UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FCFM

Universidad Autónoma de Nuevo León

Facultad de Ciencias
Físico-Matemáticas

Análisis Numérico
Proyecto #1: Solución de Sistemas de Ecuaciones por el Método de Montante

Equipo:

Melissa Mayte Luis de Luna	2034250
Diego Andrés Pérez Arellano	1899770
Kenneth Martin Rodriguez Garcia	1916780

Profesora: María del Carmen Martínez Cejudo

Grupo: 031

Índice

Historia del Método Bareiss-Montante

En 1968, el matemático Edward W. Bareiss descubrió un método para obtener el determinante de una matriz escalonada o de un sistema de ecuaciones lineales, que a su vez permitía también obtener la matriz inversa y, por consiguiente, la solución al sistema de ecuaciones.

Sin embargo, el método se popularizó en latinoamérica gracias al profesor René Mario Montante Pardo de la Facultad de Ingeniería Mecánica y Eléctrica de la UANL. Montante descubrió el método por cuenta propia 5 años después que Bareiss, pero por popularizar el método es conocido como método Montante, Bareiss o Bareiss-Montante.

"Yo pensé que eran puros determinantes. Pero cuando vi que resolvía ecuaciones lineales y con números enteros, dije: esto va a llegar muy lejos. Y ahora anda en todo el mundo. En la computadora es el más exacto del mundo" (René Montante).

A pesar de que este método es muy bueno computacionalmente por evitar muchos errores de redondeo, el profesor Montante nunca lo utilizó con computadoras pues, en sus propias palabras "yo no sé computación".

Algoritmo

Para utilizar el algoritmo de Montante se requiere que la matriz cumpla con algunas condiciones.

1) La matriz utilizada debe ser cuadrada de tamaño nXn
2) Los coeficientes deben ser enteros

La segunda condición es importante en particular, porque incluso por computadora se puede resolver de manera exacta el problema a diferencia del método Gauss-Jordan donde rápidamente trabajamos con fracciones o decimales que las computadoras tienden a redondear.

El método Bareiss-Montante tiene dos variaciones principales: una que obtiene únicamente el determinante pero no requiere de memoria auxiliar, y una variación que requiere de memoria auxiliar para una matriz nXn pero que es capaz de obtener la matriz inversa de la matriz original.

Implementación sin memoria auxiliar (sólo determinante)

Esta implementación tiene la ventaja de que no requiere de estructuras auxiliares para

1. Inicio
2. Entrada: M[n][n]
3. $M_{0,0}$ = 1 ($M_{0,0}$ es una variable única)
4. Para k desde 1 hasta n-1:
    a. Para i desde k+1 hasta n-1:
        i. Para j desde k+1 hasta n-1:
            1. $M_{i,j} = \frac{Mi,jMk,k - Mi,kMk,j}{Mk-1,k-1}$
5. Salida: $M_{n,n}$ es el determinante de la matriz
6. Fin

Implementación con memoria auxiliar (determinante y matriz inversa)

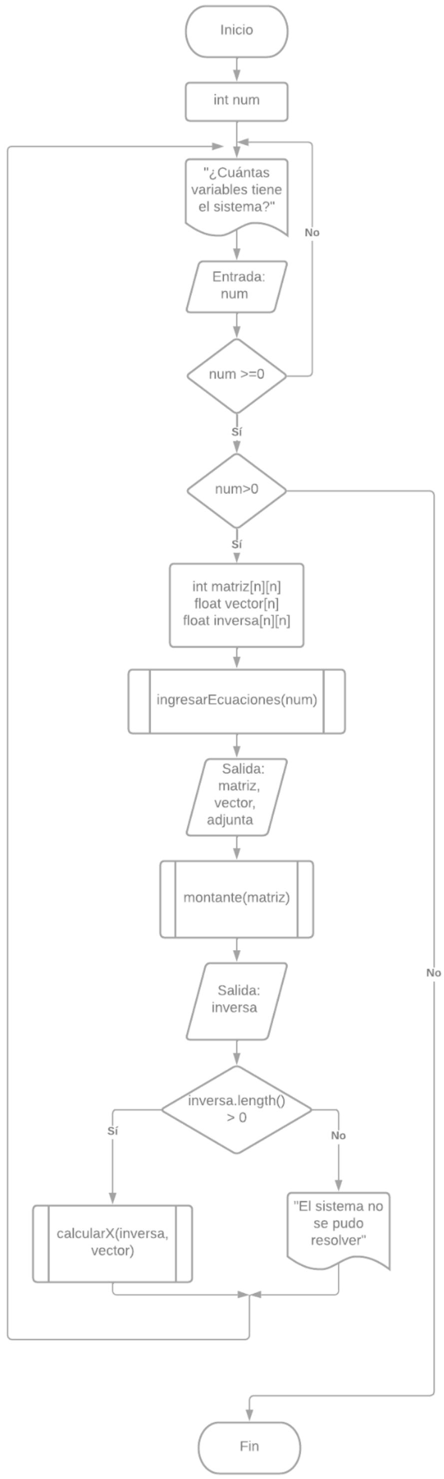
Esta implementación del algoritmo permite obtener el determinante de la matriz de coeficientes y su matriz invertida. Este método asume que ninguno de los pivotes es igual a 0; sin embargo, es posible también es posible modificar el algoritmo para cambiar de renglón en caso de que esto ocurra, tomando en cuenta de que el determinante cambiará de signo.

1. Inicio
2. Entrada: M[n][n] (matriz de coeficientes) y Adj[n][n] (matriz identidad)
3. int nuevaM[n][n], nuevaAdj[n][n], det
4. $M_{0,0}$ = 1 ($M_{0,0}$ es una variable única)
5. int pivote = 1
6. Para k desde 1 hasta n-1:
    a. Para i desde 1 hasta n-1:
        i. Para j desde 1 hasta n-1:
            1. Si i == k
                a. nueva$M_{i,j}$ = $M_{i,j}$
                b. nueva$Adj_{i,j}$ = $Adj_{i,j}$
            2. Si no
                a. nueva$M_{i,j}$ = $\frac{Mk,kMi,j - Mi,k\, Mk,j}{pivote}$
                b. nueva$Adj_{i,j}$ = $\frac{Mk,k\ Adji,j - Mi,k\ Adjk,j}{pivote}$
    b. M = nuevaM
    c. Adj = nuevaAdj
7. Salida: $M_{k,k}$ (determinante), Adj (matriz inversa)
8. Fin

Diagramas de Flujo

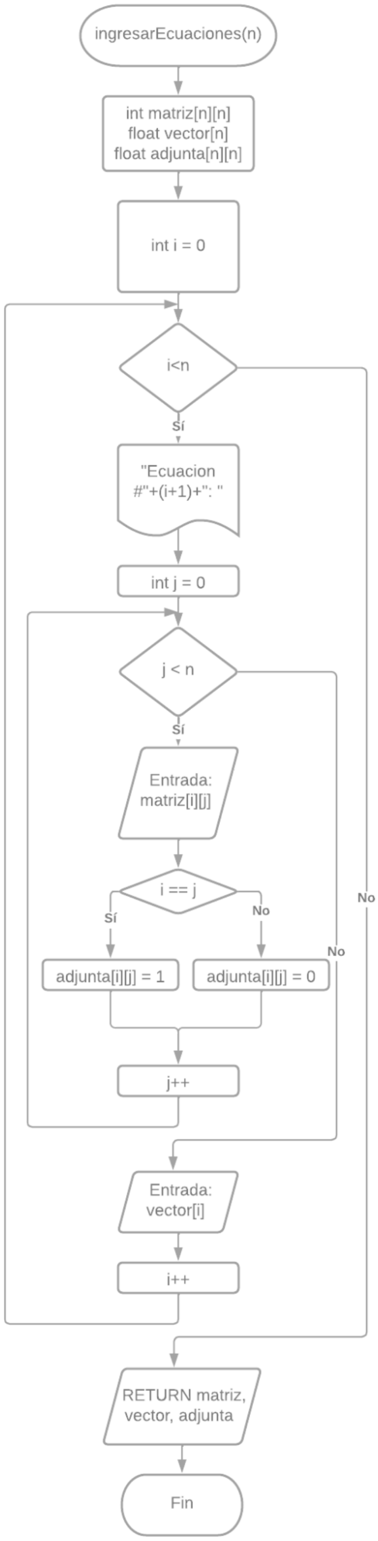
Para comenzar el desarrollo del programa, primero hicimos diagramas de flujo para tener una idea sobre cómo implementar el método.

Diagrama Principal:

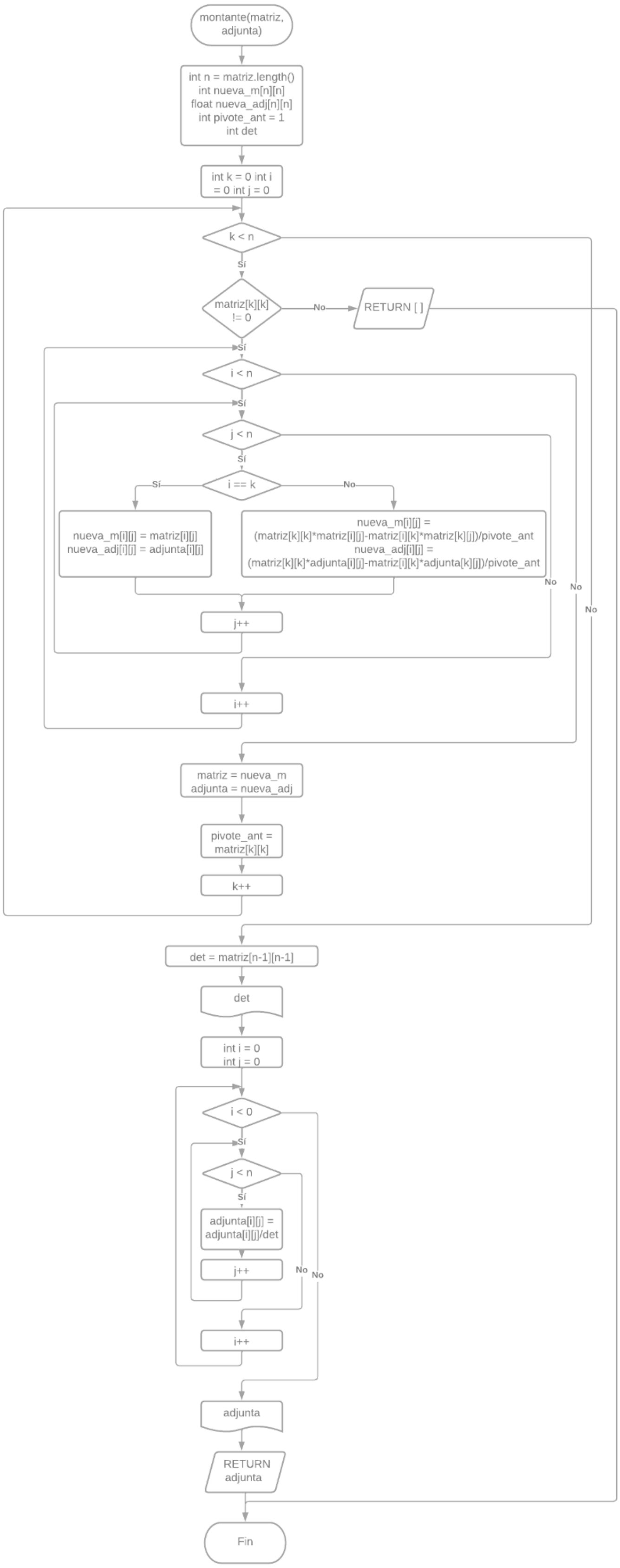


Subrutinas

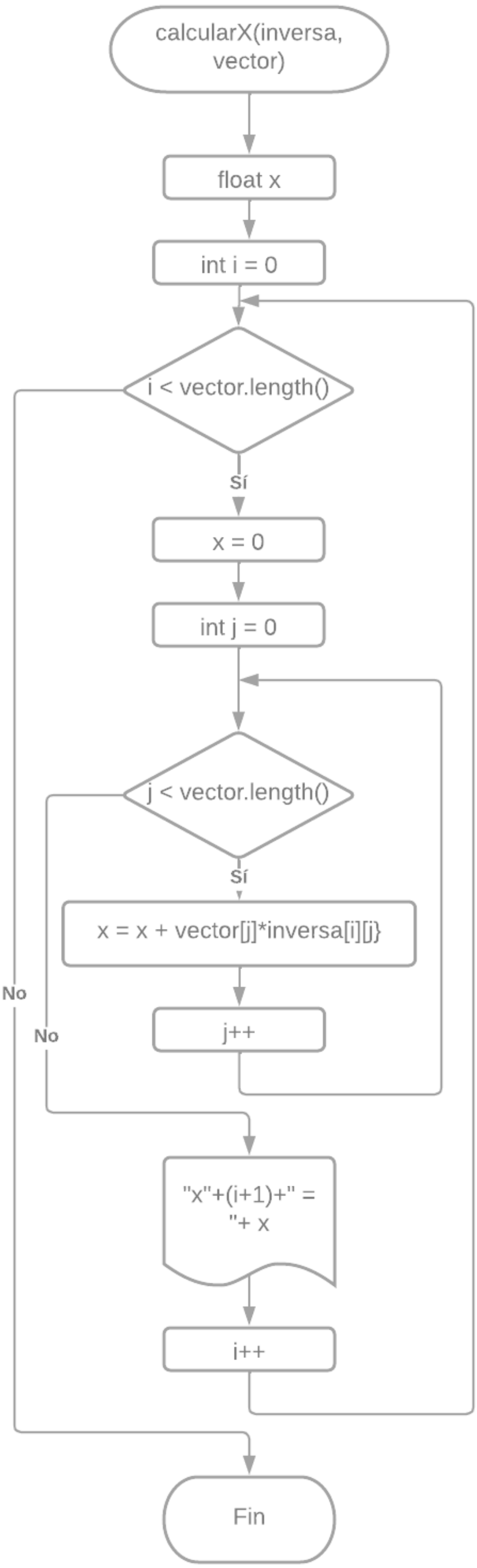
ingresarEcuaciones



montante


montante(matriz, adjunta)
int n = matriz.length()
int nueva_m[n][n]
float nueva_adj[n][n]
int pivote_ant = 1
int det
int k = 0 int i = 0 int j = 0
k < n
Sí
matriz[k][k] != 0
No
RETURN [ ]
Sí
i < n
Sí
j < n
Sí
i == k
Sí
No
nueva_m[i][j] = matriz[i][j]
nueva_adj[i][j] = adjunta[i][j]
nueva_m[i][j] = (matriz[k][k]*matriz[i][j]-matriz[i][k]*matriz[k][j])/pivote_ant
nueva_adj[i][j] = (matriz[k][k]*adjunta[i][j]-matriz[i][k]*adjunta[k][j])/pivote_ant
No
No
No
j++
i++
matriz = nueva_m
adjunta = nueva_adj
pivote_ant = matriz[k][k]
k++
det = matriz[n-1][n-1]
det
int i = 0
int j = 0
i < 0
Sí
j < n
Sí
adjunta[i][j] = adjunta[i][j]/det
j++
No
No
i++
adjunta
RETURN adjunta
Fin

calcularX


calcularX(inversa, vector)
float x
int i = 0
i < vector.length()
Sí
x = 0
int j = 0
j < vector.length()
Sí
x = x + vector[j]*inversa[i][j}
j++
No
No
"x"+(i+1)+" = "+ x
i++
Fin

Implementación en Python 3

Para realizar una implementación computacional del método, decidimos utilizar Python 3. La decisión no fue tomada de manera aleatoria, sino que consideramos las siguientes ventajas y desventajas sobre la herramienta.

Ventajas

1. Sintaxis muy legible que facilita el desarrollo
2. Es un lenguaje interpretado, por lo que no requiere ser compilado antes de ejecutar un script, ni de adaptarlo a una plataforma en específico
3. Es un lenguaje de tipado dinámico, por lo que no es estricto al asignar valores a una variable.
4. Se pueden regresar múltiples valores de una función sin necesidad de un contenedor o estructura (desempaquetado de tuplas)
5. Utilizar contenedores (ej: arreglos) como parámetros no requiere utilizar punteros, ya que todas los objetos y contenedores se manejan por referencia.

Desventajas

1. El tipado dinámico vuelve el programa propenso a errores de diseño.
2. Los números flotantes tienen solo tanta precisión, por lo que en ocasiones puede no llegarse a la respuesta correcta, aunque sí a una muy cercana (Ej: 0.1+0.2 != 0.3)
3. Para programas más pesado, no es tan eficiente en memoria ni en rendimiento como otros lenguajes (ej.: C++)

Código Fuente

```
#Funciones o Métodos

#Validar que regrese un entero positivo
def numVariables()->int:
   while True:
       try:
           num = int(input("¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: "))
       except ValueError:
           print("Error, debe ingresar un número entero positivo")
       else:
           if num > 0:
```

```
                return num
            else:
                print("El número de variables debe ser mayor a 
0")

# Preguntar al usuario si desea seguir en el programa o salir
def continuar()->str:
    # Puede regresar una cadena o un booleano
    sn = ""
    sn = input("Desea ingresar otro sistema de ecuaciones? S/n: 
").lower()
    while sn != "s" and sn != "n":
        sn = input("S/n: ").lower()
    #Al devolver una comparación, regresa un booleano
    return sn == "s"

#Validar que el coeficiente ingresado sea entero
def validarInt(x_i)->int:
    while True:
        try:
            coef = int(input("Coeficiente de x"+str(x_i)+": "))
        except ValueError:
            print("Entrada inválida, ingrese un número entero 
como coeficiente")
        else:
            return coef

#Validar que el vector de términos independientes sea un 
número flotante
def validarFloat(ecuacion: str)->float:
    while True:
        try:
            val = float(input(ecuacion))
        except:
            print("Entrada inválida, ingrese un número entero o 
flotante")
        else:
```

```
            return val

def imprimirMatriz(matriz: list):
    for vec in matriz:
        renglon = "|"
        for num in vec:
            renglon +=" "+str(num)+" "
        renglon += "|"
        print(renglon)
    print("")

def imprimirMatrizYAdjunta(matriz: list, adj: list):
    n = len(matriz)
    for i in range(n):
        renglon = "|"
        for j in range(n):
            renglon +=" "+str(matriz[i][j])+" "
        renglon += "|"
        for j in range(n):
            renglon += " "+str(adj[i][j])+" "
        renglon += "|"
        print(renglon)

# Generar la matriz correspondiente al problema, preguntando
por los coeficientes
#Generar la matriz identidad asociada a la matriz del problema
def ingresarEcuaciones(num: int)->tuple:
    #Inicializar la matriz cuadrada
    matriz = [[0 for j in range(num)] for i in range(num)]
    #Inicializar el vector
    vector = [0 for i in range(num)]

    for i in range(num):
        ecuacion = ""
        print("\nEcuación #"+str(i+1))
        for j in range(num):
            matriz[i][j] = validarInt(j+1)
```

```
            if j > 0 and matriz[i][j] >= 0:
                ecuacion += "+"
            ecuacion += str(matriz[i][j])+"x"+str(j+1)+" "
        ecuacion += "= "
        vector[i] = validarFloat(ecuacion)
        print("-"*20)
    #Python puede regresar una tupla que puede ser
"desempacada" al recibirla
    return matriz, vector

# Método principal
# Decidir si será un método recursivo o solamente iterativo
# Revisar excepciones.
def montante(matriz: "list[list]")->list:
    #Almacenar el tamaño de la matriz cuadrada
    n = len(matriz)
    #Inicializar la matriz adjunta como una matriz identidad de
tamaño nXn
    adjunta = [[0 if i!=j else 1 for i in range(n)] for j in
range(n)]
    #Inicializar el pivote anterior
    pivote_ant = 1
    #Inicializar las matrices auxiliares para realizar las
operaciones
    nueva_m = [["x" for i in range(n)] for j in range(n)]
    nueva_adj = [["x" for i in range(n)] for j in range(n)]
    #En caso de haber un cambio de renglon, cambiaría el signo
del determinante
    #Con la variable "signo" podemos llevar el control para
obtener el determinante de la matriz original
    signo = 1
    #Ciclo para controlar cuantas veces se debe de realizar el
proceso
    imprimirMatrizYAdjunta(matriz, adjunta)
    for k in range(n):
        print("\nPivote anterior: "+str(pivote_ant))
```

```
        #No hay necesidad de una variable "pivote actual" pues
es el elemento ubicado en k,k
        print("Pivote actual: "+str(matriz[k][k]))
        #Verifica que el pivote no sea 0, pues causaría
problemas en la siguiente iteracion
        if matriz[k][k] == 0:
            for i in range(k+1, n):
                if matriz[i][k] != 0:
                    print("\n=>")
                    print("Cambio de renglón\n\n=>\n")
                    #Cuando se encuentra un renglón que cumpla
con la confición deseada, se intercambian sus contenidos
                    aux = matriz[k].copy()
                    matriz[k] = matriz[i].copy()
                    matriz[i] = aux.copy()
                    aux = adjunta[k].copy()
                    adjunta[k] = adjunta[i].copy()
                    adjunta[i] = aux.copy()
                    imprimirMatrizYAdjunta(matriz, adjunta)
                    del aux
                    #Actualizamos el signo del determinante
                    signo *=-1
                    break
            else:
                if matriz[k][k] == 0:
                    #Si el pivote = 0 persiste, el método
regresa una matriz vacía
                    return []
        #Ciclos del método principal
        for i in range(n):
            for j in range(n):
                if i == k:
                    #Si el renglon de la iteración es igual al
del pivote actual, se copian los numeros a la matriz nueva
                    nueva_m[i][j] = matriz[i][j]
                    nueva_adj[i][j] = adjunta[i][j]
                else:
```

```
                    #Si no, se realizan las operaciones
                    nueva_m[i][j] = (matriz[k][k]*matriz[i][j]
- matriz[i][k]*matriz[k][j])/pivote_ant
                    nueva_adj[i][j] =
(adjunta[i][j]*matriz[k][k] -
matriz[i][k]*adjunta[k][j])/pivote_ant
        #Se copian los elementos de las matrices auxiliares a
las principales
        for i in range(n):
            for j in range(n):
                matriz[i][j] = nueva_m[i][j]
                adjunta[i][j] = nueva_adj[i][j]
        pivote_ant = matriz[k][k]
        print("\n=>\n")
        imprimirMatrizYAdjunta(matriz, adjunta)
    print("-"*20)
    print("Determinante: "+str(matriz[n-1][n-1]*signo))
    for i in range(n):
        for j in range(n):
            adjunta[i][j] = adjunta[i][j]/matriz[n-1][n-1]
    print("-"*20)
    print("Matriz inversa: ")
    imprimirMatriz(adjunta)
    return adjunta

#Con la matriz inversa ya podemos calcular los valores de las
variables
def calcularVariables(inversa: list, vector: list):
    print("-"*20)
    for i in range(len(vector)):
        x = 0
        for j in range(len(vector)):
            x += vector[j]*inversa[i][j]
        print("x"+str(i+1)+" = "+str(x))
    print("-"*20)
```

```
#Llamada al programa, lo equivalente a la función main() en
otros lenguajes
if __name__=="__main__":
   print("Método Bareiss-Montante")
   # Inicializar variables
   cont = True
   while cont:
       #Decidir tamaño de la matriz
       num = numVariables()
       #Desempacamos tuplas para recibir dos componentes de la
respuesta
       matriz, vector = ingresarEcuaciones(num)
       inversa = montante(matriz)
       if len(inversa) > 0:
           calcularVariables(inversa, vector)
       else:
           print("No se pudo resolver el sistema de
ecuaciones, porque el pivote actual es igual a 0")
       #Preguntar si desea continuar
       cont = continuar()
   # Fin
```

Manual de Usuario

El programa es un script de Python 3, por lo que no requiere ser compilado ni de un archivo ejecutable para utilizarlo. Sólo requiere tener instalado el intérprete de Python 3 para poder ejecutar el programa.

Antes de poder utilizar el programa es necesario descargarlo del siguiente repositorio: https://github.com/KBlacksmith/Montante/blob/main/montante.py. A su vez, es necesario tener una instalación de Python 3 en el equipo, o utilizar un ambiente de desarrollo web.

Una vez descargado el archivo, podemos utilizar Python 3 desde una línea de comandos (en Windows) o terminal (en Linux o Mac). Nos movemos a la ubicación del archivo y utilizamos el comando: “python3 montante.py”. Otras alternativas son utilizar un ambiente de desarrollo como IDLE, o incluso un editor como VS Code para ejecutar el script.

Ahora si podemos utilizar el programa, donde lo primero que veremos será el siguiente mensaje de bienvenida:

```
Método Bareiss-Montante
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?:
```

Ahora podemos ingresar la cantidad de variables para el sistema. El programa sólo permite números enteros mayores a 0. Si ingresamos un número menor o igual a 0, un número decimal, o cualquier otra entrada, el programa arrojará un mensaje de error, y le seguirá preguntando al usuario por un número hasta que reciba una entrada válida.

```
Método Bareiss-Montante
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: -1.5
Error, debe ingresar un número entero positivo
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?:
```

Una vez que el usuario ingrese un número válido de variables, el programa le solicitará al usuario llenar los datos de “n” variables por ecuación (más un número por término independiente de la ecuación), para “n” ecuaciones, siendo “n” el número ingresado por el usuario.

De acuerdo al algoritmo de Bareiss-Montante, los coeficientes de las ecuaciones deben ser enteros para que durante el proceso no requieran del uso de decimales. Por esta razón, si el usuario intenta ingresar un número flotante recibirá un mensaje de error como en el caso anterior y le solicitará nuevamente ingresar un número entero para el coeficiente. Sin embargo, esta restricción no existe para el término independiente por lo que solo debemos validar que sea un número, si es entero o flotante es irrelevante.

```
Método Bareiss-Montante
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: -1.5
Error, debe ingresar un número entero positivo
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: 3

Ecuación #1
Coeficiente de x1: 3
Coeficiente de x2: 6
Coeficiente de x3: -1
3x1 +6x2 -1x3 = 12.5
-------------------
```

Este proceso debe ser repetido “n” veces para llenar todas las ecuaciones.

```
Método Bareiss-Montante
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: -1.5
Error, debe ingresar un número entero positivo
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: 3

Ecuación #1
Coeficiente de x1: 3
Coeficiente de x2: 6
Coeficiente de x3: -1
3x1 +6x2 -1x3 = 12.5
-------------------

Ecuación #2
Coeficiente de x1: 7
Coeficiente de x2: -1
Coeficiente de x3: 9
7x1 -1x2 +9x3 = 2
-------------------

Ecuación #3
Coeficiente de x1: -2
Coeficiente de x2: -1
Coeficiente de x3: -1
-2x1 -1x2 -1x3 = -6
```

Una vez ingresemos el último dato, el problema desplegará el proceso realizado en la pantalla, y finalmente mostrará el resultado del sistema de ecuaciones. El proceso puede ser visualizado para que el usuario pueda identificar los pasos seguidos por el programa para llegar a la solución.

```
--------------------
| 3  6  -1 | 1  0  0 |
| 7  -1  9 | 0  1  0 |
| -2  -1  -1 | 0  0  1 |

Pivote anterior: 1
Pivote actual: 3

=>

| 3  6  -1 | 1  0  0 |
| 0.0  -45.0  34.0 | -7.0  3.0  0.0 |
| 0.0  9.0  -5.0 | 2.0  0.0  3.0 |

Pivote anterior: 3
Pivote actual: -45.0

=>

| -45.0  0.0  -53.0 | -1.0  -6.0  -0.0 |
| 0.0  -45.0  34.0 | -7.0  3.0  0.0 |
| -0.0  0.0  -27.0 | -9.0  -9.0  -45.0 |

Pivote anterior: -45.0
Pivote actual: -27.0

=>

| -27.0  -0.0  -0.0 | 10.0  7.0  53.0 |
| -0.0  -27.0  -0.0 | -11.0  -5.0  -34.0 |
| -0.0  0.0  -27.0 | -9.0  -9.0  -45.0 |
--------------------
```

El programa imprime 3 resultados: el determinante, la matriz inversa de la matriz original, y los valores de las incógnitas.

```
--------------------
Determinante: -27.0
--------------------
Matriz inversa:
| -0.37037037037037035  -0.25925925925925924  -1.962962962962963 |
| 0.4074074074074074  0.18518518518518517  1.2592592592592593 |
| 0.3333333333333333  0.3333333333333333  1.6666666666666667 |

--------------------
x1 = 6.629629629629631
x2 = -2.0925925925925926
x3 = -5.166666666666667
--------------------
Desea ingresar otro sistema de ecuaciones? S/n:
```

En algunos casos, el sistema de ecuaciones no puede ser resuelto, por lo que se imprime en pantalla un mensaje que indica esto mismo. Esto ocurre cuando un pivote es igual a 0 y el renglón no puede ser intercambiado por uno debajo de este.

```
Pivote anterior: 1
Pivote actual: 0.0
No se pudo resolver el sistema de ecuaciones, porque el pivote actual es igual a 0
Desea ingresar otro sistema de ecuaciones? S/n:
```

Independientemente del resultado obtenido, el programa le solicita al usuario indicar si desea continuar o no, representado con una “s” para “Sí” o una “n” para “No”. No importa si la letra es mayúscula o minúscula, el programa lo puede validar. **Nota:** El programa acepta “s”, “S”, “n” y “N”, pero no acepta “Sí” o “No”, ni ninguna variación de mayúsculas o minúsculas de estas dos palabras.

Ejemplo

Para verificar que el programa ha sido exitoso, resolveremos un problema analógicamente y lo verificaremos corriendo el programa.
Consideremos el siguiente sistema de 3 ecuaciones y 3 incógnitas:

Ecuación #1: $2x_1 + 3x_2 + 4x_3 = 18$
Ecuación #2: $7x_1 + 8x_2 + x_3 = 22$
Ecuación #3: $4x_1 + 2x_2 + 5x_3 = 33$

Los podemos representar de forma matricial, y con su matriz adjunta:

$$\left|\begin{array}{ccc|ccc} 2 & 3 & 4 & 1 & 0 & 0 \\ 7 & 8 & 1 & 0 & 1 & 0 \\ 4 & 2 & 5 & 0 & 0 & 1 \end{array}\right|$$

Pivote anterior = 1
Pivote actual = 2

=>

$$\left|\begin{array}{ccc|ccc} 2 & 3 & 4 & 1 & 0 & 0 \\ 0 & -5 & -26 & -7 & 2 & 0 \\ 0 & -8 & -6 & -4 & 0 & 2 \end{array}\right|$$

Pivote anterior = 2
Pivote actual = -5

=>

$$\left|\begin{array}{ccc|ccc} -5 & 0 & 29 & 8 & -3 & 0 \\ 0 & -5 & -26 & -7 & 2 & 0 \\ 0 & 0 & -89 & -18 & 8 & -5 \end{array}\right|$$

Pivote anterior = -5
Pivote actual = -89

=>

$$\left|\begin{array}{ccc|ccc} -89 & 0 & 0 & 38 & -7 & -29 \\ 0 & -89 & 0 & -31 & -6 & 26 \\ 0 & 0 & -89 & -18 & 8 & -5 \end{array}\right|$$

Con esto encontramos que el determinante de la matriz de coeficientes es -89. Ahora dividimos cada elemento de la matriz adjunta entre el determinante, para obtener la matriz inversa
Inversa =

$$\left|\begin{array}{ccc} -38/89 & 7/89 & 29/89 \\ 31/89 & 6/89 & -26/89 \\ 18/89 & -8/89 & 5/89 \end{array}\right|$$

Una vez obtuvimos la matriz inversa, la multiplicamos por el vector de términos independientes del sistema de ecuaciones para obtener los valores de las variables.
Entonces obtenemos que:
x_1 = 4.797752808
x_2 = -1.887640449
x_3 = 3.516853932

Resolución Mediante el Programa

Una vez que obtuvimos los resultados de manera analógica, ingresaremos el sistema de ecuaciones al programa para resolverlo y verificar que efectivamente funciona.

Primero le indicamos al programa que nuestro sistema es de 3 variables.

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: /home/kenneth/git-repos/montante/montante.py =========
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: 3

Ecuación #1
Coeficiente de x1:
```

Después llenamos los datos para los coeficientes y términos independientes del sistema:

```
¿Cuántas variables tiene el sistema de ecuaciones (0 para salir)?: 3

Ecuación #1
Coeficiente de x1: 2
Coeficiente de x2: 3
Coeficiente de x3: 4
2x1 +3x2 +4x3 = 18
--------------------

Ecuación #2
Coeficiente de x1: 7
Coeficiente de x2: 8
Coeficiente de x3: 1
7x1 +8x2 +1x3 = 22
--------------------

Ecuación #3
Coeficiente de x1: 4
Coeficiente de x2: 2
Coeficiente de x3: 5
4x1 +2x2 +5x3 = 33
```

Finalmente, corroboramos que los resultados obtenidos son los mismos en ambos casos.

```
--------------------
| 2  3  4 | 1  0  0 |
| 7  8  1 | 0  1  0 |
| 4  2  5 | 0  0  1 |

Pivote anterior: 1
Pivote actual: 2

=>

| 2  3  4 | 1  0  0 |
| 0.0  -5.0  -26.0 | -7.0  2.0  0.0 |
| 0.0  -8.0  -6.0 | -4.0  0.0  2.0 |

Pivote anterior: 2
Pivote actual: -5.0

=>

| -5.0  0.0  29.0 | 8.0  -3.0  -0.0 |
| 0.0  -5.0  -26.0 | -7.0  2.0  0.0 |
| 0.0  0.0  -89.0 | -18.0  8.0  -5.0 |

Pivote anterior: -5.0
Pivote actual: -89.0

=>

| -89.0  0.0  -0.0 | 38.0  -7.0  -29.0 |
| -0.0  -89.0  -0.0 | -31.0  -6.0  26.0 |
| 0.0  0.0  -89.0 | -18.0  8.0  -5.0 |
--------------------
Determinante: -89.0
--------------------
Matriz inversa:
| -0.42696629213483145  0.07865168539325842  0.3258426966292135 |
| 0.34831460674157305  0.06741573033707865  -0.29213483146067415 |
| 0.20224719101123595  -0.0898876404494382  0.056179775280898875 |

--------------------
x1 = 4.797752808988764
x2 = -1.8876404494382006
x3 = 3.5168539325842696
--------------------
```

En efecto, podemos ver que los resultados obtenidos son los mismos y, por tanto, nuestro programa es capaz de replicar el algoritmo de Bareiss-Montante

Bibliografía

- *Bareiss, E. H. (n.d.). Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. American Mathematical Society. Retrieved February 19, 2022, from https://www.ams.org/journals/mcom/1968-22-103/S0025-5718-1968-0226829-0/S0025-5718-1968-0226829-0.pdf*
- *Campbell, S. (2022, January 1). Python vs C++: What's the difference? Guru99. Retrieved February 21, 2022, from https://www.guru99.com/python-vs-c-plus-plus.html*

- *Salazár, L. (2020, January 30). El Método Montante, de la UANL Para El Mundo. Punto U - Universidad Autónoma de Nuevo León. Retrieved February 20, 2022, from https://puntou.uanl.mx/legado-uni/el-metodo-montante-de-la-uanl-para-el-mundo/*