Práctica 3: Resolución numérica de SEL

■ Resolución mediante comandos directos de Mathematica

```
(*comenzamos con un ejemplo de entradas aleatorias*)
n = 50;                    (*número de ecuaciones e incógnitas*)
A = Table[2 * Random[], {i, n}, {j, n}];  (*matriz A aleatoria*)
b = Table[Random[], {i, n}];          (*vector b aleatorio*)
LinearSolve[A, b] ;
(*Resolución DIRECTA (simbólica y/o numérica) del sistema Ax=b*)
x1 = %;                 (*guardamos la solución*)
LinearSolve[A, b] // Timing (*Idem, pero dando el tiempo empleado*)

(*aplicamos ahora reducción por filas a la matriz ampliada*)
At = Transpose[A];          (*auxiliar, para hacer la matriz ampliada*)
Ab = Transpose[AppendTo[At, b]];        (*Matriz (A|b) ampliada*)
Dimensions[Ab]            (*Vemos sus dimensiones, n x (n+1), una columna más*)
Ab = RowReduce[Ab];
(*Aplica GAUSS-JORDAN a la matriz ampliada y la llamamos igual*)
x2 = Transpose[Ab][[n + 1]] ;  (*la última columna contiene la solución*)
Max[Abs[x1 - x2]](*comparamos resultados*)

(*diferencia temporal del LinearSolve simbólico y numérico*)
Clear[A, b, n]
n = 12;
A = Table[E^(-i * j), {i, n}, {j, n}];
b = Table[1, {i, n}];
LinearSolve[A, b]; // Timing
A = N[A];
LinearSolve[A, b]; // Timing
```

- **Ejercicio 1.**
  Repetir el proceso anterior varias veces aumentando el n para comparar los tiempos
  (Elimina algunas salidas que seguramente se salgan de la pantalla)
  (En un PIV 1500Mhz, 256Mb para n=2000 el tiempo es 42 segundos)

- **Ejercicio 2.**
  Resuelve numéricamente Ax = b para la matriz

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & \vdots \\ \vdots & 0 & -1 & \dots & -1 & 0 \\ \vdots & \vdots & \vdots & -1 & 2 & -1 \\ 0 & 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$ **de dimensión (50 x 50) y b=** $$\begin{pmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ \vdots \\ 50 \end{pmatrix}$$ **.**

■ Métodos de descomposición directa

```
n = 5;
A = Table[E^(-1. / i * j), {i, n}, {j, n}];
{q, r} = QRDecomposition[A];
(*da una descomposición QU (o QR) de la
  matriz: la salida consiste en una lista formada por la matriz
     ortogonal Q y la triangular superior R tales que Q^t.R==A ,
 (no necesariamente la de Householder)*)
MatrixForm[q] // N
MatrixForm[r // N]
Transpose[q].r - A // N  (*debería ser la matriz cero... ¿lo es?*)
```

- **Ejercicio 3.**
  Verifica que Q es ortogonal en el ejemplo propuesto.

■ Resolución por métodos iterativos

```
(*comenzamos con el método de Gauss-seidel en otro ejemplo*)
Clear[A, b, n, k, i, j];
n = 50;
A = Table[Random[], {i, n}, {j, n}];
Do[A[[i, i]] = n, {i, n}] (*así conseguimos que A sea EDD y el método converja*)
b = Table[Random[], {i, n}];

x = Table[0, {i, n}]; (*iniciamos x^0 como el vector cero, por ejemplo*)
```

(*recordamos $x_k^{(n+1)} = \left(b_k - \sum_{j=1}^{k-1} a_{kj}\, x_j^{(n+1)} - \sum_{j=k+1}^{n} a_{kj}\, x_j^{(n)} \right) \Big/ a_{kk}$ para G-S*)

```
For[k = 1, k <= n, k++,
    x[[k]] =
  (b[[k]] - Sum[A[[k, j]] * x[[j]], {j, 1, k - 1}] - Sum[A[[k, j]] * x[[j]], {j, k + 1, n}]) /
   A[[k, k]]
   ] (*hemos programado una iteración; ahora hacemos varias usando otro For*)
```

```
x = Table[0, {i, n}];
For[i = 1, i <= 10, i++,
         For[k = 1, k ≤ n, k++,
         x[[k]] =
   (b[[k]] - Sum[A[[k, j]] * x[[j]], {j, 1, k - 1}] - Sum[A[[k, j]] * x[[j]], {j, k + 1, n}]) /
    A[[k, k]]
         ]
   ]
```

```
A.x - b (*Veamos el residuo*)
```

```
(*Modificación para que se detenga si la distancia entre dos iteraciones sucesivas
  (con la norma del máximo por ejemplo) es menor que una cierta tolerancia*)
x = Table[0, {i, n}];
tol = 10^(-6);
For[i = 1, i <= 100, i++,
         y = x;       (*hay que guardar la iteración anterior antes de modificarla*)
         For[k = 1, k ≤ n, k++,
         x[[k]] =
   (b[[k]] - Sum[A[[k, j]] * x[[j]], {j, 1, k - 1}] - Sum[A[[k, j]] * x[[j]], {j, k + 1, n}]) /
    A[[k, k]]
           ];
         distancia = Max[Abs[x - y]];  (*calculamos la distancia*)
          Print["iteración n. = ", i, "    distancia = ", distancia];
         If[distancia < tol, Break[]] (*hacemos el test*)   ]
```

- **Ejercicio 4.**
  Modifica el programa para que en cada iteración muestre la solución aproximada y aplícalo con una matriz 5x5 que

- **Ejercicio 5.**
  Usa una matriz aleatoria cualquiera y verifica que G-S no tiene por qué converger

- **Ejercicio 6.**
  Programa tú el método de Jacobi y el de Relajación para un peso genérico ω.
  Para la matriz del ejercicio 2, compara la velocidad