Sistemas de Ecuaciones Lineales I

- **Preliminares:** Expresión matricial. Dificultades numéricas.
- Estudiar diferentes métodos para resolver $\boldsymbol{Ax} = \boldsymbol{b}$.

Expresión matricial

- Todo sistema de ecuaciones lineales puede escribirse matricialmente:

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\ \vdots & \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n &= b_n \end{cases} \qquad \Longleftrightarrow \qquad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

donde

$$\boldsymbol{A} := \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n} \qquad \text{y} \qquad \boldsymbol{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^n$$

son los datos y $\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$ es el vector de incógnitas.

Objetivo: Estudiar diferentes métodos para resolver $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Entender de alguna forma lo que realiza el comando \.

Hasta el momento en este curso, para resolver sl sistema de ecuaciones

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

hemos utilizado el comando \ (*backslash*) de OCTAVE, es decir,

$$\boldsymbol{x} = \boldsymbol{A} \setminus \boldsymbol{b}.$$

Matriz inversa

- El sistema de ecuaciones lineales $\boldsymbol{Ax} = \boldsymbol{b}$ tiene solución única si y sólo si $\boldsymbol{A}$ es una matriz no singular.

  Recordemos que una matriz $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ es no singular si y sólo si se cumple cualquiera de estas condiciones:

  1. $\boldsymbol{A}$ es invertible: $\exists \boldsymbol{A}^{-1} \in \mathbb{R}^{n\times n} : \ \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$;
  2. $\det(\boldsymbol{A}) \neq 0$;
  3. todas las filas (y columnas) de $\boldsymbol{A}$ son l.i.: $\text{rango}(\boldsymbol{A}) = n$.
  4. 0 no es valor propio de $\boldsymbol{A}$: $0 \notin \sigma(\boldsymbol{A})$.

- Si $\boldsymbol{A}$ es no singular, entonces

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad \Longleftrightarrow \qquad \boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}.$$

  Sin embargo, en general, no es conveniente calcular la matriz inversa $\boldsymbol{A}^{-1}$ para resolver un sistema de ecuaciones, pues hacerlo así resulta mucho más costoso computacionalmente.

Matriz inversa (cont.)

- Una manera natural de calcular la inversa de una matriz $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ consiste en resolver n sistemas de ecuaciones lineales. Si llamamos $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^n$ a las columnas de $\boldsymbol{A}^{-1}$:

$$\boldsymbol{A}^{-1} = \left[\begin{array}{c|c|c} & & \\ \boldsymbol{c}^1 & \cdots & \boldsymbol{c}^n \\ & & \end{array}\right], \qquad \boldsymbol{c}^1, \ldots, \boldsymbol{c}^n \in \mathbb{R}^n,$$

entonces

$$\left[\begin{array}{c|c|c} & & \\ \boldsymbol{A}\boldsymbol{c}^1 & \cdots & \boldsymbol{A}\boldsymbol{c}^n \\ & & \end{array}\right] = \boldsymbol{A}\left[\begin{array}{c|c|c} & & \\ \boldsymbol{c}^1 & \cdots & \boldsymbol{c}^n \\ & & \end{array}\right] = \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I} = \left[\begin{array}{c|c|c} & & \\ \boldsymbol{e}^1 & \cdots & \boldsymbol{e}^n \\ & & \end{array}\right],$$

donde las columnas $\boldsymbol{e}^1, \ldots, \boldsymbol{e}^n$ de $\boldsymbol{I}$ son los vectores de la base canónica de $\mathbb{R}^n$. Por lo tanto, $\boldsymbol{A}^{-1}$ puede calcularse columna por columna resolviendo:

$$\boxed{\boldsymbol{A}\boldsymbol{c}^i = \boldsymbol{e}^i, \qquad i = 1, \ldots, n.}$$

Dificultades numéricas

Los siguientes aspectos deben tenerse en cuenta al diseñar un algoritmo para resolver un sistema de ecuaciones lineales:

1. **Costo operacional.**
2. **Costo de almacenamiento.**
3. **Precisión de los resultados.**

1. Costo operacional

El tiempo de cálculo del computador necesario para resolver el sistema debe ser lo menor posible. Una medida standard del costo operacional es la cantidad de operaciones aritméticas $(+,\ -,\ *,\ /)$ que requiere un algoritmo. Éste usualmente se expresa en flop *(floating point operations)*.

- **Mal ejemplo: Regla de Cramer.** Permite calcular explícitamente la solución de un sistema $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ mediante:

$$x_i = \frac{\det(\boldsymbol{A}_i)}{\det(\boldsymbol{A})}, \qquad i = 1, \ldots, n,$$

donde $\boldsymbol{A}_i$ es la matriz que se obtiene a partir de $\boldsymbol{A}$ reemplazándo en ésta su columna i-ésima por el segundo miembro $\boldsymbol{b}$.
Si los determinantes se calculan mediante la fórmula recursiva usual de desarrollo por fila (o por columna), el costo operacional de la Regla de Cramer es de aproximadamente $(n+1)!$ flop.

Costo operacional (continuación)

- **Buen ejemplo: Método de Eliminación Gaussiana.** Este procedimiento se basa en el método algebraico de **transformaciones elementales**. Su costo operacional veremos que es de aproximadamente $\frac{2}{3}n^3$ flop.
- **Comparación:**
  En un computador de 1 Gflop (10^9 flop) por segundo:

n	10	15	20	100	1000	2000
Regla de Cramer						
flop	4×10^7	2×10^{13}	5×10^{19}	10^{160}	"∞"	"∞"
tiempo	0.04 s	5.5 horas	1500 años	"∞"	"∞"	"∞"
Eliminación Gaussiana						
flop	666	2250	5333	7×10^5	7×10^8	5×10^9
tiempo	0. s	0. s	0. s	0 s	0.73 s	4.88 s

2. Costo de almacenamiento

Es la cantidad de posiciones de memoria que requiere el computador para ejecutar un algoritmo (representación de los datos, variables auxiliares, etc.).

Ejemplo. Creamos en OCTAVE dos matrices, una matriz $\mathbf{A}$ de 100×100 y otra matriz $\mathbf{B}$ de $10,000 \times 10,0000$, cuyos elementos son 1.

```
>> A = ones(100,100);
>> B = ones(10000,10000);
```

Costo de almacenamiento (continuación)

Con el comando whos podemos ver la cantidad de memoria utilizada por cada matriz:

```
whos
Variables in the current scope:

   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        A           100x100                  80000  double
        B         10000x10000              800000000  double
```

Observamos que la matriz $\mathbf{A}$ utiliza $80,000$ bytes ($0.08\,\text{MB}$) y la matriz $\mathbf{B}$ utiliza 8×10^8 bytes ($800\,\text{MB}$).

Costo de almacenamiento (continuación)

En muchas aplicaciones los sistemas de ecuaciones lineales que deben resolverse involucran matrices de gran tamaño, pero tales que la mayor parte de sus entradas son nulas. Estas matrices se denominan **dispersas** o **ralas** (en inglés y en OCTAVE, **sparse**) y existen técnicas para almacenarlas que sólo requieren una cantidad de posiciones de memoria aproximadamente igual al número de entradas no nulas de la matriz.

```
>> n = 7;
>> A = diag(ones(n-1,1),-1)+2*diag(ones(n,1))+diag(ones(n-1,1),1);
>> A
A =

   2   1   0   0   0   0
   1   2   1   0   0   0
   0   1   2   1   0   0
   0   0   1   2   1   0
   0   0   0   1   2   1
   0   0   0   0   1   2
```

Ejemplo de costo almacenamiento (continuación)

Creamos tambin la matriz `B=sparse(A)`. Los elementos de **A** y **B** son los mismos, pero la matriz **B** está siendo almacenada en formato sparse.

```
>> n = 10000;
>> A =  diag(ones(n-1,1),-1)+2*diag(ones(n,1))+diag(ones(n-1,1),1);
>> B = sparse(A);
>> whos
Variables in the current scope:

   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        A       10000x10000               800000000  double
        B       10000x10000                  559976  double
```

Para $n = 10,000$, vemos que el costo de almacenamiento de **A** es de 8×10^8 bytes (800 MB).
En cambio **B** utiliza sólo 5.59976×10^5 bytes (0.559976 MB).

3. Precisión de los resultados

- La resolución de un sistema de ecuaciones lineales en el computador involucra la propagación de **errores en los datos** y **errores de redondeo**. Por ello:

  1. Hay que disponer de alguna técnica que permita predecir cuando la resolución de un sistema de ecuaciones puede propagar drásticamente estos errores.

  2. Hay que diseñar métodos numéricos estables, que reduzcan la propagación de los errores de redondeo tanto como sea posible.

  3. Hay que diseñar técnicas computacionales que nos permitan, después de calcular la solución de un sistema de ecuaciones, estimar a posteriori la precisión de la solución calculada. Es decir, testear si el error con el que se la calculó está por debajo de una tolerancia aceptable.

Solución de sistemas con matriz triangular

- Dadas

$$\boldsymbol{L} = \begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \quad \text{y} \quad \boldsymbol{U} = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}$$

  decimos que $\boldsymbol{L}$ es triangular inferior y $\boldsymbol{U}$ es triangular superior.

- Dado que

$$\det(\boldsymbol{L}) = l_{11} l_{22} \cdots l_{nn} \quad \text{y} \quad \det(\boldsymbol{U}) = u_{11} u_{22} \cdots u_{nn},$$

  una matriz triangular es no singular si y sólo si sus términos diagonales son todos no nulos.

- La resolución de sistemas de ecuaciones lineales con matrices triangulares es muy sencilla y su costo operacional es bajo.

Solución de sistemas con matriz triangular inferior

Consideremos el siguiente sistema triangular inferior:

$$\underbrace{\begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}}_{\boldsymbol{L}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{pmatrix} b_1 \\ b2 \\ \vdots \\ b_n \end{pmatrix}}_{\boldsymbol{b}}$$

- Procedemos por **sustitución progresiva**, es decir, x_1 se despeja de la primera ecuación, x_2 de la segunda y así sucesivamente hasta llegar a x_n.

$$\begin{cases} l_{11}x_1 & = & b_1 & \Rightarrow & x_1 = b_1/l_{11} \\ l_{21}x_1 + l_{22}x_2 & = & b_2 & \Rightarrow & x_2 = (b_2 - l_{21}x_1)/l_{22} \\ \vdots & & \vdots & & \\ l_{n1}x_1 + \cdots + l_{nn}x_n & = & b_n & \Rightarrow & x_n = (b_n - l_{n1}x_1 - \cdots - l_{nn-1}x_{n-1})/l_{nn} \end{cases}$$

- Deducción del algoritmo:

Notemos que para $i = 1, ..., n$, se tiene que

$$\begin{aligned} x_i &= \left(b_i - l_{i1}x_1 - b_i - l_{i2}x_2 - \cdots - l_{i,i-1}x_{i-1}\right)/l_{ii} \\ &= \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right)/l_{ii}. \end{aligned}$$

Es decir, obtenemos el siguiente algoritmo

for $i = 1, \ldots, n$

$$\left| \quad x_i = \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right)\right.$$

end

Eliminación Gaussiana

El **método de eliminación gaussiana (M.E.G.)** consiste en reducir mediante **transformaciones elementales** un sistema $\boldsymbol{Ax} = \boldsymbol{b}$ a otro **equivalente** (es decir, que tenga la misma solución), de la forma

$$\boldsymbol{Ux} = \tilde{\boldsymbol{b}},$$

donde $\boldsymbol{U}$ es una matriz **triangular superior**. Luego, el sistema resultante se resuelve por el algoritmo descrito para matrices triangulares.

Además, con la ayuda de los multiplicadores, podemos formar la siguiente matriz triangular inferior

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix}.$$

Notemos que

$$\boldsymbol{LU} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & -3 & 2 \\ 0 & 2 & 3 \\ 0 & 0 & -12 \end{pmatrix} = \begin{pmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{pmatrix} = \boldsymbol{A}.$$

Lo anterior se conoce como factorización LU de la matriz $\boldsymbol{A}$.

Solución de sistemas mediante factorización LU:

Si $\boldsymbol{A}$ una matriz de $n \times n$ invertible.

1. Realizar la descomposición LU de la matriz $\boldsymbol{A}$. Entonces

$$\boldsymbol{Ax} = \boldsymbol{b} \iff \boldsymbol{L}(\boldsymbol{Ux}) = \boldsymbol{b} \iff \begin{cases} \boldsymbol{Ly} = \boldsymbol{b}, \\ \boldsymbol{Ux} = \boldsymbol{y}. \end{cases}$$

2. Resolver $\boldsymbol{Ly} = \boldsymbol{b}$ y, luego,
3. Resolver $\boldsymbol{Ux} = \boldsymbol{y}$.

Observaciones: Sea $\boldsymbol{A}$ una matriz de $n \times n$ invertible.

- el costo operacional del proceso de Eliminación Gaussiana es de

$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n \text{ flops.}$$

Si n es muy grande ($n \gg 1$), entonces el término que domina es $\frac{2}{3}n^3$, es decir,

$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n \approx \frac{2}{3}n^3 \text{ flops.}$$

- el costo operacional de resolver un sistema $\boldsymbol{Ax} = \boldsymbol{b}$ mediante factorización LU es:

1. Realizar la descomposición LU de la matriz $\boldsymbol{A}$:

$$\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n \text{ flops.}$$

2. Resolver $\boldsymbol{Ly} = \boldsymbol{b}$: n^2 flops.
3. Resolver $\boldsymbol{Ux} = \boldsymbol{y}$: : n^2 flops.
   Costo total:

$$\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n \text{ flops} \approx \frac{2}{3}n^3 \quad \text{si} \quad n \gg 1.$$

Estrategia de pivoteo parcial

Necesidad del pivoteo: Consideremos la matriz $\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & -1 & 1 \\ -5 & 2 & -3 \end{pmatrix}$ e intentemos obtener su factorización LU:

$$\boldsymbol{A} = \begin{pmatrix} \textcircled{0} & 1 & 1 \\ \textcircled{1} & -1 & 1 \\ \textcircled{-5} & 2 & -3 \end{pmatrix} \xrightarrow[F_3 - (m_{31})F_1]{F_2 - (m_{21})F_1} \longrightarrow$$

donde

$$m_{21} = \frac{-2}{0} \qquad \text{y} \qquad m_{31} = \frac{4}{0}.$$

Observamos que no podemos continuar debido a la división por cero causada porque el pivote es 0. Es decir, el algoritmo de eliminación gaussiana (o el de factorización LU) sólo puede llevarse a cabo **si todos los pivotes son no nulos.**

- Para poder resolver el sistema, debe **intercambiarse la primera fila con cualquiera de las otras de manera de evitar el pivote cero**. Por ejemplo, podemos intercambiar la primera con la tercera fila:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & -1 & 1 \\ -5 & 2 & -3 \end{pmatrix} \longrightarrow \begin{pmatrix} -5 & 2 & -3 \\ 1 & -1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

- Por otra parte, puede demostrarse que la **estabilidad** del método de eliminación gaussiana en cuanto a **propagación de errores de redondeo** se deteriora si los multiplicadores m_{ij} son números muy grandes en módulo.
- Una forma de evitar ambos inconvenientes, pivotes nulos y multiplicadores grandes en módulo, es realizar en cada paso el intercambio de ecuaciones que produzca el **pivote mayor posible en módulo**. Esto estrategia se denomina **pivoteo parcial**.