# Tarea Programada

**Indicaciones Generales.** La solución de la tarea puede ser presentada en grupos de 3 o 4 estudiantes, los grupos tienes que estar conformados por estudiantes matriculados en el mismo grupo. La tarea tiene como fecha límite de entrega el viernes 27 de Octubre, antes de las 12:00 medio día. No se calificarán tarea entregada posterior a la fecha y hora indicada. En caso de comprobarse un fraude en la solución de la tarea, se aplicarán las normativas internas vigentes del ITCR.

La tarea consiste en desarrollar tres programas que se detallan más abajo. Los estudiantes podrán utilizar el lenguaje de programación de su elección, tanto para el desarrollo de los algoritmos de programación, como para la visualización de la información (interfaces). También se debe elaborar y presentar un manual de usuario (en formato pdf) donde venga la información personal de cada miembro del grupo, además debe de indicar la forma correcta de inserción de los datos y el uso correcto de los distintos programas. Debe mostrar el código, en digital.

**Objetivos a evaluar.**

1. Aplicar los algoritmos de algebra matricial
2. Comprender los conceptos involucrados en el algoritmo de Eliminación Gaussiana.
3. Comprender e implementar el algoritmo de Bareiss.
4. Aplicar los conceptos de transformaciones para la representación gráfica de vectores en 3D.
5. Fomentar la capacidad de análisis y de razonamiento deductivo.
6. Promover el desarrollo de habilidades para la resolución de problemas utilizando herramientas de programación

**Criterios de evaluación** La tarea de programada será evaluada de la siguiente manera:

| | |
|---|---|
| **1.** Manual de usuario | 10 puntos |
| El manual debe de presentarse en formato pdf, y debe contener la información completa de los integrantes del grupo, nombre del profesor del curso y número de grupo. Además debe indicar la forma correcta de insertar de datos, la secuencia de ejecución de las funciones, la forma de acceder a la salida de datos, entre otros detalles que se consideren necesarios para la correcta ejecución de los programas. | |
| **2.** Funcionalidad del programa | |
| **a.)** Programa uno: Asistente para realizar Eliminación Gaussiana. | 25 puntos |
| **b.)** Programa dos: Algoritmo de Bareiss | 25 puntos |
| **c.)** Programa tres: Aplicaciones de las transformaciones para la presentación gráfica de vectores en 3D. | 25 puntos |
| **3.** Interfaz | 15 puntos |
| **Total** | **100 puntos** |

## 1.1 Asistente para practicar eliminación Gaussiana

1. **Descripción del programa:** Ver un vídeo con una idea general del programa
   Se debe implementar un programa similar al que se muestra en las figuras que siguen:

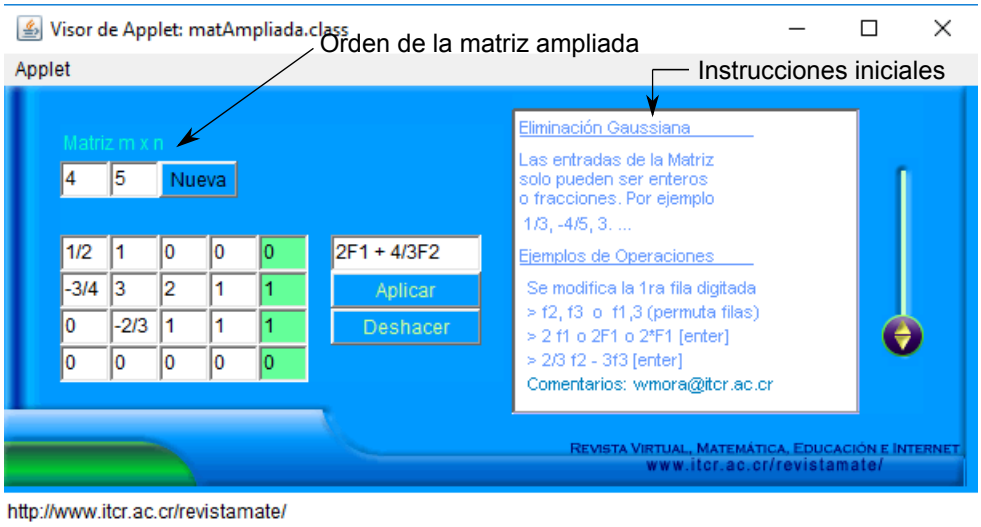



**Figura 1.1:** Esta una idea solamente, de la interfaz

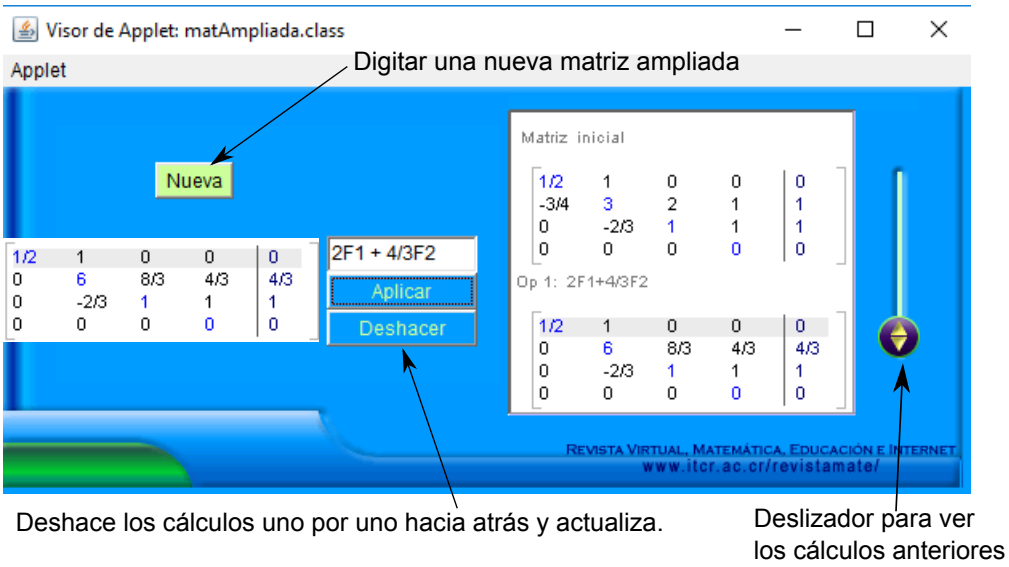



**Figura 1.2:** Esta una idea solamente, de la interfaz

El programa incluye dos campos de texto para indicar el orden de la matriz ampliada, digamos $m \times n$ (con $n, m \leq 5$). Con esta información abrimos una malla de campos de texto para digitar la matriz ampliada. La matriz solo recibe enteros o fracciones $\frac{p}{q} \in \mathbb{Q}$. La aritmética debe ser exacta (es decir, sin decimales). Puede usar una clase **`Rational`** si la tiene disponible, sino puede ver Rational.java.html.

Hay un campo de texto para indicar la operación elemental que se quiere ejecutar. Tratamos de usar lenguaje natural. La operación se puede indicar en minúsculas o mayúsculas pero solo usando la letra **`F`** o la letra **`f`**, además de comas, punto y como y números. Solo se permite digitar:

a.) Cambio de filas: **`fi,j`** o **`fi,fj`** o **`fi;j`** o **`fi; fj`** o **`Fi; Fj`** etc.

b.) Operación estándar: **`bFj`** o **`aFi + bFj`** o **`afi + bfj`** o **`a*Fi + b*Fj`** etc. y, en las sumas, se modifica la última fila: Si la operación es **`aFi + bFj,`** se modifica la fila **`Fj`**.

Si la sintaxis no se satisface, se debe generar un mensaje de error y alguna indicación adicional, como poner el texto del campo en rojo.

La operación se aplica con **`Enter`** en el campo de texto, o con el botón **`Aplicar.`** Si es la primera vez, y en las operaciones que siguen, la nueva matriz se imprime a la izquierda como indica la segunda figura que está más arriba (puede seguir utilizando los campos de texto). A la derecha se imprime cada una de las matrices que se van obteniendo en el proceso precedidas por la operación que se aplicó. Un deslizador vertical permite ver el historial. El botón **`Deshacer`** deshace todas las operaciones, una por una, y actualiza la lista de matrices a ambos lados.

Debe resaltar la fila que se modificó y también el pivote actual, en la matriz de la izquierda.

## 1.2 Eliminación Gaussiana libre de fracciones – Algoritmo de Bareiss

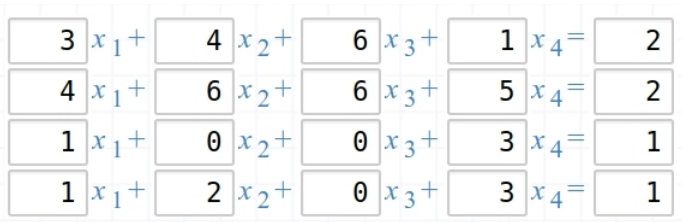
Ver un vídeo con una idea general del programa

**Introducción.** En álgebra computacional y también en ingeniería, es frecuente encontrar sistemas de ecuaciones $AX = B$ donde las entradas son enteros y/o contienen parámetros "libres" o polinomios en una o varias variables con coeficientes enteros. Hay una serie de métodos en cálculo con matrices que son específicos para el cálculo simbólico y el cálculo con arimética exacta (con divisiones exactas, es decir, con residuo nulo) como los métodos de condensación,la eliminación Gaussiana "libre de fracciones" (con uno o dos pasos), entre otros. Este último algoritmo, llamado algoritmo de Bareiss [Bareiss,1968], también se usa para cálculo simbólico de determinantes (estos determinantes se usan intensivamente en teoría de números, solución simbólica de sistemas de ecuaciones no lineales, en factorización de polinomios, en cálculo de integrales racionales, etc).

El algoritmo también sirve para entradas racionales o con decimales, pero para este último caso es mejor usar algoritmos especializados, porque hay que batallar con los errores de redondeo y obtener información acerca de que "tan soluble" es el sistema de ecuaciones (sistemas mal o bien condicionados), etc. Es usual usar métodfos como descomposición **QR** o descomposición en valores singulares (**SVD**), por ejemplo.

**Intrucciones.**

Esta parte de la tarea consiste en implementar el algoritmo de Bareiss (eliminación Gaussiana libre de fracciones, método de "un solo paso") para resolver sistemas *con entradas enteras únicamente* para sistemas con un máximo de cinco ecuaciones y cinco incognitas. Debe agregar un campo de texto para indicar el orden del sistema que se debe desplegar y luego generar la malla de campos de texto, como por ejemplo:



En este caso, la matriz ampliada inicial es $A^{(0)} = \left(\begin{array}{cccc|c} 3 & 4 & 6 & 1 & 2 \\ 4 & 6 & 6 & 5 & 2 \\ 1 & 0 & 0 & 3 & 1 \\ 1 & 2 & 0 & 3 & 1 \end{array}\right)$

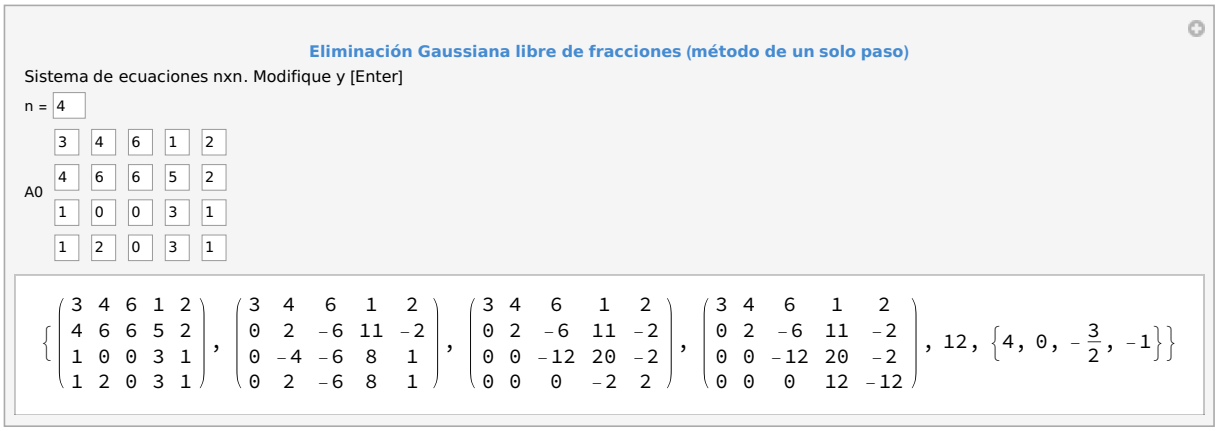



**Figura 1.3:** Esta es una idea solamente, de la interfaz

**Algoritmo** El algoritmo esta basado en un teorema llamado "La identidad de Sylvester" [Geddes, 1992]. Esta identidad permite un proceso de eliminación que solo usa los dos pivotes anteriores y el cálculo de determinantes $2 \times 2$.

Aunque el mismo algoritmo se puede usar para sistemas $m \times n$, solo vamos a considerar el caso $n \times n$.

Consideremos un sistema $A_{n\times n}X = B$ con entradas enteras y solución única ($|A| \neq 0$). Sea $A^{(0)}$ la matriz ampliada $A|B$, es decir, $A^{(0)} = A|B$. El algoritmo "calcula" $n$ matrices

$$A^{(0)} \to A^{(1)} \to A^{(2)} \to ... \to A^{(n-1)}$$

Las entradas de la matriz $k-$ésima se denotan $a_{ij}^{(k)}$, es decir, $A^{(k)} = \left[a_{ij}^{(k)}\right]_{n\times n}$. El resultado principal es:

> La última matriz ampliada $A^{(n-1)}$ está en forma escalonada y además el determinante de la matriz asociada al sistema es **Det**$(A)$= $a_{nn}^{(n-1)}$

**Pasos**.

1. Como el algoritmo usa en cada iteración los dos últimos pivotes, se requiere definir un "pivote auxiliar" para el inicio, denotado $a_{00}^{(-1)} = 1$ y $A^{(0)} = [a_{ij}^{(0)}]$

2. Ahora calculamos cada una de las matrices $A^{(k)} = [a_{ij}^{(k)}]$ con $k = 1, 2, ..., n-1$. Si el pivote $a_{kk}^{(k-1)} \neq 0$,

$$a_{ij}^{(k)} = \frac{a_{kk}^{(k-1)} a_{ij}^{(k-1)} - a_{kj}^{(k-1)} a_{ik}^{(k-1)}}{a_{k-1,k-1}^{(k-2)}}, \quad \text{para} \quad k+1 \leq i \leq n, \;\; k+1 \leq j \leq n+1$$

   Cada vez que calculamos $A^{(k)}$, manualmente anulamos las entradas de la columna $k$ que están debajo de la diagonal: $a_{sk}^{(k)} = 0$ para $s = k+1, ..., n$.

Si el pivote $a_{kk}^{(k-1)} = 0$, hacemos cambio de fila y llevamos un control de los cambio de fila en una variable que inicia en 1, *sign=1* (si hay cambio de fila *sign*=$-$*sign*), para ajustar el determinante al final.

3. Este proceso termina en $A^{(n-1)}$. Esta matriz ampliada esta en forma escalonada y además el determinante de la matriz asociada al sistema es **Det**$(A)$= $sign \cdot a_{nn}^{(n-1)}$

4. Solución del sistema $AX = B$. Primero se hace con sustitución hacia atrás "libre de fracciones" y luego se ajusta con el determinante (estas fórmulas requieren haber guardado los ingredientes previamente, en los cálculos anteriores)

$$x_n^* = a_{n,n+1}^{(n-1)}$$

$$x_k^* = \frac{a_{n,n}^{(n-1)} a_{k,n+1}^{(k-1)} - \sum_{j=k+1}^{n} a_{k,j}^{(k-1)} \boldsymbol{x_j}}{a_{k,k}^{(k-1)}} \quad \text{para} \quad k = n-1,\, n-2,\, ...,1$$

Finalmente: $x_i = \dfrac{x_i^*}{\mathbf{Det}(A)}$ para $i = 1, 2, ..., n$

---

**Salida.** El programa deberá imprimir cada una de las matrices $A^{(0)}$, $A^{(1)}$, $A^{(2)}$, ..., $A^{(n-1)}$, el determinante y la solución del sistema. Si la matriz asociada no es invertible (no hay posiblidad de encontrar un pivote no nulo, en el paso $k$) entonces se indica que el sistema no tiene solución única.

En el caso de la matriz de entrada de la figura que esta arriba, la salida debería ser

$$A^{(0)} = \left(\begin{array}{cccc|c} 3 & 4 & 6 & 1 & 2 \\ 4 & 6 & 6 & 5 & 2 \\ 1 & 0 & 0 & 3 & 1 \\ 1 & 2 & 0 & 3 & 1 \end{array}\right), \; A^{(1)} = \left(\begin{array}{cccc|c} 3 & 4 & 6 & 1 & 2 \\ 0 & 2 & -6 & 11 & -2 \\ 0 & -4 & -6 & 8 & 1 \\ 0 & 2 & -6 & 8 & 1 \end{array}\right),$$

$$A^{(2)} = \left(\begin{array}{cccc|c} 3 & 4 & 6 & 1 & 2 \\ 0 & 2 & -6 & 11 & -2 \\ 0 & 0 & -12 & 20 & -2 \\ 0 & 0 & 0 & -2 & 2 \end{array}\right), \; A^{(3)} = \left(\begin{array}{cccc|c} 3 & 4 & 6 & 1 & 2 \\ 0 & 2 & -6 & 11 & -2 \\ 0 & 0 & -12 & 20 & -2 \\ 0 & 0 & 0 & 12 & -12 \end{array}\right)$$

**Det**$(A)$= $sign \cdot a_{44}^{(3)} = 12$

$x_1^* = 48,\; x_2^* = 0,\; x_3^* = -18,\; x_4^* = -12$

Y la solución del sistema: $x_1 = \dfrac{48}{12} = 4,\; x_2 = \dfrac{0}{12} = 0,\; x_3 = \dfrac{-18}{12} = -\dfrac{3}{2},\; x_4 = \dfrac{-12}{12} = -1$

**¿Por qué funciona el método?.** Una idea muy intuitiva de por qué funciona el método, y la justificación teórica, la puede ver en [Geddes, pp. 389-399]

### Opcional.

Se debe presentar el programa anterior según la especificación que sigue. Los puntos adicionales que se gana son: 10 puntos, aún así, la nota final de la tarea no puede superar el 100 %.

**Aplicar el algoritmo a una matriz simbólica.** Para este ejercicio vamos a suponer que las entrada de la matriz ampliada tiene números enteros y/o polinomios con coeficientes enteros en la variable $x$ (el caso general sería admitir entradas con polinomios en varias variables con coeficientes enteros). Ahora, se aplica el algoritmo pero, como las entradas son polinomios constantes (los enteros) u otro tipo de polinomios $P(x)$, entonces las operaciones de suma, resta, multiplicación y división se deben hacer con una clase que sume, reste, multiplique y divida polinomios (las divisiones son exactas, van a tener resto 0, por lo que solo se necesita calcular el cociente). En el archivo adjunto se incluye una clase **`Qpolinomio.java`** que lee los polinomios, hace las operaciones aritméticas solicitadas e imprime (no necesariamente se debe programar en Java). Y, el algoritmo debe funcionar bien.

**Ejemplo:** Recibe un sistema cuyos parámetros son polinomios en $x$ con coeficientes enteros. Digamos que $A^{(0)} = \left(\begin{array}{ccc|c} 2 & x & 2 & 1 \\ 1 & x-1 & 4 & 2 \\ 1 & 0 & x & x \end{array}\right)$, entonces el programa entrega

$$A^{(0)} = \left(\begin{array}{ccc|c} 2 & x & 2 & 1 \\ 1 & x-1 & 4 & 2 \\ 1 & 0 & x & x \end{array}\right),\ A^{(1)} = \left(\begin{array}{ccc|c} 2 & x & 2 & 1 \\ 0 & x-2 & 6 & 3 \\ 0 & -x & 2(x-1) & 2x-1 \end{array}\right),$$

$$A^{(2)} = \left(\begin{array}{ccc|c} 2 & x & 2 & 1 \\ 0 & x-2 & 6 & 3 \\ 0 & 0 & x^2+2 & x^2-x+1 \end{array}\right)$$

**Det**$(A)$= $sign \cdot a_{33}^{(2)} = x^2+2$

$x_1^* = x(x+1),\ x_2^* = -3x,\ x_3^* = x^2-x+1$

Y la solución del sistema: $x_1 = \dfrac{x(x+1)}{x^2+2},\ x_2 = -\dfrac{3x}{x^2+2},\ x_3 = \dfrac{x^2-x+1}{x^2+2}$

**Nota:** Observe que el algoritmo debería (aunque no se pide en esta tarea) analizar los casos especiales en los que se asumió que el pivote no se anula. Por ejemplo, si un pivote fue $x-2$, debemos analizar el caso $x=2$ como paso especial y así sucesivamente...

## 1.3 Aplicación de las transformaciones lineales a la visualización de figuras 3D

Ver un vídeo con una idea general del programa

**Introducción.** En este ejercicio vamos a graficar puntos $P \in \mathbb{R}^3$, bases $B$ del espacio vectorial $\mathbb{R}^3$ y la representación de las coordenadas de cada punto $P$ en cada base $B$, en 3D. Además vamos a implementar una rotación alrededor del eje $Z$ vía una transformación, para mejorar la visualización. *Solo vamos a usar la clase* **`"Graphics2D"`** *de su lenguaje* (es decir, no se puede usar los métodos de una clase **`Graphics3D`**).

**Intrucciones:**

Debemos implementar un programa que recibe un punto $P \in \mathbb{R}^3$ con coordenadas racionales $\frac{a}{b} \in \mathbb{Q}$. El programa recibe las coordenadas de $P$ en base canónica: $P = (p_1, p_2, p_3)$. El programa también recibe una base cualquiera $B = \{\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}\}$ de $\mathbb{R}^3$. Observe que debe verificar, cada vez que se modifique $B$, que este conjunto sea de tres elementos y que sea l.i., en caso contrario se dispara un mensaje de error.

Luego, el programa debe calcular las coordenadas de $P$ en la base $B$, es decir, $[P]_B = (a_1, a_2, a_3)$ si $P = a_1\boldsymbol{u} + a_2\boldsymbol{v} + a_3\boldsymbol{w}$. El programa imprime estas nuevas coordenadas $(a_1, a_2, a_3)$ y grafica en 3D, usando una transformación isométrica (ver más abajo). Los ejes $X$, $Y$ y $Z$ los gráfica en un gris suave, solo como referencia. En el mismo sistema se grafica, en un color fuerte, el punto $P$, los nuevos ejes generados por la base $B$ y las coordenadas de $P$ en esta base. Debe usar líneas puntadas para hacer esto último. Uds puede elegir cómo implementar los campos de texto para recibir la información. Además el programa debe tener un deslizador para rotar la escena alrededor del eje $Z$.

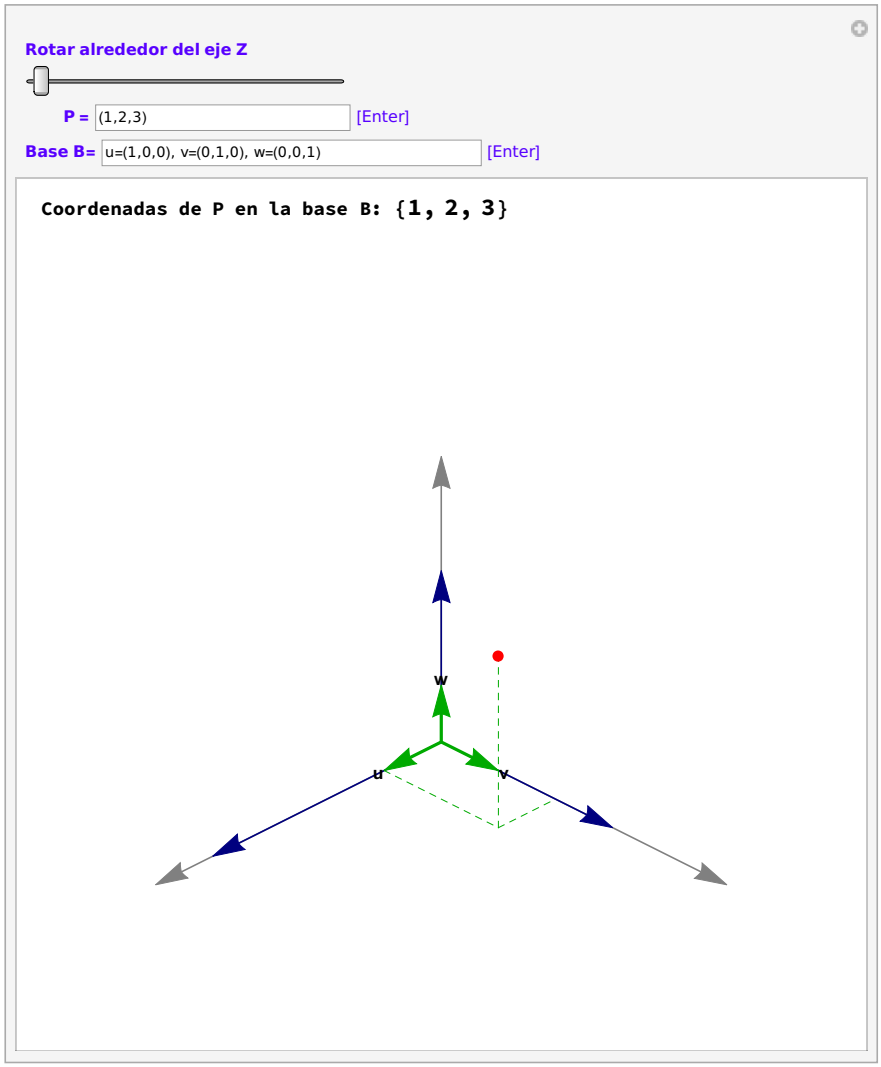



**Figura 1.4:** Esta una idea solamente, de la interfaz

**Métodos para graficar.** Para graficar vamos a usar una transformacion lineal llamada *isometría*. Esta

transformación toma un punto $P \in \mathbb{R}^3$ y devuelve coordenadas 2D para graficar en pantalla (esta proyección es muy usada en juegos). La representación gráfica que obtenemos es una imagen en perpectiva (isométrica). También vamos a usar una "matriz de rotación" para rotar la escena alrededor del eje $Z$ solamente, con un deslizador (puede usar eventos de ratón si prefiere).

La transformacion isométrica es $\texttt{Iso}(x,y,z) = (y-x,\ -\frac{1}{2}(x+y-2z)) \in \mathbb{R}^2$.

Por ejemplo: $\texttt{Iso}(0,0,0) = (0,0) \in \mathbb{R}^2$ e $\texttt{Iso}(2,4,4) = (2,1) \in \mathbb{R}^2$

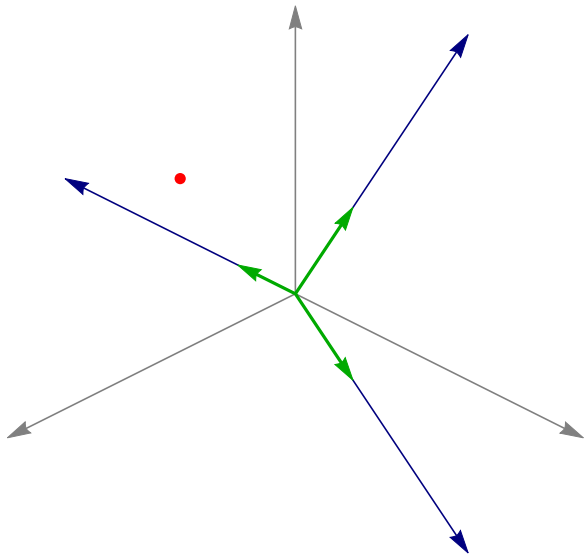
De esta manera podemos crear métodos para dibujar puntos, líneas y flechas "en perpectiva".

En seudocódigo sería:

```
Iso[x,y,z] = {y-x,  -0.5(x+y-2z}
linea[P,Q] = line2D[Iso[P],Iso[Q]]
flecha[P,Q] = arrow2D[Iso[P],Iso[Q]]
punto[P] = point2D[Iso[P]]
```

Por ejemplo (seudocódigo),

```
// punto
P = {4, 2, 5};
//Base
u = {1, 0, 1};
v = {1, 2, 0}; w = {0, 1, 2};
Graphics2D[{
(*Ejes XYZ *)
Gray,
flecha[{0, 0, 0}, {5, 0, 0}],
flecha[{0, 0, 0}, {0, 5, 0}],
flecha[{0, 0, 0}, {0, 0, 5}],
(*Nuevos ejes, base B*)
RGBColor[0, 0, 0.5],
flecha[{0, 0, 0}, 4 u],
flecha[0*v, 3 v],
flecha[{0, 0, 0}, 3 w],
(*Flechas*)
Thick, RGBColor[0, 0.67, 0],
flecha[{0, 0, 0}, u],
flecha[{0, 0, 0}, v],
flecha[{0, 0, 0}, w],
(*Punto P*)
Red, punto[P]
}]
```



**Rotación.** Para hacer la rotación de un punto $P = (x,y,z)$ alrededor del eje $Z$ debemos usar una transformación lineal denotada con $R_z(\theta)$. Esta transformación, aplicada a un punto $P \in \mathbb{R}^3$, rota este punto alrededor del eje $Z$, en un ángulo $\theta$, en radianes o grados, depende de cómo la defina.

Para operar, usamos la representación matricial de esta transformación: $R_z(\theta) = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Para hacer que la escena rote alrededor del eje $Z$ ponemos un deslizador que controle un parámetro $\theta$ y definimos una nueva función **IsoR**$(x, y, z, \theta)$:

$$\textbf{IsoR}(x, y, z, \theta) = \textbf{Iso}\left( R_z(\theta) \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right)$$

y volvemos a crear la escena, redefiniendo los comandos anteriores con **IsoR.**

Las líneas punteadas, para las coordenadas de $P$, se deben hacer dibujando un paralelogramo.

Para hacer pruebas, se sugiere usar $P = (1, 2, 3)$ con los conjuntos $B_1 = \{(1,0,0), (0,1,0), (0,0,1)\}$, $B_2 = \{(1,0,1), (0,1,1), (0,0,1)\}$, y $B_1 = \{(1,0,1), (0,1,1), (1,0,1)\}$ (este último conjunto debe disparar un mensaje de error, pues el conjunto no es l.i.)

## Bibliografía


1.) E. H. Bareiss. "Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination". Math. Comp. 22(103) pp. 565-578 (1968).

2.) John Vince. *Mathematics for Computer Graphics.* Springer, 2010. 3rd Edition

3.) K. Geddes, S. Czapor, G. Labahn. *Algorithms for Computer Algebra.* Springer; 1992.