Métodos Numéricos

Resolución de Sistemas Lineales de Ecuaciones

Roqueta, Matías Daniel

Centro Atómico Bariloche y Instituto Balseiro, Comisión Nacional de Energía Atómica

Resumen

Usando el lenguaje Octave se comparó el rendimiento de diferentes métodos numéricos para resolver sistemas lineales de ecuaciones de dimensiones altas.

Se estudiaron el método directo de factorización LU y el método iterativo de gradientes conjugados, especificando los casos de uso de cada método. Se estudió como escala rendimiento el método con la dimensión del problema, para que tipo de problema son favorables, y para que tipos de problemas dejan de ser prácticos.

Introducción

En casi toda aplicación de las matemáticas se presenta el problema de resolver un sistema lineal de n ecuaciones con n incógnitas, expresado de la forma matricial $Ax = b$ con $A \in \mathbb{R}^{n\times n}$ y $x,\, b \in \mathbb{R}^n$.

Es un problema de simple resolución para casos de dimensión chica, pero escala rápidamente en complejidad operativa con la dimensión del problema. Es en casos de dimensión mayor que se vuelve necesario el desarrollo de un algoritmo que sistematiza la resolución del problema usando herramientas computacionales de forma eficiente.

Una familia de estos algoritmos son los métodos directos, que encuentran el vector solución luego de realizar un número finito y determinado de pasos. Por su carácter determinista es de interés la complejidad algorítmica del método usando la notación O grande.

Un método directo es la factorización LU. Consiste en un algoritmo que encuentra los factores $A = LU$ con L triangular inferior y U triangular superior para separar un problema complejo en dos problemas simples

$$LUx = b \qquad \Longrightarrow \qquad Ly = b \qquad Ux = y \tag{1}$$

El algoritmo tiene el problema de que incluye divisiones por 0 siempre que la matriz A tiene un 0 en su diagonal principal. Esto se evita agregando una operación adicional de intercambio de filas previo a cada paso multiplicativo, llamada la operación de *pivot*.

$$P^{-1}LUx = b \qquad \Longrightarrow \qquad Ly = Pb \qquad Ux = y \tag{2}$$

Donde P es una matriz de 1's dispersos que contempla todos los intercambios de filas realizados.

El apéndice presenta dos algoritmos de factorización escritos en Octave, uno con pivot y otro sin el. Se puede ver que el orden de complejidad de ambos algoritmos es de $O(n^3)$.

Un problema de la factorización LU es que para ciertos tipos de matrices ralas, las matrices L y U tienen más elementos llenos que la matriz A. Octave implementa un nivel más de factorización con la función **lu** que intenta mitigar este efecto.[1]

$$A = P^{-1}LUQ^{-1} \tag{3}$$

Donde P y Q son matrices de pivot asociadas a L y U.

Otra familia de algoritmos son los métodos iterativos, estos métodos definen una sucesión de vectores $x^{[k]} \in \mathbb{R}^n$ que sea convergente a la solución, $x^{[k]} \longrightarrow x$.

Un ejemplo de método iterativo es el método de gradientes conjugados. Este método hace uso del funcional de un sistema lineal, definido para matrices simétricas

$$\Phi(x) = \tfrac{1}{2}\, x^T Ax - x^T b \tag{4}$$

Si A es además definida positiva, se puede ver que Φ describe una ecuación parabólica en $\mathbb{R}^n$. Al tomar gradiente de la ecuación 4 se obtiene

$$\nabla\Phi = Ax - b$$

Por lo que el vector x que resuelve el sistema es el que minimiza el funcional.

El método de gradientes conjugados consiste en minimizar Φ partiendo de un $x^{[0]}$ arbitrario e iterando

$$x^{[k+1]} = x^{[k]} + \alpha_k\, d^{[k]} \tag{5}$$

Con direcciones $d^{[k]}$ que forman un espacio *A-ortogonal* y coeficientes α_k que representan la distancia de $x^{[k]}$ al mínimo de Φ restringido a la correspondiente dirección.[2]

La A-ortogonalidad de los vectores $d^{[k]}$ implica que el método converge a la solución en no más de n pasos $\left(\forall k > n : d^{[k]} = 0\right)$, pero si se admite tolerancia de error

el método llega al resultado mucho antes. Que tanto antes depende del número de condición de la matriz, $\kappa(A)$.

$$\kappa(A) = \left\|A^{-1}\right\| \left\|A\right\| \tag{6}$$

Entre menor el número de condicionamiento, más rápida será la convergencia[1], esto conduce a que para matrices con números de condición altos sea más eficiente multiplicarlas por una matriz de precondicionamiento M^{-1} antes de iniciar el método iterativo.

La elección de M necesita equilibrar la disminución de iteraciones necesarias y el costo adicional en tiempo de las operaciones de precondicionamiento, una buena elección son las matrices LU incompletas, que son los factores LU de la matriz A, pero quitando los elementos en índices que corresponden a 0's de la matriz A.

La función **pcg** de Octave ejecuta el método de gradientes conjugados con o sin precondicionador, la factorización LU incompleta se obtiene de la función **ilu**.[1]

Una consideración importante para resolver sistemas de dimensiones altas es el uso de matrices ralas, o *sparse matrix*. Estas son estructuras de datos que solo almacenan los componentes $A_{ij} \neq 0$ y sobrecargan las operaciones entre matrices para omitir pasos triviales.

Resultados

Métodos Directos - Factorización LU

En la primera parte de la experiencia se compara el rendimiento del método de factorización LU con y sin pivot para resolver el problema $Ax = b$.

Un primer ensayo se realiza con matrices A y x con todos sus elementos aleatorios. Se determina $b = Ax$ para generar un problema de solución conocida y comparar esta solución con la obtenida por el método obteniendo así el error. El error se define con la norma 1

$$\varepsilon = \|x - x_m\|_1$$

Donde x es el vector solución conocido y x_m es el vector solución calculado por el método.

El sistema generado se resuelve primero con las dos subrutinas presentadas en el apéndice 1. Acto seguido se repite el proceso pero reemplazando el elemento $A_{11} = 10^{-12}$ para ver la susceptibilidad del método sin pivot a números cercanos a 0 en la diagonal principal.

Ya que este experimento tiene elementos aleatorios, en todos los casos se repite el proceso 5 veces y se registra la media de los errores.

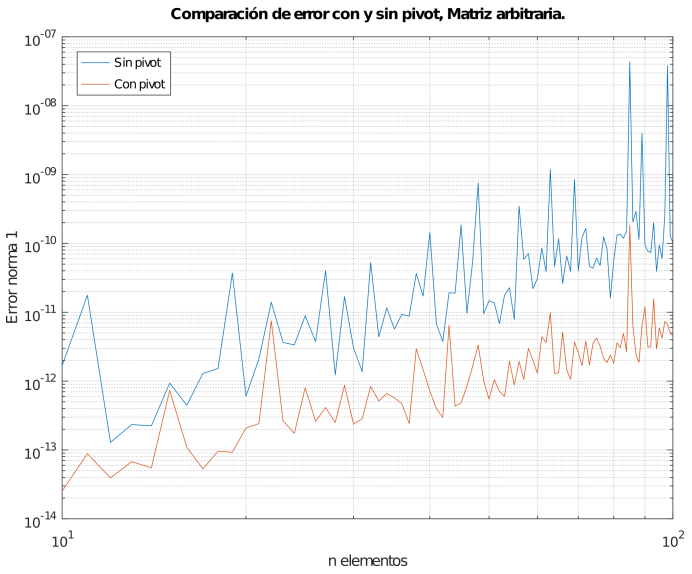

Figura 1: Error norma 1 del método directo de factorización LU para matrices de números aleatorios, en problemas de dimensión 10 a dimensión 100.

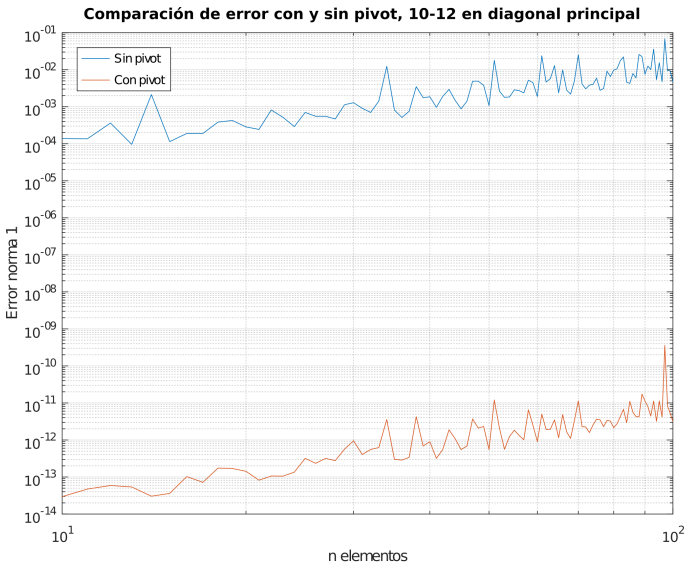

Figura 2: Error norma 1 del método directo de factorización LU para matrices de números aleatorios pero con $A_{11} \sim 0$, en problemas de dimensión 10 a dimensión 100.

Las figuras 1 y 2 muestran que el método con pivot es siempre más preciso que el que no lo usa.

Sin embargo para una matriz totalmente aleatoria la diferencia en precisión es de 10^{-11} contra 10^{-10} en di-

[1]Como caso límite, $\kappa(A) = 1$ implica que el método converge en la primera iteración, pero la única matriz con $\kappa = 1$ es la identidad y el problema era trivial desde un principio.

mensiones donde usar el método ya dejó de ser práctico.

Esto cambia cuando se agrega un número chico en la diagonal principal, el error del método con pivot se mantiene idéntico al caso anterior mientras que el del método sin pivot crece 10 órdenes de magnitud.

A continuación se compara la diferencia en tiempo de ejecución. Por ser métodos directos, el tiempo de ejecución es invariante ante los valores de los elementos de la matriz.

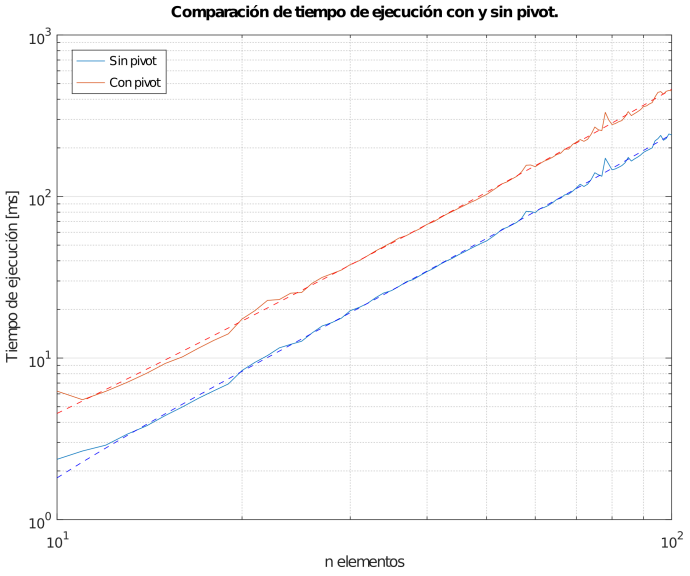

Figura 3: Tiempo de ejecución de métodos directos de factorización LU para resolver problemas de dimensión 10 a dimensión 100. En linea punteada se muestra un ajuste polinómico de grado 3, el orden de complejidad del algoritmo.

La figura 3 muestra tendencia a rectas paralelas en escala log-log. La complejidad del método con pivot es mayor, pero solo por un factor multiplicativo, ambos algoritmos son de orden de complejidad $O(n^3)$.

Métodos Iterativos - Gradientes Conjugados

Para estudiar el rendimiento de los métodos iterativos se usa una matriz deterministas que cumple los requisitos de simetría y definición positiva.

Se usa una matriz $A \in \mathbb{R}^{N\times N}$, con $N = (2n)^3$ y $n \in \mathbb{N}$ heptadiagonal con elementos definidos por

$$A_{ij} = \begin{vmatrix} 6 & si\ i = j \\ -1 & si\ |i-j| \in \{1,\ n,\ 2n^2\} \\ 0 & en\ otros\ casos \end{vmatrix} \quad (7)$$

Es una matriz rala proveniente de problemas físicos de difusión en tres dimensiones, y se puede ver que la dimensión del problema escala cúbicamente respecto a n.

Primero se estudia la necesidad de usar el almacenamiento ralo para problemas de esta magnitud, comparando el tiempo de ejecución de la función **pcg** de Octave sin precondicionador usando almacenamiento ralo y almacenamiento lleno.

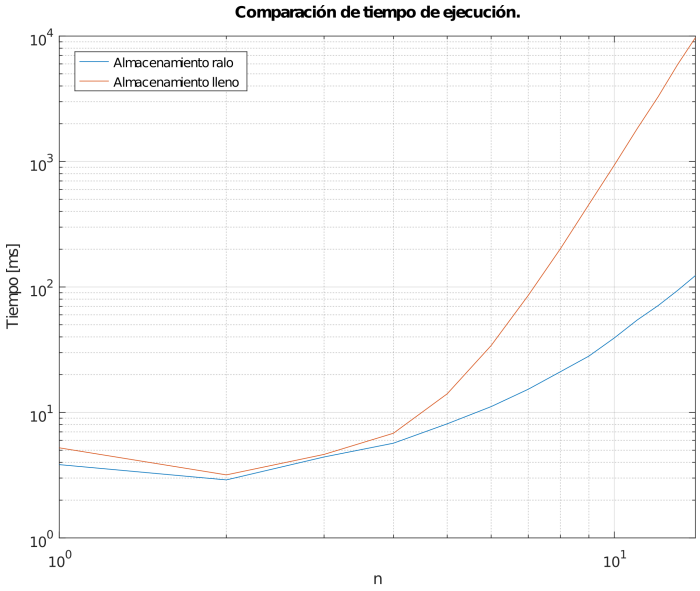

Figura 4: Tiempo de resolución del problema definido por la ecuación 7 por **pcg** sin precondicionador con n de 1 a 14 usando almacenamiento ralo y almacenamiento lleno.

En el siguiente experimento se compara efecto de precondicionar. Se registran tres tiempos, el tiempo ejecución del método **pcg** sin precondicionador, el tiempo de ejecución de **pcg** con matrices LU incompletas, y este mismo tiempo más el tiempo de factorización **ilu**.

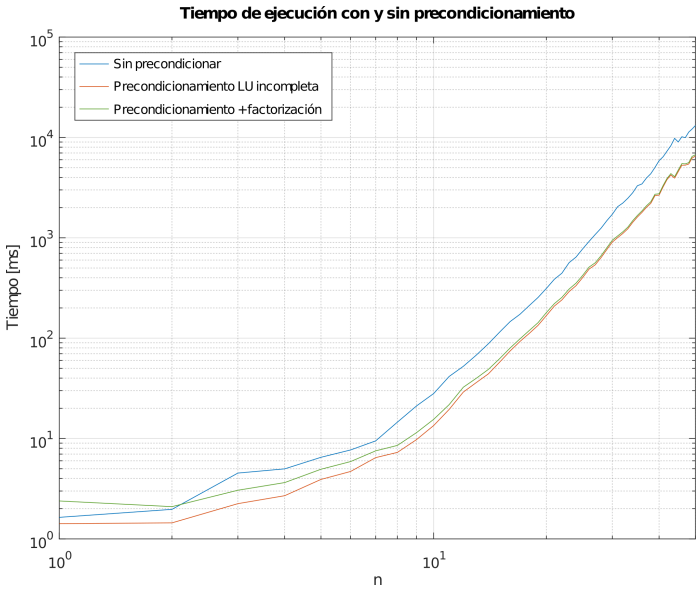

Figura 5: Tiempo de resolución del problema definido por la ecuación 7 por **pcg** sin precondicionador y precondicionado con **ilu** con n de 1 a 40.

Efectivamente el precondicionamiento acelera la con-

vergencia del método iterativo y eso se ve reflejado en el tiempo de ejecución, además la figura 5 muestra que por más que el tiempo de factorización **ilu** afecte a n chicos, se vuelve despreciable conforme n aumenta.

Para explicar la convergencia más rápida del método precondicionado hay que ver como es el número de condición de la matriz sobre la cual se itera. En la figura 6 se compara como evoluciona el número de condición con la dimensión del problema.

Calcular $\kappa(A)$ es un proceso algorítmicamente complejo, por lo que se limita el cálculo a n chicos.

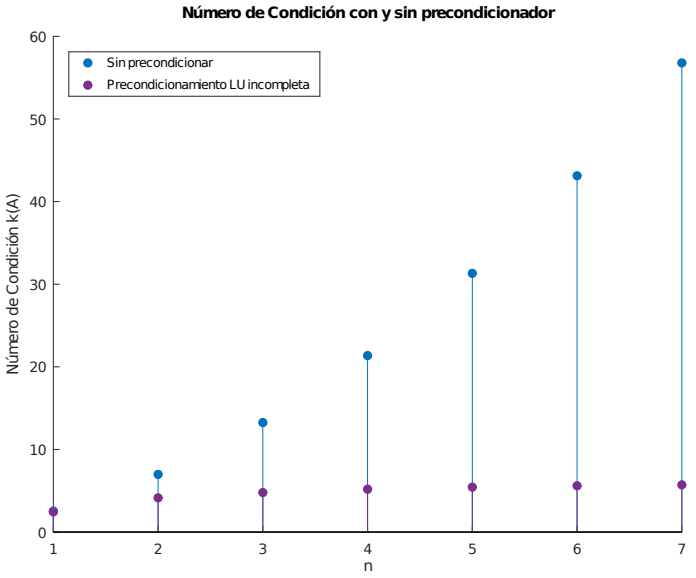

Figura 6: Comparación de números de condición de las matrices definidas por la ecuación 7 comparados con las mismas condicionadas por **ilu**.

La figura 6 muestra como este precondicionamiento logra, con un producto matricial no muy costoso, mantener un número de condicionamiento bajo mientras que el número de condición de A sin condicionar crece.

Métodos Directos contra Iterativos

Finalmente se compara el rendimiento para resolver el problema definido por la ecuación 7 de métodos iterativos contra métodos directos. Se usan matrices ralas y como método directo la factorización LU más eficiente para matrices ralas, invocando

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

En la figura 7 se ve que el método directo es en realidad más eficiente para n chicos, hasta un $n = 4$ pero incrementa mucho más rápido con n que cualquier método iterativo.

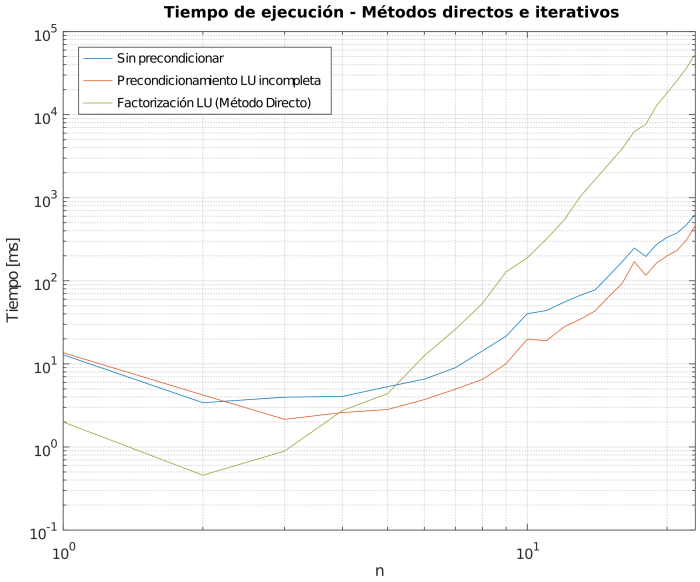

Figura 7: Comparación del tiempo de resolución del problema definido por la ecuación 7 por métodos directos e iterativos, con n de 1 a 22.

Para entender por que sucede esto conviene analizar las matrices generadas con **lu** y compararlas con las matrices generadas con **ilu**.

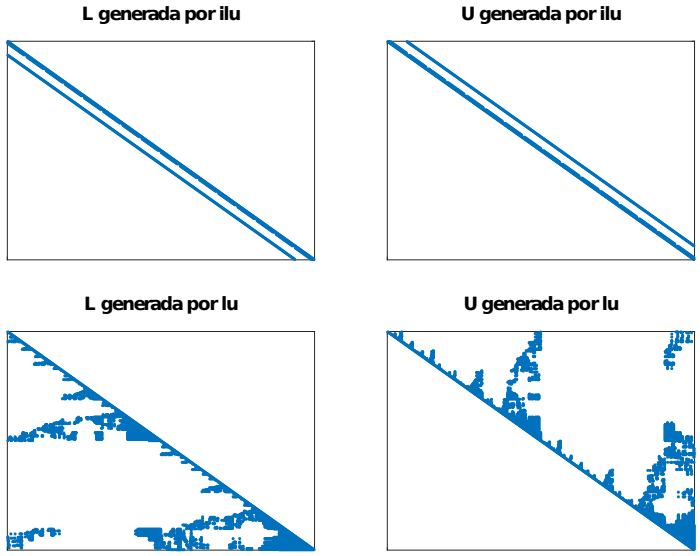

Figura 8: Estructura de elementos diferentes de 0 de matrices LU completas e incompletas. Por ser A simétrica, las matrices L y U tienen el mismo número de elementos.

La figura 8 ilustra cualitativamente la estructura de las matrices LU en el caso particular de $n = 4$. Las matrices LU generadas por esta invocación de **lu** son las de mínimo llenado posible, pero siguen teniendo cada una más elementos que A.

En la figura 9 se compara cuantitativamente como crece el número de elementos de L (o bien de U) en función de n haciendo uso de la función **nnz**.

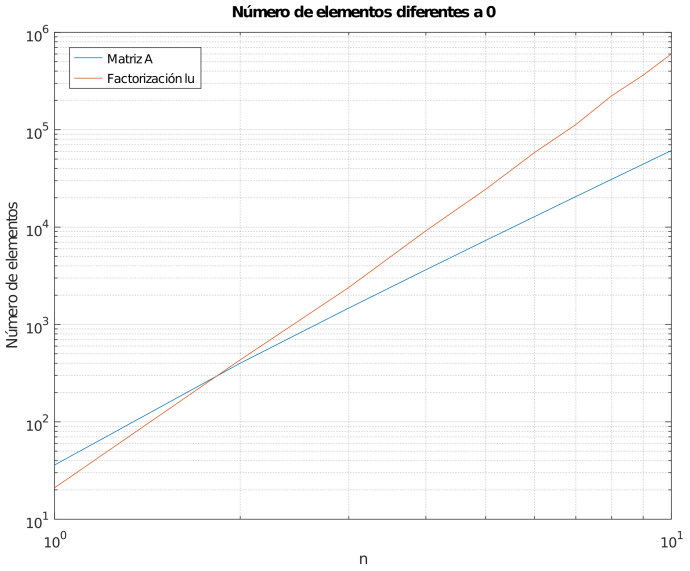

Figura 9: Número de elementos diferentes a 0 en los factores LU optimizados de la matriz definida por la ecuación 7 en función de n.

La ecuación 7 implica que el número de elementos diferentes a 0 de A incrementa cúbicamente con n, la figura 9 implica que el número de elementos diferentes a 0 de los factores LU de la misma incrementan según un polinomio de mayor orden, excediendo el número de elementos de A en $n = 2$.

Las operaciones con matrices ralas tienen complejidad algorítmica dependiente del número de elementos diferentes a 0, no de la dimensión de la matriz. Esto explica el rápido crecimiento en el tiempo de ejecución para el método directo, este es un problema en el que realizar una factorización LU completa, aún una optimizada, incrementa la complejidad algorítmica del problema.[2]

Conclusiones

Estos resultados no implican que los métodos directos sean ineficientes, la primera serie de experimentos muestra que funcionan bien para matrices de elementos arbitrarios, en particular cuando se le agrega un pivot que mantiene al resultado del método confiable en toda circunstancia sin incrementar el orden de complejidad.

Sin embargo, para problemas de mayor escala y donde la estructura de la matriz a resolver es conocida, el método directo deja de ser eficiente y se hace evidente la necesidad de usar un método iterativo, haciendo uso de las herramientas de matrices ralas.

Por más que aparente restrictiva la limitación del método de gradientes conjugados problemas definidos por matrices simétricas definidas positivas, muchos problemas en la práctica son de este tipo. Existen además generalizaciones del método que extienden su alcance a problemas que no cumplen esas condiciones.

Referencias

[1] Octave Forge, https://octave.sourceforge.io/docs.php, *Octave Forge Function Reference*, 2002 - 2008 ed.

[2] T. Carlone, "Overview of the conjugate gradient method," in *The Robotics Institute* (C. M. University, ed.), 2013.

[2] Es cierto que el método iterativo además acepta una tolerancia de error que no está presente en el método directo, pero en la introducción que el método de gradientes conjugados converge a la solución exacta en un número finito de iteraciones por lo que es también un método directo. La figura 7 sugiere que un método de gradientes conjugados como método directo es de menor orden de complejidad para este problema que una factorización LU.

Apéndice - Algoritmo de Factorización LU

En este apéndice se muestran los algoritmos de resolución de un sistema de ecuaciones lineales por factorización *LU* usado en la primera mitad de la experiencia, escritos en lenguaje Octave.

```
function x = LU_pivot(A, b)
 n = length(b);
 L = eye(n); U = A;
 for k = 1:n
  [_, p] = max(abs(U(k:n,k))); p=p+k-1;
  [U(p,k:n), U(k,k:n)] = deal(U(k,k:n), U(p,k:n));
  [b(p), b(k)] = deal(b(k), b(p));
  for j = 1:k-1
   [L(p,j), L(k,j)] = deal(L(k,j), L(p,j));
  end
  for i = k+1:n
   L(i,k) = U(i,k)/U(k,k);
   U(i,k:n) = U(i,k:n)-L(i,k)*U(k,k:n);
  end
 end
 x = LU_solve(A, b)
```

```
function x = LU_standard(A, b)
 n = length(b);
 L = eye(n); U = A;
 for k = 1:n
  for i = k+1:n
   L(i,k) = U(i,k)/U(k,k);
   U(i,k:n) = U(i,k:n)-L(i,k)*U(k,k:n);
  end
 end
 x = LU_solve(A, b)
```

```
function x = LU_solve(A, b)
 y = zeros(n,1);
 for k = 1:n
  y(k) = b(k);
  for i = 1:k-1
   y(k) = y(k) - y(i)*L(k,i);
  end
 end
 x = zeros(n,1);
 for k = n:-1:1
  x(k) = y(k);
  for i = k+1:n
   x(k) = x(k)-x(i)*U(k,i);
  end
  x(k)=x(k)/U(k,k);
 end
```