

Métodos directos para sistemas lineales

Matemáticas de la Especialidad
Manuel Colera Rico
Enero de 2026

1. Introducción

En el mundo de la ingeniería, es frecuente tener que resolver sistemas lineales de la forma

$$\mathbf{Ax} = \mathbf{b}, \quad (1)$$

donde $\mathbf{A} \in \mathbb{R}^{N \times N}$ es una matriz cuadrada no singular conocida, $\mathbf{b} \in \mathbb{R}^N$ es un vector conocido y $\mathbf{x} \in \mathbb{R}^N$ es un vector incógnita. A modo de ilustración, pueden consultarse los enunciados de los problemas 6.3 y 6.8, aunque aparecerán más ejemplos en temas posteriores.

La solución \mathbf{x} a la ecuación (1) se denota por

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}. \quad (2)$$

En muchas ocasiones, la expresión (2) se interpreta erróneamente como “ \mathbf{x} es igual a la *matriz inversa* de \mathbf{A} multiplicada por \mathbf{b} ”. Aunque desde un punto de vista teórico esto es cierto, en la práctica (salvo casos muy excepcionales), calcular la matriz inversa es muy costoso y numéricamente poco preciso. En su lugar, hay que interpretar \mathbf{A}^{-1} como un *operador* que recibe un vector \mathbf{b} y devuelve la solución \mathbf{x} del sistema $\mathbf{Ax} = \mathbf{b}$. O, simplemente, interpretar (2) como “ \mathbf{x} es la solución de $\mathbf{Ax} = \mathbf{b}$ ”.

El algoritmo más apropiado para resolver (1) depende de las propiedades de la matriz \mathbf{A} . En este tema se estudiarán *métodos directos*, es decir, aquéllos que son capaces de hallar la solución exacta (ignorando errores de redondeo) en un número finito de pasos. Existen también *métodos iterativos*, es decir, métodos que generan secuencias de aproximaciones $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots$ cada vez más próximas a la solución exacta.

En Matlab, la forma más sencilla de resolver (1) mediante un método directo es con el comando barra invertida (`\`), es decir, mediante la orden

$$\mathbf{x} = \mathbf{A} \backslash \mathbf{b};$$

Cuando se ejecuta la barra invertida, Matlab analiza primero qué propiedades tiene la matriz \mathbf{A} y elige el algoritmo más apropiado. Aunque esto es suficiente en algunas ocasiones, si se desean resolver muchos sistemas caracterizados por la misma matriz \mathbf{A} , es conveniente conocer el algoritmo utilizado con el fin de reutilizar algunas operaciones intermedias y reducir drásticamente el coste computacional.

En lo que sigue, se utilizará la siguiente notación. Dada una matriz cualquiera $\mathbf{B} \in \mathbb{R}^{M \times N}$:

- $B_{i:}$ representa la fila i , i.e., $B_{i:} = [B_{i1}, \dots, B_{iN}]$,
- $B_{i,a:b}$ representa la subfila i comprendida entre las columnas a y b , i.e., $B_{i,a:b} = [B_{ia}, \dots, B_{ib}]$,
- $B_{:,i}$ representa la columna i , i.e., $B_{:,i} = [B_{1i}, \dots, B_{Mi}]^T$,
- $B_{a:b,i}$ representa la subcolumna i comprendida entre las filas a y b , i.e., $B_{a:b,i} = [B_{ai}, \dots, B_{bi}]^T$.

2. Factorización LU con pivotaje parcial

En el caso más general, es decir, si A no es simétrica y no tiene ninguna propiedad especial (por ejemplo, muchos elementos nulos), el método más apropiado para resolver (1) es la *factorización LU con pivotaje parcial*. La idea es descomponer A de la forma

$$PA = LU, \quad (3)$$

donde $P \in \mathbb{R}^{N \times N}$ es una *matriz de permutación* (es decir, PA tiene las mismas filas que A pero ordenadas de distinta forma), $L \in \mathbb{R}^{N \times N}$ es una matriz *triangular inferior*, y $U \in \mathbb{R}^{N \times N}$ es una matriz *triangular superior*.

Es fácil demostrar que, en términos de (3), el sistema original (1) se puede reescribir como $LUx = Pb$ y, por tanto, x se puede obtener resolviendo sucesivamente los sistemas

$$Ly = Pb, \quad (4)$$

$$Ux = y. \quad (5)$$

A diferencia del sistema original (1), los sistemas (4)-(5) son muy fáciles de resolver porque las matrices que los definen son triangulares. Véanse los ejercicios 2-3 al respecto.

Los factores P , L y U se calculan mediante un proceso de eliminación gaussiana con pivotaje de filas. En particular, supongamos que existen una matriz de permutación P^i , una matriz triangular inferior L^i y una matriz (no necesariamente triangular superior) U^i , de las mismas dimensiones que A y de la forma

$$L^i = \left[\begin{array}{cccc|c} 1 & & & & \\ * & 1 & & & \\ * & * & \ddots & & \\ * & * & \ddots & 1 & \\ \hline L_{i1}^i & L_{i2}^i & \dots & L_{i,i-1}^i & 1 \\ \vdots & \vdots & & \vdots & \ddots \\ L_{m1}^i & L_{m2}^i & \dots & L_{m,i-1}^i & 1 \\ \vdots & \vdots & & \vdots & \ddots \\ * & * & \dots & * & 1 \end{array} \right], \quad U^i = \left[\begin{array}{cccc|cccc} * & * & * & * & * & \dots & \dots & * \\ & * & * & * & * & \dots & \dots & * \\ & & \ddots & \vdots & \vdots & & & \vdots \\ & & & * & * & \dots & \dots & * \\ \hline & & & & U_{ii}^i & \dots & \dots & U_{iN}^i \\ & & & & \vdots & & & \vdots \\ & & & & U_{mi}^i & \dots & \dots & U_{mN}^i \\ & & & & \vdots & & & \vdots \\ & & & & U_{Ni}^i & \dots & \dots & U_{NN}^i \end{array} \right], \quad (6)$$

tales que

$$P^i A = L^i U^i. \quad (7)$$

En la expresión (6), las líneas continuas separan la fila/columna $i-1$ de la i , y los asteriscos denotan elementos no nulos cuyo valor exacto no es relevante para el razonamiento que sigue.

2.1. Primer paso: pivotaje de filas

En primer lugar, para garantizar una buena estabilidad numérica, de entre los candidatos $U_{i1}^i, \dots, U_{Ni}^i$, se selecciona aquél de mayor valor absoluto. (Si hay varios que cumplen esa condición, se toma por convenio el que aparece en primer lugar en la lista anterior.) Supongamos que dicho candidato es el asociado a la *fila m* de la matriz U^i , es decir,

$$m = \min_{j=i, \dots, N} j \quad \text{tal que} \quad |U_{ji}^i| \geq |U_{ki}^i|, \quad k = i, \dots, N.$$

A continuación, se ha de permutar la **fila i** de \mathbb{U} con la **fila m** . Para ello, definimos una matriz de permutación $\Pi^i \in \mathbb{R}^{N \times N}$ tal que $\Pi^i \mathbb{B}$ es el resultado de intercambiar las filas i y m en una matriz cualquiera $\mathbb{B} \in \mathbb{R}^{N \times N}$, y reescribimos (7) como

$$\begin{aligned}\Pi^i \mathbb{P}^i \mathbb{A} &= \Pi^i \mathbb{L}^i (\Pi^i)^T \Pi^i \mathbb{U}^i, \\ \mathbb{P}^{i+1} \mathbb{A} &= \tilde{\mathbb{L}}^i \tilde{\mathbb{U}}^i,\end{aligned}\tag{8}$$

donde se ha hecho uso de que cualquier matriz de permutación es ortonormal, es decir, $(\Pi^i)^T \Pi^i = \mathbb{I}$. Notemos que

- La matriz $\mathbb{P}^{i+1} := \Pi^i \mathbb{P}^i$ es una nueva matriz de permutación que se obtiene al permutar las filas **i** y **m** de \mathbb{P}^i .
- La matriz $\tilde{\mathbb{L}}^i := \Pi^i \mathbb{L}^i (\Pi^i)^T$, que resulta de intercambiar filas y columnas **i** y **m** en \mathbb{L}^i , mantiene la misma estructura que \mathbb{L}^i , aunque intercambiando las subfilas **i** y **m** debajo de la diagonal, es decir,

$$\tilde{\mathbb{L}}^i = \left[\begin{array}{cccc|cccc} 1 & & & & & & & \\ * & 1 & & & & & & \\ * & * & \ddots & & & & & \\ * & * & \ddots & 1 & & & & \\ \hline L_{m1}^i & L_{m2}^i & \cdots & L_{m,i-1}^i & 1 & & & \\ \vdots & \vdots & & \vdots & & \ddots & & \\ L_{i1}^i & L_{i2}^i & \cdots & L_{i,i-1}^i & & & 1 & \\ \vdots & \vdots & & \vdots & & & & \ddots \\ * & * & \cdots & * & & & & 1 \end{array} \right] = \left[\begin{array}{cccc|cccc} 1 & & & & & & & \\ * & 1 & & & & & & \\ * & * & \ddots & & & & & \\ * & * & \ddots & 1 & & & & \\ \hline \tilde{L}_{i1}^i & \tilde{L}_{i2}^i & \cdots & \tilde{L}_{i,i-1}^i & 1 & & & \\ \vdots & \vdots & & \vdots & & \ddots & & \\ \tilde{L}_{m1}^i & \tilde{L}_{m2}^i & \cdots & \tilde{L}_{m,i-1}^i & & & 1 & \\ \vdots & \vdots & & \vdots & & & & \ddots \\ * & * & \cdots & * & & & & 1 \end{array} \right].$$

- La matriz $\tilde{\mathbb{U}}^i := \Pi^i \mathbb{U}^i$, que resulta de intercambiar las filas **i** y **m** en \mathbb{U}^i , mantiene la misma estructura que \mathbb{U}^i , es decir,

$$\tilde{\mathbb{U}}^i = \left[\begin{array}{cccc|cccc} * & * & * & * & * & \cdots & \cdots & * \\ & * & * & * & * & \cdots & \cdots & * \\ & & \ddots & \vdots & \vdots & & & \vdots \\ & & & * & * & \cdots & \cdots & * \\ \hline & & & & U_{mi}^i & \cdots & \cdots & U_{mN}^i \\ \vdots & & & & \vdots & & & \vdots \\ U_{ii}^i & \cdots & \cdots & U_{iN}^i & & & & \\ \vdots & & & \vdots & & & & \vdots \\ U_{Ni}^i & \cdots & \cdots & U_{NN}^i & & & & \end{array} \right] = \left[\begin{array}{cccc|cccc} * & * & * & * & * & \cdots & \cdots & * \\ & * & * & * & * & \cdots & \cdots & * \\ & & \ddots & \vdots & \vdots & & & \vdots \\ & & & * & * & \cdots & \cdots & * \\ \hline & & & & \tilde{U}_{ii}^i & \cdots & \cdots & \tilde{U}_{iN}^i \\ \vdots & & & & \vdots & & & \vdots \\ \tilde{U}_{mi}^i & \cdots & \cdots & \tilde{U}_{mN}^i & & & & \\ \vdots & & & \vdots & & & & \vdots \\ \tilde{U}_{Ni}^i & \cdots & \cdots & \tilde{U}_{NN}^i & & & & \end{array} \right].$$

- No es necesario calcular explícitamente la matriz Π^i para hallar las matrices anteriores.

2.2. Segundo paso: eliminación gaussiana

En segundo lugar, se realiza un proceso de eliminación gaussiana para conseguir ceros debajo del (nuevo) pivote \tilde{U}_{ii}^i . En particular, definimos una nueva matriz \mathbb{U}^{i+1} tal que

- (i) las filas $k = 1, \dots, i$ de \mathbb{U}^{i+1} coinciden con las de $\tilde{\mathbb{U}}^i$, es decir,

$$U_{k,:}^{i+1} = \tilde{U}_{k,:}^i,\tag{9}$$

(ii) las filas $k = i + 1, \dots, N$ se definen de acuerdo a

$$L_{ki}^i := \frac{\tilde{U}_{ki}^i}{\tilde{U}_{ii}^i}, \quad U_{k,:}^{i+1} = \tilde{U}_{k,:}^i - L_{ki}^i \tilde{U}_{i,:}^i. \quad (10)$$

Notemos que la estrategia de pivotaje empleada anteriormente garantiza que $\tilde{U}_{ii}^i \neq 0$ y que, por tanto, exista L_{ki}^i . La nueva matriz \mathbb{U}^{i+1} será de la forma

$$\mathbb{U}^{i+1} = \left[\begin{array}{cccc|cccc} * & * & * & * & * & \dots & \dots & * \\ & * & * & * & * & \dots & \dots & * \\ & & \ddots & \vdots & \vdots & & & \vdots \\ & & & * & * & \dots & \dots & * \\ \hline & & & \tilde{U}_{ii}^i & \tilde{U}_{i,i+1}^i & \dots & \dots & \tilde{U}_{iN}^i \\ & & & 0 & U_{i+1,i+1}^{i+1} & \dots & \dots & U_{i+1,N}^{i+1} \\ & & & \vdots & \vdots & \ddots & & \vdots \\ & & & \vdots & \vdots & & \ddots & \vdots \\ & & & 0 & U_{N,i+1}^{i+1} & \dots & \dots & U_{NN}^{i+1} \end{array} \right].$$

A partir de (9)-(10), es fácil ver que la matriz $\tilde{\mathbb{U}}^i$ puede obtenerse a partir de la nueva matriz \mathbb{U}^{i+1} de la siguiente forma:

- (i) $\tilde{U}_{k,:}^i = U_{k,:}^{i+1}$ para $k = 1, \dots, i$,
- (ii) $\tilde{U}_{k,:}^i = U_{k,:}^{i+1} + L_{ki}^i U_{i,:}^{i+1}$ para $k = i + 1, \dots, N$.

Matricialmente, esto se escribe como

$$\tilde{\mathbb{U}}^i = \Lambda^i \mathbb{U}^{i+1}, \quad \Lambda^i := \left[\begin{array}{cccc|cccc} 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & & \\ & & & 1 & & & & \\ \hline & & & & 1 & & & \\ & & & & L_{i+1,i}^i & 1 & & \\ & & & & \vdots & & \ddots & \\ & & & & \vdots & & & 1 \\ & & & & L_{Ni}^i & & & 1 \end{array} \right]. \quad (11)$$

Sustituyendo (11) en (8), se tiene

$$\mathbb{P}^{i+1} \mathbf{A} = \tilde{\mathbf{L}}^i \tilde{\mathbb{U}}^i = \tilde{\mathbf{L}}^i \Lambda^i \mathbb{U}^{i+1} = \mathbf{L}^{i+1} \mathbb{U}^{i+1},$$

donde $\mathbb{L}^{i+1} := \tilde{L}^i \Lambda^i$. Es fácil comprobar que \mathbb{L}^{i+1} se obtiene añadiendo a la matriz $\tilde{\mathbb{L}}^i$ los términos $L_{i+1,i}^i, \dots, L_{N,i}^i$ debajo del i -ésimo término diagonal, es decir,

$$\mathbb{L}^{i+1} = \left[\begin{array}{cccc|cccc} 1 & & & & & & & \\ * & 1 & & & & & & \\ * & * & \ddots & & & & & \\ * & * & \ddots & 1 & & & & \\ \hline \tilde{L}_{i1}^i & \tilde{L}_{i2}^i & \dots & \tilde{L}_{i,i-1}^i & 1 & & & \\ \vdots & \vdots & & \vdots & L_{i+1,i}^i & 1 & & \\ \tilde{L}_{m1}^i & \tilde{L}_{m2}^i & \dots & \tilde{L}_{m,i-1}^i & \vdots & & \ddots & \\ \vdots & \vdots & & \vdots & \vdots & & & 1 \\ * & * & \dots & * & L_{Ni}^i & & & 1 \end{array} \right].$$

2.3. Resumen del algoritmo

Así pues, de acuerdo a las secciones anteriores, si partimos de una matriz de permutación \mathbb{P}^i y de unas matrices \mathbb{L}^i y \mathbb{U}^i de la forma (6) tales que se cumple $\mathbb{P}^i \mathbf{A} = \mathbb{L}^i \mathbb{U}^i$, entonces es posible obtener otra matriz de permutación \mathbb{P}^{i+1} y otras matrices \mathbb{L}^{i+1} y \mathbb{U}^{i+1} con una estructura similar a (6) tales que $\mathbb{P}^{i+1} \mathbf{A} = \mathbb{L}^{i+1} \mathbb{U}^{i+1}$. La diferencia radica en que la nueva matriz \mathbb{U}^{i+1} tiene más ceros debajo de diagonal que \mathbb{U}^i , mientras que la nueva matriz \mathbb{L}^{i+1} tiene menos ceros debajo de la diagonal, aunque sigue siendo triangular inferior.

Por tanto, comenzamos tomando $\mathbb{P}^1 = \mathbb{I}_{N \times N}$, $\mathbb{L}^1 = \mathbb{I}_{N \times N}$, $\mathbb{U}^1 = \mathbf{A}$ (en cuyo caso (6) y (7) se satisfacen automáticamente), y realizamos el proceso anterior $N - 1$ veces. De esta forma, generamos sucesivamente los conjuntos de matrices $\{\mathbb{P}^2, \mathbb{L}^2, \mathbb{U}^2\}, \dots, \{\mathbb{P}^N, \mathbb{L}^N, \mathbb{U}^N\}$. Al terminar el proceso, \mathbb{P}^N es una matriz de permutación, \mathbb{L}^N es una matriz triangular inferior y \mathbb{U}^N es una matriz triangular superior. Por tanto, tenemos la factorización (3) buscada, donde $\mathbb{P} = \mathbb{P}^N$, $\mathbb{L} = \mathbb{L}^N$, $\mathbb{U} = \mathbb{U}^N$.

Para implementar la factorización LU con pivotaje parcial, conviene darse cuenta de que no es necesario calcular explícitamente las matrices Π^i , Λ^i , ni almacenar las matrices intermedias $\tilde{\mathbb{L}}^i$, $\tilde{\mathbb{U}}^i$, etc. En su lugar, es posible trabajar únicamente con tres matrices en memoria, \mathbb{P} , \mathbb{L} y \mathbb{U} , que se inicializan con los valores indicados anteriormente, $\mathbb{I}_{N \times N}$, $\mathbb{I}_{N \times N}$ y \mathbf{A} , y se modifican continuamente de acuerdo a las operaciones que hay que hacer sobre las mismas. Véanse el ejercicio 1 y el algoritmo 1. (En realidad, dada la estructura de \mathbb{L}^i y \mathbb{U}^i (cf. (6)), es posible almacenar estas dos matrices en el espacio ocupado por una sola matriz, aunque por simplicidad aquí se almacenarán de forma separada.)

Nota 1: El coste de realizar la factorización LU es $\mathcal{O}(N^3)$, mientras que el de resolver los sistemas triangulares asociados (4)-(5) es $\mathcal{O}(N^2)$. Por tanto, si han de resolverse varios sistemas $\mathbf{A}\mathbf{x}_1 = \mathbf{b}_1$, $\mathbf{A}\mathbf{x}_2 = \mathbf{b}_2$, ... caracterizados por la misma matriz \mathbf{A} , es conveniente reutilizar la factorización LU de la misma, especialmente si N es grande. Es decir, se calcula primero la factorización $\mathbf{P}\mathbf{A} = \mathbb{L}\mathbb{U}$, y luego se resuelve cada sistema aplicando directamente (4)-(5).

En Matlab, la factorización LU con pivotaje parcial se puede calcular con el comando

$$[\mathbb{L}, \mathbb{U}, \mathbb{P}] = \text{lu}(\mathbf{A});$$

mientras que los sistemas triangulares asociados (4)-(5) se pueden resolver mediante

$$\mathbf{x} = \mathbb{U} \setminus (\mathbb{L} \setminus (\mathbb{P} * \mathbf{b}));$$

Nótese el uso de los paréntesis en la expresión anterior para asegurar que las operaciones se hacen en el orden correcto.

3. Matrices dispersas

En muchas ocasiones, la matriz A del sistema no es *llena*, sino *dispersa*, es decir, está formada por una gran cantidad de elementos nulos. De forma resumida, se dispone de N ecuaciones con N incógnitas, pero en cada ecuación sólo intervienen explícitamente unas pocas incógnitas. Esto sucede, por ejemplo, en problemas de circuitos eléctricos y estructuras, así como en la resolución de algunas ecuaciones en derivadas parciales. Véase el problema 6.8 a modo de ejemplo.

Cuando una matriz $A \in \mathbb{R}^{M \times N}$ es dispersa, es conveniente no almacenar todos sus elementos, sino únicamente aquéllos no nulos, especialmente si la matriz es de grandes dimensiones. De esta forma, el consumo de memoria se reduce de $\mathcal{O}(MN)$ a $\mathcal{O}(M)$ (suponiendo que el número de elementos no nulos por fila es $\mathcal{O}(1)$). Además los productos de la matriz con otras matrices o vectores se pueden realizar de forma mucho más eficiente si sólo se tienen en cuenta los términos no nulos.

Supongamos que queremos almacenar la matriz de dimensiones 2×5

$$A = \begin{bmatrix} & 1.1 & -2.2 & & 1.5 \\ \pi & & & 12 & \end{bmatrix}. \quad (12)$$

como matriz dispersa. Esto se puede conseguir mediante las órdenes:

```
filas = [1, 1, 1, 2, 2];
cols  = [2, 3, 5, 1, 4];
vals  = [1.1, -2.2, 1.5, pi, 12];
A      = sparse(filas, cols, vals, 2, 5);
```

Es decir, se crean tres vectores que contienen la fila, la columna y el valor correspondiente a cada elemento no nulo, y se ejecuta la orden `sparse` especificando la dimensión de la matriz. Es importante notar que:

- No es necesario introducir los elementos en ningún tipo de orden; Matlab automáticamente los reordena de la forma más conveniente.
- Si en los vectores `filas`, `cols`, `vals` se introducen **varios valores asociados a una misma posición (fila y columna)** en la matriz, el comando `sparse` los suma.

Así pues,

```
filas = [2, 1, 1, 2, 1, 2, 2];
cols  = [1, 3, 5, 4, 2, 4, 4];
vals  = [pi, -2.2, 1.5, 10, 1.1, 6, -4];
A      = sparse(filas, cols, vals, 2, 5);
```

genera la misma matriz.

La matriz indicada por (12) es demasiado pequeña para apreciar el beneficio de declarar una matriz como dispersa. Un ejemplo más característico sería el indicado por la figura 1, que muestra los patrones de elementos no nulos correspondientes a las matrices de los problemas 6.6 y 6.8. (Dichos patrones pueden obtenerse con el comando `spy(A)`.) Como puede verse, el número de elementos no nulos es muy inferior al número total de elementos de la matriz, de ahí que resulte ventajoso declarar la matriz como dispersa. Notemos también que para considerar que una matriz es dispersa

no es necesario que los elementos no nulos sigan un patrón en particular; simplemente, se necesita que el número de elementos no nulos sea muy pequeño en comparación con el número total de elementos.

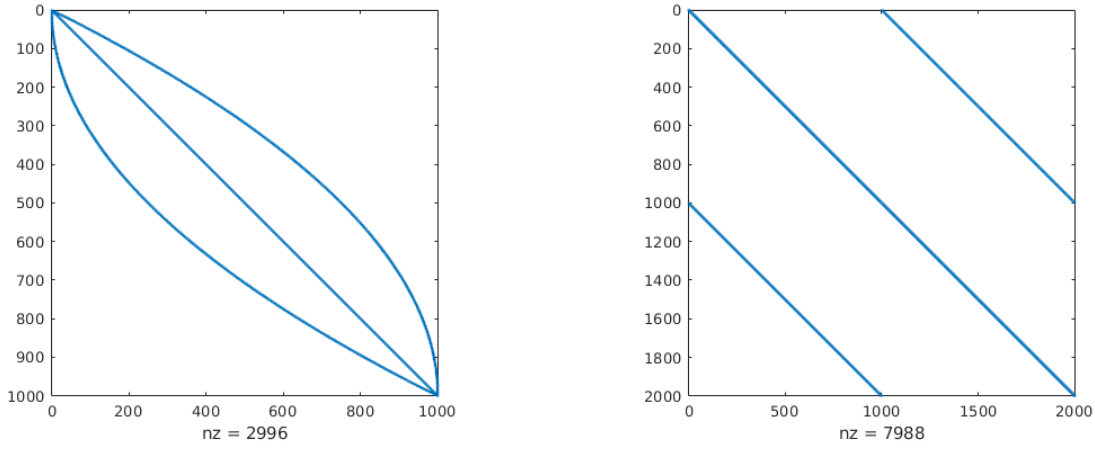


Figura 1: Ejemplos de patrones de elementos no nulos en matrices dispersas. Las filas y columnas están indicadas en los ejes, y los elementos no nulos están resaltados en azul.

4. Factorización LU dispersa

En el caso de matrices dispersas, la factorización LU incluye también pivotaje de columnas, es decir, es de la forma

$$PAQ = LU, \quad (13)$$

donde A es la matriz a factorizar, P y Q son matrices de permutación, L es una matriz triangular inferior y U es una matriz triangular superior. Todas estas matrices son cuadradas. Si la factorización (13) se aplica al sistema original $Ax = b$, éste se lee como

$$\begin{aligned} PAQQ^T x &= Pb, \\ LUQ^T x &= Pb, \end{aligned}$$

y por tanto x puede hallarse a partir de

$$Ly = Pb, \quad (14)$$

$$Uz = y, \quad (15)$$

$$Q^T x = z. \quad (16)$$

A diferencia del sistema original, los sistemas (14)-(15) son triangulares y por tanto mucho más fáciles de resolver, mientras que la solución de (16) es simplemente $x = Qz$ ya que Q es ortonormal.

Al aplicar pivotaje de filas y columnas (o *pivotaje total*), se consigue que el producto PAQ tenga los elementos no nulos más cerca de la diagonal que la matriz original A , lo que resulta en unas matrices L y U más dispersas y, por tanto, en una mayor eficiencia. Un ejemplo característico puede ser el mostrado en la figura 2 (correspondiente a la matriz del problema 6.6 o la figura 1(izqda.)), que compara el patrón de elementos no nulos de los factores L y U para los casos de pivotaje parcial (cf. sección 2) y de pivotaje total. Puede verse que en el caso de pivotaje parcial los factores L y U sufren

un fenómeno conocido como *llenado*, mientras que para el caso de pivotaje total el número total de elementos no nulos se mantiene más reducido.

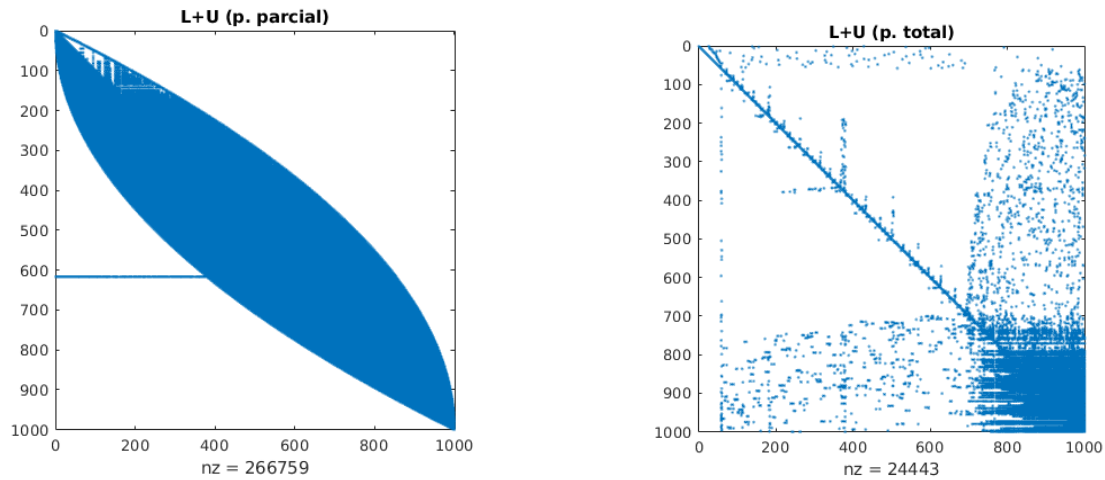


Figura 2: Ejemplo de patrón de elementos no nulos en los factores L y U en la descomposición LU con pivotaje parcial (izqda.) y con pivotaje total (dcha.). El número de elementos no nulos se indica en la parte inferior de cada gráfica.

Para hallar las matrices de permutación P y Q tales que los factores L y U quedan lo más dispersos posible, hay que emplear unos algoritmos de permutación que quedan fuera del alcance de este curso. En lo que sigue, la factorización LU dispersa se calculará mediante el comando:

$$[L, U, P, Q] = \text{lu}(A);$$

mientras que la resolución de (14)-(16) se puede implementar como

$$x = Q * (U \setminus (L \setminus (P * b)));$$

Nótese de nuevo el uso de los paréntesis para garantizar que las operaciones se realizan en el orden más adecuado.

Se recomienda resolver los problemas 6.6-6.7.

Nota 2: Para el caso de matrices dispersas, no hay una expresión clara para el coste computacional de la factorización LU, ya que depende del nivel de llenado de los factores L y U . Sin embargo, sigue siendo mucho más costoso calcular la factorización que resolver los sistemas triangulares asociados. Por tanto, si hay que resolver varios sistemas lineales caracterizados por la misma matriz, es importante reutilizar dicha factorización para reducir el coste computacional.

5. Otras factorizaciones

La factorización LU estudiada anteriormente (cf. secciones 2 y 4) es la más adecuada para el caso general (matriz no simétrica). Ahora bien, conviene destacar que existen otras factorizaciones. Por ejemplo, si $A \in \mathbb{R}^{N \times N}$ es una matriz *simétrica definida positiva*¹, puede utilizarse la *factorización de*

¹Recuérdese que una matriz A simétrica definida positiva es aquella que, además de ser simétrica ($A^T = A$), satisface la propiedad $x^T A x > 0$, $\forall x \neq 0$.

Cholesky

$$\begin{aligned} A &= LL^T \quad (\text{matriz llena}), \\ Q^T A Q &= LL^T \quad (\text{matriz dispersa}), \end{aligned}$$

donde L es una matriz triangular inferior y Q es una matriz de permutación. Dicha factorización se obtiene mediante el comando

```
L = chol(A, 'lower'); %Matriz llena
[L,~,Q] = chol(A, 'lower'); %Matriz dispersa
```

Por otra parte, si se dispone de una matriz rectangular $A \in \mathbb{R}^{M \times N}$, $M \geq N$, puede utilizarse la *factorización QR*:

$$A = QR, \quad (17)$$

Es importante resaltar que hay un cambio de notación en (17). Ahora, $Q \in \mathbb{R}^{M \times N}$ no es una matriz de permutación, sino una matriz ortonormal tal que su espacio columna es el mismo que el de A , mientras que $R \in \mathbb{R}^{N \times N}$ es una matriz triangular superior. Dicho cambio de notación es habitual en la literatura. La factorización anterior puede obtenerse con el comando

```
[Q,R] = qr(A, 0);
```

Notemos también que ahora no se tratan por separado los casos de A llena y dispersa. Esto se debe a que, en general, la matriz Q no es dispersa en ninguno de los dos casos. No obstante, si A es dispersa, es posible encontrar factorizaciones QR alternativas (consúltese `help qr`), aunque quedan fuera del alcance de este curso.

La factorización QR es útil para hallar una base ortogonal del espacio columna de A (los vectores de dicha base son las columnas de Q), y también para resolver sistemas sobredeterminados por mínimos cuadrados. Es decir, si se tiene un sistema de la forma $Ax = b$ con más ecuaciones que incógnitas, éste no tendrá solución en un caso general. En su lugar, se trata de hallar el vector x que minimiza $\|Ax - b\|_2$. Es posible demostrar que x es entonces la solución de

$$A^T A x = A^T b. \quad (18)$$

Utilizando la factorización QR, es posible resolver (18) sin necesidad de calcular explícitamente (y factorizar) $A^T A$. En efecto, si sustituimos (17) en (18), se tiene

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

La expresión anterior involucra dos sistemas triangulares y, por tanto, sencillos de resolver.

6. Problemas

6.1. Factorización LU con pivotaje parcial: implementación

Ejercicio 1: Implementar una función $[L, U, P] = \text{LUFactor}(A)$ que reciba una matriz A y que devuelva los factores de la descomposición $PA = LU$. Para ello, puede procederse como se indica en el algoritmo 1 y puede partirse de la función `LUFactor_incompleto`.

Algoritmo 1: Factorización $PA = LU$

```
1 Datos: Matriz  $A \in \mathbb{R}^{N \times N}$ .
2 Resultado: Factores  $P, L, U$  de la descomposición LU con pivotaje parcial.
3 inicializar  $P = I_{N \times N}, L = I_{N \times N}, U = A$ ,
4 for  $i=1:N-1$  do
5     hallar menor  $m \in \{i, \dots, N\}$  tal que  $|U_{mi}| \geq |U_{ki}|, k = i, \dots, N$ ,
6     permutar fila  $P_{i:}$  de  $P$  con fila  $P_{m:}$ ,
7     permutar subfila  $L_{i,1:i-1}$  de  $L$  con subfila  $L_{m,1:i-1}$ ,
8     permutar subfila  $U_{i,i:N}$  por de  $U$  con subfila  $U_{m,i:N}$ ,
9     for  $k=i+1:N$  do
10        actualizar  $L_{ki} \leftarrow U_{ki}/U_{ii}$ ,
11        actualizar  $U_{k,i:N} \leftarrow U_{k,i:N} - L_{ki}U_{i,i:N}$ .
12    end
13 end
```

Ejercicio 2: Demostrar que el sistema (4)-(5) puede resolverse siguiendo las expresiones

$$\begin{aligned} \mathbf{c} &= P\mathbf{b}, \\ y_i &= c_i - \sum_{j=1}^{i-1} L_{ij}y_j, \quad i = 1, \dots, N, \\ x_i &= \frac{1}{U_{ii}} \left(y_i - \sum_{j=i+1}^N U_{ij}x_j \right), \quad i = N, \dots, 1. \end{aligned}$$

Ejercicio 3: Implementar una función $\mathbf{x} = \text{LUSolve}(L, U, P, \mathbf{b})$ que reciba los factores L, U, P de la descomposición LU con pivotaje parcial y un vector \mathbf{b} y que resuelva el sistema $LU\mathbf{x} = P\mathbf{b}$ siguiendo las expresiones indicadas en el ejercicio 2. No se permite utilizar aquí el comando `\` de Matlab.

6.2. Factorización LU con pivotaje parcial: coste computacional

Ejercicio 4: Implementar una función $SL_LU_coste(Nv)$ que reciba un vector Nv con varios valores de la dimensión N del sistema lineal (por ejemplo, $Nv=[10, 100, 200, 500, 1000]$), y que realice las siguientes tareas:

- Para cada valor de N :
 - Crear una matriz aleatoria $A \in \mathbb{R}^{N \times N}$ y un vector aleatorio $\mathbf{x}^* \in \mathbb{R}^N$ (utilizar el comando `rand`). Definir $\mathbf{b} := A\mathbf{x}^*$.
 - Hallar la factorización LU con la función `LUFactor`. Medir el tiempo empleado (comandos `tic` y `toc`).

- Resolver el sistema $\mathbf{Ax} = \mathbf{b}$ con la función `LUSolve`. Medir el tiempo empleado. Hallar el error $e := \|\mathbf{x} - \mathbf{x}^*\|_\infty$ entre la solución numérica \mathbf{x} y la solución exacta \mathbf{x}^* (utilizar el comando `norm`).

- Representar e frente a N en escala logarítmica (comando `loglog`).
- Representar los tiempos empleados en la factorización LU y en la resolución del sistema frente a N , también en escala logarítmica.

Puede utilizarse la función `SL_LU_coste_incompleto(Nv)`.

Ejercicio 5: Probar la función anterior con distintos valores de Nv . Por ejemplo, puede empezarse por $Nv=[10, 100, 200, 500, 1000]$, aunque los valores más apropiados para estudiar los resultados dependen del ordenador. Para ordenadores más potentes, pueden tomarse valores más grandes de N . ¿Cómo depende el coste computacional de la factorización LU con el tamaño del sistema N ? ¿Y el de la resolución de los sistemas triangulares asociados? A la vista de los resultados, ¿qué habría que hacer si hay que resolver muchos sistemas lineales $\mathbf{Ax} = \mathbf{b}$ con la misma matriz \mathbf{A} pero distintos valores del vector \mathbf{b} ?

Ejercicio 6: Repetir los ejercicios 4-5 utilizando las funciones incluidas en Matlab; en particular, utilizando el comando `lu` en lugar de `LUFactor`, y utilizando el comando barra invertida `\` para resolver los sistemas triangulares asociados. ¿Se observa alguna mejora en los tiempos de cálculo?

6.3. Cargas eléctricas en una barra

A lo largo del eje x hay N cargas eléctricas puntuales, Q_1, \dots, Q_N , situadas de forma equiespaciada entre los puntos $(0.1, 0)$ y $(0.9, 0)$. Asimismo, existen dos hilos infinitos de densidad de carga λ perpendiculares al plano del papel que pasan por los puntos $(0, 0)$ y $(1, 0)$, véase Fig. 3.

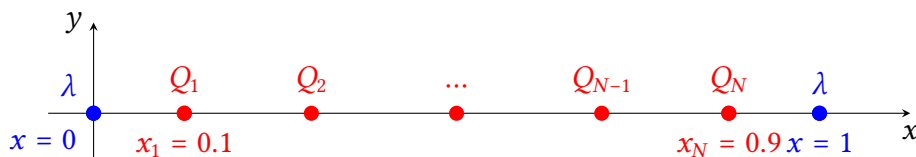


Figura 3: Esquema de las partículas y los hilos con carga eléctrica.

Se desea hallar el valor Q_i de cada carga sabiendo que el sistema está en equilibrio. Se sabe que esto sucede cuando el campo eléctrico sobre cada carga Q_i generado por los hilos y el resto de cargas es nulo, es decir,

$$2\lambda \left(\frac{1}{x_i} - \frac{1}{1-x_i} \right) + \sum_{j=1, j \neq i}^N Q_j \frac{x_i - x_j}{|x_i - x_j|^3} = 0, \quad i = 1, \dots, N, \quad (19)$$

siendo x_i la coordenada horizontal de la carga Q_i .

Ejercicio 7: Implementar una función `SL_ProblemaCargas(lambda, N)` que reciba λ y N y realice las siguientes tareas:

1. Plantear el sistema (19) y resolverlo mediante la factorización LU con pivotaje parcial. Puede resultar útil la función `linspace`.

2. Representar un diagrama con los valores de Q_i frente a la coordenada x_i . Se recomienda utilizar la opción ' -xb ' de la función `plot`.
3. Mostrar por pantalla la suma de todas las cargas. Puede resultar útil el comando `sum`.

Ejercicio 8: Ejecutar la función anterior para $\lambda = 1.2$ y $N = 10, 100, 1000$. ¿Cuál es el valor obtenido en cada caso para la suma de cargas?

Ejercicio 9: Ejecutar la función anterior para $\lambda = 1.2$ y un valor impar de N . ¿Qué sucede? ¿Cómo se puede interpretar desde un punto de vista físico?

6.4. Factorización de Cholesky

Si A es una matriz simétrica definida positiva, podemos buscar una factorización de la forma $A = LL^T$, donde L es una matriz triangular inferior. Entonces,

$$A_{ij} = \sum_{k=1}^j L_{ik}L_{jk} = \sum_{k=1}^{j-1} L_{ik}L_{jk} + L_{ij}L_{jj}. \quad (20)$$

Supongamos que recorremos por filas el bloque triangular inferior de A , es decir, recorremos sus elementos en el orden $A_{11}, A_{21}, A_{22}, A_{31}, A_{32}, A_{33}, A_{41}, \dots$. En ese caso, puede comprobarse los términos que intervienen la Ec. (20) para un cierto A_{ij} son los que han intervenido en el término anterior (A_{ij-1} si $j > 1$ o $A_{i-1,i-1}$ en caso contrario) más un nuevo término, L_{ij} . Es decir, si recorremos el bloque triangular inferior de A por filas, por cada término A_{ij} podrá calcularse un nuevo término L_{ij} . Esto nos deja el siguiente algoritmo: para $i = 1, \dots, N$,

$$L_{ij} = \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk} \right), \quad j = 1, \dots, i-1, \quad (21)$$

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}L_{ik}}. \quad (22)$$

Ejercicio 10: Crear una función `L=CholeskyFactor(A)` que, recibida una matriz simétrica definida positiva A , calcule el factor L de la descomposición de Cholesky mediante el algoritmo indicado por las Ecs (21)-(22).

Ejercicio 11: Crear una función `x=LLTSolve(L,b)` que resuelva el sistema asociado, $LL^T \mathbf{x} = \mathbf{b}$, haciendo uso de que la matriz L es triangular inferior. No se permite utilizar aquí el comando `\` de Matlab.

Ejercicio 12: Representar, en gráficas con escala logarítmica-logarítmica, la dependencia del coste computacional (tiempo de cálculo) de la factorización de Cholesky frente al tamaño del sistema, N , así como el error cometido al resolver un sistema aleatorio. Incluir en las mismas gráficas el coste y el error de la factorización LU con pivotaje parcial. (Véase el ejercicio 5.) ¿Cuál de las dos factorizaciones es más rápida? ¿Y cuál es más precisa? NOTA: para generar una matriz aleatoria A que sea simétrica y definida positiva, se pueden utilizar los comandos `B=rand(N,N)`; `A=B.'*B`.

6.5. Factorización QR

Sea

$$\mathbf{A} = [\mathbf{a}_1 | \cdots | \mathbf{a}_N] \in \mathbb{R}^{M \times N}$$

una matriz rectangular, con $M \geq N$. Mediante el proceso de Gram-Schmidt, es posible obtener la descomposición $\mathbf{A} = \mathbf{Q}\mathbf{R}$, donde

$$\mathbf{Q} = [\mathbf{q}_1 | \cdots | \mathbf{q}_N] \in \mathbb{R}^{M \times N}$$

es una matriz ortonormal, y $\mathbf{R} \in \mathbb{R}^{N \times N}$ es una matriz triangular superior. Notemos que la descomposición también se puede escribir como

$$\mathbf{a}_j = \sum_{i=1}^j \mathbf{q}_i r_{ij}, \quad (23)$$

ya que la j -ésima columna del producto (\mathbf{A}) de dos matrices (\mathbf{Q} y \mathbf{R}) es igual a la primera matriz (\mathbf{Q}) por la j -ésima columna de la segunda matriz (\mathbf{R}).

En particular, cada vector columna \mathbf{q}_j se obtiene de restarle a \mathbf{a}_j su proyección sobre el espacio lineal $\text{span}\{\mathbf{q}_1, \dots, \mathbf{q}_{j-1}\}$ y normalizar el resultado, es decir,

$$\mathbf{v}_j = \mathbf{a}_j - \sum_{i=1}^{j-1} (\mathbf{a}_j \cdot \mathbf{q}_i) \mathbf{q}_i, \quad \mathbf{q}_j = \frac{\mathbf{v}_j}{\|\mathbf{v}_j\|}, \quad j = 1 \dots, N, \quad (24)$$

de tal forma que

$$\mathbf{a}_j = \sum_{i=1}^{j-1} (\mathbf{a}_j \cdot \mathbf{q}_i) \mathbf{q}_i + \|\mathbf{v}_j\| \mathbf{q}_j. \quad (25)$$

Identificando (23) con (25) se tiene que $r_{ij} = \mathbf{q}_i \cdot \mathbf{a}_j$ si $i < j$, y $r_{jj} = \|\mathbf{v}_j\|$.

Ejercicio 13: Sea π_k la proyección de un vector cualquiera $\mathbf{a} \in \mathbb{R}^M$ sobre el espacio $S_k := \text{span}\{\mathbf{q}_1, \dots, \mathbf{q}_k\}$, $k \leq N$, y sea

$$\mathbf{v} := \mathbf{a} - \pi_k \mathbf{a}.$$

la componente ortogonal de \mathbf{a} sobre el espacio S_k . Demostrar que

$$\mathbf{v} \cdot \mathbf{q} = \mathbf{a} \cdot \mathbf{q}, \quad \forall \mathbf{q} \perp S_k.$$

Solución: Si empleamos la definición de \mathbf{v} , se tiene

$$\mathbf{v} \cdot \mathbf{q} = \mathbf{a} \cdot \mathbf{q} - \pi_k \mathbf{a} \cdot \mathbf{q}$$

Ahora bien, puesto que $\pi_k \mathbf{a} \in S_k$, se tiene $\pi_k \mathbf{a} \cdot \mathbf{q} = 0$ para todo \mathbf{q} ortogonal a S_k y queda demostrado el enunciado anterior. \square

Ejercicio 14: Se considera ahora un espacio de dimensión superior $S_l := \text{span}\{\mathbf{q}_1, \dots, \mathbf{q}_l\}$, con $l > k$, de tal forma que $S_k \subset S_l$. Demostrar que las componentes ortogonales a S_l de \mathbf{v} y \mathbf{a} son iguales, es decir,

$$\mathbf{v} - \pi_l \mathbf{v} = \mathbf{a} - \pi_l \mathbf{a}.$$

Solución: Si empleamos nuevamente la definición de \mathbf{v} , se tiene

$$\mathbf{v} - \pi_l \mathbf{v} = (\mathbf{a} - \pi_k \mathbf{a}) - \pi_l (\mathbf{a} - \pi_k \mathbf{a}) = \mathbf{a} - \pi_k \mathbf{a} - \pi_l \mathbf{a} + \pi_l (\pi_k \mathbf{a}).$$

Ahora bien, puesto que $\pi_k \mathbf{a} \in S_k \subset S_l$, entonces se tiene $\pi_l (\pi_k \mathbf{a}) = \pi_k \mathbf{a}$, y la expresión anterior se reduce a $\mathbf{a} - \pi_l \mathbf{a}$. \square

El algoritmo de Gram-Schmidt original, dado por (24), es inestable. Una forma más estable de implementarlo consiste en actualizar progresivamente el valor de cada vector \mathbf{v}_j (es decir, de la componente ortogonal de cada vector \mathbf{a}_j respecto del espacio $\text{span}\{\mathbf{q}_1, \dots, \mathbf{q}_{j-1}\}$) conforme se calculan los vectores \mathbf{q}_i haciendo uso de los resultados de los ejercicios 13-14. Esto se detalla en el algoritmo 2.

Algoritmo 2: Método de Gram-Schmidt modificado

```

1 Datos: Matriz  $\mathbf{A} \in \mathbb{R}^{M \times N}$ .
2 Resultado: Factores  $\mathbf{Q}$  y  $\mathbf{R}$  de la descomposición QR.
3 inicializar  $\mathbf{v}_j = \mathbf{a}_j, j = 1, \dots, N$ ,
4 for  $i=1:N$  do
5     calcular  $R_{ii} = \|\mathbf{v}_i\|, \mathbf{q}_i = \mathbf{v}_i/R_{ii}$ ,
6     for  $j=i+1:N$  do
7         proyectar  $\mathbf{a}_j$  sobre  $\mathbf{q}_i$ :  $R_{ij} = \mathbf{q}_i \cdot \mathbf{v}_j$ ,
8         actualizar componente ortogonal sobre  $S_i$ :  $\mathbf{v}_j \leftarrow \mathbf{v}_j - R_{ij}\mathbf{q}_i$ .
9     end
10 end

```

Ejercicio 15: Implementar una función $[Q, R] = \text{QRFactor}(\mathbf{A})$ según se indica en el algoritmo 2.

6.6. Factorización LU dispersa (I)

Sea $\mathbf{A} = \{A_{ij}\}$ una matriz de dimensiones $N \times N$ cuyos elementos no nulos son $A_{11} = A_{NN} = 1$ y

$$A_{i,i-\Delta_i} = -3, \quad A_{ii} = 7, \quad A_{i,i+\Delta_i} = -4, \quad i = 2, \dots, N-1,$$

donde $\Delta_i := \lceil (N-i)(i-1)/(N-1) \rceil$. NOTA: $\lceil \cdot \rceil$ denota la función techo, que se puede hallar con el comando `ceil` de Matlab.

Ejercicio 16: Implementar una función $\text{SL_LUDispersa}(N)$ que reciba N y realice las siguientes tareas:

- Crear la matriz \mathbf{A} .
- Crear un vector \mathbf{x}^* aleatorio y definir $\mathbf{b} = \mathbf{A}\mathbf{x}^*$.
- Hallar la factorización $\mathbf{PA} = \mathbf{LU}$ (es decir, la factorización LU con pivotaje parcial). Mostrar por pantalla el tiempo empleado y representar el patrón de elementos no nulos de \mathbf{PA} y $\mathbf{L} + \mathbf{U}$ con la orden `spy`. (Emplear aquí el comando `lu` de Matlab.)
- Resolver el sistema $\mathbf{Ax} = \mathbf{b}$ a partir de la factorización anterior. Mostrar por pantalla el tiempo de empleado y el error $\|\mathbf{x} - \mathbf{x}^*\|_\infty$.
- Hallar la factorización $\mathbf{PAQ} = \mathbf{LU}$ (es decir, la factorización LU para matrices dispersas). Mostrar por pantalla el tiempo empleado y representar el patrón de elementos no nulos de \mathbf{PAQ} y $\mathbf{L} + \mathbf{U}$.
- Resolver el sistema $\mathbf{Ax} = \mathbf{b}$ a partir de la factorización anterior. Mostrar por pantalla el tiempo de empleado y el error $\|\mathbf{x} - \mathbf{x}^*\|_\infty$.

Puede utilizarse la función $\text{SL_LUDispersa_incompleto}(N)$.

Ejercicio 17: Ejecutar la función anterior para valores cada vez mayores de N , pero sin que el ordenador se quede sin memoria. Obsérvense los patrones de elementos no nulos de \mathbf{PA} y de \mathbf{PAQ} . ¿Cuál tiene los elementos más cerca de la diagonal?

Ejercicio 18: Variando N nuevamente, obsérvense ahora los patrones de elementos no nulos de $\mathbb{L} + \mathbb{U}$ para ambas factorizaciones. ¿Qué factorización tiene un menor número de elementos no nulos?

Ejercicio 19: ¿Qué puede decirse sobre los tiempos de cálculo y sobre el error cometido?

6.7. Factorización LU dispersa (II)

Sea $\mathbf{A} = \{A_{ij}\}$ una matriz de dimensiones $N \times N$ cuyos elementos no nulos son $A_{11} = 1$ y

$$A_{ij} = i + j, \quad i = 2, \dots, N, \quad \text{si } |i - j| \text{ es múltiplo de } \lceil i/1.5 \rceil.$$

Ejercicio 20: Implementar una función $SL_LUDispersa2(N)$ que reciba N y realice las mismas tareas que las indicadas en el ejercicio 16. Realizar de nuevo los ejercicios 17-19.

6.8. Circuito eléctrico

Un circuito eléctrico plano está formado por $2N$ nodos. La posición de cada nodo i viene dada por las coordenadas polares

$$\begin{aligned} r_i = 1, \quad \theta_i &= \cos\left(\frac{2\pi}{N-1}(i-1)\right), \quad i = 1, \dots, N, \\ r_i = 2, \quad \theta_i &= \cos\left(\frac{2\pi}{N-1}(i-(N+1))\right), \quad i = N+1, \dots, 2N, \end{aligned}$$

Nótese que el nodos 1 y el N son coincidentes, al igual que los nodos $N+1$ y $2N$. Esto se hace para facilitar la construcción del sistema. Véase Fig. 4.

Los nodos 1 y N están conectados a tierra, mientras que en los nodos $N+1$ y $2N$ se aplica una tensión $\bar{V} = 2.8$. Esto proporciona las ecuaciones

$$V_1 = 0, \quad V_N = 0, \quad V_{N+1} = \bar{V}, \quad V_{2N} = \bar{V}. \quad (26)$$

Para el resto de nodos, se tiene en cuenta la siguiente información:

- Un nodo interior i , $i = 2, \dots, N-1$ está conectado a los nodos $i-1$, $i+1$ y $N+i$ mediante resistencias de valor $R = 1.5$.
- Un nodo exterior $N+i$, $i = 2, \dots, N-1$, está conectado a los nodos $N+i-1$, $N+i+1$ e i mediante resistencias del mismo valor $R = 1.5$.

Si aplicamos la ley de Kirchhoff (i.e., conservación de la intensidad de corriente), esto proporciona las ecuaciones

$$\frac{3V_i}{R} - \frac{V_{i-1}}{R} - \frac{V_{i+1}}{R} - \frac{V_{N+i}}{R} = 0, \quad i = 2, \dots, N-1, \quad (27)$$

$$\frac{3V_{N+i}}{R} - \frac{V_{N+i-1}}{R} - \frac{V_{N+i+1}}{R} - \frac{V_i}{R} = 0, \quad i = 2, \dots, N-1. \quad (28)$$

Las ecuaciones anteriores determinan un sistema lineal que permite hallar la tensión V_i en cada nodo.

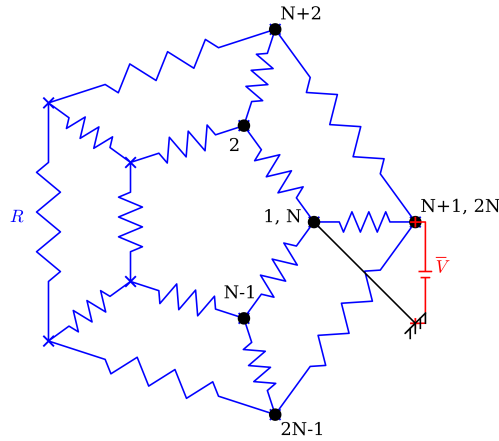


Figura 4: Esquema del circuito eléctrico.

Ejercicio 21: Implementar una función $SL_CircuitoElectrico(N)$ que, recibido N , plantee y resuelva el sistema de ecuaciones (26)-(28). Asimismo, utilizando el comando `plot3`, la función debe realizar una representación 3D para visualizar el potencial en cada nodo como una nube de puntos (opción `'*b'`).

Bibliografía

- [1] Davis, T.A.; Rajamanickam, S.; Sid-Lakhdar, W.M., “A survey of direct methods for sparse linear systems,” *Acta Numerica*, vol. 25:pp. 383–566, 2016, URL <http://dx.doi.org/10.1017/S0962492916000076>.
- [2] Trefethen, L.N.; Bau, D., *Numerical Linear Algebra*, SIAM, 2022, URL <https://epubs.siam.org/doi/book/10.1137/1.9781611977165>.
- [3] Chapra, S.C.; Canale, R.P., *Numerical methods for engineers*, McGraw-Hill Education, New York, NY, eighth edition ed., 2021.
- [4] Chasnov, J.R., “Numerical Methods for Engineers,” Tech. rep., The Hong Kong University of Science and Technology, 2020, URL <https://www.math.hkust.edu.hk/~machas/numerical-methods-for-engineers.pdf>.
- [5] Burden, R.; Faires, J.; Burden, A., *Numerical analysis*, Cengage Learning, 2015, URL <https://books.google.es/books?id=9DV-BAAAQBAJ>.
- [6] Kincaid, D.; Cheney, E.W., *Numerical analysis: mathematics of scientific computing*, no. 2 in Pure and applied undergraduate texts, American Mathematical Society, Providence, Rhode Island, third edition ed., 2009.