

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN.
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS.

ANÁLISIS NUMÉRICO.
PROYECTO 1: MÉTODO MONTANTE.

PROFESORA: MARÍA DEL CARMEN MARTÍNEZ CEJUDO

ALUMNO	MATRICULA
FRANCISCO JAVIER MARTÍNEZ PALOMAR.	1937837
KEVIN ALEXIS NÁJERA ÁBREGO.	1798194
ELÍ ISRAEL DELGADO ESCÁRCEGA.	1821213

San Nicolás de los Garza, Nuevo León.

Índice

Índice.....	1
Introducción.....	2
Método escogido.....	3
¿Qué es el método Montante?.....	3
¿Por qué se escogió el método montante?	3
Código.	4
Diagrama de flujo.	11
Manual de usuario y funcionamiento.....	14
Abrir la aplicación.....	14
Funcionamiento.....	15
Uso del programa (Análisis de Mallas).....	20
Conclusión.....	21
Referencias.	22

Introducción.

El presente escrito es una descripción de un programa realizado para el curso de Análisis Numérico. Dicho programa utiliza el método Montante para resolver un sistema de ecuaciones dado, además de obtener el determinante de la matriz de coeficientes, así como su adjunta y determinante.

Se presenta una breve descripción de que es el método utilizado, por qué se escogió, presentación y descripción de la codificación, diagrama de flujo del procedimiento principal, imágenes de muestra de dicho programa en ejecución, así como la explicación de su uso.

Método escogido.

¿Qué es el método Montante?

El Método Montante, llamado así debido a su descubridor, René Mario Montante Pardo, es un algoritmo del álgebra lineal para determinar las soluciones de un sistema de ecuaciones lineales, encontrar matrices inversas, matrices de adjuntos y determinantes.

El método fue descubierto en el 1973 por René Mario Montante Pardo, egresado de la Facultad de Ingeniería Mecánica y Eléctrica. La característica principal del Método Montante es que trabaja con enteros, lo cual hace que el resultado sea exacto, aunque se resuelva con computadora, ya que evita que se redondeen los números.

¿Por qué se escogió el método montante?

Como equipo, decidimos escoger el método Montante debido a su mayor característica, la cual es el manejo de números enteros durante el procedimiento. Es común que, trabajando con otros métodos para resolución de sistemas de ecuaciones, como eliminación Gaussiana simple o Gauss-Jordan; se obtengan cocientes o valores decimales durante el proceso, cosa que no sucede con el procedimiento elegido, a menos que los valores iniciales ingresados sean fracciones o decimales.

Además de lo anterior, la información extra que nos proporciona, como lo es el determinante, la matriz de adjunta y de coeficientes; lo hacen muy completo y eficaz para determinar si un sistema tiene solución única o no, por ejemplo, cuando el determinante es igual a cero.

Código.

Nuestro código fue desarrollado en lenguaje C y compilado como una aplicación ejecutable, es decir, con extensión .exe

Haciendo un poco a un lado las validaciones del programa y la lectura de datos en este, nos queda el código principal que describe el algoritmo del método de Montante. A continuación, aremos una breve descripción de en qué consiste dicho código.

Describiendo la primera línea del código podemos decir que esta función es la primera que se ejecutara a la hora de mandar a resolver el sistema de ecuaciones, está hecha para resolver un entero que nos indicara si fue o no posible resolver el sistema de ecuaciones. Además, en esta misma línea podemos ver que recibirá dos matrices de una dimensión no especificada para posteriormente ser utilizadas en el proceso, así como un vector donde se almacenaran los resultados de las ecuaciones a la hora de hacer la lectura.

```
int leerMatriz(float matrizI[MAX][MAX], float matrizA[MAX][MAX], float
igualdad[MAX]){
```

A continuación, tenemos algunas de las variables que utilizaremos para leer opciones elegidas por el usuario, numero de ecuaciones a introducir por este mismo.

```
    int n, option;
    char str[MAX];

    do {

        clear();
        printf("\n\n\tMETODO MONTANTE\n");
```

En la sección de código siguiente, preguntamos al usuario el número de ecuaciones que desea ingresar, en caso de no ingresar un dato invalido esta pregunta se volverá a imprimir en pantalla.

```
        do {
            printf(" \nIngrese la cantidad de ecuaciones: ");
            in(str);
            n = valInt(str);
        } while(n < 2);

        clear();
        int i, j;
```

A continuación, se le pide al usuario que ingrese los datos de las ecuaciones, para esto se utilizan dos ciclos for que se repetirán cada uno n veces, donde n es el número de ecuaciones que se pidió anteriormente, esto debido a que la dimensión del sistema de ecuaciones sin tomar en cuenta la parte de la igualdad será de nxn, pudiendo de esta manera obtener una inversa de la matriz asociada y por ende una solución. De igual manera a como se indicó anteriormente, en caso de ingresar un dato no valido, la sentencia do while se ejecutará y volverá a pedir el valor en cuestión.

```

printf("\n\n\t Ingrese los coeficientes de las ecuaciones");
for(i = 0 ; i < n ; i++){
    printf("\n\n\t Ecuaci%cn %d\n", 162,i+1);
    for(j = 0 ; j < n ; j++){
        do{
            printf(" x%d: ",j+1);
            in(str);
            matrizI[i][j] = valFloat(str);
        }while(!isNumber(str));
    }
    /*Leo las igualdades de cada ecuacion*/
    printf(" = ");
    in(str);
    igualdad[i] = valFloat(str);
}

```

La sentencia clear(); la utilizamos para borrar lo que se encuentra en pantalla actualmente y dar paso a las siguientes instrucciones o información, como lo es en este caso, ahora mostraremos el sistema de ecuaciones que se leyó anteriormente para darle la oportunidad al usuario de verificar la información. En caso de que no sea el sistema de ecuaciones correcto, el usuario deberá indicarlo eligiendo la opción 2, del menú y de esta forma se ejecuta la sentencia do while y te pide que ingreses nuevamente la cantidad de ecuaciones y los coeficientes de esta.

```

clear();
printf("\n\n %cSu sistema de ecuaciones es el siguiente?\n\n",168);
/*Imprimo la matriz que forman los coeficientes de la ecuacion*/
for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < n ; j++){
        if(j==n-1) printf(" %0.2f x%d = ",matrizI[i][j],j+1);
        else printf(" %0.2f x%d + ",matrizI[i][j],j+1);
    }
    /*Imprimo el vector solucion */
    printf("%0.2f\n",igualdad[i]);
}
do{
    printf("\n\n [1.Si 2.No]: ");

```

```

        in(str);
        option = valInt(str);
    }while(option!=1 && option!=2);

    }while(option != 1);
    return n;
}

```

Si el usuario confirma la correcta lectura de su sistema de ecuaciones, esta parte del programa finaliza y manda de regreso el valor de n, que recordaremos es el número de ecuaciones que queremos ingresar, este valor será de mucha ayuda para los cálculos del método montante.

La Función metodoMontante describe el proceso general del método, de igual manera que la anterior devuelve un entero que indica que los cálculos has finalizado correctamente, y que el sistema tiene una solución, o en caso contrario que este no la tiene. Pasamos por parámetros la matriz de elementos asociados al sistema de ecuaciones, una matriz auxiliar y el número de ecuaciones n.

```

int metodoMontante(float matrizI[MAX][MAX], float matrizA[MAX][MAX], float
igualdad[MAX], int n) {
    int i, j;
    system("cls");
    printf("\n\n SOLUCION DEL SISTEMA POR EL METODO MONTANTE\n\n");

```

En esta parte del codigo inicializo en cero todos los valores de la matriz identidad que se utiliza en el procedimiento de este método.

```

    /*Inicializo la matriz identidad de dimension nxn*/
    for(i = 0 ; i < n ; i++){
        for(j = 0 ; j < n ; j++){
            if(i==j) matrizA[i][j]=1.0;
            else matrizA[i][j]=0.0;
        }
    }

```

Posteriormente en esta parte imprimimos en pantalla el estado inicial de la matriz asociada al sistema de ecuaciones, extendiéndola con la matriz identidad al lado derecho de esta. De esta manera el usuario ve el proceso de principio a fin.

```

    /*Imprimo la matriz asociada al sistema con la identidad que extiende*/
    printf(" Estado inicial:\n\n");
    for(i = 0 ; i < n ; i++){
        for(j = 0 ; j < n ; j++){
            printf(" %0.2f\t",matrizI[i][j]);
        }
        printf("\t");
        for(j = 0 ; j < n ; j++){

```

```

printf(" %0.2f\t",matrizA[i][j]);
    }
    printf("\n");
}

```

Aquí inicializamos las variables de los pivotes que el algoritmo utiliza para las operaciones, inicializando el pivote anterior en 1 y definiendo el actual como el de la posición [k][k] de la matriz, que será el elemento de la matriz diagonal desde el primero hasta el último de esta, ya que notamos que el proceso tiene n corridas y los pivotes siempre son los de la matriz diagonal, donde k es el índice del ciclo for más externo que va haciendo el recorrido del proceso en cada una de las n corridas.

```

float pivAnt=1, pivAct;
int k;
for(k = 0 ; k < n ; k++){
pivAct=matrizI[k][k];

```

Aquí es la parte más importante del todo el proceso la cual describe el movimiento entre pivotes y operaciones cruzadas que irán cambiando los valores de la matriz que inicialmente ingresamos y la de la matriz identidad que tenemos al lado izquierdo de esta.

```

/*operaciones cruzadas para obtener numeros en la fila diferente a la del pivote*/
for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < 2*n ; j++){

```

En esta parte del código le decimos al programa que ignore la fila y columna en la que se encuentre el pivote, ya que solo cambiarán los valores del resto de la matriz, mientras que los valores que están debajo del pivote o por encima de este permanecen siendo ceros.

En el caso donde el elemento que los índices de los ciclos for de i y j recorren, no se encuentren en el valor de i y j en que se encuentra el pivote, entonces, se mandará a llamar la función de opCruz, la cual devolverá el valor de las operaciones cruzadas que generan su nuevo valor. En caso de que $j > n$, esto significa que la posición del índice ya está en la matriz identidad de la izquierda y también se debe de mandar a llamar la función de opCruz, solo que en este caso el valor que regrese lo asignaremos a la matrizA (matriz que inicialmente contiene a la identidad).

```

        if(i!=k && j!=k){
                                if(j<n){
matrizI[i][j]=opCruz(n,pivAnt,k,i,j,matrizI,matrizA);
                                }
                                else
matrizA[i][j-
n]=opCruz(n,pivAnt,k,i,j,matrizI,matrizA);
                                }
        }
    }
}

```


En esta parte del código se van imprimiendo las corridas, como se mencionó anteriormente el proceso conta de tantas corridas como numero de ecuaciones se tenga. Hasta que finalmente del lado derecho solo tendremos un mismo valor en la diagonal principal, el cual será el determinante de la matriz y en la matrizA que anteriormente tenía la identidad, ahora tendremos la traspuesta de la matrizI, la matriz inicial del sistema de ecuaciones.

```

/*Imprimo la corrida en la que voy, el método consta de n corridas*/
printf("\n\n Corrida %d:\n",k+1);
for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < n ; j++){
        if(j==k && i!=k) matrizI[i][j]=0;
        printf(" %0.2f\t",matrizI[i][j]);
    }
    printf("\t");
    for(j = 0 ; j < n ; j++){
        printf(" %0.2f\t",matrizA[i][j]);
    }
    printf("\n");
}
pivAnt=pivAct;
if(pivAnt==0.0) {
    return 0;
}
}
//system("color 0A");

```

A continuación, se imprimen los resultados obtenidos en el proceso anterior.

```

/*Imprimo los resultados obtenidos Det, Mt, M-1 y solucion.*/
printf("\n\n ##### RESULTADOS OBTENIDOS #####\n\n");
if(matrizI[0][0]!=0){
    printf(" det= %0.2f",matrizI[0][0]);
}

```

Esta parte del código imprime la matrizA, donde se formó la matriz traspuesta de la matriz inicial, de igual manera se utilizan dos ciclos for para recorrer todos los elementos de la misma, y cada que se termina de imprimir una fila se imprime un salto de línea.

```

printf("\n\n Matriz traspuesta del sistema:\n",k+1);
for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < n ; j++){
        printf(" %0.2f\t",matrizA[i][j]);
    }
    printf("\n");
}

```

Para la impresión de la matriz inversa se divide cada uno de los elementos de la matrizA, que contiene la matriz traspuesta, entre alguno de los elementos de la diagonal principal de matrizI, que como recordaremos contiene el determinante de la matriz. Por lo que obtendremos como resultado la matriz inversa donde antes estuvo la matriz traspuesta. Finalmente procedemos a imprimir el resultado como la matriz inversa.

```
printf("\n\n Matriz Inversa del sistema:\n",k+1);
for(i = 0 ; i < n ; i++){
    for(j = 0 ; j < n ; j++){
        matrizA[i][j] = matrizA[i][j]/matrizI[0][0];
        printf(" %0.2f\t",matrizA[i][j]);
    }
    printf("\n");
}
```

La solución del sistema la obtenemos en el siguiente fragmento de código, donde se utilizan dos ciclos for para recorrer la matriz inversa del sistema, cada elemento de la hilera de esta se va multiplicando por el elemento del vector de igualdades del sistema inicial y se van sumando hasta terminar con la hilera, esta primer suma representara el valor del primer elemento del vector solucion[] donde se almacenaran las n soluciones del sistema. En seguida se va imprimiendo el valor correspondiente de las soluciones.

```
printf("\n\n **SOLUCION DEL SISTEMA**:\n");
float solucion[ ];
for(i = 0 ; i < n ; i++){
    solucion[i]=0;
    for(j = 0 ; j < n ; j++){
        solucion[i] = solucion[i] + igualdad[j]*matrizA[i][j];
    }
    printf(" x%d = %f\n",i+1,solucion[i]);
}

return 1;
}
```

La Función opCruz, aunque es la más corta de las dos anteriores no es menos importante, y es que es en esta donde se realizan la mayoría de las operaciones del procedimiento. En esta función se realiza el calculo del nuevo valor de cada elemento que no se encuentra en la fila o columna de el pivote actual. Se devuelve un valor flotante ya que, aunque este método tiene la capacidad de trabajar solo con números enteros en todo el proceso, si entre los datos hay números flotantes estos pueden generar que se trabaje con números flotantes, por lo que el programa está preparado para tratar con dichos datos.

Como entrada en los parámetros de la función tenemos el número de ecuaciones, el valor del pivote anterior, las coordenadas del pivote actual, la posición i y j del elemento al cual se le calcula el valor y las matrices *matrizI* y *matrizA* con que se trabaja en el procedimiento.

```
float opCruz(int n, int pivAnt, int x, int i, int j, float matrizI[MAX][MAX], float
matrizA[MAX][MAX]) {
    float resultado=0.0;
```

En esta función existen dos posibilidades, que se trate del valor de un elemento de la matriz inicial (*matrizI*) o de la matriz identidad (*matrizA*), esto lo determinamos con el valor de j, y es que en la parte del código donde la mandamos a llamar este valor de j, varía hasta 2n, para recorrer la hilera completa de la *matrizI* y la *matrizA*. Por lo que si entra en el caso donde $j < n$ esto significa que está en la posición de un elemento de la *matrizI*.

El cálculo se determinó mediante el análisis de las operaciones que se realizan, se notó que las operaciones siguen siempre el mismo patrón, se multiplica el valor del pivote actual ($matrizI[x][x]$) por el valor de cualquier elemento que no esté en la fila o columna del pivote actual ($matrizI[i][j]$) y a esto le restamos el elemento que está en la columna del pivote actual pero en la fila del elemento al que se le calcula el valor ($matrizI[x][j]*matrizI[i][x]$) por el elemento que está en la columna del elemento al que se le calcula el valor y en la fila del pivote actual ($matrizI[i][x]$), y todo esto entre el valor de el pivote anterior (*pivAnt*). Lo anterior es lo mismo para el caso donde $j > n$ solo que se opera el elemento de la *matrizA* para los cálculos, y solo en el caso donde se utiliza j en las operaciones hay que restarle n para que se ubique en la posición real de la *matrizA*.

```
    if(j<n){

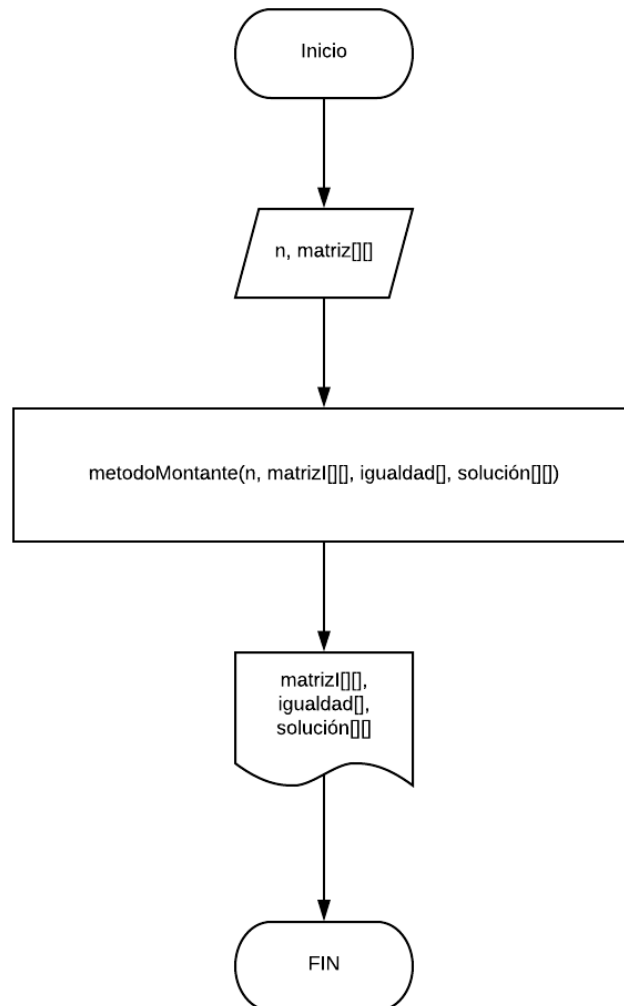
        resultado = (matrizI[x][x]*matrizI[i][j]-matrizI[x][j]*matrizI[i][x])/pivAnt;
    } else {
        resultado = (matrizI[x][x]*matrizA[i][j-n]-matrizA[x][j-n]*matrizI[i][x])/pivAnt;
    }

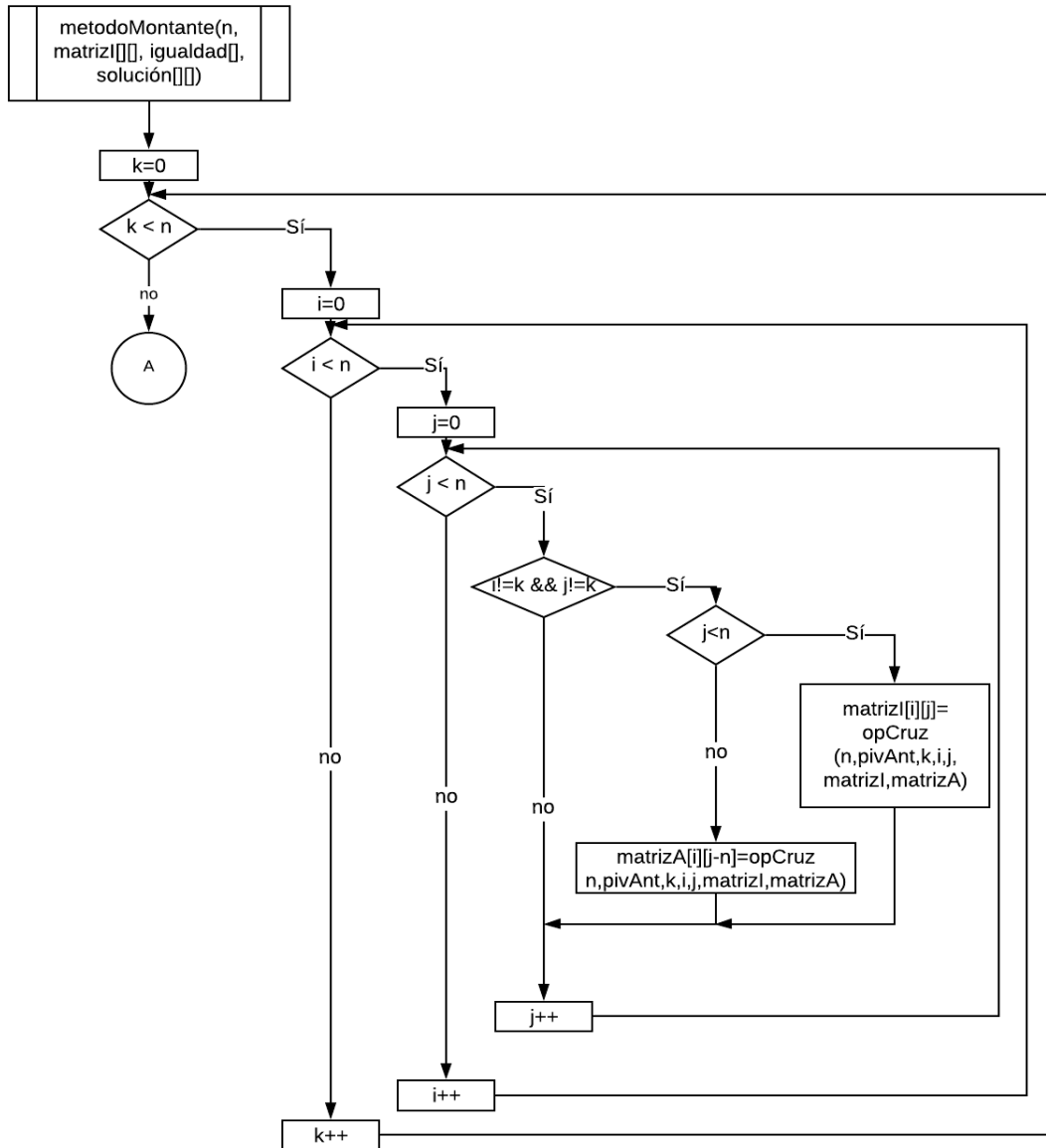
    return resultado;
}
```

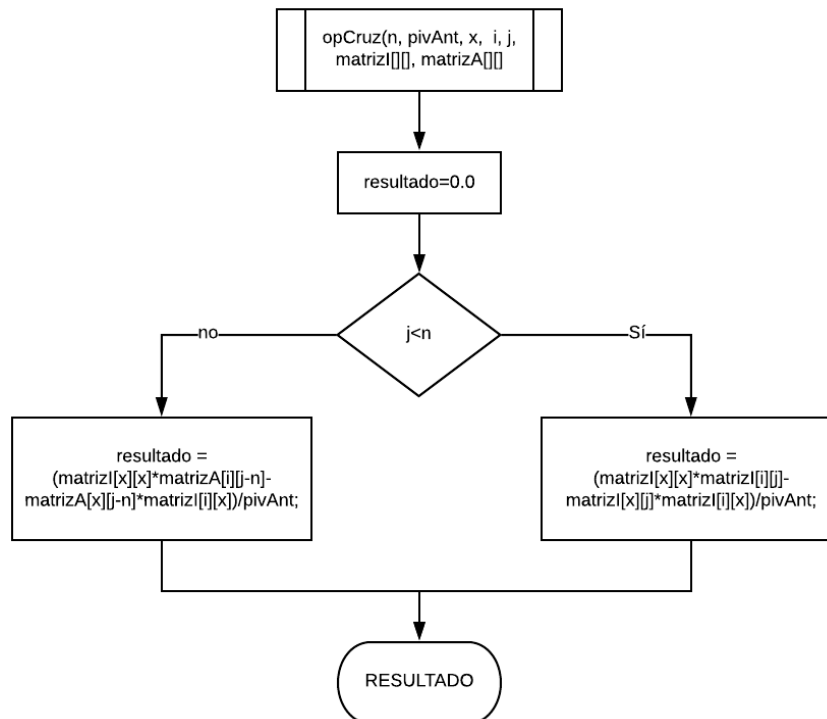
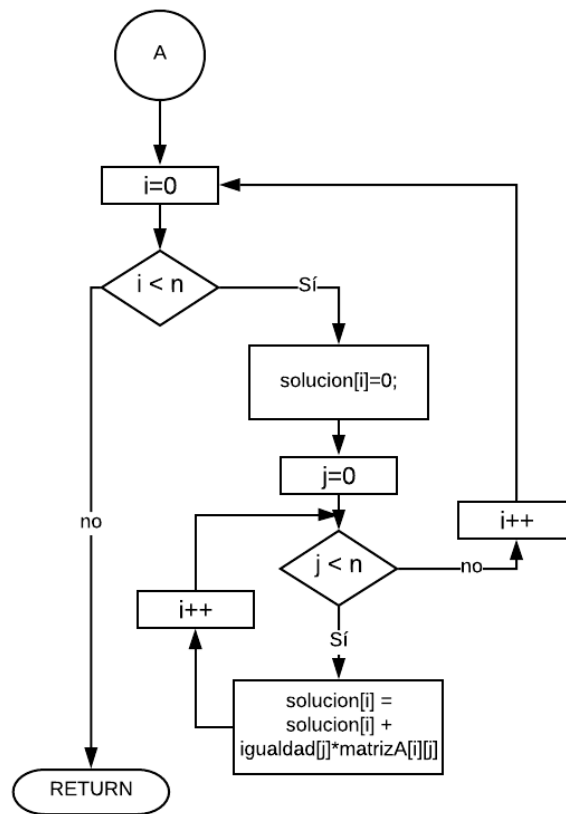
De esta manera se garantiza que el valor devuelto se cumplirá para todos los valores que correspondan para un sistema de ecuaciones de dimensión nxn.

Diagrama de flujo.

A continuación, se presenta el diagrama de flujo para el método montante.







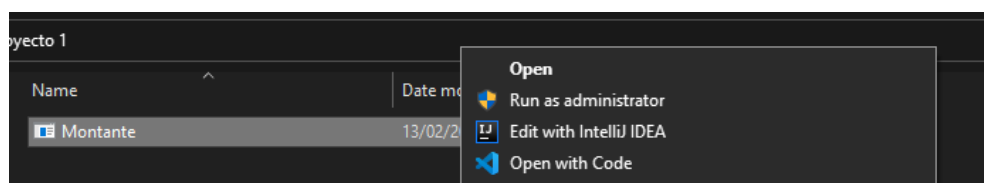
Manual de usuario y funcionamiento.

El programa Montante.exe tiene como finalidad resolver sistemas de ecuaciones lineales de n ecuaciones, a través del uso del Método de Montante. Además de mostrar la solución, el programa es capaz de ir desplegando el procedimiento, e indicar si un sistema tiene o no solución única. Análogo, se muestra también el valor del determinante de la matriz de coeficientes de la ecuación ingresada, su matriz adjunta, así como su inversa.

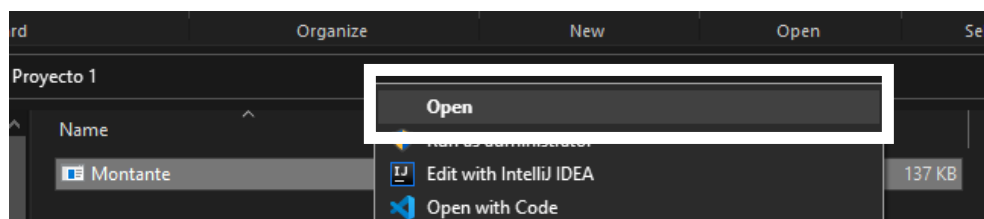
Abrir la aplicación.

Para poder abrir la aplicación se realizan los siguientes pasos.

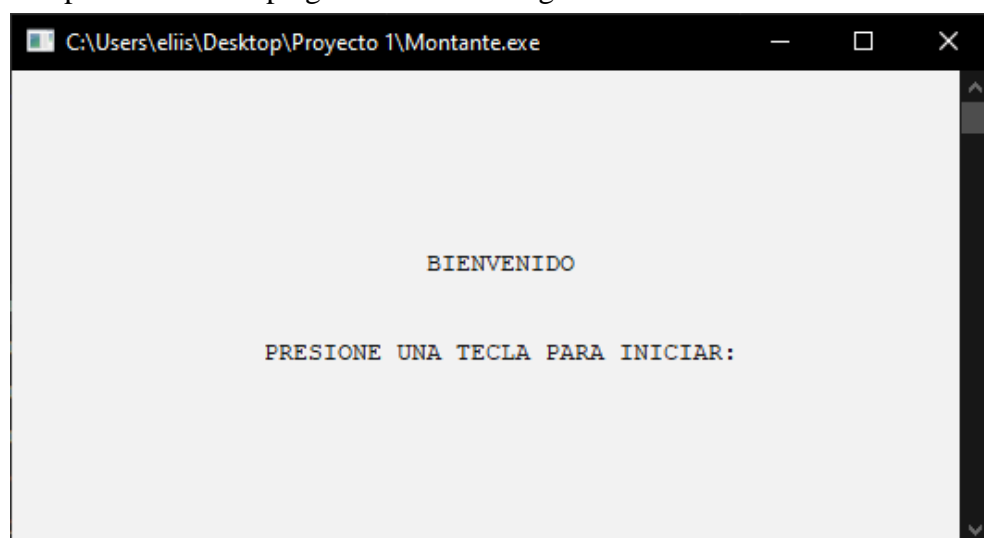
1. Clic izquierdo en nuestra aplicación Montante.exe:



2. Clic derecho en abrir u open (dependiendo del idioma del equipo):



3. La aplicación se desplegará como en la siguiente ventana:



Funcionamiento.

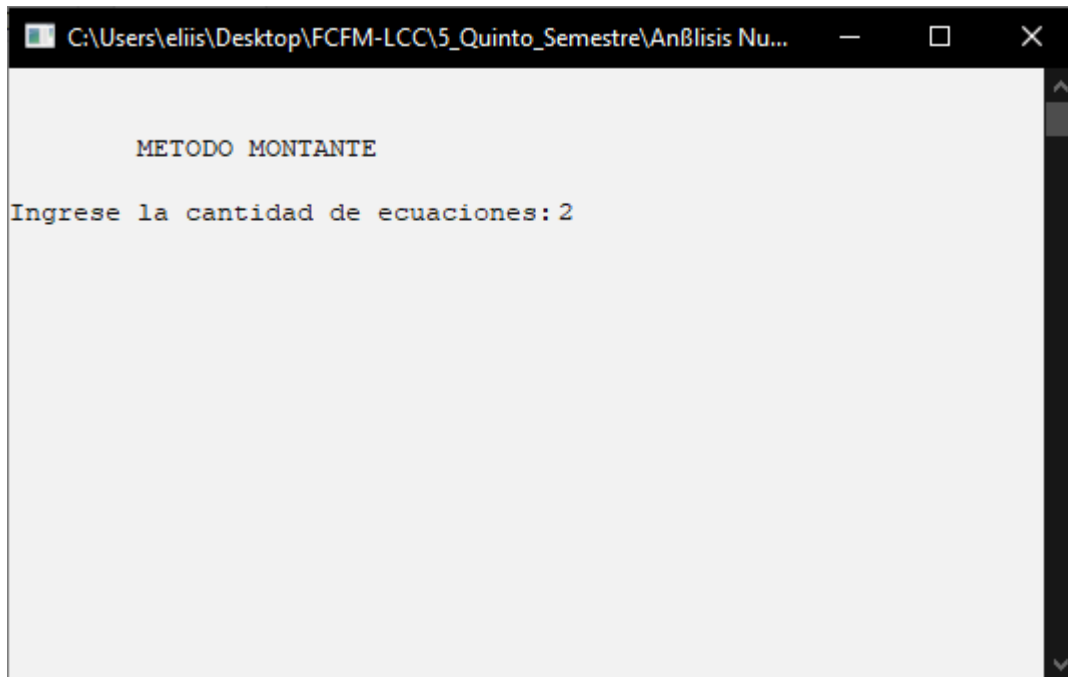
Después de abrir la aplicación, se desplegará una ventana de bienvenida. Para iniciarla debemos presionar cualquier tecla. Una vez hecho esto, nos mostrará un menú de inicio:



Debemos escoger una opción ingresando un número (1 o 2), y dar enter: Si escogemos la opción 2, nos desplegará la ventana de Fin del programa:

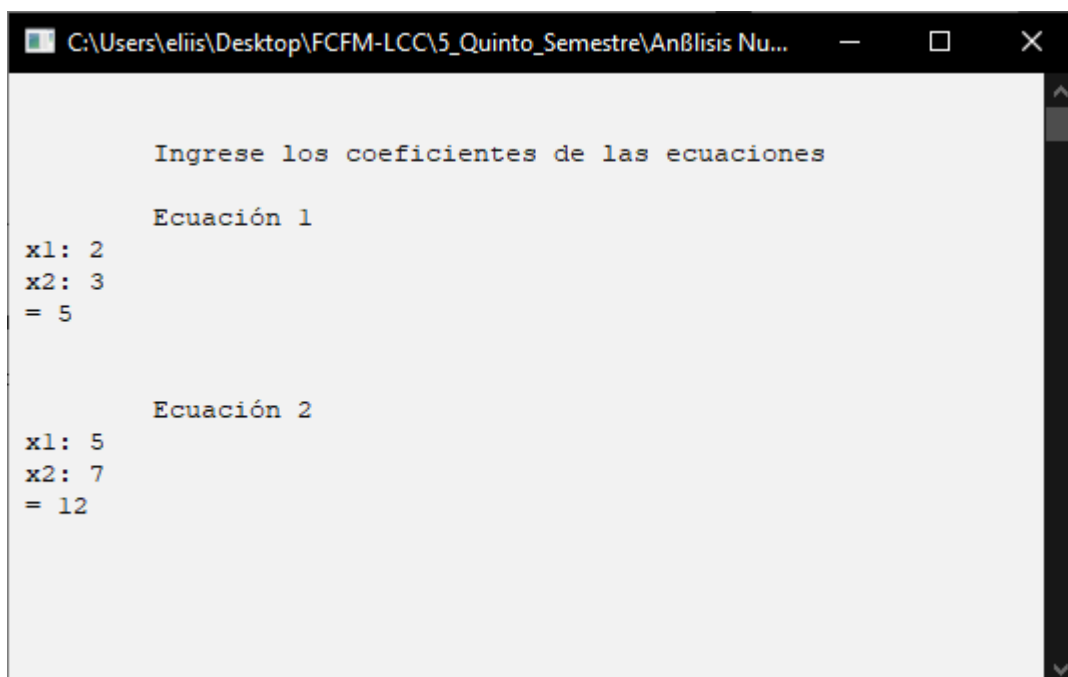


Si escogemos la opción 1, podremos ingresar la ecuación a resolver. Nos preguntará la cantidad de ecuaciones que queremos ingresar, este número debe ser mayor a 2:



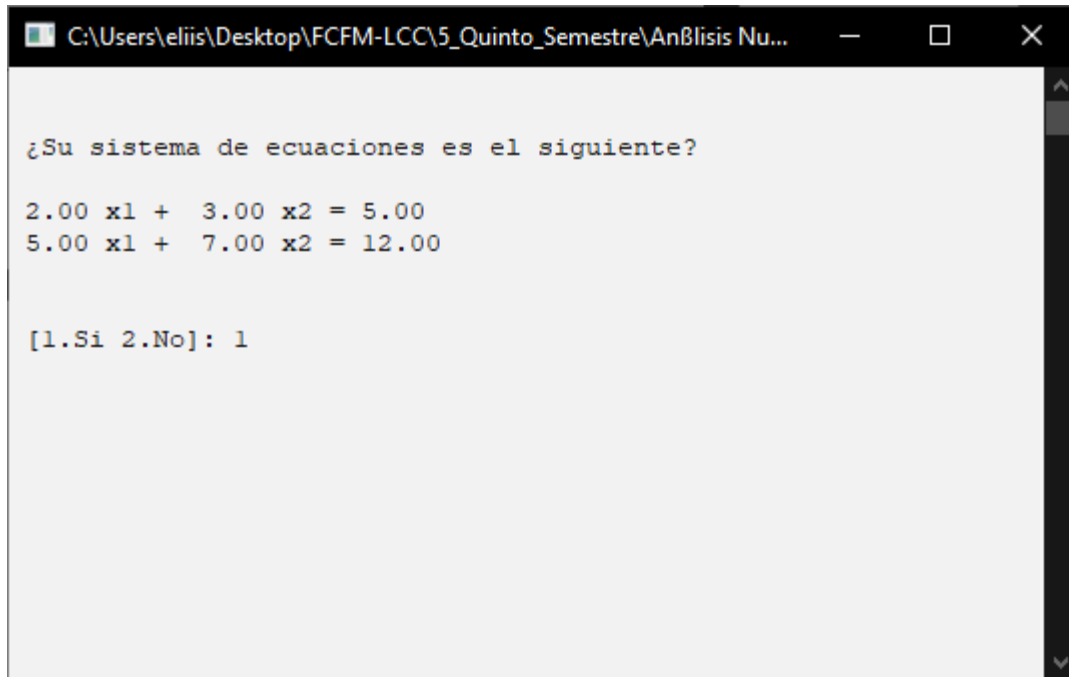
```
C:\Users\eliis\Desktop\FCFM-LCC\5_Quinto_Semestre\Análisis Nu...  
  
METODO MONTANTE  
Ingrese la cantidad de ecuaciones:2
```

Una vez escogido el tamaño, ingresaremos la ecuación coeficiente a coeficiente. Debemos ingresarlo y dar enter uno a uno. Para la siguiente ecuación $\begin{cases} 2x_1 + 3x_2 = 5 \\ 5x_1 + 7x_2 = 12 \end{cases}$ esta sería la entrada correspondiente:



```
C:\Users\eliis\Desktop\FCFM-LCC\5_Quinto_Semestre\Análisis Nu...  
  
Ingrese los coeficientes de las ecuaciones  
  
Ecuación 1  
x1: 2  
x2: 3  
= 5  
  
Ecuación 2  
x1: 5  
x2: 7  
= 12
```

Nos desplegará una venta de confirmación donde deberemos seleccionar una opción (1 o 2). En caso de seleccionar no, nos volverá a pedir la dimensión de la ecuación y la ecuación:



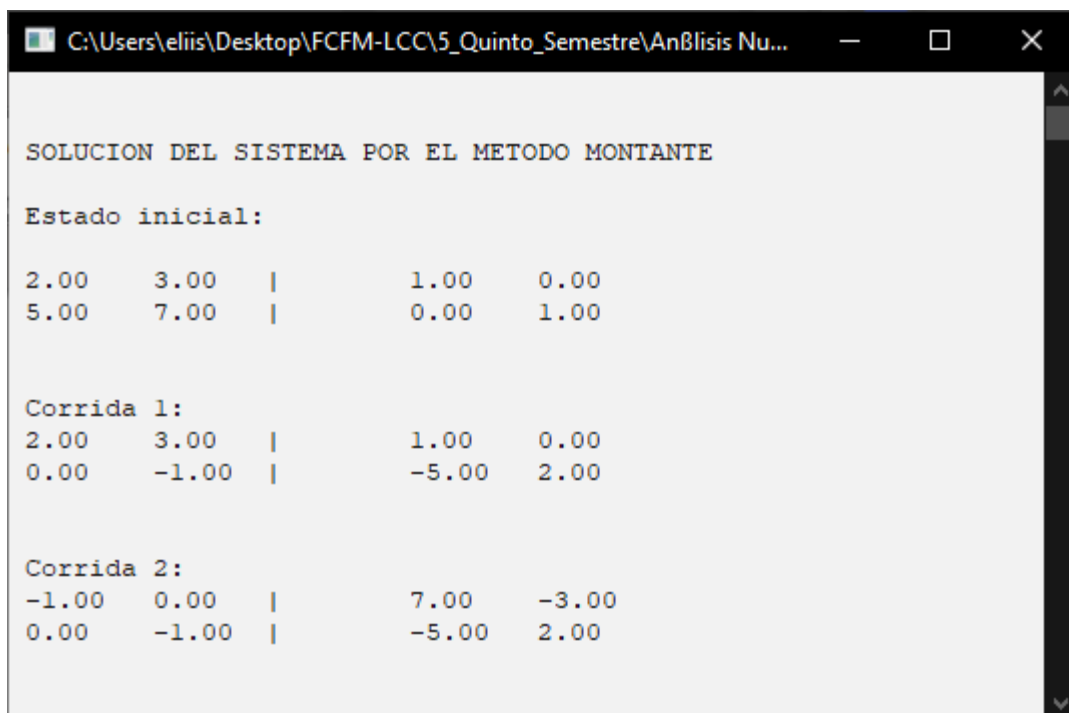
```
C:\Users\eliis\Desktop\FCFM-LCC\5_Quinto_Semestre\Análisis Nu...

¿Su sistema de ecuaciones es el siguiente?

2.00 x1 + 3.00 x2 = 5.00
5.00 x1 + 7.00 x2 = 12.00

[1.Si 2.No]: 1
```

Si seleccionamos que sí, nos mostrará el resultado y procedimiento para la resolución:



```
C:\Users\eliis\Desktop\FCFM-LCC\5_Quinto_Semestre\Análisis Nu...

SOLUCION DEL SISTEMA POR EL METODO MONTANTE

Estado inicial:

2.00    3.00    |    1.00    0.00
5.00    7.00    |    0.00    1.00

Corrida 1:
2.00    3.00    |    1.00    0.00
0.00   -1.00    |   -5.00    2.00

Corrida 2:
-1.00    0.00    |    7.00   -3.00
0.00   -1.00    |   -5.00    2.00
```

```
C:\Users\eliis\Desktop\FCFM-LCC\5_Quinto_Semestre\Análisis Nu...

##### RESULTADOS OBTENIDOS #####

det= -1.00

Matriz traspuesta del sistema:
7.00    -3.00
-5.00    2.00

Matriz Inversa del sistema:
-7.00    3.00
5.00    -2.00

**SOLUCION DEL SISTEMA**:
x1 = 1.000000
x2 = 1.000000

PRESIONE UNA TECLA PARA CONTINUAR: _
```

Una vez vista la respuesta, si damos clic a cualquier tecla, nos desplegará el menú de inicio nuevamente:

```
C:\Users\eliis\Desktop\Proyecto 1\Montante.exe

MENU:

1. Método montante.
2. Salir.

Seleccione una opción:
```

En caso de haber ingresado una ecuación sin solución única, el programa nos indicará, esto es que el determinante sea igual a cero. Por ejemplo, para la ecuación $\begin{cases} -x_1 + x_2 = 3 \\ x_1 - x_2 = 6 \end{cases}$ obtenemos:

```
C:\Users\eliis\Desktop\FCFM-LCC\5_Quinto_Semestre\Análisis Num0...

Ingresa los coeficientes de las ecuaciones

Ecuación 1
x1: -1
x2: 1
= 3

Ecuación 2
x1: 1
x2: -1
= 6
```

```
C:\Users\eliis\Desktop\FCFM-LCC\5_Quinto_Semestre\Análisis Num0...

-1.00  1.00 |  1.00  0.00
 1.00 -1.00 |  0.00  1.00

Corrida 1:
-1.00  1.00 |  1.00  0.00
 0.00  0.00 | -1.00 -1.00

Corrida 2:
 0.00  0.00 | -1.00 -1.00
 0.00  0.00 | -1.00 -1.00

El sistema NO TIENE SOLUCIÓN
det = 0

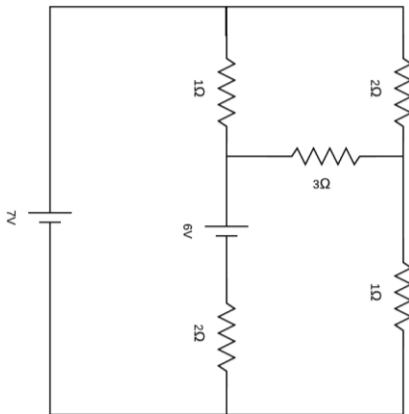
PRESIONE UNA TECLA PARA CONTINUAR:
```

Uso del programa (Análisis de Mallas).

Con el método de la corriente de malla se pueden resolver circuitos al escribir la ley de voltaje de Kirchhoff para corrientes que fluyen en los lazos de un circuito.

El método de análisis de mallas es muy utilizado para resolver circuitos resistivos (circuitos con sólo resistencias) lineales.

Ejemplo:



Genera las ecuaciones:

$$\begin{cases} 3I_1 - I_2 - 2I_3 = 1 \\ -I_1 + 6I_2 - 3I_3 = 0 \\ -2I_1 - 3I_2 + 6I_3 = 6 \end{cases}$$

Resolviendo en nuestro programa tenemos:

```
C:\Users\eliis\Desktop\FCFM-LCC\5_...
**SOLUCION DEL SISTEMA**:
x1 = 3.000000
x2 = 2.000000
x3 = 3.000000
```

$$I_1 = 3A$$

$$I_2 = 2A$$

$$I_3 = 3A$$

Conclusión

A través de la codificación de programa para la resolución de un sistema de ecuaciones, pudimos alcanzar una mejor comprensión del método de Montante y ver las grandes ventajas que este tiene por manejar números enteros.

Los procesos realizados en el programa son eficientes y se logra el objetivo de resolver las ecuaciones ingresadas, además de representar el procedimiento paso a paso para que el usuario pueda comparar el método realizado.

Referencias.

Métodos Numéricos. Método Montante. Recuperado de:
<https://metnum2016.wordpress.com/2016/10/11/228/>