Curso de Métodos Numéricos

DEMAT, Universidad de Guanajuato

Clase 6: Introducción a la solución de sistemas de ecuaciones lineales

- Normas vectoriales y matriciales
- Sistemas con matrices triangulares
- Algoritmos de eliminación Gaussiana
- Pivoteo parcial y total
- Repaso de memoria dinámica

MAT–251

Dr. Joaquín Peña Acevedo
CIMAT A.C.
e-mail: joaquin@cimat.mx

Normas para vectores

Definición de norma vectorial

Una norma en $\mathbb{R}^n$ es una función $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ que tiene las propiedades:

1. $\|\boldsymbol{x}\| \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^n$,
2. $\|\boldsymbol{x}\| = 0$ si y sólo si $\boldsymbol{x} = 0$,
3. $\|\alpha \boldsymbol{x}\| = |\alpha| \, \|\boldsymbol{x}\| \quad \forall \boldsymbol{x} \in \mathbb{R}^n, \alpha \in \mathbb{R}$,
4. $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\| \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Ejemplos: Para $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\top$

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \qquad \text{(Norma 2)}$$

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i| \qquad \text{(Norma 1)}$$

$$\|\boldsymbol{x}\|_\infty = \max_{1 \leq i \leq n} |x_i| \qquad \text{(Norma infinito)}$$

Normas para vectores

En general, para $p \geq 1$, la norma p del vector $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ es

$$\|\boldsymbol{x}\|_p = \left(|x_1|^p + \cdots + |x_n|^p\right)^{1/p}.$$

Normas para matrices

Definición de norma matricial

Una norma en $\mathbb{R}^{n\times n}$ es una función $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$ que tiene las propiedades para todo $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ y $\alpha \in \mathbb{R}$:

1. $\|\boldsymbol{A}\| \geq 0$ y $\|\boldsymbol{A}\| = 0$ si y sólo si $\boldsymbol{A} = 0$,
2. $\|\alpha\boldsymbol{A}\| = |\alpha|\,\|\boldsymbol{A}\|$,
3. $\|\boldsymbol{A} + \boldsymbol{B}\| \leq \|\boldsymbol{A}\| + \|\boldsymbol{B}\|$,

Ejemplos: Para $\boldsymbol{A} = [a_{ij}] \in \mathbb{R}^{n\times n}$,

Norma de Frobenius :
$$\|\boldsymbol{A}\|_F = \left(\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2\right)^{1/2}$$

Norma natural inducida por la norma vectorial $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$, definida por

$$\|\boldsymbol{A}\| = \sup_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

Normas naturales

Una norma natural tiene las siguientes propiedades

- $\|\boldsymbol{I}\| = 1$.
- $\|\boldsymbol{Ax}\| \leq \|\boldsymbol{A}\| \, \|\boldsymbol{x}\|$.
- $\|\boldsymbol{AB}\| \leq \|\boldsymbol{A}\| \, \|\boldsymbol{B}\|$ (Propiedad de consistencia).

Normas naturales (I)

Otra alternativa para definir la norma natural es

$$\|\boldsymbol{A}\| = \sup_{\|x\|\neq 0} \|\boldsymbol{A}\boldsymbol{x}\| = \max_{\|x\|=1} \|\boldsymbol{A}\boldsymbol{x}\|.$$

Normas naturales (II)

Se puede ver que las normas naturales inducidas por las normas vectoriales $\|\cdot\|_1$ y $\|\cdot\|_\infty$ se pueden calcular mediantes las siguientes fórmulas:

Norma infinito: $$\|\boldsymbol{A}\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|$$

Normas naturales (III)

Norma 1: $$\|\boldsymbol{A}\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}|$$

Normas naturales (IV)

Proposición

Sea $\boldsymbol{A} = [a_{ij}]$ una matriz de tamaño $m \times n$. Entonces

$$\max_{ij} |a_{ij}| \leq \|\boldsymbol{A}\|_2 \leq \sqrt{mn} \max_{ij} |a_{ij}|$$

Normas matriciales consistentes (I)

Definición

Sean $\|\cdot\|_{ln}$, $\|\cdot\|_{lm}$ y $\|\cdot\|_{mn}$ normas en $\mathbb{R}^{l\times n}$, $\mathbb{R}^{l\times m}$ y $\mathbb{R}^{m\times n}$. Se dice que esas normas son *consistentes* si

$$\|\boldsymbol{AB}\|_{ln} \leq \|\boldsymbol{A}\|_{lm}\,\|\boldsymbol{B}\|_{mn}$$

para todo $\boldsymbol{A} \in \mathbb{R}^{l\times m}$ y $\boldsymbol{B} \in \mathbb{R}^{m\times n}$.

Definición

Sea $\|\cdot\|_\alpha$ una norma en $\mathbb{C}^m$, $\|\cdot\|_\beta$ una norma en $\mathbb{C}^n$ y $\|\cdot\|$ una norma matricial en $\mathbb{C}^{m\times n}$. Se dice que $\|\cdot\|$ es una norma *subordinada* a las normas $\|\cdot\|_\alpha$ y $\|\cdot\|_\beta$ si

$$\|\boldsymbol{Ax}\|_\alpha \leq \|\boldsymbol{A}\|\,\|\boldsymbol{x}\|_\beta \qquad \forall\, \boldsymbol{A} \in \mathbb{C}^{m\times n},\ \boldsymbol{x} \in \mathbb{C}^n.$$

Normas matriciales consistentes (II)

Por lo anterior, se ve que las normas naturales son subordinadas y consistentes.

Proposición

Si el producto $\boldsymbol{AB}$ está definido, entonces

$$\|\boldsymbol{AB}\|_F \leq \|\boldsymbol{A}\|_F \, \|\boldsymbol{B}\|_F.$$

Se puede ver que la norma del máximo no es consistente:

$$\|\boldsymbol{A}\|_{\max} = \max_{i,j} |a_{i,j}|.$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\|\boldsymbol{AA}\|_{\max} = \left\| \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \right\|_{\max} = 2 > 1 \cdot 1 = \|\boldsymbol{A}\|_{\max} \, \|\boldsymbol{A}\|_{\max}$$

comentario sobre las normas en $\mathbb{R}^n$

Proposición

Si $\|\cdot\|_a$ y $\|\cdot\|_b$ son dos norma[illegible] $\mathbb{R}^n$, entonces exi[illegible] constantes $\alpha, \beta \in \mathbb{R}$ tales que para todo $\boldsymbol{x} \in \mathbb{R}^n$

$$\alpha\|\boldsymbol{x}\|_[illegible] \quad \|\boldsymbol{x}\|_[illegible] \quad \beta\|\boldsymbol{x}\|_a$$

En general, cuando dos normas cum[illegible] las [illegible]ualdades anteriores se dice que son *equivalentes*.

Un comentario sobre las normas en $\mathbb{R}^n$

Proposición

Si $\|\cdot\|_a$ y $\|\cdot\|_b$ son dos normas en $\mathbb{R}^n$, entonces existen constantes $\alpha, \beta \in \mathbb{R}$ tales que para todo $\boldsymbol{x} \in \mathbb{R}^n$

$$\alpha\|\boldsymbol{x}\|_a \leq \|\boldsymbol{x}\|_b \leq \beta\|\boldsymbol{x}\|_a$$

En general, cuando dos normas cumplen las desigualdades anteriores se dice que son *equivalentes*.

Desde el punto de vista computacional, esto nos da la libertad de usar en los algoritmos una norma que no sea costosa de calcular y que no introduzca demasiados errores al calcularla.

Comparación entre normas vectoriales

$$\begin{aligned}
\|\boldsymbol{x}\|_2 &\leq \|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2 \\
\|\boldsymbol{x}\|_\infty &\leq \|\boldsymbol{x}\|_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty \\
\|\boldsymbol{x}\|_\infty &\leq \|\boldsymbol{x}\|_1 \leq n\,\|\boldsymbol{x}\|_\infty
\end{aligned}$$

Comparación entre normas matriciales

Para una matriz $\boldsymbol{A}$ de tamaño $m \times n$ se cumple:

$$\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_F \leq \sqrt{n}\|\boldsymbol{A}\|_2$$

$$\frac{1}{\sqrt{n}}\|\boldsymbol{A}\|_\infty \leq \|\boldsymbol{A}\|_2 \leq \sqrt{m}\|\boldsymbol{A}\|_\infty$$

$$\frac{1}{\sqrt{m}}\|\boldsymbol{A}\|_1 \leq \|\boldsymbol{A}\|_2 \leq \sqrt{n}\,\|\boldsymbol{A}\|_1$$

Sistemas de ecuaciones lineales (SEL)

Consideremos el caso en que tenemos tantas ecuaciones como incógnitas, de la forma

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \qquad i = 1, 2, \ldots, n.$$

Entonces podemos definir

$$\boldsymbol{A} \in \mathbb{R}^{n \times n}, \qquad \boldsymbol{b} \in \mathbb{R}^n.$$

$$\boldsymbol{A} = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Caso particular: Matriz diagonal

Consideremos el caso en que la matriz del sistema es una matriz diagonal,

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}.$$

Si $\boldsymbol{A}$ es invertible, entonces la solución del sistema $\boldsymbol{Ax} = \boldsymbol{b}$ es

$$\begin{aligned} x_1 &= \frac{b_1}{a_{11}}, \\ x_2 &= \frac{b_2}{a_{22}}, \\ &\vdots \\ x_n &= \frac{b_n}{a_{nn}}. \end{aligned}$$

Caso particular: Matriz triangular inferior (I)

Consideremos el sistema

$$\boldsymbol{Lx} = \boldsymbol{b}, \qquad \text{donde} \qquad \boldsymbol{L} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}$$

Proposición.

Sea $\boldsymbol{L}$ una matriz triangular inferior. El sistema $\boldsymbol{Lx} = \boldsymbol{b}$ tiene solución si y sólo si los elementos de su diagonal son diferentes de cero.

La solución se puede calcular mediante *sustitución hacia adelante*:

Caso particular: Matriz triangular inferior (II)

$$
\begin{aligned}
x_1 &= \frac{b_1}{l_{11}}, \\
x_2 &= \frac{b_2 - l_{21}x_1}{l_{22}}, \\
&\vdots \\
x_i &= \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right) \qquad i = 1, 2, \ldots, n.
\end{aligned}
$$

- Como el sistema $\boldsymbol{Lx} = \boldsymbol{b}$ tiene solución única para cada $\boldsymbol{b}$, la matriz $\boldsymbol{L}$ es no singular.

Caso particular: Matriz triangular superior (I)

Consideremos el sistema

$$\boldsymbol{Ux} = \boldsymbol{b}, \qquad \text{donde} \qquad \boldsymbol{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Proposición.

Sea $\boldsymbol{U}$ una matriz triangular superior. El sistema $\boldsymbol{Ux} = \boldsymbol{b}$ tiene solución si y sólo si los elementos de su diagonal son diferentes de cero.

La solución se puede calcular mediante *sustitución hacia atrás*:

Caso particular: Matriz triangular superior (II)

$$
\begin{aligned}
x_n &= \frac{b_n}{u_{nn}}, \\
x_{n-1} &= \frac{b_{n-1} - u_{n-1,n} x_n}{u_{n-2,n-2}}, \\
&\vdots \\
x_i &= \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \qquad i = n, n-1, \ldots, 1.
\end{aligned}
$$

Eliminación Gaussiana (I)

- El algoritmo está basado en aplicar operaciones elementales de fila a la matriz aumentada $[\,\boldsymbol{A} \mid \boldsymbol{b}\,]$.
- Como primer paso, se usa la primera ecuación para producir $n-1$ ceros como coeficientes de la variable x_1 en todas la ecuaciones, excepto en la primera.

  La primera ecuación se dice es la *ecuación de pivoteo* y al elemento a_{11} se le llama el *pivote*.

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \longrightarrow \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n2} & \cdots & a'_{nn} & b'_n \end{array}\right]$$

- Los datos originales se sobreescriben con los nuevos valores calculados.
- Para $i = 2, 3, \ldots, n$, se hace

Eliminación Gaussiana (II)

$$
\begin{aligned}
a_{ij} &\longleftarrow a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} \qquad (1 \leq j \leq n) \\
b_i &\longleftarrow b_i - \frac{a_{i1}}{a_{11}} b_1
\end{aligned}
$$

- La primera ecuación ya no se modifica.
- El proceso se repite usando la segunda ecuación como ecuación de pivoteo, con a_{22} como pivote.
- En general, cuando a_{ll} es el pivote, se aplican las siguientes operaciones

Eliminación Gaussiana (III)

Para $i = l+1, \ldots, n$, se hace

$$a_{ij} \longleftarrow a_{ij} - \frac{a_{il}}{a_{ll}} a_{lj} \qquad (l+1 \leq j \leq n)$$

$$b_i \longleftarrow b_i - \frac{a_{il}}{a_{ll}} b_l$$

Hay que tener cuidado de que los pivotes no sean cero.
Al final se obtiene una matriz de la forma

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ 0 & 0 & a_{33} & \cdots & a_{3n} & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} & b_n \end{array}\right]$$

El cual corresponde a un sistema con una matriz triangular superior.

Necesidad de pivoteo (I)

Consideremos el sistema

$$\begin{array}{rcrcrcr} x & + & y & + & 2z & = & 9 \\ x & + & y & + & 3z & = & 12 \\ 2x & + & 3y & + & z & = & 11 \end{array}$$

El sistema tiene solución $x = 1, y = 2, z = 3$.
Si aplicamos el algoritmo de eliminación Gaussiana, obtenemos

$$\begin{array}{rcrcrcr} x & + & y & + & 2z & = & 9 \\ 0 & + & 0 & + & z & = & 3 \\ 0 & + & y & - & 3z & = & -7 \end{array}$$

Tenemos el problema de que no podemos usar el coeficiente de y en la segunda ecuación como pivote.

Otro problema se ve en el siguiente caso:

$$\begin{array}{rcrcrcr} x & + & y & + & 2z & = & 9 \\ x & + & 1.0000001y & + & 3z & = & 12.0000002 \\ 2x & + & 3y & + & z & = & 11 \end{array}$$

Necesidad de pivoteo (II)

Este sistema tiene la misma solución que el sistema anterior. Al aplicar el algoritmo obtenemos

$$\begin{array}{rcrcrcr} x & + & y & + & 2z & = & 9 \\ 0 & + & 0.0000001y & + & z & = & 3.0000002 \\ 0 & + & y & - & 3z & = & -7 \end{array}$$

Podemos tener problemas si usamos 0.0000001 como pivote.

- Se tiene que elegir el pivote de modo que no sea cero o demasiado pequeño.
- La selección de la ecuación de la cual tomar el pivote se llama *pivoteo*.

Pivoteo parcial (I)

- El intercambio de filas de la matriz es una operación elemental que no cambia la solución del problema.
- Entonces antes de aplicar el algoritmo, buscamos en la primera columna el índice

$$i_1 = \arg\max_{1 \le i \le n} |a_{i1}|$$

- Hacemos el intercambio de la fila 1 con la fila i_1.

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{i_1 1} & a_{i_1 2} & \cdots & a_{i_1 n} & b_{i_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \longrightarrow \left[\begin{array}{cccc|c} a_{i_1 1} & a_{i_1 2} & \cdots & a_{i_1 n} & b_{i_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right]$$

- Se aplica el algoritmo de eliminación Gaussiana para la primera columna.

voteo parcial (II)

- Después, se determina el índice

$$i_2 = \arg\max_{2 \le i \le n} |a_{i2}|$$

  e intercambiamos las filas 2 e i_2.
- Continuamos de esta manera.

Pivoteo total (I)

Considere el sistema de ecuaciones

$$\begin{aligned} 2x &+ 2y &+ 4z &= 18 \\ x &+ 1.0000001y &+ 3z &= 12.0000002 \\ 0.00000002x &+ 0.00000003y &+ 0.00000001z &= 0.00000011 \end{aligned}$$

que se obtiene del sistema de ecuaciones anterior y, por tanto, tiene la misma solución.
Después aplicar el primer paso del algoritmo, obtenemos el sistema aumentado

$$\left[\begin{array}{lll|l} 2.0 & 2.00000000 & 4.00000000 & 18.00000000 \\ 0.0 & 0.00000010 & 1.00000000 & 3.00000020 \\ 0.0 & 0.00000001 & -0.00000003 & -0.00000007 \end{array}\right]$$

Para el siguiente paso, se toma a 0.0000001 como pivote, por lo que podemos tener problemas.
Para evitarlos, hay que usar pivoteo total, en el cual se elige el pivote inicial como

$$(i_1, j_1) = \arg \max_{1 \le i,j \le n} |a_{ij}|$$

Pivoteo total (II)

Hay que intercambiar las filas 1 e i_1, y reordenar las variables de modo que se intercambien las columnas 1 y j_1.

$$\left[\begin{array}{ccccc|c} a_{11} & \cdots & a_{1j_1} & \cdots & a_{1n} & b_1 \\ \vdots & \cdots & \vdots & \ddots & \vdots & \vdots \\ a_{i_1 1} & \cdots & a_{i_1 j_1} & \cdots & a_{i_1 n} & b_{i_1} \\ \vdots & \cdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nj_1} & \cdots & a_{nn} & b_n \end{array}\right] \longrightarrow \left[\begin{array}{ccccc|c} a_{i_1 1} & \cdots & a_{i_1 j_1} & \cdots & a_{i_1 n} & b_{i_1} \\ \vdots & \cdots & \vdots & \ddots & \vdots & \vdots \\ a_{11} & \cdots & a_{1j_1} & \cdots & a_{1n} & b_1 \\ \vdots & \cdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \cdots & a_{nj_1} & \cdots & a_{nn} & b_n \end{array}\right]$$

$$\longrightarrow \left[\begin{array}{ccccc|c} a_{i_1 j_1} & \cdots & a_{i_1 1} & \cdots & a_{i_1 n} & b_{i_1} \\ \vdots & \cdots & \vdots & \ddots & \vdots & \vdots \\ a_{1j_1} & \cdots & a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & \cdots & \vdots & \ddots & \vdots & \vdots \\ a_{nj_1} & \cdots & a_{n1} & \cdots & a_{nn} & b_n \end{array}\right]$$

- En este caso el orden de las variables cambia.
- Es más costoso computacionalmente que el pivoteo parcial.

Observación

- El método de eliminación Gaussiana con pivoteo parcial es un método que indica cuando el sistema

$$\boldsymbol{Ax} = \boldsymbol{b}$$

es consistente o no.

- En el caso en que el sistema es consistente, también el método indica si hay solución única o no.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1,n-1} & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2,n-1} & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3,n-1} & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & a_{nn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

Complejidad del algoritmo

Sea $\boldsymbol{M} = [\boldsymbol{A}\,|\,\boldsymbol{b}]$. Supongamos que no hace falta intercambiar filas, entonces el algoritmo de eliminación Gaussiana es

```
for i=1:(n-1)
    for k=(i+1):n
        alpha = -M(k,i)/M(i,i);
        for j=i:(n+1),
            M(k,j) = M(k,j) + alpha*M(i, j);
        end
    end
end
```

Si contamos el número de operaciones vemos que el algoritmo es de orden

$$O(n^3)$$

Memoria para arreglos 1D

- Revisar el código *memoria1D.cpp* para ver como crear arreglos 1D.
- En el programa se calcula el tiempo que demora hacer la suma de los elementos dos arreglos usando dos formas diferentes de accesar a los elementos del arreglo.

Memoria para arreglos 2D

Para asignar memoria a un arreglo 2D de tamaño para almacenar la información de una matriz $m \times n$ hacemos lo siguiente:

1. Creamos de manera dinámica un arreglo de apuntadores. Tantos apuntadores como filas tenga el arreglo (m).
2. Creamos un bloque de memoria de tamaño mn para almacenar todas las entradas.
3. Hacemos que el primer apuntador apunte a esta posición de memoria.
4. Hacemos que el resto de los apuntadores apunten a las posiciones del bloque de memoria reservada en donde empiezan cada fila de la matriz.

Paso 1: Creación del arreglo de apuntadores

```
double **createMatrix(int nr, int nc) {
    int i;
    double **mat;

    // Reservamos memoria
    mat = (double **) malloc( (nr)*sizeof(double *));
    if(mat==NULL) return(NULL);
    mat[0] = (double *) malloc( nr*nc*sizeof(double));
    if(mat[0]==NULL) return(NULL);
    for(i=1; i<nr; ++i)
        mat[i] = mat[i-1] + nc;
    return(mat);
}
```

Ahora queremos que `mat[i]` apunte a la fila correspondiente. Primero debemos reservar toda la memoria que vamos a necesitar y usar hacer que `mat[0]` apunte a ésta.

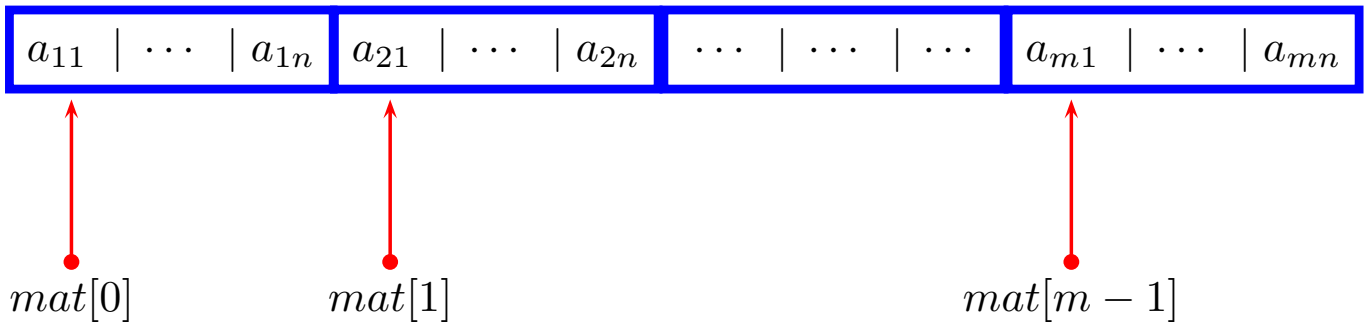
Pasos 2,3 y 4: Inicialización de los apuntadores

```
double **createMatrix(int nr, int nc) {
    int i;
    double **mat;

    // Reservamos memoria
    mat = (double **) malloc( (nr)*sizeof(double *));
    if(mat==NULL) return(NULL);
    mat[0] = (double *) malloc( nr*nc*sizeof(double));
    if(mat[0]==NULL) return(NULL);
    for(i=1; i<nr; ++i)
        mat[i] = mat[i-1] + nc;
    return(mat);
}
```

`mat[0]` apunta al inicio del bloque de memoria de tamaño `nr*nc`. Cada apuntador `mat[i]` hace referencia a una localidad de memoria diferente.

Pasos 2, 3 y 4: Inicialización de los apuntadores



En el código, la primera línea crea el bloque de memoria para almacenar todos los elementos de la matriz (el espacio representado en azul en la figura) y la asigna a `mat[0]`.
El ciclo los inicializa el resto de los apuntadores a las posiciones en donde debe empezar cada fila.

Para $i = 0, \ldots, nr-1$ y $j = 0, \ldots, nc-1$, cada variable `mat[i][j]` sirve para almacenar el elemento en la columna $j+1$ de la fila $i+1$ de la matriz $\boldsymbol{A}$.

Liberación de memoria

Para liberar la memoria asignada a la matriz, primero liberamos la memoria del bloque creado para almacenar todos los elementos, y luego se libera la memoria que solicitamos para el arreglo de apuntadores:

```
void freeMatrix(double **mat) {
    free(mat[0]);
    free(mat);
}
```

Ejemplos

- Revisar el código *memoria1D.c* para ver como crear arreglos 1D, y realizar operaciones entre ellos.
- Revisar el código *productoMatrizVector.c* para ver como la manera de crear um arreglo 2D para construir una matriz y un arreglo 1D, y calcular el productor entre la matriz y el vector.