

MAT-468: Sesión 2, Solución de sistemas lineales

Felipe Osorio

<http://fosorios.mat.utfsm.cl>

Departamento de Matemática, UTFSM



El problema de resolver el **sistema lineal**

$$Ax = b,$$

es central en cálculo científico. Es bien sabido que:

- ▶ Si existe solución el sistema, entonces se dice **consistente**.
- ▶ Una solución x puede ser escrita como $A^{-1}b$, para A^{-1} alguna **inversa** de A .
- ▶ Si A es **cuadrada** entonces la solución es dada por $A^{-1}b$.

Nuestro objetivo es:

- ▶ Describir algunos métodos para **calcular una solución**.
- ▶ **Nunca** se obtendrá A^{-1} .



El problema de resolver el **sistema lineal**

$$Ax = b,$$

es central en cálculo científico. Es bien sabido que:

- ▶ Si existe solución el sistema, entonces se dice **consistente**.
- ▶ Una solución x puede ser escrita como $A^{-1}b$, para A^{-1} alguna **inversa** de A .
- ▶ Si A es **cuadrada** entonces la solución es dada por $A^{-1}b$.

Nuestro objetivo es:

- ▶ Describir algunos métodos para **calcular una solución**.
- ▶ **Nunca** se obtendrá A^{-1} .



El sistema **más simple** para resolver es del tipo

$$\mathbf{T}\mathbf{x} = \mathbf{b},$$

donde \mathbf{T} es **matriz triangular** $n \times n$, $\mathbf{x} \in \mathbb{R}^n$ y $\mathbf{b} \in \mathbb{R}^n$.

Suponga que \mathbf{T} es **matriz triangular superior**. En este caso debemos resolver el sistema:

$$\begin{pmatrix} t_{11} & t_{12} & \dots & t_{1n} \\ 0 & t_{22} & \dots & t_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & \dots & & t_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix},$$

utilizando **sustitución hacia atrás** (Asumiremos que $t_{ii} \neq 0$, $i = 1, \dots, n$).



Sustitución hacia atrás

Note que en el sistema de ecuaciones anterior podemos resolver la **última ecuación** de manera trivial:

$$x_n = b_n/t_{nn},$$

Reemplazando esta **ecuación en la anterior**, tenemos

$$x_{n-1} = \frac{1}{t_{n-1,n-1}} (b_{n-1} - t_{n-1,n} x_n)$$

Procediendo de ese modo, obtenemos la solución,

$$x_i = \frac{1}{t_{ii}} \left(b_i - \sum_{j=i+1}^n t_{ij} x_j \right),$$

para $i = 1, \dots, n$.

Observación: Note que, es posible sobrescribir la solución \mathbf{x} en el vector \mathbf{b} .



- ▶ El “sistema triangular” más simple surge cuando T es **diagonal**
- ▶ Cuando T es **triangular inferior**, el sistema es resuelto por **sustitución adelante**.
- ▶ El que T sea **triangular unitaria** ($t_{ii} = 1$) no añade dificultad al problema.
- ▶ La **inversa de una matriz triangular inferior** (superior) también es una matriz triangular inferior (superior).
- ▶ Lo anterior permite obtener la **inversa de una matriz triangular “in-place”**.



Resultado 1 (Factorización LU)

Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$ matriz cuadrada tal que todos sus cofactores principales son no nulos, es decir

$$a_{11} \neq 0, \quad \det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \neq 0, \quad \det \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \neq 0, \quad \dots \quad \det(\mathbf{A}) \neq 0.$$

Entonces, existe una única matriz triangular inferior unitaria \mathbf{L} y una matriz triangular superior \mathbf{U} , tal que $\mathbf{A} = \mathbf{L}\mathbf{U}$, y

$$\det(\mathbf{A}) = u_{11} u_{22} \cdots u_{nn}.$$



Algoritmo 1: Factorización LU

Entrada: Matriz $A \in \mathbb{R}^{n \times n}$.

Salida : Factores L y U , matrices triangulares inferior y superior, respectivamente.

```
1 begin
2   Hacer  $L = 0$  y  $U = I_n$ .
3   for  $i = 1$  to  $n$  do
4     for  $j = i$  to  $n$  do
5       |  $l_{ji} = a_{ji} - \sum_{k=1}^{i-1} l_{jk}u_{ki}$ 
6     end
7     for  $j = i + 1$  to  $n$  do
8       |  $u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj})/l_{ii}$ 
9     end
10  end
11 end
```



La principal utilidad de la **factorización LU** es resolver sistemas lineales del tipo

$$Ax = LUx = b,$$

mediante resolver los **sistemas triangulares**

$$Lz = b, \quad Ux = z,$$

que deben ser desarrollados via sustitución adelante y hacia atrás, respectivamente.

Observaciones:

- ▶ Hallar la factorización LU es equivalente a resolver un sistema lineal mediante eliminación gaussiana.
- ▶ Existen generalizaciones de LU para **matrices rectangulares** y **singulares**.



La principal utilidad de la **factorización LU** es resolver sistemas lineales del tipo

$$Ax = LUx = b,$$

mediante resolver los **sistemas triangulares**

$$Lz = b, \quad Ux = z,$$

que deben ser desarrollados via sustitución adelante y hacia atrás, respectivamente.

Observaciones:

- ▶ Hallar la factorización LU es equivalente a resolver un sistema lineal mediante **eliminación gaussiana**.
- ▶ Existen generalizaciones de LU para **matrices rectangulares** y **singulares**.



Resultado 2 (Factorización Cholesky)

Sea $A \in \mathbb{R}^{n \times n}$ es matriz **simétrica y definida positiva**, entonces existe una única matriz triangular superior $G \in \mathbb{R}^{n \times n}$ con elementos diagonales positivos tal que

$$A = G^T G$$

Observación

Note que si usamos la factorización Cholesky para resolver el sistema $Ax = b$. Entonces debemos resolver los sistemas triangulares

$$G^T y = b, \quad y \quad Gx = y.$$

En efecto,

$$Ax = (G^T G)x = G^T (Gx) = G^T y = b.$$



Resultado 2 (Factorización Cholesky)

Sea $A \in \mathbb{R}^{n \times n}$ es matriz simétrica y definida positiva, entonces existe una única matriz triangular superior $G \in \mathbb{R}^{n \times n}$ con elementos diagonales positivos tal que

$$A = G^T G$$

Observación

Note que si usamos la factorización Cholesky para resolver el sistema $Ax = b$. Entonces debemos resolver los sistemas triangulares

$$G^T y = b, \quad y \quad Gx = y.$$

En efecto,

$$Ax = (G^T G)x = G^T (Gx) = G^T y = b.$$



Algoritmo 2: Factorización Cholesky

Entrada: Matriz $A \in \mathbb{R}^{n \times n}$.

Salida : Factor Cholesky $T \in \mathbb{R}^{n \times n}$.

```
1 begin
2    $t_{11} = \sqrt{a_{11}}$ .
3   for  $j = 2$  to  $n$  do
4     |  $t_{1j} = a_{1j}/t_{11}$ .
5   end
6   for  $i = 2$  to  $n$  do
7     |  $t_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} t_{ki}^2}$ ,
8     | for  $j = i + 1$  to  $n$  do
9       | |  $t_{ij} = (a_{ij} - \sum_{k=1}^{i-1} t_{ki}t_{kj})/t_{ii}$ 
10    | end
11  end
12 end
```



Operador Sweep

El **operador Sweep** es un método alternativo para invertir matrices y resolver sistemas lineales.

Suponga que $\mathbf{A} = (a_{ij})$ es matriz cuadrada $n \times n$. Aplicando el operador Sweep sobre el k -ésimo elemento diagonal de \mathbf{A} ($a_{kk} \neq 0$) permite obtener la matriz \mathbf{B} , definida como:

$$\begin{aligned}b_{kk} &= \frac{1}{a_{kk}}, \\b_{ik} &= -\frac{a_{ik}}{a_{kk}}, \quad i \neq k, \\b_{kj} &= \frac{a_{kj}}{a_{kk}}, \quad j \neq k, \\b_{ij} &= a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}, \quad i, j \neq k,\end{aligned}$$

y escribimos $\mathbf{B} = \text{Sweep}(k)\mathbf{A}$.



Propiedades:

- ▶ $\text{Sweep}(k) \text{Sweep}(k) \mathbf{A} = \mathbf{A}$.
- ▶ $\text{Sweep}(k) \text{Sweep}(r) \mathbf{A} = \text{Sweep}(r) \text{Sweep}(k) \mathbf{A}$.
- ▶ $\mathbf{A}^{-1} = \prod_{i=1}^n \text{Sweep}(i) \mathbf{A}$.



Forma particionada del Sweep

Considere $\mathbf{A} \in \mathbb{R}^{n \times n}$ matriz particionada como:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix},$$

donde $\mathbf{A}_{11} \in \mathbb{R}^{r \times r}$ ($r < n$). Suponga que se aplica el operador Sweep sobre los elementos diagonales de \mathbf{A}_{11} . De este modo,

$$\mathbf{B} = \prod_{i=1}^r \text{Sweep}(i) \mathbf{A} = \begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{pmatrix},$$

con

$$\begin{aligned} \mathbf{B}_{11} &= \mathbf{A}_{11}^{-1}, & \mathbf{B}_{12} &= \mathbf{A}_{11}^{-1} \mathbf{A}_{12}, \\ \mathbf{B}_{21} &= -\mathbf{A}_{21} \mathbf{A}_{11}^{-1}, & \mathbf{B}_{22} &= \mathbf{A}_{22} - \mathbf{A}_{21} \mathbf{A}_{11}^{-1} \mathbf{A}_{12}. \end{aligned}$$



- ▶ Si A es matriz simétrica, el operador Sweep preserva la simetría de A .
- ▶ Existen varias definiciones ligeramente diferentes del operador Sweep.
- ▶ Problemas de inestabilidad pueden ocurrir cuando algún a_{kk} es cercano a cero.



Ideas:

- ▶ Es sabido que los métodos directos son bastante **sensibles a errores de redondeo**.
- ▶ El refinamiento iterativo permite **mejorar la solución obtenida** por un método directo.

Suponga una **solución aproximada** $x^{(0)}$ del sistema lineal $Ax = b$. El refinamiento iterativo es un proceso que construye la secuencia $\{x^{(k)}\}$, basada en el **vector de residuos**:

$$r^{(k)} = b - Ax^{(k)}, \quad r = 0, 1, \dots,$$

De este modo, resolviendo el sistema lineal $A\delta = r$, obtenemos que $x + \delta$ corresponde a una **solución del sistema original**.

El procedimiento es útil si somos capaces de calcular el vector de residuos con una **muy alta precisión**. Lo anterior motiva el siguiente algoritmo.



Ideas:

- ▶ Es sabido que los métodos directos son bastante **sensibles a errores de redondeo**.
- ▶ El refinamiento iterativo permite **mejorar la solución obtenida** por un método directo.

Suponga una **solución aproximada** $\mathbf{x}^{(0)}$ del sistema lineal $\mathbf{Ax} = \mathbf{b}$. El refinamiento iterativo es un proceso que construye la secuencia $\{\mathbf{x}^{(k)}\}$, basada en el **vector de residuos**:

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Ax}^{(k)}, \quad r = 0, 1, \dots,$$

De este modo, resolviendo el sistema lineal $\mathbf{A}\delta = \mathbf{r}$, obtenemos que $\mathbf{x} + \delta$ corresponde a una **solución del sistema original**.

El procedimiento es útil si somos capaces de calcular el vector de residuos con una **muy alta precisión**. Lo anterior motiva el siguiente algoritmo.



Ideas:

- ▶ Es sabido que los métodos directos son bastante **sensibles a errores de redondeo**.
- ▶ El refinamiento iterativo permite **mejorar la solución obtenida** por un método directo.

Suponga una **solución aproximada** $\mathbf{x}^{(0)}$ del sistema lineal $\mathbf{Ax} = \mathbf{b}$. El refinamiento iterativo es un proceso que construye la secuencia $\{\mathbf{x}^{(k)}\}$, basada en el **vector de residuos**:

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Ax}^{(k)}, \quad r = 0, 1, \dots,$$

De este modo, resolviendo el sistema lineal $\mathbf{A}\delta = \mathbf{r}$, obtenemos que $\mathbf{x} + \delta$ corresponde a una **solución del sistema original**.

El procedimiento es útil si somos capaces de calcular el vector de residuos con una **muy alta precisión**. Lo anterior motiva el siguiente algoritmo.



Algoritmo 3: Refinamiento iterativo.

Parámetros: Tolerancia τ , y número máximo de iteraciones k_{\max} .

```
1 begin
2    $k \leftarrow 0$ 
3   Calcular  $r^{(k)} = b - Ax^{(k)}$  en alta precisión
4   Resolver el sistema  $A\delta = r^{(k)}$  para obtener  $\delta^{(k)}$ 
5   Actualizar  $x^{(k+1)} = x^{(k)} + \delta^{(k)}$ 
6   if  $\|\delta^{(k)}\|_{\infty} \leq \tau \|x^{(k+1)}\|_{\infty}$  then
7     return  $x = x^{(k+1)}$ , y detener el algoritmo.
8   else if  $k < k_{\max}$  then
9     Hacer  $k \leftarrow k + 1$  y  $x^{(k)} \leftarrow x^{(k+1)}$ .
10    Volver a Paso 3.
11  else
12    Indicar: El algoritmo no converge después de  $k_{\max}$  iteraciones.
13  end
14 end
```



Algoritmo 4: Refinamiento iterativo.

Parámetros: Tolerancia τ , y número máximo de iteraciones k_{\max} .

```
1 begin
2    $k \leftarrow 0$ 
3   Calcular  $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$  en alta precisión
4   Resolver el sistema  $\mathbf{A}\boldsymbol{\delta} = \mathbf{r}^{(k)}$  para obtener  $\boldsymbol{\delta}^{(k)}$ 
5   Actualizar  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}^{(k)}$ 
6   if  $\|\boldsymbol{\delta}^{(k)}\|_{\infty} \leq \tau\|\mathbf{x}^{(k+1)}\|_{\infty}$  then
7     return  $\mathbf{x} = \mathbf{x}^{(k+1)}$ , y detener el algoritmo.
8   else if  $k < k_{\max}$  then
9     Hacer  $k \leftarrow k + 1$  y  $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(k+1)}$ .
10    Volver a Paso 3.
11  else
12    Indicar: El algoritmo no converge después de  $k_{\max}$  iteraciones.
13  end
14 end
```



Algoritmo 5: Refinamiento iterativo.

Parámetros: Tolerancia τ , y número máximo de iteraciones k_{\max} .

```
1 begin
2    $k \leftarrow 0$ 
3   Calcular  $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$  en alta precisión
4   Resolver el sistema  $\mathbf{A}\boldsymbol{\delta} = \mathbf{r}^{(k)}$  para obtener  $\boldsymbol{\delta}^{(k)}$ 
5   Actualizar  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}^{(k)}$ 
6   if  $\|\boldsymbol{\delta}^{(k)}\|_{\infty} \leq \tau \|\mathbf{x}^{(k+1)}\|_{\infty}$  then
7     return  $\mathbf{x} = \mathbf{x}^{(k+1)}$ , y detener el algoritmo.
8   else if  $k < k_{\max}$  then
9     Hacer  $k \leftarrow k + 1$  y  $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(k+1)}$ .
10    Volver a Paso 3.
11  else
12    Indicar: El algoritmo no converge después de  $k_{\max}$  iteraciones.
13  end
14 end
```



- ▶ Note que el algoritmo puede **no alcanzar convergencia**.
- ▶ El uso de refinamiento iterativo como un método general es **limitado por la alta precisión** en el paso 1.
- ▶ La factorización de A **es disponible** luego de calcular $x^{(0)}$.
- ▶ El **costo computacional** del refinamiento iterativo suele ser pequeño.



1. Escriba una rutina para resolver el sistema de ecuaciones lineales $Ax = b$ con:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & -1 & -1 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 9 \\ -1 \end{pmatrix}$$

usando la **descomposición LU** y **refinamiento iterativo**.

2. **Implementar el operador Sweep** usando su lenguaje de programación favorito.
3. Pruebe su rutina para **obtener la inversa** de una matriz usando el operador Sweep con la siguiente matriz:

$$B = \begin{pmatrix} 30 & 16 & 46 \\ 16 & 10 & 26 \\ 46 & 26 & 72 \end{pmatrix}.$$

4. Verifique que B es matriz semidefinida positiva usando la **factorización Cholesky**. ¿En cuál etapa del algoritmo el procedimiento falla?

