Universidad Autónoma de Querétaro

FACULTAD DE INGENIERÍA

Tarea 6: Descompocisión LU

Análisis numérico

Autor:
J.A. Salinas Sánchez
Marzo2022

Índice general

Capítulo 1

Introducción

A pesar de ya conocer múltiples métodos, todos muy eficaces, universales y bastante precisos en cuanto a la resolución de sistemas de ecuaciones se refiere. Algunos de estos métodos pecan de poco eficientes según la cantidad de datos a tratar, del sistema en sí o solamente de su mero funcionamiento. Es por eso que hoy se explora la opción de utilizar el método de factorización LU o descomposición **Lower-Upper** de sus siglas en inglés.

Como lo indica su nombre en inglés, la descomposición/factorización LU-consiste en buscar dos matrices triangulares, una superior (*Upper*) y una inferior (*Lower*), las cuales, al multiplicarse, regresen a la matriz original como resultado. El propósito de hacer esto es descomponer un sistema de ecuaciones nxn original en dos sistemas de ecuaciones más simples que pueden ser resuletos con cualquier otro método. Pudiendo reducir la magnitud del algoritmo desde $O(n^3)$ hasta $O(n^2)$.

Al igual que la factorización de expresiones algebraicas escalares o de cualquier otro objeto matemático, ésta se basa en el principio de hacer una ecuación entre cada uno de sus términos y un factor, de modo que cuando se multiplique a los términos por dicho factor, se obtenga de regreso la expresión original. Para ello, siguiendo las reglas de la multiplicación entre matrices, las matrices L y U tienen que ser cuadradas del mismo tamaño de la matriz original A, tienen que ser diagonales inferior y superior, respectivamente, y una tiene que tener una diagonal de sólo unos. Entonces, se buscaría lo siguiente:

$$A = L \cdot U = \begin{bmatrix} L_{11} & \dots & L_{1n} \\ \vdots & \dots & \vdots \\ L_{n1} & \dots & L_{nn} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & \dots & U_{1n} \\ \vdots & \dots & \vdots \\ U_{n1} & \dots & U_{nn} \end{bmatrix} = \tag{1.1}$$

$$\begin{bmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \dots & \vdots \\ A_{n1} & \dots & A_{nn} \end{bmatrix} \therefore \sum_{i,j=1}^{n} L_{ij} U_{ji} = A_{ij} \tag{1.2}$$

Lo curioso es que los coeficientes inferiores de la matriz L son los factores utilizados en la eliminación gaussiana, por lo que se puede aplicar eliminación gaussiana sobre A y guardar los coeficientes en L, resolver L y anexar los términos independientes en U, para luego resolver U. Aquí es donde se puede simplificar aún más el algoritmo.

Capítulo 2

Metodología

Dependiendo del problema, se usó un script el cual incluía sólo la resolución mediante descomposición LU con o sin pivoteo, o que incluía lo anterior más la comprobación de la descomposición. Se muestra en el siguiente pseudocódigo.

```


main(){
  uk=numero de inc gnitas
  //definir la matriz expandida del sistema de ecuaciones
  //definir una matriz temporal para guardar las soluciones
  LU(matriz,temporal,uk)

}

LU(matriz,temporal,uk){

  u[uk][uk+1]=zeros//iniciar la matriz u en ceros
  l[uk][uk+1]=I+v[0,0,0,0]//iniciar la matriz l como I m s una
   columna
  listaparaarmarl=[]
  //a adir los t rminos independientes de matriz a l
  //eliminaci n gausiana sobre matriz
    //ir a adiendo los ratios de la eliminaci n a la lista
  //a adir solo los coeficientes de matriz a u
  //vaciar listaparaarmarl en l debajo de su diagonal

  //resolver el sistema de ecuaciones de l
  //a adir los resultados como t rminos independientes en u
  //resolver para u
  //devolver resultados
  (si el problema lo pide, multiplicar l*u y mostrarla)

}
```

2.1. Problemas

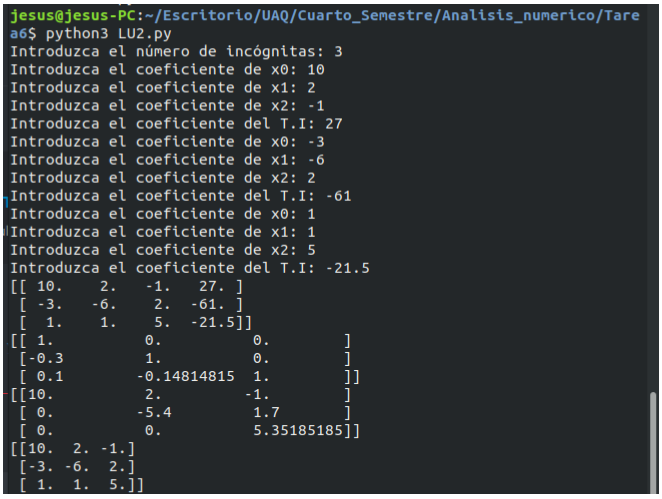
2.1.1. 9.2

```
#coding:utf8
import numpy as np

def main():

```

```
    uk=int(input('Introduzca el n mero de inc gnitas: '))
    matriz1=np.zeros((uk,uk+1))
    tempmatrix=np.zeros(uk)

    for i in range(0,uk):
        for j in range(0,uk+1):
            if(j==uk):
                matriz1[i][j]=float(input(f'Introduzca el coeficiente del T.I: '))
            else:
                matriz1[i][j]=float(input(f'Introduzca el coeficiente de x{j}: '))
    LU(matriz1,tempmatrix,uk)

def LU(m1,tm,uk):
    i=0
    j=0
    k=0

    u=np.zeros((uk,uk))
    l=np.zeros((uk,uk))
    lelements=[]
    print('A=')
    print(m1)
    for i in range(uk):
        l[i][i]=1

    for i in range(uk):
        for j in range(i+1, uk):
            ratio = m1[j][i]/m1[i][i]
            lelements.append(ratio)
            for k in range(uk+1):
                m1[j][k] = m1[j][k]-ratio*m1[i][k]

    for i in range(uk):
        for j in range(uk):
            u[i][j]=m1[i][j]

    k=0
    for j in range(uk):
        for i in range(j+1,uk):
            l[i][j]=lelements[k]
            k+=1


    print('L=')
    print(l,"")
    print('U=')
    print(u,"")
    print('L*U=')
    print(np.dot(l,u),"")
    return(0)


if __name__=='__main__':
    main()
```

2.1.2. 9.3

```
#coding:utf8
import numpy as np

def main():
```



```
jesus@jesus-PC:~/Escritorio/UAQ/Cuarto_Semestre/Analisis_numerico/Tare
a6$ python3 LU2.py
Introduzca el número de incógnitas: 3
Introduzca el coeficiente de x0: 10
Introduzca el coeficiente de x1: 2
Introduzca el coeficiente de x2: -1
Introduzca el coeficiente del T.I: 27
Introduzca el coeficiente de x0: -3
Introduzca el coeficiente de x1: -6
Introduzca el coeficiente de x2: 2
Introduzca el coeficiente del T.I: -61
Introduzca el coeficiente de x0: 1
Introduzca el coeficiente de x1: 1
Introduzca el coeficiente de x2: 5
Introduzca el coeficiente del T.I: -21.5
[[ 10.    2.   -1.   27. ]
 [ -3.   -6.    2.  -61. ]
 [  1.    1.    5.  -21.5]]
[[ 1.          0.          0.        ]
 [-0.3        1.          0.        ]
 [ 0.1       -0.14814815  1.        ]]
[[10.          2.         -1.        ]
 [ 0.         -5.4         1.7       ]
 [ 0.          0.          5.35185185]]
[[10.  2. -1.]
 [-3. -6.  2.]
 [ 1.  1.  5.]]
```

Figura 2.1: Programa que muestra las matrices U y L en las cuales se descompuso la matriz de coeficientes del sistema de ecuaciones original, y cómo su multiplicación devuelve la matriz original.

```

    uk=int(input('Introduzca el n mero de inc gnitas: '))
    matriz1=np.zeros((uk,uk+1))
    tempmatrix=np.zeros(uk)

    for i in range(0,uk):
        for j in range(0,uk+1):
            if(j==uk):
                matriz1[i][j]=float(input(f'Introduzca el
   coeficiente del T.I: '))
            else:
                matriz1[i][j]=float(input(f'Introduzca el
   coeficiente de x{j}: '))
    LU(matriz1,tempmatrix,uk)

def LU(m1,tm,uk):
    i=0
    j=0
    k=0

    u=np.zeros((uk,uk+1))
    l=np.zeros((uk,uk+1))
    lelements=[]

    for i in range(uk):
        l[i][i]=1

    for i in range(uk):
        l[i][uk]=m1[i][uk]


    for i in range(uk):
        for j in range(i+1, uk):
            ratio = m1[j][i]/m1[i][i]
            lelements.append(ratio)
            for k in range(uk+1):
                m1[j][k] = m1[j][k]-ratio*m1[i][k]

    for i in range(uk):
        for j in range(uk):
            u[i][j]=m1[i][j]
```

```

    k=0
    for j in range(uk):
        for i in range(j+1,uk):
            l[i][j]=lelements[k]
            k+=1

    lelements=gauss(l,tm,uk)

    for i in range(uk):
        u[i][uk]=lelements[i]

    print(gauss(u,tm,uk))
    return(0)

def gauss(m1,tm,uk):
    i=0
    j=0
    k=0

    for i in range(uk):
        for j in range(i+1, uk):
            ratio = m1[j][i]/m1[i][i]
            for k in range(uk+1):
                m1[j][k] = m1[j][k]-ratio*m1[i][k]

    tm[uk-1] = m1[uk-1][uk]/m1[uk-1][uk-1]

    for i in range(uk-2,-1,-1):
        tm[i] = m1[i][uk]
        for j in range(i+1,uk):
            tm[i] = tm[i] - m1[i][j]*tm[j]
        tm[i] = tm[i]/m1[i][i]

    return(tm)

if __name__=='__main__':
    main()
```

Figura 2.2: Sistema de ecuaciones resuelto mediante descomposición LU.

2.1.3. 9.4

```
#coding:utf8
import numpy as np

def main():
```

```
    uk=int(input('Introduzca el n mero de inc gnitas: '))
    matriz1=np.zeros((uk,uk+1))
    tempmatrix=np.zeros(uk)

    for i in range(0,uk):
        for j in range(0,uk+1):
            if(j==uk):
                matriz1[i][j]=float(input(f'Introduzca el coeficiente del T.I: '))
            else:
                matriz1[i][j]=float(input(f'Introduzca el coeficiente de x{j}: '))
    LU(matriz1,tempmatrix,uk)

def LU(m1,tm,uk):
    i=0
    j=0
    k=0

    u=np.zeros((uk,uk+1))
    l=np.zeros((uk,uk+1))
    lelements=[]

    for i in range(uk):
        l[i][i]=1

    for i in range(uk):
        l[i][uk]=m1[i][uk]


    for i in range(uk):
        for j in range(i+1, uk):
            ratio = m1[j][i]/m1[i][i]
            lelements.append(ratio)
            for k in range(uk+1):
                m1[j][k] = m1[j][k]-ratio*m1[i][k]

    for i in range(uk):
        for j in range(uk):
            u[i][j]=m1[i][j]

    k=0
    for j in range(uk):
        for i in range(j+1,uk):
            l[i][j]=lelements[k]
            k+=1

    lelements=gauss(l,tm,uk)

    for i in range(uk):
        u[i][uk]=lelements[i]

    print(gauss(u,tm,uk))
    return(0)

def gauss(m1,tm,uk):
    i=0
    j=0
    k=0

    for i in range(uk):
        for j in range(i+1, uk):
```

```
            ratio = m1[j][i]/m1[i][i]
            for k in range(uk+1):
                m1[j][k] = m1[j][k]-ratio*m1[i][k]

    tm[uk-1] = m1[uk-1][uk]/m1[uk-1][uk-1]

    for i in range(uk-2,-1,-1):
        tm[i] = m1[i][uk]
        for j in range(i+1,uk):
            tm[i] = tm[i] - m1[i][j]*tm[j]
        tm[i] = tm[i]/m1[i][i]

    return(tm)

if __name__=='__main__':
    main()
```

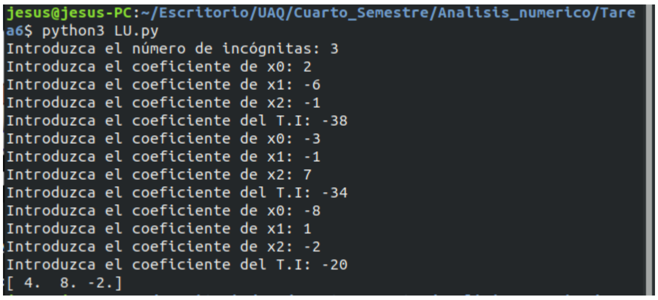


Figura 2.3: Sistema de ecuaciones resulteo con factorización LU.

2.1.4. 9.7

```
#coding:utf8
import numpy as np

def main():

    uk=int(input('Introduzca el n mero de inc gnitas: '))
    matriz1=np.zeros((uk,uk+1))
    tempmatrix=np.zeros(uk)

    for i in range(0,uk):
        for j in range(0,uk+1):
            if(j==uk):
                matriz1[i][j]=float(input(f'Introduzca el coeficiente del T.I: '))
            else:
                matriz1[i][j]=float(input(f'Introduzca el coeficiente de x{j}: '))
    LU(matriz1,tempmatrix,uk)

def LU(m1,tm,uk):
    i=0
    j=0
    k=0

    u=np.zeros((uk,uk))
    l=np.zeros((uk,uk))
    lelements=[]
    print('A=')
    print(m1)
```

```
    for i in range(uk):
        l[i][i]=1

    for i in range(uk):
        for j in range(i+1, uk):
            ratio = m1[j][i]/m1[i][i]
            lelements.append(ratio)
            for k in range(uk+1):
                m1[j][k] = m1[j][k]-ratio*m1[i][k]

    for i in range(uk):
        for j in range(uk):
            u[i][j]=m1[i][j]

    k=0
    for j in range(uk):
        for i in range(j+1,uk):
            l[i][j]=lelements[k]
            k+=1


    print('L=')
    print(l,"")
    print('U=')
    print(u,"")
    print('L*U=')
    print(np.dot(l,u),"")
    return(0)


if __name__=='__main__':
    main()
```

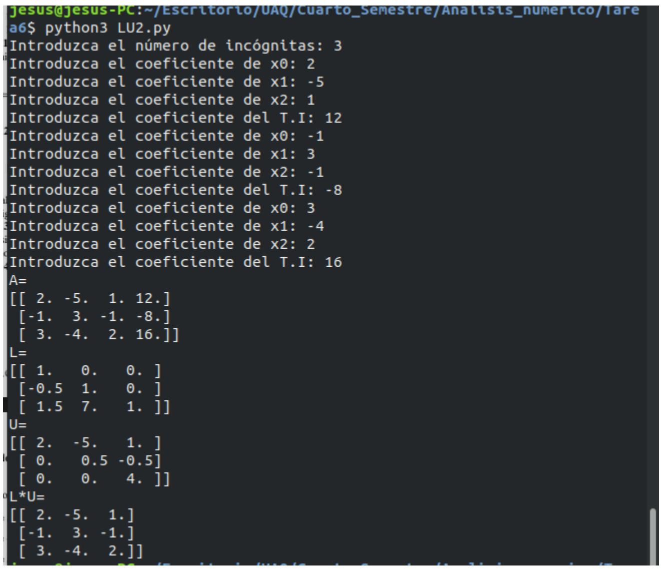


Figura 2.4: Comprobación del éxito de una descomposición LU en el sistema mostrado.

2.1.5. 9.19

Capítulo 3

Conclusión

A pesar de la ya existencia y conocimiento de distintos métodos de resolución de sistemas de ecuaciones lineales, es necesario, por la mejora considerable en eficiencia y por la flexibilidad para resolver más sistemas de ecuaciones. Además de aprovechar las propiedades algebraicas de los tensores de n dimensión, lo que después podría ayudar a desarrollar algoritmos para otras aplicaciones.

Bibliografía

(1) Chapra, S.C., 2015. In R. P. Canale, ed. Métodos numéricos para ingenieros. McGrawHIll Education, pp. 219–236.