

Instituto Tecnológico de Costa Rica  
Departamento de Matemática

Manual de Usuario  
Tarea Programada

José Navarro Acuña, 2016254241  
Josué Suárez Campos, 2016089518  
Martín Vargas Campos, 2015017184

Álgebra Lineal para Computación  
David Lowell Lovelady  
Grupo 3

II Semestre, 2017

## Índice

Asistente para realizar Eliminación Gaussiana. ....	4
Algoritmo Bareiss (Eliminación Gaussiana libre de fracciones).....	26
Aplicaciones de las transformaciones para la presentación grafica de vectores en 3D.....	33

## Introducción

El siguiente documento tiene como fin, ser una guía para el correcto uso de los tres programas implementados en dicho proyecto, mostrando la forma correcta de insertar los datos, la secuencia específica que recorren los diversos algoritmos para dar con la solución para el problema propuesto, además de la forma de salida de los datos con la solución que los programas muestran respectivamente. Los tres programas implementados corresponden a: Asistente para realizar Eliminación Gaussiana, Algoritmo de Bareiss y, por último, Aplicaciones de las transformaciones para la presentación grafica de vectores en 3D.

De forma general, para la elaboración de dichos programas se trataron los temas de algoritmos de algebra matricial, además de comprender los conceptos involucrados en el algoritmo de eliminación gaussiana, el algoritmo de bareiss, y la aplicación de los conceptos de transformaciones para la representación de vectores en 3D respectivamente.

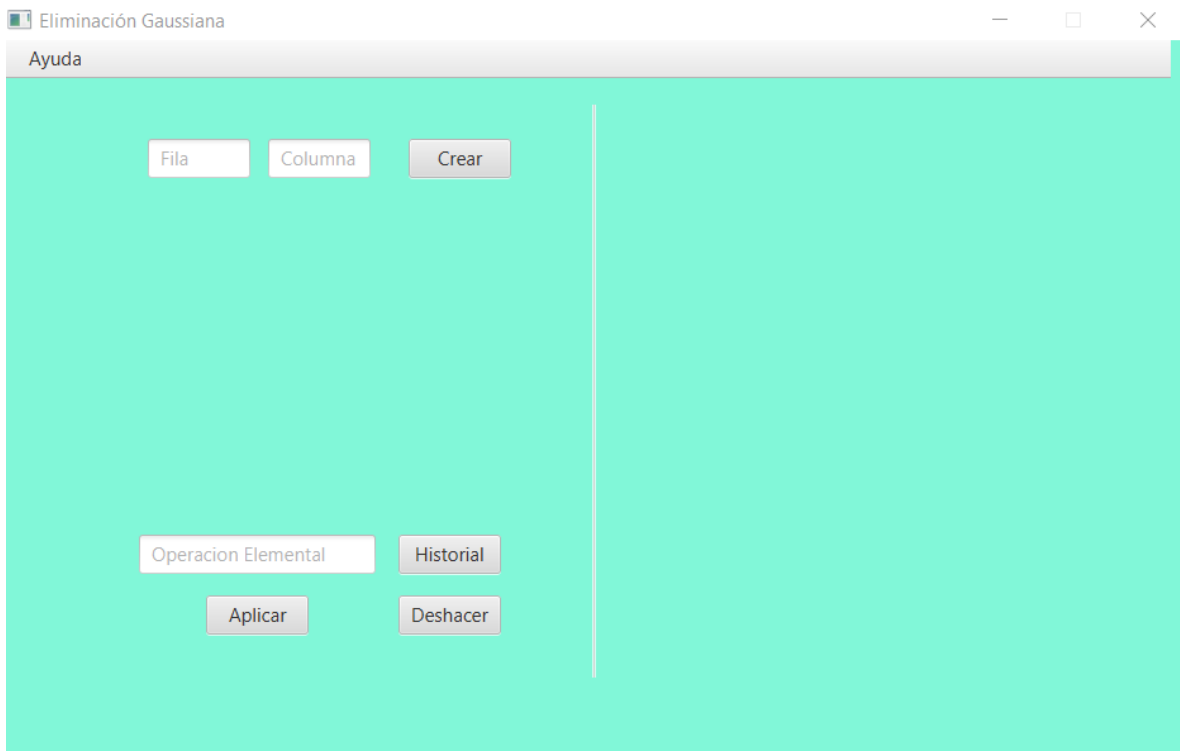
El entorno y herramientas utilizadas para la realización del proyecto corresponden a, el lenguaje de programación Java, cuyos IDE's corresponden a NetBeans y Eclipse junto con sus herramientas JavaFx. En dichos entornos, se desarrolló todo el tema lógico que conlleva la implementación de los algoritmos para la resolución de los problemas, de forma análoga, se efectuó el apartado visual de los programas, es decir, las interfaces.

## I. Asistente para realizar Eliminación Gaussiana.

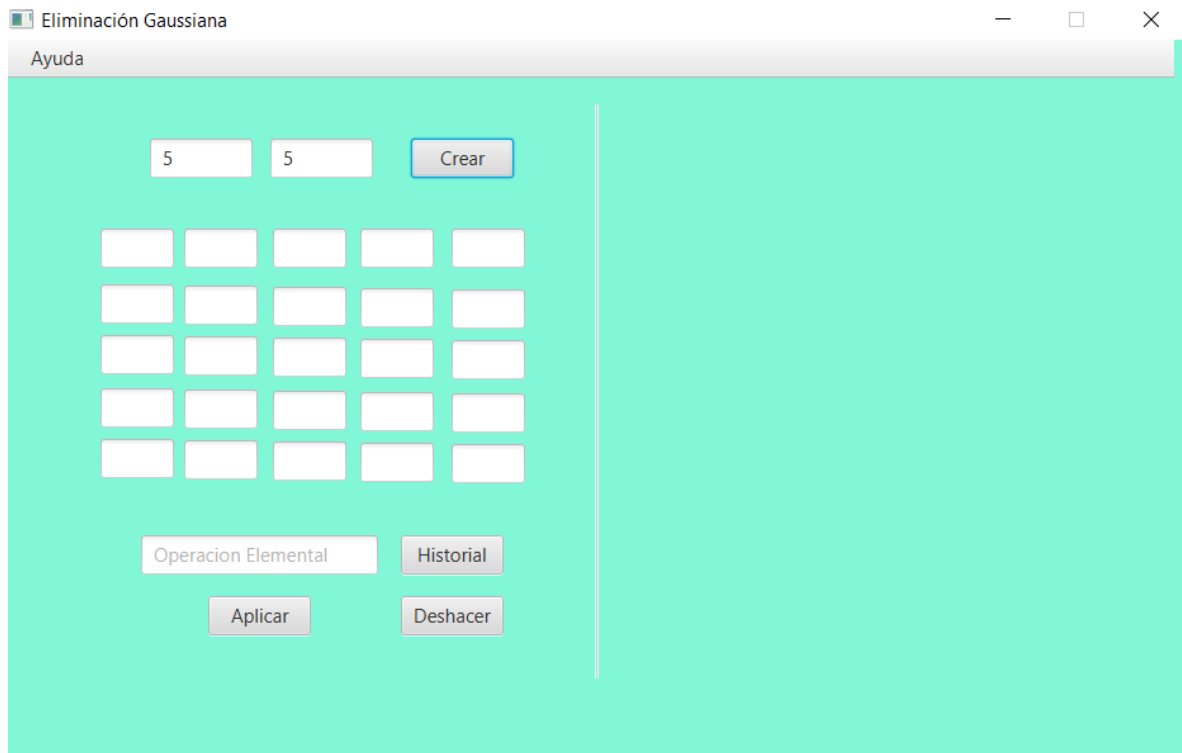
El siguiente apartado muestra el diseño y funcionamiento de un programa asistente para la ejecución de operaciones elementales en una matriz con dimensiones dadas por el usuario.

- **Forma correcta de ejecutar el programa e inserción de los datos.**

Para la creación de la matriz, inserte la cantidad de filas y columnas, la cantidad de estas deben ser menor o igual a 5 (Figura 1). Una vez puesta la cantidad de elementos que desea, presione el botón Crear (Figura 2).



*Figura 1. Inserción de filas y columnas.*



*Figura 2. Creación de la matriz*

Seguidamente inserte los valores numéricos en la matriz. Como restricción, el programa solo acepta números enteros y fraccionarios (Figura 3).

Eliminación Gaussiana

Ayuda

5 5 Crear

2/3	24	15	11/3	7
12	3/10	11	22	23
5/6	32	11/2	6	8
9	12	15	17	5/3
18	15/2	19	21	4/3

Operacion Elemental Historial

Aplicar Deshacer

*Figura 3. Inserción de datos en la matriz.*

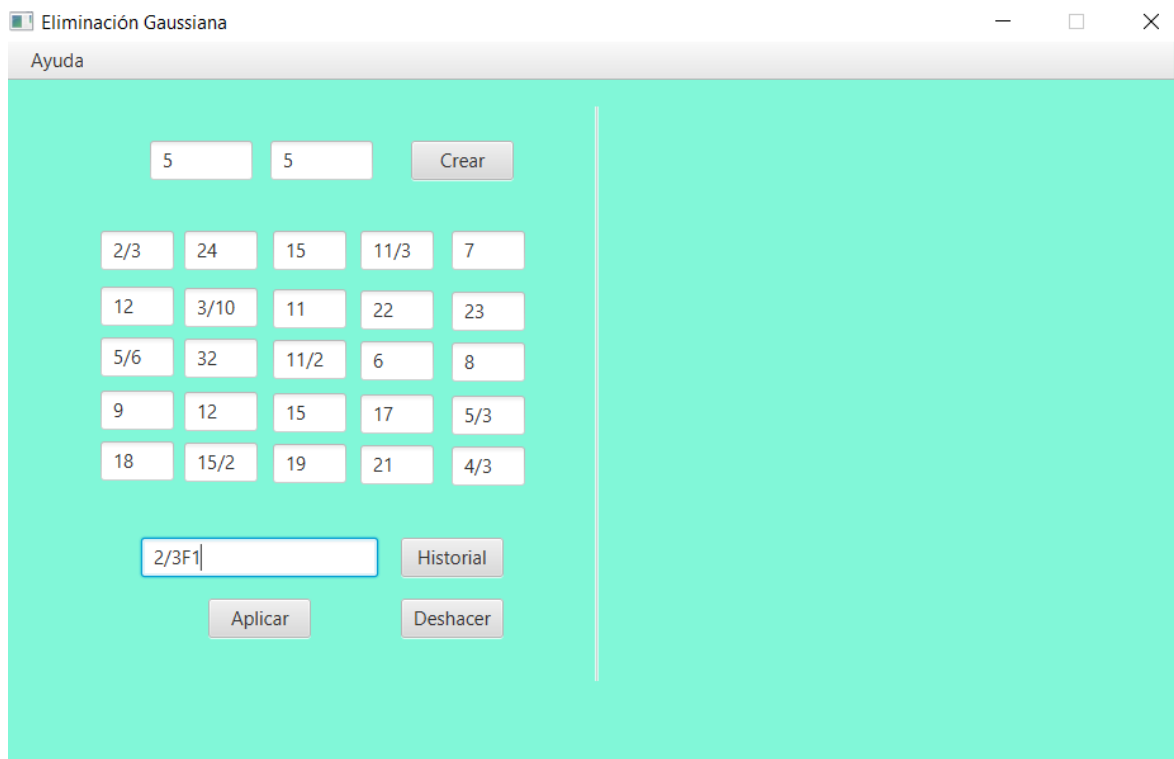
A continuación, inserte la operación elemental que desee realizar y presione el botón Aplicar para realizarla.

El formato de escritura de las operaciones es la siguiente:

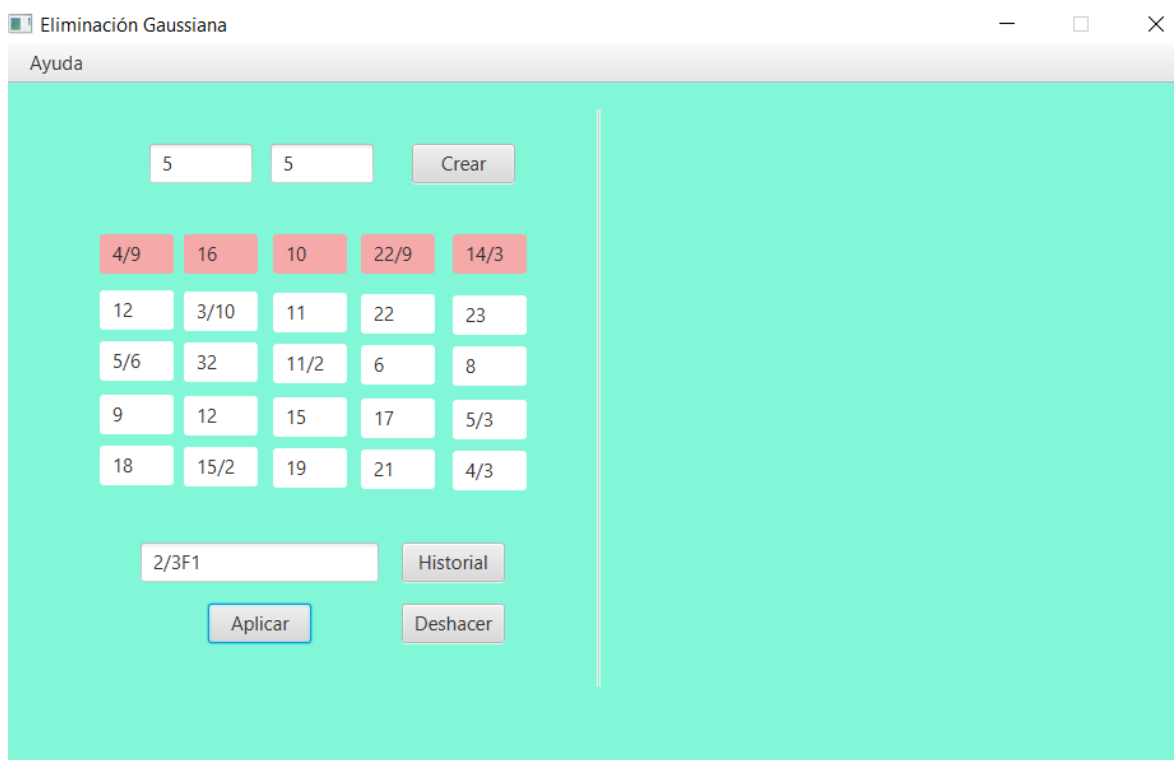
- Multiplicación:  $X * F_n$  ó  $X F_n$
- Suma:  $X F_n + Y F_m$
- Cambio de filas:  $F_n; F_m$  ó  $F_n, F_m$

***F<sub>n</sub>***: F representa la fila, puede estar mayúscula o minúscula, y la n representa el número de fila, iniciando por 1.

***X e Y***: representan el número utilizado para realizar la multiplicación.



*Figura 4. Inserción de la operación elemental.*



*Figura 5. Resultado de la operación elemental.*

Aplicada la operación elemental, si lo desea puede deshacerla presionando el botón Deshacer y en seguida eliminará la última operación elemental aplicada.



*Figura 6. Botón deshacer*



Si desea ver el historial de cambios de la matriz presione el botón Historial y mediante los botones Siguiente y Anterior podrá observar los cambios realizados.



*Figura 7. Historial*

## - Secuencia de ejecución de las funciones.

Al crear la matriz la primera función llamada es **crearMatriz**, la cual hace llamada a la función **cambiarVisibilidad** que cambia la visibilidad de todos los campos de texto de la matriz a **false**, esto para que en cada creación de matriz se limpie la matriz anterior. Volviendo a crearMatriz esta verifica mediante un If si el número de columnas y filas insertadas es válido, en caso contrario mandar los respectivos mensajes de alerta.

```
public void crearMatriz(ActionEvent event){
    cambiarVisibilidad(false);
    try{
        if ( (Integer.parseInt(numeroColumnas.getText()) <= 5 ) & (Integer.parseInt((numeroFilas.getText())) <= 5)){
            columnas= Integer.parseInt(numeroColumnas.getText());
            filas = Integer.parseInt((numeroFilas.getText()));
        }
        else{
            MostrarAlerta("No se puede crear la matriz", "Favor añada una cantidad de filas y columnas menor o igual a 5", AlertType.ERROR);
        }
    }
    catch(Exception e){
        MostrarAlerta("No se puede crear la matriz", "Favor añada la cantidad de filas y columnas para crear la matriz", AlertType.ERROR);
    }
}
```

```
public void cambiarVisibilidad(boolean x){
    campoTexto1.setVisible(x);
    campoTexto2.setVisible(x);
    campoTexto3.setVisible(x);
    campoTexto4.setVisible(x);
    campoTexto5.setVisible(x);

    campoTexto6.setVisible(x);
    campoTexto7.setVisible(x);
    campoTexto8.setVisible(x);
    campoTexto9.setVisible(x);
    campoTexto10.setVisible(x);

    campoTexto11.setVisible(x);
    campoTexto12.setVisible(x);
    campoTexto13.setVisible(x);
    campoTexto14.setVisible(x);
    campoTexto15.setVisible(x);

    campoTexto16.setVisible(x);
    campoTexto17.setVisible(x);
    campoTexto18.setVisible(x);
    campoTexto19.setVisible(x);
    campoTexto20.setVisible(x);

    campoTexto21.setVisible(x);
    campoTexto22.setVisible(x);
    campoTexto23.setVisible(x);
    campoTexto24.setVisible(x);
    campoTexto25.setVisible(x);
}
```

Continuando con la función crearMatriz esta mediante dos ciclos For recorriendo la cantidad de filas y columnas, va activando los campos de texto necesarios.

```
for (int contadorFilas = 1; contadorFilas <= filas; contadorFilas ++ ){

    for (int contadorColumnas = 1; contadorColumnas <= columnas; contadorColumnas ++ ) {

        switch(contadorFilas){

            case 1:
                switch(contadorColumnas){
                    case 1:
                        campoTexto1.setVisible(true);
                        break;
                    case 2:
                        campoTexto2.setVisible(true);
                        break;

                    case 3:
                        campoTexto3.setVisible(true);
                        break;

                    case 4:
                        campoTexto4.setVisible(true);
                        break;

                    case 5:
                        campoTexto5.setVisible(true);
                        break;

                    default:
                        //Print
                        break;

                case 2:
                    switch(contadorColumnas){
                        case 1:
                            campoTexto6.setVisible(true);
                            break;
                        case 2:
                            campoTexto7.setVisible(true);
                            break;

                        case 3:
                            campoTexto8.setVisible(true);
                            break;

                        case 4:
                            campoTexto9.setVisible(true);
                            break;

                        case 5:
                            campoTexto10.setVisible(true);
                            break;

                        default:
                            //Print
                            break;

                    }
                }
                break;
            }
        }
    }
}
```

```

case 3:
    switch(contadorColumnas){
        case 1:
            campoTexto11.setVisible(true);
            break;
        case 2:
            campoTexto12.setVisible(true);
            break;

        case 3:
            campoTexto13.setVisible(true);
            break;

        case 4:
            campoTexto14.setVisible(true);
            break;

        case 5:
            campoTexto15.setVisible(true);
            break;

        default:
            //Print
            break;
    }
    break;
case 4:
    switch(contadorColumnas){
        case 1:
            campoTexto16.setVisible(true);
            break;
        case 2:
            campoTexto17.setVisible(true);
            break;

        case 3:
            campoTexto18.setVisible(true);
            break;

        case 4:
            campoTexto19.setVisible(true);
            break;

        case 5:
            campoTexto20.setVisible(true);
            break;

        default:
            //Print
            break;
    }
    break;

```

```

case 5:
    switch(contadorColumnas){
        case 1:
            campoTexto21.setVisible(true);
            break;
        case 2:
            campoTexto22.setVisible(true);
            break;

        case 3:
            campoTexto23.setVisible(true);
            break;

        case 4:
            campoTexto24.setVisible(true);
            break;

        case 5:
            campoTexto25.setVisible(true);
            break;

        default:
            //Print
            break;
    }
    break;
default:
    break;
}

```

Una vez creada la matriz y el usuario habiendo insertado la operación elemental, con el botón Aplicar se llama la función **recabarDatos**, la cual se encarga de extraer los números de los campos de texto de la matriz y la operación elemental. Además de esto quita los espacios en blanco de los campos de texto y verifica que sea legítimos y llama a la función **parsearOpElemental**. En caso contrario muestra un mensaje de error.

```
public void recabarDatos(){
    Boolean seguir = true;
    Boolean alerta = false;
    String operacionElemental = campoOE.getText();

    matrizNumeros[0][0] = campoTexto1.getText();
    matrizNumeros[0][1] = campoTexto2.getText();
    matrizNumeros[0][2] = campoTexto3.getText();
    matrizNumeros[0][3] = campoTexto4.getText();
    matrizNumeros[0][4] = campoTexto5.getText();

    matrizNumeros[1][0] = campoTexto6.getText();
    matrizNumeros[1][1] = campoTexto7.getText();
    matrizNumeros[1][2] = campoTexto8.getText();
    matrizNumeros[1][3] = campoTexto9.getText();
    matrizNumeros[1][4] = campoTexto10.getText();

    matrizNumeros[2][0] = campoTexto11.getText();
    matrizNumeros[2][1] = campoTexto12.getText();
    matrizNumeros[2][2] = campoTexto13.getText();
    matrizNumeros[2][3] = campoTexto14.getText();
    matrizNumeros[2][4] = campoTexto15.getText();

    matrizNumeros[3][0] = campoTexto16.getText();
    matrizNumeros[3][1] = campoTexto17.getText();
    matrizNumeros[3][2] = campoTexto18.getText();
    matrizNumeros[3][3] = campoTexto19.getText();
    matrizNumeros[3][4] = campoTexto20.getText();

    matrizNumeros[4][0] = campoTexto21.getText();
    matrizNumeros[4][1] = campoTexto22.getText();
    matrizNumeros[4][2] = campoTexto23.getText();
    matrizNumeros[4][3] = campoTexto24.getText();
    matrizNumeros[4][4] = campoTexto25.getText();
}
```

```

for (int contadorFilas = 0; contadorFilas < filas; contadorFilas ++ ){
    for (int contadorColumnas = 0; contadorColumnas < columnas; contadorColumnas ++ ) {

        matrizNumeros[contadorFilas][contadorColumnas] = matrizNumeros[contadorFilas][contadorColumnas].replaceAll(" ", "");

        if (matrizNumeros[contadorFilas][contadorColumnas].length()==0){
            alerta = true;
            seguir = false;
        }
    }
}

if ((seguir == true) & (alerta == false)){
    try{
        parsearOpElemental(operacionElemental);
    }catch(Exception e){
        MostrarAlerta("Error con la operación elemental", "Parece que has ingresado una operación elemental o números de casilla inválidos", Alert
    }
}
else{
    MostrarAlerta("Completa todos los campos antes de continuar", "Parece que has olvidado ingresar un dígito en alguna casilla en blanco!", Alert
}
}

```

La función **parsearOpElemental** se encarga de recorrer la operación elemental indica por el usuario en busca de que procedimiento seguir. Inicialmente podemos observar el cómo se adapta la operación elemental a un formato que se pueda procesar, además se inicializan las variables que serán utilizadas. Seguidamente mediante un ciclo While se itera en la operación elemental, carácter por carácter.

```
public void parsearOpElemental(String operacionElemental){
    operacionElemental = operacionElemental.replaceAll("\\*", "");
    operacionElemental = operacionElemental.replaceAll(" ", "");
    operacionElemental = operacionElemental.replaceAll(";", ",");
    operacionElemental = operacionElemental.toLowerCase();

    char [] opElemArray = operacionElemental.toCharArray();

    int contador = 0;
    int factor;
    String factorString;
    String numerador;
    String denominador;

    DespintarFila();

    while (contador < opElemArray.length){
        factor = 0;
        factorString = "";
        StringBuilder sb = new StringBuilder();
        StringBuilder numeradorSB= new StringBuilder();
        StringBuilder denominadorSB= new StringBuilder();
        numerador = "";
        denominador = "";
```



Mediante la siguiente comparativa If verifica si el carácter que está siendo iterado es un número, esto debido a que el formato de la multiplicación siempre inicia con el número por delante. Si se cumple la condición, mediante un ciclo While se va guardando el número en las variables.

```
if (Character.isDigit(opElemArray[contador])){

    while (opElemArray[contador] != 'f'){
        try{
            if (opElemArray[contador] == '/'){
                numeradorSB = sb;
                contador+=1;
                while (opElemArray[contador] != 'f'){
                    denominadorSB.append(opElemArray[contador]);
                    contador+= 1;
                }
                break;
            }
        }

        sb.append(opElemArray[contador]);
        contador += 1;
    }
}
```

A continuación, se utiliza otro If para verificar si los dos siguientes caracteres son una F y un número, esto representando la fila a operar. Si se cumple la condicional llama a función **AgregarHistorial** la cual se encarga de añadir la matriz al historial antes de realizar nuevas operaciones con esta.

```

if (opElemArray[contador] == 'f' && Character.isDigit(opElemArray[contador+1]) == true){
    contador+=1;
    int filaMult = Character.getNumericValue(opElemArray[contador]);
    AgregarHistorial();
    multiplicarFila(opElemArray, factor, filaMult, numerador, denominador);
    PintarFila(filaMult);
    contador += 1;
    try
    {
        if (opElemArray[contador] == '+'){
            factorString = "";
            sb = new StringBuilder();
            numeradorSB = new StringBuilder();
            denominadorSB = new StringBuilder();
            contador+= 1;
            if (Character.isDigit(opElemArray[contador]) == true){
                while (opElemArray[contador] != 'f'){
                    try{
                        if (opElemArray[contador] == '/'){
                            numeradorSB = sb;
                            contador+=1;
                            while (opElemArray[contador] != 'f'){
                                denominadorSB.append(opElemArray[contador]);
                                contador+= 1;
                            }
                        }
                        break;
                    }
                }
                sb.append(opElemArray[contador]);
                contador += 1;
            }
        }
    }
}

```

```

public void AgregarHistorial(){
    StringBuilder strMatriz = new StringBuilder();
    String x = campoOE.getText();

    x = x.toUpperCase();

    for( int i=0; i<columnas; i++){
        for( int j=0; j<filas; j++){

            strMatriz.append(matrizNumeros[i][j]);
            strMatriz.append(",");
        }

        strMatriz.append(".");
    }

    String matriz = strMatriz.toString();

    for( int i=0; i<vectorHistorial.length; i++){

        if (vectorHistorial[i]==null){
            vectorHistorial[i] = matriz;
            System.out.print("\nAgregando la matriz "+i+": "+vectorHistorial[i]+"\n");
            vectorOE[i] = x;
            System.out.print("\nAgregando la OE: "+vectorOE[i]+"\n");
            ultimoHistorial = i;
            break;
        }
    }
}

```

Luego llama a la función **multiplicarFila**, la cual, como su nombre lo indica, se encarga de multiplicar los números de la fila dada. Cada elemento de la fila es iterado por medio de un ciclo For. Mediante el If que se encuentra dentro del For valida si la variable numerador se encuentra vacía, esto debido a que si se encuentra vacío significaría que no es un número fraccionario. En ambos casos se crea el objeto **Rational**, si es fraccionario se envía el numerador y denominador, en caso contrario se envía la variable factor y un denominador de 1. La clase **Rational** es importante en el algoritmo ya que permite resolver operaciones matemáticas conservando el formato fraccional.

```

public void multiplicarFila(char [] opElemenArray, int factor, int filaM, String numerador, String denominador){

    Rational resultadoRational;
    filaM = filaM - 1;
    if (filaM >= 0){
        System.out.print("\nMultiplicarFila\n");
        Rational FactorRacional;
        for (int contadorColumnas = 0; contadorColumnas < columnas; contadorColumnas ++ ) {
            if (!numerador.equals("")){
                FactorRacional = new Rational(Integer.parseInt(numerador), Integer.parseInt(denominador));
            }
            else{
                FactorRacional = new Rational(factor, 1);
            }
        }
    }
}

```

Seguidamente se itera el número de la matriz y crea su respectivo objeto **Rational**. Por último estos son multiplicados y se inserta el valor en la matriz.

```

String num = matrizNumeros[filaM][contadorColumnas];
int tamaño = num.length();
int contador = 0;
StringBuilder numeradorStr = new StringBuilder();
StringBuilder denominadorStr = new StringBuilder();
String numeradorM;
String denominadorM;

while (tamaño > contador){
    char aChar = num.charAt(contador);
    if (aChar == '/'){
        contador+=1;
        while (tamaño > contador){
            aChar = num.charAt(contador);

            denominadorStr.append(aChar);
            contador+=1;
        }
        break;
    }
    numeradorStr.append(aChar);
    contador+=1;
}

numeradorM = numeradorStr.toString();

```

```

if (denominadorStr.toString().equals("")){
    denominadorM = "1";
}
else{
    denominadorM = denominadorStr.toString();
}

Rational ElemMatriz = new Rational(Integer.parseInt(numeradorM), Integer.parseInt(denominadorM));
resultadoRacional = FactorRacional.times(ElemMatriz);
matrizNumeros[filam][contadorColumnas] = resultadoRacional.toString();
System.out.print("\nEl resultado de la multiplicación entre "+FactorRacional+" y "+ElemMatriz+" es de: "+resultadoRacional.toString()+"\n");

```

Devuelta con la función **parsearOpElem** se verifica si tiene un carácter extra en la operación elemental, si se cumple significaría que es una suma, por lo que se sigue iterando en los siguientes números y se llama a la función **sumarFilas**

```

if (opElemArray[contador] == '+'){

    factorString = "";
    sb = new StringBuilder();
    numeradorSB = new StringBuilder();
    denominadorSB = new StringBuilder();
    contador+= 1;
    if (Character.isDigit(opElemArray[contador]) == true){
        while (opElemArray[contador] != 'f'){
            try{
                if (opElemArray[contador] == '/'){
                    numeradorSB = sb;
                    contador+=1;
                    while (opElemArray[contador] != 'f'){
                        denominadorSB.append(opElemArray[contador]);
                        contador+= 1;
                    }
                    break;
                }
                sb.append(opElemArray[contador]);
                contador += 1;
            }
        }

        factorString = sb.toString();
        numerador= numeradorSB.toString();
        denominador = denominadorSB.toString();

        factor = Integer.parseInt(factorString);

        if (opElemArray[contador] == 'f' && Character.isDigit(opElemArray[contador+1]) == true){
            contador+=1;
            int filaSuma = Character.getNumericValue(opElemArray[contador]);
            AgregarHistorial();
            sumarFilas(opElemArray, factor, filaMult, filaSuma, numerador, denominador);
            PintarFila(filaSuma);
            contador+=1;
        }
    }
}

```

La función **sumarFilas** realiza la llamada a la función **multiplicarFilas**, esto para realizar la multiplicación del segundo sumando. Utilizando ciclos While itera sobre los números de la matriz, seguidamente se crean los objetos **Rational** y se realiza la suma respectiva.

```
public void sumarFilas(char [] opElemArray, int factor, int filaPrimera, int fila, String num)
{
    System.out.print("\nSuma de filas\n");
    multiplicarFila(opElemArray, factor, fila, numerador, denominador);
    filaPrimera = filaPrimera - 1;
    fila = fila - 1;
    Rational resultadoRational;

    if ((filaPrimera >= 0) & (fila >= 0)){
        for (int contadorColumnas = 0; contadorColumnas < columnas; contadorColumnas ++ ) {

            String num2= matrizNumeros[filaPrimera][contadorColumnas];
            String numFraccionario = matrizNumeros[fila][contadorColumnas];

            StringBuilder numeradorSuma = new StringBuilder();
            StringBuilder denominadorSuma = new StringBuilder();
            StringBuilder numeradorSuma2 = new StringBuilder();
            StringBuilder denominadorSuma2 = new StringBuilder();
            int tamaño = numFraccionario.length();
            int tamaño2 = num2.length();
            int contador = 0;
```

```

        while (tamaño > contador){
            char aChar = numFraccionario.charAt(contador);
            if (aChar == '/'){
                contador+=1;
                while (tamaño > contador){
                    aChar = num2.charAt(contador);

                    denominadorSuma.append(aChar);
                    contador+=1;
                }
                break;
            }
            numeradorSuma.append(aChar);
            contador+=1;
        }
        contador=0;

        while (tamaño2 > contador){
            char aChar = num2.charAt(contador);

            if (aChar == '/'){
                contador+=1;
                while (tamaño2 > contador){
                    aChar = num2.charAt(contador);
                    denominadorSuma2.append(aChar);
                    contador+=1;
                }
                break;
            }
        }

String numSum = numeradorSuma.toString();
String denSum;

if (denominadorSuma.toString().equals("")){
    denSum = "1";
}
else{
    denSum = denominadorSuma.toString();
}

String numSum2 = numeradorSuma2.toString();
String denSum2;

if (denominadorSuma2.toString().equals("")){
    denSum2 = "1";
}
else{
    denSum2 = denominadorSuma2.toString();
}

Rational ElemMatriz1 = new Rational(Integer.parseInt(numSum), Integer.parseInt(denSum));
Rational ElemMatriz2 = new Rational(Integer.parseInt(numSum2), Integer.parseInt(denSum2));

resultadoRational = ElemMatriz1.plus(ElemMatriz2);

```

Volviendo a la función **parsearOpElem**, recordemos que inicialmente había un **If** que verificaba si el primer carácter de la operación elemental es un número, en caso contrario, es decir, que este fuera un carácter se abre una nueva comparación para verificar si es un cambio de filas. Mediante 2 **If** se verifica el formato que debe seguir el cambio de filas y si se cumple si llama la función **IntercambiarFilas**.

```
if (opElemArray[contador] == 'f' && Character.isDigit(opElemArray[contador+1]) == true && opElemArray[contador+2] == ','){
    int fila1 = Character.getNumericValue(opElemArray[contador+1]);

    if (opElemArray[contador+3] == 'f' && Character.isDigit(opElemArray[contador+4]) == true){
        int fila2 = Character.getNumericValue(opElemArray[contador+4]);
        AgregarHistorial();
        intercambiarFilas(opElemArray, fila1, fila2);
        PintarFila(fila1);
        PintarFila(fila2);
        contador += 5;
    }
}
```

```
public void intercambiarFilas(char [] opElemArray, int fila1, int fila2){
    System.out.print("\nCambiar Filas\n");
    fila1 = fila1 - 1;
    fila2 = fila2 - 1;

    for (int contadorColumnas = 0; contadorColumnas < columnas; contadorColumnas ++ ) {

        String x = matrizNumeros[fila1][contadorColumnas];
        String y = matrizNumeros[fila2][contadorColumnas];

        String temp = x;
        x = y;
        y = temp;

        matrizNumeros[fila1][contadorColumnas] = x;
        matrizNumeros[fila2][contadorColumnas] = y;
        System.out.print("\nSe ha intercambio "+x+" con "+y+"\n");

    }
    //AgregarHistorial();
    ImprimirMatriz();
}
```



- **Forma de acceder a la salida de datos.**

Cada operación elemental realizada quedaría en la interfaz de la matriz, aquella donde primeramente se ingresaron los datos. Esta irá cambiando según las operaciones dadas. Además para percibir cual fue la fila modificada se marca con color rojo la fila a la cual se aplicó la operación.

## II. Algoritmo Bareiss (Eliminación Gaussiana libre de fracciones).

En el área de ingeniería y álgebra computacional, es muy frecuente hacer uso de sistemas de ecuaciones donde las entradas son valores enteros. Existen actualmente una serie de métodos en cálculo con matrices cuyo objetivo es el cálculo simbólico y el cálculo con aritmética exacta, es decir, con divisiones exactas donde el residuo es nulo, por ejemplo, el método de condensación de eliminación Gaussiana “libre de fracciones” y el algoritmo de Bareiss, donde éste último efectúa el cálculo simbólico de determinantes.

### - Forma correcta de ejecutar el programa e inserción de los datos.

De primera instancia, el programa tiene como principal componente una barra menú como se muestra en la figura 1, donde el ítem *Archivo* despliega dos subítems los cuales son *Genera Nueva Matriz* y *Resolver Sistema*.

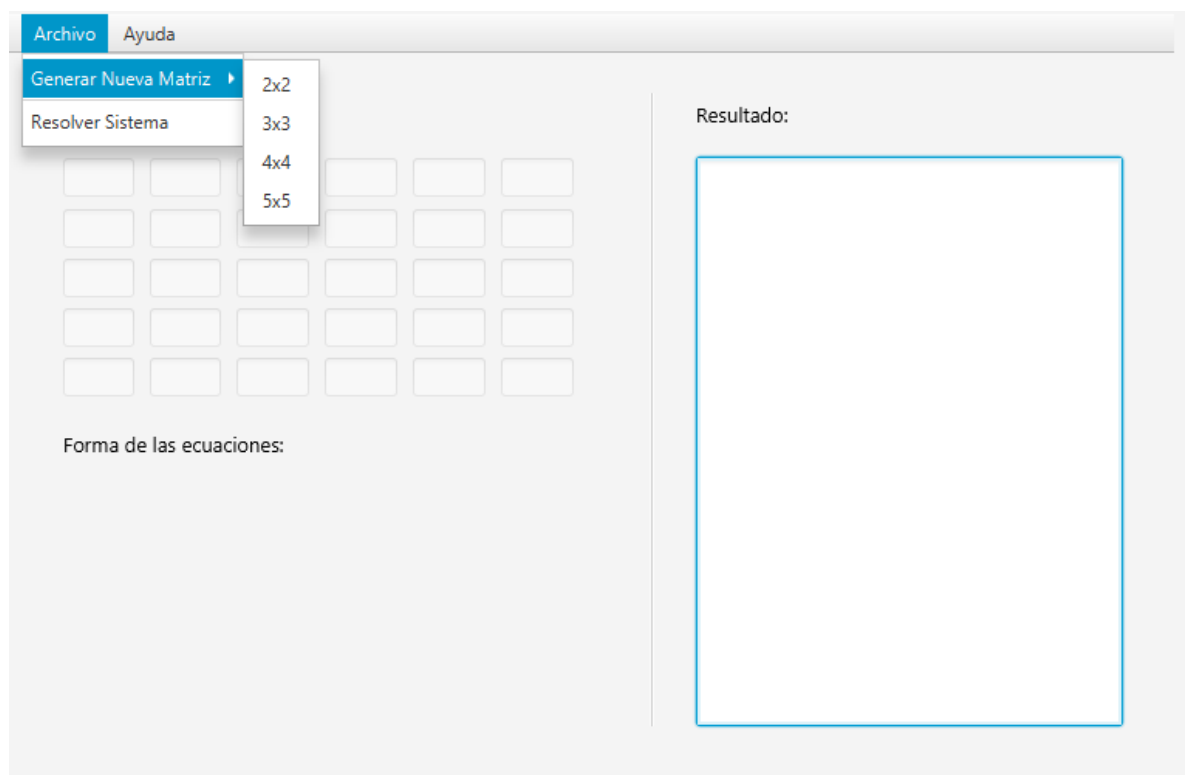


Figura 1. Barra menú.

Como se puede observar, el subitem *Genera Nueva Matriz* despliega cuatro tipos de tamaño de la matriz a escoger (2x2, 3x3, 4x4, 5x5), una vez seleccionado el tipo de tamaño deseado, se habilitarán los campos de texto necesarios para el tamaño escogido, el siguiente paso consistiría entonces en ingresar los valores del sistema de ecuaciones que se desea resolver, como se muestra en la siguiente figura.

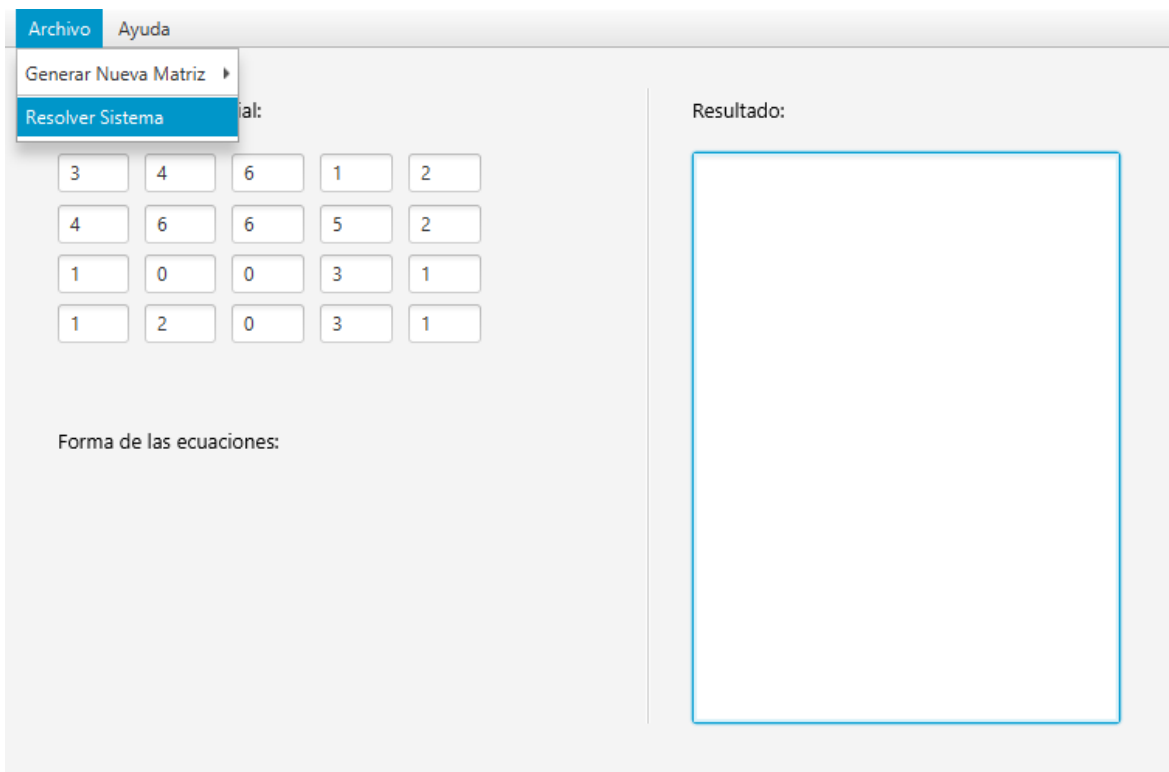
Matriz ampliada inicial:				
3	4	6	1	2
4	6	6	5	2
1	0	0	3	1
1	2	0	3	1

Forma de las ecuaciones:

Resultado:

Figura 2. Ingreso de los valores del sistema deseado.

Una vez concluido el paso anterior, se procede a seleccionar el subitem *Resolver Sistema*, como su nombre lo indica (Ver figura 3), donde se ejecutará el algoritmo Bareiss para encontrar la solución del sistema ingresado.



Efectuado el paso anterior, se mostrará la solución del sistema ingresado.

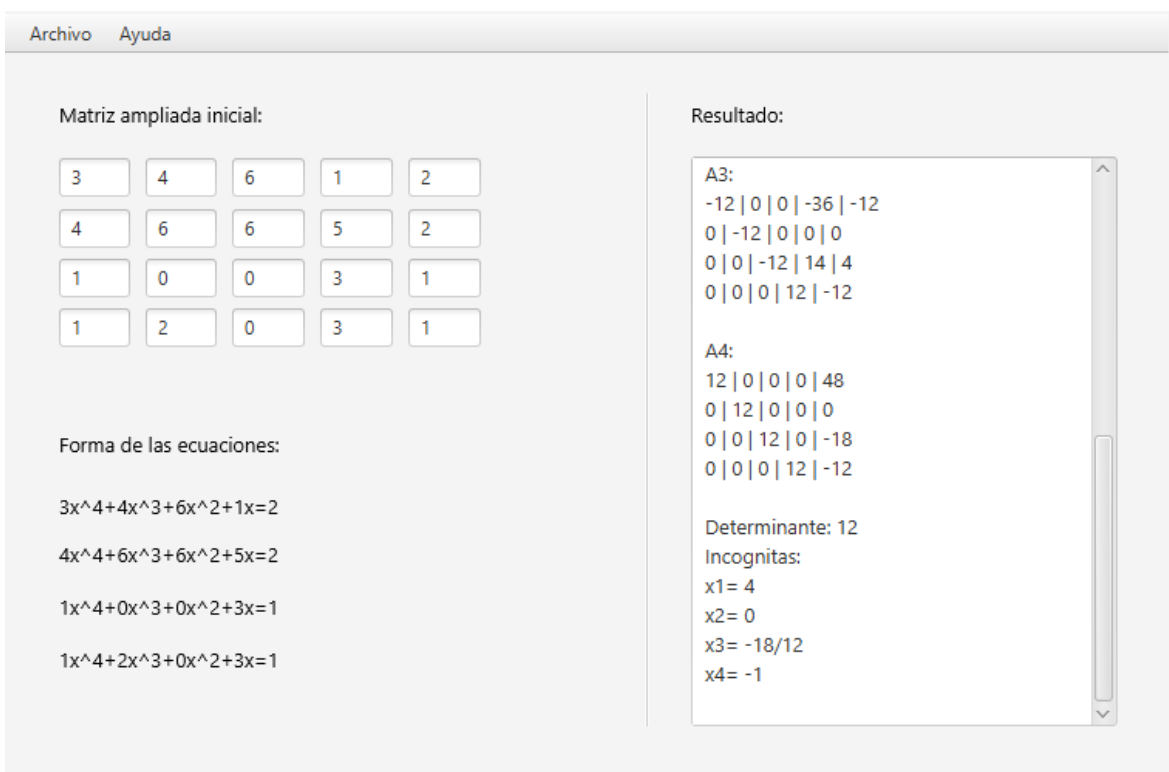


Figura 4. Resolución del sistema ingresado.

Como se puede observar, en la parte inferior izquierda se aprecian las formas de las ecuaciones que conforman el sistema ingresado previamente, esto por el motivo de dar una mejor perspectiva al usuario de los datos que se están ingresando al programa.

- **Secuencia de ejecución de las funciones.**

De primera instancia, se ejecuta la siguiente función, la cual se encarga de dar inicio al algoritmo.

```
public boolean IniciarProceso()
{
    matricesResultantes = ""; incognitasSistema = "";

    if (!ObtenerPivotes())
    {
        IntercambiarFilas();
    }

    AgregarMatrizResultante(0);

    int pivoteAnterior = 1;
    for (int i = 0; i < dimension; i++) {
        DiagonalDeterminante(i);

        try
        {
            Paso_1(i, pivoteAnterior);
            Paso_2(i, pivoteAnterior);
        }
        catch (ArithmeticException e)
        {
            Alerta.MostrarAlerta("Sistema de ecuaciones inválido", "Al |
            return false;
        }

        PonerCerosColumna(i);
        pivoteAnterior = matriz[i][i];
        AgregarMatrizResultante(i + 1);
    }
    EncontrarIncognitas();
    return true;
}
```

De acuerdo con la imagen anterior, primeramente, se comprueba la existencia de ceros en la diagonal:

```
private boolean ObtenerPivotes()
{
    pivotes = new int[dimension];
    for (int i = 0; i < dimension; i++)
    {
        if (matriz[i][i] == 0) { return false; }

        pivotes[i] = matriz[i][i];
    }
    return true;
}
```

De ser verdadera la afirmación, se intercambian filas de la matriz ampliada inicial hasta que su diagonal no tenga:

```
private void IntercambiarFilas()
{
    int[] filaAux = new int[dimension];
    int problema = -1;
    int contador = 0;

    while(((problema = ValidarDiagonal()) != -1) && contador < 1000)
    {
        for(int i = 0; i < dimension; i++)
        {
            filaAux = matriz[problema];
            matriz[problema] = matriz[i];
            matriz[i] = filaAux;

            signoDeterminante *= -1;
        }
        contador++;
    }
}

private int ValidarDiagonal()
{
    for(int i = 0; i < dimension; i++)
    {
        if(matriz[i][i] == 0) { return i; }
    }
    return -1;
}
```

Caso contrario, de no existir ceros en la diagonal, se continúa con el algoritmo.

En la siguiente línea se añade la matriz inicial al resultado que se lleva hasta el momento, sin embargo, no es de importancia del algoritmo Bareiss.

Se inicia un ciclo *for* cuya condición se ejecuta dependiendo de la dimensión del sistema escogido con anterioridad. Posterior, se ejecuta la siguiente función:

```
private void DiagonalDeterminante(int indice)
{
    for (int i = 0; i < indice; i++)
    {
        matriz[i][i] = matriz[indice][indice];
    }
}
```

Donde se asigna a la diagonal del sistema, el índice proveniente del ciclo anterior.

Seguidamente, se efectúa la llamada a dos funciones cuyas operaciones a ejecutar corresponden a las definiciones que establece el algoritmo Bareiss:

```
private void Paso_1(int indice, int pivoteAnterior)
{
    for (int i = 0; i < indice; i++)
    {
        for (int j = indice + 1; j < dimension + 1; j++)
        {
            matriz[i][j] = (((matriz[i][indice] * matriz[indice][j]) -
                               (matriz[indice][indice] * matriz[i][j])) * -1) / pivoteAnterior;
        }
    }
}

private void Paso_2(int indice, int pivoteAnterior)
{
    for (int i = indice + 1; i < matriz.length; i++)
    {
        for (int j = indice + 1; j < dimension + 1; j++)
        {
            matriz[i][j] = ((matriz[indice][indice] * matriz[i][j]) -
                               (matriz[i][indice] * matriz[indice][j])) / pivoteAnterior;
        }
    }
}
```

En vez concluido lo anterior, se procede a poner los ceros de la columna donde se ubica el pivote actual:

```
private void PonerCerosColumna(int pColumna)
{
    for (int x = 0; x < dimension; x++)
    {
        if (x == pColumna) { continue; }
        else { matriz[x][pColumna] = 0; }
    }
}
```

Nuevamente, se almacena la matriz que se lleva hasta al momento, por motivos del resultado y vista requerida.

Una vez concluido, el ciclo *for* del inicio se procede, a encontrar las incógnitas del sistema:

```
private void EncontrarIncognitas()
{
    boolean infinitaSoluciones = false;
    boolean sinSolucion = false;
    String matrizResultante[] = new String [dimension];

    for (int i = 0; i < dimension; i++)
    {
        if( matriz[i][i] == 0 )
        {
            if (matriz[i][dimension] == 0) { infinitaSoluciones = true; }
            else { sinSolucion = true; }
        }
        else
        {
            if (matriz[i][dimension] % matriz[i][i] == 0)
            {
                matrizResultante[i] = Integer.toString(matriz[i][dimension] / matriz[i][i]);
            }
            else
            {
                matrizResultante[i] = Integer.toString(matriz[i][dimension]) + "/" + Integer.toString(matriz[i][i]);
            }
        }
    }
}
```

#### - Forma de acceder a la salida de datos.

Como se puede apreciar anteriormente en la figura 4, la salida de los datos del programa con la solución del problema se halla al lado derecho de la interfaz del programa mediante un componente *TextArea*, se muestran todas las matrices iniciando por la matriz ampliada inicial hasta la matriz final con los valores del determinante en la diagonal y sus respectivos valores en cero.

De manera consecutiva, se muestra el valor del determinante y el valor de cada incógnita según el sistema ingresado previamente.



### III. Aplicaciones de las transformaciones para la presentación grafica de vectores en 3D.

#### - Forma correcta de ejecutar el programa e inserción de los datos.

Primero se procede a ingresar los datos del punto P a ser dibujado en la escena en un formato X,Y,Z. El programa solo recibirá números y las comas (,) para separar dichos datos. El respectivo botón "Enter" a la derecha del campo de entrada se encarga de agregar las coordenadas al sistema, debe ser oprimido para una secuencia exitosa.

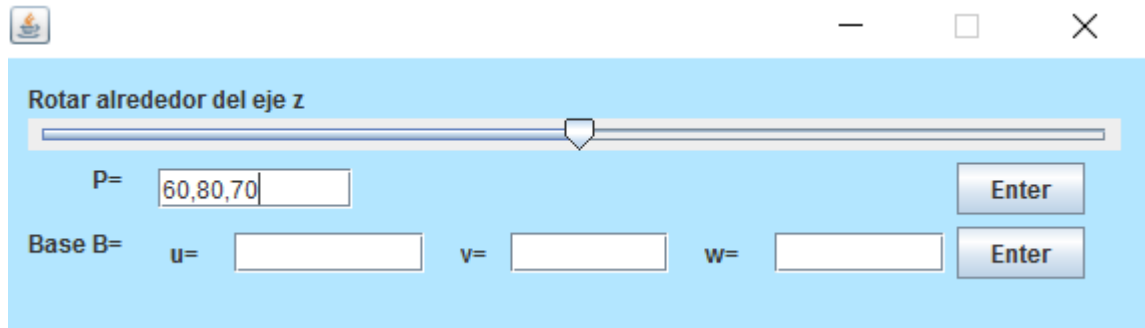


Figura 1. Ingreso de datos

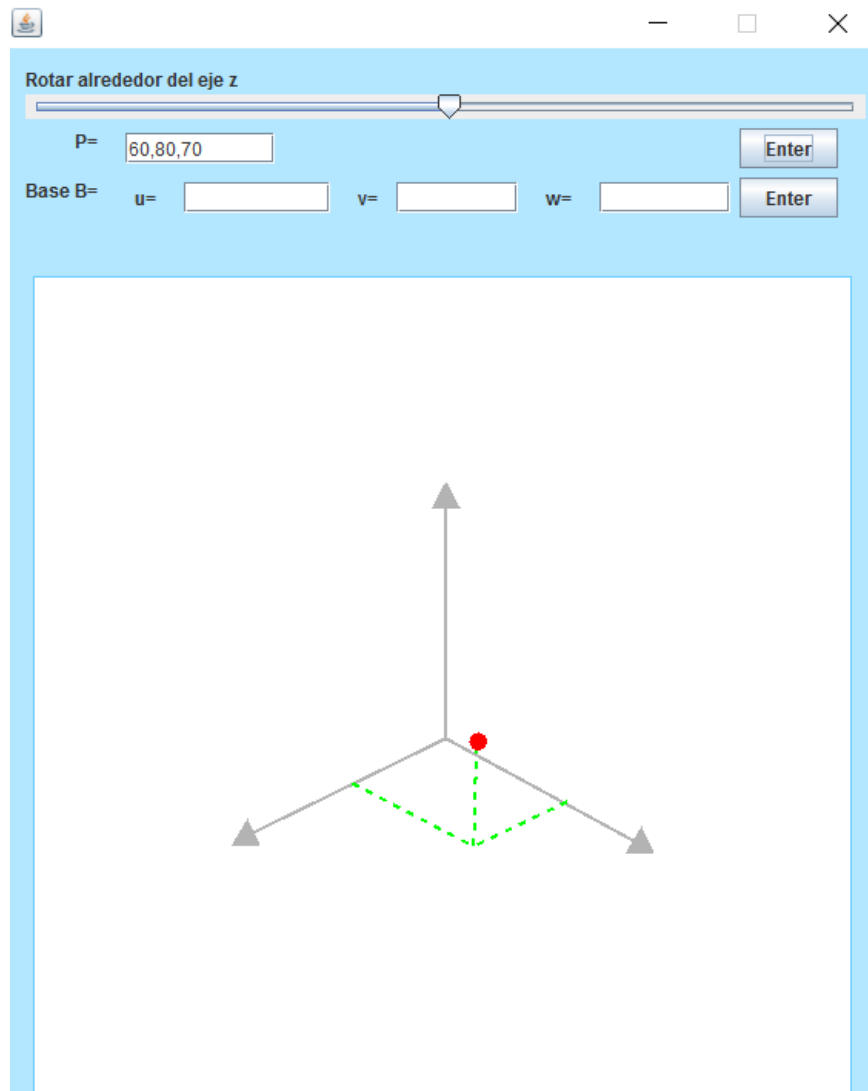


Figura 2. Resultado

Seguidamente se ingresan los valores de la base B en sus respectivos campos u, v, w. Al igual que el punto anterior solo se reciben números y la coma para separar. Al oprimir el segundo botón "Enter" a la derecha del último campo de la base se encarga de actualizar los últimos datos y automáticamente generar la escena.

Rotar alrededor del eje z

P=

Base B= u=  v=  w=

Figura 3. Ingreso datos de la base.

Es de suma importancia que cada vez que se quiera actualizar la escena se debe oprimir el segundo botón "Enter" y cuando se desee cambiar el punto P se debe oprimir su respectivo botón "Enter" nuevamente para actualizar los datos.

La función de la barra que se encuentra en la parte de arriba de la interfaz rotará la escena generada sobre el eje z.

Rotar alrededor del eje z

Figura 4. Barra de rotación.

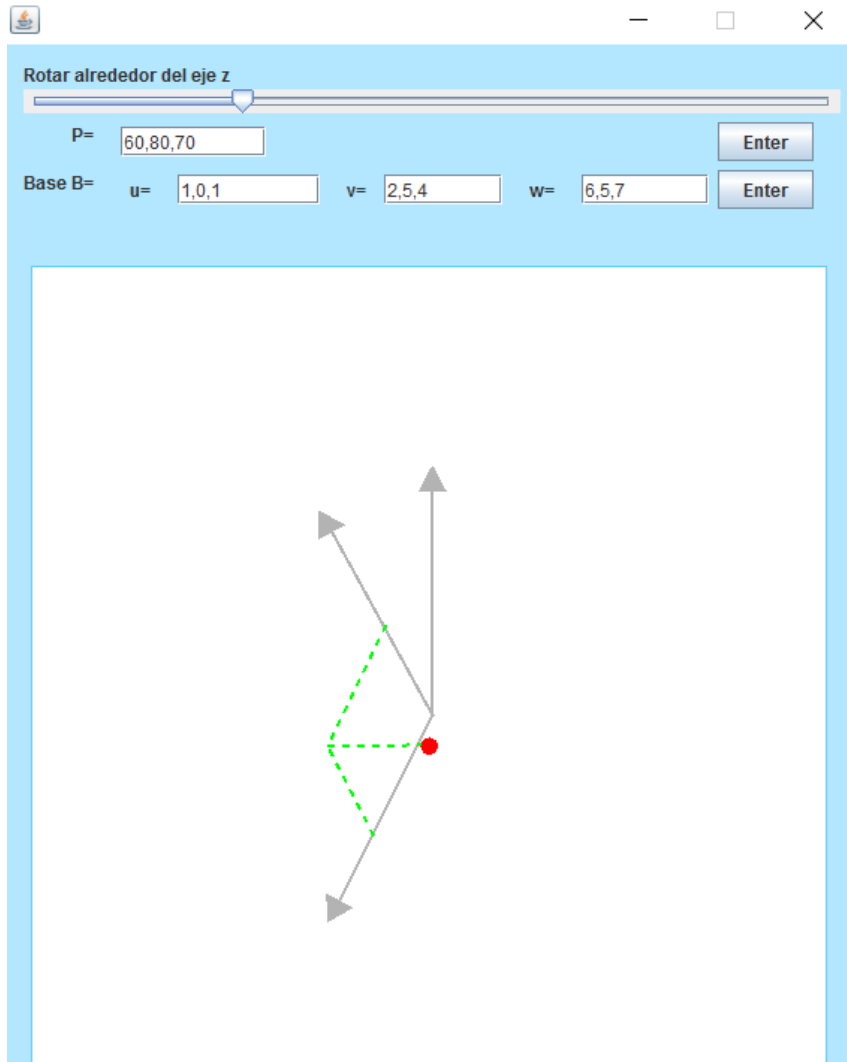


Figura 5. Rotación de gráfica

## - Secuencia de ejecución de las funciones.

Al comienzo al ingresar las coordenadas X, Y, Z del punto P, estas se guardan en el sistema para dibujar dicho punto más adelante en la ejecución. Aquí se valida que la entrada sea válida.

```
private void PuntoActionPerformed(java.awt.event.ActionEvent evt) {  
    String entrada = Textol.getText();  
    if (entrada.matches("^[.,0-9]+$")) {  
        String[] coordenates = entrada.split(",");  
        eje_x = Float.parseFloat(coordenates[0]);  
        eje_y = Float.parseFloat(coordenates[1]);  
        eje_z = Float.parseFloat(coordenates[2]);  
        Slider.setValue(2);  
        drawing_flag = 1;  
        CaculateRotation(eje_x, eje_y, eje_z, angulo);  
    }  
    else  
    {  
        JOptionPane.showMessageDialog(this, "La entrada ingresada no es permitida.\n El formato es: X,Y,Z", "Error", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

Seguidamente se procede a recibir y validar los valores de la base B.

```
private void BaseActionPerformed(java.awt.event.ActionEvent evt) {  
    boolean continuar = ObtenerDatos(false); // Verificar si es l.independiente  
  
    if (continuar)  
    {  
        ObtenerDatos(true); // Obtener los datos pero con la combinacion lineal  
        boolean termino = EjecutarBareiss();  
  
        if (termino)  
        {  
            if (determinante != 0)  
            {  
                a_1 = incognitas_a[0];  
                a_2 = incognitas_a[1];  
                a_3 = incognitas_a[2];  
  
                //eje_x = a_1; eje_y = a_2; eje_z = a_3;  
                CaculateRotation(a_1, a_2, a_3, angulo);  
                setBaseAxis();  
                RefreshScreen();  
            }  
            else  
            {  
                JOptionPane.showMessageDialog(this, "El sistema ingresado no es l.independiente.", "Error", JOptionPane.ERROR_MESSAGE);  
            }  
        }  
        else  
        {  
            JOptionPane.showMessageDialog(this, "El sistema ingresado es inválido.", "Error", JOptionPane.ERROR_MESSAGE);  
        }  
    }  
    else  
    {  
        JOptionPane.showMessageDialog(this, "Por favor inserte correctamente los valores de los tres conjuntos: U, V y W.", "Error", JOptionPane.ERROR_MESSAGE);  
    }  
}
```

Una vez que se poseen todos los datos se procede a realizar los respectivos cálculos utilizando las formulas proporcionadas por el documento de la tarea.

```

void CalculatePoint(float x, float y, float z){
    X = y - x;
    Y = (float)-0.5*(x + y - 2*z);

    //Calculate X axis line
    Xline1 = (y-y) - x; Yline1 = (float)-0.5*(x + (y-y) - 2*(z-z));
    Xline2 = y - x; Yline2 = (float)-0.5*(x + y - 2*(z-z));

    //Calculate Y axis line
    Y_Xline1 = y - (x-x); Y_Yline1 = (float)-0.5*((x-x) + y - 2*(z-z));
    Y_Xline2 = y - x; Y_Yline2 = (float)-0.5*(x + y - 2*(z-z));

    //Calculate Z axis line
    Z_Xline1 = y - x; Z_Yline1 = (float)-0.5*(x + y - 2*(z-z));

}

void setBaseAxis(){...11 lines }

void CaculateRotation(float x, float y, float z, double rotacion){
    rotacion = -rotacion;
    float a = (float)Math.cos(rotacion);float b = -(float)Math.sin(rotacion);float c = 0;
    float d = (float)Math.sin(rotacion);float e = (float)Math.cos(rotacion);float f = 0;
    float g = 0;float h = 0;float i = 1;

    float result1 = a*x + b*y + c*z;
    float result2 = d*x + e*y + f*z;
    float result3 = g*x + h*y + i*z;

    CalculatePoint(result1, result2, result3);

}

```

Y finalmente se procede a dibujar la escena con los resultados de los cálculos anteriores.

```

void RefreshScreen(){
    if(drawing_flag == 0){return;}
    Graphics obj = this.getGraphics();

    // CLEAN PART
    obj.clearRect(21, (ScreenSizeHeight/2)-(ScreenSizeHeight/4)+1, (ScreenSizeWidth-45)-1, (ScreenSizeHeight/2)+(ScreenSizeHeight/4)-35);

    t.setColor(Color.decode("#66ccff"));
    t.drawRect(20, (ScreenSizeHeight/2)-(ScreenSizeHeight/4), (ScreenSizeWidth-45), (ScreenSizeHeight/2)+(ScreenSizeHeight/4));
    t.setColor(Color.decode("#ffffff"));
    t.fillRect(21, (ScreenSizeHeight/2)-(ScreenSizeHeight/4)+1, (ScreenSizeWidth-45)-1, (ScreenSizeHeight/2)+(ScreenSizeHeight/4));

    obj.setColor(Color.decode("#b3b3b3"));

    Graphics2D flecha = (Graphics2D) obj;

    flecha.setStroke(new BasicStroke(2));

    // FLECHA EJE Z
    flecha.drawLine(OriginX, OriginY, OriginArrow1X, OriginArrow1Y);
    flecha.drawPolygon(new int[]{OriginArrow1X-8,OriginArrow1X, OriginArrow1X+8}, new int[]{OriginArrow1Y,OriginArrow1Y-15,OriginArrow1Y}, 3);
    flecha.fillPolygon(new int[]{OriginArrow1X-8,OriginArrow1X, OriginArrow1X+8}, new int[]{OriginArrow1Y,OriginArrow1Y-15,OriginArrow1Y}, 3);

    //Draw X axis dotted line
    flecha.setColor(Color.GREEN);
    Stroke dotted = new BasicStroke(2, BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL, 0, new float[] {5}, 0);
    flecha.setStroke(dotted);
    flecha.drawLine(OriginX+(int)Xline1, OriginY-(int)Yline1, OriginX+(int)Xline2, OriginY-(int)Yline2);
    //Draw Y axis dotted line
    flecha.drawLine(OriginX+(int)Y_Xline1, OriginY-(int)Y_Yline1, OriginX+(int)Y_Xline2, OriginY-(int)Y_Yline2);
    //Draw Z axis dotted line
    flecha.drawLine(OriginX+(int)Z_Xline1, OriginY-(int)Z_Yline1, OriginX+(int)X, OriginY-(int)Y);

    //Draw red dot
    flecha.setStroke(new BasicStroke(2));
    flecha.setColor(Color.red);
    Ellipse2D.Double shape = new Ellipse2D.Double(OriginX+X-4, OriginY-Y-3, 10, 10);
    flecha.draw(shape);
    flecha.fill(shape);

    flecha.setColor(Color.BLUE);
    flecha.drawLine(OriginX, OriginY, (int)(OriginX+ZbaseX), (int)(OriginY-ZbaseY));
    flecha.rotate(angulo, OriginX, OriginY);
    flecha.setColor(Color.decode("#b3b3b3"));
    // FLECHA EJE X
    flecha.drawLine(OriginX, OriginY, OriginArrow2X, OriginArrow2Y);
    flecha.drawPolygon(new int[]{OriginArrow2X-11,OriginArrow2X-3,OriginArrow2X+5}, new int[]{OriginArrow2Y+7,OriginArrow2Y-8,OriginArrow2Y+7}, 3);
    flecha.fillPolygon(new int[]{OriginArrow2X-11,OriginArrow2X-3,OriginArrow2X+5}, new int[]{OriginArrow2Y+7,OriginArrow2Y-8,OriginArrow2Y+7}, 3);

    // FLECHA EJE Y
    flecha.drawLine(OriginX, OriginY, OriginArrow3X, OriginArrow3Y);
    flecha.drawPolygon(new int[]{OriginArrow3X-5,OriginArrow3X+3,OriginArrow3X+11}, new int[]{OriginArrow3Y+7,OriginArrow3Y-8,OriginArrow3Y+7}, 3);
    flecha.fillPolygon(new int[]{OriginArrow3X-5,OriginArrow3X+3,OriginArrow3X+11}, new int[]{OriginArrow3Y+7,OriginArrow3Y-8,OriginArrow3Y+7}, 3);

    flecha.setColor(Color.BLUE);
    flecha.drawLine(OriginX, OriginY, (int)(OriginX+XbaseX), (int)(OriginY-XbaseY));

    flecha.drawLine(OriginX, OriginY, (int)(OriginX+YbaseX), (int)(OriginY-YbaseY));

    flecha.dispose();
}

```

- **Forma de acceder a la salida de datos.**

La escena desplegada en pantalla corresponde a un gráfico con sus tres ejes X, Y, Z el cual demarca dentro de sí un punto cuyas coordenadas son ingresadas por el usuario. También se denota un nuevo gráfico de tres ejes que corresponde al de la base que igualmente fue ingresado por el usuario. La escena tiene la cualidad de rotar alrededor del eje Z (eje gris).