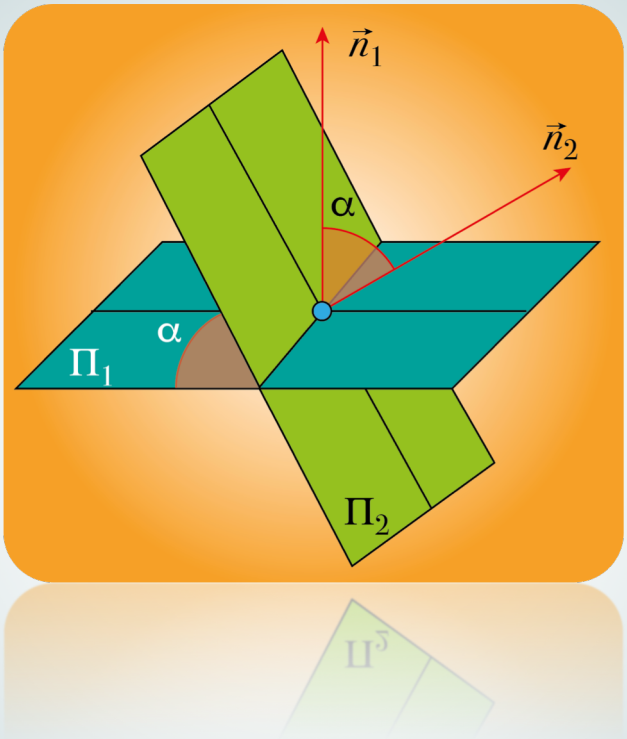
Álgebra y Geometría

Práctica 4. Sistemas de ecuaciones lineales (II)



Rodrigo García Manzanas
Ruth Carballo Fidalgo

Departamento de Matemática Aplicada y
Ciencias de la Computación

Este tema se publica bajo Licencia:
Creative Commons BY-NC-SA 4.0

Grado en Ingeniería Civil

G1954: Álgebra y Geometría

Práctica 4: Sistemas de ecuaciones lineales (II)

Rodrigo García Manzanas (rodrigo.manzanas@unican.es)

Ruth Carballo Fidalgo (ruth.carballo@unican.es)

Objetivos

- Resolución de sistemas homogéneos
- Resolución de sistemas utilizando factorización de matrices
- Resolución de sistemas mediante métodos iterativos

Resolución de sistemas homogéneos

Como sabes, los sistemas homogéneos siempre son compatibles, pudiendo tener solución única (la trivial) o infinitas soluciones. Comencemos con un sistema homogéneo determinado como el siguiente:

$$\begin{cases} x - 3y + 4z &= 0 \\ x - 2y + 5z &= 0 \\ 2x - y - 3z &= 0 \end{cases}$$

```
A = [1 -3 4; 1 -2 5; 2 -1 -3];  % matriz de coeficientes
b = zeros(3,1);  % términos indep.
rank(A)  % R-F: S.C.D.
```

```
ans = 3
```

Por tener solución única, podríamos resolverlo utilizando cualquiera de los métodos $1-6$ que vimos en la práctica anterior (aunque sabemos de antemano que la solución será $x = y = z = 0$):

```
%% Método 1 (escalonada reducida)
rref(A)  % solución: (x=0, y=0, z=0)
```

```
ans = 3×3
     1     0     0
     0     1     0
     0     0     1
```

```
%% Método 2 ("linsolve")
linsolve(A, b)  % solución: (x=0, y=0, z=0)
```

```
ans = 3×1
     0
     0
     0
```

```
%% Método 3 ("\")
A\b  % solución: (x=0, y=0, z=0)
```

```
ans = 3×1
     0
     0
     0
```

```
%% Método 4 (simbólico)
syms x y z real  % defino las incógnitas del sistema como variables simbólicas
X = [x y z]';  % vector de incógnitas simbólicas
sol = solve(A*X == b, [x y z]);  % resuelvo el sistema en su forma matricial
[sol.x, sol.y, sol.z]  % solución: (x=0, y=0, z=0)
```

```
ans = (0  0  0)
```

```
%% Método 5 (inversa)
inv(A)*b  % solución: (x=0, y=0, z=0)
```

```
ans = 3×1
     0
     0
     0
```

```
%% Método 6 (Cramer)
Ax = [b, A(:,2:3)];
Ay = [A(:,1), b, A(:,3)];
Az = [A(:,1:2), b];
[det(Ax)/det(A), det(Ay)/det(A), det(Az)/det(A)]  % solución: (x=0, y=0, z=0)
```

```
ans = 1×3
     0     0     0
```

Pasemos ahora al caso de sistemas homogéneos indeterminados (con infinitas soluciones), como el que se muestra a continuación:

$$\begin{cases} 2x + y + z &= 0 \\ 3x - z &= 0 \\ 10x + 5y + 5z &= 0 \end{cases}$$

```
A = [2 1 1; 3 0 -1; 10 5 5];  % matriz de coeficientes
b = zeros(3,1);  % términos indep.
rank(A)  % R-F: S.C.I.
```

```
ans = 2
```

Tal y como vimos en la práctica anterior, podemos recurrir al cálculo simbólico para resolver este tipo de sistemas:

```
rref(A)  % escalonada reducida por filas --> I.P: {x,y}, P.L.: {z}
```

```
ans = 3×3
    1.0000         0   -0.3333
         0    1.0000    1.6667
         0         0         0
```

```
syms x y z real  % defino las incógnitas del sistema como simbólicas
X = [x y z]';  % construyo el vector de incógnitas simbólicas
sol = solve(A*X == b, [x y]);  % resuelvo en las I.P.
sol = [sol.x sol.y z]  %  la solución del sistema vendrá dada en funcíón
```

```
sol =
```

$$\left(\frac{z}{3} \quad -\frac{5\,z}{3} \quad z\right)$$

```
% de "z", el único P.L.
A*subs(sol, z, -7.964)' - b  % comprobación para z=-7.964
```

```
ans =
```

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Sin embargo, MATLAB cuenta con un comando específico para resolver directamente sistemas homogéneos (ya sean determinados o indeterminados): *null.* Este comando es muy útil (sólo requiere como entrada la matriz de coeficientes del sistema, A), pero hay que saber cómo interpretar el resultado que devuelve. En el caso de sistemas homogéneos determinados como el que utilizamos en el primer ejemplo de esta práctica, *null* devolverá una matriz vacía, lo que indica que la única solución del sistema es la trivial.

$$\begin{cases} x - 3y + 4z &= 0 \\ x - 2y + 5z &= 0 \\ 2x - y - 3z &= 0 \end{cases}$$

```
A = [1 -3 4; 1 -2 5; 2 -1 -3];  % matriz de coeficientes
null(A, 'r')  % sol. trivial: (x=0, y=0, z=0)
```

```
ans =

  3×0 empty double matrix
```

***Nota**: En el comando null, el argumento de entrada "r" es opcional. Sin embargo, nosotros vamos a usarlo siempre, pues da lugar a soluciones compuestas por números racionales (que se pueden expresar como una fracción), que resultan más manejables. Lógicamente, las soluciones devueltas con y sin "r" son igual de válidas.*

En el caso de sistemas homogéneos indeterminados como el que hemos utilizado como segundo ejemplo en esta práctica, la solución vendrá dada por la combinación lineal de las columnas de la matriz que devuelve *null.*

```
A = [2 1 1; 3 0 -1; 10 5 5];  % matriz de coeficientes
solh = null(A, 'r')
```

```
solh = 3×1
    0.3333
   -1.6667
    1.0000
```

En este caso, *null* devuelve una única columna, por lo que la solución del sistema será $\alpha\left(\frac{1}{3}, -\frac{5}{3}, 1\right)$, siendo α un parámetro libre. Comprobemos para un valor cualquiera de α que la solución hallada es correcta:

```
alpha = -6.78;  % fijo un valor cualquiera para el P.L.
sol = alpha*solh;  % solución
A*sol  % comprobación
```

```
ans = 3×1
10^-14 ×
    0.0888
    0.0888
    0.7105
```

Veamos otro sistema homogéneo indeterminado en cuyas soluciones intervengan más parámetros libres, como por ejemplo en este:

$$\begin{cases} 4x + 5y + 6z &= 0 \\ 8x + 10y + 12z &= 0 \end{cases}$$

```
A = [4 5 6; 8 10 12];
rank(A)  % R-F: S.C.I.
```

```
ans = 1
```

```
solh = null(A, 'r')  % la sol. del sistema dependerá de dos P.L.
```

```
solh = 3×2
   -1.2500   -1.5000
    1.0000         0
         0    1.0000
```

En este caso, la solución será $\alpha\left(-\frac{5}{4}, 1, 0\right) + \beta\left(-\frac{3}{2}, 0, 1\right)$. Comprobémoslo:

```
alpha = -1.23; beta = pi/6;  % doy valores cualesquiera a los P.L.
sol = alpha*solh(:,1) + beta*solh(:,2)
```

```
sol = 3×1
    0.7521
   -1.2300
    0.5236
```

```
A*sol  % comprobación
```

```
ans = 2×1
     0
     0
```

Resolución de sistemas utilizando factorización de matrices

En primer lugar, vamos a resolver el sistema del apartado *b)* del ejercicio *10* de la hoja de problemas utilizando la factorización LU:

$$\begin{cases} x-2y-3z &= 2 \\ 2x+y+z &= 1 \\ x+3y-2z &= -1 \end{cases}$$

```
A = [1 -2 -3; 2 1 1; 1 3 -2];  % matriz coefs.
b = [2 1 -1]';  % términos indep.
[rank(A) rank([A b])]  % R-F: S.C.D.
```

```
ans = 1×2
     3     3
```

```
[L, U, P] = lu(A)  % factorizacón PA = LU
```

```
L = 3×3
    1.0000         0         0
    0.5000    1.0000         0
    0.5000   -1.0000    1.0000
U = 3×3
    2.0000    1.0000    1.0000
         0   -2.5000   -3.5000
         0         0   -6.0000
P = 3×3
     0     1     0
     1     0     0
     0     0     1
```

```
P*A - L*U  % comprobación factorización
```

```
ans = 3×3
     0     0     0
     0     0     0
     0     0     0
```

Vemos que la matriz P no es la identidad, lo que quiere decir que MATLAB ha realizado algún intercambio de filas en el proceso de escalonamiento de A. Sabemos que en estas circunstancias se cumplirá la factorización $PA = LU$, o lo que es lo mismo, $A = P^{-1}LU$. Por tanto, el sistema $A\vec{x} = \vec{b}$ podrá escribirse como $P^{-1}LU\vec{x} = \vec{b}$. Si llamamos $U\vec{x} = \vec{y}$, podremos descomponerlo en dos subsistemas que pueden resolverse inmediatamente (comprueba que son triangulares):

- 1) $U\vec{x} = \vec{y}$
- 2) $P^{-1}L\vec{y} = \vec{b}$

```
%% Resuelvo 2)
y = inv(P)*L\b
```

```
y = 3×1
    1.0000
    1.5000
         0
```

```
%% Resuelvo 1)
sol = U\y  % solución: (x=4/5, y=-3/5, z=0)
```

```
sol = 3×1
    0.8000
   -0.6000
         0
```

```
A*sol - b  % comprobación
```

```
ans = 3×1
10^-15 ×
         0
         0
    0.2220
```

Resolvamos ahora el sistema del apartado *a)* del ejercicio *11* utilizando la factorización de Cholesky:

$$\begin{cases} x - y + z = 1 \\ -x + 5y - 5z = 0 \\ x - 5y + 6z = 1 \end{cases}$$

```
A = [1 -1 1; -1 5 -5; 1 -5 6];  % matriz de coefs.
b = [1 0 1]';  % términos indep.
[rank(A) rank([A b])]  % R-F: S.C.D.
```

```
ans = 1×2
     3     3
```

Antes de nada, tendremos que comprobar la matriz A admite factorización de Cholesky.

```
issymmetric(A)  % A simétrica
```

```
ans = logical
   1
```

```
[det(A(1,1)), det(A(1:2,1:2)), det(A)]  % A definida positiva
```

```
ans = 1×3
     1     4     4
```

En estas condiciones se cumplirá que $A = LL^t$, siendo L una matriz triangular inferior.

```
L = chol(A, 'lower')  % factorización de Cholesky
```

```
L = 3×3
     1     0     0
    -1     2     0
     1    -2     1
```

```
A - L*L'  % comprobación factorización
```

```
ans = 3×3
     0     0     0
     0     0     0
     0     0     0
```

Por tanto, el sistema $A\vec{x} = \vec{b}$ podrá escribirse como $LL^t\vec{x} = \vec{b}$. Si llamamos $L^t\vec{x} = \vec{y}$, podremos descomponerlo en dos subsistemas que pueden resolverse inmediatamente (es evidente que son triangulares):

- 1) $L^t\vec{x} = \vec{y}$
- 2) $L\vec{y} = \vec{b}$

```
%% Resuelvo 2)
y = L\b
```

```
y = 3×1
    1.0000
    0.5000
    1.0000
```

```
%% Resuelvo 1)
sol = L'\y  % solución: (x=5/4, y=5/4, z=1)
```

```
sol = 3×1
    1.2500
    1.2500
    1.0000
```

```
A*sol - b  % comprobación
```

```
ans = 3×1
     0
     0
     0
```

Resolución de sistemas mediante métodos iterativos

En esta sección nos centraremos en los métodos de Jacobi y Gauss-Seidel. Pero antes, haremos una breve introducción al uso de funciones en MATLAB. Una **función** es un programa que establece una comunicación con el exterior mediante argumentos de entrada y salida. Las funciones deben guardarse en un fichero con el mismo nombre que la propia función y extensión **.m**. La estructura de cualquier función en MATLAB es la siguiente:

```
function [argumentos de salida] = nombre_funcion(argumentos ...
   de entrada)
% comentarios
instrucciones
end
```

Por ejemplo, la función que se muestra a continuación permitiría calcular el área de un círculo conocido su radio:

```
function [A] = area_circ(r)
    A = pi*r^2;
end
```

Para que MATLAB renonozca esta función tenemos dos opciones:

- Situarnos en la misma ruta (directorio) en la que hayamos salvado la función. Esta opción sólo será válida mientras permanezcamos en el mismo directorio.
- Utilizar el comando *addpath* para incluir en memoria la ruta (directorio) en la que hayamos guardado la función:

```
addpath directorio_donde_has_salvado_la_funcion
```

A partir de este momento, podremos llamar a la función desde la línea de comandos:

```
area_circ(5)  % area de un circulo de radio = 5 unidades
```

Para aplicar los métodos de Jacobi y Gauss-Seidel necesitaremos las funciones *Jacobi* y *GaussSeidel*, respectivamente. Las tienes colgadas en *Moodle;* descárgalas a tu ordenador y utiliza el comando *addpath* para cargarlas en memoria.

A partir de este momento será inmediato resolver sistemas *cuadrados* (mismo número de ecuaciones que de incógnitas) por cualquiera de estos métodos. Como puedes ver, ambas funciones requieren como argumentos de entrada la matriz de coeficientes, el vector de términos independientes, una solución inicial (cualquiera) y el error máximo que estamos dispuestos a cometer con la solución aproximada (haz un *help* de ambas funciones).

Como sabes, la convergencia de los algoritmos de Jacobi y Gauss-Seidel sólo está asegurada para sistemas cuya matriz de coeficientes sea *diagonalmente dominante*. Por tanto, antes de resolver el sistema que aparece a continuación por cualquiera de ambos métodos, conviene comprobar que efectivamente la matriz de coeficientes sea diagonalmente dominante:

$$\begin{cases} 10x - y - 2z &= 6 \\ x + 5y - 3z &= 4 \\ 4x - 2y - 8z &= 5 \end{cases}$$

```
A = [10 -1 -2; 1 5 -3; 4 -2 -8]  % matriz de coefs.
```

```
A = 3×3
    10    -1    -2
     1     5    -3
     4    -2    -8
```

```
A_sindiag = tril(A, -1) + triu(A, 1)  % matriz de coefs., poniendo ceros
```

```
A_sindiag = 3×3
     0    -1    -2
     1     0    -3
     4    -2     0
```

```
% en la diagonal
abs(diag(A)) > sum(abs(A_sindiag), 2)  % efectivamente, A es diagonalmente
```

```
ans = 3×1 logical array
   1
   1
   1
```

```
% dominante
```

Una vez hemos comprobado que la convergencia está asegurada, es inmediato resolver el sistema:

```
b = [6 4 5]'; x0 = [0 0 0]'; err = 0.001;

% Jacobi
[sol_Jacobi, niter_Jacobi] = Jacobi(A, b, x0, err)  % sol. aproximada (10
```

```
sol_Jacobi = 3×1
    0.5510
    0.4175
   -0.4539
niter_Jacobi = 10
```

```
% iteraciones)
A*sol_Jacobi - b  % comprobación
```

```
ans = 3×1
10^-3 ×
   -0.2291
    0.0361
   -0.3630
```

```
% Gauss-Seidel
[sol_GaussSeidel, niter_GaussSeidel] = GaussSeidel(A, b, x0, err)  % sol. aprox.
```

```
sol_GaussSeidel = 3×1
    0.5510
    0.4174
```

```
   -0.4539
niter_GaussSeidel = 6
```

```
% (6 iteraciones)
A*sol_GaussSeidel - b  % comprobación
```

```
ans = 3×1
10^-3 ×
    0.1827
   -0.3823
         0
```

Ejercicios propuestos

Ejercicio 1:

Resuelve los siguientes sistemas:

$$a)\ \begin{bmatrix} -5 & 10 & 11 & 10 & -67 \\ 2 & -4 & 5 & 12 & 47 \\ -1 & 2 & 21 & 34 & 27 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad b)\ \begin{bmatrix} -5 & -7 & 11 \\ 9 & -4 & 5 \\ -1 & 5 & 21 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Ejercicio 2:

Obtén la solución general del siguiente sistema:

$$\begin{bmatrix} -1 & 2 & -1 & 4 & 5 \\ -2 & 3 & -10 & -14 & 0 \\ 3 & -2 & 4 & 14 & 6 \\ -6 & 4 & -8 & -28 & -12 \\ -3 & 2 & -4 & -14 & -6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Ejercicio 3:

Si se puede, resuelve el siguiente sistema:

$$\begin{bmatrix} 0 & 4 & 11 & -9 \\ 4 & 0 & -5 & 0 \\ 11 & -5 & 3 & -3 \\ -9 & 0 & -3 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = \begin{bmatrix} 4 \\ -3 \\ 0 \\ 1 \end{bmatrix}$$

- *a)* Utilizando la factorización LU
- *b)* Utilizando la factorización de Cholesky

Ejercicio 4:

Resuelve por Jacobi y por Gauss-Seidel los siguientes sistemas. ¿Cuántas iteraciones has necesitado para llegar a una solución aproximada (con un error de 0.001) en cada caso? Prueba qué ocurre al comenzar el proceso iterativo desde distintas soluciones iniciales.

a)
$$\begin{cases} 3x - y + z = 4 \\ 2x + 5y + 2z = -5 \\ x + 2y + 4z = 20 \end{cases}$$

b)
$$\begin{cases} 2x = 3 \\ x + 1.5y = 4.5 \\ -3y + 0.5z = -6.6 \\ 2x - 2y + z + t = 0.8 \end{cases}$$

c)
$$\begin{cases} 2x - y + z = 1 \\ 3x + 3y + 9z = 0 \\ 3x + 3y + 5z = 4 \end{cases}$$