**Universidad Autónoma de Nuevo León**

**Facultad de Ciencias Físico Matemáticas**

**Análisis Numérico**

**Proyecto 2:** Birge-Vieta

**Grupo:** 037

**Alumno Matricula**

Carlos Enrique Castillo Mayorga 1965541

Maximiliano Rada Moreno 1723609

Monterrey, Nuevo León, al 12 de marzo del 2025

**Índice**

[Introducción 2](#_Toc190802764)

[Marco teórico 3](#_Toc190802765)

[1. Sistemas de ecuaciones lineales 3](#_Toc190802766)

[2. Métodos de resolución de sistemas de ecuaciones 3](#_Toc190802767)

[3. El método Montante 4](#_Toc190802768)

[3.1. Principios fundamentales del método Montante 4](#_Toc190802769)

[3.2. Proceso de aplicación del método Montante 4](#_Toc190802770)

[4. Aplicaciones del método Montante 5](#_Toc190802771)

[5. Ventajas y desventajas del método Montante 5](#_Toc190802772)

[5.1. Ventajas 5](#_Toc190802773)

[5.2. Desventajas 5](#_Toc190802774)

[Código 6](#_Toc190802775)

[Manual de usuario 13](#_Toc190802776)

[Requisitos 13](#_Toc190802777)

[Ejecución del Programa 13](#_Toc190802778)

[Uso 13](#_Toc190802779)

[Mensajes de Error 15](#_Toc190802780)

[Ejemplo de ejecución 16](#_Toc190802781)

[Ejecución exitosa 16](#_Toc190802782)

[Ejecución matriz no invertible 17](#_Toc190802783)

[Ejecución con matriz con diagonal con 0 18](#_Toc190802784)

[Ejecución con matriz con solo 0 19](#_Toc190802785)

[Calcular otra matriz 19](#_Toc190802786)

[Diagrama de Flujo 20](#_Toc190802787)

[Conclusión 21](#_Toc190802788)

[Carlos Enrique Castillo Mayorga 21](#_Toc190802789)

[Maximiliano Rada Moreno 21](#_Toc190802790)

[Bibliografía 21](#_Toc190802791)

# Introducción

El método de Birge-Vieta es una técnica numérica utilizada para encontrar las raíces de polinomios. Este método es una variante del método de Newton-Raphson, específicamente diseñada para polinomios, lo que permite una convergencia más rápida y eficiente. Fue desarrollado por el matemático estadounidense George D. Birkhoff y el ingeniero brasileño Vicente Vieta en la década de 1930. El método de Birge-Vieta es particularmente útil en aplicaciones donde se requiere una alta precisión en la determinación de las raíces de polinomios, como en la ingeniería, la física y la economía.

El método de Birge-Vieta se basa en la aplicación iterativa del método de Newton-Raphson, pero con una optimización que aprovecha la estructura polinómica para reducir el número de operaciones necesarias en cada iteración. Esto se logra mediante el uso de la división sintética, que permite calcular tanto el valor del polinomio como su derivada en un punto dado de manera eficiente.

# Marco teórico

## 1. Polinomios y sus raíces

Un polinomio de grado n se expresa de la siguiente manera:

Donde an, an-1, … , a0 son los coeficientes del polinomio y x es la variable. Las raíces del polinomio son los valores de xque satisfacen la ecuación P(x)=0.

## 2. Métodos de resolución de raíces de polinomios

Existen varios métodos para encontrar las raíces de polinomios, los cuales pueden clasificarse en métodos exactos y métodos numéricos. Entre los métodos numéricos se encuentran:

* **Método de Newton-Raphson:** Un método iterativo que utiliza la derivada del polinomio para aproximar las raíces.
* **Método de la bisección:** Divide el intervalo en dos partes y selecciona el subintervalo que contiene la raíz.
* **Método de la secante:** Similar al método de Newton-Raphson, pero utiliza una aproximación de la derivada.
* **Método de Birge-Vieta:** Una variante del método de Newton-Raphson optimizada para polinomios.

## 3. El método Birge-Vieta

El método de Birge-Vieta es una técnica desarrollada por George D. Birkhoff y Vicente Vieta en la década de 1930 para encontrar las raíces de polinomios mediante un procedimiento basado en la división sintética y el método de Newton-Raphson.

### 3.1. Principios fundamentales del método Birge-Vieta

El método de Birge-Vieta se basa en los siguientes principios:

* **Uso de la división sintética:** Permite calcular eficientemente el valor del polinomio y su derivada en un punto dado.
* **Iteración de Newton-Raphson:** Utiliza la fórmula de Newton-Raphson para aproximar las raíces del polinomio.
* **Optimización para polinomios:** Aprovecha la estructura polinómica para reducir el número de operaciones necesarias en cada iteración.

### 3.2. Proceso de aplicación del método Birge-Vieta

El procedimiento para aplicar el método Montante consiste en los siguientes pasos:

1. Se construye la *matriz aumentada* del sistema.
2. Se elige el primer pivote (generalmente si es distinto de cero).
3. Se transforman los elementos de la matriz mediante la regla:

Donde es el pivote anterior y los valores se actualizan en cada iteración.

1. Se repite el proceso hasta obtener una matriz diagonal con los coeficientes de las incógnitas en la diagonal principal.
2. Se extraen las soluciones del sistema a partir de la matriz resultante.

Este procedimiento garantiza una solución exacta sin la necesidad de operaciones que introduzcan errores de redondeo.

## 4. Aplicaciones del método Montante

El método Montante es útil en diversas áreas, entre ellas:

* **Álgebra lineal**: Resolución de sistemas de ecuaciones en modelos matemáticos.
* **Computación y algoritmos**: Implementación en programas de resolución numérica de ecuaciones.
* **Criptografía**: Uso en cálculos donde se requiere preservar la exactitud de los valores.
* **Ingeniería y economía**: Modelado de sistemas con múltiples variables.

## 5. Ventajas y desventajas del método Montante

### 5.1. Ventajas

* Mayor estabilidad numérica: Reduce los errores de redondeo al evitar divisiones sucesivas.
* Fácil implementación: Sigue un esquema sistemático y uniforme.
* Mayor precisión: Mantiene coeficientes enteros o racionales durante todo el proceso.

### 5.2. Desventajas

* Menos eficiente en grandes sistemas: Para sistemas con muchas ecuaciones, otros métodos pueden ser más rápidos.
* Dependencia de pivotes adecuados: Si un pivote es cero, es necesario intercambiar filas para evitar errores.

# Código

#### Método Montante

import copy

def metodo\_montante(matriz, rest):

#Obtenemos el tamano de la matriz

n = len(rest)

#Generamos una matriz identidad de tamano n

identidad = [[1 if i == j else 0 for j in range(n)]for i in range(n)]

#Declaramos el primer pivote

pivote\_ant = 1

#Creamos una iteracion para resolver la matriz

for k in range(n):

#Guardamos el valor de la posicion (k,k) que sera con el que estemos trabajando (pivote actual)

actual = matriz[k][k]

#Guardamos la columna k con la cual tambien estaremos trabajando

otras = [fila[k] for fila in matriz]

#Creamos otra iteracion en la cual haremos que todas las pocisiones en la columna k a excepcion de (k,k) sean 0

for j in range(n):

if k != j:

matriz[j][k] = 0

#Creamos otra iteracion donde los valores por arriba de (k,k) en la digonal sean igual a este

for i in range(k):

matriz[i][i] = actual

# Modificamos los valores de la matriz usando el método de Montante

for i in range(n):

if i != k:

for j in range(k+1, n):

matriz[i][j] = (actual \* matriz[i][j] - otras[i] \* matriz[k][j]) // pivote\_ant

#Realizamos el mismo procedimiento pero con la matriz identidad

for i in range(n):

if i != k:

for j in range(n):

identidad[i][j] = (actual \* identidad[i][j] - otras[i] \* identidad[k][j]) // pivote\_ant

#Modificamos el pivote anterior

pivote\_ant = actual

#Una vez obtenida la determinanate y la Adjunta obtenemos la inversa

Determinante = pivote\_ant

Adjunta = identidad

inverversa = copy.deepcopy(Adjunta)

for i in range(n):

for j in range(n):

inverversa[i][j] = inverversa[i][j] / Determinante

Valores\_resultantes = [0] \* n

for i in range(n):

for j in range(n):

Valores\_resultantes[i] = Valores\_resultantes[i] + (rest[j] \* Adjunta[i][j])/Determinante

return Valores\_resultantes, Determinante, Adjunta, inverversa

#### Cambio de lugar de las Filas

def pivoteo(matriz, actual, siguiente):

#Realizamos el cambio de filas en la matriz

matriz[actual], matriz[siguiente] = matriz[siguiente], matriz[actual]

return matriz

#### Validación si la matriz tiene algun 0 en la Diagonal

def validar (matriz):

n = len(matriz)

for i in range(n):

if matriz[i][i] == 0:

return True

return False

#### Función principal para resolver la matriz

def solucionar\_matriz(matriz, rest):

n = len(rest)

#Validamos que la matriz no algun 0 en la diagonal

cero = validar(matriz)

#Si llegara a tener un 0 en la diagonal realizamos movimientos en las filas

actual = 0 #Iniciamos el contador de la casilla que vamos a cambiar

siguiente = actual + 1 #Iniciamos el contador de la casilla con la cual la vamos a cambiar

iteraciones = 0 #iniciamos el contador de iteraciones para realizar los cambios

while cero and iteraciones < n:

#Llamamos a la funcion pivoteo que realizara los cambios en la matriz

matriz = pivoteo(matriz, actual, siguiente)

#Volvemos a validar matriz

cero = validar(matriz)

#Si hay 0 avanzamos en la matriz para realizar cambios

if cero:

#Si siguiente no ha llegado al ultima fila de la matriz avanzamos

if siguiente + 1 < n:

actual = actual +1

siguiente = actual +1

#Si no volvemos a cambiar desde la primer fila e incrementamos las iteraciones

else:

actual = 0

siguiente = actual +1

iteraciones += 1

#Salimos de la funcion

#Validamos que si podemos realizar el metodo motante

if cero:

print("ERROR: la matriz ingresada no es apta para el metodo montante, ya que no hay una combinacion de filas con una diagonal que no tenga 0, intente nuevamente con otra matriz")

return False

else:

#Se calcula el resultado por el metodo montante

resultado = metodo\_montante(matriz, rest)

return resultado

#### Main (Captura de la matriz)

def main():

     op = 1

     while op == 1:

        evaluador1 = False

        #Se ingresa el tamaño de la matriz

        print("Escriba el tamaño de la matriz:")

        #Se comprueba que el valor sea un numero entero positivo

        while evaluador1 == False:

            try:

                n = int(input())

                if n > 0:

                    evaluador1 = True

                else:

                    print("ERROR: Debe de ser un numero positivo: Intente nuevamente:")

            except ValueError:

                print("ERROR: Debe ingresar un numero entero positivo. Intente nuevamente:")

        #Se crea la Matriz junto a los valores de sus funciones

        matriz = [[0 for \_ in range(n)] for \_ in range(n)]

        valores = [0 for \_ in range(n)]

        #Se evalua que la matriz sea invertible

        while True:

            for i in range(n):

                for j in range(n+1):

                    evaluador2 = False

                    if(j<n):

                        print(f"Ingrese el valor de X [{i+1}][{j+1}]:")

                        #se comprueba que los valores dentro de la matriz sean numeros

                        while evaluador2 == False:

                            try:

                                matriz [i][j] = float(input())

                                evaluador2 = True

                            except ValueError:

                                print("ERROR: Debe ingresar un numero. Intente nuevamente:")

                    else:

                        print(f"Ingrese el valor de la funcion {i+1}:")

                        #se evalua que los valores de los resultados sean numeros

                        while evaluador2 == False:

                            try:

                                valores [i] = float(input())

                                evaluador2 = True

                            except ValueError:

                                print("ERROR: Debe ingresar un numero. Intente nuevamente:")

            try:

                resultado, determinante, adjunta, inversa = solucionar\_matriz(matriz, valores)

                break

            except ZeroDivisionError:

                print("ERROR: la matriz ingresada no es invertible por ende no es apta para el metodo montante, intente nuevamente con otra matriz")

except TypeError:

                print("")

        if resultado:

            print("El resultado es el siguiente")

            for i in range(n):

                print(f"X{i+1} = {resultado[i]}")

            print(f"\nEl determinante de la matriz es:{determinante}\n")

            print("\nLa adjunta de la matriz es:\n")

            for j in range(n):

                for k in range(n):

                    print(f"[{adjunta[j][k]}]", end=" ")

                print("")

            print("\nla inversa de la matriz es:\n")

            for p in range(n):

                for q in range(n):

                    print(f"[{inversa[p][q]}]", end=" ")

                print("")

        print('¿Desea calcular otra matriz?')

        opcion = input("Si / No\n")

        if opcion == 'si' or opcion == 'Si' or opcion == 'SI' or opcion == 'sI':

            op = 1

        else:

            op = 0

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# Manual de usuario

## Requisitos

Python 3 instalado en el sistema.

Conocimientos básicos sobre matrices y sistemas de ecuaciones lineales.

## Ejecución del Programa

Para ejecutar el programa, simplemente ejecute el archivo Python en una terminal o entorno de desarrollo:

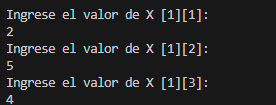
*python nombre\_del\_archivo.py*

## Uso

1. **Ingresar el tamaño de la matriz**: El usuario debe ingresar un número entero positivo que represente la dimensión de la matriz cuadrada.

*Ejemplo:*

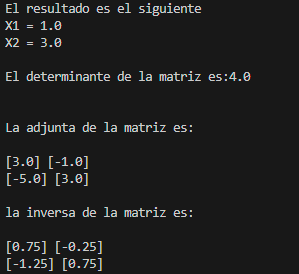
1. **Ingresar los coeficientes de la matriz**: El programa solicitará los valores de cada elemento de la matriz, verificando que sean números válidos.

*Ejemplo:*

1. **Ingresar los valores de la función** (*vector de resultados*): Cada ecuación del sistema tiene un resultado, que también debe ser ingresado como un número.

*Ejemplo:*

1. **Cálculo del resultado**: El programa aplicará el Método de Montante para calcular los valores de las variables y mostrará los resultados en pantalla.

*Ejemplo:*

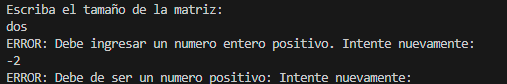
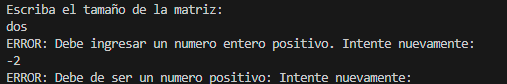
1. **Desea ingresar otra matriz**: Al terminar de calcular el resultado de evaluar una matriz y cuando la matriz no es invertible, se le pregunta a el usuario si quiere volver a realizar el proceso.

*Ejemplo:*

## Mensajes de Error

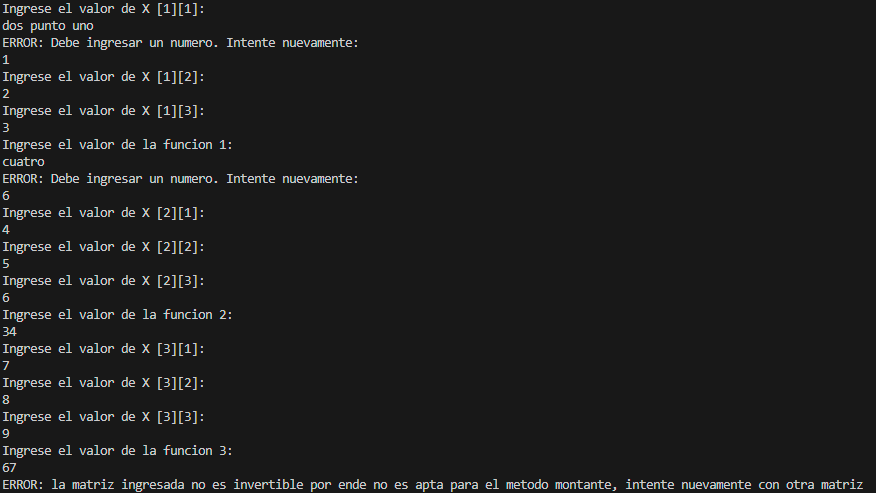
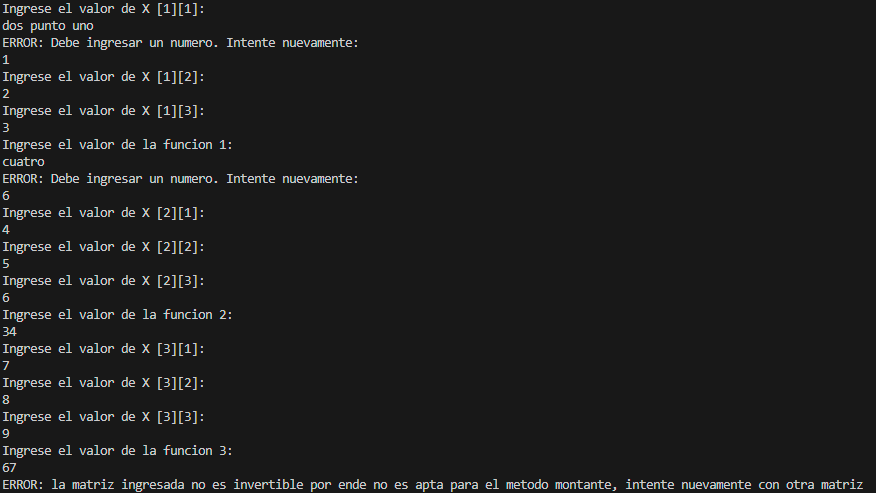
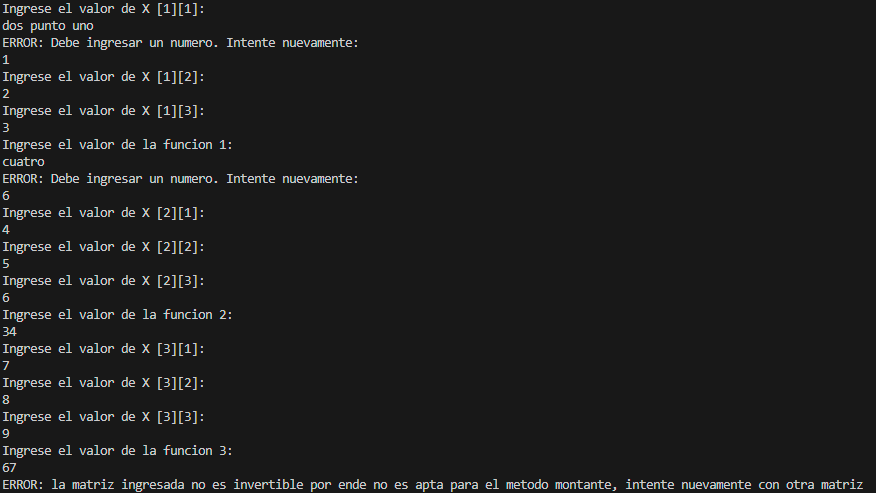
* Si el usuario ingresa un valor no numérico, se mostrará un mensaje de error y se solicitará nuevamente el dato.

*Casos de aparición :*



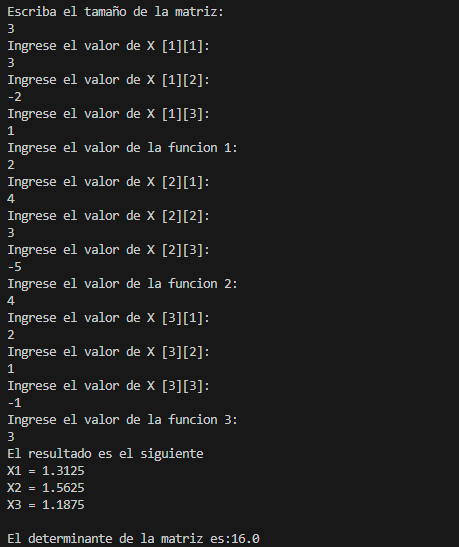
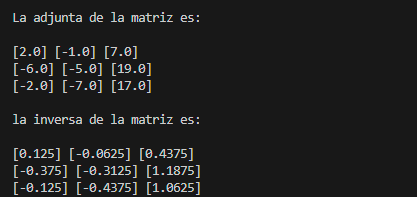
* Si la matriz ingresada no es invertible, se mostrará un mensaje indicando que no es apta para el método y se pedirá una nueva matriz.

*Casos de aparición:*

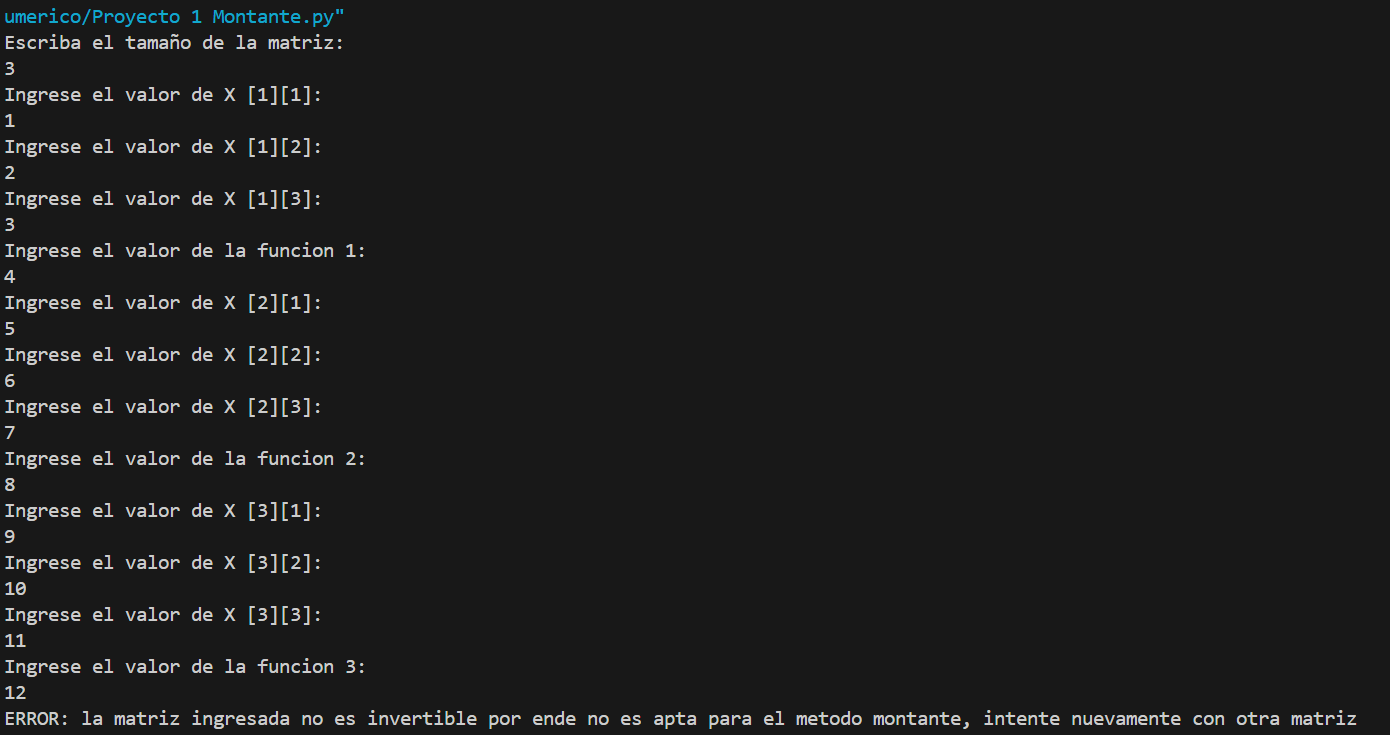


# Ejemplo de ejecución

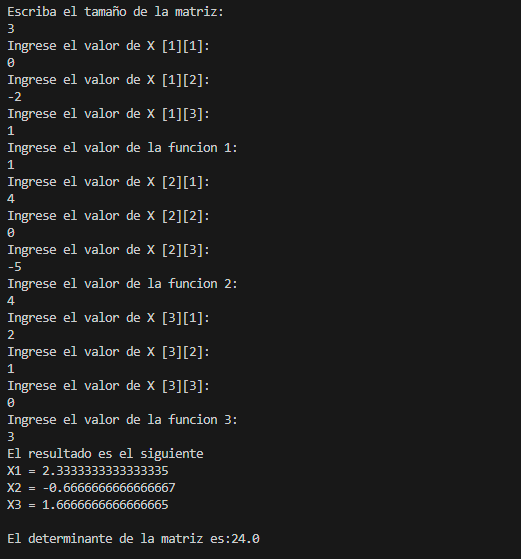
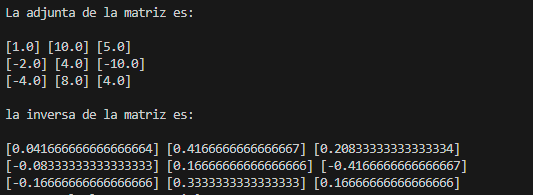
## Ejecución exitosa

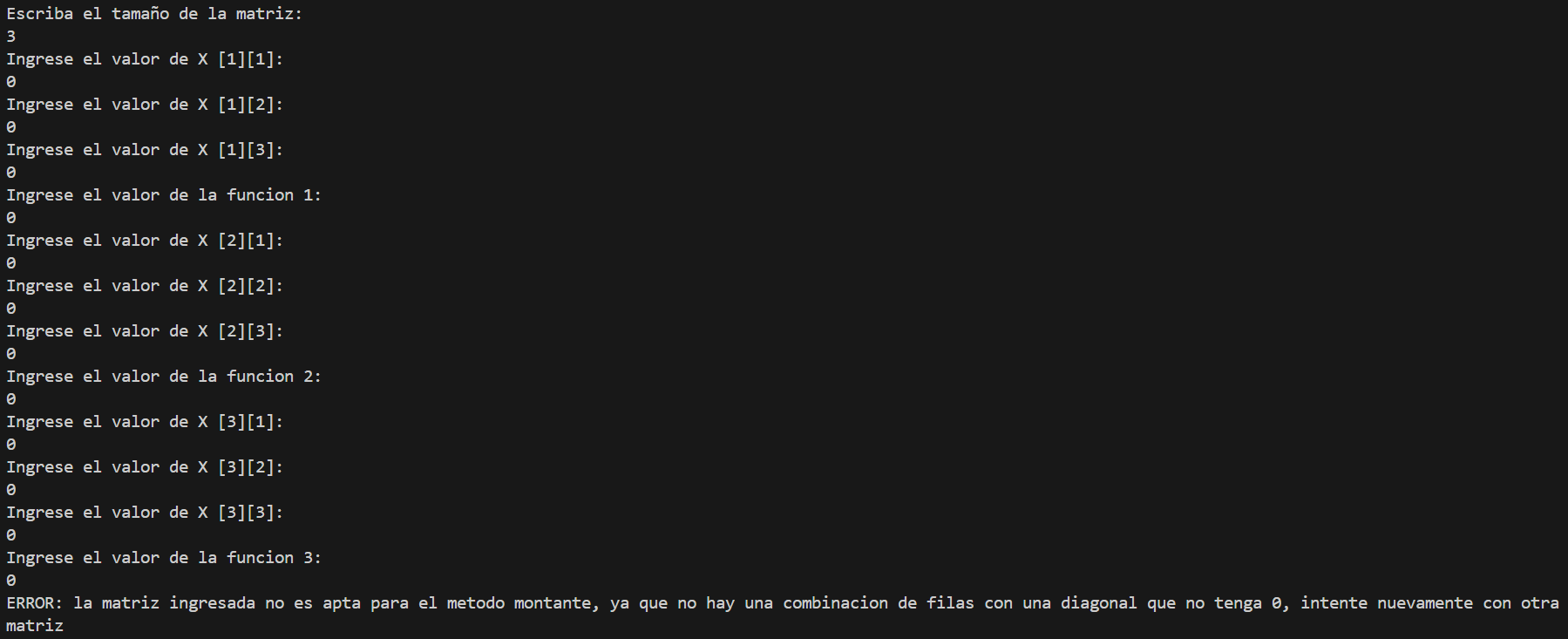
## Ejecución matriz no invertible



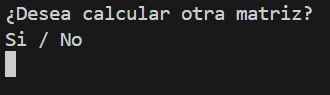
## Ejecución con matriz con diagonal con 0



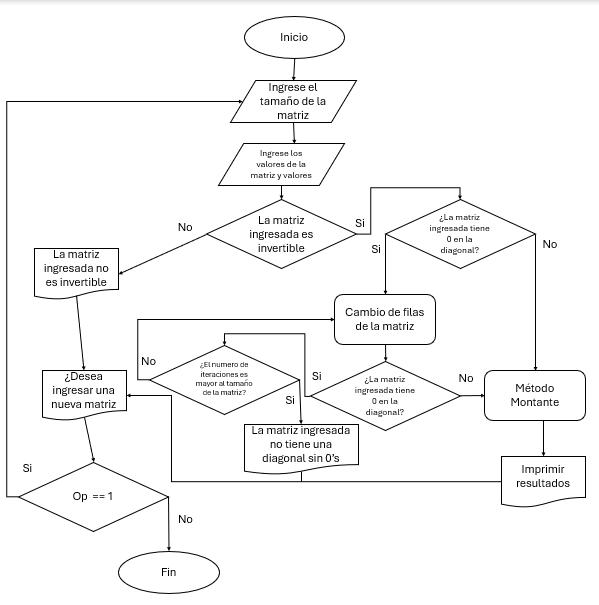
## Ejecución con matriz con solo 0



## Calcular otra matriz



# Diagrama de Flujo



# Conclusión

### Carlos Enrique Castillo Mayorga

En este proyecto aprendí sobre métodos numéricos y como usar programación para resolver problemas matemáticos. Fue interesante ver como los conceptos teóricos se pueden aplicar en la práctica. Al principio me pareció un poco complicado, pero con el tiempo logre entender mejor como funciona el metodo Montante para resolver sistemas de ecuaciones. Al final, puede hacer un programa que funciona , lo cual me da mucha satisfacción.

### Maximiliano Rada Moreno

Durante el desarrollo del proyecto pudimos observar el funcionamiento del método montante, cuáles son sus beneficios y cuáles son sus desventajas frente a otro tipo de métodos, así mismo desarrollamos un proyecto en el lenguaje Python con el cual implementamos este método para la resolución de matrices n x n, validado que estas pudieran ser resueltas, pudiendo ser inversas y hubiera una combinación de sus filas en la cual no aparezcan 0’s en la diagonal, cabe resaltar que por este método además de la solución de cada una de las variables poder obtener las determínate, la matriz inversa, y la matriz adjunta. La parte compleja de realizar este proyecto no fue el lenguaje o la resolución de la matriz a través del método escogido, la verdadera dificultad que llegamos a tener al realizar el código fue la validación de la matriz, que esta fuera ideal para resolver a través de este método, ya que tuvimos que validar varias cosas para que esto pudiera ser resuelto de esta manera, fuera de eso el manejo del código en la resolución del método montante fue fácil. Otro de los problemas que llegamos a tener fue que al tener que obtener las soluciones, al manejar decimales y no fracciones obtenemos resultados aproximados a los esperados, sin embargo, no podemos optar por métodos de redondeo ya que afectaría a las soluciones con soluciones decimales.

# Bibliografía

**Montante Pardo, R.** (1976). *Un método para la solución de sistemas de ecuaciones lineales*. Universidad Autónoma de Nuevo León.

**Chapra, S. C., & Canale, R. P.** (2020). *Métodos numéricos para ingenieros* (8ª ed.). McGraw-Hill.