Solución de Sistemas de ecuaciones lineales y factorización LU con Python

Johan Posada y Juan Morales

Mayo 14 2024

1 Introducción

... El objetivo de este documento es explicar cómo se puede construir un programa en Python que permita resolver sistemas de ecuaciones lineales de única solución.

2 Construyendo el programa

En un principio el programa se había construido sin usar POO, sin embargo se decidió hacer uso de este paradigma de la programación para mejorar su estructura. El código se puede acceder en el perfil de GitHub del desarrollador del código: https://github.com/johanP051/Equations-systems.git

Para empezar, el sistema de ecuaciones lineales debe seguir esta forma:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
$$

$$
\left(\begin{array}{ccc|c}
a_{11} & a_{12} & \cdots & b_1 \\
a_{21} & a_{22} & \cdots & b_2 \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & b_n
\end{array}\right)
$$

Donde la última columna es el vector de igualdad. El programa se diseñó para que el usuario pueda ingresar los valores de los coeficientes y luego el de los términos independientes, luego se encarga de realizar las operaciones necesarias para encontrar los valores de las variables o incógnitas.

Para empezar veamos el siguiente código:

```python
import numpy as np

class SistemaEcuaciones:
    def __init__(self, ecuaciones, variables):
        self.ecuaciones = ecuaciones
        self.variables = variables

        # Lista para almacenar las filas de la matriz
        self.filas = []
        # Matriz para realizar las operaciones de Gauss-Jordan
        self.matrizReducir = None
```

Listing 1: Atributos de la clase

Fíjese que se ha creado una clase llamada SistemaEcuaciones, despúes en el método def __init__ se crean los atributos del objeto: ecuaciones y variables que son los valores que se solicitarán al usuario. Además se crea una lista llamada filas que almacenará los renglones de la matriz y un arreglo llamado matrizReducir que se usará para realizar las operaciones de Gauss-Jordan (Por el momento solo se define la variable como self.matrizReducir, por eso toma el valor None, eso después va a cambiar cuando el usuario digite los datos).

```python
def recoger_datos(self):
    for i in range(self.ecuaciones):
        elementosFila = []
        print(f"\nPara la ecuacion numero {i + 1}:")

        for j in range(self.variables):
            Xn = int(input(f"Inserte el valor del coeficiente X{j + 1}: "))
            elementosFila.extend([Xn])

        igualdad = int(input(f"A que valor esta igualada la ecuacion?: "))
        elementosFila.extend([igualdad])

        # Agregar la fila a la lista de filas, cada fila representa un solo elemento de la lista
        self.filas.append(elementosFila)
```

Listing 2: Método para recoger los datos

Aquí se crea un método llamado recoger_datos que solicita al usuario los valores de los coeficientes de las variables y los términos independientes. Nótese que hay un bucle for anidado, el bucle externo se usa únicamente para decirle al usuario cuál ecuación o fila de la matriz está digitando (i), mientras que el bucle interno se usa para recorrer las columnas (j) solicitando los valores de los coeficientes, una vez termine este bucle se le pide el valor al que está igualada la ecuación y el bucle externo se vuelve a repetir hasta que finalice el rango del número de ecuaciones. Cada elemento de la fila se almacena en una lista llamada elementosFila y luego cada una se agrega a la lista filas que es una lista de listas.

Para entender bien el siguiente método y el siguiente ejemplo sobre cómo funciona el algoritmo para reducir un sistema de 3x3, una vez se entienda, esto se puede extrapolar a una nxn.

Tomemos la siguiente matriz:

$$\left(\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right)$$

El objetivo de la simplificacion es que los pivotes sean igual a 1. Si simplificamos desde la fila con índice 0 hasta la fila con índice 3, vamos a dividir cada fila teniendo en cuenta el elemento de la columna correspondiente, obtendremos lo siguiente:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{pmatrix} \begin{array}{l} F_1/a_{11} \to F_1 \\ F_2/a_{21} \to F_2 \\ F_3/a_{31} \to F_3 \end{array}$$

Una vez hecho esto, se tiene una matriz simplificada que se puede reducir más fácil::

$$\left(\begin{array}{ccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} \\ 1 & a_{22}/a_{21} & a_{23}/a_{21} & b_2/a_{21} \\ 1 & a_{32}/a_{31} & a_{33}/a_{31} & b_3/a_{31} \end{array}\right) \begin{array}{l} F_2 - F_1 \to F_2 \\ F_3 - F_1 \to F_3 \end{array}$$

Reduciendo teniendo como pivote la fila 1 (indice 0):

$$\left(\begin{array}{ccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} \\ 0 & (a_{22}/a_{21}) - (a_{12}/a_{11}) & (a_{23}/a_{21}) - (a_{13}/a_{11}) & (b_2/a_{21}) - (b_1/a_{11}) \\ 0 & (a_{32}/a_{31}) - (a_{12}/a_{11}) & (a_{33}/a_{31}) - (a_{13}/a_{11}) & (b_3/a_{31}) - (b_1/a_{11}) \end{array}\right)$$

Ahora se debe volver a simplificar la matriz para que el pivote de la fila 2 (indice 1) sea igual a 1 y luego reducir las filas restantes.

$$\left(\begin{array}{ccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} \\ 0 & (a_{22}/a_{21}) - (a_{12}/a_{11}) & (a_{23}/a_{21}) - (a_{13}/a_{11}) & (b_2/a_{21}) - (b_1/a_{11}) \\ 0 & (a_{32}/a_{31}) - (a_{12}/a_{11}) & (a_{33}/a_{31}) - (a_{13}/a_{11}) & (b_3/a_{31}) - (b_1/a_{11}) \end{array}\right) \begin{array}{l} F_2/(a_{22}/a_{21}) - (a_{12}/a_{11}) \to F_2 \\ F_3/(a_{32}/a_{31}) - (a_{12}/a_{11}) \to F_3 \end{array}$$

La matriz simplifcada se puede reducir de nuevo teniendo como pivote la fila 2 (indice 1):

$$\left(\begin{array}{ccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} \\ 0 & 1 & [(a_{23}/a_{21}) - (a_{13}/a_{11})]/[(a_{22}/a_{21}) - (a_{12}/a_{11})] & \cdots \\ 0 & 1 & [(a_{33}/a_{31}) - (a_{13}/a_{11})]/[(a_{32}/a_{31}) - (a_{12}/a_{11})] & \cdots \end{array}\right) F_3 - F_2 \to F_3$$

$$\left(\begin{array}{ccc|c} 1 & a_{12}/a_{11} & a_{13}/a_{11} & b_1/a_{11} \\ 0 & 1 & [(a_{23}/a_{21}) - (a_{13}/a_{11})]/[(a_{22}/a_{21}) - (a_{12}/a_{11})] & \cdots \\ 0 & 0 & \cdots & \cdots \end{array}\right) F_3 - F_2 \rightarrow F_3$$

Después se simplifica la fila 3 para lograr una matriz triangular superior con unos en la diagonal principal y ceros debajo de la diagonal principal. De igual manera el algoritmo revisa si hay más filas restantes para reducir, en este caso no hay más filas restantes, por lo que la primera parte del proceso termina.

Si notamos bien, el proceso se hace por cada fila y termina termina cuando las columnas de la matriz de coeficientes se acaba:

- En el paso 1 se simplificó desde la fila 1 hasta la fila 3 y se redujo desde la fila 2 hasta la fila 3.
- En el paso 2 se simplificó desde la fila 2 hasta la fila 3 y se redujo solamente la fila 3.
- En el paso 3 se simplificó solamente la fila 3 y la reducción no se hace, pues en este caso la fila a reducir sería la número 4 y como no existe, entonces el bucle for no se ejecuta.

```python
    def gauss_jordan(self):
        inicioFilaSimplificacion = -1
        finalFilaSimplificacion = self.ecuaciones
        elementoFilaSimplificacion = -1

        inicioFilaReduccion = 0
        finalFilaReduccion = self.ecuaciones
        filaPivote = -1

        columnas = self.variables

        while columnas >= 1:
            inicioFilaSimplificacion += 1
            finalFS = finalFilaSimplificacion
            elementoFilaSimplificacion += 1

            # Simplificar la fila actual dividiendo por el elemento de la columna correspondiente
            for filaS in range(inicioFilaSimplificacion, finalFS):
                filaSimplificada = self.matrizReducir[filaS]
                filaSimplificada = filaSimplificada / filaSimplificada[elementoFilaSimplificacion]
                self.matrizReducir[filaS] = filaSimplificada

            print(f"\nResultado de la simplificacion: ")
```

```python
            print(self.matrizReducir)

            inicioFilaReduccion += 1
            finalFR = finalFilaReduccion
            filaPivote += 1

            # Reducir las filas restantes restando la fila pivote multiplicada por el elemento correspondiente
            for filaR in range(inicioFilaReduccion, finalFR):
                filaReducida = self.matrizReducir[filaR]
                filaReducida = filaReducida - self.matrizReducir[filaPivote]
                self.matrizReducir[filaR] = filaReducida

            print(f"\nResultado de la Reduccion: \n{self.matrizReducir}")

            columnas -= 1
        print("\nSegunda parte del Gauss Jordan:")
        self.gauss_jordan_segunda_parte()
```

Listing 3: Método para resolver el sistema de ecuaciones

Como se explicó antes, en el método gauss_jordan se inicializan las variables que se usarán para simplificar y reducir la matriz mientras que el número de variables sea mayor o igual a 1.

```python
    def gauss_jordan_segunda_parte(self):
        inicioFilaSimplificacion = 0
        finalFilaSimplificacion = self.ecuaciones
        elementoFilaSimplificacion = self.variables

        inicioFilaReduccion = 0
        finalFilaReduccion = self.ecuaciones
        filaPivote = self.ecuaciones

        columnas = self.variables

        while columnas >= 1:
            inicioFS = inicioFilaSimplificacion
            finalFilaSimplificacion -= 1
            elementoFilaSimplificacion -= 1

            # Simplificar la fila actual dividiendo por el elemento de la columna correspondiente
            for filaS in reversed(range(inicioFS, finalFilaSimplificacion + 1)):
```

```
                filaSimplificada = self.matrizReducir[filaS]
                filaSimplificada = filaSimplificada / filaSimplificada[elementoFilaSimplificacion]
                self.matrizReducir[filaS] = filaSimplificada

            print(f"\nResultado de la simplificacion: ")
            print(self.matrizReducir)

            inicioFR = inicioFilaReduccion
            finalFilaReduccion -= 1
            filaPivote -= 1

            # Reducir las filas restantes restando la fila pivote multiplicada por el elemento correspondiente
            for filaR in reversed(range(inicioFR, finalFilaReduccion)):
                filaReducida = self.matrizReducir[filaR]
                filaReducida = filaReducida - self.matrizReducir[filaPivote]
                self.matrizReducir[filaR] = filaReducida

            print(f"\nResultado de la Reduccion: \n{self.matrizReducir}")
            columnas -= 1
```

Listing 4: Método para resolver el sistema de ecuaciones

En el método gauss_jordan_segunda_parte se realiza el mismo proceso que en el método gauss_jordan pero en sentido contrario, es decir, se simplifica y reduce desde la última fila hasta la primera. Python en la función range() crea una lista iterable, desde la primera fila hasta la última, sin embargo, necesitamos iterar desde la última fila hasta la primera, por eso se usa la función reversed() que invierte el orden de la lista.

```
    # Solicitar el numero de ecuaciones y variables al usuario
    ecuaciones = int(input("Inserte el numero de ecuaciones: "))
    variables = int(input("Inserte el numero de variables: "))

    # Crear una instancia de la clase SistemaEcuaciones y resolver el sistema
    sistema = SistemaEcuaciones(ecuaciones, variables)
    sistema.recoger_datos()
    sistema.resolver()
```

Listing 5: Método para resolver el sistema de ecuaciones

Por último se crea el objeto sistema, haciendo una instancia a la clase SistemaEcuaciones y se llama al método recoger_datos para que el usuario pueda ingresar los valores de las ecuaciones y variables.

Probando el código

Para probar el código, se van a usar los ejercios de la página x del libro algebra lineal de Stanley I. Grossman, 7ma edición:

References

[1] Torres Solís, M., Villalobos Castillo, N. (s.f). Factorización LU. Universidad del Bío-Bío. Recuperado de http://repobib.ubiobio.cl/jspui/bitstream/123456789/1811/1/Torres_Solis_Marcos.pdf