# Reporte de Proyecto

## Biblioteca para álgebra lineal

Fhernanda M. Romo Olea

Enero 2021

## 1. Introducción

En la biblioteca implementada abarca la teoría básica de espacios vectoriales de dimensión finita, combinaciones lineales, dependencia e independencia lineal, bases y dimensión, temas frecuentemente abordados en el álgebra lineal. La intención de ésta es de facilitar la comprensión y los cálculos más recurridos en la materia, tal como la resolución de sistemas de ecuaciones lineales o cálculos con matrices. La implementación se ha realizado utilizando el lenguaje de programación funcional *Haskell*, el usuario final puede, tanto utilizar la biblioteca para realizar los cálculos como mejor lo prefiera, o servirse del programa que permite interactuar con esta de forma indirecta y sencilla.

## 2. Espacios Vectoriales

Debido a que la implementación gira en torno a la teoría básica del álgebra lineal, es necesario definir lo que es un espacio vectorial.

**Definición 1** *Un espacio vectorial o espacio lineal $V$ sobre un campo $K$ consiste en un conjunto en el que están definidas dos operaciones, llamadas .ªdiciónτ "multiplicación por escalares", tal que para cualquier par de elementos $x$, $y$ en $V$, y $f$ en $K$ se cumpla que $x + y$ sea único en $V$, al igual que $fx$.*

Adicional a la definición anterior, un espacio vectorial cumple también 8 propiedades, tales como la conmutatividad, la asociatividad, existencia de neutros aditivos y multiplicativos, entre otras. Los elementos pertenecientes a $V$ son llamados vectores y los que pertenecen a $K$ escalares.

Múltiples ejemplos de espacios vectoriales pueden ser mencionados como ejemplo, sin embargo, tres de los más relevantes y utilizados son los siguientes, cabe mencionar que de igual forma, estos serán los espacios vectoriales que más adelante se implementarán.

*Siendo $F$ un campo*

- El espacio vectorial $F^n$ de n-dimensionales
- El espacio vectorial $M_{mxn}(F)$ de matrices $m$ x $n$
- Finalmente el espacio vectorial $P(F)$ de todos los polinomios

### 2.1. Implementación

Para representar espacios vectoriales en la implementación se define una clase de tipos $EV$ que posee una restricción a solamente tipos que implementen la clase $Fractional$, esto para simular que se está trabajando en un campo.

```
class EV a where
    suma :: (Fractional e) => (a e) -> (a e) -> (a e)
    multiplica :: (Fractional e) => e -> (a e) -> (a e)
```

**Clase de tipos representando a un espacio vectorial**

De esta forma, para que alguna estructura pueda ser un espacio vectorial, estará obligada a definir la suma de vectores y la multiplicación por escalares en dicha estructura, cumpliendo así con los requisitos más importantes para que esta sea considerada espacio vectorial. Es de esta forma, que se puede proceder a definir el tipo ***Vector*** y el tipo ***Matriz***.

```haskell
-- tipo de dato vector
data Vector a = Vector [a]

-- se vuelve participe de la clase de tipos EV
instance EV Vector where
    suma (Vector x) (Vector y) = Vector (zipWith (+) x y)
    multiplica k x = fmap (*k) x
```

**Se define el tipo Vector como instancia de EV**

En el código anterior se define de forma esencial el tipo ***Vector***, en la implementación original, primero se verifica que ambos vectores posean la misma longitud para poder sumarse, y por supuesto, se define la función *fmap* para ***Vectores*** que permite definir de forma elegante la multiplicación por escalares.

Si bien los polinomios cuentan con una gran importancia en el álgebra lineal, y figuran entre los elementos más recurridos al momento de trabajar con espacios vectoriales, es posible representar y trabajar con polinomios por medio de vectores, así el tipo ***Vector*** de la implementación será de ayuda para representar n-ádas y a los polinomios.

```haskell
-- tipo de dato matriz
data Matriz a = Matriz [Vector a]

-- se define como instancia de EV
instance EV Matriz where
    suma (Matriz xs) (Matriz ys) = let p = zip xs ys
                in Matriz (map (\(a,b) -> suma a b) p)
    multiplica a m = fmap (*a) m
```

**Se define el tipo Matriz como instancia de EV**

Se define una matriz como una lista de *vectores fila*, por ejemplo:

`La representación:`

`m = Matriz [Vector [1,2,3], Vector [1,2,3], Vector [0,0,0]]`

`es equivalente a la matriz:`

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 0 & 0 & 0 \end{pmatrix}$$

Similar a lo ocurrido con los vectores, en este caso antes de poder realizar la suma se verifica que en efecto la operación pueda ser posible verificando que las matrices tengan el mismo tamaño, la función *fmap* nuevamente sirve de apoyo para instanciar *multiplica*.
Aprovechando la implementación de matrices, el tipo ***Matriz*** será utilizado también como apoyo para la resolución de otro tipo de problemas, tales como la resolución de sistemas de ecuaciones lineales y la determinación de dependencia lineal e independencia lineal, lo cual se tratará más adelante.

# 3. Trabajando con matrices

El capitulo anterior trató únicamente lo relacionado a los espacios vectoriales y cómo se definen estos dentro de la implementación de la biblioteca. Al igual que dentro de la teoría del álgebra lineal; si bien a las matrices de determinada dimensión, con coeficientes en un campo, se les ve como un espacio vectorial, también es posible servirse de ellas para hallar soluciones a determinados problemas que salen del alcance de lo que lo que las matrices como espacio vectorial permiten hacer. Ejemplo de ello hallar solución a un sistema de ecuaciones lineales, como se explicó anteriormente. En este capítulo, los algoritmos a exponer se sirven de estas estructuras para la resolución de los problemas que se plantearán.