresos fiscales se hacen decrecientes tiende a infinito (o dicho de otra manera, no hay una tasa suficientemente alta a partir de la cual los ingresos fiscales se hacen decrecientes en la tasa). He escrito un programa que pone de manifiesto estos puntos con gran nitidez. Todos los comandos del programa que viene a continuación son conocidos y lo único que tenemos que hacer ahora es leerlo con detenimiento. El nombre del programa es **laffer1.m** y es este:

%Este programa muestra la curva de Laffer para una colección de impuestos

%para ello repetimos una operación de cálculo del equilibrio variando

%en cada iteración la cantidad de impuestos. Cuando el impuesto sea tan alto

%que la cantidad de equilibrio con impuesto sea igual o menor que cero

%detenemos el programa y hacemos un gráfico con los ingresos fiscales.

[1] Basta con hacer la segunda derivada con respecto a t y comprobar que es negativa.

```
%Después cambiamos dos parámetros de la función de oferta para hacerla
%más inelástica y repetimos el experimento.
clear
%definición de los parámetros
a = 10;
b = 0.1;
c = [2 -10];
d = [0.5 1.4];
for j=1:length(d);
    %Cálculo del equilibrio sin impuestos
    Qstar(j) = (a-c(j))/(b+d(j));
    pstar(j) = (a*d(j)+b*c(j))/(b+d(j));
    %Máximo número de iteraciones, incremento y tasa inicial
    maxit = 6000;
    inc = 0.5;
    tasa(1) = 0;
    for i=1:maxit
       pc(i,j) = (a*d(j)+b*(c(j)+tasa(i)))/(b+d(j));
       pv(i,j) = ((a-tasa(i))*d(j)+b*c(j))/(b+d(j));
       Qt(i,j) = (a-(c(j)+tasa(i)))/(b+d(j));
       EE(i,j) = tasa(i)*Qt(i,j);
       EC(i,j) = (a-pc(i))*Qt(i,j)/2;
       EP(i,j) = (pv(i)-c(j))*Qt(i,j)/2;
       PIE(i,j)= tasa(i)*(Qstar(j)-Qt(i,j))/2;
       ET(i,j) = EE(i,j)+EC(i,j)+EP(i,j);
       if Qt(i,j)<=0; break; end
       tasa(i+1)= tasa(i)+inc;
    end
    plot(tasa, EE(:,j), 'r')
    title('Curva de Laffer')
    xlabel('Impuesto')
    ylabel('Recaudación Fiscal')
       if j==1
            gtext('Reagan')
       else
            gtext('Ricardo')
       end
    grid
```

```
    hold on
end
```

El programa comienza con los comentarios y el comando **clear**. Después viene la definición de los parámetros del modelo donde tanto **c** como **d** han sido definidos como dos vectores fila de dos dimensiones. Los dos primeros valores para **c** y **d** se usarán en un experimento, y luego los cambiaremos para que MatLab repita el mismo experimento pero con los segundos parámetros. Para ello vamos a necesitar dos bucles, uno que controle quién es la función de oferta, y otro bucle para realizar la construcción de la curva de Laffer. El primer bucle comienza con **for j=1:length(d);** y termina en el último **end** del programa. Nótese que la tabulación del programa está hecha de tal manera que es fácil identificar quién va con quién. El comando **for j=1:length(d);** itera en el índice j y avanza en incrementos de una unidad hasta llegar a ser la longitud (ver »**help length**) del vector **d** que como sabemos es 2. La razón por la que pongo **length(d)** y no pongo un 2 directamente es que es posible que en el futuro yo quiera experimentar con tres funciones de oferta, y en ese caso tendría que modificar elementos del programa que no tienen una relación directa con el experimento (como por ejemplo la dimensión de un bucle). Los números que aparecen en un programa sin que se sepa muy bien de dónde vienen se llaman números mágicos y su existencia en un programa denotan falta de estilo. Cuando el programa entra el el bucle **j** por primera vez el valor de **j** es **j**= 1, y por tanto la secuencia de comandos

```
%Cálculo del equilibrio sin impuestos
    Qstar(j) = (a-c(j))/(b+d(j));
    pstar(j) = (a*d(j)+b*c(j))/(b+d(j));
```

calcula el par (p^*, Q^*) con los primeros valores de **c** y **d**, es decir calcula el equilibrio con una función de oferta particular. Una vez que ha hecho esto el programa sigue tal y como lo hubiera hecho el programa de la sección anterior llamado **laffer.m**. Así llegamos al segundo bucle del programa con el comando **for i=1:maxit** donde se inicia el cálculo de la curva de Laffer. Dentro de este bucle encontramos ahora cadenas de comandos del estilo **Qt(i,j) = (a-(c(j)+tasa(i)))/(b+d(j));** que como se puede ver dependen de los dos índices **i** y **j**. Esta asignación que he puesto como ejemplo hace que MatLab genere una matriz llamada **Qt** en la que va a ordenar por columnas los **j** experimentos que realicemos. Será por tanto una matriz con **i** filas y 2 columnas. Se sale del bucle cuando en la estructura condicional **if Qt(i,j)<=0; break; end;** la condición es verdadera y se devuelve el control al bucle **j** para que realice su segunda iteración. Una vez que se sale del bucle

j los experimentos han concluido y pasamos a hacer un gráfico con los comandos **plot**, **title**, **xlabel** y **ylabel**. Parte de la realización del gráfico consiste en poner un título distinto a cada representación de la curva de Laffer y por ello hacemos uso de la estructura condicional

```
if j==1
    gtext('Reagan')
else
    gtext('Ricardo')
end
```

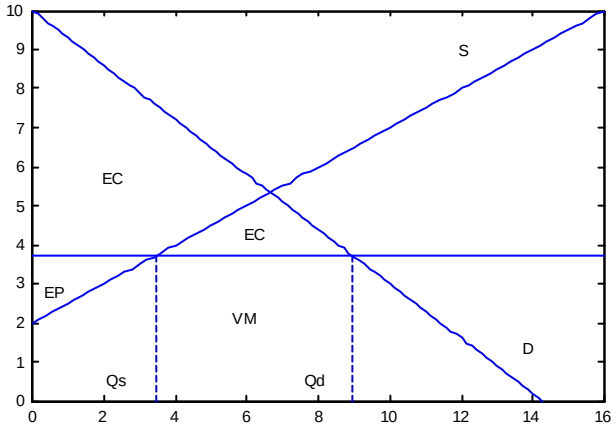
que dice: si estamos en la primera iteración (j= 1) usa el comando **gtext** (get text, o atrapa el texto que escribo a continuación: **Reagan**) para que en el lugar del gráfico que yo quiera pueda poner la cadena de texto con sólo pulsar el botón izquierdo del ratón en el lugar seleccionado. En caso contrario, es decir **j**$\neq$ 1 utiliza el mismo comando pero ahora para introducir la cadena de texto **Ricardo**. El comando **gtext** no devuelve el control al programa hasta que la operación de representar la cadena de texto haya terminado. Finalmente haz una malla (**grid**) y por último mantén el gráfico a la espera de que llegue otro que quiero pintar sobre la misma ventana (**hold on**). Este último comando es muy útil cuando los vectores que queremos representar son de distinta dimensión. El comando **hold on** (ver »**hold on**) adapta el tamaño de la ventana para dar cabida a ambas representaciones en ejes automáticamente seleccionados por MatLab. El programa termina.

Ronald Reagan adoptó una política fiscal muy expansiva a través de una reducción de impuestos que supuestamente le debía reportar unos ingresos fiscales superiores con los que poder financiar un proyecto bélico costosísimo. El resultado fue un déficit fiscal sin precedentes. Aquí tenemos una prueba empírica de que Ricardo tenía razón y que los economistas neoclásicos que postulan una función de oferta agregada más bien inelástica están en lo cierto. Por el contrario aquellos que piensan que la función de oferta agregada es elástica están equivocados.

6.2 Aranceles

Otra forma en la que los gobiernos intervienen en los mercados es a través de la imposición fiscal en las fronteras. Los impuestos que gravan a las mercancías que provienen del resto del mundo se llaman aranceles y como vamos a ver tienen un impacto muy negativo sobre el bienestar de la población planetaria. Vamos a suponer que hay muchos países productores de la mer-

cancía Q y que el país que estamos tratando es un pequeño productor de esa mercancía y tampoco es un gran consumidor de la misma. Bienes de estas características son casi todos los productos agrícolas que conocemos. Vamos a imaginar que el resto del mundo tiene tantos productores que la función de oferta internacional de la mercancía Q tiene una elasticidad infinita y por tanto la función de oferta queda bien descrita a través de la relación funcional $p(Q) = p^I$, es decir, al precio internacional[2] p^I se oferta cualquier cantidad, a un precio inferior a p^I no encontramos proveedores y a precios superiores nadie quiere comprar. En estas condiciones podemos ver cuál será el excedente de productores y consumidores nacionales cuando el mercado en el que se vende la mercancía Q está abierto y no hay intervenciones.



La figura muestra las funciones de demanda y oferta nacionales marcadas con D y S respectivamente. Dados los precios internacionales y representados a través de la recta horizontal vemos que se produce un exceso de oferta sobre demanda. Este exceso se cierra con importaciones y que viene dado por $Q_d - Q_s$. Las áreas marcadas con EC son el excedente de los consumidores (la diferencia entre la disposición a pagar y lo efectivamente pagado) y el área marcada con EP es el excedente de los productores (la diferencia entre el precio mínimo para iniciar la producción y lo efectivamente cobrado). Como se puede apreciar el incremento del bienestar derivado del comercio internacional recae exclusivamente sobre los consumidores, ya que el excedente de los productores es mayor cuando la economía está cerrada. Esta

[2]Los precios internacionales son simplemente los precios corregidos por los tipos de cambio nominales.

simple observación nos va a ayudar a entender muchas cosas en relación al comercio internacional. Vamos a construir un pequeño programa que haga algunos cálculos, pero antes tenemos que hacer otros cálculos nosotros mismos y a mano. Las cantidades marcadas con Q_d y Q_s en el gráfico de arriba vienen dadas por la proyección sobre las funciones de demanda y oferta de los precios internacionales:

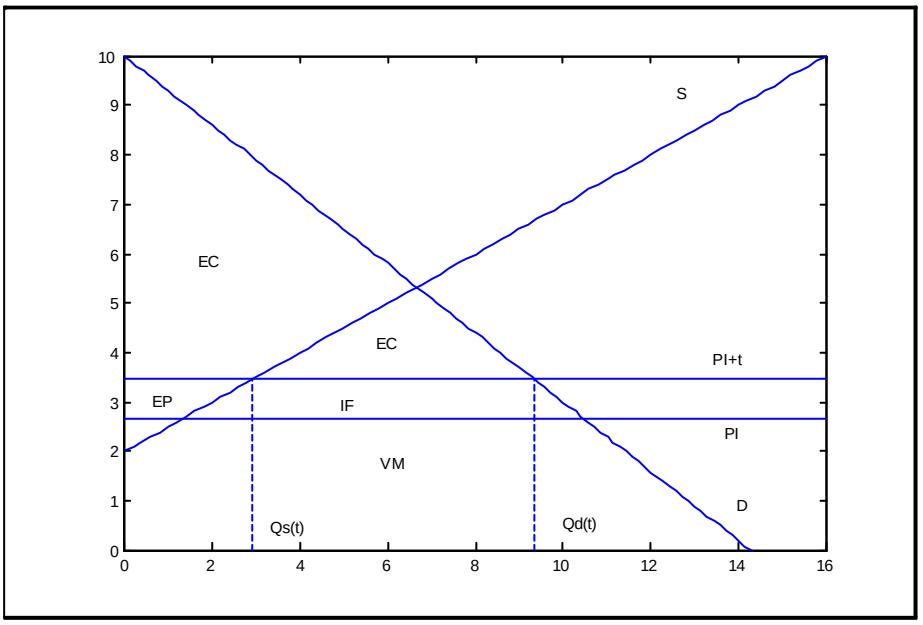
$$\begin{aligned} Q_d &= \frac{a - p^I}{b} \\ Q_s &= \frac{p^I - c}{d} \\ M &= Q_d - Q_s \\ VM &= p^I(Q_d - Q_s) \end{aligned}$$

Una vez que tenemos las cantidades en demanda a los precios internacionales y el volumen de producción nacional a los precios internacionales podemos calcular las cantidades físicas importadas restando una de otra. El valor de mercado de las importaciones es el producto del precio unitario por la cantidad importada. La magnitud VM (Valor de las iMportaciones) es muy importante porque representa la transferencia monetaria que desde este mercado se realiza al resto del mundo. El programa comenzaría así:

```
%Este programa hace cálculos con una función de oferta y demanda en
%un mercado abierto al resto del mundo
clear
%definición de los parámetros
a = 10;
b = 0.1;
c = 2;
d = 0.5;
%Cálculo del equilibrio de la economía cerrada
Qstar = (a-c)/(b+d);
pstar = (a*d+b*c)/(b+d);
%Definición de los precios internacionales
frac = 0.7;          %frac<1 implica importaciones
pinter = frac*pstar;
%Obtención de las importaciones
Qd = (a-pinter)/b;
Qs = (pinter-c)/d;
```

M = Qd-Qs;
VM = pinter*(Qd-Qs);

A partir de aquí podemos empezar un ejercicio similar al que hemos realizado en la sección anterior. Hay que notar que desde el punto de vista de la teoría económica el ejercicio de encontrar una curva de Laffer para el sector exterior tiene más sentido que hacerlo para la economía cerrada. La razón está en que muchos mercados de interés en la Unión Europea y en los Estados Unidos están prácticamente cerrados a la competencia exterior debido a los aranceles, por tanto aquí sí podemos estar seguros de que las tasas arancelarias están muy por encima del máximo de la curva de Laffer. Nuestro interés ahora será no sólo determinar cuáles son las ganancias de bienestar para los consumidores de estos dos países citados sino también calcular la transferencia de renta que se efectuaría al resto del mundo como consecuencia de una disminución en los aranceles. Para incluir al gobierno basta con imponer un impuesto a las importaciones, que no es otra cosa que afectar a la función de oferta agregada del resto del mundo. El arancel transforma el precio internacional haciéndolo pasar de p^I a $p^I + t$, tal y como se muestra en el siguiente gráfico.



Como podemos ver, el valor de las importaciones (VM) ha disminuido porque han disminuido las importaciones, ha aparecido un excedente del

estado al que he marcado con IF (de ingreso fiscal), el excedente de los productores ha aumentado, debido a que ahora el resto del mundo es menos competitivo y da espacio a que empresas nacionales menos competitivas que el resto del mundo puedan vender sus productos, disminuyendo, como es lógico, el excedente de los consumidores. Al tiempo se produce la famosa pérdida irrecuperable de eficiencia o de bienestar. El cálculo de las expresiones para $Q_s(t)$ y $Q_d(t)$ es muy sencillo. La función de oferta está formada por los pares $(p, Q_s) \in \Re^2_+$ tales que la igualdad $p = c + dQ_s$ sea cierta. Por ser una función (aplicación biyectiva a la que a cada elemento del conjunto inicial le corresponde una y solamente una imagen) para obtener la expresión de $Q_s(t)$ basta con hacer que $p = p^I + t$, de modo que sustituyendo en la expresión de la función de oferta obtenemos $p^I + t = c + dQ_s$. Haciendo lo mismo con la función de demanda y despejando las cantidades obtenemos:

$$\begin{aligned} Q_d(t) &= \frac{a - (p^I + t)}{b} \\ Q_s(t) &= \frac{(p^I + t) - c}{d} \end{aligned}$$

Que como vemos son expresiones muy parecidas a las que hemos encontrado con anterioridad. De hecho si hacemos $t = 0$ en las dos ecuaciones anteriores encontramos las cantidades demandadas y ofertadas que hemos calculado sin impuestos. Para calcular el volumen de importaciones y el valor de las importaciones basta con operar como ya lo hicimos anteriormente.

$$\begin{aligned} M(t) &= Q_d(t) - Q_s(t) \\ VM(t) &= p^I(Q_d(t) - Q_s(t)) \end{aligned}$$

A nuestro programa le podemos añadir las siguientes líneas de comandos:

```
%Aranceles
tasa = 0.5;
ptasa = tasa+pinter;
%Obtención de las importaciones con aranceles
Qdt = (a-ptasa)/b;
Qst = (ptasa-c)/d;
Mt = Qdt-Qst;
VMt = pinter*(Qdt-Qst);
```

y así calcular las importaciones y el valor de las importaciones (o la transferencia monetaria al resto del mundo). Los excedentes de productores y

consumidores también pueden ser introducidos,

$$\begin{aligned} EC &= \frac{1}{2}\left[a-\left(p^I+t\right)\right]Q_d(t)=\frac{1}{2b}\left[a-\left(p^I+t\right)\right]^2, \\ EP &= \frac{1}{2}\left[\left(p^I+t\right)-c\right]Q_s(t)=\frac{1}{2d}\left[\left(p^I+t\right)-c\right]^2, \\ IF &= t\left(Q_d(t)-Q_s(t)\right)=\frac{t}{db}\left(ad-(d+b)(p^I+t)+bc\right)=t\frac{d+b}{db}\left(p^*-(p^I+t)\right). \end{aligned}$$

De las anteriores expresiones podemos ver que *i*) el excedente de los consumidores es decreciente en t, *ii*) el excedente de los productores es creciente en t, y *iii*) los ingresos fiscales son 0 en dos casos, uno obvio cuando $t=0$ y otro igualmente obvio cuando $p^I+t=p^*$. En el segundo caso el arancel es tal que iguala los precios internacionales con los precios que hubieran prevalecido en un mercado cerrado. Este nivel de arancel se corresponde con $t=p^*-p^I=T.E.C.$ (que es nuestra humanitaria Tarifa Exterior Comunitaria). Como la función $IF(t)$ es continua en t y vemos que hay un intervalo en el que toca dos veces el 0 entonces tiene que tener (al menos) un máximo. Echando una derivadilla con respecto a t e igualando a 0, obtenemos que la tasa en la que los ingresos fiscales por importaciones son máximos es

$$t=\frac{1}{2}\left(p^*-p^I\right).$$

Incorporar estas expresiones a nuestro programa es escribir:

```
%Obtención de los excedentes, arancel máximo y TEC
EC = 1/(2*b)*(a-ptasa)^2;
EP = 1/(2*d)*(ptasa-c)^2;
IF = tasa*(b+d)/(b*d)*(pstar-ptasa);
tmax = 1/2*(pstar-pinter);
AEC = pstar-pinter;
```

La única anotación sobre este último trozo de código es que a la TEC la he llamado AEC en el programa. La razón es que el programa que estoy ejecutando se llama **tec.m** y no es conveniente que un programa se llame igual que una de las variables que contiene ya que MatLab puede confundirse. Ahora vamos a hacer una pequeña modificación sobre el programa que nos permita calcular los excedentes, los valores de mercado de las importaciones y el volumen de importaciones cuando vamos variando los aranceles desde 0 hasta el valor máximo TEC.

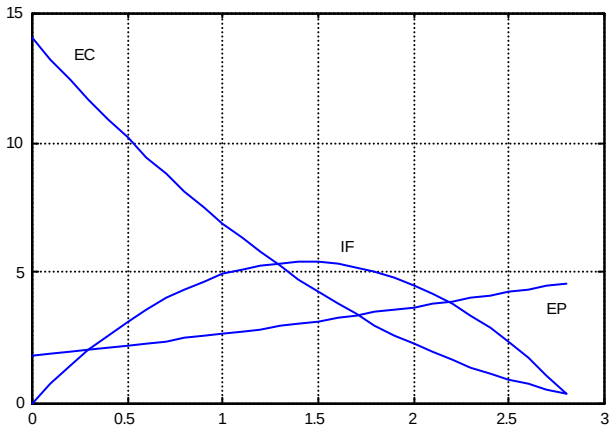
```
%Este programa hace cálculos con una función de oferta y demanda en
```

```
%un mercado abierto al resto del mundo. Computa la curva de Laffer para el
%sector exterior
clear
%Definición de los parámetros
a = 10;
b = 0.4;
c = 2;
d = 6;
%Cálculo del equilibrio de la economía cerrada
Qstar = (a-c)/(b+d);
pstar = (a*d+b*c)/(b+d);
%Definición de los precios internacionales
frac = 0.7; %frac<1 implica importaciones
pinter = frac*pstar;
TEC = pstar-pinter;
tmax = 1/2*(pstar-pinter);
tasa = [0:0.1:TEC];
for i=1:length(tasa)
    ptasa(i)= tasa(i)+pinter;
    Qd(i) = (a-ptasa(i))/b;
    Qs(i) = (ptasa(i)-c)/d;
    M(i) = Qd(i)-Qs(i);
    VM(i) = pinter*(Qd(i)-Qs(i));
    EC(i) = 1/(2*b)*(a-ptasa(i))^2;
    EP(i) = 1/(2*d)*(ptasa(i)-c)^2;
    IF(i) = tasa(i)*(b+d)/(b*d)*(pstar-ptasa(i));
end
plot(tasa, EP, 'r', tasa, IF, 'b', tasa, EC, 'm')
figure
plot(tasa, VM./(pinter*Qs)*100, 'b')
ylabel('% del valor de la producción nacional transferido')
xlabel('Arancel')
grid
```

Este programa no tiene novedades salvo la división *array* efectuada en la cadena **plot(tasa, VM./(pinter*Qs)*100, 'b')** donde se divide componente a componente el vector **VM** entre el vector resultante del producto

pinter*Qs. El resultado de la cuenta **VM./(pinter*Qs)*100** es la transferencia realizada al resto del mundo como porcentaje del valor de mercado a precios internacionales de la producción doméstica. Para los parámetros escogidos, cuando el arancel es 0, el valor de ese porcentaje es del 980.6452% que no es un número despreciable. Si $t = 0$ el valor de mercado de la producción nacional sería $p^I * Q_s(t = 0) = 6.65 * 0.775 = 5.1538$. El valor de las importaciones con $t = 0$ es **VM(1)**= 50.54 y de aquí el resultado. Otra novedad es el comando **figure** que ordena a MatLab abrir una nueva ventana de gráficos para hacer un nuevo **plot**. El resto es conocido.

Vamos a utilizar el programa para analizar sistemas fiscales que aun siendo ineficientes sean pareto superiores a las asignaciones impuestas por un arancel del tipo TEC.



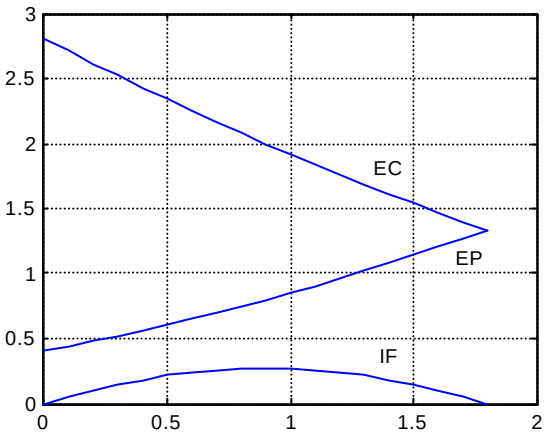
Excedentes con distintos aranceles

Este gráfico ha sido generado con el conjunto de parámetros

$$\begin{array}{cccc} \text{Demanda} & a = 10 & b = 0.4 \\ \text{Oferta} & c = 2 & d = 6 \end{array}$$

donde se puede apreciar que la curva de demanda es bastante elástica en relación a la curva de oferta. Esta elasticidad hace que la distorsión inducida por el arancel sea muy fuerte. La distorsión llega al punto que si fijáramos el arancel en el nivel $t_{\max}$, el incremento en el precio con arancel sobre el precio internacional sería de un 64% aproximadamente, y el ingreso fiscal sería mucho más que suficiente como para compensar a los productores nacionales de

las pérdidas de bienestar ocasionadas por una disminución del arancel desde TEC hasta $t_{\max}$. Este no es, sin embargo, el único escenario posible. Probemos ahora con unas funciones de oferta y demanda con mucha pendiente, es decir, muy inelásticas (que como sabemos inducen una menor distorsión en presencia de impuestos).



Excedentes con funciones inelásticas

Para hacer este gráfico he cambiado los parámetros dentro del archivo **teclaf.m** a estos otros:

$$\begin{array}{ccc} \text{Demanda} & a = 10 & b = 6 \\ \text{Oferta} & c = 2 & d = 6 \end{array}$$

Como se puede apreciar, al ser la distorsión menor, los ingresos fiscales son menores y no pueden compensar a los productores de las pérdidas de bienestar ocasionadas por el paso de un arancel TEC a otro arancel de menor cuantía. Nuestro análisis concluye con la importante consideración de que la receta de política va a ser muy dependiente de la elasticidad de las funciones de oferta y demanda. Por tanto si los distintos productos producidos en el resto del mundo, es decir, fuera de las fronteras de la Unión Europea, tienen distintas elasticidades de oferta y demanda en los mercados nacionales, los aranceles deberían recoger este hecho en vez de aplicarse un arancel flexible como la TEC, que se adapta producto a producto para cerrar los mercados. Un sistema arancelario más razonable para los consumidores nacionales, los productores nacionales, los estados de la Unión Europea, y sobre todo, los

países pobres del resto del mundo, debería tener en cuenta que las distintas elasticidades de demanda y oferta de cada uno de los productos afectan al bienestar de todos ellos. Así, podríamos argumentar junto con Ricardo, que gravar bienes de baja elasticidad es menos lesivo que gravar bienes de elasticidad alta. Si una política así se llevara a cabo, los países del tercer mundo podrían especializarse en la producción de bienes de alta elasticidad con la finalidad de venderlos dentro de las áreas de comercio ricas.

7 Equilibrio

En la sección anterior hemos tratado aspectos de la oferta y la demanda relacionados con la estructura impositiva de los estados y hemos llegado a conclusiones nítidas sobre algunas características que debe cumplir un sistema impositivo que quiera ser muy recaudador y poco distorsionante. A lo largo de toda la sección hemos adoptado una definición de equilibrio que decía que en todo momento del tiempo $Q_d = Q_s$. Esta definición del equilibrio competitivo es sin duda muy útil y ha rendido un servicio a la teoría económica de un valor incalculable. No obstante es posible que alguno de ustedes desee experimentar con definiciones del equilibrio alternativas. En esta sección vamos a formular una variante del equilibrio de mercado según la cual son necesarios algunos periodos de prueba y error hasta que los agentes económicos aprenden cuáles son los precios de equilibrio. Vamos a imaginar que las funciones inversas de demanda y oferta vienen dadas por las expresiones lineales usadas en secciones anteriores. Así $p = a - bQ_d$ y $p = c + dQ_s$ representan a la demanda de mercado y a la oferta de mercado respectivamente. La noción de equilibrio será $Q_d(p^*) = Q_s(p^*) = Q$ y por tanto el par $(p^*, Q^*) = \left(\frac{ad+cb}{b+d}, \frac{a-c}{b+d}\right)$, pero ahora en vez de tener un ajuste instantáneo en un solo periodo vamos a ver un proceso en el que tanto los consumidores como los productores tienen que modificar sus planes de consumo e inversión en el tiempo hasta que encuentran el equilibrio. El proceso de ajuste vendrá dado por una ley de movimiento que gobierna el mercado:

$$p_{t+1} - p_t = \alpha(Q_{dt} - Q_{st}) \text{ con } \alpha > 0$$

Esta ecuación dice que si en el intante de tiempo t, $Q_{dt} - Q_{st} > 0$ entonces $p_{t+1} - p_t > 0$, lo cual significa simplemente que si en el instante t hay un exceso de demanda entonces los precios en $t + 1$ serán mayores que los precios en t. La magnitud del incremento en los precios será una proporción constante

dada por $\alpha > 0$ del exceso de demanda. En caso de haber exceso de oferta, siguiendo el mismo razonamiento, los agentes económicos esperan que los precios bajen ($p_{t+1} < p_t$). Sustituyendo en la ley de movimiento del mercado las cantidades en oferta y demanda en el instante t obtenemos:

$$p_{t+1} - p_t = \alpha \left(\frac{a - p_t}{b} - \frac{p_t - c}{d} \right)$$

que operando levemente se transforma en:

$$p_{t+1} - p_t = \alpha \left(\frac{ad + cb}{db} \right) - \alpha \left(\frac{d + b}{db} \right) p_t$$

si multiplicamos y dividimos el primer cociente del lado derecho de la ecuación anterior por la constante $(d + b)$ y tenemos en cuenta el valor de p^* en el equilibrio obtenemos

$$p_{t+1} - p_t = \alpha \left(\frac{d + b}{db} \right) p^* - \alpha \left(\frac{d + b}{db} \right) p_t$$

o bien,

$$p_{t+1} - p_t = \alpha \left(\frac{d + b}{db} \right) (p^* - p_t)$$

Que como vemos es una ecuación en diferencias de primer orden que nos lleva a pensar que las diferencias entre los precios de hoy y mañana son múltiplos de las diferencias entre los precios de hoy y el precio de equilibrio. Hagamos un pequeño programa para ver el comportamiento de esta ecuación.

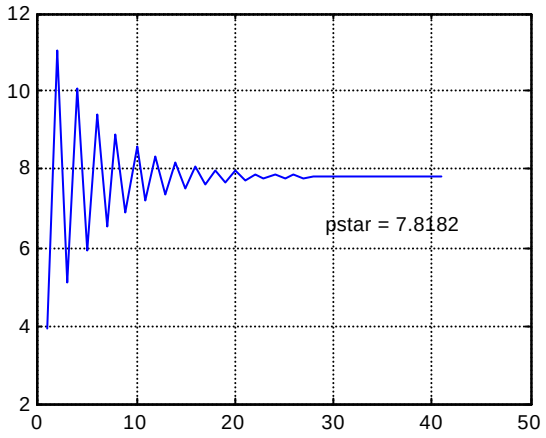
```
%Este programa establece las condiciones para la estabilidad de un
%mercado en el que las expectativas sobre precios dependen linealmente
%de los excesos de demanda.
clear
%Definición de los parámetros
a = 10;
b = 0.6;
c = 2;
d = 1.6;
alpha = 0.8;
T = 40;
```

```
eqfrac=0.5;
%Cálculo del equilibrio
Qstar = (a-c)/(b+d);
pstar = (a*d+b*c)/(b+d);
p(1) = eqfrac*pstar;
for t=1:T
    p(t+1) = alpha*(d+b)/(b*d)*(pstar-p(t))+p(t);
end
plot(p)
```

Desde el punto de vista de la programación este programa no contiene ningún elemento nuevo. El programa establece en un apartado los parámetros del modelo que son los parámetros de la función de demanda y oferta. Se define el parámetro $\boldsymbol{\alpha}$ que es la velocidad del ajuste, el parámetro **T** que es la longitud de la secuencia de precios que deseamos crear, y por último el parámetro **eqfrac** que es la proporción del precio de equilibrio **pstar** con la que vamos a iniciar la secuencia de precios tal y como se puede ver en la cadena **p(1) = eqfrac*pstar;**. Hay que notar que cuando queramos modificar el comportamiento del programa sólo tendremos que hacer cambios en la parte del programa de definición de los parámetros, es decir, no hay números mágicos en ninguna parte. Una vez que tenemos definidos los parámetros del modelo (y del programa) entramos en la parte en la que se calcula el par (**pstar, Qstar**) de equilibrio. Después entramos en un bucle **for end** donde se construye la secuencia de precios. Finalmente hacemos un gráfico para echarle un vistazo. Bajo la colección de parámetros

$$\begin{array}{ll} a = 10 & b = 0.6 \\ c = 2 & d = 1.6 \\ \alpha = 0.8 & \alpha\frac{d+b}{db} = 1.8333 \end{array}$$

obtenemos como resultado de la simulación el siguiente gráfico para p_t.

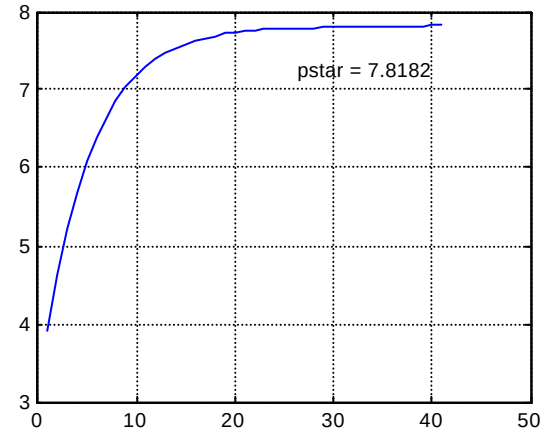


Convergencia no uniforme

En tanto que si realizamos la misma simulación bajo el conjunto de parámetros:

$$\begin{array}{ll} a = 10 & b = 0.6 \\ c = 2 & d = 1.6 \\ \alpha = 0.08 & \alpha\frac{d+b}{db} = 0.1780 \end{array}$$

obtenemos un resultado completamente distinto como se puede ver en la siguiente figura:

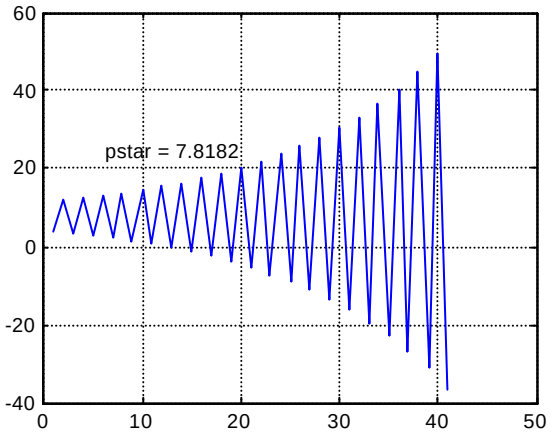


Convergencia uniforme

Que a su vez es completamente distinto de los resultados que se obtienen bajo la siguiente parametrización:

$$\begin{array}{ll} a = 10 & b = 0.6 \\ c = 2 & d = 1.6 \\ \alpha = 0.9 & \alpha\frac{d+b}{db} = 2.0025 \end{array}$$

en la que como podemos ver al no haber cambiado ninguno de los parámetros que caracterizan a la oferta y la demanda el equilibrio es el mismo, pero al modificar la velocidad de ajuste podemos encontrarnos con una secuencia no convergente que nos lleve a tener una secuencia de precios divergente en la que los precios serán negativos en algunos periodos.



No convergencia

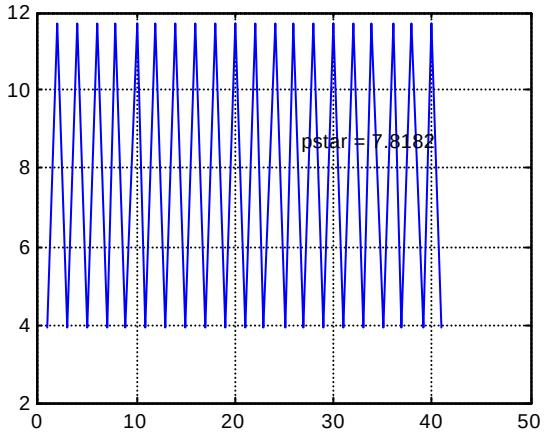
Según lo que hemos visto parece que en el momento en que $\alpha\frac{d+b}{db} > 2$ la secuencia de precios es no convergente. Veamos que sucede si buscamos un α tal que $\alpha\frac{d+b}{db} = 2$. Bastaría con sustituir la línea de código en la que se define α y escribir la siguiente expresión:

alpha = 2*(b*d)/(b+d);

para conseguir el resultado que estamos buscando. Así la nueva parametrización sería:

$$\begin{array}{cc} a = 10 & b = 0.6 \\ c = 2 & d = 1.6 \\ \alpha = 0.8727 & \alpha\frac{d+b}{db} = 2 \end{array}$$

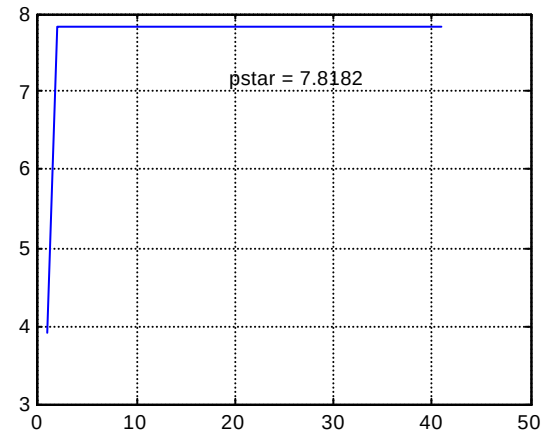
bajo la cual el resultado es:



Ciclo

De la expresión general $p_{t+1} - p_t(k-1) = kp^*$ donde $k = \alpha\frac{d+b}{db}$, observamos que un valor de $k = 1$ nos llevaría al equilibrio en tan sólo un periodo. Veámoslo:

$$\begin{array}{cc} a = 10 & b = 0.6 \\ c = 2 & d = 1.6 \\ \alpha = 0.4364 & \alpha\frac{d+b}{db} = 1 \end{array}$$



Equilibrio instantáneo

Vemos por tanto que nuestra definición de equilibrio instantáneo en el que requeríamos $Q_{dt} = Q_{st}$ para todo t no es más que un caso particular de nuestra nueva noción de equilibrio. Todo esto estaría muy bien si no fuera porque hemos encontrado unos resultados que amenazan a la teoría. Hemos visto que bajo determinadas condiciones los mercados no estarían nunca en equilibrio o se alejarían continuamente de él.

Vamos a imaginar ahora que la función de oferta sufre choques estocásticos en alguno de sus parámetros. Como sabemos, la función de oferta en mercados competitivos representa a la tecnología. Si suponemos que la tecnología está sometida a fluctuaciones de naturaleza desconocida e incontrolable haríamos bien en modelizarlo introduciendo estas distorsiones haciendo a alguno de los parámetros estocástico. Para ello construimos un programa similar al que tenemos con algunas variantes. Veámoslo:

```
%Este programa establece las condiciones para la estabilidad de un
%mercado en el que las expectativas sobre precios dependen linealmente
%de los excesos de demanda. La pendiente de la función de oferta ( o su
% elasticidad) fluctúa aleatoriamente.
clear
%Definición de los parámetros
a = 10;
b = 6;
c = 2;
deq = 1.6;
%alpha = 5.07;
alpha = 0.05*(b*deq)/(b+deq);
T = 500;
for j=1:T
    d(j) = deq+randn;
    Qstar(j) = (a-c)/(b+d(j));
    pstar(j) = (a*d(j)+b*c)/(b+d(j));
    p(1) = pstar(1);
    p(j+1) = alpha*(d(j)+b)/(b*d(j))*(pstar(j)-p(j))+p(j);
end
plot((1:T), p(1:T), 'r',(1:T), pstar, 'b')
sound(p, 50)
```

Cambiando los parámetros de velocidad de ajuste en **alpha = 0.05*(b*deq)/(b+deq);** podemos encontrar dinámicas muy sorprendentes en las que los mercados es-

tán fluctuando cerca del equilibrio y luego sorpresivamente alejándose. La última línea de comandos muestra que los vectores se pueden oir con MatLab.

8 Sistemas lineales

8.1 Método

Con MatLab resolver un sistema de ecuaciones lineales es muy sencillo. Basta con escribir ordenadamente un par de matrices. Imaginemos por ejemplo que deseamos resolver el siguiente sistema de ecuaciones:

$$\begin{aligned} x + 3y + 6z &= 9 \\ x + 9y + 18z &= 7 \\ 2x + 7y + 7z &= 2 \end{aligned}$$

Este sistema lo podemos escribir en notación matricial como:

$$\begin{bmatrix} 1 & 3 & 6 \\ 1 & 9 & 18 \\ 2 & 7 & 7 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 9 \\ 7 \\ 2 \end{bmatrix}$$

O bien, de un modo más compacto, como $A_{3x3} * \overline{x}_{3x1} = c_{3x1}$. Premultiplicando la expresión anterior por la inversa de A tendríamos $(A^{-1}A)\overline{x} = A^{-1}c$. y obtendríamos la solución del sistema:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 6 \\ 1 & 9 & 18 \\ 2 & 7 & 7 \end{bmatrix}^{-1} * \begin{bmatrix} 9 \\ 7 \\ 2 \end{bmatrix}$$

Pues bien, ya no tendríamos más que introducir estas expresiones en un archivo legible por MatLab de la siguiente manera:

```
%Este programa resuelve sistemas de ecuaciones lineales.
%Para hacerlo hay que introducir la matriz de coeficientes A
%e introducir el vector de términos constantes d
clear
A = [1 3 6; 1 9 18; 2 7 7];
d = [9; 7; 2];
```

sol = inv(A)*d

La matriz $\mathbf{A}_{3x3}$ se introduce como hemos visto separando con **;** las columnas. Definimos el vector de términos constantes **d** haciéndolo compatible con la dimensión de **A**, de modo que la multiplicación matricial tenga sentido. Finalmente creamos el vector solución **sol** como el producto de la inversa de **A** (ver »help **inv**) y **d**.

8.2 Análisis input-output

En esta sección vamos aplicar el método de la sección anterior para hacer análisis sectorial de una economía. Las tablas input-output (o insumo-producto en Hispanoamérica) son un registro sistemático de la actividad productiva de un país basado en el sistema de cuentas nacionales. Un modelo input-output Funciona de la siguiente manera:

$$\begin{array}{c} \\ \text{Input} \\ \begin{bmatrix} I \\ II \\ III \\ \vdots \\ N \end{bmatrix} \end{array} \quad \begin{array}{c} \text{Output} \\ \begin{array}{ccccc} I & II & III & \cdots & N \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & & a_{2n} \\ a_{31} & a_{32} & a_{33} & & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & & a_{nn} \end{bmatrix} \end{array}$$

Para producir una unidad de bien es necesaria la participación de unos insumos. Cuando uno toma una simple cerveza en un bar está consumiendo un bien final. Una vez que la cerveza ha sido bebida todo un proceso muy complicado ha finalizado. La cantidad de sectores productivos que han intervenido ha sido enorme. En primer lugar el sector de la construcción debió hacer un edificio en el que el bar estaba, pero el edificio fue diseñado por una arquitecto que trabajaba en el sector servicios y que aprendió su oficio en la universidad. La cerveza fue producida en un lugar lejano usando trabajadores y máquinas y usaron como insumo una cantidad de cebada que a su vez fue producida por un sector primario que se llama agricultura. El campesino utilizó abonos y tierra además de su propio trabajo para hacer crecer la cebada. Este campesino a su vez debió refrescarse de cuando en cuando y lo que hizo fue tomar cerveza (luego la cerveza es un insumo de sí misma). Una vez que la cerveza fue producida hubo que transportarla, para lo cual se usaron camiones y trenes (posiblemente importados los dos).

El sector del transporte también jugó un papel importante como se ve. Al conductor del camión se le pinchó una rueda al entregar la cerveza, de modo que usó parte de los bienes producidos por el sector del caucho y puso una rueda nueva (y luego se tomó una cerveza), además de gasolina tuvo que pagar unos seguros de accidente, con lo que el sector servicios intervino otra vez. El dueño del bar ha pagado además del alquiler del local del bar una factura de luz importante, lo cual alegró a Iberdrola quien construyó un embalse con el que produce electricidad, y este embalse se lo hizo el sector de la construcción naturalmente. Las mesas y las sillas se hicieron con madera y plástico que eran de otro agricultor y de la misma petrolera que hizo el caucho con el que se hizo la rueda que el camionero puso en su camión. Y la historia sigue hasta que uno la interrumpe con un largo trago de cerveza fresca (opps!, se me había olvidado la nevera que la enfrió, que fue producida con...). Esto es una economía en funcionamiento. Nosotros podemos representarla a través de la matriz de arriba. Si la leemos de arriba a abajo en la columna 1 encontramos los requerimientos (unitarios) que el sector I ha necesitado para poder hacer una unidad de su producto. Así decimos con razón que el sector I necesitó a_{11} unidades producidas por él mismo, además necesitó a_{21} unidades producidas por el sector II y a_{31} unidades del sector III, así como ..., y finalmente a_{n1} unidades del sector n. Si leemos de izquierda a derecha por filas nos encontramos con la producción que el primer sector tuvo que realizar para satisfacer las necesidades de los otros sectores. Así, el sector I tuvo que producir a_{11} unidades de producto para usarlas él mismo, produjo también a_{12} unidades a petición del sector II y ..., finalmente el sector n le pidió que produjera a_{1n} unidades para satisfacer sus necesidades productivas. Pero esto no es todo ya que esta descripción que les he hecho sólo ha incluido a las demandas intermedias, es decir, sólo hemos incluido en la tabla las necesidades que cada sector productivo a tenido del resto de sectores. No se olviden de ustedes mismos que son los que al final pidieron la cerveza para beberla, y es lo que se denomina la demanda final de cerveza. La matriz de arriba se llama matriz de insumos intermedios ya que sólo registra las demandas realizadas entre sectores y que utilizan los bienes producidos como un insumo para producir otros bienes. El nivel de producción que la industria I tiene que tener para satisfacer no sólo a la demanda de insumos intermedios, sino a la demanda final es:

$$x_1 = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + d_1$$

o expresado de otro modo

$$(1 - a_{11})x_1 - a_{12}x_2 - \cdots - a_{1n}x_n = d_1$$

Nótese que los coeficientes de este problema estarían tomados directamente de la matriz de coeficientes de la tabla insumo-producto salvo los elementos de la diagonal principal que tendrían un $(1-a_{ii})$. Así, el sistema de ecuaciones sería:

$$\begin{aligned}
(1 - a_{11})x_1 - a_{12}x_2 - \cdots - a_{1n}x_n &= d_1 \\
-a_{21}x_1 + (1 - a_{22})x_2 - \cdots - a_{2n}x_n &= d_2 \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots &= \cdots \\
-a_{n1}x_1 - a_{n2}x_2 - \cdots + (1 - a_{nn})x_n &= d_n
\end{aligned}$$

Que expresado en notación matricial sería lo mismo que escribir:

$$\begin{bmatrix} (1 - a_{11}) & -a_{12} & \cdots & -a_{1n} \\ -a_{21}x_1 & (1 - a_{22}) & \cdots & -a_{2n} \\ \vdots & \vdots & & \vdots \\ -a_{n1} & -a_{n2} & \cdots & (1 - a_{nn}) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}$$

O de un modo más compacto aún, usando la notación estándar de que I_n es la matriz identidad de dimensión $n \times n$, escribimos:

$$(I_n - A_{n\times n})\, x_{n\times 1} = d_{n\times 1}$$

La matriz $(I_n - A_{n\times n})$ se llama matriz tecnológica y suponemos que tiene inversa, que es básicamente lo mismo que decir que no hay procesos tecnológicos que sean una transformación lineal de otros procesos existentes. Si la inversa existe entonces podemos obtener el vector de demandas intermedias que satisfacen, dada la tecnología, a las demandas finales simplemente premultiplicando en ambos miembros de la ecuación por la inversa de la matriz tecnológica:

$$x = (I - A)^{-1}\, d$$

Ahora el ejercicio consiste simplemente en escribir un programa que resuelva el problema de los requerimientos mínimos de producción intermedia con la finalidad de satisfacer la demanda.

Ejercicio: Escribir un programa para encontrar los requerimientos de producción intermedia que haga posible el plan de desarrollo de un país que desee una demanda de $d_1 = 10, d_2 = 5$ y $d_3 = 6$, con una matriz de coeficientes

$$A = \begin{matrix} 0.2 & 0.3 & 0.2 \\ 0.4 & 0.1 & 0.2 \\ 0.1 & 0.3 & 0.2 \end{matrix}$$