TEMA 5:

Sistemas de Ecuaciones Lineales

Sistemas de Ecuaciones Lineales

Nuestro objetivo es la resolución de sistemas lineales

$$\begin{cases} 2x - y - z = 1 \\ -x + 2y - 3z = 0 \\ 3x - 3y + 4z = 2 \end{cases} \xrightarrow[A\cdot\underline{x} = \underline{b}]{\text{Matricialmente}} \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -3 \\ 3 & -3 & 4 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

Aquí tenemos muy claro cuando hay solución o no y como hallarla.

Lo que nos interesa es encontrar **métodos eficaces** de resolución.

Esquema: 1) Repaso conceptos de Álgebra Lineal.

2) Métodos directos.

3) Consideraciones sobre errores

REPASO ALGEBRA: Matrices especiales

Diagonal: $D_{ij} = 0 \quad para \quad i \neq j$

Identidad: $D_{ij} = \delta_j^i = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

Triangular Superior (ceros por debajo de diagonal) $U_{ij} = 0 \quad \text{para} \quad i > j$

Triangular Inferior (ceros por encima de diagonal) $L_{ij} = 0 \quad \text{para} \quad i < j$

Ortogonales: (matrices de giro) inversa = transpuesta $Q^{-1} = Q^T$

Singular: $\begin{cases} \det(A) = 0 \\ A\bar{x} = 0 \quad para \quad \bar{x} \neq 0 \\ A\bar{x}_1 = A\bar{x}_2 \quad para \quad \bar{x}_1 \neq \bar{x}_2 \end{cases}$

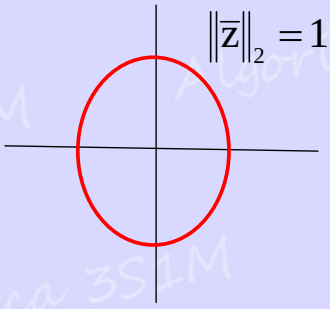
Normas de Vectores

Norma de un vector (medida de su tamaño):

$$\|\bar{x}\|_1 = \sum_k |x_k|, \quad \|\bar{x}\|_2 = \left(\sum_k |x_k|^2 \right)^{1/2} \quad ... \quad \|\bar{x}\|_\infty = \max\{|x_k|\}$$

Caso 2D. Sea un vector z de componentes (x,y) $\bar{z} = \begin{pmatrix} x \\ y \end{pmatrix}$

$$\|\bar{z}\|_2 = \sqrt{x^2 + y^2}$$



En MATLAB: `norm(z)` o `norm(z,2)`

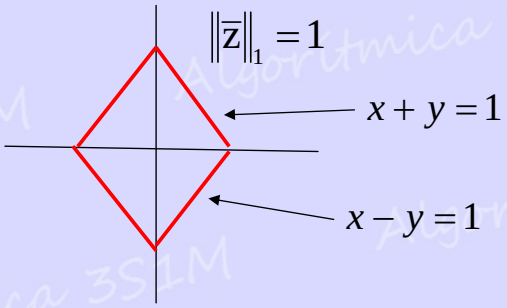
Normas de Vectores

Norma de un vector (medida de su tamaño):

$$\|\bar{x}\|_1 = \sum_k |x_k|, \quad \|\bar{x}\|_2 = \left(\sum_k |x_k|^2 \right)^{1/2} \quad \ldots \quad \|\bar{x}\|_\infty = \max\{|x_k|\}$$

Caso 2D. Sea un vector z de componentes (x,y) $\bar{z} = \begin{pmatrix} x \\ y \end{pmatrix}$

$$\|\bar{z}\|_1 = |x| + |y|$$



En MATLAB: `norm(z,1), sum(abs(z))`

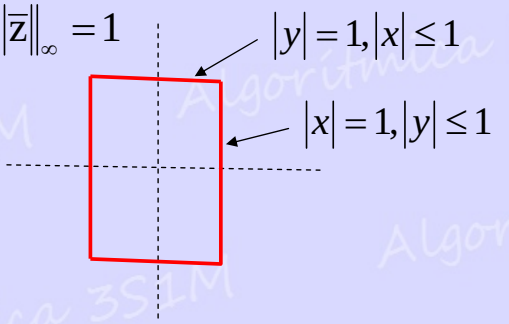
Normas de Vectores

Norma de un vector (medida de su tamaño):

$$\|\bar{x}\|_1 = \sum_k |x_k|, \quad \|\bar{x}\|_2 = \left(\sum_k |x_k|^2 \right)^{1/2} \quad \ldots \quad \|\bar{x}\|_\infty = \max\{|x_k|\}$$

Caso 2D. Sea un vector z de componentes (x,y) $\bar{z} = \begin{pmatrix} x \\ y \end{pmatrix}$

$$\|\bar{z}\|_\infty = \max\{|x|, |y|\}$$



En MATLAB: `norm(z,Inf), max(abs(z))`

Normas de Matrices

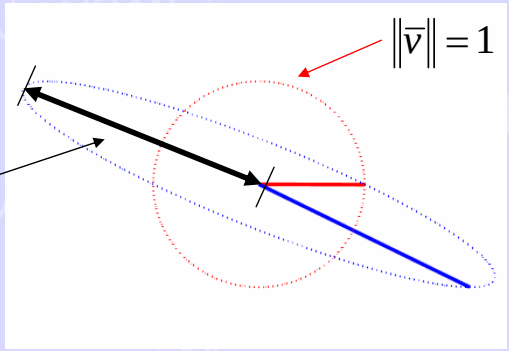
Norma de una matriz = máxima magnificación al aplicarse a un vector.

$$\|A\| = \max\left\{\frac{\|A\bar{x}\|}{\|\bar{x}\|}\right\} = \max_{\|\bar{x}\|=1}\{\|A\bar{x}\|\}$$

Si la norma de A es 5, ningún vector puede hacerse más de 5 veces más grande cuando se le aplica la matriz A.

Las usaremos para cotas: $\|A\bar{x}\| \le \|A\| \cdot \|\bar{x}\|$

Máxima magnificación de la matriz A sobre los vectores con ||v||=1

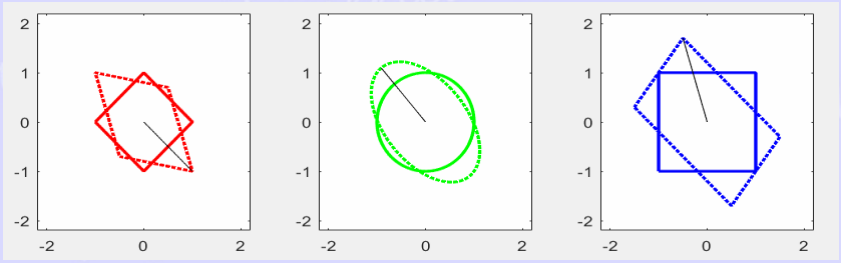


Cálculo de Normas matriciales

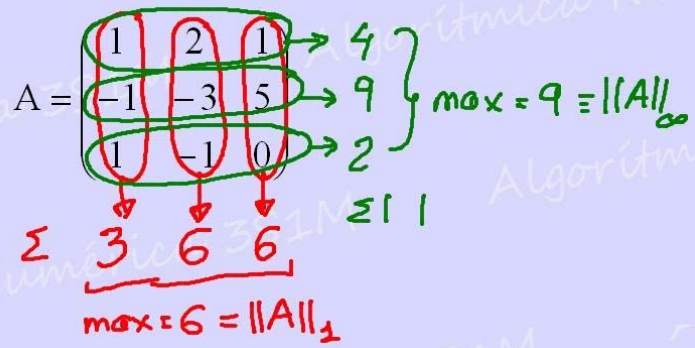
Dependen de la norma vectorial usada:

$$\|A\|_1 = \max_j\left\{\sum_i |A(i,j)|\right\}$$

$$\|A\|_\infty = \max_i\left\{\sum_j |A(i,j)|\right\}$$



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



Operaciones básicas Álgebra Lineal

Producto escalar de dos vectores de tamaño N

$$\left(\bar{x}\right)\left(\bar{y}\right) = \sum_k x_k y_k$$

$\propto N$ operaciones

Producto matriz-vector:
N productos escalares

$$\left(A\right)\left(\bar{y}\right) = \left(\bar{c}\right), c_i = \sum_k A_{ik} y_k$$

$\propto N^2$ operaciones

Producto matriz-matriz:
NxN productos escalares.

$$\left(A\right)\left(B\right) = \left(C\right), \quad C_{ij} = \sum_k A_{ik} B_{kj}$$

$\propto N^3$ operaciones

Algunos comandos MATLAB sobre matrices

det(A): determinante de una matriz

inv(A): inversa de la matriz A

x=A\b : solución del sistema Ax = b ("equivalente" a inv(A)*b)

---

norm(v), norm(A) : norma de vector v o matriz A (por defecto usa norma 2)

norm(v,1), norm(v,Inf): norma 1 o infinito de v (idem para matrices)

---

v=diag(A): extrae diagonal de la matriz A a un vector

A=diag(v): crea una matriz diagonal A con los datos del vector v en la diagonal.

triu(A): conserva los elementos de A de la diagonal hacia arriba (u=upper).

tril(A): conserva los elementos de A de la diagonal hacia abajo (l=lower).

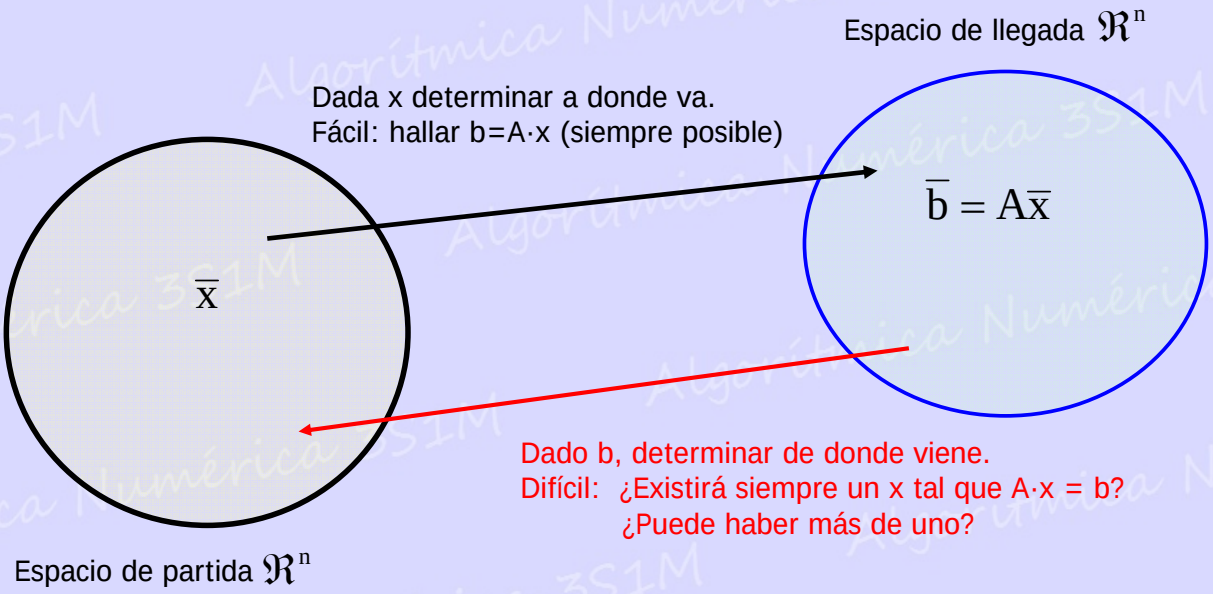
---

eye (N,N): matriz identidad N x N

rand(N,N), randn(N,N) crean matrices aleatorias de tamaño N x N

Nuestro Problema

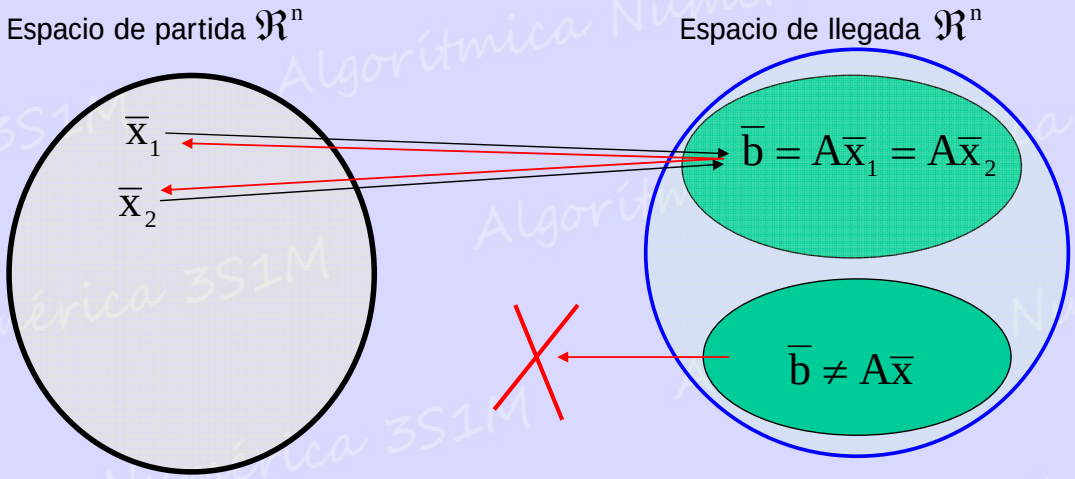
Una matriz A (de tamaño n x n) es una aplicación de $\Re^n$ en $\Re^n$



Dada matriz A (n x n) y vector b, hallar vector x (incógnitas) verificando A·x=b.

Matrices singulares

Dada matriz A y vector b, hallar vector x (incógnitas) que verifica Ax=b.



Si A es singular dos vectores x1 y x2 pueden ir a parar al mismo b.
Dada b, es imposible decidir de cuál venimos → solución múltiple.

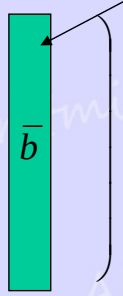
Si algunos {b's} se "llevan" varios vectores del espacio de partida, habrá otros {b's} del espacio de llegada a los que no llega ningún A·x (no existe solución).

Resolución de un sistema lineal

Si A no es singular sabemos que existe una única solución y sabemos hallarla.

Recordar por ejemplo la regla de Cramer:

$$x_k = \frac{\det(A_k)}{\det(A)} \quad con \quad A_k = \left(\right.$$



$\left. \right)$

Poner el vector b en la k-ésima columna de la matriz A

Coste computacional = N + 1 determinantes.

¿Cuánto cuesta hallar un determinante?

Si consideramos el desarrollo de un determinante por filas o columnas:
1 det(N) = N·det(N-1) = N·(N-1)·det(N-2) ... = N·(N-1)·(N-2) ...·1 = N!

Un algoritmo del orden de N! es algo inimaginable.

El absurdo de un algoritmo en N!

Super ordenador Columbia de la NASA (2004-2013)

En su momento uno de los más rápidos existentes

Velocidad = 50 Teraflops = $50 \cdot 10^{12}$ operaciones/seg

¿Qué tamaño de sistema podríamos resolver en un tiempo razonable?

Por ejemplo en 1 minuto = 60 segundos

En ese tiempo nos daría tiempo a hacer $60 \times 50 \cdot 10^{12} \approx 3 \cdot 10^{15}$ operaciones.

Pero 18! ya es del orden de $6 \cdot 10^{15}$. ¿Sólo un sistema de 18 x 18?

¿Y si le damos más tiempo? Incluso disponiendo de $T = 5 \cdot 10^{17}$ segundos (la edad del Universo) sólo nos daría tiempo a resolver un sistema de 30 x 30.

Obviamente así no se resuelven los sistema lineales (ni los determinantes).

Caso sencillo: Sistemas triangulares

El problema de un sistema de ecuaciones es que no es posible ir despejando una incógnita tras otra, al estar todas revueltas en cada ecuación.

Algunos sistemas particulares no presentan este problema. Por ejemplo:

$$\begin{cases} 3x - 3y - 2z = 8 \\ 3y + z = -1 \\ 2z = 4 \end{cases}$$

1) Despejar z en la última ecuación: z=2

2) 2ª ecuación → 3y+2 = -1 ➔ despejo y = (-1-2)/3 = -1

3) 1ª ecuación → 3x + 3 -4 =8 ➔ despejo x = (8-3+4)/3 = 3

Un sistema triangular (en este caso A es triangular superior) es fácil de resolver

Resolución sistemas triangulares

$$\begin{cases} 3x - 3y - 2z = 8 \\ 3y + z = -1 \\ 2z = 4 \end{cases} \xrightarrow[\text{triangular}]{\text{A es matriz}} \begin{pmatrix} 3 & -3 & -2 \\ 0 & 3 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 8 \\ -1 \\ 4 \end{pmatrix}$$

Una matriz triangular superior, U, tiene ceros por debajo de la diagonal.

$$\begin{pmatrix} U_{11} & U_{12} & & & U_{1n} \\ 0 & U_{22} & & & U_{2n} \\ 0 & 0 & & & \\ 0 & 0 & & U_{n-1,n-1} & U_{n-1,n} \\ 0 & 0 & 0 & 0 & U_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \\ b_{n-1} \\ b_n \end{pmatrix}$$

1) Despejar x(n) en la última ecuación

2) Conocida x(n), despejar x(n-1) en la penúltima ecuación

3) ...

Coste Computacional

$$x_n = \frac{b_n}{u_{nn}}$$ → 1div — 1 op

$$x_{n-1} = \frac{(b_{n-1} - u_{n-1,n}x_n)}{u_{n-1,n-1}}$$ → 1 div + 1 mult/suma — 2 op

$$x_{n-2} = \frac{(b_{n-2} - u_{n-2,n-1}x_{n-1} - u_{n-2,n}x_n)}{u_{n-2,n-2}}$$ → 1 div + 2 mult/suma — 3 op

$$x_r = \frac{\left(b_r - \sum_{k=r+1}^{n} u_{rk}x_k\right)}{u_{rr}}$$

$$x_1 = \frac{(b_1 - u_{12}x_2 - u_{13}x_3 - \ldots - u_{1n}x_n)}{u_{11}}$$ → 1 div + (n-1) mult/suma — n op

Coste computacional total: $1+2+3+\ldots+n \approx \frac{n^2}{2}$

Código para resolver triangular superior U

$$x_N = \frac{b_N}{U_{NN}}$$

Incógnitas desde r = (N-1)→1

$$x_r = \frac{\left(b_r - \sum_{k=r+1}^{N} U_{rk}x_k\right)}{U_{rr}}$$

```
function x=solve_U(U,b)
N=length(b);  x = NaN*zeros(N,1);
x(N)=b(N)/U(N,N); % resuelvo x(N)
for r=N-1:-1:1 % resuelvo x(N-1),…,x(2),x(1)
  S = b(r);
  for k=r+1:N, S = S - U(r,k)*x(k); end
  x(r) = S/U(r,r);
end
return
```

Triangular inferior → idem

Una matriz triangular superior, L, tiene ceros por encima de la diagonal.

$$\begin{pmatrix} L_{11} & 0 & 0 & & 0 \\ L_{21} & L_{22} & 0 & & 0 \\ L_{31} & L_{32} & L_{33} & & \\ \dots & \dots & \dots & & 0 \\ L_{n1} & L_{n2} & L_{n3} & \dots & L_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \\ b_{n-1} \\ b_n \end{pmatrix}$$

1) despejar x1

2) Conocida x1, despejar x2

Mismo número de operaciones, $1+2+3+\dots+n=\frac{n^2}{2}$

Código para triangular inferior L

$$x_1 = \frac{b_1}{L_{11}}$$

for r = 2 → N

$$x_r = \frac{\left(b_r - \sum_{k=1}^{r-1} L_{rk} x_k\right)}{L_{rr}}$$

```
function x=solve_L(L,b)
N=length(b);  x = NaN*zeros(N,1);
x(1)=b(1)/L(1,1);
for r=2:N % resuelvo x(2),…,x(N-1),x(N)
    S = b(r);
    for k=1:r-1, S = S - L(r,k)*x(k); end
    x(r) = S/L(r,r);
end
return
```

Objetivo: llegar a sistema triangular

- Pasar de un sistema A·x=b a un sistema triangular U·x=c equivalente.
- Si U es una matriz triangular (p.e. superior), ya sabemos como resolver U·x=c.
- La primera tarea es llenar la primera columna con ceros por debajo de diagonal:

$$\begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix} \xrightarrow{?} \begin{pmatrix} A'_{11} & A'_{12} & \dots & A'_{1n} \\ 0 & A'_{22} & \dots & A'_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & A'_{n2} & \dots & A'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \dots \\ b'_n \end{pmatrix}$$

Si lo conseguimos podremos aplicar el mismo algoritmo al subsistema marcado y podremos llegar a nuestro objetivo → sistema triangular superior, Ux = c.

Operaciones permitidas: sumar, restar, combinar filas (ecuaciones)

Algoritmo de eliminación de Gauss

Objetivo: poner la 1ª columna con 0's por debajo de diagonal.

1. Usar elemento de la diagonal A(1,1) como **punto de apoyo o pivote**.
2. Para cada fila por debajo (desde k=2 hasta N):
   - Calcula cociente m = A(k,1) / A(1,1) = (1er elemento fila k) / pivote
   - Fila' (k) = Fila (k) – m · Fila (1)
   - Se garantiza que el 1er elemento de la nueva fila es ahora nulo:

$$A'_{k1} = A_{k1} - m \cdot A_{11} = A_{k1} - \frac{A_{k1}}{A_{11}} A_{11} = 0$$

Al terminar, todos los primeros elementos de las filas 2 a N serán nulos.

Repetir para las siguiente columnas usando A(2,2), A(3,3), etc. como pivotes.

Código de Eliminación de Gauss

```
function [U c]=gauss(A,b)   %Eliminación de Gauss: Ax=b  =>  Ux=c

N = length(b); % Dimensiones del sistema
A = [A b];     % Pego vector b a matriz A para hacer mismas operaciones

for col=1:N-1, % indice col recorre columnas de matriz A desde 1 a N-1

  pivote = A(col,col);    % Pivote = elemento diagonal

  for k=col+1:N,          % Indice k recorre filas por debajo de diagonal
    m = A(k,col)/pivote;  % Cociente entre 1er elemento fila y pivote
    A(k,:)=A(k,:)- m * A(col,:);  % Fila(k) = Fila(k) - m x fila(col)
  end
  % Al terminar, la columna col tiene 0's por debajo de la diagonal
end

 U = A(:,1:N);  % Extraigo matriz triangular final
 c = A(:,N+1);  % Separo vector c del sistema Ux=c equivalente

return
```


Coste computacional de Gauss

1er paso

$$\begin{pmatrix} A'_{11} & A'_{12} & \dots & A'_{1n} \\ 0 & A'_{22} & \dots & A'_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & A'_{n2} & \dots & A'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \dots \\ b'_n \end{pmatrix}$$

→ Procesamos del orden de N filas con N elementos cada una:

~ N x N operaciones

2º paso

$$\begin{pmatrix} A'_{11} & A'_{12} & \dots & A'_{1n} \\ 0 & A'_{22} & \dots & A'_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & A'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \dots \\ b'_n \end{pmatrix}$$

→ Procesamos del orden de N-1 filas con N-1 elementos cada una: ~(N-1) x (N-1) operaciones

Coste total del algoritmo de eliminación de Gauss:

$$N^2 + (N-1)^2 + \dots + 3^2 + 2^2 + 1 = \sum_{k=1}^{N} k^2 = \frac{N^3}{3}$$

Pivotaje

Problema obvio del algoritmo de eliminación de Gauss: ¿y si el pivote=0?

SOLUCIÓN: Cambiarlo por otro que no lo sea.
¿Cómo? Intercambiando filas = cambiar orden de las ecuaciones.

Si A(k,k)=0 :

1. Buscar posible candidato alternativo en la misma columna, por debajo de la diagonal (para no estropear trabajo hecho)

2. Intercambiar filas (sin olvidarse del vector b)

$$\left(\begin{array}{ccccc|c} 1 & 2 & -1 & 0 & 1 & 1 \\ 0 & -1 & 3 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 3 & 7 & 5 & 2 \\ 0 & 0 & 1 & 0 & -2 & 3 \end{array}\right) \rightarrow \left(\begin{array}{ccccc|c} 1 & 2 & -1 & 0 & 1 & 1 \\ 0 & -1 & 3 & 1 & -1 & 0 \\ 0 & 0 & 3 & 7 & 5 & 2 \\ 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 1 & 0 & -2 & 3 \end{array}\right)$$

¿Y si todo lo que queda por debajo son ceros?
Entonces la matriz es singular y no se puede resolver el sistema.

Pivotaje

El pivotaje se ha introducido para evitar algo imposible (una división por cero)

En Algorítmica Numérica si algo es imposible, algo parecido dará problemas.
En la práctica **SIEMPRE** debemos aplicar técnicas de pivotaje.

Antes de usar el elemento de la diagonal para llenar de ceros la columna se elige el mejor pivote posible (el mayor en valor absoluto) de la columna.

¿Dónde buscamos el mejor pivote?

$$\left(\begin{array}{ccccc} 1 & 2 & -1 & 0 & 1 \\ 0 & -1 & 3 & 1 & -1 \\ 0 & 0 & 1 & -1 & 3 \\ 0 & 0 & 2 & 4 & 5 \\ 0 & 0 & 3 & -6 & -2 \end{array}\right)$$

Pivote parcial
Intercambio de filas

$$\left(\begin{array}{ccccc} 1 & 2 & -1 & 0 & 1 \\ 0 & -1 & 3 & 1 & -1 \\ 0 & 0 & 1 & -1 & 3 \\ 0 & 0 & 2 & 4 & 5 \\ 0 & 0 & 3 & -6 & -2 \end{array}\right)$$

Pivote Total
Intercambio filas/columnas

Código de Gauss con pivotaje (para LAB)

```
for col=1:N-1, % recorro columnas de 1 a N-1 de la matriz A

 % Aquí vendría código pivotaje

 % Resto del código queda como estaba

  pivote = A(col,col);     % Pivote = elemento diagonal
  for k=col+1:N,           % Recorro filas (k's) por debajo diagonal
    m = A(k,col)/pivote;   % Cociente 1er elemento fila/pivote
    A(k,:)=A(k,:)-m*A(col,:); % Fila(k) = Fila(k) - m x fila(col)
  end

end
```

1. Buscar en columna col desde diagonal hasta abajo.
2. Encontrar el valor máximo (valor absoluto) y determinar en que fila n está.
3. Intercambiar fila n y fila col.

Problema 2

Dado el sistema $\begin{cases} \varepsilon x + y = 1 \\ x - y = 0 \end{cases}$ demostrar que su solución exacta es $x = y = \frac{1}{1+\varepsilon}$

Para ε =0.001 resolver (con una precisión de 3 cifras significativas) usando eliminación de Gauss. Comparar con solución exacta.

Repetir intercambiando las ecuaciones.

$$\left.\begin{array}{l} \varepsilon x + y = 1 \\ x - y = 0 \longrightarrow x = y \end{array}\right\} \quad \varepsilon x + x = 1 = (\varepsilon + 1)x \Rightarrow x = \frac{1}{1+\varepsilon}$$

Para $\varepsilon = 0.001$ la solución sería x=y=1/1.001 = 0.999001...

Veamos qué resultados obtenemos trabajando con 3 cifras significativas.

Tras cada operación, se guarda el resultado con 3 cifras significativas
Pista: pensar en notación científica, tres cifras significativas = 3 cifras de mantisa.

Problema 2 (Gauss sin pivote)

$$\begin{matrix} 1.00\times10^{-3}x + y = 1 \\ x - y = 0 \end{matrix} \longrightarrow \left(\begin{matrix} 0.001 & 1 \\ 1 & -1 \end{matrix}\middle|\begin{matrix} 1 \\ 0 \end{matrix}\right) \qquad m = \frac{1}{0.001} = 1000$$

Fila 2 = fila 2 – 1000 · (fila 1)

= 1 – 1000 x 0.001 = 0

= -1 - 1000 x 1 = -1001 = $-1.001\times10^{3} \approx -1.00\times10^{3} = -1000$

= 0 - 1000 x 1 = -1000 = $-1.00\times10^{3} = -1000$

$$\left(\begin{matrix} 0.001 & 1 \\ 0 & -1000 \end{matrix}\middle|\begin{matrix} 1 \\ -1000 \end{matrix}\right) \rightarrow \begin{matrix} 0.001x + y = 1 \\ -1000y = -1000 \end{matrix}$$

De la segunda ecuación se obtiene y=1

La primera ecuación queda 0.001x + 1 = 1 → 0.001x = 0 x=0

El resultado de la segunda incógnita (y) es “correcto” (para la precisión usada), pero el de la primera (x) es inaceptable.

Problema 2 (pivotando)

$$\left(\begin{matrix} 0.001 & 1 \\ 1 & -1 \end{matrix}\middle|\begin{matrix} 1 \\ 0 \end{matrix}\right) \longrightarrow \left(\begin{matrix} 1 & -1 \\ 0.001 & 1 \end{matrix}\middle|\begin{matrix} 0 \\ 1 \end{matrix}\right) \qquad m = \frac{0.001}{1} = 0.001$$

Fila 2 = fila 2 – 0.001 (fila 1):

```
0.001 - 0.001·1 = 0
1-0.001·(-1) = 1.001 ~ 1.00  (con 3 cifras)
1-0.001·0 = 1
```

$$\left(\begin{matrix} 1 & -1 \\ 0 & 1.00 \end{matrix}\middle|\begin{matrix} 0 \\ 1 \end{matrix}\right) \longrightarrow \begin{matrix} x - y = 0 \\ y = 1 \end{matrix}$$

De la segunda ecuación sale y=1 . De la primera tenemos que x = y = 1

El resultado correcto era x = y = 0.999001...

El resultado obtenido (x=y=1) es casi correcto (dentro de la precisión esperada)

Si calculamos con sólo 3 cifras no podemos esperar soluciones con más cifras.

La ineficiencia de la eliminación de Gauss

Dado sistema A·x=b, la eliminación de Gauss lo convierte en U·x=c ($\frac{N^3}{3}$ ops)

Si al rato me piden A·x=b', con el mismo algoritmo llego a U·x=c' (mismo coste)

Esto es un gran desperdicio:

1. La mayoría de las operaciones del primer caso se repetirán en el segundo (todas las que se hacían sobre las componentes de la matriz A)

2. La matriz final U es la misma en los dos casos (las decisiones sobre que filas combinar/intercambiar solo dependían de lo que hubiera en A, no en b o b').

Idea: si hubiera anotado todas las operaciones que le hicimos al vector b
podría repetir solo esas operaciones con b' (mucho menos trabajo)
y podría volver a usar la matriz U (a la que de todas formas llegaría).

Vamos a proponer un algoritmo que a primera vista parece distinto, pero que es equivalente a esta idea de guardar U + (operaciones sobre b para llegar a c)

Factorización LU de una matriz

Dada A, hallar L (triangular inferior) y U (triangular inferior) tal que A = L·U

Conocida esa factorización podría resolver fácilmente cualquier sistema A·x=b:

$$Ax = b \rightarrow L\underbrace{U \cdot x}_{\downarrow} = b \qquad \text{Defino } c = U \cdot x$$

$$Lc = b \quad \text{(hallo c en } \frac{N^2}{2} \text{ ops. al ser L triangular)}$$

$$\downarrow$$

$$Ux = c \quad \text{(resuelvo x en } \frac{N^2}{2} \text{ ops. al ser U triangular)}$$

Conocida la factorización A=L·U puedo resolver Ax=b para cualquier b con sólo resolver 2 sistemas triangulares (con un coste de N^2 operaciones)

La factorización LU de una matriz es una especie de pre-procesado de A, que nos facilita la posterior resolución de cualquier sistema del tipo A·x = b.

También puede usarse para otras aplicaciones (determinantes, inversas)

Comparación Factorización LU / Gauss

a) Gauss: $Ax = b \rightarrow \frac{N^3}{3} \rightarrow U \cdot x = c \rightarrow \frac{N^2}{2} \rightarrow x$

$Ax' = b' \rightarrow \frac{N^3}{3} \rightarrow U \cdot x' = c' \rightarrow \frac{N^2}{2} \rightarrow x'$

Resolver cada sistema con Gauss es del orden de $\frac{N^3}{3}$

b) A $\xrightarrow{?}$ L,U / A=LU

$Ax = b \xrightarrow[\frac{N^2}{2}]{Lc = b} c \xrightarrow[\frac{N^2}{2}]{Ux = c} x \quad N^2$

$Ax' = b' \xrightarrow[\frac{N^2}{2}]{Lc' = b'} c' \xrightarrow[\frac{N^2}{2}]{Ux' = c'} x' \quad N^2$

Conocidas L,U resolver cada sistema es del orden de N^2

¿Cómo hallo una factorización LU de A y cuánto me va a costar?

Factorización LU a partir de eliminación Gauss

Supongamos que tenemos la U obtenida con Gauss. ¿Cuál sería la L tal que L·U=A?

- Factorización LU:

$L \cdot U \cdot x = b$ ----defino c=Ux----> $L \cdot c = b$ ----resuelvo c----> $U \cdot x = c$ → Resuelvo: misma x

- Gauss:

$A \cdot x = b$ ----Paso a sistema triangular U----> $U \cdot x = c$ → Resuelvo: misma x

Si la U es común, el vector c obtenido al resolver L·c = b en la factorización LU deber ser el mismo vector c al que se llegaba por eliminación de Gauss.

El vector c solo depende de L y b: de alguna forma en la matriz L se guardan las operaciones hechas en Gauss sobre el vector b para convertirlo en c.

Esto incluía acordarse de los cocientes {m's} y las filas involucradas.

Factorización LU a partir de eliminación Gauss

Un posible (no es el único) algoritmo de factorización LU es aprovechar la eliminación de Gauss para obtener U. La matriz L correspondiente tiene 1's en la diagonal y el resto de sus elementos de L son los cocientes {m's}.

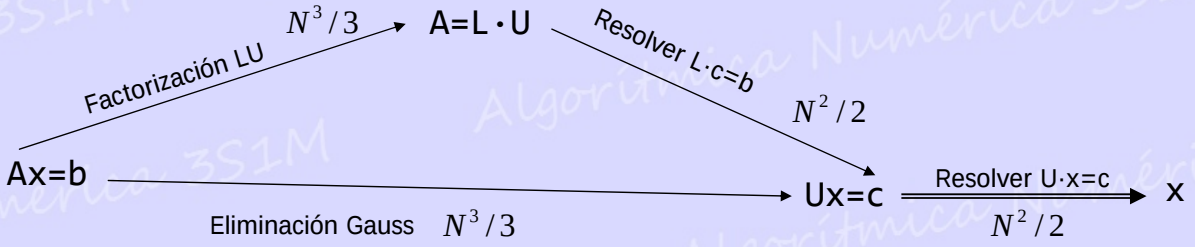
$$A = \begin{pmatrix} 1 & 0 & 0 & & 0 \\ L_{21} & 1 & 0 & & 0 \\ L_{31} & L_{32} & 1 & & \\ \dots & \dots & \dots & & 0 \\ L_{n1} & L_{n2} & L_{n3} & \dots & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & & & U_{1n} \\ 0 & U_{22} & & & U_{2n} \\ 0 & 0 & & & \\ 0 & 0 & & U_{n-1,n-1} & U_{n-1,n} \\ 0 & 0 & 0 & 0 & U_{nn} \end{pmatrix}$$

L(k,j)

```
for j=1:N-1,              % Barro columnas desde 1 a N-1
  pivote = A(j,j);        % Pivote = elemento de la diagonal
  for k=j+1:N,            % Barro filas debajo de diagonal para poner 0's
    m = A(k,j)/pivote;    % Cociente entre 1er elemento fila y pivote
    ...
  end
end
```


Coste computacional Factorización LU

Coste computacional ~ Eliminación de Gauss (un poco menos al no hacer operaciones con el vector b)



Si hemos pivotado (intercambio de filas) L · U = permutación de filas de A.

En ese caso hay que guardar las permutaciones realizadas para aplicarselas al b antes de empezar a resolver los dos sistemas triangulares.

Determinante de A=LU (prob. 3,4,6)

Recordad aplicación ingenua de la regla del determinante: coste de N!

Como A = L·U, usaremos la propiedad de que det (A x B) = det (A) x det (B)

$$A = \begin{pmatrix} 1 & 0 & 0 & & 0 \\ L_{21} & 1 & 0 & & 0 \\ L_{31} & L_{32} & 1 & & \\ \dots & \dots & \dots & & 0 \\ L_{n1} & L_{n2} & L_{n3} & \dots & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & & & U_{1n} \\ 0 & U_{22} & & & U_{2n} \\ 0 & 0 & & & \\ 0 & 0 & & U_{n-1,n-1} & U_{n-1,n} \\ 0 & 0 & 0 & 0 & U_{nn} \end{pmatrix}$$

$$\det(A) = \det(L) \cdot \det(U) = (1 \cdot 1 \cdot \dots \cdot 1)(U_{11} \cdot U_{22} \cdot \dots \cdot U_{NN})$$

Coste computacional = factorización LU + N multiplicaciones (despreciable) = $\frac{N^3}{3}$

Si ha habido pivotaje cambiaremos el signo por cada permutación de filas.

Inversa de A=LU (probs. 6, 9)

$$\begin{pmatrix} & A & \end{pmatrix} \begin{pmatrix} & A^{-1} & \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} & A & \end{pmatrix} \left(\begin{pmatrix} x_1 \end{pmatrix} \begin{pmatrix} x_2 \end{pmatrix} \begin{pmatrix} x_3 \end{pmatrix} \right) = \left(\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right)$$

Invertir matriz = resolver N sistemas con $b = \begin{pmatrix} 1 \\ 0 \\ \dots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ \dots \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \end{pmatrix}$ usando la misma A.

Coste computacional = factorización LU + resolver N sistemas =

$$\frac{N^3}{3} + N \cdot (N^2) = N^3 \longrightarrow \frac{4}{3} N^3$$

Optimizando un poco las operaciones pueden bajarse hasta N^3

Al resolver A·x=b, ¿usamos inv(A)*b o A\b?

Para resolver A·x=b, ¿qué es preferible? a) Disponer de la inversa de A: A^{-1}

b) Disponer de L,U tal que $A = L \cdot U$

¿Desde qué situación cuesta menos hallar x?

Cuesta lo mismo. En a) hay que hacer $x = A^{-1}b$ (matriz x vector) = N^2

En b) hay que resolver 2 sist. triangulares $2\frac{N^2}{2} = N^2$

No hay ventajas en disponer de la inversa frente a una factorización LU

Y calcular la inversa (N^3) es más costoso que hacer una factorización LU ($\frac{N^3}{3}$)

De hecho, para calcular la inversa lo más eficaz es factorizar previamente A=LU

Es por esto por lo que MATLAB recomienda usar x=A\b en vez de x=inv(A)*b

Resumen de lo publicado

- Resolver sistemas triangulares (superiores o inferiores) es sencillo y tiene un coste computacional de $N^2/2$.

- La eliminación de Gauss nos permite pasar de un sistema A·x=b al sistema equivalente triangular superior U·x=c con un coste de $N^3/3$.

- La misma eliminación de Gauss nos permite también factorizar una matriz como A=L·U (triangular inferior ·triangular superior) en $N^3/3$

- Conocida la factorización A=LU podemos resolver el sistema A·x=b si resolvemos 2 sistemas triangulares (L y U) con un coste total de N^2 .

- La factorización LU de una matriz puede ser también el paso previo para otras aplicaciones (cálculo de la inversa, determinante, etc.)

Problemas del TEMA

- Resolución de sistemas triangulares (L,U)
- Algoritmo de Eliminación de Gauss para pasar de Ax=b => Ux=c
- Algoritmo de Factorización LU (eliminación de Gauss + guardar m's)
- Uso de pivotaje parcial (aplicable a ambos casos anteriores).
- Aplicaciones de la factorización A=LU:
  - Resolver LU x = b como solución de 2 triangulares (L y U)
  - Cálculo de inversa de una matriz,
  - Determinante de una matriz.

Todo esto, por supuesto, para sistemas pequeños, 2x2 o 3x3 máximo

Eliminación Gauss: prob 3a, prob 4, Factorizar LU: prob 3b, prob 4

Uso de pivotaje: prob 2, prob 3a Aplicaciones: prob 3, prob 5, prob 6

Problema 3

Resolver el siguiente sistema Ax=b, con

$$A = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 5 & 1 \\ 1 & 1 & 3 \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

por eliminación de Gauss. Repetir calculando previamente su factorización LU.

¿Merece la pena pasar por la factorización LU en este caso?

Respuesta: no, para resolver un único sistema eliminación de Gauss es tan eficiente como hacer la factorización LU y resolver 2 triangulares.

Problema 3: resolver con eliminación Gauss

$$\left(\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ -1 & 5 & 1 & \cdot 1 \\ 1 & 1 & 3 & 0 \end{array}\right) \quad \begin{array}{l} m = -1 \rightarrow \text{Fila 2 = Fila 2 + Fila 1} \\ m = 1 \rightarrow \text{Fila 3 = Fila 3 – Fila 1} \end{array}$$

$$\left(\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 0 & 4 & 2 & \cdot 2 \\ 0 & 2 & 2 & -1 \end{array}\right) \quad m = \frac{2}{4} = \frac{1}{2} \rightarrow \text{Fila 3 = Fila 3 – 0.5 Fila 2}$$

$$\left(\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 0 & 4 & 2 & \cdot 2 \\ 0 & 0 & 1 & -2 \end{array}\right) \longrightarrow \begin{cases} x - y + z = 1 \longrightarrow x = \frac{9}{2} \\ 4y + 2z = 2 \longrightarrow y = \frac{3}{2} \\ z = -2 \end{cases}$$

Problema 3: factorización LU

U es la matriz triangular superior que salía con Gauss

L es triangular inferior con 1's en la diagonal y los cocientes m's obtenidos antes:

$$U = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 4 & 2 \\ 0 & 0 & 1 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & 0.5 & 1 \end{pmatrix}$$

Para resolver A·x=b, primero resolvemos L·c = b

$$\begin{cases} x = 1 \longrightarrow x = 1 \\ -x + y = 1 \longrightarrow y = 2 \\ x + 0.5y + z = 0 \longrightarrow z = -2 \end{cases} \qquad c = \begin{pmatrix} 1 \\ 2 \\ -2 \end{pmatrix}$$

Que es justo el vector c al que llegábamos con eliminación Gauss

Solo quedaría resolver Ux=c que es lo que ya está hecho en eliminación de Gauss.

Otra opción para factorizar L·U

Si no se requiere pivotaje (y no nos piden hacer eliminación de Gauss) podemos plantear directamente la factorización L·U como:

$$L \cdot U = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{pmatrix} \cdot \begin{pmatrix} d & e & f \\ 0 & g & h \\ 0 & 0 & i \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -3 \\ 3 & -3 & 4 \end{pmatrix} = A$$

Si se plantean las ecuaciones en el orden correcto, en cada ecuación solo nos aparecerá incógnita que desconocemos.

Empezamos con los productos escalares correspondientes a la 1ª fila de la matriz A, los más sencillos:

$$\begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{pmatrix} \cdot \begin{pmatrix} d & e & f \\ 0 & g & h \\ 0 & 0 & i \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -3 \\ 3 & -3 & 4 \end{pmatrix} \longrightarrow d = 2 \quad e = -1 \quad f = -1$$

Otra opción para factorizar L·U

Conocidas d, f, e las substituimos en nuestra factorización

$$L \cdot U = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & -1 & -1 \\ 0 & g & h \\ 0 & 0 & i \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -3 \\ 3 & -3 & 4 \end{pmatrix} = A$$

y seguimos con los productos correspondientes a la 2ª fila de la matriz A:

$$\begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & -1 & -1 \\ 0 & g & h \\ 0 & 0 & i \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -3 \\ 3 & -3 & 4 \end{pmatrix} \longrightarrow \begin{array}{l} 2a = -1 \\ -a + h = -3 \\ -a + g = 2 \end{array}$$

$$2a = -1 \rightarrow a = -\frac{1}{2} \qquad -a + g = 2 \rightarrow g = 2 + a = \frac{3}{2} \qquad -a + h = -3 \rightarrow h = a - 3 = -\frac{7}{2}$$

Otra opción para factorizar L·U

Incorporamos los nuevos valores (a, g, h) conocidos

$$L\cdot U=\begin{pmatrix}1 & 0 & 0\\ -1/2 & 1 & 0\\ b & c & 1\end{pmatrix}\cdot\begin{pmatrix}2 & -1 & -1\\ 0 & 3/2 & -7/2\\ 0 & 0 & i\end{pmatrix}=\begin{pmatrix}2 & -1 & -1\\ -1 & 2 & -3\\ 3 & -3 & 4\end{pmatrix}=A$$

y terminamos con los productos correspondientes a la 3ª fila de la matriz A:

$$\begin{pmatrix}1 & 0 & 0\\ -1/2 & 1 & 0\\ b & c & 1\end{pmatrix}\cdot\begin{pmatrix}2 & -1 & -1\\ 0 & 3/2 & -7/2\\ 0 & 0 & i\end{pmatrix}=\begin{pmatrix}2 & -1 & -1\\ -1 & 2 & -3\\ 3 & -3 & 4\end{pmatrix}$$

$$2b=3$$

$$-b+(3/2)\cdot c=-3$$

$$-b+(7/2)\cdot c+i=4$$

$$2b=3 \rightarrow b=\frac{3}{2}$$

$$c=\frac{2}{3}(b-3) \rightarrow c=-1$$

$$i=4+b-(7/2)\cdot c \rightarrow i=4+\frac{3}{2}-\frac{7}{2}\cdot 1=2$$

Otra opción para factorizar L·U

Y terminamos con la misma factorización que antes:

$$L\cdot U=\begin{pmatrix}1 & 0 & 0\\ -1/2 & 1 & 0\\ 3/2 & -1 & 1\end{pmatrix}\cdot\begin{pmatrix}2 & -1 & -1\\ 0 & 3/2 & -7/2\\ 0 & 0 & 2\end{pmatrix}=\begin{pmatrix}2 & -1 & -1\\ -1 & 2 & -3\\ 3 & -3 & 4\end{pmatrix}=A$$

Prob 6: factorización L·U de A = $\begin{pmatrix}2 & 1\\ -2 & 2\end{pmatrix}$ $\qquad$ $\begin{pmatrix}1 & 0\\ a & 1\end{pmatrix}\begin{pmatrix}b & c\\ 0 & d\end{pmatrix}=\begin{pmatrix}2 & 1\\ -2 & 2\end{pmatrix}$

1ª fila: b = 2, y queda c = 1 $\qquad$ $\begin{pmatrix}1 & 0\\ a & 1\end{pmatrix}\begin{pmatrix}2 & 1\\ 0 & d\end{pmatrix}=\begin{pmatrix}2 & 1\\ -2 & 2\end{pmatrix}$

2ª fila: 2a =-2 → a =-1
a+d = 2 → d = 2-a = 3 $\qquad$ $A=\begin{pmatrix}2 & 1\\ -2 & 2\end{pmatrix}=\begin{pmatrix}1 & 0\\ -1 & 1\end{pmatrix}\begin{pmatrix}2 & 1\\ 0 & 3\end{pmatrix}$

Resumen métodos directos

Estrategias presentadas para resolver un sistema lineal en este tema:

1) Eliminación de Gauss ($A \cdot x = b \rightarrow U \cdot x = c$) + resolución sistema triangular.
2) Factorización $A = L \cdot U$ + resolución de dos triangulares $L \cdot c = b$, $U \cdot x = c$

El primer caso es adecuado cuando sólo tenemos un vector b (o sabemos a priori todos los vectores b's que vamos a necesitar resolver). El segundo cuando tenemos que resolver para b's distintos (no al mismo tiempo), o como paso previo para calcular determinante, inversa,...

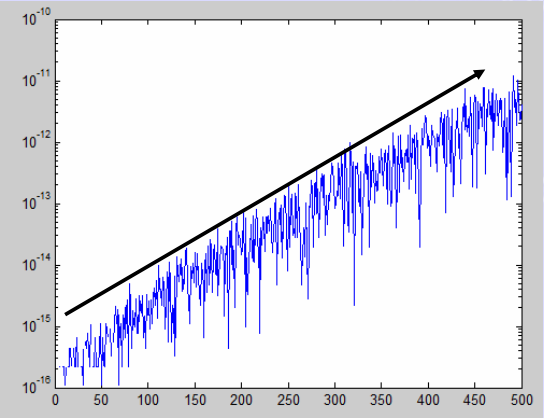
- La factorización $A = L \cdot U$ (1's en la diagonal de la L) presentada no es única, es la que sale de forma natural al aplicar eliminación de Gauss.

Existen otros métodos directos, muchos de ellos basados en el mismo concepto de factorizar A en matrices que facilitan las posteriores operaciones. Ejemplos:

- Factorización QR, donde $A = Q \cdot R$ con R triangular superior y Q ortonormal.
- Factorización Cholesky: si A es simétrica definida positiva $A = L \cdot L^T$
- Factorización SVD (descomposición en valores singulares).

Limitaciones de los métodos directos

Un método directo debe pasar por unos pasos predefinidos. Los inevitables errores en el proceso van estropeando CADA VEZ MÁS la solución final.



Gráfica del error en la resolución de un sistema triangular inferior.

El error aumenta progresivamente para las sucesivas componentes.

Esta acumulación inevitable de los errores hace que los métodos directos no sean adecuados para sistemas muy grandes.

Alternativa: métodos iterativos como los usados con ecuaciones no lineales.

x0 hipótesis inicial (un vector) -> x1, x2, x3, ..., xn ==> vector solución x.

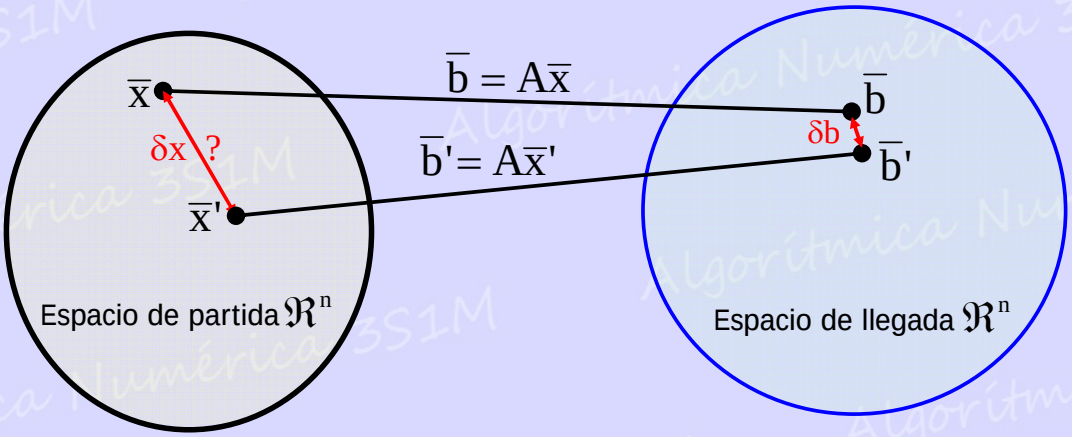
Métodos directos: para sistemas "pequeños": N = ~100's o 1000's

Métodos iterativos: para sistemas grandes: de miles o millones de ecuaciones.

Estudio de errores en métodos directos.

¿Qué pasa si perturbamos el sistema A·x=b, cambiando ligeramente b (por b')?

Obviamente la solución también se modificará (ahora será x', con A·x'=b')



¿Cuál es el cambio de la solución, $\delta x=(x'-x)$, en el espacio de partida causado por una perturbación $\delta b = (b'-b)$ en el espacio de llegada?

Propagación de una perturbación

Se trata de relacionar $\delta\bar{b} = \left(\bar{b}'-\bar{b}\right)$ con $\delta\bar{x} = \left(x'-x\right)$

$$b = Ax \Rightarrow \|b\| \le \|A\| \cdot \|x\| \Rightarrow \frac{1}{\|x\|} \le \|A\| \frac{1}{\|b\|}$$

$$(x'-x) = \left(A^{-1}b'-A^{-1}b\right) = A^{-1}(b'-b) \Rightarrow \|x'-x\| \le \|A^{-1}\| \cdot \|b'-b\|$$

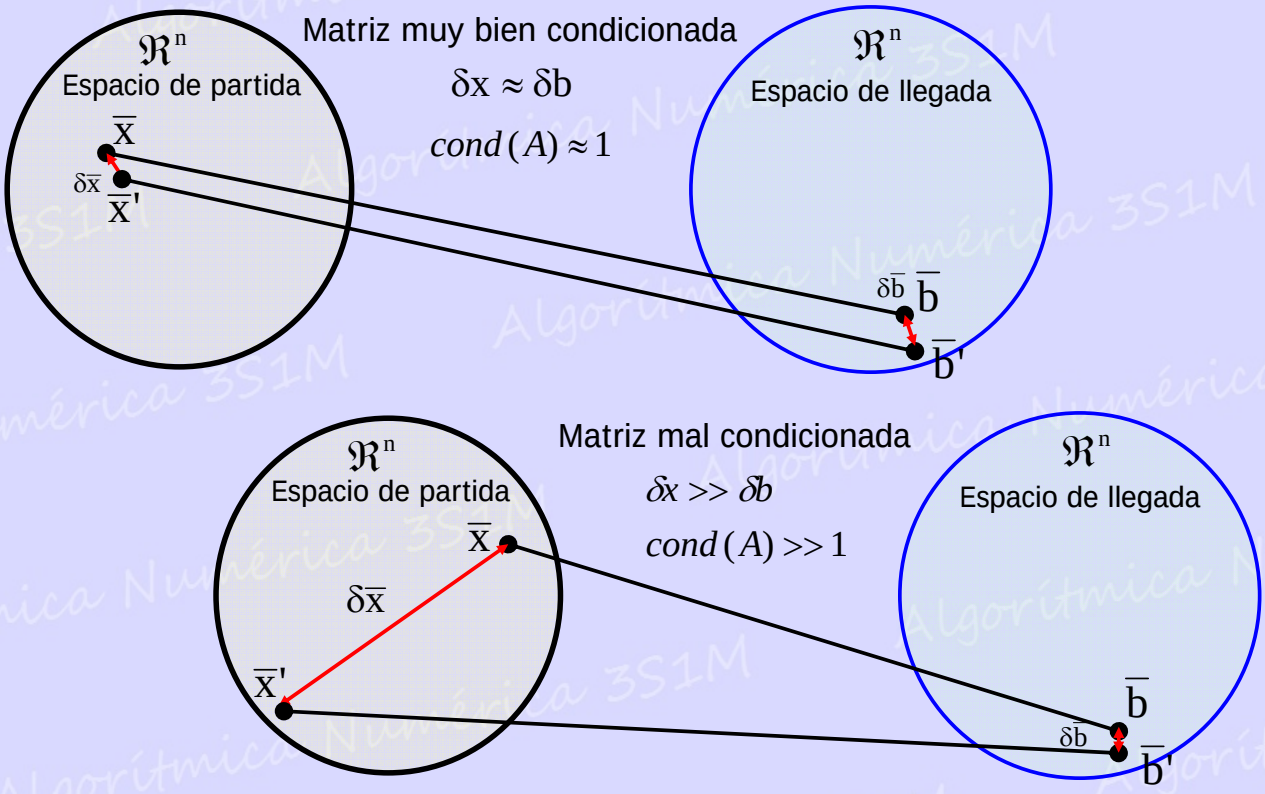
$$\frac{\|x'-x\|}{\|x\|} \le \|A\| \cdot \|A^{-1}\| \cdot \frac{\|b'-b\|}{\|b\|}$$

Error relativo de la perturbación en la solución x

Número de condición, c(A), o condicionamiento de A

Error relativo de la perturbación en los datos b.

Matrices mal y bien condicionadas



Condicionamiento de una matriz

El factor $c(A) = \|A\| \cdot \|A^{-1}\|$ actúa como magnificador de las perturbaciones y se le denomina condicionamiento de A: en MATLAB cond(A)

¿Qué tiene esto que ver con el error? Aquí no estamos hablando de un error sino de una perturbación totalmente voluntaria, cambiar b por b'.

¿ Es realmente voluntaria la perturbación ?

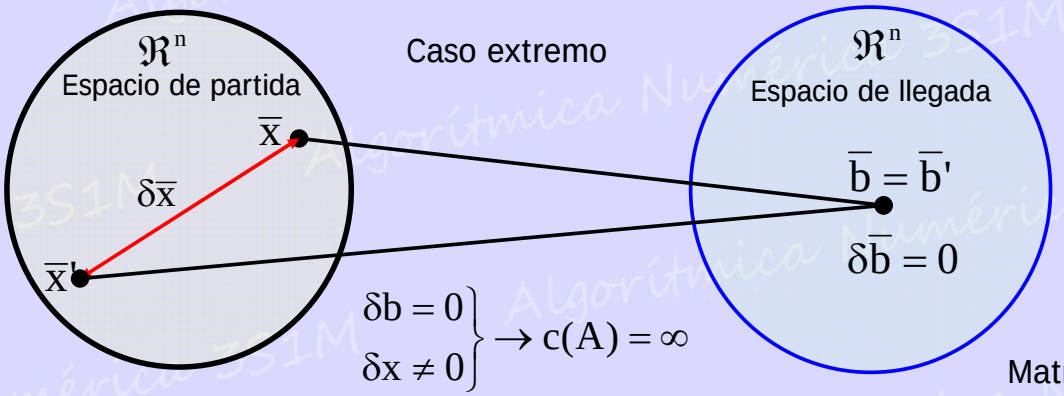
En la vida real siempre hacemos las cuentas con b' y no con el b deseado:

- Si nuestro vector b proviene de datos reales, en realidad estaremos usando un vector b' modificado por los posibles errores de medida.
- Como mínimo la precisión de la máquina causará una perturbación del orden de 1e-16 (en error relativo)

¿Cuántas cifras decimales podríamos perder con un condicionamiento C=1000?

E_rel (x) <= C(A) · E_rel (b)

Caso extremo de mal condicionamiento



$$\left.\begin{array}{l}\delta b = 0\\ \delta x \neq 0\end{array}\right\} \rightarrow c(A) = \infty$$

Matriz singular

En Álgebra Lineal una matriz es singular (det=0) o no lo es (det <>0).

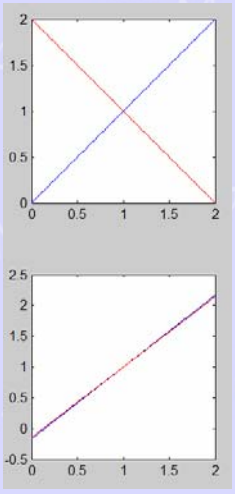
En Algorítmica Numérica si una matriz tiene un comportamiento "similar" a una singular (número de condición muy alto) se considera numéricamente singular.

¿Para que número de condición una matriz se declara numéricamente singular?

Ilustración del condicionamiento en 2D

$$\begin{pmatrix}1 & -1\\ 1 & 1\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} = \begin{pmatrix}0\\ 2\end{pmatrix} \rightarrow \begin{cases}x - y = 0\\ x + y = 2\end{cases}$$

cond(A) = 1



$$s = \begin{pmatrix}1\\ 1\end{pmatrix}$$

$$\begin{pmatrix}7 & -6\\ -8 & 7\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} = \begin{pmatrix}1\\ -1\end{pmatrix} \rightarrow \begin{cases}7x - 6y = 1\\ -8x + 7y = -1\end{cases}$$

cond(A) ~200

$$s = \begin{pmatrix}1\\ 1\end{pmatrix}$$

El condicionamiento de una matriz mide lo cerca que está de ser singular: en 2D lo "paralelas" que son las rectas cuyo corte hay que determinar.

- En el 1er caso tenemos rectas perpendiculares (el mejor caso posible).

El punto de corte (solución) en x=1,y=1 es muy fácil de ver.

- En el 2º caso las rectas son "casi" paralelas y es difícil decir donde se cortan.

¿Cómo saber si nuestra solución es correcta?

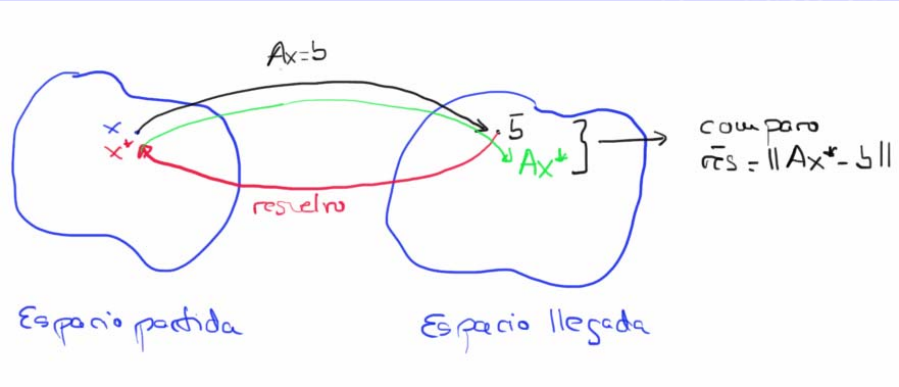
Resolvemos A·x=b → x* (solución calculada, seguramente distinta de s correcta)

¿Cómo podemos saber si lo hemos hecho bien?

En el LAB solemos hacer trampas y usar un sistema cuya solución s conocemos y podemos comparar con la obtenida e=(x*-s). ¿Qué hacer en un caso real?

IDEA (??): hallar el residuo r = Ax* - b (que siempre puedo calcular).

Idealmente r = 0: podría usar la norma de dicho residuo ||r|| (o mejor ||r|| / ||b||) como medida de la bondad de la solución.



El mal condicionamiento ataca de nuevo

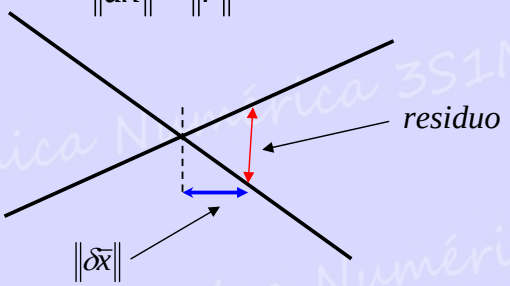
$$\frac{\|\delta\bar{x}\|}{\|\bar{x}\|} = \frac{\|\bar{x}-\bar{x}^*\|}{\|\bar{x}\|} \le cond(A)\frac{\|\bar{r}\|}{\|\bar{b}\|} \quad con \quad \bar{r} = A\bar{x}^* - \bar{b}$$

Si el sistema (matriz A) está mal condicionado podemos tener un residuo pequeño y aún así tener una gran discrepancia entre solución calculada x* y la verdadera x.

Un residuo pequeño no garantiza que la solución calculada este cerca de la verdadera

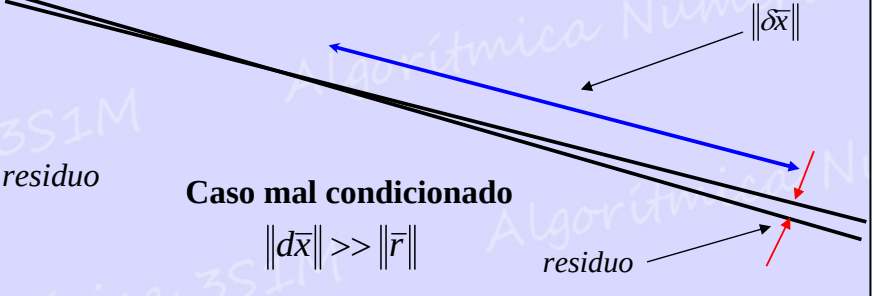
Caso bien condicionado

$\|d\bar{x}\| \approx \|\bar{r}\|$



Caso mal condicionado

$\|d\bar{x}\| >> \|\bar{r}\|$



Propiedades del número de condición

E_rel (x) = c(A) · E_rel (b)

Solución ↑ ↖ Datos problema

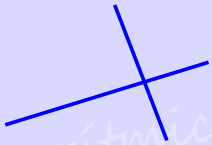
En el mejor de los casos E_rel de los datos será la precisión máquina (1e-15)

El condicionamiento nos hace perder del orden de **log10(c(A)) cifras**.

Propiedades:

c(A) >=1 (las cosas no van a mejor)

c(A) = 1 para matrices ortonormales (mejor caso posible)
cuyas "rectas" se cortan en ángulo recto



c(A)>> para matrices "numéricamente" próximas a ser singulares
cuyas "rectas" son casi paralelas

Estimación del número de condición c(A)

A) Aplicar la definición, hallar inversa, calcular normas $\|A\|, \|A^{-1}\|$, y calcular:

$$c(A) = \|A\| \cdot \|A^{-1}\|$$

Problemas 6 y 9

B) Estimar el número de condición a través de sus efectos en la perturbación de un sistema o en la relación entre residuo y perturbación de la solución.

$$\frac{\|x'-x\|}{\|x\|} \le c(A) \cdot \frac{\|b'-b\|}{\|b\|} \qquad \frac{\|x - x^*\|}{\|x\|} \le c(A)\frac{\|r\|}{\|b\|} \quad \text{con} \quad r = Ax^* - b$$

Problemas 7 y 8

En cualquier caso el número de condición hallado dependerá de la norma usada.

Si no se dice nada se recomienda usar la norma infinito por su simplicidad.