Prueba de Evaluación Continua 2

Adolfo Hilario Garcia

30 de octubre de 2020

El firmante de este trabajo reconoce que todo él es original, de su única autoía, escritura y redacción, y que allí donde han sido empleadas ideas o datos de otros autores, su trabajo ha sido reconocido y ubicado, con suficiente detalle, como para que el lector pueda consultar lo afirmado sobre él.

1 Introducción y enunciado

La PEC 2 de Física Computacional II plantea un problema común de teoría de circuitos: aplicar las leyes de las corrientes y los voltajes de Kirchhoff a un determinado circuito eléctrico. La Figura 1 representa un circuito eléctrico de cuatro mallas cerradas sobre el que aplicar las reglas de Kirchhoff y crear un sistema de ecuaciones a resolver.

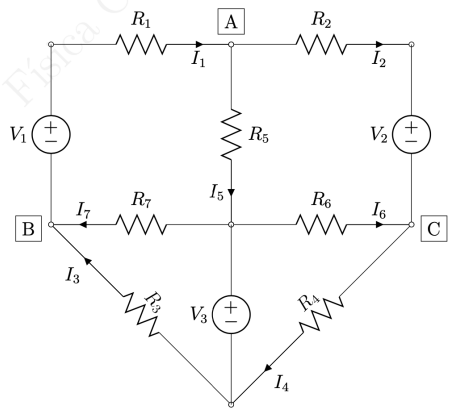

Figura 1: *Circuito eléctrico a partir del cual se desarrolla la prueba*

Las reglas de Kirchhoff se pueden sintetizar de la forma que sigue:

1. En cualquier nodo, la suma de las corrientes que entran en ese nodo es igual a las corrientes que salen, o lo que es lo mismo:

$$\sum_{i=1}^{k} I_k = 0 \tag{1}$$

2. En un circuito cerrado la suma de todas las caídas de tensión es igual a la tensión total suministrada, esto es:

$$\sum_{i=1}^{j} V_j - \sum_{i=1}^{k} I_k R_k = 0 \tag{2}$$

donde V_k representa la fem de la j-ésima fuente de voltaje y $I_k R_k$ las caídas de potenciales en la resistencia R_k en las que se ha aplicado la ley de Ohm, $V_k = I_k R_k$.

A continuación se presentan los objetivos de esta PEC:

(a) Obtener, aplicando las leyes de Kirchhoff un sistema de ecuaciones y analizar sus características.

(b) Obtener, para un orden determinado de las ecuaciones, una solución al sistema mediante la factorización LU de Crout y calcular el coste computacional de éste método.

(c) Volver a solucionar el sistema cambiando el orden de las ecuaciones y comparando los resultados y el coste computacional para este orden concreto respecto al anterior.

(d) Solucionar el sistema mediante la factorización de Doolittle y comparar los resultados y el coste computacional para este método respecto a los anteriores.

(e) Solucionar el sistema de ecuaciones con una nueva factorización L^*U^* y comparar los resultados y el coste computacional para este método respecto a los anteriores.

2 Metodología

Para realizar esta PEC se han escrito programas que agilicen la tarea a la hora de realizar cálculos y obtener resultados. Se ha utilizado para escribirlos el lenguaje de programación C en el editor de texto Geany, que fue introducido en Física Computacional I. También han sido utilizados programas con funciones de cálculo simbólico como Maple o Matlab para comprobar las soluciones obtenidas en cada ejercicio.

La temática de esta prueba permite al programador crear un único programa que incluye los distintos métodos de factorización LU y que éste elige desde `main`. A grandes rasgos, el programa consiste en:

1. Un primer conjunto de funciones que escriben un archivo de datos (en LaTeX o en texto plano) a partir de un conjunto de datos obtenido de un tipo concreto (`int` o `float`). En el caso que nos ocupa, principalmente se escribirán matrices cuadradas o vectores columna.

2. Una función que se encarga de asegurar que se cumple una condición esencial en la factorización LU, que la matriz de coeficientes A se puede triangularizar por Gauss, o lo que es lo mismo, que los elementos de la diagonal $a_{ii} \neq 0$. Para ello, permuta las filas de A hasta conseguir una secuencia válida que aplica tanto a la matriz de coeficientes como a b.

3. Las función `solver` que resuelve un sistema de ecuaciones factorizado de la forma LU haciendo $Ly = b$ y $Ux = y$.

4. Las funciones que LU-factorizan la matriz A por Crout o por Doolittle y que una vez tienen L y U, se las mandan a `solver` para que nos devuelva la solución.
5. Una función general llamada `solucionasistema`, que se encarga de llamar a `permuta` para entonces poder aplicar la factorización que se elija desde `main`.
6. La función `main`, desde la que se define la matriz A, el vector b, y algunas variables más que se escribirán a lo largo de las funciones mencionadas anteriormente, como el vector `secuencia`, que guardará la nueva posición de las filas en A tras aplicarle `permuta`. También elegimos el método concreto que se aplicará. Por último, `main` llama a `solucionasistema`, que como hemos dicho, llama a todas las demás.

En la Sección 5 se tratará en profundidad cada programa por separado.

3 Resultados

A continuación se exponen los resultados junto con los razonamientos e interpretaciones pertinentes para cada apartado.

3.1 Apartados (a)

De aplicar la primera regla de Kirchhoff a cada nodo de la Figura 1 se obtiene el conjunto de ecuaciones:

$$\left.\begin{array}{rcrl} A & : & I_1 - I_2 - I_5 & = 0 \\ B & : & I_1 - I_3 - I_7 & = 0 \\ C & : & I_2 - I_4 + I_6 & = 0 \end{array}\right\} \tag{3}$$

Aplicando la ley de corrientes a cada una de las mayas obtenemos un segundo conjunto de igualdades:

$$\left.\begin{array}{rcrl} \text{Sup. izq.} & : & I_1 + 5I_5 + 5I_7 & = 5 \\ \text{Sup. der.} & : & -3I_2 + 5I_5 + 5I_6 & = 10 \\ \text{Inf. izq.} & : & -10I_3 + 5I_7 & = 10 \\ \text{Inf. der.} & : & 10I_4 + 5I_6 & = 10 \end{array}\right\} \tag{4}$$

De esta forma, podemos definir el sistema de ecuaciones siguiente:

$$\left.\begin{array}{rl} I_1 - I_2 - I_5 & = 0 \\ I_1 - I_3 - I_7 & = 0 \\ I_2 - I_4 + I_6 & = 0 \\ I_1 + 5I_5 + 5I_7 & = 5 \\ -3I_2 + 5I_5 + 5I_6 & = 10 \\ -10I_3 + 5I_7 & = 10 \\ 10I_4 + 5I_6 & = 10 \end{array}\right\} \tag{5}$$

Se trata de un sistema lineal de 7 ecuaciones con siete incógnitas $\{I_1, I_2, I_3, I_4, I_5, I_6, I_7\}$ que corresponden a la intensidad de corriente que pasa por cada resistencia.

3.2 Apartado (b)

Para comprender el método de Crout, primero debemos definir la factorización LU. La factorización matricial LU consiste en expresar una matriz cuadrada A como producto de dos matrices triangulares una triangular inferior, L, y otra triangular superior U (esto es, $A = LU$). En concreto, la factorización de Crout exige que la diagonal principal de U esté formada por unos.

La factorización de Crout sirve para resolver sistemas matriciales de la forma $Ax = b$. Expresado en forma matricial, nuestro sistema de la sección anterior se convierte en

$$\begin{pmatrix} 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & 11 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 5 & 0 & 5 \\ 0 & -3 & 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & -10 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 10 & 0 & 5 & 0 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \\ 10 \\ 10 \\ 10 \end{pmatrix} \tag{6}$$

No obstante, el sistema no cumple con los requisitos para ser factorizado, pues hay elementos en la diagonal principal de A que son nulos. Tendremos pues que permutar algunas filas para poder ejecutar el algoritmo sin problemas. Mediante el algoritmo de la función `permuta` se obtiene la nueva secuencia de filas que debemos escribir: $\{1, 3, 2, 7, 4, 5, 6\}$. Ahora el sistema toma la siguiente forma:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 10 & 0 & 5 & 0 \\ 1 & 0 & 0 & 0 & 5 & 0 & 5 \\ 0 & -3 & 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & -10 & 0 & 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \\ 5 \\ 10 \\ 10 \end{pmatrix} \tag{7}$$

que como se puede observar, tiene una diagonal principal distinta de cero, por lo que ya nos permite aplicar la factorización LU de Crout. A continuación se presentan las matrices, L y U, respectivamente, calculadas por el algoritmo `LU_crout`:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 6 & 0 & 0 \\ 0 & -3 & 0 & -3 & 5 & 10{,}75 & 0 \\ 0 & 0 & -10 & -10 & -10 & 12{,}5 & 28{,}1783 \end{pmatrix} \tag{8}$$

$$U = \begin{pmatrix} 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0{,}5 & 0 \\ 0 & 0 & 0 & 0 & 1 & -0{,}25 & 0{,}833333 \\ 0 & 0 & 0 & 0 & 0 & 1 & -0{,}387597 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{9}$$

Se puede comprobar que $A = LU$. Con esta información ya podemos llamar a `solver` para que nos de la solución. Primero obtenemos que

$$Ly = b \quad \longrightarrow \quad y = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0{,}666667 \\ 0{,}899225 \\ 0{,}547455 \end{pmatrix} \tag{10}$$

Haciendo $y = Ux$ y sustituyendo hacia atrás se obtiene colección de soluciones deseada

$$x = \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \end{pmatrix} = \begin{pmatrix} -0{,}178817 \\ -0{,}667125 \\ -0{,}726273 \\ 0{,}444292 \\ 0{,}488308 \\ 1{,}11142 \\ 0{,}547455 \end{pmatrix} \Longrightarrow \begin{cases} I_1 &= -0.18\,\text{A} \\ I_2 &= -0.67\,\text{A} \\ I_3 &= -0.73\,\text{A} \\ I_4 &= 0.44\,\text{A} \\ I_5 &= 0.49\,\text{A} \\ I_6 &= 1.11\,\text{A} \\ I_7 &= 0.55\,\text{A} \end{cases} \tag{11}$$

donde el signo de la intensidad especifica el sentido de la corriente a lo largo del conductor.

Coste computacional

El coste computacional de un algoritmo consiste en el número de operaciones totales que realiza el computador al ejecutar dicho algoritmo. Se pueden dividir en tres bloques principales: productos y cocientes; sumas y restas, y raíces cuadradas.

Para mayor facilidad a la hora de mostrar el coste computacional de los distintos algoritmos creados en la prueba, se hará referencia a los "pasos"de los algoritmos de Burden y Faires (2011). En el caso de la factorización LU por Crout y Doolittle, se utilizará el algoritmo 6.4. de la página 406. De esta forma podemos ir contando el coste de C_p (coste de productos y cocientes) y de C_s (coste de sumas y restas) en función de la dimensión n de la matriz de coeficientes.

Para el caso que nos ocupa, se obtienen las siguientes igualdades:

$$\begin{aligned} \text{PASO 2} \quad : \quad & C_p = 2(n-1) \\ & C_s = 0 \end{aligned}$$

$$\text{PASO 4} \quad : \quad C_p = \sum_{k=2}^{n-1} k = \frac{(n-2)(n-1)}{2}$$
$$C_s = n - 2 + \sum_{k=3}^{n-1} k = n - 2 + \frac{(n-3)(n-2)}{2}$$

$$\text{PASO 5} \quad : \quad C_p = \sum_{i=1}^{n-2} (n-i)\left[2 + 2\frac{(n-1)n}{2}\right] = \frac{(n+1)(n-2)}{2}\left[2 + (n-1)n\right]$$
$$C_s = \frac{(n+1)(n-2)}{2}\left[2 + (n-3)(n-2)\right]$$

$$\text{PASO 6} \quad : \quad C_p = \sum_{k=2}^{n-1} k = \frac{(n-2)(n-1)}{2}$$
$$C_s = 1 + \sum_{k=3}^{n-1} k = 1 + \frac{(n-3)(n-2)}{2}$$

Ahora pasamos a los 4 pasos de resolución por sustitución hacia adelante en $Ly = b$ y hacia atrás en $Ux = y$ mostrados en la parte inferior de la página 406:

$$\text{SOLVER 1} \quad : \quad C_p = 1$$
$$C_s = 0$$

$$\text{SOLVER 2} \quad : \quad C_p = (n-2)\left[1 + \sum_{k=1}^{n-1} k\right] = (n-2)\left[1 + \frac{(n-1)n}{2}\right]$$
$$C_s == (n-2)\left[1 + \sum_{k=2}^{n-1} k\right] = (n-2)\left[1 + \frac{(n-2)(n-1)}{2}\right]$$

$$\text{SOLVER 3} \quad : \quad C_p = 1$$
$$C_s = 0$$

$$\text{SOLVER 4} \quad : \quad C_p = (n-1)\left[1 + \sum_{k=1}^{n-1} k\right] = (n-1)\left[1 + \frac{(n-1)n}{2}\right]$$
$$C_s = (n-1)\left[1 + \sum_{k=2}^{n-1} k\right] = (n-1)\left[1 + \frac{(n-2)(n-1)}{2}\right]$$

Donde se ha utilizado que la suma de una sucesión de números naturales es acotada, esto es $\sum_{k=1}^{n} k = n(n+1)/2$. Sumando todas estas expresiones y sustituyendo n por 7 se obtiene $C_p = 946$, y $C_s = 382$. Por lo tanto, la factorización por Crout tiene un coste computacional de 946 productos y/o cocientes y 382 sumas y/o restas.

3.3 Apartado (c)

Al reordenar las filas de la matriz A y el vector columna b (6) según la secuencia pedida en el enunciado, que escrita en números sería $\{7, 3, 5, 1, 4, 2, 6\}$ (donde cada elemento del vector es la antigua fila de A en su nueva posición) y reordenando las columnas según la secuencia $\{4, 6, 2, 5, 1, 7, 3\}$ se obtiene el siguiente sistema matricial:

$$\begin{pmatrix} 10 & 5 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & -3 & 5 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 1 & 5 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 5 & -10 \end{pmatrix} \begin{pmatrix} I_4 \\ I_6 \\ I_2 \\ I_5 \\ I_1 \\ I_7 \\ I_3 \end{pmatrix} = \begin{pmatrix} 10 \\ 0 \\ 10 \\ 0 \\ 5 \\ 0 \\ 10 \end{pmatrix} \tag{12}$$

Ahora, la matriz de coeficientes A es una matriz banda, en concreto una matriz tridiagonal. Podemos aplicar de nuevo el método de Crout utilizado en el apartado anterior y se obtendrían resultados similares. No obstante, existe una variante del método para matrices tridiagonales que rebaja el coste computacional, pues tanto la factorización LU como la solución en dos pasos es más rápida que utilizando el método tradicional.

Cabe añadir también que la matriz no sufrirá ninguna permutación de renglones pues la propia definición de matriz tridiagonal implica que la diagonal principal sea distinta de cero.

Al aplicar el método de Crout para matrices tridiagonales se obtienen las siguientes matrices L y U:

$$L' = \begin{pmatrix} 10 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1{,}5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & -6{,}33333 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1{,}78947 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 3{,}79412 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2{,}31783 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & -12{,}1572 \end{pmatrix} \tag{13}$$

$$U' = \begin{pmatrix} 1 & 0{,}5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0{,}666667 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -0{,}789474 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -0{,}558824 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1{,}31783 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0{,}431438 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{14}$$

El algoritmo creado (VEASE) soluciona directamente el sistema sin llamar a `solver`, dada la sencillez de las operaciones. Tenemos que el vector $y = Ux$ es igual a:

$$L'y' = b' \quad \longrightarrow \quad y' = \begin{pmatrix} 1 \\ 0{,}666667 \\ -1{,}05263 \\ 0{,}588235 \\ 0{,}542636 \\ 0{,}234114 \\ -0{,}726272 \end{pmatrix} \tag{15}$$

Entonces, el vector solución x' es igual a:

$$x' = \begin{pmatrix} I_4 \\ I_6 \\ I_2 \\ I_5 \\ I_1 \\ I_7 \\ I_3 \end{pmatrix} = \begin{pmatrix} 0{,}444292 \\ 1{,}11142 \\ -0{,}667125 \\ 0{,}488308 \\ -0{,}178817 \\ 0{,}547455 \\ -0{,}726272 \end{pmatrix} \Longrightarrow \begin{cases} I_1 &= -0.18\,\text{A} \\ I_2 &= -0.67\,\text{A} \\ I_3 &= -0.73\,\text{A} \\ I_4 &= 0.44\,\text{A} \\ I_5 &= 0.49\,\text{A} \\ I_6 &= 1.11\,\text{A} \\ I_7 &= 0.55\,\text{A} \end{cases} \tag{16}$$

Al reordenar los términos se puede apreciar que los resultados son idénticos respecto a la distribución de filas y columnas anterior utilizando un método computacionalmente más ágil, como se comprobará en la sección que sigue.

Coste computacional

Siguiendo los pasos realizados en el algoritmo 6.7 de la página 422 de Burden y Faires (2011) tenemos que

$$\begin{aligned} \text{PASO 1} \quad &: \quad C_p = 2 \\ &\qquad\; C_s = 0 \end{aligned}$$

$$\begin{aligned} \text{PASO 2} \quad &: \quad C_p = 4(n-2) \\ &\qquad\; C_s = 2(n-2) \end{aligned}$$

$$\begin{aligned} \text{PASO 3} \quad &: \quad C_p = 3 \\ &\qquad\; C_s = 2 \end{aligned}$$

$$\begin{aligned} \text{PASO 4} \quad &: \quad C_p = n-1 \\ &\qquad\; C_s = n-1 \end{aligned}$$

Con lo que se obtiene que $C_p = 31$ y $C_s = 18$, dos números mucho menores que los que se obtendrían si hiciesemos una factorización LU convencional, sin tener en cuenta las características específicas de la matriz de coeficientes.

3.4 Apartado (d)

La factorización LU por Doolittle sigue esencialmente los mismos pasos que la de Crout, solo que ahora es L la que tiene una diagonal de unos. Llamando a la función `LU_doolittle` para que factorice A' se obtienen las siguientes matrices:

$$L'' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0{,}1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3{,}33333 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}157895 & 1 & 0 & 0 & 0 \\ 0 & 0 & -0 & -2{,}79412 & 1 & 0 & 0 \\ 0 & 0 & -0 & -0 & -0{,}263566 & 1 & 0 \\ 0 & 0 & -0 & -0 & 0 & 2{,}15719 & 1 \end{pmatrix} \tag{17}$$

$$U'' = \begin{pmatrix} 10 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1{,}5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -6{,}33333 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1{,}78947 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3{,}79412 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2{,}31783 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -12{,}1572 \end{pmatrix} \tag{18}$$

Una vez calculadas L y U, se llama a `solver` para que acabe la tarea. Primero, se hace $Ly = b$ para obtener y:

$$L''y'' = b' \quad \longrightarrow \quad y'' = \begin{pmatrix} 10 \\ 1 \\ 6{,}66667 \\ -1{,}05263 \\ 2{,}05882 \\ 0{,}542636 \\ 8{,}82943 \end{pmatrix} \tag{19}$$

$$x'' = \begin{pmatrix} I_4 \\ I_6 \\ I_2 \\ I_5 \\ I_1 \\ I_7 \\ I_3 \end{pmatrix} = \begin{pmatrix} 0{,}444292 \\ 1{,}11142 \\ -0{,}667125 \\ 0{,}488308 \\ -0{,}178817 \\ 0{,}547455 \\ -0{,}726272 \end{pmatrix} \Longrightarrow \begin{cases} I_1 & = -0.18\,\text{A} \\ I_2 & = -0.67\,\text{A} \\ I_3 & = -0.73\,\text{A} \\ I_4 & = 0.44\,\text{A} \\ I_5 & = 0.49\,\text{A} \\ I_6 & = 1.11\,\text{A} \\ I_7 & = 0.55\,\text{A} \end{cases} \tag{20}$$

Coste computacional

El algoritmo que factoriza una matriz en una triangular inferior y una triangular superior por Doolittle es idéntica en cuanto a coste computacioanl respecto a la factorización por Crout. Por lo tanto, se obtiene que se realizan 946 productos y/o cocientes y 382 sumas y/o restas.

3.5 Apartado (e)

Partiendo del hecho que los elementos de la diagonal de U^* y de L^* deben ser iguales, se puede pensar que el enunciado nos pide una factorización LL^T mediante el algoritmo de Cholesky. Sin embargo, el sistema expresado en (12) no se puede utilizar para este método en concreto, pues requiere que A' sea una matriz definida positiva. Veamos por qué no se cumplen las condiciones.

Una matriz es definida positiva si:

1. tiene inversa,
2. los elementos de la diagonal son mayores que cero,
3. el elemento de la diagonal es el máximo de su fila en valor absoluto y

4. para cada $i \neq j$ se cumple que $(a_{ij})^2 < a_{ii}a_{jj}$.

La primera de las condiciones se cumple, pues $|A'| = -5825$, que es distinto de cero. Al aplicar el algoritmo de permutación añadiendo las tres condiciones siguientes se llega a la conclusión de que no hay combinación posible que cumpla con las tres condiciones al mismo tiempo. En conclusión, la matriz no puede ser convertida en definida positiva, y, por tanto, no se le puede aplicar el algoritmo de Cholesky.

Sin embargo, se puede forzar una factorización LU en que los elementos de las diagonales sean iguales, una especie de híbrido entre una factorización LU y el método de Cholesky.

Como se muestra en detalle en la Sección 5, al suponer que $L_{ii} = U_{ii}$, tenemos que, según el algoritmo de factorización LU:

$$L_{ii} = U_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}U_{ki}} \tag{21}$$

Que es muy parecido a lo que obtendríamos si hacemos Cholesky. No obstante, al tratarse de una matriz que no es definida positiva, la raíz puede hacerse negativa a partir de cierto punto. Teniendo esto en cuenta, se obtienen las matrices L y U mostradas a continuación:

$$L^* = \begin{pmatrix} 3{,}16 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ -0{,}32 + 0{,}00i & 1{,}22 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 4{,}08 + 0{,}00i & 0{,}00 + 2{,}52i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}40i & 0{,}00 + 1{,}34i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + -3{,}74i & 1{,}95 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & -0{,}51 + 0{,}00i & 1{,}52 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 3{,}28 + 0{,}00i & 0{,}00 + 3{,}49i \end{pmatrix}$$

$$U^* = \begin{pmatrix} 3{,}16 + 0{,}00i & 1{,}58 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 1{,}22 + 0{,}00i & 0{,}82 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 2{,}52i & 0{,}00 + -1{,}99i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 1{,}34i & 0{,}00 + -0{,}75i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 1{,}95 + 0{,}00i & 2{,}57 + 0{,}00i & 0{,}00 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 1{,}52 + 0{,}00i & 0{,}66 + 0{,}00i \\ 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 0{,}00i & 0{,}00 + 3{,}49i \end{pmatrix}$$

que cumplen que sus elementos diagonales son iguales. Además, al utilizar estas matrices para solucionar el sistema se obtiene que:

$$L^*y^* = b' \quad \longrightarrow \quad y^* = \begin{pmatrix} 3{,}1623 + 0{,}0000i \\ 0{,}8165 + 0{,}0000i \\ 0{,}0000 + -2{,}6491i \\ 0{,}0000 + 0{,}7869i \\ 1{,}0570 + 0{,}0000i \\ 0{,}3564 + 0{,}0000i \\ 0{,}0000 + -2{,}5323i \end{pmatrix} \tag{22}$$

Al hacer $y^* = U^*x^*$, se obtiene el conjunto de soluciones:

$$x'' = \begin{pmatrix} I_4 \\ I_6 \\ I_2 \\ I_5 \\ I_1 \\ I_7 \\ I_3 \end{pmatrix} = \begin{pmatrix} 0{,}4443 \\ 1{,}1114 \\ -0{,}6671 \\ 0{,}4883 \\ -0{,}1788 \\ 0{,}5475 \\ -0{,}7263 \end{pmatrix} \Longrightarrow \begin{cases} I_1 &= -0.18\,\text{A} \\ I_2 &= -0.67\,\text{A} \\ I_3 &= -0.73\,\text{A} \\ I_4 &= 0.44\,\text{A} \\ I_5 &= 0.49\,\text{A} \\ I_6 &= 1.11\,\text{A} \\ I_7 &= 0.55\,\text{A} \end{cases} \tag{23}$$

Efectivamente, se obtiene la misma solución aún tratándose de matrices complejas.

Coste computacional

El coste computacional en sumas y/o restas, y en productos y/o cocientes es el mismo que respecto al método de Crout y Doolittle. Ahora bien, al añadir raíces cuadradas como se muestra en la ecuación (21), tenemos que se añaden un total de $2 + (n+1)(n-2)/2$ raíces, esto es, un total de 17 raíces cuadradas.

4 Conclusiones

En esta PEC se ha resuelto un sistema de 7 ecuaciones con 7 incógnitas mediante 4 métodos diferentes, todos partiendo de una factorización de la matriz de coeficientes en dos matrices, una triangular inferior y una triangular superior. Se ha comprobado como, según el orden de filas y columnas con que se plantee el sistema, el coste computacional puede varíar completamente.

Además, se ha escrito código en distintos softwares, en el caso que nos ocupa, en C y Matlab, y se ha llegado a las mismas soluciones con una precisión de 10^{-5}.

Por último, cabe destacar que los métodos de factorización de matrices existían años antes de que se convirtiesen en un método tan eficiente, computacionalmente hablando, para resolver sistemas. Esta PEC no hace sino despertar una pregunta, ¿qué métodos matemáticos utilizaremos en las próximas décadas, incluso siglos, con aplicaciones que ahora mismo no somos capaces de concebir?

5 Listados

A continuación se presentan los listados de código con que se han obtenido los resultados de esta memoria. Se seguirá el orden expuesto en la Sección 2.

5.1 Funciones que crean archivos de datos

Estas funciones resultan muy útiles a la hora de trasladar datos de un lado a otro con facilidad y rapidez, sobretodo dado el alto número de decimales de las soluciones. A continuación se presentan los dos ejemplos más utilizados en esta PEC, `escribe_matriz_LaTeX_float` (listado 1) y `escribe_matriz_CSV_float` (listado 2). El nombre describe perfectamente su cometido: escribir una matriz de flotantes en formato LaTeX y en CSV. También he utilizado, en menor medida

`escribe_matriz_LaTeX_int` y `escribe_matriz_CSV_int`, iguales a las expuestas pero cambiando el "placeholder"para la escritura en el archivo.

Listado 1: *Función que escribe una matriz de flotantes en formato LaTeX*

```
1   void escribe_matriz_LaTeX_float(int filas, int columnas,
2                float M[filas][columnas], char archivo[100]){
3       int i,j;
4       FILE *archivodatos;
5       char tipo_matriz[100] = "pmatrix";
6
7       archivodatos = fopen(archivo, "w");
8       fprintf(archivodatos,"\\begin{%s}\n\t",tipo_matriz);
9       for (i=0;i<filas;i++){
10          for (j=0;j<columnas-1;j++){
11              fprintf(archivodatos,"%0.6g & ",M[i][j]);
12          }
13          fprintf(archivodatos,"%0.6g\\\\\n",M[i][j]);
14          if (i<filas-1) {fprintf(archivodatos,"\t");}
15      }
16      fprintf(archivodatos,"\\end{%s}",tipo_matriz);
17      fclose(archivodatos);
18  }
```

Listado 2: *Función que escribe una matriz de flotantes en formato CSV*

```
1   void escribe_matriz_CSV_float(int filas, int columnas,
2                float M[filas][columnas], char archivo[100]){
3       int i,j;
4       FILE *archivodatos;
5
6       archivodatos = fopen(archivo, "w");
7       for (i=0;i<filas;i++){
8           for (j=0;j<columnas-1;j++){
9               fprintf(archivodatos,"%f; ",M[i][j]);
10          }
11          fprintf(archivodatos,"%f\n",M[i][j]);
12      }
13      fclose(archivodatos);
14  }
```

En esta PEC se trabaja con matrices cuadradas, pero de todas formas se han escrito estas funciones para cualquier array de cualquier dimensión de `filas` filas y `columnas` columnas.

5.2 Permutación de filas

Este algoritmo permuta filas de una matriz de forma que la permutación final devuelva una matriz con elementos diagonales diferentes de cero. También permuta las filas del vector b de forma correspondiente. El algoritmo (listado 3) consta de tres bloques principales:

1. Se crean e inicializan la matriz C y el vector `secuencia`. La matriz C apunta para cada columna de la matriz de entrada M qué filas contienen elemento no nulo en esa columna. Cada fila que anota es candidata es candidata a ser permutada a la posición en fila que ocupa la columna, y, por tanto, es candidata a crear un elemento de diagonal en esa columna que sea distinto de cero. Como se puede observar, tanto el la C como `secuencia` se inicializan a -1. Esto es porque al estar escribiendo en C, el elemento 0 corresponde

al primer elemento. Para evitar cualquier confusión sabemos que el algoritmo, aunque puede decir que "hay un elemento en la fila 0", nunca va a clasificar una fila negativa.

2. Empezando por la columna 1 y para cada columna, buscamos entre las filas de C una secuencia de elementos que no se repitan, es decir,elementos que todavía no estén guardados en la secuencia. La secuencia nos da el nuevo orden de filas de la matriz que cumple que $M[i][i] \neq 0$. En el caso de que, recorriendo C, se encuentre un número repetido, se volverá a empezar desde la primera columna tomando otra fila candidata como origen.

3. Una vez obtenida la secuencia, se crea `M2` y `b2`, que no son mas que M y b con la secuencia de permutación aplicada.

Para el sistema matricial (6), se obtiene la siguiente matriz C (desplazando un elemento a la derecha los números obtenidos):

$$C = \begin{pmatrix} 1 & 1 & 2 & 3 & 1 & 3 & 2 \\ 2 & 3 & 6 & 7 & 4 & 5 & 4 \\ 4 & 5 & 0 & 0 & 5 & 7 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{24}$$

El algoritmo empieza por $C_{11} = 1$ y empieza a desplazarse hacia la derecha buscando un elemento no repetido. Verá que no puede elegir de nuevo el 1 de C_{12} y elegirá el 3 de C_{22}, y así sucesivamente.

No obstante, hay soluciones que el algorimto no encuentra. Pongamos, por ejemplo, que al algoritmo se le presenta la siguiente matriz C:

$$C = \begin{pmatrix} 1 & 3 & 3 & 5 \\ 2 & 4 & 0 & 0 \\ 4 & 0 & 0 & 0 \end{pmatrix} \tag{25}$$

dado que el 3 no aparece en la primera columna, el algoritmo siempre lo va a elegir, lo cual provocará que falle en la siguiente columna. No es capaz de ver que simplemente selecionando el 4 de la segunda columna ya sortea todos los problemas, no es capaz de mirar un paso hacia delante. Tampoco es capaz de volver un paso hacia atrás, que sería igual de válido. En el momento que se encuentra el 3 de la tercera columna empezando por el 1 de la primera, empezará de nuevo con el 2 de C_{12}, llegando al mismo resultado.

5.3 Resolución de los sistemas

Una vez se han obtenido las matrices L y U, la función con que estemos trabajando llama a `solver`, que simplemente consiste en sustituir hacia delante para despejar y de $Ly = b$ y en despejar hacia atrás para obtener x de $y = Ux$ (listado 5). En el apartado (e) se ha realizado en Matlab un código similar para esta función, que proporciona resultados idénticos.

5.4 Las función principal

Para ejecutar el programa, primero se debe definir la matriz A y el vector b en `main`, así como algunos arrays vacíos que se escribirán en otras funciones. A continuación escribimos el método que queremos utilizar, done 1 corresponde a la factorización LU convencional por Crout, 2 al método de Doolittle, y 3 al método de Crout para matrices tridiagonales. Con esto, ya podemos llamar a la función `solucionasistema`, que ejecuta todas las funciones mencionadas a lo largo de la sección de la forma que se muestra.

5.5 Funciones que factorizan matrices

A continuación se exponen las distintas funciones utilizadas para factorizar matrices a lo largo de la prueba.

LU por Crout

Este programa, utilizado para el Apartado (b), consta de los siguientes pasos:

1. Crea e inicializa las matrices `U` y `L`, y cumpliendo que la diagonal de `U` esté formada por unos.
2. Mediante diferentes fórmulas, calcula la primera fila de `U` y la primera columna de `L`.
3. Calcula los elementos de la diagonal de `L` mediante la expresión de la línea 34.
4. Calcula los elementos del i-ésimo renglón de `U` y la i-ésima columna de `L`.
5. Calcula el n-ésimo elemento de la diagonal de `L`.
6. Guarda las matrices en archivos de datos y llama a `solver` (listado !!!) para que resuelva el sistema con las matrices obtenidas.

Es importante hacer una aclaración sobre los límites de los bucles en C. Para mayor agilidad mental a la hora de leer los bucles, se han escrito los límites idénticos a los que aparecen en el libro base. Esto quiere decir que los contadores dentro del bucle estarán desplazados hacia atrás, dado que en lenguaje C se inician los contadores a cero.

LU por Doolittle

Este método es idéntico al método expuesto en la subsección anterior, haciendo que la diagonal de L esté formada por unos, con los cambios en las ecuaciones que se muestran en el listado 7.

LU para matrices tridiagonales

El listado 8 muestra el código necesario para LU-factorizar una matriz tridiagonal por Crout, con solución incorporada, a diferencia de los dos listados expuestos anteriormente. Básicamente consiste en seguir las pautas del algoritmo 6.7 de Burden y Faires 2011, en que se va calculando el vector $z = Ux$ al mismo tiempo que se van calculando los elementos de `L` y `U`, para finalmente realizar la sustitución hacia atrás que aisla las soluciones `x[i-1]`.

Algoritmo del apartado (e)

Debido a la falta de tiempo para crear funciones complejas en C, el apartado (e) ha sido escrito y resuelto en Matlab, como se muestra en el listado 9. En realidad, se trata de un algoritmo muy parecido al de la resolución por factorización LU mostrado en listados anteriores, pero incorporando la raíz cuadrada mostrada en la ecuación (21) ya que ahora `L(i,i)` = `U(i,i)`, escrito en Matlab.

Cabe destacar las líneas 21 y 22 de código, en las que en vez de utilizar un nuevo contador llamado `suma`, como ocurre en C cada vez que queremos realizar un sumatorio, ahora se pueden interpretar los productos matriciales mostrados en dichas líneas como un producto de un vector fila por un vector columna, esto es, un producto escalar, ya que `i` y `j` están fijados en cada iteración. Atajos como este hacen de Matlab un software más cómodo que C en ciertas ocasiones.

6 Bibliografía

Burden, Richard L. y J. Douglas Faires (2011). *Análisis numérico*. 9.ª ed. (vid. págs. 5, 8, 14).

Listado 3: *Función que permuta filas de matrices para hacerla LU-factorizable*

```
float permuta(int DIM, float M[DIM][DIM], float b[DIM],
        int secuencia[DIM], float M2[DIM][DIM], float b2[DIM]){

    int C[DIM][DIM];
    int i, j,k, n1;
    int solucion_encontrada;

    //--------------------------------------

    char archivo_CSV_C[100] = "matriz_C.csv";
    char archivo_CSV_secuencia[100] = "matriz_secuencia.csv";

    for(i=0;i<DIM;i++){
        for(j=0;j<DIM;j++){
            C[i][j] = -1;   //C[i][j]>=0, por tanto, inicializamos a -1.
        }
    }

    for(i=0;i<DIM;i++){
        secuencia[i] = -1;  //secuencia[i]>=0, por tanto, inicializamos a -1.
    }

    for(j=0;j<DIM;j++){
        k=0;
        for(i=0;i<DIM;i++){
            if(M[i][j]!=0){
                C[k][j]=i;
                k++;
            }
        }
    }

    escribe_matriz_CSV_int(DIM,DIM,C,archivo_CSV_C);

    // ----------------------------------------

    n1=0;

    for(i=0;i<DIM;i++){
        if (C[i][0]>=0){n1+=1;}
    }

    solucion_encontrada=0;
    k=0;
    while(solucion_encontrada==0 && k<n1){
        for(j=0;j<DIM;j++){
            if(j==0){i=k;}else{i=0;}
            while(se_repite(DIM, secuencia, C[i][j])){i++;}
            if(C[i][j]<0){break;}else{secuencia[j] = C[i][j];}
        }
        if(C[i][j]>=0){solucion_encontrada=1;}
        k++;
    }

    escribe_matriz_CSV_int(DIM,1,secuencia,archivo_CSV_secuencia);

    // ----------------------------------------

    for(i=0;i<DIM;i++){
        for(j=0;j<DIM;j++){
            M2[i][j] = M[secuencia[i]][j];
            b2[j] = b[secuencia[j]];
        }
    }
}
```

Listado 4: *Función que resuelve un sistema de ecuaciones ya LU-factorizado*

```
float solver(int DIM, float L[DIM][DIM], float U[DIM][DIM], float b[DIM]){

float y[DIM];
float suma;
int i, j;
float x[DIM];

for(i=0;i<DIM;i++){
x[i]=0;
y[i]=0;
}

y[0] = b[0]/L[0][0];

for(i=2;i<=DIM;i++){
suma=0;
for(j=1; j<=i-1; j++){
suma += L[i-1][j-1]*y[j-1];
}
y[i-1] = (1/L[i-1][i-1])*(b[i-1]-suma);
}
x[DIM-1] = y[DIM-1]/U[DIM-1][DIM-1];

for(i=DIM-1;i>=1; i--){
suma=0;
for(j=i+1; j<=DIM; j++){
suma += U[i-1][j-1]*x[j-1];
}
x[i-1] = (1/U[i-1][i-1])*(y[i-1] - suma);
}

char archivo_LaTeX_x[100] = "vector_x.tex";
char archivo_CSV_x[100] = "vector_x.csv";

char archivo_LaTeX_y[100] = "vector_y.tex";
char archivo_CSV_y[100] = "vector_y.csv";

escribe_matriz_LaTeX_float(DIM,1,y,archivo_LaTeX_y);
escribe_matriz_LaTeX_float(DIM,1,x,archivo_LaTeX_x);
escribe_matriz_CSV_float(DIM,1,y,archivo_CSV_y);
escribe_matriz_CSV_float(DIM,1,x,archivo_CSV_x);

}
```

Listado 5: *Función general que llama a las demás funciones, estructurando el trabajo realizado*

```
float solucionasistema(int DIM, float A[DIM][DIM], float b[DIM],
int secuencia[DIM], float A2[DIM][DIM], float b2[DIM], int metodo){

permuta(DIM,A,b,secuencia,A2,b2);

A=A2;   //trabajamos con la matriz permutada.
b=b2;   //trabajamos con el vector b permutado.

if(metodo==1){LU_crout(DIM,A,b);}
else if(metodo==2){LU_doolittle(DIM,A,b);}
else if (metodo==3){crout_tridiagonal_con_solver(DIM,A,b);}
}
```

Listado 6: *Función que LU-factoriza por el método de Crout*

```
float LU_crout(int DIM, float A[DIM][DIM], float b[DIM]){

    float U[DIM][DIM], L[DIM][DIM];
    float suma;
    int i,j,k;

    for(i=0; i<DIM; i++){
        for(j=0;j<DIM; j++){
            L[i][j]= 0;
            U[i][j]= 0;
        }
    }

    for(i=0; i<DIM; i++){
        U[i][i] = 1;
    }

    L[0][0] = A[0][0];

    for(j=2; j<=DIM; j++){
        U[0][j-1] = A[0][j-1]/L[0][0];
        L[j-1][0] = A[j-1][0]/U[0][0];
    }

    for(i=2; i<=DIM-1; i++){

        suma=0;
        for(k=1; k<=i-1; k++){
            suma += L[i-1][k-1]*U[k-1][i-1];
        }
        L[i-1][i-1] = A[i-1][i-1] - suma;

        for(j=i+1; j<=DIM; j++){

            suma = 0;
            for(k=1; k<=i-1; k++){
                suma += L[i-1][k-1]*U[k-1][j-1];
            }
            U[i-1][j-1] = (1/L[i-1][i-1])*(A[i-1][j-1] - suma);

            suma=0;
            for(k=1; k<=i-1; k++){
                suma += L[j-1][k-1]*U[k-1][i-1];
            }
            L[j-1][i-1] = (1/U[i-1][i-1])*(A[j-1][i-1] - suma);
        }

    }

    suma=0;
    for(k=1; k<=DIM-1; k++){
        suma += L[DIM-1][k-1]*U[k-1][DIM-1];
    }
    L[DIM-1][DIM-1] = A[DIM-1][DIM-1] - suma;

    char archivo_LaTeX_L[100] = "matriz_L.tex";
    char archivo_CSV_L[100] = "matriz_L.csv";
    char archivo_LaTeX_U[100] = "matriz_U.tex";
    char archivo_CSV_U[100] = "matriz_U.csv";

    escribe_matriz_LaTeX_float(DIM,DIM,L, archivo_LaTeX_L);
    escribe_matriz_LaTeX_float(DIM,DIM,U, archivo_LaTeX_U);
    escribe_matriz_CSV_float(DIM,DIM,L, archivo_CSV_L);
    escribe_matriz_CSV_float(DIM,DIM,U, archivo_CSV_U);

    solver(DIM, L, U, b);
    }
```

Listado 7: *Función que LU-factoriza por el método de Doolittle*

```
float LU_doolittle(int DIM, float A[DIM][DIM], float b[DIM]){

    float U[DIM][DIM], L[DIM][DIM];
    float suma;
    int i,j,k;

    for(i=0; i<DIM; i++){
    for(j=0;j<DIM; j++){
    L[i][j]= 0;
    U[i][j]= 0;
    }
    }

    for(i=0; i<DIM; i++){
    L[i][i] = 1;
    }

    U[0][0] = A[0][0];

    for(j=2; j<=DIM; j++){
    U[0][j-1] = A[0][j-1]/L[0][0];
    L[j-1][0] = A[j-1][0]/U[0][0];
    }

    for(i=2; i<=DIM-1; i++){

    suma=0;
    for(k=1; k<=i-1; k++){
    suma += L[i-1][k-1]*U[k-1][i-1];
    }
    U[i-1][i-1] = A[i-1][i-1] - suma;

    for(j=i+1; j<=DIM; j++){

    suma = 0;
    for(k=1; k<=i-1; k++){
    suma += L[i-1][k-1]*U[k-1][j-1];
    }
    U[i-1][j-1] = (1/L[i-1][i-1])*(A[i-1][j-1] - suma);

    suma=0;
    for(k=1; k<=i-1; k++){
    suma += L[j-1][k-1]*U[k-1][i-1];
    }
    L[j-1][i-1] = (1/U[i-1][i-1])*(A[j-1][i-1] - suma);
    }

    }

    suma=0;
    for(k=1; k<=DIM-1; k++){
    suma += L[DIM-1][k-1]*U[k-1][DIM-1];
    }
    U[DIM-1][DIM-1] = A[DIM-1][DIM-1] - suma;

    char archivo_LaTeX_L[100] = "matriz_L.tex";
    char archivo_CSV_L[100] = "matriz_L.csv";
    char archivo_LaTeX_U[100] = "matriz_U.tex";
    char archivo_CSV_U[100] = "matriz_U.csv";

    escribe_matriz_LaTeX_float(DIM,DIM,L, archivo_LaTeX_L);
    escribe_matriz_LaTeX_float(DIM,DIM,U, archivo_LaTeX_U);
    escribe_matriz_CSV_float(DIM,DIM,L, archivo_CSV_L);
    escribe_matriz_CSV_float(DIM,DIM,U, archivo_CSV_U);

    solver(DIM, L, U, b);
    }
```

Listado 8: *Función que resuelve por Crout un sitema con matriz de coeficientes tridiagonal*

```
float crout_tridiagonal_con_solver(int DIM, float A[DIM][DIM], float b[DIM]){

    float U[DIM][DIM], L[DIM][DIM];
    float x[DIM], z[DIM];
    int i,j,k;

    for(i=0; i<DIM; i++){
        for(j=0;j<DIM; j++){
            L[i][j]= 0;
            U[i][j]= 0;
        }
    }


    for(i=0; i<DIM; i++){
        U[i][i]= 1;
    }

    L[0][0] = A[0][0];
    U[0][1] = A[0][1]/L[0][0];
    z[0] = b[0]/L[0][0];

    for(i=2; i<=DIM-1; i++){
        L[i-1][i-2] = A[i-1][i-2];
        L[i-1][i-1] = A[i-1][i-1] - L[i-1][i-2]*U[i-2][i-1];
        U[i-1][i] = A[i-1][i]/L[i-1][i-1];
        z[i-1] = (b[i-1] - L[i-1][i-2]*z[i-2])/L[i-1][i-1];
    }

    L[DIM-1][DIM-2] = A[DIM-1][DIM-2];
    L[DIM-1][DIM-1] = A[DIM-1][DIM-1] - L[DIM-1][DIM-2]*U[DIM-2][DIM-1];
    z[DIM-1] = (b[DIM-1] - L[DIM-1][DIM-2]*z[DIM-2])/L[DIM-1][DIM-1];

    x[DIM-1] = z[DIM-1];

    for(i=DIM-1;i>=1;i--){
        x[i-1] = z[i-1] - U[i-1][i]*x[i];
    }

    char archivo_LaTeX_L[100] = "matriz_L.tex";
    char archivo_CSV_L[100] = "matriz_L.csv";

    char archivo_LaTeX_U[100] = "matriz_U.tex";
    char archivo_CSV_U[100] = "matriz_U.csv";

    escribe_matriz_LaTeX_float(DIM,DIM,L, archivo_LaTeX_L);
    escribe_matriz_LaTeX_float(DIM,DIM,U, archivo_LaTeX_U);
    escribe_matriz_CSV_float(DIM,DIM,L, archivo_CSV_L);
    escribe_matriz_CSV_float(DIM,DIM,U, archivo_CSV_U);

    char archivo_LaTeX_x[100] = "vector_x.tex";
    char archivo_CSV_x[100] = "vector_x.csv";

    char archivo_LaTeX_y[100] = "vector_z.tex";
    char archivo_CSV_y[100] = "vector_z.csv";

    escribe_matriz_LaTeX_float(DIM,1,z,archivo_LaTeX_y);
    escribe_matriz_LaTeX_float(DIM,1,x,archivo_LaTeX_x);
    escribe_matriz_CSV_float(DIM,1,z,archivo_CSV_y);
    escribe_matriz_CSV_float(DIM,1,x,archivo_CSV_x);
}
```

Listado 9: *Código Matlab que resuelve la factorización cumpliendo los requisitos del apartado (e)*

```
n = size(A,1);

% Inicialización de matrices L y U
L = zeros(n,n); U = zeros(n,n);

% Primer elemento de L y U
L(1,1) = sqrt(A(1,1));
U(1,1) = L(1,1);

% Primera fila de U y primera columna de L
for j = 2 : n
    U(1,j) = A(1,j)/L(1,1);
    L(j,1) = A(j,1)/U(1,1);
end

% Los demás elementos excepto el último
for i = 2 : n-1
    L(i,i) = sqrt(A(i,i) - L(i,1:i-1)*U(1:i-1,i));
    U(i,i) = L(i,i);
    for j = i + 1 : n
        U(i,j) = 1/L(i,i)*(A(i,j) - L(i,1:i-1)*U(1:i-1,j));
        L(j,i) = 1/U(i,i)*(A(j,i) - L(j,1:i-1)*U(1:i-1,i));
    end
end

% Último elemento de L y U
L(n,n) = sqrt(A(n,n) - L(n,1:n-1)*U(1:n-1,n));
U(n,n) = L(n,n);
```