

Métodos Numéricos

Departamento de Matemática Aplicada II

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Lección 2: Sistemas de ecuaciones lineales

Índice

1. Sistemas compatibles determinados	2
1.1. La eliminación de Gauss con pivoteo parcial	3
1.2. Normas	7
1.3. Condicionamiento de un sistema de ecuaciones lineales	10
2. Sistemas sobredeterminados	13
2.1. Soluciones en el sentido de los mínimos cuadrados	13
2.2. La factorización QR	17
2.3. Factorización QR reducida y mínimos cuadrados	19
3. Apéndice A: Matrices dispersas	21
4. Apéndice B: Reordenamientos matriciales y el fenómeno de llenado	27
5. Apéndice C: El método de Householder	32
6. Soluciones de los proyectos	34

Los sistemas de ecuaciones lineales no solo aparecen en una gran variedad de áreas científicas, sino también indirectamente en la solución numérica de numerosos problemas puramente matemáticos. Por ejemplo, surgen como etapas intermedias en la resolución de problemas de optimización, de aproximación de funciones, de ecuaciones diferenciales ordinarias, de ecuaciones en derivadas parciales, etc. Sin duda, el estudio de métodos numéricos para la resolución de sistemas de ecuaciones lineales es uno de los pilares del análisis numérico.

En la práctica, y casi exclusivamente, las matrices que encontramos asociadas a los sistemas de ecuaciones lineales, o bien son cuadradas e invertibles, o bien rectangulares (con más filas que columnas) y de rango máximo. En el primer caso, dichos sistemas tienen solución única en el sentido habitual y, en el segundo, también solución única, pero en el sentido de los mínimos cuadrados.

1. Sistemas compatibles determinados

Por **sistema de ecuaciones lineales** entenderemos toda ecuación del tipo $Ax = b$, donde A es una matriz real de orden $m \times n$ y b un vector de \mathbb{R}^m . En esta sección nos centraremos en el caso en que la matriz A es cuadrada ($m = n$) e invertible y, por tanto, el sistema $Ax = b$ tiene solución única. Lógicamente, el problema numérico a tratar será la determinación efectiva y eficiente de dicha solución.

Desde un punto de vista puramente algebraico, la solución de $Ax = b$ es simplemente

$$x = A^{-1}b,$$

donde A^{-1} es la matriz inversa de A . Sin embargo, desde un punto de vista computacional, calcular la inversa para obtener posteriormente la solución es un procedimiento costoso, innecesario y sensible a los errores de redondeo. Actualmente, el método de propósito general considerado más eficiente para resolver dichos sistemas de ecuaciones se basa en la denominada eliminación de Gauss con pivoteo parcial.

Para un usuario de MATLAB®, la resolución de un sistema $Ax = b$ consiste simplemente en ejecutar

Command Window

```
>> x=A\b
```

Es interesante observar que la orden `\` de MATLAB®, denominada «barra invertida» (*backslash*), no es sino una notación sencilla que denota todo un abanico de distintos métodos numéricos.

Ejemplo 1.1 ► Sistema compatible determinado

El sistema lineal de ecuaciones

$$\begin{bmatrix} 3 & 2 & 0 \\ 4 & -1 & 5 \\ 7 & 1 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 13 \\ 20 \end{bmatrix}$$

posee solución única, puesto que su matriz de coeficientes tiene determinante no nulo. La ejecución en MATLAB® del siguiente conjunto de instrucciones proporciona la

solución.

```

Command Window
>> format
>> A=[3 2 0;4 -1 5;7 1 -8];
>> det(A)

ans =

    143
>> b=[7 13 20]';
>> x=A\b

x =

    3.0000
   -1.0000
    0.0000

```

1.1. La eliminación de Gauss con pivoteo parcial

El método de eliminación de Gauss consiste en la transformación de un sistema de ecuaciones dado, $Ax = b$, en otro equivalente, $Ux = b'$, donde U es una matriz triangular superior.

El primer paso consiste en restar múltiplos de la primera ecuación a las demás ecuaciones para hacer desaparecer (eliminar) la primera incógnita. Posteriormente, el proceso se repite con la segunda incógnita, y así sucesivamente, hasta que la última incógnita pueda despejarse.

El proceso de triangularización de la matriz A consta de $n - 1$ etapas,

$$A := A^{(1)} \rightarrow A^{(2)} \rightarrow \dots \rightarrow A^{(n-1)} \rightarrow A^{(n)} := U,$$

donde las matrices $A^{(j)}$, $j = 2, \dots, n$, tienen ceros debajo de los $j - 1$ primeros elementos diagonales. Así, la matriz en la etapa j viene dada por

$$A^{(j)} = \begin{bmatrix} a_{11}^{(j)} & a_{12}^{(j)} & \dots & a_{1j}^{(j)} & a_{1j+1}^{(j)} & \dots & a_{1n}^{(j)} \\ 0 & a_{22}^{(j)} & \dots & a_{2j}^{(j)} & a_{2j+1}^{(j)} & \dots & a_{2n}^{(j)} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & a_{jj}^{(j)} & a_{jj+1}^{(j)} & \dots & a_{jn}^{(j)} \\ 0 & \dots & 0 & a_{j+1j}^{(j)} & a_{j+1j+1}^{(j)} & \dots & a_{j+1n}^{(j)} \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & a_{nj}^{(j)} & a_{nj+1}^{(j)} & \dots & a_{nn}^{(j)} \end{bmatrix}.$$

La matriz en la etapa $j+1$ surge al anular las entradas situadas bajo el elemento diagonal $a_{jj}^{(j)}$. Se obtiene, así,

$$A^{(j+1)} = \begin{bmatrix} a_{11}^{(j)} & a_{12}^{(j)} & \dots & a_{1j}^{(j)} & a_{1j+1}^{(j)} & \dots & a_{1n}^{(j)} \\ 0 & a_{22}^{(j)} & \dots & a_{2j}^{(j)} & a_{2j+1}^{(j)} & \dots & a_{2n}^{(j)} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & a_{jj}^{(j)} & a_{jj+1}^{(j)} & \dots & a_{jn}^{(j)} \\ 0 & \dots & 0 & 0 & a_{j+1j+1}^{(j+1)} & \dots & a_{j+1n}^{(j+1)} \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & a_{nj+1}^{(j+1)} & \dots & a_{nn}^{(j+1)} \end{bmatrix}.$$

Para conseguir los ceros ubicados bajo $a_{jj}^{(j)}$ se construyen los multiplicadores

$$l_{j+1j} = \frac{a_{j+1j}^{(j)}}{a_{jj}^{(j)}}, \quad l_{j+2j} = \frac{a_{j+2j}^{(j)}}{a_{jj}^{(j)}}, \quad \dots, \quad l_{nj} = \frac{a_{nj}^{(j)}}{a_{jj}^{(j)}},$$

y se realizan las correspondientes operaciones elementales:

$$\begin{aligned} F_{j+1} &\leftarrow F_{j+1} - l_{j+1j} F_j, \\ F_{j+2} &\leftarrow F_{j+2} - l_{j+2j} F_j, \\ &\vdots \\ F_n &\leftarrow F_n - l_{nj} F_j, \end{aligned}$$

donde F_i denota la fila i -ésima de la matriz.

En la etapa final del proceso llegamos a la matriz triangular superior U , cuya relación con la matriz inicial A se establece a través de la denominada factorización LU de A ,

$$A = LU,$$

donde

$$L = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n1} & \dots & l_{nn-1} & 1 \end{bmatrix}$$

es una matriz triangular inferior con unos en la diagonal y con los multiplicadores elementales calculados en el proceso de eliminación de Gauss en sus lugares correspondientes.

Para poder llevar a cabo el proceso de eliminación de Gauss descrito es necesario que los elementos diagonales que vayamos obteniendo, $a_{ii}^{(i)}$, sean todos distintos de cero. Estos elementos diagonales no nulos se conocen como **pivotes del método de eliminación de Gauss**. Si en algún paso de la eliminación se obtiene algún $a_{ii}^{(i)}$ nulo, será preciso permutar la fila i -ésima con alguna posterior.

Debemos añadir que el proceso de intercambio de filas es también significativo desde el punto de vista de la estabilidad numérica. En general, la eliminación de Gauss es inestable numéricamente si los pivotes son muy pequeños o, equivalentemente, si los multiplicadores son muy grandes¹. Para evitar esta situación, se introduce la estrategia de pivoteo parcial: en el paso i -ésimo de la eliminación de Gauss se determina el menor índice k_i donde se alcanza el máximo de los elementos

$$|a_{ii}^{(i)}|, \dots, |a_{ni}^{(i)}|$$

y, posteriormente, se intercambian las filas i y k_i y se realiza la eliminación. Conviene observar que en este caso los multiplicadores verifican $|l_{ki}| \leq 1$, para todo $k = i + 1, \dots, n$.

Para la mayoría de las matrices que aparecen en la práctica, el comportamiento del pivoteo parcial es más que aceptable, siendo la estrategia con la que suelen diseñarse los algoritmos de eliminación de Gauss.

El procedimiento que combina (i) las reducciones a cero de los elementos situados bajo la diagonal (eliminación de Gauss) con (ii) los intercambios de filas (pivoteo parcial) es conocido como **método de eliminación de Gauss con pivoteo parcial**. Al finalizar el proceso, el producto LU no es igual a la matriz A , sino a una matriz PA obtenida permutando las filas de A , lo que conduce a la denominada **factorización $PA = LU$ de la matriz A** . En la diagonal de U aparecen los pivotes del método de eliminación de Gauss. La matriz P es una **matriz de permutación**, que se obtiene aplicando sobre la matriz identidad las mismas permutaciones de filas que se hacen en el proceso de eliminación de Gauss. Es fácil comprobar que las matrices de permutación cumplen $P^{-1} = P^T$, es decir, son matrices ortogonales.

Una vez obtenida una factorización $PA = LU$, podemos resolver el sistema $Ax = b$ de la siguiente manera:

1. Primero permutamos las ecuaciones para obtener el sistema

$$PAx = Pb$$

o, equivalentemente,

$$LUx = Pb.$$

Si denotamos Ux por \tilde{b} , obtenemos el sistema

$$L\tilde{b} = Pb.$$

Dado que L es triangular inferior con diagonal de unos, la resolución de $L\tilde{b} = c$, donde c denota el vector Pb , se realiza utilizando el siguiente procedimiento, conocido como **algoritmo de sustitución progresiva**:

$$\begin{aligned} \tilde{b}_1 &= c_1, \\ \tilde{b}_j &= c_j - (l_{j1}\tilde{b}_1 + \dots + l_{j,j-1}\tilde{b}_{j-1}), \quad \text{para } j = 2, \dots, n. \end{aligned}$$

2. Una vez obtenido el vector \tilde{b} , podemos calcular la solución resolviendo el sistema

$$Ux = \tilde{b}.$$

¹La aparición de pivotes pequeños puede ocasionar, entre otros efectos indeseables, un aumento significativo de los errores de redondeo.

Puesto que U es triangular superior, en la resolución de este último sistema usaremos el **algoritmo de sustitución regresiva**:

$$x_n = \frac{\tilde{b}_n}{u_{nn}},$$

$$x_j = \frac{\tilde{b}_j - (u_{j,j+1}x_{j+1} + \cdots + u_{jn}x_n)}{u_{jj}}, \text{ para } j = n-1, \dots, 1.$$

En suma, vemos que la resolución de un sistema compatible determinado $Ax = b$ de orden n requiere tres pasos:

1. Obtener una factorización $PA = LU$ de la matriz A mediante eliminación de Gauss con pivoteo parcial y considerar el sistema equivalente $PAx = Pb$.
2. Resolver el sistema triangular inferior $Lz = Pb$ mediante el algoritmo de sustitución progresiva.
3. Resolver el sistema triangular superior $Ux = z$ mediante el algoritmo de sustitución regresiva.

Es fácil comprobar que el cálculo de la factorización $PA = LU$ de A requiere un número de *flop*, esto es, de operaciones en punto flotante, del orden de $\frac{2n^3}{3}$ y cada una de las resoluciones de los sistemas triangulares requiere un número de *flop* del orden de n^2 . Se tiene, por tanto, que el costo computacional de la resolución del sistema $Ax = b$ recae fundamentalmente en el primer paso.

La resolución de un sistema genérico $Ax = b$, donde A es una matriz cuadrada, se realiza en MATLAB® con la orden «barra invertida», \backslash , tal como se mencionó en el inicio de la lección. Esta orden efectúa los tres pasos vistos previamente. Si la matriz A es triangular, el operador \backslash utiliza el método de sustitución regresiva o progresiva, según corresponda.

La orden `lu` de MATLAB® proporciona la factorización $PA = LU$ asociada a la implementación de la eliminación de Gauss con pivoteo parcial. En concreto, si ejecutamos

```
Command Window
>> [L,U,P]=lu(A)
```

obtenemos como argumentos de salida, respectivamente, las matrices L , U y P .

Proyecto 1.1 (solución en página 34)

Considera la matriz A y el vector b siguientes:

$$A = \begin{bmatrix} 1 & -7 & 1 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 10 \\ 7 \\ 16 \end{bmatrix}.$$

Mediante el método de eliminación de Gauss con pivoteo parcial, calcula analíticamente una factorización $PA = LU$ de la matriz A . Utiliza esta factorización para calcular, de nuevo analíticamente, la solución del sistema lineal de ecuaciones $Ax = b$.

Ejercicio 1.1

Considera de nuevo la matriz A y el vector b dados en el proyecto 1.1.

1. Calcula en MATLAB® una factorización $PA = LU$ de A y comprueba que coincide con la que obtuviste analíticamente en ese proyecto.
2. Resuelve el sistema lineal de ecuaciones $Ax = b$ utilizando la factorización calculada en el apartado anterior y comprueba que tiene la misma solución que la obtenida analíticamente en el proyecto mencionado.
3. Resuelve otra vez el sistema $Ax = b$, pero utiliza ahora la orden `\` de MATLAB® y observa que se alcanza de nuevo la misma solución.
4. ¿Cómo podrías calcular en MATLAB® la matriz A^{-1} utilizando la orden `\`?

Problema 1.1

1. Diseña una función de MATLAB® que calcule, aplicando el método de eliminación de Gauss, pero sin realizar pivoteo parcial, las matrices L y U de la factorización $A = LU$ de una matriz A cuadrada de orden n .
2. Sean matrices aleatorias A_n de órdenes $n = 10, 20, 40, \dots, 1280$. Para cada uno de estos valores de n , considera el sistema $A_n x = b_n$, donde b_n es la suma de las columnas de A_n . ¿Cuál es su solución exacta?

Elabora una tabla con los errores relativos que se cometen al resolver los sistemas $A_n x = b_n$, para $n = 10, 20, 40, \dots, 1280$, mediante:

- i) la función diseñada en el apartado 1;
- ii) la orden `\` de MATLAB®.

Observarás un crecimiento del error cuando se resuelve el sistema sin realizar pivoteo parcial. ¿A qué crees que se debe este crecimiento del error?

Observarás también cierto crecimiento del error en el caso en que el sistema se resuelve con la orden `\`. Esto puede parecer ahora misterioso. Para entender qué está ocurriendo, regresa a este problema una vez hayas visto el punto 1.3 de la lección.

1.2. Normas

Dado un sistema $Ax = b$, es habitual que los coeficientes de A y b no se conozcan (o no se puedan manejar) exactamente. Por ejemplo, dichos valores pueden proceder de experimentos o simplemente ser valores numéricos que requieren redondearse para ser volcados en un ordenador. En otras palabras, cuando resolvemos un sistema $Ax = b$ en realidad, y en casi todos los casos, estamos trabajando con cierto sistema perturbado (quizá solo levemente) $\hat{A}x = \hat{b}$. De ahí que sea crucial medir la sensibilidad de la solución de $Ax = b$ ante perturbaciones en A y/o en b .

Para el análisis matemático de tales perturbaciones se utilizan las normas vectoriales y matriciales, que desempeñan un papel similar al concepto de módulo o valor absoluto en los números reales.

Recordemos algunas de las normas vectoriales más empleadas en \mathbb{R}^n . Dado un vector $x = [x_1, \dots, x_n]^T$, se define su p -norma, $\|\cdot\|_p$, donde $1 \leq p \leq \infty$, como

$$i) \|x\|_p := \sqrt[p]{|x_1|^p + \dots + |x_n|^p}, \text{ si } 1 \leq p < \infty,$$

$$ii) \|x\|_\infty := \max\{|x_1|, \dots, |x_n|\}, \text{ si } p = \infty.$$

Sin duda, la más utilizada es la **norma vectorial euclídea** ($p = 2$),

$$\|x\|_2 = \sqrt{x_1^2 + \dots + x_n^2},$$

pero también conviene conocer la norma de Manhattan ($p = 1$),

$$\|x\|_1 = |x_1| + \dots + |x_n|,$$

y la norma del máximo ($p = \infty$). Estas normas pueden calcularse en MATLAB® con la orden `norm`³.

Ejemplo 1.2 ► Normas de vector

Las normas euclídea, de Manhattan y del máximo del vector $[1, 2, -4, 7, -2]^T$ se calculan en MATLAB® mediante las siguientes instrucciones:

```
Command Window
>> x=[1 2 4 -7 -2];
>> norm(x) % = norm(x,2) norma euclídea

ans =

    8.6023

>> norm(x,1) % 1-norma

ans =

    16

>> norm(x,Inf) % norma del máximo

ans =

    7
```

²Véase la nota 4.

³La opción por defecto de esta orden es la norma euclídea.

Las normas matriciales se introducen a partir de las normas vectoriales. En concreto, para una matriz A de orden $m \times n$ se define su p -norma, $\|\cdot\|_p$ ⁴, como

$$\|A\|_p := \max \left\{ \frac{\|Ax\|_p}{\|x\|_p} : x \in \mathbb{R}^n, x \neq 0 \right\},$$

donde $1 \leq p \leq \infty$.

Si la norma de vector utilizada es la euclídea, la correspondiente norma de matriz recibe también el nombre de **norma matricial euclídea**.

Si bien la definición anterior es útil, conceptualmente hablando, no es realmente operativa desde un punto de vista computacional. Una forma alternativa de calcular la norma euclídea de una matriz A involucra los autovalores⁵ de $A^T A$. Observemos que los autovalores de esta matriz, al ser siempre simétrica y semidefinida positiva, son todos reales y no negativos. En particular, podemos ordenarlos de la siguiente manera: $0 \leq \lambda_1 \leq \dots \leq \lambda_n$. Con esta notación, se puede probar que

$$\|A\|_2 = \sqrt{\lambda_n}.$$

Ejemplo 1.3 ► Norma euclídea de matriz

Consideremos la matriz

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \\ 0 & 1 \end{bmatrix}.$$

Se tiene que

$$A^T A = \begin{bmatrix} 5 & 10 \\ 10 & 26 \end{bmatrix},$$

cuyos autovalores son 1 y 30. La norma euclídea de A es, por tanto,

$$\|A\|_2 = \sqrt{30} \approx 5.4772.$$

Verificamos este resultado en MATLAB[®]^a:

Command Window

```
>> A=[2 3;1 4;0 1];
>> norm(A) % = norm(A,2)

ans =

    5.4772
```

^aLa orden `norm` de MATLAB[®] permite calcular tanto normas de vectores como normas de matrices. También en el caso de matrices la opción por defecto de la orden es la norma euclídea.

⁴Si una propiedad se verifica independientemente de la p -norma que se utilice (sea de vector o de matriz), no será necesario especificar el subíndice p en el símbolo de la norma.

⁵La orden de MATLAB[®] `eig` (de *EIGenvalue*, autovalor en alemán) permite obtener todos los autovalores de una matriz cuadrada dada, teniendo en cuenta la multiplicidad algebraica de cada autovalor.

A continuación, suponiendo que las dimensiones son compatibles con la operación, describimos varias propiedades de las normas de matriz:

- $\|A\| \geq 0$, para toda matriz A de orden $m \times n$.
- $\|\alpha A\| = |\alpha| \|A\|$, para toda matriz A de orden $m \times n$ y todo $\alpha \in \mathbb{R}$.
- $\|A + B\| \leq \|A\| + \|B\|$, para cualquier par de matrices A y B .
- $\|I_n\| = 1$, siendo I_n la matriz identidad de orden n .
- $\|O_{m \times n}\| = 0$, siendo $O_{m \times n}$ la matriz nula de orden $m \times n$.
- $\|Ax\| \leq \|A\| \|x\|$, para toda matriz A y todo vector x .
- $\|AB\| \leq \|A\| \|B\|$, para cualquier par de matrices A y B .
- Si Q es una matriz ortogonal⁶, entonces $\|AQ\|_2 = \|QA\|_2 = \|A\|_2$, para cualquier matriz A . En particular, $\|Q\|_2 = 1$.

1.3. Condicionamiento de un sistema de ecuaciones lineales

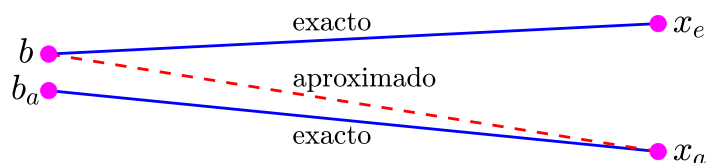
Cuando se resuelve un sistema de ecuaciones lineales $Ax = b$ en aritmética de punto flotante, tanto la matriz A como el vector b sufren perturbaciones debidas al redondeo. Estos errores de redondeo se propagan y afectan a la exactitud de la solución x . El grado de precisión de la solución depende directamente de la buena o mala condición del sistema.

El estudio del condicionamiento de un sistema de ecuaciones lineales $Ax = b$ puede enfocarse respecto a perturbaciones en el vector b o respecto a perturbaciones en la matriz A ⁷. En ambos casos, una misma expresión de la norma matricial de A sirve eficazmente como número de condición.

- Consideremos inicialmente una perturbación $b_a = b + \delta_b$ en el segundo miembro del sistema «exacto» $Ax = b$. Si denotamos por x_e la solución exacta del sistema, se tiene que $Ax_e = b$. Sea $x_a = x_e + \delta_x$ la solución exacta del sistema «perturbado» $Ax = b_a$. Por tanto, x_a es la solución aproximada de $Ax = b$.

Es decir:

1. El sistema exacto $Ax = b$ tiene a x_e como solución exacta, esto es, $Ax_e = b$.
2. El sistema exacto $Ax = b$ tiene a x_a como solución aproximada.
3. El sistema perturbado $Ax = b_a$ tiene a x_a como solución exacta, es decir, $Ax_a = b_a$ o, lo que es lo mismo, $A(x_e + \delta_x) = b + \delta_b$.



⁶Una matriz cuadrada Q es ortogonal si $Q^{-1} = Q^T$.

⁷En la práctica se dan ambas situaciones.

Puesto que $Ax_e = b$, de

$$A(x_e + \delta_x) = Ax_e + A\delta_x = b + \delta_b$$

se deduce que

$$A\delta_x = \delta_b.$$

En consecuencia

$$\delta_x = A^{-1}\delta_b.$$

Por tanto,

$$\|\delta_x\| \leq \|A^{-1}\| \|\delta_b\|. \quad (1)$$

Por otro lado, de $Ax_e = b$ se deduce que

$$\|b\| \leq \|A\| \|x_e\|.$$

Suponiendo que b y x_e son vectores no nulos, podemos expresar la desigualdad anterior en la forma

$$\frac{1}{\|x_e\|} \leq \frac{\|A\|}{\|b\|}. \quad (2)$$

Multiplicando las desigualdades (1) y (2) obtenemos

$$\frac{\|\delta_x\|}{\|x_e\|} \leq \|A^{-1}\| \|A\| \frac{\|\delta_b\|}{\|b\|},$$

que es otra forma de escribir

$$\text{Rel}(x_a) \leq \|A^{-1}\| \|A\| \text{Rel}(b_a).$$

- Cuando se consideran perturbaciones (invertibles) $A_a = A + \delta_A$ en el sistema $Ax = b$ (junto con las correspondientes perturbaciones en la solución $x + \delta_x$), también puede derivarse una expresión asintóticamente similar. En concreto, para $\|\delta_A\| \approx 0$, se obtiene

$$\frac{\|\delta_x\|}{\|x_e\|} \lesssim \|A^{-1}\| \|A\| \frac{\|\delta_A\|}{\|A\|},$$

o sea,

$$\text{Rel}(x_a) \lesssim \|A^{-1}\| \|A\| \text{Rel}(A_a).$$

En otras palabras, la expresión $\|A^{-1}\| \|A\|$ es un factor de amplificación que mide hasta qué punto pueden crecer los errores relativos cometidos en los datos de entrada A y b .

A tenor de estas desigualdades con los errores relativos, es lógico definir el **número de condición** de la matriz invertible A , $\text{cond}(A)$ ⁸, como

$$\text{cond}(A) := \|A^{-1}\| \|A\|.$$

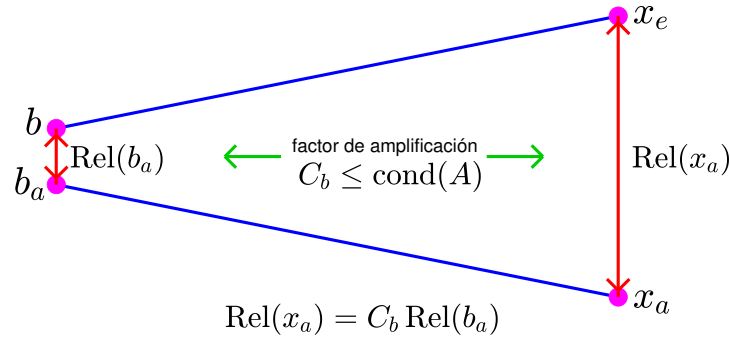
Observemos que $\text{cond}(A)$ es, de hecho, una cota superior del factor «real» de amplificación C_b del error cometido en b y una cota superior aproximada del factor «real» de amplificación C_A del error cometido en A . Estos factores «reales» de amplificación C_b y C_A son, en general, desconocidos.

⁸Cuando sea necesario explicitar que el número de condición se ha calculado utilizando la norma $\|\cdot\|_p$, escribiremos $\text{cond}_p(A)$.

En suma,

$$i) \operatorname{Rel}(x_a) = C_b \operatorname{Rel}(b_a) \leq \operatorname{cond}(A) \operatorname{Rel}(b_a),$$

$$ii) \operatorname{Rel}(x_a) = C_A \operatorname{Rel}(A_a) \lesssim \operatorname{cond}(A) \operatorname{Rel}(A_a).$$



Proyecto 1.2 (solución en página 38)

Comprueba las siguientes propiedades del número de condición:

1. $\operatorname{cond}(A) \geq 1$, para toda matriz cuadrada A .
2. $\operatorname{cond}_2(AQ) = \operatorname{cond}_2(QA) = \operatorname{cond}_2(A)$, para toda matriz ortogonal Q y toda matriz cuadrada A del mismo orden que Q .
3. $\operatorname{cond}_2(Q) = 1$, para toda matriz ortogonal Q .

Ejemplo 1.4 ► Números de condición

Sea A una matriz cuadrada.

- Si $\operatorname{cond}(A) = 1$, se tiene que

$$\operatorname{Rel}(x_a) \leq \operatorname{Rel}(b_a), \quad \operatorname{Rel}(x_a) \lesssim \operatorname{Rel}(A_a).$$

Esta situación es óptima, puesto que el error, si bien se transmite inevitablemente al resolver el sistema lineal, no crece por encima del error de los datos de entrada. Por consiguiente, es razonable esperar que todos los dígitos de la solución x_a sean significativos correctos, ya que los errores de A y b se deberán, con toda probabilidad, únicamente al redondeo, esto es, serán del orden de 10^{-16} .

- Si $\operatorname{cond}(A) = 10^5$, se tiene que

$$\operatorname{Rel}(x_a) \leq 10^5 \operatorname{Rel}(b_a), \quad \operatorname{Rel}(x_a) \lesssim 10^5 \operatorname{Rel}(A_a).$$

La situación ahora ha empeorado, puesto que el error transmitido tiene un margen de crecimiento de hasta cien mil veces el error de los datos de entrada. En consecuencia, es razonable esperar que la solución del sistema lineal de ecuaciones x_a pueda llegar a perder hasta 5 dígitos significativos correctos y constar, finalmente, de tan solo 11.

Problema 1.2

Se define la matriz de Hilbert de orden n , H_n , como aquella matriz cuyas entradas valen

$$H_n(i, j) = \frac{1}{i + j - 1}, \quad i, j = 1, 2, \dots, n.$$

1. Construye una función de MATLAB® que, dado n , devuelva la matriz de Hilbert de orden n . Utiliza posteriormente la orden `hilb` de MATLAB® para comprobar que tu función está bien diseñada.
2. Calcula los números de condición de las 15 primeras matrices de Hilbert.
3. Para $n = 5, 10, 15$, sean los vectores b_n formados por la suma de las columnas de H_n . ¿Cuál es la solución exacta de los sistemas $H_n x = b_n$? Aproxima dichas soluciones con la orden `\` de MATLAB®. Comenta y justifica las diferencias en precisión observadas en las tres soluciones numéricas.

Proyecto 1.3 (solución en pág 40)

El propósito de este proyecto es convencerte de que en los problemas reales las matrices con las que te vas a encontrar no tienen, en general, un número de condición tan elevado como el que poseen las matrices de Hilbert, estudiadas en el problema 1.2. Las matrices de Hilbert son la excepción, no la norma.

Genera aleatoriamente en MATLAB® un millón matrices cuyo orden sea también un número aleatorio comprendido entre dos y cien. Calcula sus números de condición y clasifícalos según su tamaño. ¿Qué observas?

2. Sistemas sobredeterminados

Hay muchas situaciones en las que se desea obtener cierto modelo matemático lineal que ajuste un conjunto de datos conocidos. Esto conduce usualmente a la resolución de un sistema de ecuaciones lineales con más ecuaciones que incógnitas, que casi siempre resulta ser incompatible. Para dichos sistemas se introduce un concepto nuevo de solución (que coincide con el usual cuando el sistema es compatible) denominado solución en el sentido de los mínimos cuadrados.

2.1. Soluciones en el sentido de los mínimos cuadrados

Sean una matriz A real de orden $m \times n$ y un vector $b \in \mathbb{R}^m$. En lo que sigue supondremos que el rango de A es máximo. Si $m > n$, diremos que $Ax = b$ es un **sistema lineal sobredeterminado**. En estas circunstancias, es muy improbable que el sistema sea compatible. Por ello, introducimos una generalización del concepto de solución. Se dice que $\tilde{x} \in \mathbb{R}^n$ es una **solución en el sentido de los mínimos cuadrados** del sistema $Ax = b$ si minimiza la norma

euclídea del residuo $r = Ax - b$, es decir, si

$$\|A\tilde{x} - b\|_2 \leq \|Ax - b\|_2,$$

para todo $x \in \mathbb{R}^n$. La solución en el sentido de los mínimos cuadrados \tilde{x} es, por tanto, el punto donde se alcanza el mínimo

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2.$$

Es fácil deducir que la solución en el sentido de los mínimos cuadrados es única. También es inmediato comprobar que la solución en el sentido de los mínimos cuadrados de un sistema compatible determinado coincide con la solución única del sistema⁹.

Desde un punto de vista geométrico, el teorema de la mejor aproximación establece que la solución en el sentido de los mínimos cuadrados siempre existe y es justamente la solución clásica del sistema

$$Ax = P_{\text{col}(A)}b,$$

donde $P_{\text{col}(A)}b$ denota la proyección ortogonal de b sobre el subespacio vectorial $\text{col}(A)$ generado por las columnas de A .

En particular, si \tilde{x} es la solución en el sentido de los mínimos cuadrados, entonces

$$A\tilde{x} - b \perp \text{col}(A),$$

esto es,

$$A^T(A\tilde{x} - b) = 0,$$

de donde

$$A^T A \tilde{x} = A^T b.$$

El sistema $A^T A x = A^T b$ es siempre compatible determinado y se conoce con el nombre de **ecuaciones normales de Gauss**. La solución única de este sistema es, por tanto, la solución en el sentido de los mínimos cuadrados del sistema $Ax = b$.

Es posible también deducir las ecuaciones normales de Gauss sin necesidad de utilizar argumentos geométricos. De hecho, la solución en el sentido de los mínimos cuadrados es un mínimo de la función real de n variables

$$f(x) = \|Ax - b\|_2^2 = (Ax - b)^T(Ax - b) = x^T A^T A x - 2x^T A^T b + b^T b.$$

Al imponer que el gradiente de esta función se anule,

$$\nabla f(x) = 2A^T A x - 2A^T b = 0,$$

surgen las ecuaciones normales de Gauss,

$$A^T A x = A^T b.$$

La solución única \tilde{x} de este sistema es el único punto crítico de la función y se convierte, por tanto, en la única candidata a solución en el sentido de los mínimos cuadrados, a falta

⁹Cuando sea necesario distinguir entre ambos conceptos de solución, llamaremos «clásica» a la solución del sistema compatible determinado.

de verificar, finalmente, que se trata de un mínimo. Observemos que la matriz hessiana de f viene dada por

$$H_f(x) = 2A^T A.$$

Puesto que $A^T A$ es una matriz simétrica definida positiva, \tilde{x} es un mínimo relativo. De hecho, es un mínimo global y único. En definitiva, \tilde{x} es la solución en el sentido de los mínimos cuadrados del sistema $Ax = b$.

Un procedimiento razonable, a priori, para resolver las ecuaciones normales de Gauss sería ejecutar

```
Command Window
>> x=(A'*A)\(A'*b)
```

Sin embargo, en muchas ocasiones este planteamiento conduce a resultados poco precisos. El problema radica en que las ecuaciones normales de Gauss pueden estar, en general, mal condicionadas, debido a que el número de condición de $A^T A$ puede ser mucho mayor que el de A ¹⁰.

Ejemplo 2.1 ► Ecuaciones normales de Gauss

Consideremos la matriz

$$A = \begin{bmatrix} 1.000001 & 1 & 1.000005 \\ 1 & 1 & 1.000008 \\ 1 & 1 & 1.000001 \\ 1 & 1 & 1.000002 \end{bmatrix}$$

Calculamos en MATLAB® su número de condición:

```
Command Window
>> A=[1.000001 1 1.000005;1 1 1.000008;1 1 1.000001;1 1 1.000002];
>> format short e
>> cond(A)

ans =

5.7090e+06
```

En consecuencia, la solución en el sentido de los mínimos cuadrados de un sistema que tuviera A por matriz puede llegar a perder hasta 6 dígitos significativos correctos y constar solamente de 10.

Sin embargo, si calculamos esta solución a través de las ecuaciones normales de Gauss, la solución en el sentido de los mínimos cuadrados puede verse afectada por una pérdida

¹⁰En concreto, se puede probar que el número de condición de $A^T A$ coincide con el cuadrado del número de condición de A .

de hasta 13 dígitos significativos correctos (y constar, por tanto, únicamente de 3), debido al tamaño del número de condición de $A^T A$ ^a:

```
Command Window
>> cond(A'*A)

ans =

    3.2584e+13
```

^aSi calculamos en MATLAB® el número de condición de A^2 obtenemos $3.2593e+13$. La discrepancia con el valor del número de condición de $A^T A$, esto es, $3.2584e+13$, se debe a errores de redondeo.

En resumen, los métodos que se han estudiado en las secciones anteriores de esta lección no son, en general, adecuados para el problema que estamos considerando en esta sección. Para la resolución eficiente de los problemas de mínimos cuadrados, MATLAB® utiliza un algoritmo basado en la denominada factorización QR de una matriz (Q es un sustituto de la letra O en *Orthogonal* y R procede de *upper Rectangular*). La factorización QR y el algoritmo basado en esta factorización se estudian, respectivamente, en las dos secciones siguientes. El método que permite obtener una factorización QR completa de una matriz se describe en el apéndice C.

Cuando deseamos obtener la solución en el sentido de los mínimos cuadrados de un sistema lineal sobredeterminado $Ax = b$, ejecutamos directamente la instrucción

```
Command Window
>> x=A\b
```

La orden `\` de MATLAB® detecta que, en este caso, la matriz A no es cuadrada y aplica el algoritmo basado en la factorización QR de la matriz A mencionado previamente¹¹.

Ejemplo 2.2 ► Solución en el sentido de los mínimos cuadrados

Dado el sistema lineal de ecuaciones

$$\begin{cases} x_1 + 2x_2 = 3, \\ 2x_1 + 3x_2 = 5, \\ x_1 + 3x_2 = 2, \end{cases}$$

vamos a comprobar que es incompatible y a obtener una solución en el sentido de los mínimos cuadrados. Veremos, además, que esta solución es única.

Se puede constatar analíticamente la incompatibilidad del sistema (por ejemplo, aplicando el método eliminación de Gauss a la matriz ampliada), pero lo haremos utili-

¹¹Es interesante notar que este algoritmo basado en la factorización QR de la matriz permite calcular también la solución (clásica) de un sistema compatible determinado.

zando MATLAB®. Observemos en el siguiente cálculo que el rango de la matriz A no coincide con el rango de la matriz ampliada $[A \ b]$, luego el sistema es, efectivamente, incompatible. El rango de A es máximo, lo que implica que la solución en el sentido de los mínimos cuadrados es única.

```
Command Window
>> A=[1 2;2 3;1 3];
>> b=[3 5 2]';
>> rank(A)

ans =

     2

>> rank([A b])

ans =

     3
```

Vamos, finalmente, a calcular la solución:

```
Command Window
>> format
>> x=A\b

x =

     3.0000
    -0.2727
```

2.2. La factorización QR

Dada una matriz A de orden $m \times n$, con $m \geq n$, se denomina **factorización QR reducida** de A a toda expresión $A = QR$, donde Q es una matriz de orden $m \times n$ cuyas columnas forman un conjunto ortonormal de vectores de \mathbb{R}^m (en particular, Q es una matriz ortogonal si $m = n$) y R es una matriz triangular superior de orden n . Asimismo, se denomina **factorización QR completa** de A a toda expresión $A = QR$, donde Q es una matriz ortogonal de orden m y R es una matriz trapezoidal superior (esto es, sus elementos r_{ij} , para $i > j$, son nulos) de orden $m \times n$.

La orden `qr` de MATLAB® proporciona ambas factorizaciones, tanto la reducida como la completa.

Ejemplo 2.3 ► Factorizaciones QR completa y reducida

Consideremos la matriz

$$A = \begin{bmatrix} 2 & 0 \\ 2 & 3 \\ 2 & 3 \\ 2 & 0 \end{bmatrix}.$$

Una factorización QR completa se obtiene como sigue:

```
Command Window
>> A=[2 0;2 3;2 3;2 0];
>> [Q,R]=qr(A)

Q =

   -0.5000    0.5000   -0.1000   -0.7000
   -0.5000   -0.5000   -0.7000    0.1000
   -0.5000   -0.5000    0.7000   -0.1000
   -0.5000    0.5000    0.1000    0.7000

R =

   -4    -3
    0    -3
    0     0
    0     0
```

Observemos que

```
Command Window
>> Q'*Q

ans =

   1.0000    0.0000    0.0000   -0.0000
   0.0000    1.0000    0.0000    0
   0.0000    0.0000    1.0000   -0.0000
  -0.0000     0   -0.0000    1.0000
```

Esto es, $Q^T Q = I_n$, donde I_n denota la matriz identidad de orden n , dado que Q es una matriz ortogonal.

Asimismo, la factorización QR reducida se obtiene con la misma orden `qr`, pero introduciendo un segundo argumento:

```

Command Window
>> [Q,R]=qr(A,0)

Q =

   -0.5000    0.5000
   -0.5000   -0.5000
   -0.5000   -0.5000
   -0.5000    0.5000

R =

   -4    -3
    0    -3

```

Como puede verse, se han eliminado las dos últimas columnas de la matriz Q completa y las dos últimas filas de la matriz R completa.

Las columnas de la matriz Q reducida forman un conjunto ortonormal de vectores, de donde se verifica

```

Command Window
>> Q'*Q

ans =

   1.0000    0.0000
   0.0000    1.0000

```

2.3. Factorización QR reducida y mínimos cuadrados

Sea A una matriz de orden $m \times n$, con $m > n$, y de rango máximo. Consideremos el sistema lineal sobredeterminado $Ax = b$, donde b un vector de \mathbb{R}^m . Puesto que $m > n$, la orden «barra invertida» de MATLAB® busca, por defecto, la solución en el sentido de los mínimos cuadrados del sistema. Veamos cómo la calcula.

Supongamos que se ha obtenido una factorización QR reducida de A . Recordemos que Q es de orden $m \times n$ y R es cuadrada de orden n . Puesto que A y Q tienen rango máximo (por definición, las columnas de Q forman un conjunto ortonormal de vectores), R es invertible. Dado que la solución en el sentido de los mínimos cuadrados x que buscamos verifica las ecuaciones normales de Gauss, se tiene que

$$A^T Ax = A^T b.$$

Imponiendo que $A = QR$, obtenemos

$$R^T Q^T QRx = R^T Q^T b.$$

Pero $Q^T Q = I_n$, de donde,

$$R^T R x = R^T Q^T b.$$

Puesto que R es invertible, el sistema anterior equivale al sistema triangular superior

$$R x = Q^T b,$$

que puede resolverse mediante el algoritmo de sustitución regresiva.

Ejemplo 2.4 ► Factorizaciones QR completa y reducida

El sistema lineal sobredeterminado

$$\begin{bmatrix} 2 & 0 \\ 2 & 3 \\ 2 & 3 \\ 2 & 0 \end{bmatrix} x = \begin{bmatrix} 1 \\ 0 \\ 1 \\ -1 \end{bmatrix}$$

podría resolverse, indistintamente, de las dos maneras recogidas en el siguiente código:

```
Command Window
>> format long
>> A=[2 0;2 3;2 3;2 0];
>> b=[1 0 1 -1]';
>> x=A\b

x =

    0.0000000000000000
    0.1666666666666667

>> [Q,R]=qr(A,0);
>> x=R\u{Q}'*b)

x =

    0.0000000000000000
    0.1666666666666667
```

Para matrices con $m \approx n$, el costo computacional de resolver en el sentido de los mínimos cuadrados el sistema $Ax = b$ mediante el procedimiento descrito anteriormente es aproximadamente el doble de lo que requiere la resolución de un sistema cuadrado $Ax = b$ de orden n mediante eliminación de Gauss. Este mayor costo computacional está más que justificado teniendo en cuenta la mayor precisión que se alcanza usando la factorización QR , dado que, a diferencia de las ecuaciones normales de Gauss, en el caso del sistema $Rx = Q^T b$ el número de condición de la matriz R coincide con el número de condición de A (observemos que Q tiene número de condición uno).

3. Apéndice A: Matrices dispersas

Una **matriz dispersa** es una matriz de gran tamaño en la que la mayoría de los elementos son cero. Posiblemente el caso más sencillo de matriz dispersa sea el de las matrices banda, esto es, aquellas matrices cuyos elementos no nulos se encuentran en una banda en torno a la diagonal principal.

En general, toda matriz banda A de orden n tiene asociados dos números enteros no-negativos, k_l y k_u , conocidos como anchura de banda inferior y anchura de banda superior, respectivamente, tales que, para cada $i = 1, 2, \dots, n$, $a_{ij} = 0$ si $j \notin [i - k_l, i + k_u]$. Asimismo, la anchura de banda (total) de A se define como $k_l + 1 + k_u$.

Ejemplo 3.1 ► Matriz banda

Los elementos no nulos de la matriz

$$\begin{bmatrix} 10 & 6 & 0 & 0 & 0 & 0 & 0 \\ 1 & 20 & 5 & 0 & 0 & 0 & 0 \\ 1 & 1 & 30 & 4 & 0 & 0 & 0 \\ 0 & 3 & 1 & 40 & 3 & 0 & 0 \\ 0 & 0 & 5 & 1 & 50 & 2 & 0 \\ 0 & 0 & 0 & 7 & 1 & 60 & 1 \\ 0 & 0 & 0 & 0 & 9 & 1 & 70 \end{bmatrix}$$

están situados en una banda en torno a la diagonal principal formada por cuatro diagonales.

Observemos que, en cada fila, la matriz tiene a lo sumo un elemento positivo a la derecha de la diagonal principal, y dos a la izquierda. Esto es, su anchura de banda superior k_u es uno, su anchura de banda inferior k_l es dos y su anchura de banda (total) $k_l + 1 + k_u$ es cuatro.

Lo singular de una matriz banda A es que, debido tanto al elevado número de ceros que tiene como a la posición de dichos ceros, el costo computacional para resolver un sistema lineal de ecuaciones $Ax = b$ puede reducirse de modo significativo. Por ejemplo, se pueden eliminar todo tipo de operaciones superfluas (como sumar cero a una cantidad) en la implementación del método de eliminación de Gauss y de los métodos de sustitución regresiva y progresiva.

Hay otro aspecto crucial en el manejo de estas matrices: el desarrollo de procedimientos eficientes (en términos de recursos utilizados) para su almacenamiento en memoria. En lo que sigue abordaremos la forma en la que MATLAB® trata esta cuestión.

El formato más usual para almacenar una matriz dispersa A es el denominado formato tríada. La idea es manejar un vector a junto con dos vectores (de las mismas dimensiones que a) de números naturales, i_a y j_a , de modo que las componentes l -ésimas de i_a y j_a indiquen, respectivamente, la fila y la columna de A en la que se encuentra el elemento l -ésimo de a . Con este formato, obviamente, sólo se almacenan los elementos no nulos de A .

Ejemplo 3.2 ► La orden sparse

Consideremos la matriz

$$A = \begin{bmatrix} 12 & -4 & 0 & 0 & 0 \\ 7 & 3 & 0 & 0 & -8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -13 & 11 & 0 \\ 0 & 0 & 2 & 7 & -4 \end{bmatrix}.$$

Para almacenar A en formato disperso comenzamos escribiendo un vector con las componentes no nulas de A . El orden de colocación de las componentes de A en el vector es arbitrario:

Command Window

```
>> a=[12 7 -4 3 -13 2 11 7 -8 -4];
```

En este caso los elementos de la matriz se han escrito por columnas. A continuación, indicamos, en sendos vectores, la fila y la columna de cada elemento del vector, siguiendo el orden previamente elegido:

Command Window

```
>> fila=[1 2 1 2 4 5 4 5 2 5];
>> columna=[1 1 2 2 3 3 4 4 5 5];
```

Finalmente, generamos la matriz en formato disperso:

Command Window

```
>> A=sparse(fila,columna,a)
```

A =

```
(1,1)    12
(2,1)     7
(1,2)   -4
(2,2)     3
(4,3)  -13
(5,3)     2
(4,4)    11
(5,4)     7
(2,5)   -8
(5,5)   -4
```

De los 25 elementos de los que consta la matriz, MATLAB® únicamente almacena las 10 componentes no nulas.

Para pasar al formato usual (o lleno) se usa la orden `full`.

Ejemplo 3.3 ► Paso a formato lleno

Siguiendo con la matriz A del ejemplo 3.2, almacenada en formato disperso, su paso a formato lleno se efectúa en la forma

```
Command Window
>> A=full(A)

A =

    12    -4     0     0     0
     7     3     0     0    -8
     0     0     0     0     0
     0     0    -13    11     0
     0     0     2     7    -4
```

Asimismo, se puede pasar del formato usual al disperso también con la orden `sparse`. Es importante no olvidar que el almacenamiento en modo disperso o lleno de una matriz afecta solamente a la manera de almacenarla en memoria, pero no tiene ningún efecto sobre cualquier operación matemática que se realice con ella.

Otra orden de interés relacionada con el formato tríada es `nnz` (*Number of Non-Zeros*), que proporciona el número de elementos no nulos de la matriz. Es habitual medir la dispersión de una matriz en MATLAB® mediante su densidad

```
Command Window
density=nnz(A)/numel(A)
```

donde `numel` (*NUMber of ELeMents*) provee el número total de elementos de la matriz.

Muchas veces es útil, y recomendable, visualizar la distribución de los elementos no nulos de una matriz dispersa con la orden `spy`. Esta orden genera una representación visual de la matriz, de forma que dibuja los elementos no nulos como puntos de color azul y los nulos como puntos de color blanco.

Ejemplo 3.4 ► Matriz dispersa

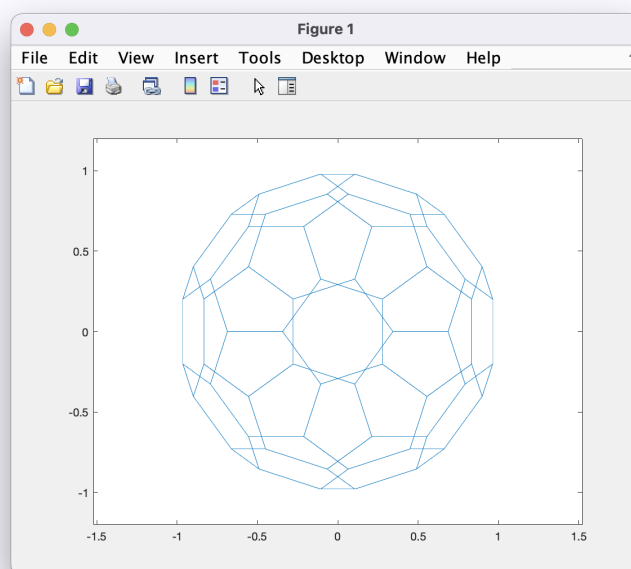
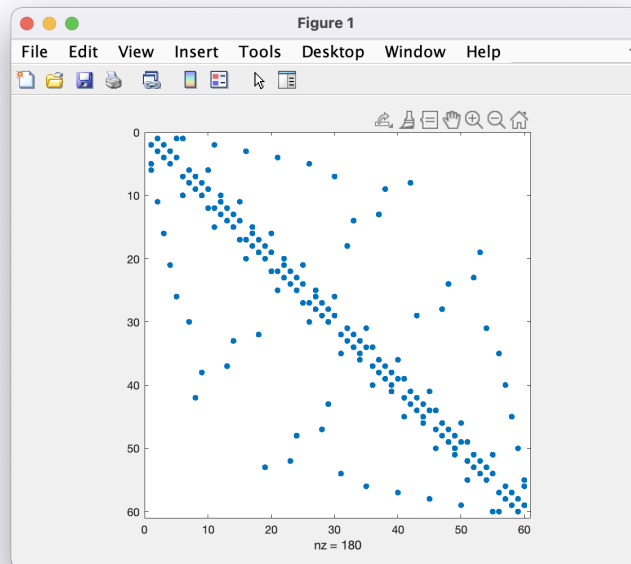
En la teoría de grafos es habitual manejar matrices bastante dispersas. En concreto, dado un cierto conjunto de n vértices unidos por aristas, la matriz A de adyacencia del grafo es una matriz cuadrada de orden n tal que a_{ij} vale uno si el vértice i -ésimo está unido con el j -ésimo y cero en caso contrario.

Los siguientes gráficos muestran la matriz de adyacencia de la molécula de *fullereno*, tercera forma molecular estable conocida del carbono, tras el grafito y el diamante, y el correspondiente grafo.

Command Window

```
>> [A,v]=bucky;  
>> spy(A)  
>> gplot(A,v),axis([-1 1 -1.2 1.2]),axis equal
```

El vector v contiene las coordenadas en el espacio tridimensional de los 60 vértices de la molécula y A es la correspondiente matriz de adyacencia entre esos vértices.



Finalmente, calculamos la densidad de la matriz A :

```
Command Window
>> density=nnz(A)/numel(A)

density =

    0.0500
```

Otra orden importante para la construcción de matrices dispersas es `spdiags` (*SParse DIAGonals*). Esta orden permite generar y almacenar en modo disperso matrices cuyos elementos no nulos se sitúan sobre las diagonales. Este tipo de matrices aparece profusamente en la resolución numérica de numerosos problemas técnicos. Para ver cómo se utiliza, posiblemente la mejor opción sea comentar con detalle un ejemplo concreto.

Ejemplo 3.5 ► La orden `spdiags`

Consideremos la matriz

$$A = \begin{bmatrix} 10 & 1 & 0 & 0 & 0 & 0 \\ -2 & 20 & 2 & 0 & 0 & 0 \\ 0 & -3 & 30 & 3 & 0 & 0 \\ 0 & 0 & -4 & 40 & 4 & 0 \\ 0 & 0 & 0 & -5 & 50 & 5 \\ 0 & 0 & 0 & 0 & -6 & 60 \end{bmatrix}.$$

Suprimiendo los elementos no nulos y colocando asteriscos en las posiciones $(1, 1)$ y $(3, 6)$, generamos, a partir de las diagonales con elementos no nulos, la matriz

$$C = \begin{bmatrix} * & 1 & 2 & 3 & 4 & 5 \\ 10 & 20 & 30 & 40 & 50 & 60 \\ -2 & -3 & -4 & -5 & -6 & * \end{bmatrix}.$$

Las columnas de la matriz C^T constituyen el primer argumento de `spdiags`. Las posiciones ocupadas por los dos asteriscos pueden rellenarse con cualquier número. El segundo argumento debe ser la posición de las diagonales que forman las columnas de C^T . En nuestro caso, se trata del vector $[-1, 0, 1]$ (a la diagonal principal se le asigna el índice 0, a la primera diagonal por encima de la principal, el índice +1 y a la primera diagonal por debajo de la principal, el índice -1; en general, a cada diagonal se le asigna el índice correspondiente a la posición que ocupa con respecto a la diagonal principal). Finalmente, los argumentos tercero y cuarto tienen que denotar, respectivamente, el número de filas y el número de columnas de la matriz. En nuestro caso, ambos números valen 6.

En otras palabras, en MATLAB® la matriz A podría generarse en modo disperso mediante el siguiente conjunto de instrucciones:

```
Command Window
>> d1=-[2:6 0]';
>> d2=10*(1:6)';
>> d3=[0 1:5]';
>> A=spdiags([d1 d2 d3],[-1 0 1],6,6);
```

Observemos que los vectores $d1, d2, d3$ son todos vectores columna y que a los asteriscos les hemos dado el valor cero (podríamos haberles dado cualquier otro valor).

Para generar matrices aleatorias dispersas, MATLAB® tiene las órdenes `sprand` y `sprandn`. Con tres argumentos, es decir,

```
Command Window
>> A=sprand(m,n,density)
```

MATLAB® genera una matriz aleatoria de orden $m \times n$ con aproximadamente $m*n*density$ entradas no nulas y con valores distribuidos uniformemente en $[0, 1]$. Asimismo, con cuatro argumentos, es decir,

```
Command Window
>> A=sprand(m,n,density,rc)
```

se genera la matriz de modo que el recíproco de su número de condición sea aproximadamente rc . El uso de `sprandn` es similar pero con la distribución normal.

Ejemplo 3.6 ► La orden `sprand`

```
Command Window
>> A=sprand(100,100,0.1,1e-6);
>> density=nnz(A)/numel(A)

density =

                0.0986

>> rc=1/conddest(A)

rc =

5.0109e-07
```

Se ha utilizado la orden `conddest`, que estima el número de condición de una matriz dispersa.

4. Apéndice B: Reordenamientos matriciales y el fenómeno de llenado

Como ya se ha comentado, la estructura de las matrices banda permite implementar la eliminación de Gauss con una notable reducción del coste computacional. Por ejemplo, para matrices tridiagonales es posible diseñar un procedimiento que requiere un número de *flop* del orden de únicamente $3n$, frente a los $\frac{2n^3}{3}$ del caso general.

Como cabría esperar, para matrices banda generales la reducción del coste computacional depende fuertemente de las anchuras de banda superior e inferior. En concreto, en una matriz banda de orden n con anchuras de banda respectivas k_u y k_l , la eliminación de Gauss sin intercambio de filas puede diseñarse con un costo computacional del orden de $ck_l(1 + k_u)n$ *flop*, siendo c cierta constante universal. Si hay intercambio de filas, el costo crece hasta $ck_l(1 + k_u + k_l)n$. Este incremento está ligado al fenómeno de llenado.

En general, el proceso de eliminación de Gauss no respeta el carácter disperso de una matriz. Es más, durante dicho proceso es frecuente que componentes nulas de la matriz pasen a ser no nulas. Este fenómeno se conoce como «llenado» (también se emplea su nombre original en inglés, *fill-in*). Más que la densidad de la matriz, lo que influye realmente en este fenómeno es la distribución de los elementos no nulos. El siguiente ejemplo muestra con claridad este aspecto.

Ejemplo 4.1 ► El fenómeno de llenado

Consideremos la matriz simétrica

$$A_n = \begin{bmatrix} 10 & \frac{1}{n}e^t \\ \frac{1}{n}e & T \end{bmatrix}, \quad n \geq 1,$$

donde e es un vector columna constituido por n unos y T es una matriz tridiagonal de orden n . La diagonal principal de T está formada por cuatros y sus dos restantes diagonales no triviales, la superdiagonal primera y la subdiagonal primera, están compuestas por el valor -1 en todas sus posiciones. Se puede probar que A_n es definida positiva para todo n .

El siguiente código de MATLAB® calcula la matriz A_{80} .

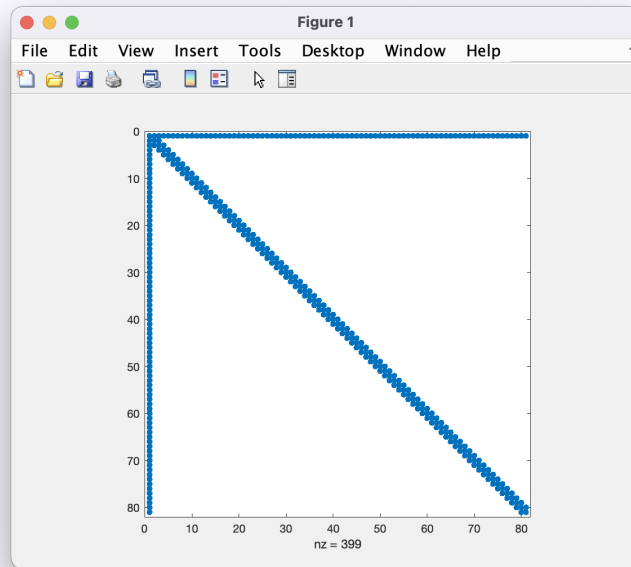
Command Window

```
>> n=80;
>> e=ones(n,1);
>> T=spdiags([-e 4*e -e],[-1 0 1],n,n);
>> A=[10 (1/n)*e';(1/n)*e T];
```

Podemos visualizar esta matriz mediante la siguiente instrucción:

Command Window

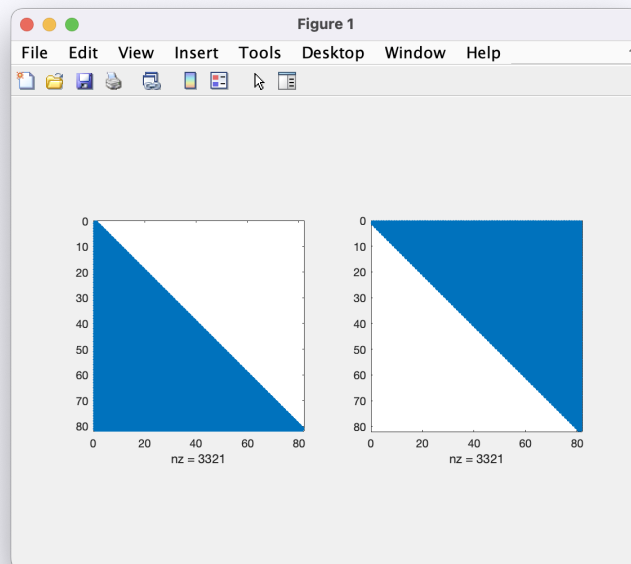
```
>> spy(A)
```



Vemos que la densidad de A es $\frac{399}{6400} \approx 0.062$. Sin embargo, al utilizar la orden lu se tiene lo siguiente:

Command Window

```
>> [L,U,P]=lu(A);
>> subplot(1,2,1),spy(L),subplot(1,2,2),spy(U)
```



Observemos que las densidades de L y U son ahora iguales a $\frac{3321}{6400} \approx 0.5189$. Es de-

cir, hemos comenzado con una matriz altamente dispersa A y al final del proceso de eliminación de Gauss se manejan matrices triangulares L y U completamente llenas.

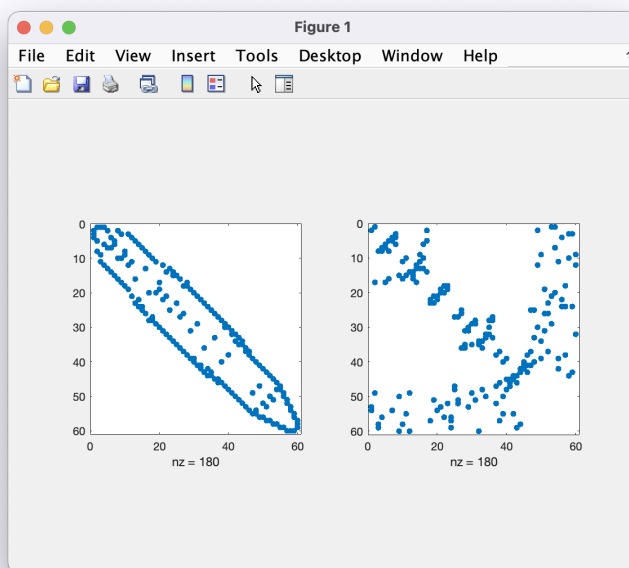
Lógicamente, el fenómeno de llenado conlleva automáticamente un aumento sustancial del tiempo de computación. Para matrices de dimensión relativamente alta, este aumento significa, muchas veces, la inviabilidad del proceso de eliminación de Gauss. Hay, por consiguiente, una pregunta que surge de modo completamente natural: ¿qué puede hacerse en situaciones como la descrita en el ejemplo anterior? La respuesta que proporciona MATLAB®, aunque sencilla, es usualmente muy eficaz: reordenar filas y/o columnas de la matriz dispersa para conseguir una distribución de los elementos no nulos más razonable, es decir, menos sensible al fenómeno de llenado.

De los distintos órdenes de MATLAB® que realizan reordenamientos matriciales, las dos más importantes son:

1. Orden `symrcm`, basada en el método *RCM* (*Reverse Cuthill-McKee Ordering*). Consiste esencialmente en reordenar las filas y las columnas de la matriz para convertirla en una matriz banda de anchura de banda lo más pequeña posible.
2. Orden `amd`, basada en el método *MDO* (*Minimum Degree Ordering*). Este reordenamiento se apoya en un algoritmo bastante sofisticado de la teoría de grafos y conduce a cierta estructura de tipo fractal con grandes bloques de ceros.

Ejemplo 4.2 ► Reordenamientos matriciales

El siguiente dibujo muestra el resultado de aplicar estas dos reordenaciones a la matriz de adyacencia de la molécula del fullereno que se vio en el ejemplo 3.4.



La imagen que aparece a la izquierda es el resultado de reordenar la matriz utilizando el método *RCM*, mientras que la situada a la derecha se ha obtenido reordenando la

matriz con el método *MDO*.

Ambas imágenes han sido obtenidas ejecutando el código:

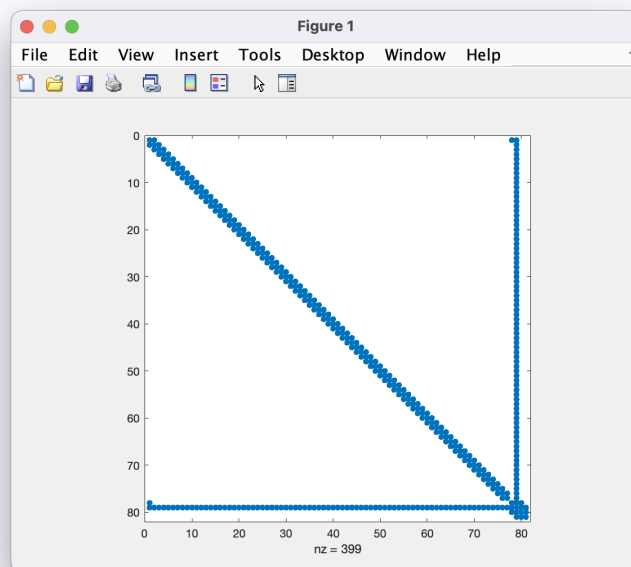
```
Command Window
>> A=bugky;
>> p1=symrcm(A);
>> A1=A(p1,p1);
>> p2=amd(A);
>> A2=A(p2,p2);
>> subplot(1,2,1),spy(A1),subplot(1,2,2),spy(A2)
```

Veamos cómo afecta la reordenación *RCM* a la matriz A de orden $n = 80$ del ejemplo 4.1.

Al ejecutar el código

```
Command Window
>> p=symrcm(A);
>> B=A(p,p);
>> spy(B)
```

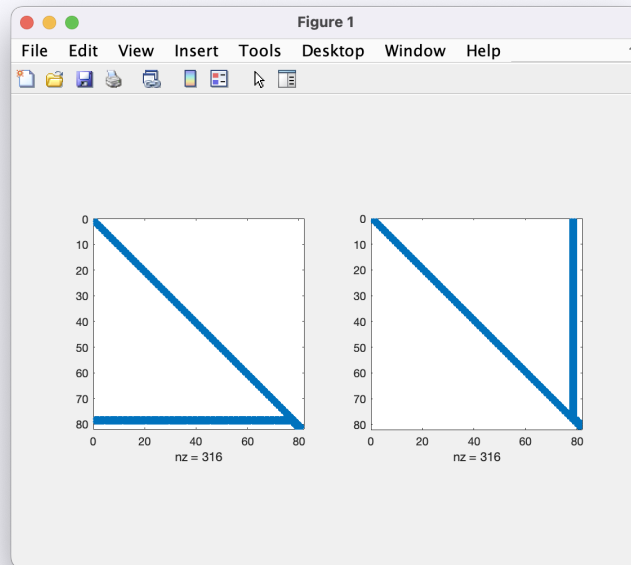
obtenemos el gráfico de la matriz B reordenada,



Como era de esperar, B tiene el mismo número de elementos no nulos que A . Sin embargo, al aplicar la orden lu, se obtienen matrices L y U que no han sufrido el fenómeno de llenado. En efecto,

```
Command Window
```

```
>> [L,U,P]=lu(B);  
>> subplot(1,2,1),spy(L),subplot(1,2,2),spy(U)
```



Conviene subrayar que, siempre que la matriz del sistema esté almacenada en formato disperso, el operador «barra inversa» de MATLAB® realiza primero un reordenamiento matricial de la matriz y, después, efectúa el proceso de eliminación de Gauss con pivoteo parcial adaptado al contexto disperso.

5. Apéndice C: El método de Householder

MATLAB® dispone de la orden `qr` para calcular factorizaciones QR completas y reducidas de matrices. Esta orden implementa el **método de Householder**, basado en transformaciones asociadas a matrices ortogonales y simétricas que generalizan, formalmente, las simetrías del plano a un espacio de dimensión n .

La factorización QR está ligada algebraicamente (en cierto sentido es equivalente) al proceso de ortogonalización de Gram-Schmidt. Por consiguiente, es posible usar esta estrategia para programar su cálculo. No obstante, esta opción suele descartarse, puesto que la programación, llamémosla «simple» o «directa», del proceso es inestable numéricamente y la programación, denominémosla «sofisticada» (y ya estable numéricamente), suele ser, en general, menos (bastante menos) eficiente que el método de Householder.

Observemos que, dado un punto P del plano, siempre podemos elegir una recta que pase por el origen de coordenadas de modo que la simetría asociada envíe P al eje OX . Es más, recordemos que la matriz S de la simetría cuyo eje es $ax + by = 0$ viene dada por

$$S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{2}{a^2 + b^2} vv^T,$$

donde $v = [a, b]^T$ (notemos que $\|v\|_2 = \sqrt{a^2 + b^2}$).

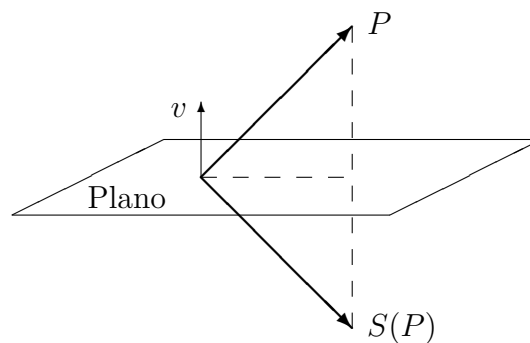
Ejemplo 5.1 ► Simetría en el plano

La simetría asociada a la bisectriz de los cuadrantes primero y tercero (cuya ecuación es $x - y = 0$) viene dada por

$$S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{2}{2} \begin{bmatrix} 1 \\ -1 \end{bmatrix} [1, -1] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Por ejemplo, el punto $[0, 1]^T$ se transforma en $[1, 0]^T$ mediante esta simetría.

En el espacio \mathbb{R}^3 , obtendríamos algo similar considerando simetrías con respecto a planos. Gráficamente,



En general, dado un hiperplano V de \mathbb{R}^m (cuya dimensión es $m - 1$) de ecuación implícita $v^T x = 0$ (v es un vector columna no nulo de \mathbb{R}^m), se denomina simetría especular respecto

a V a la matriz S_V de orden m definida por

$$S_V := I_m - \frac{2}{v^\top v} vv^\top,$$

donde I_m es la matriz identidad de orden m . Puede comprobarse fácilmente (como en el caso del plano) que S_V es siempre una matriz ortogonal y simétrica.

De modo completamente análogo al caso de \mathbb{R}^2 y de \mathbb{R}^3 , dado un vector $w \in \mathbb{R}^m$ siempre es posible elegir un hiperplano V de modo que la simetría asociada S_V envíe dicho punto al eje OX . Es decir,

$$S_V w = \begin{bmatrix} \pm \|w\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Recordemos que al ser S ortogonal se verifica que

$$\|S_V w\|_2 = \sqrt{w^\top S_V^\top S_V w} = \sqrt{w^\top w} = \|w\|_2.$$

En concreto, dado un vector $x \in \mathbb{R}^m$ no nulo, con $m \geq 2$, se define la matriz H de Householder asociada a x como

$$H := I - \frac{2}{v^\top v} vv^\top,$$

donde

$$v = x + \delta \|x\|_2 e_1,$$

siendo $e_1 = [1, 0, \dots, 0]^\top$ y

$$\delta = \begin{cases} \text{sign}(x_1), & x_1 \neq 0, \\ 1, & x_1 = 0. \end{cases}$$

Puede comprobarse que

$$Hx = \begin{bmatrix} -\delta \|x\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

La elección del signo de δ se hace para evitar posibles fenómenos de cancelación numérica al calcular la primera coordenada de v .

Volviendo a la factorización QR , dada una matriz A de orden $m \times n$ y eligiendo cuidadosamente n transformaciones de Householder, es posible generar un total de n matrices ortogonales y simétricas H_k , $k = 1, 2, \dots, n$, de orden m de modo que

$$R := H_n H_{n-1} \cdots H_2 H_1 A$$

sea una matriz trapezoidal superior de orden $m \times n$. Finalmente, si definimos

$$Q := H_1 H_2 \cdots H_{n-1} H_n$$

llegamos a que $A = QR$, donde Q es una matriz ortogonal de orden m .

6. Soluciones de los proyectos

Proyecto 1.1

Vamos a describir de forma esquematizada, pero completa cómo se aplica el método de eliminación de Gauss con pivoteo parcial al sistema lineal de ecuaciones dado en el enunciado, $Ax = b$, que conducirá a la factorización $PA = LU$ de la matriz A . Mediante esta factorización se calculará la solución del sistema.

Etapa 0 Sistema inicial

$$Ax = b \quad \equiv \quad \begin{bmatrix} 1 & -7 & 1 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 7 \\ 16 \end{bmatrix}$$

Etapa 1.1.1 Intercambio de filas

$F_1 \longleftrightarrow F_3$ (primer intercambio de filas)

$$P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$P_1Ax = P_1b \quad \equiv \quad \begin{bmatrix} 5 & -1 & 5 \\ -3 & 2 & 6 \\ 1 & -7 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 16 \\ 7 \\ 10 \end{bmatrix}$$

Etapa 1.1.2 Reducción a cero bajo el pivote

Multiplicadores: $l_{21} = -\frac{3}{5}$, $l_{31} = \frac{1}{5}$ (primera reducción)

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ \frac{3}{5} & 1 & 0 \\ -\frac{1}{5} & 0 & 1 \end{bmatrix}$$

$$M_1P_1Ax = M_1P_1b \quad \equiv \quad \begin{bmatrix} 5 & -1 & 5 \\ 0 & \frac{7}{5} & 9 \\ 0 & -\frac{34}{5} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 16 \\ \frac{83}{5} \\ \frac{34}{5} \end{bmatrix}$$

Etapa 1.2.1 Intercambio de filas

$F_2 \longleftrightarrow F_3$ (segundo intercambio de filas)

$$P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$P_2 M_1 P_1 A x = P_2 M_1 P_1 b \quad \equiv \quad \begin{bmatrix} 5 & -1 & 5 \\ 0 & -\frac{34}{5} & 0 \\ 0 & \frac{7}{5} & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 16 \\ \frac{34}{5} \\ \frac{83}{5} \end{bmatrix}$$

Etapa 1.2.2 Reducción a cero bajo el pivote

Multiplicador: $l_{32} = \frac{7/5}{-34/5} = -\frac{7}{34}$ (segunda reducción)

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{7}{34} & 1 \end{bmatrix}$$

$$M_2 P_2 M_1 P_1 A x = M_2 P_2 M_1 P_1 b \quad \equiv \quad \begin{bmatrix} 5 & -1 & 5 \\ 0 & -\frac{34}{5} & 0 \\ 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 16 \\ \frac{34}{5} \\ 18 \end{bmatrix}$$

Etapa 2 Sistema reducido

El método de eliminación de Gauss con pivoteo parcial ha finalizado. El sistema reducido es

$$Ux = b' \quad \equiv \quad \begin{bmatrix} 5 & -1 & 5 \\ 0 & -\frac{34}{5} & 0 \\ 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 16 \\ \frac{34}{5} \\ 18 \end{bmatrix}$$

donde

$$U = M_2 P_2 M_1 P_1 A = \begin{bmatrix} 5 & -1 & 5 \\ 0 & -\frac{34}{5} & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

y

$$b' = M_2 P_2 M_1 P_1 b = \begin{bmatrix} 16 \\ \frac{34}{5} \\ 18 \end{bmatrix}$$

Etapas 3 Factorización $PA = LU$

A continuación, operamos con las matrices que definen U .

$$M_2 P_2 M_1 P_1 A = U$$

$$P_2 M_1 P_1 A = M_2^{-1} U$$

$$\underbrace{P_2 M_1 P_2^{-1}}_{\widetilde{M}_1} P_2 P_1 A = M_2^{-1} U$$

$$P_2 P_1 A = \widetilde{M}_1^{-1} M_2^{-1} U$$

$$\underbrace{P_2 P_1}_P A = \underbrace{\widetilde{M}_1^{-1} M_2^{-1}}_L U$$

$$PA = LU$$

donde

$$\widetilde{M}_1 = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{5} & 1 & 0 \\ \frac{3}{5} & 0 & 1 \end{bmatrix} \quad \widetilde{M}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{5} & 1 & 0 \\ -\frac{3}{5} & 0 & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{7}{34} & 1 \end{bmatrix} \quad M_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{7}{34} & 1 \end{bmatrix}$$

$$P = P_2 P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad L = \widetilde{M}_1^{-1} M_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{5} & 1 & 0 \\ -\frac{3}{5} & -\frac{7}{34} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} 5 & -1 & 5 \\ 0 & -\frac{34}{5} & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

Hemos completado el cálculo de las matrices L , P y U de la factorización $PA = LU$ de la matriz A .

Etapa 4 Resolución del sistema

Finalmente, aplicamos la factorización $PA = LU$ a la resolución del sistema $Ax = b$.

$$Ax = b$$

$$\underbrace{PA}_{LU} x = Pb$$

$$L \underbrace{Ux}_y = Pb$$

$$Ly = Pb \longrightarrow \text{método de sustitución progresiva} \longrightarrow y$$

$$Ux = y \longrightarrow \text{método de sustitución regresiva} \longrightarrow x$$

Vamos a resolver los dos sistemas de ecuaciones lineales $Ly = Pb$ y $Ux = y$.

$$Ly = Pb \equiv \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{5} & 1 & 0 \\ -\frac{3}{5} & -\frac{7}{34} & 1 \end{bmatrix} y = \begin{bmatrix} 16 \\ 10 \\ 7 \end{bmatrix} \longrightarrow y = \begin{bmatrix} 16 \\ \frac{34}{5} \\ 18 \end{bmatrix}$$

$$Ux = y \equiv \begin{bmatrix} 5 & -1 & 5 \\ 0 & -\frac{34}{5} & 0 \\ 0 & 0 & 9 \end{bmatrix} x = \begin{bmatrix} 16 \\ \frac{34}{5} \\ 18 \end{bmatrix} \longrightarrow x = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

La solución del sistema lineal de ecuaciones es

$$x = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

Proyecto 1.2

1. Por definición, $\text{cond}(A) = \|A^{-1}\| \|A\|$.

Aplicando propiedades de las normas de matriz, se tiene que

$$\|A^{-1}\| \|A\| \geq \|A^{-1}A\| = \|I\| = 1,$$

donde I es la matriz identidad del mismo orden que A .

Así,

$$\text{cond}(A) \geq 1$$

2. Se tiene que

$$\text{cond}_2(AQ) = \|(AQ)^{-1}\|_2 \|AQ\|_2 = \sqrt{\mu_n} \sqrt{\lambda_n},$$

donde

- μ_n es el máximo autovalor de $((AQ)^{-1})^\top (AQ)^{-1}$.
- λ_n es el máximo autovalor de $(AQ)^\top AQ$.

Ahora bien,

- $((AQ)^{-1})^\top (AQ)^{-1} = (A^{-1})^\top (Q^{-1})^\top Q^{-1} A^{-1} = (A^{-1})^\top A^{-1}$, al ser Q ortogonal.
- $(AQ)^\top AQ = Q^\top A^\top AQ$. Pero esta matriz tiene los mismos autovalores que $A^\top A$, puesto que $Q^\top A^\top AQ$ y $A^\top A$ son matrices semejantes.

Por lo tanto,

- μ_n es el máximo autovalor de $(A^{-1})^\top A^{-1}$, luego $\|A^{-1}\|_2 = \sqrt{\mu_n}$.
- λ_n es el máximo autovalor de $A^\top A$, luego $\|A\|_2 = \sqrt{\lambda_n}$.

Se sigue que

$$\text{cond}_2(AQ) = \sqrt{\mu_n} \sqrt{\lambda_n} = \|A^{-1}\|_2 \|A\|_2 = \text{cond}_2(A).$$

Así,

$$\text{cond}_2(AQ) = \text{cond}_2(A)$$

De forma análoga se demostraría la otra igualdad:

$$\text{cond}_2(QA) = \text{cond}_2(A)$$

3. Aplicando el resultado del punto anterior a la matriz $A = I$, donde I es la matriz identidad, se tiene que

$$\text{cond}_2(Q) = \text{cond}_2(IQ) = \text{cond}_2(I) = 1.$$

Así,

$$\text{cond}_2(Q) = 1$$

Proyecto 1.3

El siguiente código de MATLAB® realiza las operaciones solicitadas. Hemos añadido un diagrama de barras con la distribución de las matrices por número de condición. Por ejemplo, la etiqueta 2 en el eje horizontal del diagrama de barras hace referencia a las matrices cuyo número de condición se encuentra en el intervalo $[10^2, 10^3)$.

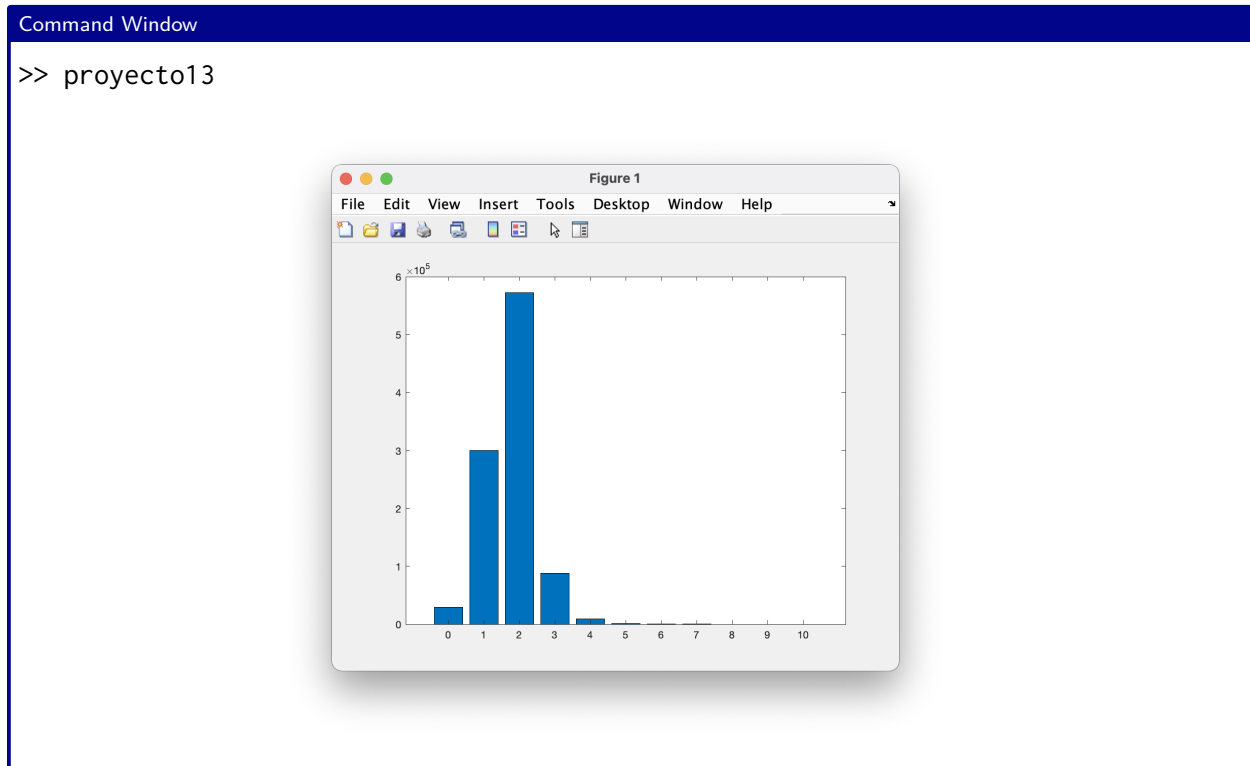
```

Editor ► Matrices aleatorias y números de condición      proyecto13.m

format short
n=zeros(11,1);
Q=10^6;
for i=1:Q
    A=randn(1+randi(99,1));
    c=cond(A);
    if 1 <= c && c < 10
        n(1)=n(1)+1;
    elseif 10 <= c && c < 10^2
        n(2)=n(2)+1;
    elseif 10^2 <= c && c < 10^3
        n(3)=n(3)+1;
    elseif 10^3 <= c && c < 10^4
        n(4)=n(4)+1;
    elseif 10^4 <= c && c < 10^5
        n(5)=n(5)+1;
    elseif 10^5 <= c && c < 10^6
        n(6)=n(6)+1;
    elseif 10^6 <= c && c < 10^7
        n(7)=n(7)+1;
    elseif 10^7 <= c && c < 10^8
        n(8)=n(8)+1;
    elseif 10^8 <= c && c < 10^9
        n(9)=n(9)+1;
    elseif 10^9 <= c && c < 10^10
        n(10)=n(10)+1;
    elseif c >= 10^10
        n(11)=n(11)+1;
    end
end
disp('Distribución de matrices por números de condición')
disp(n)
disp(' ')
disp('Distribución de matrices por tantos por ciento')
disp(100*n/Q)
bar(0:10,n),shg

```

Los resultados de la ejecución en MATLAB® de este código aparecen estructurados en la tabla mostrada más abajo.

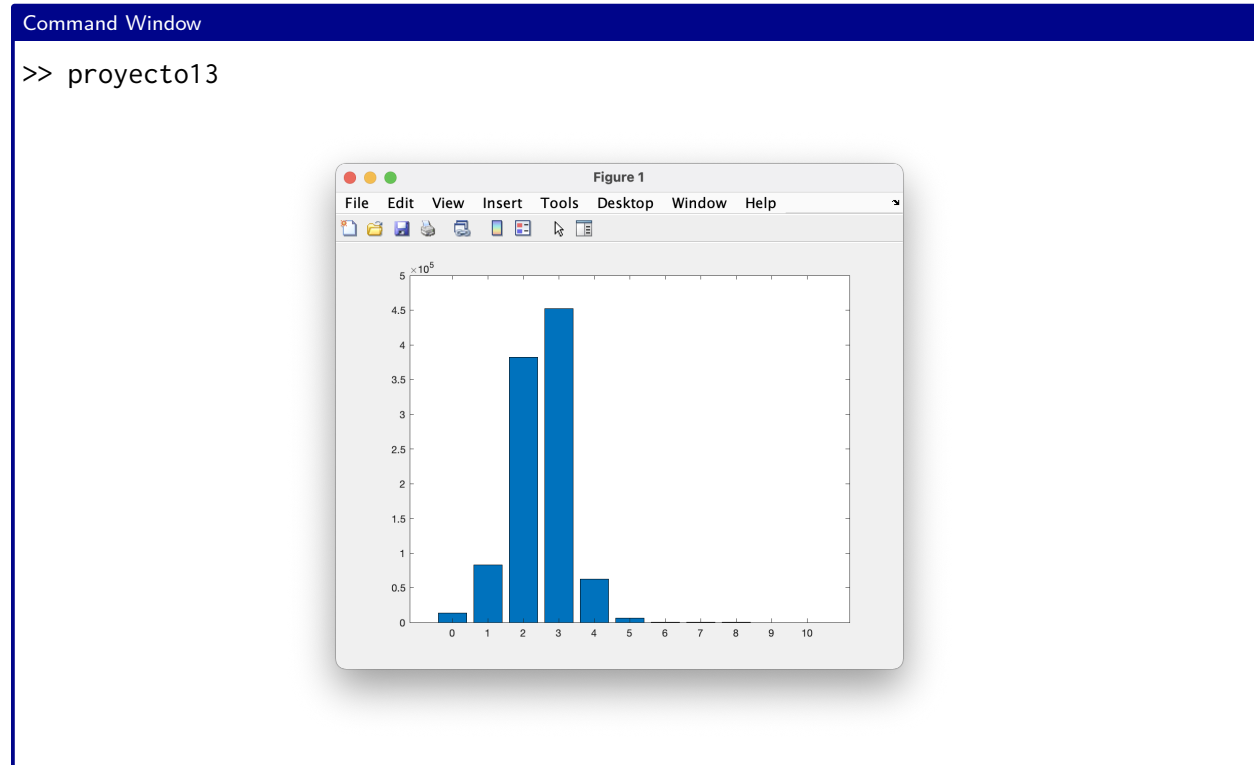


Tamaño del número de condición c	Número de matrices	% del total	% acumulado
$1 \leq c < 10$	29565	2.9565	2.9565
$10 \leq c < 10^2$	299868	29.9868	32.9433
$10^2 \leq c < 10^3$	572612	57.2612	90.2045
$10^3 \leq c < 10^4$	88180	8.8180	99.0225
$10^4 \leq c < 10^5$	8774	0.8774	99.8999
$10^5 \leq c < 10^6$	927	0.0927	99.9926
$10^6 \leq c < 10^7$	69	0.0069	99.9995
$10^7 \leq c < 10^8$	4	0.0004	99.9999
$10^8 \leq c < 10^9$	1	0.0001	100
$10^9 \leq c < 10^{10}$	0	0	
$c \geq 10^{10}$	0	0	

Fíjate en que más del 99% de las matrices tienen números de condición menores que 10^4 . Las matrices con números de condición mayores que 10^4 son residuales.

Hemos utilizado matrices aleatorias (y órdenes aleatorios), pero los resultados son consistentes. Si ejecutas el código anterior repetidas veces, verás que siempre obtienes números de la misma magnitud. Como muestra de esto, el diagrama de barras y la tabla mostrados corresponden a diferentes ejecuciones del código.

Si generas las matrices aleatorias mediante la orden `rand` en lugar de `randn`, comprobarás cierto crecimiento en los números de condición. Aún así, más del 90 % de las matrices tienen números de condición menores que 10^4 . Mostramos a continuación el diagrama de barras y la tabla con los resultados obtenidos utilizando `rand`.



Tamaño del número de condición c	Número de matrices	% del total	% acumulado
$1 \leq c < 10$	13340	1.3340	1.3340
$10 \leq c < 10^2$	83101	8.3101	9.6441
$10^2 \leq c < 10^3$	381843	38.1843	47.8284
$10^3 \leq c < 10^4$	452288	45.2288	93.0572
$10^4 \leq c < 10^5$	62565	6.2565	99.3137
$10^5 \leq c < 10^6$	6202	0.6202	99.9339
$10^6 \leq c < 10^7$	584	0.0584	99.9923
$10^7 \leq c < 10^8$	62	0.0062	99.9985
$10^8 \leq c < 10^9$	15	0.0015	100
$10^9 \leq c < 10^{10}$	0	0	
$c \geq 10^{10}$	0	0	