4101553 Métodos Numéricos aplicados a la Ingenieria Civil

Departamento de Ingeniería Civil

Universidad Nacional de Colombia

Sede Manizales

Docente: Juan Nicolás Ramírez Giraldo (jnramirezg@unal.edu.co (mailto:jnramirezg@unal.edu.co))

"*Cum cogitaveris quot te antecedant, respice quot sequantur*"

Séneca

Repositorio de la asignatura (https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/)

Descomposición LU (dootlitlle)

Es un método de descomposición de matrices cuadradas.

Sea $\underline{\underline{A}}$ una matriz cuadrada de orden n, se descompone como:

$$\underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{U}}$$

En donde:

$\underline{\underline{L}}$ es una matriz triangular inferior.

$\underline{\underline{U}}$ es una matriz triangular superior.

Su uso se extiende a dos objetivos:

- Solución de sistemas de ecuaciones lineales.
- Cálculo de la inversa de una matriz.

Particularmente, se puede realizar la descomposición a partir del método de Gauss. Y su gran utilidad está en la solución de sistemas de ecuaciones donde los coeficientes constantes ($\underline{\underline{A}}$) **no cambian**, pero el vector de constantes ($\underline{A}$ puede tomar distintos valores.

Una matriz $\underline{\underline{A}}$ solo se puede descomponer si no es singular, es decir, $|\underline{\underline{A}}| \neq 0$

Dada una matriz cuadrada de orden 3:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Su descomposición en las matrices $\underline{\underline{L}}$ y $\underline{\underline{U}}$ es:

$$\underline{\underline{L}} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

$$\underline{\underline{U}} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Aquí, los escalares l_{21}, l_{31}, l_{31}, u_{11}, u_{12}, u_{13}, u_{22}, u_{23} y u_{33} representan las 9 incógnitas que se requieren hallar para hacer la descomposición. Esta es la razón por la que la matriz $\underline{\underline{L}}$ tiene unos

Construcción del método

El método para resolver ecuaciones lineales se puede construir a partir de la configuración matricial:

$\underline{\underline{A}}\,\underline{X} = \underline{B}$ (1)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

En donde:

$\underline{\underline{A}}$ es la matriz de coeficientes constantes.

$\underline{X}$ es el vector de incógnitas.

$\underline{B}$ es el vector de constantes.

De esta manera, se puede representar $\underline{\underline{A}}$ como:

$\underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{U}}$ (2)

$$\underline{\underline{A}} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Y al vector $\underline{B}$ como la multiplicación de la matriz $\underline{\underline{L}}$ y un nuevo vector $\underline{D}$:

$\underline{B} = \underline{\underline{L}}\,\underline{D}$ (3)

Ecuacuciones disponibles:

$\underline{\underline{A}}\ \underline{X} = \underline{B}$ (1)

$\underline{\underline{A}}$ = $\underline{\underline{L}}\ \underline{\underline{U}}$ (2)

$\underline{B}$ = $\underline{\underline{L}}\ \underline{D}$ (3)

Luego, al igualar la ecuación (1) a $\underline{0}$, se obtiene:

$\underline{\underline{A}}\ \underline{X} - \underline{B} = \underline{0}$

Se reemplaza $\underline{\underline{A}}$ por $\underline{\underline{L}}\ \underline{\underline{U}}$ de acuerdo a (2):

$\underline{\underline{L}}\ \underline{\underline{U}}\ \underline{X} - \underline{B} = \underline{0}$

Y se reemplaza $\underline{B}$ por $\underline{\underline{L}}\ \underline{D}$ de acuerdo con (3):

$\underline{\underline{L}}\ \underline{\underline{U}}\ \underline{X} - \underline{\underline{L}}\ \underline{D} = \underline{0}$

Se "factoriza" $\underline{\underline{L}}$.

$\underline{\underline{L}}\ (\underline{\underline{U}}\ \underline{X} - \underline{D}) = \underline{0}$

Y aquí, la única posibilidad es que $\underline{\underline{U}}\ \underline{X} - \underline{D} = \underline{0}$, es decir:

$\underline{\underline{U}}\ \underline{X} = \underline{D}$

Por lo que se obtienen las dos ecuaciones fundamentales del método:

$\underline{\underline{L}}\ \underline{D}$ = $\underline{B}$
$\underline{\underline{U}}\ \underline{X} = \underline{D}$

Ya que $\underline{\underline{L}}$ es una matriz triangular inferior, $\underline{D}$ se halla con **sustución hacia adelante**, y al tener $\underline{D}$, de la segunda ecuación se despeja $\underline{X}$ con **sustitución hacia atrás**.

Algunas Propiedades

1. $|\underline{\underline{A}}|$ = $|\underline{\underline{L}}|\ |\underline{\underline{U}}|$
2. $\underline{\underline{A}}^{-1}$ = $\underline{\underline{U}}^{-1}\ \underline{\underline{L}}^{-1}$

Desarrollo de función en Python con Numpy de descomposición LU

Se importa el módulo `Numpy`

In [1]:
```
import numpy as np
```

Descomposición de la Matriz A en las matrices L y U con el método de la eliminación de Gauus. La matriz A se debe ingresar como una lista de listas. El programa por dentro requiere el módulo numpy. Devuelve L y U como np.array()

In [2]:
```
def np_descomposicion_LU(A):
    m = len(A)
    L = np.eye(m, dtype=float)     # Se crea L como una matriz identidad.
    U = np.array(A, dtype=float)   # Se crea U como una copia de A.

    for k in range(m):
        U1 = np.array([f[:] for f in U])  # Copia de la matriz.
        for i in range(k+1, m):
            for j in range(m):
                pivote = U[k, k]
                U1[i, j] += -U[i, k]/pivote * U[k, j]
                if j==k:
                    L[i, j] = U[i, k]/pivote
        U = U1

    return L, U
```

Ver: [10-descomposicion_LU.py (https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/blob/main/codigo/10-descomposicion_LU.py)](https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/blob/main/codigo/10-descomposicion_LU.py)

Ejemplo 1:

In [3]:
```
A = [[ 2,  1, -1,  2],
     [ 4,  5, -3,  6],
     [-2,  5, -2,  6],
     [ 4, 11, -4,  8]]

L, U = np_descomposicion_LU(A)
```

In [4]:
```
L
```

Out[4]:
```
array([[ 1.,  0.,  0.,  0.],
       [ 2.,  1.,  0.,  0.],
       [-1.,  2.,  1.,  0.],
       [ 2.,  3., -1.,  1.]])
```

In [5]:
```
U
```

Out[5]:

```
array([[ 2.,  1., -1.,  2.],
       [ 0.,  3., -1.,  2.],
```

In [6]:
```
L@U - A
```

Out[6]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

Ejemplo 2:

In [7]:
```
A = [[ 1, 1],
     [-1, 1]]

L, U = np_descomposicion_LU(A)
```

In [8]:
```
L
```

Out[8]:
```
array([[ 1.,  0.],
       [-1.,  1.]])
```

In [9]:
```
U
```

Out[9]:
```
array([[1., 1.],
       [0., 2.]])
```

In [10]:
```
L@U - A
```

Out[10]:
```
array([[0., 0.],
       [0., 0.]])
```

Solución de sistemas de ecuaciones lineales con descomposición LU

Para realizar la solución de un sistema de ecuaciones lineales mediante una descomposición LU se requiere considerar las mismas fuentes de error que en la eliminación de Gauss y la eliminación de Gauss-Jordan.

```
for j in range(m):
    pivote = U[k, k]
    U1[i, j] += -U[i, k]/pivote * U[k, j]
    if j==k:
        L[i, j] = U[i, k]/pivote
```

Por lo tanto, para el desarrollo de una función que use la descomposición LU como mecanismo de solución de sistemas de ecuaciones lineales, se requiere considerar los casos de **sistema singular** y de **pivoteo parcial** ya vistos.

En algunas referencias bibliográficas se menciona el método PLU que, en últimas es una formalidad de la **solución con pivoteo parcial**.

Miremos el siguiente ejemplo.

Dada una matriz aumentada $\underline{\underline{S}}$ de tamaño 3.

In [11]:
```
S = np.array([[1,  2,  3,  4],
              [5,  6,  7,  8],
              [9, 10, 11, 12]])
```

Si se requiere realizar **pivoteo parcial** respecto al primer elemento de la diagonal, se identifica que la operación más conveniente es el intercambio de la fila 3 por la fila 1. Esta operación, en términos formales se puede representar como: $\underline{\underline{P}}\,\underline{\underline{S}}$, donde P es:

In [12]:
```
P = np.array([[0, 0, 1],
              [0, 1, 0],
              [1, 0, 0]])
```

Con esto:

In [13]:
```
P@S
```

Out[13]:
```
array([[ 9, 10, 11, 12],
       [ 5,  6,  7,  8],
       [ 1,  2,  3,  4]])
```

Nótese que si se premultiplica la matriz $\underline{\underline{S}}$ con $\underline{\underline{P}}$, el resultado será un intercambio de filas.

¿Qué ocurre si $\underline{\underline{S}}$ se posmultiplica?

Se genera un error, porque no se puede multiplicar una matriz de 3x4 con una de 3x3, sí al revés.

Luego, para solucionar sistemas de ecuaciones lineales a partir de la descomposición LU, se obtiene a partir del ejemplo 1:

In [14]:
```
# Ejemplo 1.
# Matriz de coeficientes constantes.
A = [[ 1, -2,  1,  3],
     [ 3,  1, -4, -2],
     [ 2,  2, -1, -1],
     [ 1,  4,  2, -5]]
# Vector de constantes.
B = [-3, 7, 1, 12]
```

In [15]:
```
L, U = np_descomposicion_LU(A)   # Descomposición LU de A.
B = np.array([B]).T              # Se convierte B en un np.array de dim ad
m = len(A)                       # Tamaño del sistema.
```

In [16]:
```
# Sustitución hacia adelante.
S1 = np.append(L, B, axis=1)     # Matriz aumentada para el sistema LD=B
for i in range(m):
    for j in range(i+1, m):
        S1[j, :] = S1[j, :] - S1[i, :]*S1[j, i]
D = np.array([S1[:, -1]]).T
```

In [17]:
```
# Sustitución hacia atrás.
S2 =  np.append(U, D, axis=1)    # Matriz aumentada para el sistema UX=D
for i in range(m):
    for j in range(i+1, m):
        p = m-i-1                # Conteo hacia atrás en i.
        q = m-j-1                # Conteo hacia adelante en j.
        S2[q] = S2[q] - S2[p, :]*S2[q, p]/S2[p, p]
    S2[p, :] =  S2[p, :]/S2[p, p]
X = S2[:, -1]
```

In [18]:
```
# Impresión de resultados.
X
```

Out[18]:
```
array([ 2., -3.,  1., -4.])
```

Ver: [11-sol_descomposicion_LU.py (https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/blob/main/codigo/11-sol_descomposicion_LU.py)](https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/blob/main/codigo/11-sol_descomposicion_LU.py)

Inversión de matrices a partir de la descomposición LU

Usando la técnica de Gauss-Jordan se pueden invertir matrices a partir del siguiente algoritmo:

Dada una matriz $\underline{\underline{A}}$ de orden $m = 3$:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Su inversa se puede calcular a partir de la siguiente matriz aumentada:

$$(\underline{\underline{A}}|\underline{\underline{I_3}})$$

$$\left[\begin{array}{ccc|ccc} a_{11} & a_{12} & a_{13} & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 & 1 \end{array}\right]$$

Aplicando operaciones entre filas se debe llegar a:

$$(I_3 \,|\, A^{-1})$$

Por ejemplo:

A partir de la matriz $\underline{\underline{A}}$:

$$\underline{\underline{A}} = \begin{bmatrix} 1 & -2 & 1 & 3 \\ 3 & 1 & -4 & -2 \\ 2 & 2 & -1 & -1 \\ 1 & 4 & 2 & -5 \end{bmatrix}$$

Se obtuvieron las matrices de su descomposición $\underline{\underline{L}}$ y $\underline{\underline{U}}$:

$$\underline{\underline{L}} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 1.0 & 0.0 & 0.0 \\ 2.0 & 0.857142857142857 & 1.0 & 0.0 \\ 1.0 & 0.857142857142857 & 2.33333333333333 & 1.0 \end{bmatrix}$$

$$\underline{\underline{U}} = \begin{bmatrix} 1.0 & -2.0 & 1.0 & 3.0 \\ 0.0 & 7.0 & -7.0 & -11.0 \\ 0.0 & 0.0 & 3.0 & 2.42857142857143 \\ 0.0 & 0.0 & 0.0 & -4.23809523809524 \end{bmatrix}$$

De manera muy similar a la solución del sistema, se halla la inversa aprovechando esta propiedad:

$$\underline{\underline{A}}^{-1} = \underline{\underline{U}}^{-1} \, \underline{\underline{L}}^{-1}$$

Hallar $\underline{\underline{L}}^{-1}$ y $\underline{\underline{U}}^{-1}$ resulta sencillo porque se puede aplicar la técnica de Gauss-Jordan en cada una de ellas, pero con la particularidad de que $\underline{\underline{L}}^{-1}$ solo requiere **eliminación hacia adelante** y $\underline{\underline{U}}^{-1}$ **eliminación hacia atrás**, procesos que ya fueron programados previamente para la obtención de soluciones del sistema.

In [19]:
```
# Se crea nuevamente la matriz como una lista de listas.
A = [[ 1, -2,  1,  3],
     [ 3,  1, -4, -2],
     [ 2,  2, -1, -1],
     [ 1,  4,  2, -5]]
```

In [20]:
```
A = np.array(A, dtype=float)     # Se convierte A a un np.array

L, U = np_descomposicion_LU(A)   # Descomposición LU de A, antes creada.
m = len(L)                       # Tamaño del sistema.
```

In [21]:
```
# Eliminación hacia adelante.
S1 = np.append(L, np.eye(m), axis=1)     # Matriz aumentada de L.
for i in range(m):
    for j in range(i+1, m):
        S1[j, :] = S1[j, :] - S1[i, :]*S1[j, i]
Linv = S1[:, m:]
```

In [22]:
```
# Eliminación hacia atrás.
S2 = np.append(U, np.eye(m), axis=1)     # Matriz aumentada de U.
for i in range(m):
    for j in range(i+1, m):
        p = m-i-1                # Conteo hacia atrás en i.
        q = m-j-1                # Conteo hacia adelante en j.
        S2[q] = S2[q] - S2[p, :]*S2[q, p]/S2[p, p]
    S2[p, :] =  S2[p, :]/S2[p, p]
Uinv = S2[:, m:]
```

In [23]:
```
Ainv = Uinv@Linv  # A^(-1) = U^(-1)*U^(-1)
```

In [24]:
```
Linv
```

Out[24]:
```
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [-3.        ,  1.        ,  0.        ,  0.        ],
       [ 0.57142857, -0.85714286,  1.        ,  0.        ],
       [ 0.23809524,  1.14285714, -2.33333333,  1.        ]])
```

In [25]:
```
Uinv
```

Out[25]:
```
array([[ 1.        ,  0.28571429,  0.33333333,  0.15730337],
       [ 0.        ,  0.14285714,  0.33333333, -0.17977528],
       [ 0.        ,  0.        ,  0.33333333,  0.19101124],
       [-0.        , -0.        , -0.        , -0.23595506]])
```

In [26]:
```
Ainv
```

Out[26]:
```
array([[ 0.37078652,  0.17977528, -0.03370787,  0.15730337],
       [-0.28089888, -0.34831461,  0.75280899, -0.17977528],
       [ 0.23595506, -0.06741573, -0.11235955,  0.19101124],
       [-0.05617978, -0.26966292,  0.5505618 , -0.23595506]])
```

In [27]:
```
A @ Ainv
```

Out[27]:
```
array([[ 1.00000000e+00, -5.55111512e-17,  1.11022302e-16,
         2.77555756e-17],
       [ 4.85722573e-16,  1.00000000e+00,  1.33226763e-15,
        -3.88578059e-16],
       [ 7.63278329e-17, -4.44089210e-16,  1.00000000e+00,
        -2.22044605e-16],
       [-1.17961196e-16, -4.99600361e-16,  3.33066907e-16,
         1.00000000e+00]])
```

In [28]:
```
# Pilas que este comando "_" sirve para llamar el resultado anterior
_-np.eye(4)
```

Out[28]:
```
array([[ 0.00000000e+00, -5.55111512e-17,  1.11022302e-16,
         2.77555756e-17],
       [ 4.85722573e-16, -7.77156117e-16,  1.33226763e-15,
        -3.88578059e-16],
       [ 7.63278329e-17, -4.44089210e-16,  8.88178420e-16,
        -2.22044605e-16],
       [-1.17961196e-16, -4.99600361e-16,  3.33066907e-16,
        -3.33066907e-16]])
```

In [29]:
```
np.isclose(_, 0)
```

Out[29]:
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

Ver: [12-inv_descomposicion_LU.py (https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/blob/main/codigo/12-inv_descomposicion_LU.py)]

Descomposición de Cholesky

Es un método de descomposición de matrices cuadradas y simétricas. Su principal ventaja es que crea dos matrices de descomposición, pero la una es la transpuesta de la otra, por lo que solo se requiere hacer la mitad de las operaciones y solo se requiere almacenar la mitad de la información.

$$\underline{\underline{A}} = \underline{\underline{L}}\ \underline{\underline{L}}^T$$

La matriz $\underline{\underline{A}}$ debe ser Hermítica y definida positiva, debido a las restricciones del algoritmo.

Se construye la matriz $\underline{\underline{L}}$ a partir del siguiente algoritmo:

Para la fila k-ésima:

$$l_{ki} = \frac{a_{ki} - \sum_{j=1}^{i-1} l_{ij} l_{kj}}{l_{ii}} \text{ para } i=1,2,...,k-1$$

y

$$\sqrt{\quad}$$

Aplicando el algoritmo para el siguiente ejemplo:

In [30]:
```
A = [[ 6,   15,  55],
     [ 15,  55, 225],
     [ 55, 225, 979]]

A = np.array(A, dtype=float)     # Se convierte A a un np.array
m = len(A)
L = np.zeros((m ,m))
```

In [31]:
```
for k in range(m):
    for i in range(k):
        suma_tem = 0
        suma_tem += A[k, i]
        for j in range(i):
            suma_tem += -(L[i, j]*L[k, j])
        L[k][i] = (suma_tem/L[i, i])
    sum_tem = 0
    sum_tem += A[k, k]
    for j in range(k):
        sum_tem += -(L[k, j])**2
    L[k, k] = (sum_tem**0.5)
```

In [32]:
```
L
```

Out[32]:
```
array([[ 2.44948974,  0.        ,  0.        ],
       [ 6.12372436,  4.18330013,  0.        ],
       [22.45365598, 20.91650066,  6.11010093]])
```

Se verifica la igualdad $\underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{L}}^T$

In [33]:
```
L@L.T
```

Out[33]:
```
array([[  6.,  15.,  55.],
       [ 15.,  55., 225.],
       [ 55., 225., 979.]])
```

In [34]:
```
L@L.T-A
```

Out[34]:

```
array([[-8.88178420e-16,  0.00000000e+00,  0.00000000e+00],
```

In [35]:
```
np.isclose(L@L.T-A, 0)
```

Out[35]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

De esta manera, aplicando los mismos conceptos de la descomposición LU se puede halla la inversa de la matriz o las soluciones de un sistema que la involucren.

Ver: [13-cholesky.py (https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/blob/main/codigo/13-cholesky.py)](https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/blob/main/codigo/13-cholesky.py)