Laboratorio 2 MA-4301: Algebra Lineal Numérica 2

Gonzalo Hernández

UChile - Departamento de Ingeniería Matemática

En este laboratorio revisaremos los siguientes temas de SEL:

1) Factorización de matrices: $PA = LU, A = LU$ (Crout), $A = RR^T$ (Cholesky), $A = QR$ (Householder), $A = USV$ (Descomposición en valores singulares).

2) Métodos para valores y vectores propios: Potencia, Potencia Inversa, QR, Schur.

3) Métodos iterativos para SEL: Jacobi, Gauss - Seidel, Sobre-Relajación, Gradiente Conjugado.

Los metodos implementados en Matlab para SEL provienen de la librería LAPACK, ver ref. [2].

1 Factorización de matrices

La factorización permite explotar en forma eficiente la estructura y propiedades de ciertas matrices, lo que es de utilidad al resolver gran cantidad de SEL definidos por la misma matriz A pero diferentes vectores lado derecho $b_1, ..., b_m$. En esta sección estudiaremos las factorizaciones: $PA = LU$ (Gauss), $A = LU$ (Crout), $A = R^T R$ (Cholesky), $A = QR$ (Householder), $A = USV$ (Descomposición en valores singulares), $A = U^T TU$ (Descomposición de Schur).

Toda matriz A de $n \times n$ invertible admite una factorización $PA = LU$ de la forma:

$$A = LU = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & 0 \\ m_{31} & m_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn-1} & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{nn}^{(n)} \end{bmatrix}$$

$$\begin{aligned} m_{ik} &= \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad \forall k = 1, ..., (n-1), i = (k+1), ..., n \\ a_{ij}^{(k+1)} &= a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)} \quad \forall k = 1, ..., (n-1), i, j = (k+1), ..., n \end{aligned} \tag{1}$$

donde L es una matriz invertible triangular inferior con diagonal igual a 1 que almacena la información del pivoteo realizado sobre A y U es una matriz invertible triangular superior que resulta del proceso de triangularización de Gauss. El comando Matlab que implementa esta factorización es:

$$[L, U, P] = lu(A) \tag{2}$$

donde $PA = LU$ y P es la matriz de permutación de filas que aparece al pivotear parcialmente. Esta factorización es útil para resolver varios sistemas $Ax = b$ con diferentes vector lado derecho b. Para cada b

se resuelven dos sistemas triangulares por substitución forward y backward:

$$\begin{aligned} Ax &= b \\ PAx &= Pb \\ LUx &= Pb = \begin{cases} Ly = Pb \\ Ux = y \end{cases} \end{aligned} \tag{3}$$

El sistema $Ly = Pb$ es triangular inferior y se resuelve directamente por sustitución forward, mientras que el sistema $Ux = y$ es triangular superior y se resuelve por sustitución backwards. Por ejemplo, si $b = [1\ 2\ 3\ 4]'$ y A es la matriz de Pascal de 4×4 :

```
>> A=pascal(4)
A =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20

>> [L,U,P]=lu(A)
L =
    1.0000         0         0         0
    1.0000    1.0000         0         0
    1.0000    0.6667    1.0000         0
    1.0000    0.3333    1.0000    1.0000
U =
    1.0000    1.0000    1.0000    1.0000
         0    3.0000    9.0000   19.0000
         0         0   -1.0000   -3.6667
         0         0         0    0.3333
P =
     1     0     0     0
     0     0     0     1
     0     0     1     0
     0     1     0     0
```

En este caso, la matriz P refleja que al aplicar el método de Gauss se cambió la segunda fila por la cuarta. Si no hay cambio, P es igual a la matriz identidad.

Los sistemas triangulares $Ly = Pb, Ux = y$ quedan de la siguiente forma:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0.6667 & 1 & 0 \\ 1 & 0.3333 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 3 \\ 2 \end{bmatrix} \Rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 3 & 9 & 19 \\ 0 & 0 & -1 & -3.6667 \\ 0 & 0 & 0 & 0.3333 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

Si la matriz A es tridiagonal:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ 0 & a_{32} & a_{33} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_{(n-1),n} \\ 0 & \cdots & 0 & a_{n,(n-1)} & a_{nn} \end{bmatrix} \tag{5}$$

donde los coeficientes $a_{11}, a_{12}, a_{21}, a_{22}, a_{23}, ..., a_{n(n-1)}, a_{nn}$ son todos distintos de cero y $|a_{ii}| > |a_{i,i+1}| + |a_{i+1,i}|$ $\forall i = 1, ..., n-1$, su factorización $A = LU$ puede ser llevada a la forma:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ 0 & a_{32} & a_{33} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_{(n-1),n} \\ 0 & \cdots & 0 & a_{n,(n-1)} & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ 0 & l_{32} & l_{33} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & l_{n,(n-1)} & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & 0 & \cdots & 0 \\ 0 & 1 & u_{23} & \ddots & \vdots \\ 0 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & u_{(n-1),n} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ 0 & a_{32} & a_{33} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_{(n-1),n} \\ 0 & \cdots & 0 & a_{n,(n-1)} & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & l_{11}u_{12} & 0 & \cdots & 0 \\ l_{21} & l_{21}u_{12} + l_{22} & l_{22}u_{23} & \ddots & \vdots \\ 0 & l_{32} & l_{32}u_{23} + l_{33} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & l_{(n-1),(n-1)}u_{(n-1),n} \\ 0 & \cdots & 0 & l_{n,(n-1)} & l_{n,(n-1)}u_{(n-1),n} + l_{nn} \end{bmatrix}$$

mediante el Método de Crout:

Paso 1:
$l_{11} = a_{11}$ $u_{12} = \frac{a_{12}}{l_{11}}$
Paso 2: Para $i = 2, ..., n-1$
$l_{i,(i-1)} = a_{i,(i-1)}$ $l_{ii} = a_{ii} - l_{i,(i-1)}u_{(i-1),i}$ $u_{i(i+1)} = \frac{a_{i,(i+1)}}{l_{ii}}$
Paso 3:
$l_{n,(n-1)} = a_{n,(n-1)}$ $l_{nn} = a_{nn} - l_{n,(n-1)}u_{(n-1),n}$

Este tipo de matriz aparece al calcular la spline cúbica de una función y al resolver una ecuación diferencial parcial mediante el método de diferencias finitas. Los temas de interpolación con splines y diferencias finitas los veremos en los laboratorios 4 y 9 respectivamente. Por ejemplo, para la matriz tridiagonal:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \tag{7}$$

la factorización de Crout queda:

Paso 1: $l_{11} = a_{11} = 2$ $u_{12} = \frac{a_{12}}{l_{11}} = -\frac{1}{2}$	
Paso 2: $i = 2$ $l_{21} = a_{21} = -1$ $l_{22} = a_{22} - l_{21}u_{12} = 3 - (-1)\left(-\frac{1}{2}\right) = \frac{5}{2}$ $u_{23} = \frac{a_{23}}{l_{22}} = \frac{-1}{\frac{5}{2}} = -\frac{2}{5}$	Paso 2: $i = 3$ $l_{32} = a_{32} = -1$ $l_{33} = a_{33} - l_{32}u_{23} = 3 - (-1)\left(-\frac{2}{5}\right) = \frac{13}{5}$ $u_{34} = \frac{a_{34}}{l_{33}} = -\frac{1}{\frac{13}{5}} = -\frac{5}{13}$
Paso 3: $l_{43} = a_{43} = -1$ $l_{44} = a_{44} - l_{43}u_{34} = 2 - (-1)\left(-\frac{5}{13}\right) = \frac{21}{13}$	

(8)

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ -1 & \frac{5}{2} & 0 & 0 \\ 0 & -1 & \frac{13}{5} & 0 \\ 0 & 0 & -1 & \frac{21}{13} \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 1 & -\frac{2}{3} & 0 \\ 0 & 0 & 1 & -\frac{3}{4} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Sea $A = (a_{ij})$ una matriz simétrica de $n \times n$ a coeficientes reales. Diremos que A es definida positiva si satisface alguna de las siguientes condiciones equivalentes:

C_1) $x^T Ax > 0 \ \forall x \neq 0$

C_2) Todos los valores propios de A son positivos.

C_3) Todos los determinantes de las matrices cofactores principales son positivos.

La condición más eficiente de verificar es la C_3. Por ejemplo, para la matriz de Pascal de 4×4 :

$$\det[A_{11}] = \det(1) = 1 \qquad \det(A_{22}) = \det\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} = 1$$

$$\det(A_{33}) = \det\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix} = 1 \quad \det(A_{44}) = \det\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix} = 1 \tag{10}$$

Toda matriz simétrica y definida positiva posee una única factorización de la forma:

$$A = R^T R \tag{11}$$

donde R es una matriz triangular superior. Para calcular R se aplica el método de Cholesky:

Paso 1: Inicialmente $r_{11} = \sqrt{a_{11}}$

Para $j = 2, ..., n$

$r_{1j} = a_{1j}/r_{11}$

Paso 2: Para $i = 2, ..., n-1$:

$$r_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} r_{ki}^2\right)^{1/2}$$

Para $j = i+1, ..., n$:

$$r_{ij} = \frac{\left(a_{ij} - \sum_{k=1}^{i-1} r_{kj} r_{ki}\right)}{r_{ii}}$$

Paso 3: Finalmente:

$$r_{nn} = \left(a_{nn} - \sum_{k=1}^{n-1} r_{kn}^2\right)^{1/2}$$

La principal utilidad de esta factorización es el ahorro del cálculo de los coeficientes de la matriz triangular inferior L de la factorización $PA = LU$.

Toda matriz rectangular $A = (a_{ij})$ de $m \times n$ a coeficientes reales, donde $m \geq n$, admite una factorización:

$$A = QR = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{12} & q_{22} & \cdots & q_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{1m} & q_{2m} & \cdots & q_{mm} \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & r_{nn} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

donde Q es una matriz ortonormal de $m \times m$: $Q^T Q = I$ y R es una matriz trapezoidal superior de $m \times n$ con filas a coeficientes nulos a partir de la fila $(n+1)$. Por ejemplo:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \\ 1 & 4 & 10 \end{bmatrix} = \begin{bmatrix} -0.5 & 0.6708 & 0.5 & 0.2236 \\ -0.5 & 0.2236 & -0.5 & -0.6708 \\ -0.5 & -0.2236 & -0.5 & 0.6708 \\ -0.5 & -0.6708 & 0.5 & -0.2236 \end{bmatrix} \begin{bmatrix} -2 & -5 & -10 \\ 0 & -0.2361 & -6.7082 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{12}$$

Es posible generar una versión reducida de la factorización QR: Sea $A = (a_{ij})$ una matriz rectangular de $m \times n$ a coeficientes reales, de rango n. Sea $A = QR$ la factorización de esta matriz. Existe entonces una única factorización de A de la forma:

$$A = \widetilde{Q}\widetilde{R} \tag{13}$$

donde $\widetilde{Q}, \widetilde{R}$ son las submatrices de Q, R definidas por:

$$\widetilde{Q} = Q(1:m, 1:n) \qquad \widetilde{R} = R(1:n, 1:n) \tag{14}$$

Más aún, la matriz $\widetilde{Q}$ tiene columnas ortonormales y $\widetilde{R}$ es una matriz triangular superior que coincide con la matriz de Cholesky de la matriz simétrica y definida positiva $A^T A$, es decir $A^T A = \widetilde{R}^T \widetilde{R}$. Siguiendo con el ejemplo anterior:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \\ 1 & 4 & 10 \end{bmatrix} = \begin{bmatrix} -0.5 & 0.6708 & 0.5 & 0.2236 \\ -0.5 & 0.2236 & -0.5 & -0.6708 \\ -0.5 & -0.2236 & -0.5 & 0.6708 \\ -0.5 & -0.6708 & 0.5 & -0.2236 \end{bmatrix} \begin{bmatrix} -2 & -5 & -10 \\ 0 & -0.2361 & -6.7082 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \\ 1 & 4 & 10 \end{bmatrix} = \begin{bmatrix} -0.5 & 0.6708 & 0.5 \\ -0.5 & 0.2236 & -0.5 \\ -0.5 & -0.2236 & -0.5 \\ -0.5 & -0.6708 & 0.5 \end{bmatrix} \begin{bmatrix} -2 & -5 & -10 \\ 0 & -0.2361 & -6.7082 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \\ 1 & 4 & 10 \end{bmatrix} = \begin{bmatrix} 4 & 10 & 20 \\ 10 & 30 & 65 \\ 20 & 65 & 146 \end{bmatrix} = \begin{bmatrix} -2 & 0 & 0 \\ -5 & -0.236\,1 & 0 \\ -10 & -6.\,708\,2 & 1 \end{bmatrix} \begin{bmatrix} -2 & -5 & -10 \\ 0 & -0.2361 & -6.7082 \\ 0 & 0 & 1 \end{bmatrix}$$

Las principales aplicaciones de esta factorización están en el cálculo de valores y vectores propios y en la resolución de sistemas $Ax = b$ sobre-determinados via método de mínimos cuadrados:

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|^2 = \|QRx-b\|^2 = \left\|Rx-Q^Tb\right\|^2 = \left\|\widetilde{R}x-\widetilde{Q}^Tb\right\|^2 + \sum_{i=(n+1)}^{m}\left[\left(Q^Tb\right)_i\right]^2 \tag{16}$$

donde la segunda igualdad se obtiene por la propiedad de preservación de norma de la matrices ortonormales. Luego $x \in \mathbb{R}^n$ es solución de $\min_{x\in\mathbb{R}^n} \|Ax-b\|^2$ si y sólo si: $x = \widetilde{R}^{-1}\widetilde{Q}^Tb$.

Para calcular las matrices $\widetilde{Q}, \widetilde{R}$ se puede utilizar las matrices de transformacion de Givens o Householder, o bien el método de ortogonalización de Gram-Schmidt. Sean $a_k, \widetilde{q}_k$ las columnas k de la matrices $A, \widetilde{Q}$, $k = 1, ..., n$, respectivamente. Luego:

$$\begin{aligned}
\widetilde{q}_1 &= \frac{a_1}{\|a_1\|_2} \\
q_k &= a_k - \sum_{j=1}^{k-1}\left(\widetilde{q}_j^T a_k\right)\widetilde{q}_j^T \;\; k = 2, ..., n \\
\widetilde{q}_k &= \frac{q_k}{\|q_k\|_2} \;\; k = 2, ..., n
\end{aligned} \tag{17}$$

Una vez calculada $\widetilde{Q}$ se determina $\widetilde{R} = \widetilde{Q}^T A$. En la práctica se recomienda utilizar una versión modificada más estable ante propagación de errores del método de Gram-Schmidt. El cálculo en Matlab de la factorizacion modificada de $A = QR$ se realiza mediante los comandos:

$$\begin{aligned}
[Q, R, P] &= qr(A) \\
[Q_g, R_g, v] &= qr(A, 0)
\end{aligned} \tag{18}$$

donde:

$$\begin{aligned}
AP &= QR \\
A(\quad : \quad, v) &= Q_g R_g
\end{aligned} \tag{19}$$

Por ejemplo, para la matriz de pascal de 4×3 la factorización QR es de la forma:

```
>> A=pascal(4);
>> A=A(:,1:3)
A =
     1     1     1
     1     2     3
     1     3     6
     1     4    10
>> [Q,R,P]=qr(A)
Q =
   -0.0828   -0.7688    0.5934   -0.2236
   -0.2483   -0.5247   -0.4616    0.6708
   -0.4966   -0.1586   -0.5275   -0.6708
   -0.8276    0.3295    0.3956    0.2236
R =
  -12.0830   -1.6552   -5.3794
```

```
         0   -1.1226   -0.9762
         0         0   -0.3297
         0         0         0
P =
     0     1     0
     0     0     1
     1     0     0
```

En este caso la matriz P de permutación derecha genera una matriz R con diagonal decreciente en módulo.

```
>> [Qg,Rg,v]=qr(A,0)
Qg =
   -0.0828   -0.7688    0.5934
   -0.2483   -0.5247   -0.4616
   -0.4966   -0.1586   -0.5275
   -0.8276    0.3295    0.3956
Rg =
  -12.0830   -1.6552   -5.3794
         0   -1.1226   -0.9762
         0         0   -0.3297
v =
     3     1     2
```

En este caso se genera la versión reducida de la factorización $QR : A = \widetilde{Q}\widetilde{R}$. En el vector v se guarda la permutación de columnas de A, de manera que la matriz R quede con diagonal decreciente en módulo.

Toda matriz rectangular $A = (a_{ij})$ de $m \times n$ a coeficientes reales admite una factorización en valores singulares de la forma:

$$A = USV^T \tag{20}$$

donde U, V son matrices ortogonales de $m \times m, n \times n$ respectivamente y S es una matriz rectangular de $m \times n$ de la forma:

$$S = diag(\sigma_1, ..., \sigma_p) \qquad p = \min(m, n) \tag{21}$$

que contiene los valores singulares σ_i de A ordenados de mayor a menor, definidos según:

$$\sigma_i = \sqrt{\lambda_i(A^T A)} \tag{22}$$

Por ejemplo, para la matriz de pascal de 4×3 la factorización en valores singulares es de la forma:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \\ 1 & 4 & 10 \end{bmatrix} = \begin{bmatrix} 0.107\,8 & 0.6792 & 0.6907 & -0.2236 \\ 0.273\,9 & 0.5705 & -0.3866 & 0.6708 \\ 0.507\,8 & 0.2065 & -0.4995 & -0.6708 \\ 0.809\,6 & -0.4130 & 0.3521 & 0.2236 \end{bmatrix} \begin{bmatrix} 13.341 & 0 & 0 \\ 0 & 1.3997 & 0 \\ 0 & 0 & 0.2395 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.1274 & 0.4061 & 0.9049 \\ 0.7453 & 0.5628 & -0.3575 \\ 0.6544 & -0.7200 & 0.2310 \end{bmatrix} \tag{23}$$

Esta factorización tiene gran cantidad de propiedades y aplicaciones, ver ref. [5]. Algunas de ellas son:

P1) Si A es una matriz de $n \times n$, simétrica:

$$\sigma_i = \sqrt{\lambda_i(A^T A)} = \sqrt{\lambda_i(A^2)} = |\lambda_i(A)| \tag{24}$$

P2) $\|A\|_F^2 = \sum_{i=1}^{p} \sigma_i^2$

P3) $\|A\|_2 = \sigma_1$

P4) El $rango(A)$ es igual al número de valores singulares distintos de cero.

P5) El valor singular más pequeño es igual a la distancia en norma $\|\cdot\|_2$ a la matriz de rango deficiente más cercana a la matriz A.

Si $rango(A) = r \leq \min(m, n)$, la matriz $\widehat{A} = V\widehat{S}U^T$ de $n \times m$ se denomina la pseudo inversa de Moore - Penrose, donde $\widehat{S}$ es la matriz de $n \times m$ definida por:

$$\widehat{\Sigma} = diag(\tfrac{1}{\sigma_1}, \tfrac{1}{\sigma_2}, ..., \tfrac{1}{\sigma_r}, 0, ..., 0) \tag{25}$$

La matriz $\widehat{A}$ también es llamada la inversa generalizada de A. Si $rango(A) = n \leq m$ se tiene que $\widehat{A} = (A^TA)^{-1}A^T$. Luego, si $rango(A) = n = m \Rightarrow \widehat{A} = A^{-1}$.

En Matlab la factorización en valores singulares se calcula mediante los comandos:

$$\begin{aligned} [U, S, V] &= svd(A) \\ [U, S, V] &= svd(A, 0) \end{aligned} \tag{26}$$

donde U, V son matrices ortogonales de $m \times m, n \times n$ respectivamente y S es la matriz rectangular de $m \times n$ que contiene los valores singulares de A. La segunda opción del comando $[U, S, V] = svd(A, 0)$ produce una version reducida de la factorización, donde U, V son matrices ortogonales de $m \times n, n \times n$ respectivamente y S es una matriz cuadrada de $n \times n$. Por ejemplo, para la matriz de pascal de 4×3 las factorizaciones SVD standard y reducidas son de la forma:

```
>> [U,S,V]=svd(A)

U =
   -0.1078   -0.6792    0.6907    0.2236
   -0.2739   -0.5705   -0.3866   -0.6708
   -0.5078   -0.2065   -0.4995    0.6708
   -0.8096    0.4130    0.3521   -0.2236
S =
   13.3410         0         0
         0    1.3997         0
         0         0    0.2395
         0         0         0
V =
   -0.1274   -0.7453    0.6544
   -0.4061   -0.5628   -0.7200
   -0.9049    0.3574    0.2310

>> [U,S,V]=svd(A,0)
U =
   -0.1078   -0.6792    0.6907
   -0.2739   -0.5705   -0.3866
```

```
   -0.5078   -0.2065   -0.4995
   -0.8096    0.4130    0.3521
S =
   13.3410         0         0
         0    1.3997         0
         0         0    0.2395
V =
   -0.1274   -0.7453    0.6544
   -0.4061   -0.5628   -0.7200
   -0.9049    0.3574    0.2310
```

ACT1: Factorización de matrices:

La instrucción switch/case permite ejecutar diferentes instrucciones condicionalmente al valor de una variable real, entera o string. Su sintaxis es:

switch var

case valor1

..... //Instrucciones1 Matlab

case {valor2,valor3,valor4}

..... //Instrucciones2 Matlab

⋮

otherwise

..... //Instrucciones de Matlab

end

Si $var = valor1$ se ejecutan las Instrucciones1 y no se ejecutan las demás instrucciones; si $var = valor2, valor3, valor4$ se ejecutan las Instrucciones2 y no se ejecutan las demás instrucciones, y así sucesivamente. Si la variable var no es igual a los valores que definen los case, entonces se ejecutan las instrucciones de *otherwise*.

Utilizando la instrucción switch/case programe la función fact_matriz.m que calcula las factorizaciones de LU, Crout, Cholesky, QR, SVD y resuelve un SEL. De acuerdo a la factorización seleccionada debe recibir diferentes inputs, realizar algunos cálculos y entregar diferentes outputs.

1) Método LU:

   (a) Recibir como input una matriz A de tamaño $n \times n$ y un vector $b \in \mathbb{R}^n$.

   (b) Verificar que las dimensiones de A y b son correctas. Sino, se debe imprimir un mensaje de error y salir sin ejecutar el método.

   Hint: El comando *error* permite salir de una función sin ejecutarla debido a que se produjo algún tipo de error. Busque en el *Help* como utilizarlo.

   (c) Calcular la factorización $PA = LU$ mediante el comando $[L, U, P] = lu(A)$. Calcular la solución del sistema $Ax = b$ aplicando la factorización.

   (d) El output de este método deben ser las matrices L, U, P, la solución del SEL, el error de la solución y el tiempo de *cpu*.

2) Método de Crout:

(a) Recibir como input 4 vectores b, c, d, e.

(b) Verificar que $d, b \in \mathbb{R}^n$; $c, e \in \mathbb{R}^{n-1}$

Verificar que todos los coeficientes de c, d, e son diferentes de cero.

Verificar la condición $|a_{ii}| > |a_{i,i+1}| + |a_{i+1,i}| \ \forall i = 1, ..., n-1$.

Si alguna de estas condiciones no se cumple se debe imprimir un mensaje de error y salir sin ejecutar el método.

(c) Generar una matriz tridiagonal con c como diagonal secundaria inferior, d como diagonal principal y e como diagonal secundaria superior. Es decir:

$$A = \begin{bmatrix} d_1 & e_1 & 0 & \cdots & 0 \\ c_1 & d_2 & e_2 & \ddots & \vdots \\ 0 & c_2 & d_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & e_{(n-1)} \\ 0 & \cdots & 0 & c_{(n-1)} & d_n \end{bmatrix} \tag{27}$$

(d) Implementar el método de Crout, calculando las matrices L, U.

(e) Resolver el sistema $Ax = b$ aplicando la factorización.

(f) El output de este método deben ser las matrices L, U, la solución del SEL, el error de la solución y el tiempo de *cpu*.

Hint: Para generar en Matlab una matriz tridiagonal, puede utilizar el comando:

$$A = full(gallery('tridiag', c, d, e)) \tag{28}$$

donde c es la diagonal secundaria inferior, d la diagonal principal, e es la diagonal secundaria superior. Los vectores c, e deben tener una componente menos que d. En Matlab las matrices tridiagonales son almacenadas en forma sparse dada la gran cantidad de coeficientes nulos que poseen. El comando $full$ transforma la matriz A de sparse a standard.

Por ejemplo:

```
>> d  = [1 2 3 4 5];
>> c  = [-1 -1 -1 -1];
>> e  = [-2 -2 -2 -2];
>> A=gallery('tridiag',c,d,e)
A =
   (1,1)        1
   (2,1)       -1
   (1,2)       -2
   (2,2)        2
   (3,2)       -1
   (2,3)       -2
   (3,3)        3
   (4,3)       -1
   (3,4)       -2
```

```
   (4,4)        4
   (5,4)       -1
   (4,5)       -2
   (5,5)        5
>> A=full(gallery('tridiag',c,d,e))
A =
     1    -2     0     0     0
    -1     2    -2     0     0
     0    -1     3    -2     0
     0     0    -1     4    -2
     0     0     0    -1     5
```

3) Método de Cholesky:

   (a) Recibir como input una matriz A de tamaño $n \times n$ y un vector $b \in \mathbb{R}^n$.

   (b) Verificar que A es simétrica y definida positiva. Si la matriz A no verifica alguna de estas condiciones se debe imprimir un mensaje de error y salir sin ejecutar el método. Para determinar si A es definida positiva se debe utilizar la condición C_3.

   (c) Verificar que las dimensiones de A y b son correctas. Sino, se debe imprimir un mensaje de error y salir sin ejecutar el método.

   (d) Calcular la matriz R mediante el comando $R = chol(A)$. Calcular la solución del sistema $Ax = b$ aplicando la factorización.

   (e) El output de este método deben ser la matriz R, la solución del SEL, el error de la solución y el tiempo de *cpu*.

4) Método QR:

   (a) Recibir como input una matriz A de tamaño $m \times n$ y un vector $b \in \mathbb{R}^m$.

   (b) Verificar que las dimensiones de A y b son correctas. Sino, se debe imprimir un mensaje de error y salir sin ejecutar el método.

   (c) Calcular la factorización QR mediante el comando $[Q, R, P] = qr(A)$. Calcular la solución del sistema $Ax = b$ aplicando la factorización.

   (d) El output de este método deben ser las matrices Q, R, P, la solución del SEL, el error de la solución y el tiempo de *cpu*.

5) Método SVD:

   (a) Recibir como input una matriz A de tamaño $m \times n$ y un vector $b \in \mathbb{R}^m$.

   (b) Verificar que las dimensiones de A y b son correctas. Sino, se debe imprimir un mensaje de error y salir sin ejecutar el método.

   (c) Calcular la factorización SVD mediante el comando $[U, S, V] = svd(A)$. Calcular la solución del sistema $Ax = b$ aplicando la factorización.

   (d) El output de este método deben ser las matrices U, S, V, la solución del SEL, el error de la solución y el tiempo de *cpu*.

Hint: En Matlab es posible construir funciones que tienen un número variable de inputs y outputs mediante las instrucciones:

$$varargin, varargout \tag{29}$$

las cuales son un cell array que contienen todos los argumentos de input y output. Veamos un ejemplo de como usarlos:

```
function varargout=funct_var_arg(varargin)
n=nargin;
switch n
    case 1
        x=varargin{1}; y=exp(sin(x));
        varargout{1}=y; varargout{2}=length(y); varargout{3}=plot(x,y,'-b');
    case 3
        x=varargin{1}; y=varargin{2}; s=varargin{3};
        varargout{1}=plot(x,y,s);
    otherwise
        error('Numero de parametros incorrecto');
end
end
```

Los parámetros $nargin, nargout$ definen el número de argumentos de input y output. Se puede llamar a la función funct_var_arg.m de varias formas, por ejemplo:

```
>> x=[0:0.2:2];
>> [y,size_y,h]=funct_var_arg(x)
y =
  Columns 1 through 11
    1.0000  1.2198  1.4761  1.7588  2.0490  2.3198  2.5397  2.6790   2.7171   2.6481   2.4826
size_y = 11
h = 158.0135
>> x=[0:0.2:2]; y=x.*exp(-x.^2);
>> [h]=funct_var_arg(x,y,'-r')
h = 158.0143
```

2 Métodos para valores y vectores propios

El siguiente teorema, ver ref. [7], es el resultado principal del algebra lineal sobre el que se fundamentan una serie de métodos para SEL.

Teorema 1 *Sea $A \in \mathbb{R}^{n\times n}$. Las siguientes afirmaciones son equivalentes:*

a) A es no singular.

b) $\det(A) \neq 0$.

c) El sistema lineal $Ax = 0$ tiene solución única $x = 0$.

d) Para cualquier $b \in \mathbb{R}^n$ el sistema lineal $Ax = b$ tiene solución única.

e) Las columnas y filas de A son linealmente independientes.

La afirmación e) puede ser reformulada de la siguiente forma: A tiene rango n (número de columnas linealmente independientes).

Definición 2 *Sea $A \in \mathbb{R}^{n\times n}$. Un escalar $\lambda \in \mathbb{R}$ y un vector $x \in \mathbb{R}^n$, $x \neq 0$, se denominan valor y vector propio de A si satisfacen la ecuación:*

$$Ax = \lambda x \tag{30}$$

Por el teorema anterior concluimos que λ es un valor propio ssi:

$$\det(A - \lambda I) = 0 \tag{31}$$

que se denomina la ecuación característica de A y que define el polinomio característico:

$$p(\lambda) = \det(A - \lambda I) = 0 \Longrightarrow p(\lambda) = \sum_{i=0}^{n} c_i \lambda^i$$

Entonces A tiene n valores propios, no necesariamente distintos, que son las raíces del polinomio característico. El conjunto de valores propios de A $\sigma(A) = \{\lambda_1, \lambda_2, ..., \lambda_n\}$ es el espectro de A y su radio espectral se define según:

$$\rho(A) = \max_{i=1,...,n} |\lambda_i| \tag{32}$$

El siguiente teorema, ver refs. [7], resume las principales propiedades asociadas a los valores y vectores propios de una matriz A:

Teorema 3 *Propiedades de los valores propios $\lambda_1, \lambda_2, ..., \lambda_n$ de $A \in \mathbb{R}^{n\times n}$:*

1) Los valores propios de una matriz A triangular superior o inferior son los coeficientes de la diagonal.

2) El determinante de A es el producto de sus valores propios:

$$\det(A) = \lambda_1 \lambda_2 \cdots \lambda_n \tag{33}$$

3) La matriz A es singular ssi tiene un valor propio nulo.

4) La suma de la diagonal de A (definida como la traza) es igual a la suma de los valores propios:

$$\sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i \tag{34}$$

5) Toda matriz A con valores propios reales $\{\lambda_1, ..., \lambda_r\}, r \leq n$, no necesariamente distintos con multiplicidades m_i tal que $\sum_{i=1}^{r} m_i = n$ es similar a una forma de Jordan:

$$VAV^{-1} = J = \begin{bmatrix} J_1 & 0 & \cdots & 0 \\ 0 & J_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & J_r \end{bmatrix} \tag{35}$$

donde cada bloque J_i es de tamaño $m_i \times m_i$ y tiene la siguiente estructura:

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 1 \\ 0 & 0 & \cdots & 0 & \lambda_i \end{bmatrix} \tag{36}$$

6) Si A es una matriz simétrica a coeficientes reales:

(a) Los valores y vectores propios son reales.

(b) Los vectores propios asociados a valores propios distintos son ortogonales y por lo tanto A es similar a la matriz diagonal D con los valores propios de A:

$$V^{-1}AV = V^TAV = D = diag(\lambda_1, \lambda_2, ..., \lambda_n) \tag{37}$$

En Matlab se pueden calcular los valores y vectores propios con los comandos:

$$\begin{aligned} [V, D] &= eig(A) \\ [V, D] &= eig(A, B) \end{aligned} \tag{38}$$

En la salida V se entregan los vectores propios con norma 1 (cuál es esta norma ?) y en D una matriz diagonal con los valores propios tal que satisfacen las ecuaciones:

$$\begin{aligned} AV &= VD \\ AV &= BVD \end{aligned} \tag{39}$$

También es posible calcular el polinomio característico mediante el comando:

$$c = poly(A) \tag{40}$$

La salida de este comando es el vector de coeficientes en orden decreciente de acuerdo a la potencia y tal que $c_n = 1$:

$$c = [1 \ c_{n-1} \ c_{n-2} \ ... \ c_0] \tag{41}$$

Luego se pueden calcular las raíces de c con el comando:

$$r = roots(c) \tag{42}$$

El algoritmo para calcular $c = poly(A)$ está dado por:

```
z = eig(A);
c = zeros(n+1,1); c(1) = 1;
for j = 1:n
    c(2:j+1) = c(2:j+1)-z(j)*c(1:j);
end
```

El teorema de Abel establece que no existe una fórmula analítica para calcular las raíces de un polinomio de grado mayor o igual a 5. Por esta razón el comando $roots(c)$ calcula las raíces de c como los valores propios de la matriz compañera de c:

$$A = compan(c) \tag{43}$$

mediante el algoritmo:

```
n=length(c)-1;
A = diag(ones(n-1,1),-1);
A(1,:) = -c(2:n+1)./c(1);
eig(A);
```

donde n es el grado del polinomio c.

Los métodos numéricos que se utilizan para calcular valores y vectores propios se dividen en métodos parciales (Potencia, Potencia Inversa), que permiten calcular los valores y vectores propios extremos, y globales (QR, QR modificado con Householder y Givens), que permiten calcular todos los valores y vectores propios.

El método de la Potencia permite aproximar los valores propios de mayor y menor módulo: λ_1, λ_n (respectivamente), y sus vectores propios asociados. Este problema es de gran interés en varias aplicaciones: vibraciones en máquinas y estructuras, análisis de redes electricas, etc. Calcular los valores propios de mayor y menor módulo también tiene aplicaciones en Análisis Numérico.

Sea A una matriz de $n \times n$ diagonalizable, es decir similar a la matriz diagonal D que contiene sus valores propios:

$$V^{-1}AV = D \tag{44}$$

donde las columnas de V son los vectores propios de A. Supongamos que los valores propios de A son de la forma:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \tag{45}$$

donde λ_1 tiene multiplicidad algebraica igual a 1. La siguiente iteración del método de la potencia calcula aproximadamente λ_1 y v_1 :

Etapa 0:	Seleccionar un punto inicial $v^0 \in \mathbb{R}^n$ tal que:
	$\left\|v^0\right\|_2 = 1$
	$\lambda^0 = \left(v^0\right)^T Av^0$
	$k = 1$
Etapa 1:	Calcular
	$z^k = Av^{k-1}$
	$v^k = \frac{z^k}{\|z^k\|_2}$
	$\lambda^k = \left(v^k\right)^T Av^k$
	Si $\left\|\lambda^{k+1} - \lambda^k\right\| \leq Tol \wedge \left\|v^{k+1} - v^k\right\| \leq Tol \Rightarrow STOP$
	Si no $k = k + 1$ y volver a iterar

Método de la Potencia.

Si A es simétrica se tiene que:

$$\left|\lambda_1 - \lambda^k\right| \leq |\lambda_1 - \lambda_n| \tan^2(\theta_0) \left|\frac{\lambda_2}{\lambda_1}\right|^{2k} \tag{46}$$

donde $\cos(\theta_0) = \left|v_1^T v^0\right|$. Esta desigualdad establece que la velocidad de convergencia del método de la potencia es cuadrática con respecto al radio $\left|\frac{\lambda_2}{\lambda_1}\right|$.

Para calcular el valor propio λ_m que está más cercano a un número dado $\mu \notin \sigma(A) = \{\lambda_1, ..., \lambda_n\}$ se puede utilizar el método de la potencia inversa. Supongamos que $\exists \lambda_m \in \sigma(A)$, tal que:

$$|\lambda_m - \mu| < |\lambda_i - \mu| \ \ \forall i = 1, ..., n; \ i \neq m \tag{47}$$

La siguiente iteración del método de la potencia inversa calcula aproximadamente λ_m y v_m :

Etapa 0:	Seleccionar un punto inicial $v^0 \in \mathbb{R}^n$ tal que:
	$\left\|v^0\right\|_2 = 1$
	$\lambda^0 = \left(v^0\right)^T A v^0$
	$k = 1$
Etapa 1:	Calcular:
	$Az^k = v^{k-1}$
	$v^k = \dfrac{z^k}{\left\|z^k\right\|_2}$
	$\lambda^k = \left(v^k\right)^T A v^k$
	Si $\left\|\lambda^{k+1} - \lambda^k\right\| \leq Tol \wedge \left\|v^{k+1} - v^k\right\| \leq Tol \Rightarrow STOP$
	Si no $k = k + 1$ y volver a iterar

Método de la Potencia Inversa.

En el caso que $\mu = 0$ se tiene que $\lambda_m = \lambda_n, v_m = v_n$. Para resolver $Az^k = v^{k-1}$ en forma eficiente, se puede escoger una factorización adecuada de A. Las matrices de esta descomposición se calculan sólo una vez.

Utilizando la factorización QR es posible calcular todos los valores y vectores propios de una matriz A, reduciéndola mediante transformaciones similares adecuadas a una matriz en la cual son más fáciles de calcular. El teorema de la descomposición de Schur, ver ref. [5], establece que toda matriz $A \in \mathbb{C}^{n \times n}$ es similar a una matriz triangular superior T via matrices de transformación unitarias U:

$$U^{-1}AU = U^H AU = T = \begin{bmatrix} \lambda_1 & t_{12} & \cdots & t_{1n} \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & t_{(n-1),n} \\ 0 & 0 & 0 & \lambda_n \end{bmatrix} \tag{48}$$

En el caso de matrices a coeficientes reales la descomposición de Schur genera una matriz T triangular superior con bloques de tamaño 1 o 2. En Matlab se calcula esta factorizacion con el comando:

$$[U, T] = schur(A) \tag{49}$$

```
>> A=[3 17 -37 18 -40;1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0];

>> [U,T]=schur(A)
```

```
U =
   -0.9798    0.1936    0.0437   -0.0139    0.0199
   -0.1960   -0.9779   -0.0244    0.0554   -0.0398
   -0.0392    0.0586   -0.8818    0.2079   -0.4175
   -0.0078   -0.0518   -0.3997   -0.8039    0.4374
   -0.0016    0.0055   -0.2454    0.5544    0.7952

T =
    5.0000   18.9738  -34.2569   42.5583    8.9218
         0   -4.0000    6.7699   -8.4097   -1.7610
         0         0    2.0000   -1.8340   -0.3041
         0         0         0   -0.0000    0.6971
         0         0         0   -1.4345   -0.0000

>> eig(A)
ans =
   5.0000
  -4.0000
   2.0000
   0.0000 + 1.0000i
   0.0000 - 1.0000i

>> eig(T)
ans =
   5.0000
  -4.0000
   2.0000
  -0.0000 + 1.0000i
  -0.0000 - 1.0000i
```

Etapa 0:	Seleccionar $Q^0 \in \mathbb{R}^{n\times n}$ matriz ortogonal.
	$T^0 = \left(Q^0\right)^T AQ^0$
	$k = 1$
Etapa 1:	Calcular factorización QR de:
	$Q^k R^k = T^{k-1}$
	$T^k = R^k Q^k$
	Si $\left\|T^{k+1} - T^k\right\| \leq Tol \Rightarrow STOP$
	Si no $k = k + 1$ y volver a iterar

Iteración QR para Factorización de Schur.

Lamentablemente, como consecuencia del Teorema de Abel esta factorización no puede ser calculada directamente, pero si numéricamente mediante la iteración QR anterior. La convergencia de $T^k \underset{k\to\infty}{\longrightarrow} T$ (ver ecuación 48) está asegurada con la condición: $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, con tasa de convergencia:

$$\left|t^k_{i,(i-1)}\right| = O\left(\left|\frac{\lambda_i}{\lambda_{i-1}}\right|^k\right) \quad \forall i = 2, ..., n;\ k \to \infty \tag{50}$$

Si adicionalmente A es simétrica la secuencia T^k converge a la matriz diagonal de valores propios de A.

ACT2: Aplicación valores y vectores propios: Vibración dinámica libre en un puente, ver ref. [8].

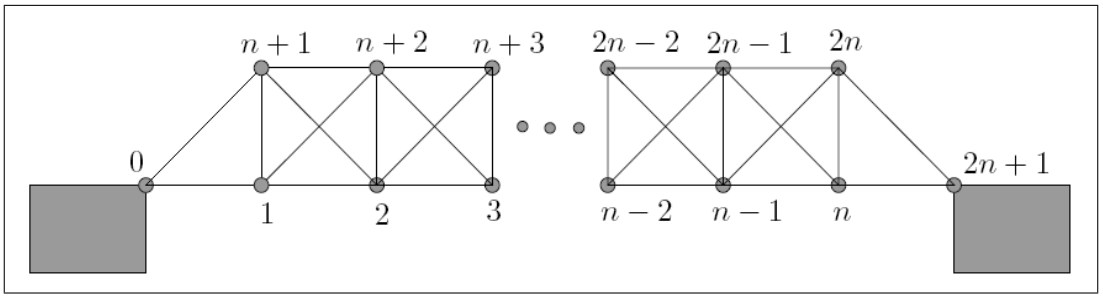

Figura 1. Estructura esquemática de un puente.

Consideremos un puente con estructura esquemática de acuerdo a la figura 1, formado por $2n$ nodos y $5n$ vigas. Los nodos 0 y $(2n+1)$ están fijos a tierra. Cada viga horizontal y vertical tiene una masa m y las vigas diagonales masa $\sqrt{2}m$. La rigidez de cada viga se representa mediante una constante de resorte $\kappa > 0$. Sean $x, y \in \mathbb{R}^{2n}$ los vectores de desplazamientos horizontales y verticales, respectivamente. La respuesta libre del puente puede ser estudiada resolviendo el siguiente problema de valores propios generalizados:

$$\begin{aligned} Mx &= \lambda Kx \qquad My = \lambda Ky \\ M &= m \cdot diag_{2n}(\alpha, b, \alpha, \gamma, b, \gamma) \\ K &= \kappa \begin{bmatrix} K_{11} & K_{12} \\ K_{12} & K_{11} \end{bmatrix} \end{aligned} \tag{51}$$

donde: $\alpha = 3+\sqrt{2}, b = (\beta, \beta, ..., \beta)^T \in \mathbb{R}^{n-2}, \beta = \frac{3}{2}+\sqrt{2}, \gamma = 1+\sqrt{2}, K_{12} = tridiag_n(-1,-1,-1)$, $K_{11} = tridiag_n(-1, d, -1)$, $d = (4,5,5,...,5,4)^T \in \mathbb{R}^n$. M es la matriz de masa y K la matriz de rigidez. Sean (λ_k, v_k) $k = 1, ..., 2n$ los pares valor y vector propio del problema (51). En el diseño de una estructura es de fundamental importancia calcular las frecuencias naturales $\omega_k = \sqrt{\lambda_k}$ y los modos de vibración v_k. Si la frecuencia de excitación de una fuerza externa (vehículos, viento, terremoto) coincide con una frecuencia natural de una extructura se produce resonancia y como consecuencia pueden aparecer grandes oscilaciones.

Programe la función vibracion_dinamica.m que calcula las frecuencias naturales ω_k y los modos de vibración v_k. Esta función debe recibir como input los parámetros n, m, κ y resolver el problema (51) de la siguiente forma:

1) Construir las matrices M, K. Verifique que K es simétrica y definida positiva.

2) Realice un cambio de variables de manera que el problema (51) quede como un problema de valores propios standard: $Cz = \mu z$.

3) Aplicando 2) calcular los valores y vectores propios del problema del problema (51), el polinomio característico c y la matriz compañera de c, utilizando los comandos de Matlab: $eig, poly, compan$. Adicionalmente calcular el error de los valores y vectores propios y el tiempo de *cpu*.

4) Aplicando 2) calcular los valores y vectores propios del problema (51) utilizando la iteración QR para la factorización de Schur. Adicionalmente calcular el error de los valores y vectores propios y el tiempo de *cpu*.

5) Comparar las soluciones calculadas en 3) y 4) en cuanto a precisión y tiempo de cálculo.

6) Ejemplifique sus cálculos para el caso $n = 5, m = 10, \kappa = 1$. Finalmente, grafique la deformación que ocurre en la estructura para alguna de las frecuencias naturales ω_k y los modos de vibración v_k calculados. Compare esta deformación con la estructura en reposo.

3 Métodos iterativos para SEL

El tema del laboratorio 1 fue resolver SEL mediante el método directo de Gauss, cuya característica principal es determinar la solución de $Ax = b$ realizando una cantidad finita de operaciones de orden $O(n^3)$. Si el computador que se está utilizando fuera capaz de operar con aritmética de infinita precisión, es decir sin errores de redondeo, la solución encontrada sería exacta.

En el caso que estemos resolviendo un SEL de gran tamaño, el orden de operaciones de Gauss se convierte en un gran obstáculo. Los método iterativos o indirectos comienzan de un punto inicial x^0 y lo mejoran sucesivamente aplicando una iteración de punto fijo de la forma:

$$\begin{aligned} x^{k+1} &= F(x^k) = Bx^k + h \\ x^k &= \begin{bmatrix} x_1^k \\ \vdots \\ x_n^k \end{bmatrix} \end{aligned} \tag{52}$$

hasta que el cambio entre 2 iteraciones consecutivas es menor que la tolerancia:

$$\left\| x^{k+1} - x^k \right\| < Tol \tag{53}$$

Las constantes $B \in \mathbb{R}^{n \times n}, h \in \mathbb{R}^n$ dependen del método utilizado.
Una iteración de un método indirecto realiza una cantidad de operaciones $O(n^2)$, un orden de magnitud menor que el método de Gauss. Luego, la conveniencia de aplicarlos se define por la cantidad de iteraciones que realiza.

A pesar que la convergencia es lenta en algunos casos, los métodos iterativos tienen las siguientes ventajas:

1) Se pueden implementar utilizando almacenamiento eficiente de matrices sparse. Es posible entonces aplicarlos a sistemas sparse de gran tamaño no necesariamente de estructura de banda, como las matrices tridiagonales.

2) Auto-corrigen errores, es decir, los errores de redondeo en una iteración son corregidos en las iteraciones siguientes.

Adicionalmente, en los últimos 30 años han surgido las técnicas de sobre-relajación que mejoran la tasa de convergencia de los metodos iterativos clásicos: Jacobi y Gauss-Seidel.

El método de Jacobi se define mediante la iteración:

$$x_i^{k+1} = \frac{b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^k}{a_{ii}} \quad i = 1, ..., n \quad k = 0, 1, 2, ... \tag{54}$$

El método de Gauss-Seidel se define mediante la iteración:

$$x_i^{k+1} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k}{a_{ii}} \quad i = 1, ..., n \quad k = 0, 1, 2, ... \tag{55}$$

La forma matricial está dada por:

$$\begin{aligned} x^{k+1} &= B_J x^k + h_J \\ x^{k+1} &= B_{GS} x^k + h_{GS} \end{aligned} \tag{56}$$

donde:

$$\begin{aligned} B_J &= -d(A)^{-1}[l(A)+u(A)] \qquad h_J = d(A)^{-1}b \\ B_{GS} &= -[d(A)+l(A)]^{-1}u(A) \qquad h_{GS} = [d(A)+l(A)]^{-1}b \end{aligned} \tag{57}$$

y a su vez:

$$\begin{aligned} (d(A))_{ij} &= \begin{cases} a_{ii} & \text{si } i=j \\ 0 & \text{en otro caso} \end{cases} \\ (l(A))_{ij} &= \begin{cases} a_{ij} & \text{si } i>j \\ 0 & \text{en otro caso} \end{cases} \\ (u(A))_{ij} &= \begin{cases} a_{ij} & \text{si } i<j \\ 0 & \text{en otro caso} \end{cases} \end{aligned} \tag{58}$$

Estos métodos convergen $\forall x^0 \in \mathbb{R}^n$ si A es estrictamente diagonal dominante:

$$|a_{kk}| > \sum_{j=1, j\neq k}^{n} |a_{kj}| \ \ \forall k = 1, ..., n \tag{59}$$

La rapidez de convergencia comparativa de Jacobi y Gauss-Seidel se obtiene de las siguiente proposiciones, ver ref. [7]:

Proposición 4 *Si x^k es la iteración k de un método iterativo para un SEL $Ax = b$ que tiene la forma:*

$$x^{k+1} = Bx^k + h \tag{60}$$

Entonces:

$$x^k \underset{k\to\infty}{\longrightarrow} x \ \ ssi \ \rho(B) < 1 \tag{61}$$

donde $\rho(B)$ es el radio espectral de B.

Proposición 5 *Si x^k es la iteración k de un método iterativo para un SEL $Ax = b$ que tiene la forma:*

$$x^{k+1} = Bx^k + h \tag{62}$$

Entonces:

$$\left\|x^{k+1} - x\right\| \approx \rho(B)^k \left\|x^0 - x\right\| \tag{63}$$

Proposición 6 *Si los métodos de Jacobi y Gauss-Seidel convergen se tiene que:*

$$0 < \rho(B_{GS}) < \rho(B_J) < 1 \tag{64}$$

Se puede mejorar la rapidez de convergencia de Gauss-Seidel aplicando una técnica de sobre-relajación (SOR) definida mediante la iteración:

$$\begin{aligned} x_i^{k+1} &= (1-\omega)x_i^k + \omega \frac{\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k\right)}{a_{ii}} \quad i = 1, ..., n \quad k = 0, 1, 2, ... \\ x^{k+1} &= (1-\omega)x^k + \omega\left(B_{GS}x^k + h_{GS}\right) \end{aligned} \tag{65}$$

para $\omega > 1$. El valor óptimo del parámetro ω está dado aproximadamente por:

$$\begin{aligned} \omega_{opt} &\approx \frac{2}{1+\sqrt{1-\left(\frac{\Delta x^{(k+p)}}{\Delta x^{(k)}}\right)^{1/p}}} \\ \Delta x^{(k)} &= \left\|x^k - x^{k-1}\right\| \end{aligned} \tag{66}$$

ver refs. [1, 4, 7]. El parámetro ω_{opt} se puede estimar haciendo $k = 10$ iteraciones con $\omega = 1$ y luego $p = 1$ iteraciones más. Se calcula ω_{opt} y se deja fijo para el resto de las iteraciones.

Para una matriz diagonal dominante, simétrica y definida positiva, el valor óptimo de ω_{opt} está dado por la fórmula:

$$\omega_{opt} = \frac{2}{1+\sqrt{1-\rho(B_{GS})}} \tag{67}$$

Por ejemplo:

$$\begin{aligned} A &= \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \\ B_{GS} &= -\begin{bmatrix} 2 & 0 & 0 & 0 \\ -1 & 3 & 0 & 0 \\ 0 & -1 & 3 & 0 \\ 0 & 0 & -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{6} & \frac{1}{3} & 0 \\ 0 & \frac{1}{18} & \frac{1}{9} & \frac{1}{3} \\ 0 & \frac{1}{36} & \frac{1}{18} & \frac{1}{6} \end{bmatrix} \\ \omega_{opt} &= \frac{2}{1+\sqrt{1-\rho(B_{GS})}} = \frac{2}{1+\sqrt{1-0.3692}} \approx 1.1147 \end{aligned} \tag{68}$$

Se puede resolver un SEL $Qx = b$ aplicando un método de optimización sin restricciones. Sea (PC) el problema de optimización cuadrático definido por:

$$\min_{x\in\mathbb{R}^n} q(x) = \tfrac{1}{2}x^tQx - b^tx \tag{69}$$

donde Q es una matriz simétrica y definida positiva. La solución única del problema anterior está dada por:

$$Qx = b \tag{70}$$

Para disminuir la cantidad de cálculo de los métodos directos (cuyo costo es $O(n^3)$) se utilizan técnicas provenientes de la optimización. Los métodos que realizan menor cantidad de cálculo son gradiente y gradiente conjugado. La iteración del método de gradiente para el problema (69) está dada por:

Etapa 0:	Seleccionar un punto inicial $x^0 \in \mathbb{R}^n$
	Definir $k = 0$
Etapa 1:	Calcular:
	$d^k = -Qx^k + b$
	Si $\|d^k\| \leq Tol \Longrightarrow STOP.$
	Si no, seguir a Etapa 2.
Etapa 2:	Calcular α^k paso:
	$\alpha^k = \dfrac{(d^k)^t d^k}{(d^k)^t Q d^k}$
	Calcular: $x^{k+1} = x^k + \alpha^k d^k$
	Definir $k = k + 1$ y volver a Etapa 1.

(71)

Algoritmo 1. Método del Gradiente.

Las características principales del método de gradiente son:

P1) Convergencia global: Para cualquier punto inicial la iteración converge al óptimo del problema (PC).

P2) La rapidez de convergencia del método es lineal: Si x^k es la sucesión de puntos determinados por el método del que:

$$\left\|x^{k+1} - x\right\|_Q \leq \frac{k_2(Q)-1}{k_2(Q)+1} \left\|x^k - x\right\|_Q \quad \forall k \geq 0 \tag{72}$$

donde x es la solución del problema (PC).

P3) Dos direcciones consecutivas generadas por el método del gradiente son ortogonales. Esta característica puede originar lentitud de convergencia.

El método de gradiente conjugado tiene también convergencia global, pero además mejora significativamente la rapidez lineal de convergencia, debido a que las direcciones de descenso que va construyendo son Q ortogonales 2 a 2. Luego, si los cálculos son desarrollados con precisión aritmética infinita, realiza a lo más n iteraciones.

Definición 7 *Un conjunto de direcciones $\{d^1, ..., d^m\}$ de $\mathbb{R}^n$, no nulas y tal que $m \leq n$, se dirá Q-conjugado 2 a 2 si:*

$$(d^i)^t Q d^j = \delta_{ij} \qquad i, j = 1, ..., m \qquad \text{donde } \delta_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{en otro caso} \end{cases}$$

En el caso $Q = I$, las direcciones son ortonormales.

Proposición 8 *Si $D = \{d^1, ..., d^m\}$ es un conjunto Q-conjugado, entonces D es de rango máximo, i.e. los vectores d^i son linealmente independientes.*

Demostración. *En efecto, sean $\alpha^i \in \mathbb{R}$; $i = 1, ..., m$, tal que:*

$$\sum_{i=1}^{m} \alpha^i d^i = 0$$

premultiplicando por $(d^j)^t Q \neq 0$:

$$\sum_{i=1}^{m} \alpha^i (d^j)^t Q d^i = \sum_{i=1}^{m} \alpha^i \delta_{ij} = \alpha^j \qquad \forall j = 1, .., m$$

Luego $\alpha^j = 0 \ \ \forall j = 1, ..., m.$ ■

El método de gradiente conjugado para el problema (PC), suponiendo que no se tienen calculadas las direcciones conjugadas, es el siguiente:

Etapa 0:	Seleccionar un punto inicial $x^0 \in \mathbb{R}^n$
	$k = 0$
	$g^0 = \nabla q(x^0) = Qx^0 - b \qquad d^0 = -\, g^0$
Etapa 1:	Calcular
	$\alpha^k = -\dfrac{(g^k)^t d^k}{(d^k)^t Q d^k}$
	$x^{k+1} = x^k + \alpha^k d^k$
	Si $\left\|x^{k+1} - x^k\right\| \leq Tol \Longrightarrow STOP$
	Si no, seguir a Etapa 2.
Etapa 2:	Calcular
	$g^{k+1} = \nabla q(x^{k+1}) = Qx^{k+1} - b$
	$\rho_k = \dfrac{(g^{k+1})^t Q d^k}{(d^k)^t\, Q d^k}$
	$d^{k+1} = -g^{k+1} + \rho^k d^k$
	$k = k + 1$ y volver a Etapa 1.

Método del Gradiente Conjugado.

Teorema 9 *El método del gradiente conjugado es un método de direcciones conjugadas. Si no termina en x_k, entonces:*

i) $\langle g^0, ..., g^k \rangle = \langle g^0, Qg^0, ..., Q^k g^0 \rangle$

ii) $\langle d^0, ..., d^k \rangle = \langle g^0, Qg^0, ..., Q^k g^0 \rangle$

iii) $(d^k)^t Q d^i = 0 \qquad \forall\, i = 0, ..., k-1$

iv) $\alpha^k = \dfrac{(g^k)^t g^k}{(d^k)^t Q d^k} = \dfrac{\|g^k\|^2}{\|d^k\|_Q^2}$

v) $\rho_k = \dfrac{(g^{k+1})^t g^{k+1}}{(g^k)^t g^k} = \dfrac{\|g^{k+1}\|^2}{\|g^k\|^2}$

Demostración. *La demostración se puede revisar en las refs. [3, 6].* ■

Como las direcciones conjugadas son Q-ortogonales entre sí, el método del gradiente conjugado converje en a lo más n iteraciones, considerando aritmética de precisión infinita.

Ejemplo 10 *Consideremos el problema cuadrático (PC) definido por la matriz de Pascal:*

$$Q = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{73}$$

Cuya solución es $x = (1, 0, 0, 0)^T$. En la siguiente tabla se muestran las iteraciones del método de gradiente conjugado a partir de $x^0 = (10, 10, 10, 10)^T$:

k	x^k	$\|x - x^k\|_2$
0	$(10, 10, 10, 10)^T$	19.5192
1	$(8.5126, 6.2243, 2.4104, -3.3104)^T$	10.5806
2	$(2.6392, -0.2346, -1.2772, 0.6178)^T$	2.1806
3	$(0.9468, 0.1244, -0.1022, 0.0289)^T$	0.1720
4	$(1, 0, 0, 0)^T$	0

ACT3: Métodos Iterativos para SEL:

Utilizando la instrucción switch/case programe la función metodos_iterativos_sel.m que calcula la solución de un SEL aplicando los métodos iterativos de Jacobi, Gauss-Seidel, Gauss-Seidel Sobre-Relajado y Gradiente Conjugado. De acuerdo a la factorización seleccionada debe recibir diferentes inputs, realizar algunos cálculos y entregar diferentes outputs.

1) Para los métodos de Jacobi y Gauss-Seidel:

   (a) Recibir como input una matriz A de tamaño $n \times n$, el vector lado derecho b de tamaño $n \times 1$, el punto inicial $x0$ de tamaño $n \times 1$, la tolerancia Tol y la cantidad máxima de iteraciones $MaxIter$. Se debe chequear si las dimensiones del SEL estan bien definidas. Si no es así, se debe imprimir un mensaje de error y salir sin ejecutar el método.

Hint: El parámetro Tol permite determinar si el método iterativo debe terminar por convergencia. Se utiliza de la siguiente forma:

$$\text{Si } \left\|x^{k+1} - x^k\right\| \leq Tol \Longrightarrow \ STOP \tag{74}$$

donde x^{k+1}, x^k son 2 iteraciones consecutivas del método iterativo.

(b) Si se llama la función sin paramétro Tol se debe definir un valor por default $Tol = 10^{-6}$.
Si se llama la función sin paramétro $MaxIter$ se debe definir un valor por default $MaxIter = n$.
Si se llama la función sin paramétro $x0$ se debe definir un valor por default $x^0 = \overrightarrow{0}$.
Hint: Puede utilizar las instrucciones $nargin, nargchk$ para verificar la cantidad de argumentos con que es llamada una función.

(c) Implementar los métodos de Jacobi y Gauss-Seidel en su forma matricial, comenzando de x^0 y teniendo como output la solución de $Ax = b$ entregada por el método, el error y el tiempo utilizado en cpu.
Hint: Puede utilizar los comandos: $triu, tril$, para construir las matrices de la iteración.

(d) Pruebe la función con varios sets de datos (A, b) generados aleatoriamente para $n = 100, 200, 300, 400, 500$. Considere matrices bien y mal condicionadas. No incluya los datos en el informe, sólo los resultados. Como varía la precisión y el tiempo de cpu al aumentar n ?

2) Para el método de Gauss-Seidel Sobre-Relajado:

(a) Recibir como input una matriz A de tamaño $n \times n$, el vector lado derecho b de tamaño $n \times 1$, el punto inicial $x0$ de tamaño $n \times 1$, la tolerancia Tol y la cantidad máxima de iteraciones $MaxIter$. Se debe chequear si las dimensiones del SEL estan bien definidas. Si no es así, se debe imprimir un mensaje de error y salir sin ejecutar el método.

(b) Definir valores por default a parámetros no especificados, de acuerdo a 1) b).

(c) Implementar el método de Gauss-Seidel Sobre-Relajado en su forma matricial, comenzando de x^0 y teniendo como output la solución de $Ax = b$ entregada por el método, el error y el tiempo utilizado en cpu. Para aplicar este método se debe calcular ω_{opt} en forma aproximada.

(d) Pruebe este método con el mismos datos que usó para Jacobi y Gauss-Seidel. Es mejor este método ? Explique en detalle.

3) Programe el método de Gradiente Conjugado de acuerdo a lo realizado para los métodos de Jacobi, Gauss-Seidel y Gauss-Seidel Sobre-Relajado.

Pruebe este método con el mismos datos que usó para los métodos anteriores.

De acuerdo a sus resultados: Cual método es mejor en precisión y tasa de convergencia ? Explique en detalle.

References

[1] Akai, T.J., Applied Numerical Methods for Engineers, John Wiley & Sons, 1994.

[2] Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK User's Guide, Third Edition, SIAM, Philadelphia, 1999. (http://www.netlib.org/lapack/lug/lapack_lug.html)

[3] Bazaraa, M., H. D. Sherali and C. M. Shetty, Nonlinear Programming: Theory and Algorithms, Second Edition, John Wiley and Sons, 1993.

[4] Kiusalaas, J., Numerical Methods in Engineering with Matlab, Cambridge University press, 2005.

[5] Golub, G.H., C.F. Van Loan, Matrix Computations, Third Edition, John Hopkins University Press, 1996.

[6] Luenberger, D.G., Linear and Nonlinear Programming, Second Edition, John Wiley and Sons, 1984.

[7] Ortega, J.M., Numerical Analysis: A Second Course, SIAM Classics in Applied Mathematics, 1990.

[8] Quarteroni, A., R. Sacco, F. Saleri, Numerical Mathematics, Second Edition, Text in Applied Mathematics, Springer, 2007.