Librería en Python para Matemáticas II (Álgebra Lineal)

https://github.com/mbujosab/LibreriaDePythonParaMates2

Marcos Bujosa

January 8, 2020

Índice

Declaración de intenciones

Uno de los objetivos que me he propuesto para el curso Matemáticas II (Álgebra Lineal) es mostrar que escribir matemáticas y usar un lenguaje de programación son prácticamente la misma cosa. Este modo de proceder debería ser un ejercicio muy didáctico ya que:

> *Un PC es muy torpe y se limita a ejecutar literalmente lo que se le indica (un PC no interpreta interpolando para intentar dar sentido a lo que se le dice... eso lo hacemos las personas, pero no los ordenadores).*
>
> *Por tanto, este ejercicio nos impone una disciplina a la que en general no estamos acostumbrados:* el ordenador hará lo que queremos solo si las expresiones tienen sentido e indican correctamente lo que queremos. *Si el ordenador no hace lo que queremos, será porque que hemos escrito las ordenes de manera incorrecta (lo que supone que también hemos escrito incorrectamente las expresiones matemáticas).*

Con esta idea en mente:

1. La notación de las notas de clase pretende ser operativa, en el sentido de que su uso se pueda traducir en operaciones que debe realizar el ordenador.

2. Muchas demostraciones son algorítmicas (al menos las que tienen que ver con el método de Gauss), de manera que dichas demostraciones describen literalmente la programación en Python de los correspondientes algoritmos.

Una librería de Python específica para la asignatura

Aunque Python tiene librerías que permiten operar con vectores y matrices, *aquí escribimos nuestra propia librería.* Con ello lograremos que la notación empleada en las notas de clase y las expresiones que usemos en Python se parezcan lo más posible.

ESTE DOCUMENTO DESCRIBE TANTO EL USO DE LA LIBRERÍA COMO EL MODO EN EL QUE ESTÁ PROGRAMADA; PERO NO ES UN CURSO DE PYTHON.

No obstante, y pese a la nota de anterior, he escrito unos notebooks de Jupyter que ofrecen unas breves nociones de programación en Python (muy incompletas). Tenga en cuenta que hay muchos cursos y material disponible en la web para aprender Python y que mi labor es enseñar Álgebra Lineal (no Python).

Para hacer más evidente el paralelismo entre las definiciones de las notas de la asignatura y el código de nuestra librería, las partes del código menos didácticas se relegan al final[1] (véase la sección *Literate programming* en la Página 97). Destacar algunas partes del código permitirá apreciar que las definiciones de las notas de la asignatura son implementadas de manera literal en nuestra librería de Python.

Tutorial previo en un Jupyter notebook

Antes de seguir, repase el Notebook **"Listas y tuplas"** en la carpeta "TutorialPython" en `https://mybinder.org/v2/gh/mbujosab/LibreriaDePythonParaMates2/master`

Y recuerde que **¡hacer matemáticas y programar son prácticamente la misma cosa!**

[1] aquellas que tienen que ver con la comprobación de que los inputs de las funciones son adecuados, con otras formas alternativas de instanciar clases, con la representación de objetos en Jupyter usando código LaTeX, etc.

Capítulo 1

Código principal de la librería. Las clases `Sistema`, `Vector`, `Matrix` y `T`

Tutorial previo en un Jupyter notebook

Antes de seguir, mírese el Notebook referente a **“Clases”** en la carpeta “TutorialPython” en
`https://mybinder.org/v2/gh/mbujosab/LibreriaDePythonParaMates2/master`

Usando lo mostrado en el Notebook anterior, definiremos una *clase* en Python para los *vectores*, otra para las *matrices*, otra para las *transformaciones elementales* y otra para las *matrices por bloques* (o matrices particionadas).

1.1 La clase `Sistema`

En las notas de la asignatura se dice que

Un *sistema* es una “lista” de objetos.

y por defecto, los sistemas se mostrarán entre *corchetes* y con los elementos de la lista *separados por “;”*. [1]

Aunque Python ya posee “listas”, vamos a crear una clase denominada `Sistema`. Al poder definir cómo se representa la clases, permitiremos que Jupyter use las representaciones especiales de los objetos que vayamos definiendo en el curso. Así por ejemplo, un sistema formado por una lista de 3 transformaciones elementales

`Sistema( [T((5,4)), T((2,3)), T({1,2})] )`

será representado en Jupyter así:

$$\left[\underset{[(5)\mathbf{4}]}{\tau};\ \underset{[(2)\mathbf{3}]}{\tau};\ \underset{[\mathbf{1}\rightleftharpoons\mathbf{2}]}{\tau}\right]$$

es decir, entre corchetes, *con los elementos mostrados con su propia representación especial* y separados por “;”. Por tanto, un `Sistema` y una `list` solo se diferenciarán en el modo de representación de los objetos contenidos en sus respectivas listas. El único atributo de `Clase` es la lista `sis` de objetos, almacenada en una `list`.

4 ⟨*Texto de ayuda de la clase* `Sistema` 4⟩≡
```
"""Clase Sistema

Un Sistema es una lista ordenada de objetos. Los Sistemas se instancian
con una lista, tupla, Vector, Matrix, BlockMatrix, o con otro Sistema
de objetos.

Parámetros:
    data (list, tuple, Vector, Matrix, BlockMatrix, Sistema): Lista o
```

[1] Como veremos, los vectores y matrices, que también son sistemas, se representarán de un modo especial.

```
        tupla de objetos (u objeto formado por un Sistema de objetos).

    Atributos:
        sis (list): lista de objetos.

    Ejemplos:
    >>> # Crear un Sistema a partir de una lista (o tupla) de números
    >>> Sistema( [1,2,3] )   # con lista
    >>> Sistema( (1,2,3) )   # con tupla

    [1; 2; 3; 4]

    >>> # Copiar un Sistema o formar un nuevo Sistema copiando el Sistema
    >>> # de un Vector, Matrix o BlockMatrix
    >>> Sistema( Sistema( [1,2,3] ) )  # copia
    >>> Sistema( A )   # Sistema con los objetos contenidos A.sis (donde A
                                        es un Vector, Matrix o BlockMatrix)
    """
```

This code is used in chunk 7.
Uses BlockMatrix 92b, Matrix 13, Sistema 7, and Vector 9b.

La clase Sistema se inicializa con una lista o tupla, o bien con otro Sistema, o bien con un Vector, Matrix, o BlockMatrix. Cuando data es una lista o tupla, el atributo lista es la lista (o la tupla convertida en lista). Si data es otro Sistema, se crea una copia del Sistema copiando su atributo lista. Los objetos Vector, Matrix, y BlockMatrix tienen un Sistema en sus respectivos atributos sis, de manera que cuando data es uno de estos objetos, se crea un Sistema que es una copia del correspondiente Sistema almacenado en data.sis (copiando la lista data.sis.lista en el atributo self.lista).

5 ⟨*Inicialización de la clase* Sistema 5⟩≡

```
    def __init__(self, data):
        """Inicializa un Sistema"""
        if isinstance(data, (list, tuple)):
            self.lista = list(data)
        elif isinstance(data, Sistema):
            self.lista = data.lista.copy()
        elif isinstance(data, (Vector, Matrix, BlockMatrix)):
            self.lista = data.sis.lista.copy()
        else:
            raise \
    ValueError('El argumento debe ser una lista, tupla, Sistema, Vector, Matrix\
     o BlockMatrix ')
```

This code is used in chunk 7.
Uses BlockMatrix 92b, Matrix 13, Sistema 7, and Vector 9b.

Un Sistema va a ser como una lista de Pyhton, salvo por su modo de representación. Así que vamos a definir los métodos de selección y modificiación de sus elementos, la concatenación y un método que cuente el número de elementos del Sistema de manera anáoga a como se hace con una list.

Así, para que un Sistema sea iterable (como lo es una list) definimos los procedimientos "mágicos" __getitem__, que permite seleccionar una componente del sistema, y __setitem__, que permite modificar una componente del Sistema. Recuerde que los índices de las listas comienzan en 0. Tambiém vamos respetar ese "pythonesco" modo de indexar los Sistemas (para que sean como las list salvo por el modo de representación).

6a ⟨*Métodos de la clase* `Sistema` *para que actue como si fuera una* `lista` 6a⟩≡

```
def __getitem__(self,i):
    """ Devuelve el i-ésimo coeficiente del Sistema """
    return self.lista[i]

def __setitem__(self,i,value):
    """ Modifica el i-ésimo coeficiente del Sistema """
    self.lista[i]=value
```

This definition is continued in chunk 6b.
This code is used in chunk 7.
Uses `Sistema` 7.

Concatenamos los `Sistema`s del mismo modo que las listas de Python, con "+". Con `len` contamos el número de elementos del `Sistema`, y con `copy` hacemos una copia, por ejemplo `z=y.copy()` hace una copia del `Sistema` `y`. Podemos comprobar si dos `Sistema`s son iguales con "`==`", y si son distintos con "`!=`".

6b ⟨*Métodos de la clase* `Sistema` *para que actue como si fuera una* `lista` 6a⟩+≡

```
def __add__(self,other):
    """ Concatena dos Sistemas """
    if not isinstance(other, Sistema):
        raise ValueError('Un Sistema solo se puede concatenar con otro Sistema')
    return Sistema(self.lista + other.lista)

def __len__(self):
    """Número de elementos del Sistema """
    return len(self.lista)

def copy(self):
    """ Copia la lista de otro Sistema"""
    return Sistema(self.lista.copy())

def __eq__(self, other):
    """Indica si es cierto que dos Sistemas son iguales"""
    return self.lista == other.lista

def __ne__(self, other):
    """Indica si es cierto que dos Sistemas son distintos"""
    return self.lista != other.lista

def __reversed__(self):
    """Devuelve el reverso de un Sistema"""
    return Sistema(list(reversed(self.lista)))
```

This code is used in chunk 7.
Uses `Sistema` 7.

La clase `Sistema` junto con el listado de sus métodos aparece en el siguiente recuadro:

7 ⟨*La clase Sistema* 7⟩≡

```
class Sistema:
    ⟨Texto de ayuda de la clase Sistema 4⟩
    ⟨Inicialización de la clase Sistema 5⟩
    ⟨Métodos de la clase Sistema para que actue como si fuera una lista 6a⟩
    ⟨Operador selector por la derecha para la clase Sistema 15⟩
    ⟨Producto de un Sistema por un Vector o una Matrix a su derecha 79⟩
    ⟨Transformaciones elementales de los elementos de un Sistema 38b⟩
    ⟨Métodos de representación de la clase Sistema 78a⟩
```

This code is used in chunk 41.

Defines:

`Sistema`, used in chunks 4–6, 8, 9a, 11, 12, 14, 15, 38, 61a, 62, 65, 66, 68b, 69a, 72, 78, 79, 81c, 89, 90, and 92a.

El resto de métodos se describen en secciones posteriores (detrás del nombre de cada trozo de código aparece el número de página donde encontrarlo).

1.2 La clase Vector

En las notas de la asignatura se dice que

Un *vector* de $\mathbb{R}^n$ es un "sistema" de n números reales;

y dicho sistema se muestra entre paréntesis, bien en forma de fila

$$\boldsymbol{v} = (v_1, \ldots, v_n)$$

o bien en forma de columna

$$\boldsymbol{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}$$

Python no posee objetos que sean "vectores". Necesitamos crearlos definiendo una nueva *clase*. El texto de ayuda de la clase `Vector` es auto-explicativo y será lo que Python nos muestre cuando tecleemos `help(Vector)`:

8 ⟨*Texto de ayuda de la clase* `Vector` 8⟩≡

```
"""Clase Vector

Un Vector es una secuencia finita de números. Se puede instanciar con
una lista, tupla o Sistema de números. Si se instancia con un Vector se
crea una copia del mismo. El atributo 'rpr' indica al entorno Jupyter
si el vector debe ser escrito como fila o como columna.

Parámetros:
    sis (list, tuple, Sistema, Vector) : Sistema de números. Debe ser
        una lista, o tupla de números, o bien otro Vector o Sistema.
    rpr (str) : Representación en Jupyter ('columna' por defecto).
        Indica la forma de representar el Vector en Jupyter. Si
        rpr='fila' se representa en forma de fila. En caso contrario se
        representa en forma de columna.

Atributos:
    sis   (Sistema): sistema de números almacenado.
    n     (int)    : número de elementos del sistema.
    rpr   (str)    : modo de representación en Jupyter.

Ejemplos:
>>> # Instanciación a partir de una lista, tupla o Sistema de números
>>> Vector( [1,2,3] )           # con lista
>>> Vector( (1,2,3) )           # con tupla
>>> Vector( Sistema( [1,2,3] ) )# con Sistema

Vector([1,2,3])
>>> # Crear un Vector a partir de otro Vector
>>> Vector( Vector([1,2,3]) )

Vector([1,2,3])
"""
```

This code is used in chunk 9b.
Uses `Sistema` 7 and `Vector` 9b.

1.2.1 Implementación de los vectores en la clase `Vector`

Método de inicialización Comenzamos la clase con el método de inicio: `def __init__(self, ... )`.

- La clase `Vector` usará dos argumentos (o parámetros). Al primero lo llamaremos `data` y podrá ser una lista, tupla o `Sistema` de números, o bien, otro `Vector`. El segundo argumento (`rpr`) nos permitirá indicar si queremos que el entorno Jupyter Notebook represente el vector en forma horizontal o en vertical. Si no se indica nada, se asumirá que la representación del vector es en vertical (`rpr='columna'`).

- Añadimos un breve texto de ayuda sobre el método `__init__` que Python mostrará con: `help Vector.__init__`

- Si `data` no es una lista, tupla, `Sistema` o `Vector` se devuelve un mensaje de error.

- Se definen tres atributos para la clase `Vector`: los atributos `sis`, `rpr` y `n`.
  - El atributo `self.sis` contiene el `Sistema` de números que constituye el vector... por tanto ***¡ya hemos traducido al lenguaje Python la definición de vector!***
  - El atributo `self.n` guarda el número de elementos del Sistema; y `self.rpr` indica si el vector ha de ser representado como fila o como columna en el entorno Jupyter (en columna por defecto).

9a ⟨*Inicialización de la clase* `Vector` 9a⟩≡

```
def __init__(self, data, rpr='columna'):
    """Inicializa Vector con una lista, tupla, Sistema o Vector"""
    if not isinstance(data, (list, tuple, Vector, Sistema) ):
        raise ValueError(' Argumento debe ser una lista, tupla, Sistema o Vector ')
    self.sis  =  Sistema(data)
    self.n    =  len (self.sis)
    self.rpr  =  rpr
```

This code is used in chunk 9b.
Uses `Sistema` 7 and `Vector` 9b.

La clase `Vector` junto con el listado de sus métodos aparece en el siguiente recuadro:

9b ⟨*Definición de la clase* `Vector` 9b⟩≡

```
class Vector:
    ⟨Texto de ayuda de la clase Vector 8⟩
    ⟨Inicialización de la clase Vector 9a⟩
    ⟨Operador selector por la derecha para la clase Vector 16b⟩
    ⟨Operador selector por la izquierda para la clase Vector 17b⟩
    ⟨Suma de Vectores 22b⟩
    ⟨Producto de un Vector por un escalar a su izquierda 23b⟩
    ⟨Producto de un Vector por un escalar, Vector, o Matrix a su derecha 25a⟩
    ⟨Definición de la igualdad entre Vectores 25b⟩
    ⟨Reverso de un Vector 80b⟩
    ⟨Opuesto de un Vector 81a⟩
    ⟨Comprobación de que un Vector es nulo 81b⟩
    ⟨Representación de la clase Vector 80a⟩
```

This code is used in chunk 41.
Defines:

`Vector`, used in chunks 4, 5, 8, 9a, 11, 12, 14, 16, 17c, 19–26, 28, 29a, 43, 61b, 69, 70, 72a, and 78–83.

En esta sección hemos visto el texto de ayuda y el método de inicialización. El resto de métodos se describen en secciones posteriores (detrás del nombre de cada trozo de código aparece el número de página donde encontrarlo).

Resumen

Los **vectores** son sistemas de números. La clase `Vector` almacena un `Sistema` en su atributo `Vector.sis`:

1. Cuando se instancia un `Vector` con una lista, tupla o `Sistema`, la correspondiente secuencia de números se almacena en el atributo `sis` en forma de `Sistema`.
2. Cuando se instancia un `Vector` con otro `Vector`, se obtiene una copia del `Vector`.
3. Asociados a los `Vector`es hay una serie de métodos que veremos más adelante.

1.3 La clase Matrix

En las notas de la asignatura usamos la siguiente definición

Llamamos *matriz* de $\mathbb{R}^{m\times n}$ a un sistema de n vectores de $\mathbb{R}^m$.

Cuando representamos las matrices, las encerramos entre corchetes

$$\mathbf{A} = [\boldsymbol{v}_1 \dots \boldsymbol{v}_n], \quad \text{donde las } n \text{ columnas } \boldsymbol{v}_i \text{ son vectores de } \mathbb{R}^m.$$

En nuestra implementación crearemos un objeto, `Matrix`, que almacene en uno de sus atributos un `Sistema` de `Vectores` (todos con el mismo número de componentes). Dicho `Sistema` será la lista de "columnas" de la matriz. El texto de ayuda de nuestra clase `Matrix` es auto-explicativo y Python lo mostrará si se teclea `help(Matrix)`.

11 ⟨*Texto de ayuda de la clase* Matrix 11⟩≡

```
"""Clase Matrix

Es un Sistema de Vectores con el mismo número de componentes. Una Matrix
se puede construir con una lista, tupla o Sistema de: Vectores con el
mismo número de componentes (serán las columnas de la matriz); una lista,
tupla o Sistema de: listas, tuplas o Sistemas con el mismo número de
componentes (serán las filas de la matriz); una Matrix (el valor devuelto
será una copia de la Matrix); una BlockMatrix (el valor devuelto es la
Matrix que resulta de unir todos los bloques)

Parámetros:
    data (list, tuple, Sistema, Matrix, BlockMatrix): Lista, tupla o
    Sistema de Vectores con el mismo núm. de componentes (columnas); o
    lista, tupla o Sistema de listas, tuplas o Sistemas con el mismo núm.
    de componentes (filas); u otra Matrix; o una BlockMatrix.

Atributos:
    sis   (Sistema): Sistema de Vectores (columnas)
    m     (int)    : número de filas de la matriz
    n     (int)    : número de columnas de la matriz

Ejemplos:
>>> # Crea una Matrix a partir de una lista de Vectores
>>> a = Vector( [1,2] )
>>> b = Vector( [1,0] )
>>> c = Vector( [9,2] )
>>> Matrix( [a,b,c] )

Matrix([ Vector([1, 2]); Vector([1, 0]); Vector([9, 2]) ])
>>> # Crea una Matrix a partir de una lista de listas de números
>>> A = Matrix( [ [1,1,9], [2,0,2] ] )
>>> A

Matrix([ Vector([1, 2]); Vector([1, 0]); Vector([9, 2]) ])
>>> # Crea una Matrix a partir de otra Matrix
>>> Matrix( A )

Matrix([ Vector([1, 2]); Vector([1, 0]); Vector([9, 2]) ])
>>> # Crea una Matrix a partir de una BlockMatrix
>>> Matrix( {1}|A|{2} )
```

```
  Matrix([ Vector([1, 2]); Vector([1, 0]); Vector([9, 2]) ])
  """
```

This code is used in chunk 13.
Uses BlockMatrix 92b, Matrix 13, Sistema 7, and Vector 9b.

1.3.1 Implementación de las matrices en la clase Matrix

Método de inicialización Comenzamos la clase con el método de inicio: `def __init__(self, sis)`.

- `Matrix` se instancia con el argumento `data`, que podrá ser *una lista, tupla o* `Sistema` *de* `Vector`*es con el mismo número de componentes (las columnas)*; pero que también se podrá ser una lista, tupla o `Sistema` de listas, tuplas o `Sistemas` con el mismo número de componentes (las filas), o una `BlockMatrix`, o bien otra `Matrix`.
- Añadimos un breve texto de ayuda del método `__init__`
- Guardamos la `lista` correspondiente al `Sistema` generado con `data`.
- El atributo `self.sis` guarda el `Sistema` de `Vector`es (que corresponden a la lista de columnas de la matriz).

  El modo de generar dicho `Sistema` difiere en función de qué tipo de objeto es el argumento `data`:

  - cuando `data` es tal que `lista` solo contiene `Vector`es con el mismo número de componentes, entonces `self.sis` es el `Sistema` compuesto por dicha lista de `Vector`es. (Esto pasa cuando `data` es una lista, tupla o `Sistema` de `Vector`es con el mismo número de componentes, o bien cuando `data` es una `Matrix`)
  - cuando `data` es una `BlockMatrix`, entonces `sis` guarda el `Sistema` compuesto por las columnas de la `Matrix` resultante de unificar los bloques en una única matriz.
  - cuando `data` es tal que `lista` es una `list`, `tuple` o `Sistema` de `list`s, `tuple`s o `Sistema`s con el mismo número de componentes; entonces se interpreta que `lista` es la "lista de filas" de la matriz, y se reconstruye la lista de columnas correspondiente a dicha matriz.

  De esta manera el atributo `self.sis` contendrá el `Sistema` de `Vector`es columna que constituye la matriz... por tanto ***¡ya hemos traducido al lenguaje Python la definición de matriz!***

- Por conveniencia definimos un par de atributos más. El atributo `self.m` guarda el número de filas de la matriz, y `self.n` guarda el número de columnas.

12 ⟨*Inicialización de la clase* Matrix 12⟩≡

```
  def __init__(self, data):
      """Inicializa una Matrix"""
      lista = Sistema(data).lista

      if isinstance(lista[0], Vector):
          ⟨Verificación de que todas las columnas de la matriz tienen la misma longitud 82b⟩
          self.sis  =  Sistema(data)
      ⟨Creación del atributo sis cuando no tenemos una lista de Vectores 81c⟩

      self.m  =  self.sis.lista[0].n
      self.n  =  len(self.sis)
```

This code is used in chunk 13.
Uses Matrix 13, Sistema 7, and Vector 9b.

La clase `Matrix` junto con el listado de sus métodos aparece en el siguiente recuadro:

13 ⟨*Definición de la clase* `Matrix` 13⟩≡

```
class Matrix:
        ⟨Texto de ayuda de la clase Matrix 11⟩
        ⟨Inicialización de la clase Matrix 12⟩
        ⟨Operador selector por la derecha para la clase Matrix 18⟩
        ⟨Operador transposición para la clase Matrix 19b⟩
        ⟨Operador selector por la izquierda para la clase Matrix 21⟩
        ⟨Suma de Matrix 26b⟩
        ⟨Producto de una Matrix por un escalar a su izquierda 27b⟩
        ⟨Producto de una Matrix por un escalar, un vector o una matriz a su derecha 29a⟩
        ⟨Definición de la igualdad entre dos Matrix 30⟩
        ⟨Transformaciones elementales de las columnas de una Matrix 39b⟩
        ⟨Transformaciones elementales de las filas de una Matrix 40⟩
        ⟨Potencia de una Matrix 29b⟩
        ⟨Reverso de una Matrix 83a⟩
        ⟨Opuesto de una Matrix 83b⟩
        ⟨Representación de la clase Matrix 82c⟩
```

This code is used in chunk 41.

Defines:

`Matrix`, used in chunks 4, 5, 11, 12, 17–21, 24–29, 32, 34c, 35, 39, 50–52, 54–63, 65–68, 70a, 72a, 73, 78b, 79, 82–84, 88, 90, 91, and 95.

En esta sección hemos visto el texto de ayuda y el método de inicialización. El resto de métodos se describen en secciones posteriores.

Resumen

Las **matrices** son sistemas de vectores (dichos vectores son sus columnas). La clase `Matrix` almacena un `Sistema` de `Vector`es en el atributo `sis` de tres modos distintos (el código de los dos últimos se puede consultar en el Capítulo 4 de este documento):

1. Cuando se instancia con una lista, tupla o `Sistema` de `Vector`es, en el atributo `sis` se almacena el `Sistema` formado por dichos `Vector`es. *Esta es la forma de crear una matriz a partir de sus columnas.*

2. Para mayor comodidad, cuando se instancia una `Matrix` con una lista, tupla o `Sistema` de listas, tuplas o `Sistema`s, se interpreta que son las filas de la matriz. Consecuentemente, se dan los pasos para describir dicha matriz como un `Sistema` de columnas, que se almacena en el atributo `sis`. (Esta forma de instanciar una `Matrix` se usará para programar la **transposición** en la Página 18).

3. Cuando se instancia con otra `Matrix`, se copia el atributo `sis` de dicha `Matrix`.

4. Cuando se instancia con una `BlockMatrix`, se unifican los bloques en una sola matriz, cuyo `Sistema` de columnas es guardado en el atributo `sis`.

5. Asociados a las `Matrix` hay una serie de métodos que veremos más adelante.

Por tanto,

- `Vector` guarda un `Sistema` de números en su atributo `sis`
- `Matrix` guarda un `Sistema` de `Vector`es en su atributo `sis`; así pues:

¡Hemos implementado en Python los vectores y matrices tal y como se definen en las notas de la asignatura!

. . . vayamos con el operador selector. . . que nos permitirá definir las operaciones de suma, producto, etc.. . .

1.4 Operadores selectores

Notación en Mates 2

- Si $\boldsymbol{v} = (v_1, \ldots, v_n)$ entonces ${}_{i|}\boldsymbol{v} = \boldsymbol{v}_{|i} = v_i$ para todo $i \in \{1, \ldots, n\}$.
- Si $\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}$ entonces $\begin{cases} \mathbf{A}_{|j} = \begin{pmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{pmatrix} \text{ para todo } j \in \{1, \ldots, m\} \\ {}_{i|}\mathbf{A} = (a_{i1}, \ldots, a_{im}) \text{ para todo } i \in \{1, \ldots, n\} \end{cases}$.

Pero puestos a seleccionar, aprovechemos la notación para seleccionar más de un elemento:

Notación en Mates 2

- ${}_{(i_1,\ldots,i_r)|}\boldsymbol{v} = \Big(\boldsymbol{v}_{i_1}, \ldots, \boldsymbol{v}_{i_r}\Big) = \boldsymbol{v}_{|(i_1,\ldots,i_r)}$ (es un vector formado por elementos de $\boldsymbol{v}$)
- ${}_{(i_1,\ldots,i_r)|}\mathbf{A} = \Big[{}_{i_1|}\mathbf{A} \ \ldots \ {}_{i_r|}\mathbf{A}\Big]^{\mathsf{T}}$ (es una matriz cuyas filas son filas de $\mathbf{A}$)
- $\mathbf{A}_{|(j_1,\ldots,j_r)} = \Big[\mathbf{A}_{|j_1} \ \ldots \ \mathbf{A}_{|j_r}\Big]$ (es una matriz formada por columnas de $\mathbf{A}$)

Queremos manejar una notación similar en Python, así que tenemos que definir el operador selector. Y queremos hacerlo con un método de Python que tenga asociado un símbolo que permita invocar el método de selección

Tutorial previo en un Jupyter notebook

Si no recuerda a qué me estoy refiriendo con los símbolos asociados a métodos, repase de nuevo la sección "Métodos especiales con símbolos asociados" del Notebook referente a **"Clases"** en la carpeta "TutorialPython" en `https://mybinder.org/v2/gh/mbujosab/LibreriaDePythonParaMates2/master`

Como los métodos `__or__` y `__ror__` tienen asociados la barra vertical a derecha e izquierda, usaremos el siguiente convenio:

Mates II	Python
$\boldsymbol{v}_{\mid i}$	`v|i`
${}_{i\mid}\boldsymbol{v}$	`i|v`
$\mathbf{A}_{\mid j}$	`A|j`
${}_{i\mid}\mathbf{A}$	`i|A`

1.4.1 Operador selector por la derecha para la clase `Sistema`.

Tal como se hace en el Tema 2 de las notas de la asignatura, vamos a permitir seleccionar elementos del `Sistema` con el operador "|" actuando por la derecha (solo por la derecha).

14 ⟨*Texto de ayuda para el operador selector por la derecha para la clase* `Sistema` 14⟩≡

```
"""
Extrae el i-ésimo componente del Sistema; o crea un Sistema con los
elementos indicados (los índices comienzan por la posición 1)
```

```
Parámetros:
    j (int, list, tuple): Índice (o lista de índices) de las columnas a
          seleccionar

Resultado:
          ?: Cuando j es int, devuelve el elemento j-ésimo del Sistema.
    Sistema: Cuando j es list o tuple, devuelve el Sistema formado por
          los elementos indicados en la lista o tupla de índices.

Ejemplos:
>>> # Extrae el j-ésimo elemento del Sistema
>>> Sistema([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | 2

Vector([0, 2])
>>> # Sistema formado por los elementos indicados en la lista (o tupla)
>>> Sistema([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | [2,1]
>>> Sistema([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | (2,1)

[Vector([0, 2]); Vector([1, 0])]
"""
```

This code is used in chunk 15.
Uses `Sistema` 7 and `Vector` 9b.

Implementación del operador selector por la derecha para la clase `Sistema`.

Cuando el argumento `i` es un número entero (`int`), seleccionamos el correspondiente elemento del atributo `lista` del `Sistema` (recuerde que en Python los índices de objetos iterables comienzan en cero, por lo que para seleccionar el elemento i-ésimo de `lista`, escribimos `lista[i-1]`; así `a|1` debe seleccionar el primer elemento del atributo `lista`, es decir `a.lista[0]`).

Una vez hemos definido el operador "`|`" cuando el argumento `i` es un entero (`int`), podemos usar el método (`self|a`) para definir el operador cuando el argumento `i` es una lista o tupla `(list,tuple)` de índices (y así generar un `Sistema` con la lista de componentes indicadas).

15 ⟨*Operador selector por la derecha para la clase* `Sistema` 15⟩≡

```
def __or__(self,i):
    ⟨Texto de ayuda para el operador selector por la derecha para la clase Sistema 14⟩
    if isinstance(i,int):
        return self.lista[i-1]

    elif isinstance(i, (list,tuple) ):
        return Sistema ([ (self|a) for a in i ])
```

This code is used in chunk 7.
Uses `Sistema` 7.

1.4.2 Operador selector por la derecha para la clase Vector.

16a ⟨*Texto de ayuda para el operador selector por la derecha para la clase* Vector 16a⟩≡
```
"""Selector por la derecha

Extrae la i-ésima componente o genera un nuevo Vector con las componentes
indicadas en una lista o tupla (los índices comienzan por la posición 1).

Parámetros:
    i (int, list, tuple): Índice (o lista de índices) de los elementos
        a seleccionar.
Resultado:
    número: Cuando i es int, devuelve el componente i-ésimo del Vector.
    Vector: Cuando i es list o tuple, devuelve el Vector formado por los
        componentes indicados en la lista o tupla de índices.
Ejemplos:
>>> # Selección de una componente
>>> Vector([10,20,30]) | 2

20
>>> # Creación de un sub-vector a partir de una lista o tupla de índices
>>> Vector([10,20,30]) | [2,1,2]
>>> Vector([10,20,30]) | (2,1,2)

Vector([20, 10, 20])
"""
```
This code is used in chunk 16b.
Uses Vector 9b.

Implementación del operador selector por la derecha para la clase Vector**.**

Como el objeto `Vector` es un `Sistema` de números, usaremos el operador selector sobre el argumento `sis` del `Vector`. Cuando el argumento `i` es un número entero (`int`), seleccionamos el correspondiente elemento del atributo `sis` del `Vector`; así `a|1` debe seleccionar el primer elemento del `Sistema` guardado en el atributo `sis`.

Una vez hemos definido el operador "`|`" cuando el argumento `i` es un entero (`int`), podemos usar el método (`self|a`) para definir el operador cuando el argumento `i` es una lista o tupla `(list,tuple)` de índices (y así generar un `Vector` con la lista de componentes indicadas).

16b ⟨*Operador selector por la derecha para la clase* Vector 16b⟩≡
```
def __or__(self,i):
    ⟨Texto de ayuda para el operador selector por la derecha para la clase Vector 16a⟩
    if isinstance(i,int):
        return self.sis|i
    elif isinstance(i, (list,tuple) ):
        return Vector ([ (self|a) for a in i ])
```
This code is used in chunk 9b.
Uses Vector 9b.

1.4.3 Operador selector por la izquierda para la clase Vector.

17a ⟨*Texto de ayuda para el operador selector por la izquierda para la clase* `Vector` 17a⟩≡
```
"""Selector por la izquierda

Hace lo mismo que el método __or__ solo que operando por la izquierda
"""
```
This code is used in chunk 17b.

Implementación del operador selector por la izquierda para la clase Vector.

Como hace lo mismo que el selector por la derecha, basta con llamar al selector por la derecha: `self|i`

17b ⟨*Operador selector por la izquierda para la clase* `Vector` 17b⟩≡
```
def __ror__(self,i):
    ⟨Texto de ayuda para el operador selector por la izquierda para la clase Vector 17a⟩
    return self | i
```
This code is used in chunk 9b.

1.4.4 Operador selector por la derecha para la clase Matrix.

17c ⟨*Texto de ayuda para el operador selector por la derecha para la clase* `Matrix` 17c⟩≡
```
"""
Extrae la i-ésima columna de Matrix; o crea una Matrix con las columnas
indicadas; o crea una BlockMatrix particionando una Matrix por las
columnas indicadas (los índices comienzan por la posición 1)

Parámetros:
    j (int, list, tuple): Índice (o lista de índices) de las columnas a
          seleccionar
      (set): Conjunto de índices de las columnas por donde particionar

Resultado:
    Vector: Cuando j es int, devuelve la columna j-ésima de Matrix.
    Matrix: Cuando j es list o tuple, devuelve la Matrix formada por las
        columnas indicadas en la lista o tupla de índices.
    BlockMatrix: Si j es un set, devuelve la BlockMatrix resultante de
        particionar la matriz por las columnas indicadas en el conjunto

Ejemplos:
>>> # Extrae la j-ésima columna la matriz
>>> Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | 2
```

```
  Vector([0, 2])
  >>> # Matrix formada por Vectores columna indicados en la lista (o tupla)
  >>> Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | [2,1]
  >>> Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | (2,1)

  Matrix( [Vector([0, 2]); Vector([1, 0])] )
  >>> # BlockMatrix correspondiente a la partición por la segunda columna
  >>> Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | {2}

  BlockMatrix( [ [ Matrix([Vector([1, 0]), Vector([0, 2])]);
                   Matrix([Vector([3, 0])]) ] ] )
  """
```

This code is used in chunk 18.
Uses `BlockMatrix` 92b, `Matrix` 13, and `Vector` 9b.

Implementación del operador selector por la derecha para la clase `Matrix`

Como el objeto `Matrix` es un sistema de `Vector`es, el código para el selector por la derecha es idéntico al de la clase `Vector` (*la partición en bloques de columnas de matrices se verá más adelante, en la sección de la clase* `BlockMatrix`).

18 ⟨*Operador selector por la derecha para la clase* `Matrix` 18⟩≡

```
  def __or__(self,j):
      ⟨Texto de ayuda para el operador selector por la derecha para la clase Matrix 17c⟩
      if isinstance(j,int):
          return self.sis|j
      elif isinstance(j, (list,tuple)):
          return Matrix ([ self|a for a in j ])

      ⟨Partición de una matriz por columnas de bloques 93c⟩
```

This code is used in chunk 13.
Uses `Matrix` 13.

1.4.5 Operador transposición de una `Matrix`.

Implementar el operador selector por la izquierda es algo más complicado que en el caso de los `Vector`es, pues ahora no es lo mismo operar por la derecha que por la izquierda. Como paso intermedio definiremos el operador transposición, que después usaremos para definir el operador selector por la izquierda (selección de filas).

Notación en Mates 2

Denotamos la *transpuesta* de $\mathbf{A}$ con: $\mathbf{A}^\intercal$; y es la matriz tal que $\left(\mathbf{A}^\intercal\right)_{|j} = {}_{j|}\mathbf{A}$; $\quad j = 1:n$.

19a
```
⟨Texto de ayuda para el operador transposición de la clase Matrix 19a⟩≡
  """
  Devuelve la traspuesta de una matriz.

  Ejemplo:
  >>> ~Matrix([Vector([1]), Vector([2]), Vector([3])])

  Matrix([Vector([1, 2, 3])])
  """
```
This code is used in chunk 19b.
Uses Matrix 13 and Vector 9b.

Implementación del operador transposición.

Desgraciadamente Python no dispone del símbolo " $^\intercal$". Así que hemos de usar un símbolo distinto para indicar transposición. Y además no tenemos muchas opciones ya que el conjunto de símbolos asociados a métodos especiales es muy limitado.

Tutorial previo en un Jupyter notebook

Si no recuerda a qué me estoy refiriendo con los símbolos asociados a métodos, repase de nuevo la sección "Métodos especiales con símbolos asociados" del Notebook referente a **"Clases"** en la carpeta "TutorialPython" en
`https://mybinder.org/v2/gh/mbujosab/LibreriaDePythonParaMates2/master`

Para implementar la transposición haremos uso del método `__invert__`, que tiene asociado el símbolo del la tilde "~", símbolo que además deberemos colocar a la izquierda de la matriz.

Mates II	Python
$\mathbf{A}^\intercal$	`~A`

Ahora recuerde que con la segunda forma de instanciar una `Matrix` (véase el resumen de la página 13) creamos una matriz a partir de la lista de sus filas. Aprovechando esta forma de instanciar podemos construir fácilmente el operador trasposición. Basta instanciar `Matrix` con la lista de `Sistema`s correspondientes a los n `Vector`es columna. (Recuerde que `range(1,self.m+1)` recorre los números: $1, 2, \dots, m$).

19b
```
⟨Operador transposición para la clase Matrix 19b⟩≡
  def __invert__(self):
      ⟨Texto de ayuda para el operador transposición de la clase Matrix 19a⟩
      return Matrix ([ (self|j).sis for j in range(1,self.n+1) ])
```
This code is used in chunk 13.
Uses Matrix 13.

1.4.6 Operador selector por la izquierda para la clase Matrix.

20 ⟨*Texto de ayuda para el operador selector por la izquierda para la clase* Matrix 20⟩≡

```
"""Operador selector por la izquierda

Extrae la i-ésima fila de Matrix; o crea una Matrix con las filas
indicadas; o crea una BlockMatrix particionando una Matrix por las filas
indicadas (los índices comienzan por la posición 1)

Parámetros:
    i (int, list, tuple): Índice (o lista de índices) de las filas a
         seleccionar
      (set): Conjunto de índices de las filas por donde particionar

Resultado:
    Vector: Cuando i es int, devuelve la fila i-ésima de Matrix.
    Matrix: Cuando i es list o tuple, devuelve la Matrix cuyas filas son
        las indicadas en la lista de índices.
    BlockMatrix: Cuando i es un set, devuelve la BlockMatrix resultante
        de particionar la matriz por las filas indicadas en el conjunto

Ejemplos:
>>> # Extrae la j-ésima fila de la matriz
>>> 2 | Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])])

Vector([0, 2, 0])
>>> # Matrix formada por Vectores fila indicados en la lista (o tupla)
>>> [1,1] | Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])])
>>> (1,1) | Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])])

Matrix([Vector([1, 1]), Vector([0, 0]), Vector([3, 3])])
>>> # BlockMatrix correspondiente a la partición por la primera fila
>>> {1} | Matrix([Vector([1,0]), Vector([0,2])])

BlockMatrix( [ [Matrix([Vector([1]),Vector([0])])],
               [Matrix([Vector([0]),Vector([2])])] ] )
"""
```

This code is used in chunk 21.
Uses BlockMatrix 92b, Matrix 13, and Vector 9b.

Implementación del operador por la izquierda para la clase Matrix.

Usando el operador selector de columnas y la transposición, es inmediato definir un operador selector de *filas*. . . ¡que son las columnas de la matriz transpuesta!

`(~self)|j`

(para recordar que se ha obtenido una fila de la matriz, representamos el Vector en horizontal: `rpr='fila'`)

Una vez definido el operador por la izquierda, podemos usarlo repetidas veces el procedimiento `(a|self)` para crear una Matrix con las filas indicadas en una lista o tupla de índices.
(*la partición en bloques de filas de matrices se verá más adelante, en la sección de la clase* BlockMatrix).

21 ⟨*Operador selector por la izquierda para la clase* Matrix 21⟩≡

```
  def __ror__(self,i):
      ⟨Texto de ayuda para el operador selector por la izquierda para la clase Matrix 20⟩
      if isinstance(i,int):
          return Vector ( (~self)|i, rpr='fila' )

      elif isinstance(i, (list,tuple)):
          return Matrix ( [ (a|self).sis  for a in i ] )

      ⟨Partición de una matriz por filas de bloques 93b⟩
```

This code is used in chunk 13.
Uses Matrix 13 and Vector 9b.

Resumen

¡Ahora también hemos implementado en Python el operador "|" (tanto por la derecha como por la izquierda tal) y como se define en las notas de la asignatura!

Ya estamos listos para definir el resto de operaciones con vectores y matrices. . .

1.5 Operaciones con vectores y matrices

Una vez definidas las clases `Vector` y `Matrix` junto con los respectivos operadores selectores "`|`", ya podemos definir las operaciones de suma y producto. Fíjese que las definiciones de las operaciones en Python (usando el operador "`|`") son idénticas a las empleadas en las notas de la asignatura:

1.5.1 Suma de `Vectores`

En las notas de la asignatura hemos definido la suma de dos vectores de $\mathbb{R}^n$ como el vector tal que

$$\boxed{(\boldsymbol{a}+\boldsymbol{b})_{|i} = \boldsymbol{a}_{|i} + \boldsymbol{b}_{|i}} \quad \text{para} \quad i = 1:n.$$

Usando el operador selector podemos "literalmente" transcribir esta definición

```
Vector ([ (self|i) + (other|i) for i in range(1,self.n+1) ])
```

donde `self` es el vector, `other` es otro vector y `range(1,self.n+1)` es el rango de valores: $1:n$.

22a ⟨*Texto de ayuda para el operador suma en la clase* `Vector` 22a⟩≡

```
"""Devuelve el Vector resultante de sumar dos Vectores

Parámetros:
    other (Vector): Otro vector con el mismo número de elementos

Ejemplo
>>> Vector([10, 20, 30]) + Vector([-1, 1, 1])

Vector([9, 21, 31])
"""
```

This code is used in chunk 22b.
Uses `Vector` 9b.

Implementación

22b ⟨*Suma de* `Vector`*es* 22b⟩≡

```
def __add__(self, other):
    ⟨Texto de ayuda para el operador suma en la clase Vector 22a⟩
    if not isinstance(other, Vector) or self.n != other.n:
        raise ValueError\
('A un Vector solo se le puede sumar otro Vector con el mismo número de componentes')

    return Vector ([ (self|i) + (other|i) for i in range(1,self.n+1) ])
```

This code is used in chunk 9b.
Uses `Vector` 9b.

1.5.2 Producto de un Vector por un escalar a su izquierda

En las notas hemos definido el producto de $\boldsymbol{a}$ por un escalar x a su izquierda como el vector tal que

$$\boxed{(x\boldsymbol{a})_{|i} = x(\boldsymbol{a}_{|i})} \quad \text{para} \quad i = 1:n.$$

cuya transcripción será

```
Vector ([ x*(self|i) for i in range(1,self.n+1) ])
```

donde `self` es el vector y `x` es un número entero, de coma flotante o una fracción (`int, float, Fraction`).

23a ⟨*Texto de ayuda para el operador producto por la izquierda en la clase* `Vector` 23a⟩≡

```
"""Multiplica un Vector por un número a su izquierda

Parámetros:
    x (int, float o Fraction): Número por el que se multiplica

Resultado:
    Vector: Cuando x es int, float o Fraction, devuelve el Vector que
        resulta de multiplicar cada componente por x
Ejemplo:
>>> 3 * Vector([10, 20, 30])

Vector([30, 60, 90])
"""
```

This code is used in chunk 23b.
Uses `Vector` 9b.

Implementación

23b ⟨*Producto de un* `Vector` *por un escalar a su izquierda* 23b⟩≡

```
def __rmul__(self, x):
    ⟨Texto de ayuda para el operador producto por la izquierda en la clase Vector 23a⟩
    if isinstance(x, (int, float, Fraction)):
        return Vector ([ x*(self|i) for i in range(1,self.n+1) ])
```

This code is used in chunk 9b.
Uses `Vector` 9b.

1.5.3 Producto de un Vector por un escalar, un Vector o una Matrix a su derecha

- En las notas se acepta que el producto de un vector $\boldsymbol{a}$ por un escalar es conmutativo. Por tanto,

$$\boxed{\boldsymbol{a}x = x\boldsymbol{a}}$$

cuya transcripción será

```
x * self
```

donde `self` es el vector y `x` es un número entero, o de coma flotante o una fracción (`int, float, Fraction`).

- El *producto punto* (o producto escalar usual en $\mathbb{R}^n$) de dos vectores $\boldsymbol{a}$ y $\boldsymbol{x}$ en $\mathbb{R}^n$ es

$$\boxed{\boldsymbol{a}\cdot\boldsymbol{x} \;=\; a_1x_1+\cdots+a_nx_n \;=\; \sum_{i=1}^n a_ix_i} \quad \text{para} \quad i=1:n.$$

cuya transcripción será

```
sum([ (self|i)*(x|i) for i in range(1,self.n+1) ])
```

donde `self` es el vector $\boldsymbol{a}$ y `x` es otro vector (`Vector`).

- El producto de un vector $\boldsymbol{a}$ de $\mathbb{R}^n$ por una matriz $\mathbf{X}$ con n filas es

$$\boxed{\boldsymbol{a}\mathbf{X} \;=\; \mathbf{X}^{\intercal}\boldsymbol{a}}$$

cuya transcripción será

```
(~x) * self
```

donde `self` es el vector y `x` es una matriz (`Matrix`). Para recordar que es una combinación de filas, la representamos como fila (la definición del producto de una `Matrix` por un `Vector` a su derecha se verá más adelante.)

24 ⟨*Texto de ayuda para el operador producto por la derecha en la clase* `Vector` 24⟩≡

```
"""Multiplica un Vector por un número, Matrix o Vector a su derecha.

Parámetros:
    x (int, float o Fraction): Número por el que se multiplica
      (Vector): Vector con el mismo número de componentes.
      (Matrix): Matrix con tantas filas como componentes tiene el Vector

Resultado:
    Vector: Cuando x es int, float o Fraction, devuelve el Vector que
       resulta de multiplicar cada componente por x.
    Número: Cuando x es Vector, devuelve el producto punto entre vectores
       (producto escalar usual en R^n)
    Vector: Cuando x es Matrix, devuelve la combinación lineal de las
       filas de x (el Vector contiene los coeficientes de la combinación)

Ejemplos:
>>> Vector([10, 20, 30]) * 3

Vector([30, 60, 90])
>>> Vector([10, 20, 30]) * Vector([1, 1, 1])

60
>>> a = Vector([1, 1])
>>> B = Matrix([Vector([1, 2]), Vector([1, 0]), Vector([9, 2])])
>>> a * B

Vector([3, 1, 11])
"""
```

This code is used in chunk 25a.
Uses `Matrix` 13 and `Vector` 9b.

Implementación

25a ⟨*Producto de un* Vector *por un escalar,* Vector*, o* Matrix *a su derecha* 25a⟩≡

```
def __mul__(self, x):
    ⟨Texto de ayuda para el operador producto por la derecha en la clase Vector 24⟩
    if isinstance(x, (int, float, Fraction)):
        return x*self

    elif isinstance(x, Vector):
        if self.n != x.n:
            raise ValueError('Vectores con distinto número de componentes')
        return sum([ (self|i)*(x|i) for i in range(1,self.n+1) ])

    elif isinstance(x, Matrix):
        if self.n != x.m:
            raise ValueError('Vector y Matrix incompatibles')
        return Vector( (~x)*self, rpr='fila' )
```

This code is used in chunk 9b.
Uses **Matrix** 13 and **Vector** 9b.

1.5.4 Igualdad entre vectores

Dos vectores son iguales cuando lo son los sistemas de números correspondientes a ambos vectores.

25b ⟨*Definición de la igualdad entre* Vector*es* 25b⟩≡

```
def __eq__(self, other):
    """Indica si es cierto que dos vectores son iguales"""
    return self.sis == other.sis

def __ne__(self, other):
    """Indica si es cierto que dos vectores son distintos"""
    return self.sis != other.sis
```

This code is used in chunk 9b.

1.5.5 Suma de matrices

En las notas de la asignatura hemos definido la suma de matrices como la matriz tal que

$$\boxed{\left(\mathbf{A}+\mathbf{B}\right)_{|j} = \mathbf{A}_{|j} + \mathbf{B}_{|j}} \quad \text{para} \quad i = 1:n.$$

de nuevo, usando el operador selector podemos transcribir literalmente esta definición

```
Matrix ([ (self|i) + (other|i) for i in range(1,self.n+1) ])
```

donde `self` es la matriz y `other` es otra matriz.

26a ⟨*Texto de ayuda para el operador suma en la clase* Matrix 26a⟩≡
```
"""Devuelve la Matrix resultante de sumar dos Matrices

Parámetros:
    other (Matrix): Otra Matrix con el mismo número de filas y columnas

Ejemplo:
>>> A = Matrix( [Vector([1,0]), Vector([0,1])] )
>>> B = Matrix( [Vector([0,2]), Vector([2,0])] )
>>> A + B

Matrix( [Vector([1,2]), Vector([2,1])] )
"""
```
This code is used in chunk 26b.
Uses Matrix 13 and Vector 9b.

Implementación

26b ⟨*Suma de* Matrix 26b⟩≡
```
def __add__(self, other):
    ⟨Texto de ayuda para el operador suma en la clase Matrix 26a⟩
    if not isinstance(other,Matrix) or (self.m,self.n) != (other.m,other.n):
        raise ValueError('A una Matrix solo se le puede sumar otra del mismo orden')

    return Matrix ([ (self|i) + (other|i) for i in range(1,self.n+1) ])
```
This code is used in chunk 13.
Uses Matrix 13.

1.5.6 Producto de una `Matrix` por un escalar a su izquierda

En las notas hemos definido

- El producto de $\mathbf{A}$ por un escalar x a su izquierda como la matriz tal que

$$\boxed{\left(x\mathbf{A}\right)_{|j} = x\left(\mathbf{A}_{|j}\right)} \quad \text{para} \quad i = 1:n.$$

  cuya transcripción será:

```
Matrix ([ x*(self|i) for i in range(1,self.n+1) ])
```

  donde `self` es la matriz y `x` es un número entero, de coma flotante o una fracción (`int, float, Fraction`).

27a ⟨*Texto de ayuda para el operador producto por la izquierda en la clase* Matrix 27a⟩≡

```
"""Multiplica una Matrix por un número a su izquierda.

Parámetros:
    x (int, float o Fraction): Número por el que se multiplica

Resultado:
    Matrix: Devuelve el múltiplo de la Matrix

Ejemplo:
>>> 10 * Matrix([[1,2],[3,4]])

Matrix([[10,20], [30,40]])
"""
```

This code is used in chunk 27b.
Uses Matrix 13.

Implementación

27b ⟨*Producto de una* Matrix *por un escalar a su izquierda* 27b⟩≡

```
def __rmul__(self,x):
    ⟨Texto de ayuda para el operador producto por la izquierda en la clase Matrix 27a⟩
    if isinstance(x, (int, float, Fraction)):
        return Matrix ([ x*(self|i) for i in range(1,self.n+1) ])
```

This code is used in chunk 13.
Uses Matrix 13.

1.5.7 Producto de una `Matrix` por un escalar, un `Vector` o una `Matrix` a su derecha

- En las notas se acepta que el producto de una `Matrix` por un escalar es conmutativo. Por tanto,

  $$\boxed{\mathbf{A}x = x\mathbf{A}}$$

  cuya transcripción será

  `x * self`

  donde `self` es la matriz y `x` es un número entero, o de coma flotante o una fracción (`int, float, Fraction`).

- El producto de $\underset{m\times n}{\mathbf{A}}$ por un vector $\boldsymbol{x}$ de $\mathbb{R}^n$ a su derecha se define como

  $$\boxed{\mathbf{A}\boldsymbol{x} \;=\; x_1\mathbf{A}_{|1} + \cdots + x_n\mathbf{A}_{|n} \;=\; \sum_{j=1}^{n} x_j\mathbf{A}_{|j}} \qquad \text{para } j = 1:n.$$

  cuya transcripción será

```
sum([ (x|j)*(self|j) for j in range(1,self.n+1) ])
```

donde `self` es el vector y `x` es otro vector (`Vector`).

- El producto de $\underset{m\times k}{\mathbf{A}}$ por otra matriz $\underset{k\times n}{\mathbf{X}}$ de $\mathbb{R}^n$ a su derecha se define como la matriz tal que

$$\boxed{(\mathbf{AX})_{|j} = \mathbf{A}(\mathbf{X}_{|j})} \qquad \text{para } j = 1:n.$$

cuya transcripción será

```
Matrix( [ self*(x|j) for j in range(1,x.n+1)] )
```

donde `self` es la matriz y `x` es otra matriz (`Matrix`).

28 ⟨*Texto de ayuda para el operador producto por la derecha en la clase* `Matrix` 28⟩≡

```
"""Multiplica una Matrix por un número, Vector o una Matrix a su derecha

Parámetros:
    x (int, float o Fraction): Número por el que se multiplica
      (Vector): Vector con tantos componentes como columnas tiene Matrix
      (Matrix): con tantas filas como columnas tiene la Matrix

Resultado:
    Matrix: Si x es int, float o Fraction, devuelve la Matrix que
       resulta de multiplicar cada columna por x
    Vector: Si x es Vector, devuelve el Vector combinación lineal de las
       columnas de Matrix (los componentes del Vector son los
       coeficientes de la combinación)
    Matrix: Si x es Matrix, devuelve el producto entre las matrices

Ejemplos:
>>> # Producto por un número
>>> Matrix([[1,2],[3,4]]) * 10

Matrix([[10,20],[30,40]])
>>> # Producto por un Vector
>>> Matrix([Vector([1, 3]), Vector([2, 4])]) * Vector([1, 1])

Vector([3, 7])
>>> # Producto por otra Matrix
>>> Matrix([Vector([1, 3]), Vector([2, 4])]) * Matrix([Vector([1,1])]))

Matrix([Vector([3, 7])])
"""
```

This code is used in chunk 29a.
Uses `Matrix` 13 and `Vector` 9b.

Implementación

29a
```
⟨Producto de una Matrix por un escalar, un vector o una matriz a su derecha 29a⟩≡
  def __mul__(self,x):
      ⟨Texto de ayuda para el operador producto por la derecha en la clase Matrix 28⟩
      if isinstance(x, (int, float, Fraction)):
          return x*self

      elif isinstance(x, Vector):
          if self.n != x.n:      raise ValueError('Vector y Matrix incompatibles')
          return sum([(x|j)*(self|j) for j in range(1,self.n+1)], V0(self.m))

      elif isinstance(x, Matrix):
          if self.n != x.m:      raise ValueError('matrices incompatibles')
          return Matrix( [ self*(x|j) for j in range(1,x.n+1)] )
```

This code is used in chunk 13.
Uses Matrix 13, V0 83c, and Vector 9b.

1.5.8 Potencias de una Matrix cuadrada

Ahora podemos calcular la n-ésima potencia de una Matrix cuando n es un entero positivo; basta multiplicar la Matrix por si misma n veces.

Si n es un entero negativo, entonces necesitamos calcular la inversa de la n-ésima potencia. Un método auxiliar calculará dicha inversa si es posible, pero para ello usará el método de eliminación guassiano que se describirá en el Capítulo 2.

29b
```
⟨Potencia de una Matrix 29b⟩≡
  def __pow__(self,n):
      """Calcula potencias de una Matrix (incluida la inversa)"""
      ⟨Método auxiliar que calcula la inversa de una Matrix 59a⟩
      if self.m != self.n:       raise ValueError('Matrix no cuadrada')
      if not isinstance(n,int):  raise ValueError('La potencia no es un entero')

      M = self
      for i in range(1,abs(n)):
          M = M * self

      return MatrixInversa(M) if n < 0 else M
```

This code is used in chunk 13.
Uses Matrix 13.

1.5.9 Igualdad entre matrices

Dos matrices son iguales solo cuando lo son las listas correspondientes a ambas.

30 ⟨*Definición de la igualdad entre dos* `Matrix` 30⟩≡

```
def __eq__(self, other):
    """Indica si es cierto que dos matrices son iguales"""
    return self.sis == other.sis

def __ne__(self, other):
    """Indica si es cierto que dos matrices son distintas"""
    return self.sis != other.sis
```

This code is used in chunk 13.

1.6 La clase transformación elemental `T`

Notación en Mates 2

Si $\mathbf{A}$ es una matriz, consideramos las siguientes transformaciones:

Tipo I: $\underset{[(\lambda)\boldsymbol{i}+\boldsymbol{j}]}{\boldsymbol{\tau}}\mathbf{A}$ suma λ veces la fila i a la fila j $(i \neq j)$; $\mathbf{A}\underset{[(\lambda)\boldsymbol{i}+\boldsymbol{j}]}{\boldsymbol{\tau}}$ lo mismo con las columnas.

Tipo II: $\underset{[(\lambda)\boldsymbol{i}]}{\boldsymbol{\tau}}\mathbf{A}$ multiplica la fila i por $\lambda \neq 0$; y $\mathbf{A}\underset{[(\lambda)\boldsymbol{j}]}{\boldsymbol{\tau}}$ multiplica la columna j por λ.

Intercambio: $\underset{[\boldsymbol{i}\rightleftharpoons\boldsymbol{j}]}{\boldsymbol{\tau}}\mathbf{A}$ intercambia las filas i y j; y $\mathbf{A}\underset{[\boldsymbol{i}\rightleftharpoons\boldsymbol{j}]}{\boldsymbol{\tau}}$ intercambia las columnas.

Comentario sobre la notación. Como una trasformación elemental es el resultado del producto con una matriz elemental, esta notación busca el parecido con la notación del producto matricial:

Al poner la *abreviatura* "$\boldsymbol{\tau}$" de la transformación elemental a derecha es como si multiplicáramos la matriz $\underset{m\times n}{\mathbf{A}}$ por la derecha por la correspondiente matriz elemental

$$\mathbf{A}_{\boldsymbol{\tau}_1} = \mathbf{A}\mathbf{I}_{\boldsymbol{\tau}_1} = \mathbf{A}\mathbf{E}_1 \quad \text{donde} \quad \mathbf{E}_1 = \mathbf{I}_{\boldsymbol{\tau}_1} \quad \text{y donde la matriz } \mathbf{I} \text{ es de orden } n.$$

De manera similar, al poner la *abreviatura* "$\boldsymbol{\tau}$" de la transformación elemental a izquierda, es como si multiplicáramos la matriz $\underset{m\times n}{\mathbf{A}}$ por la izquierda por la correspondiente matriz elemental

$$_{\boldsymbol{\tau}_2}\mathbf{A} = {}_{\boldsymbol{\tau}_2}\mathbf{I}\mathbf{A} = \mathbf{E}_2\mathbf{A} \quad \text{donde} \quad \mathbf{E}_2 = {}_{\boldsymbol{\tau}_2}\mathbf{I} \quad \text{y donde la matriz } \mathbf{I} \text{ es de orden } m.$$

Con ello se gana, entre otras cosas, que la notación sea asociativa. Pero entonces se plantea ¿qué ventaja tiene introducir en el discurso las transformaciones elementales en lugar de utilizar simplemente matrices elementales?

1. Una matriz cuadrada es un objeto muy pesado… n^2 coeficientes para una matriz de orden n. Afortunadamente una matriz elemental es casi una matriz identidad salvo por uno de sus elementos; por tanto, para describir completamente una matriz elemental basta indicar su orden n y el componente que no coincide con los de la matriz $\mathbf{I}$ de orden n. [2]

2. La ventaja es que las transformaciones elementales omiten el orden n.

Vamos a definir la siguiente traducción a Python de esta notación:

Mates II	Python	Mates II	Python
$\mathbf{A}\underset{[\boldsymbol{i}\rightleftharpoons\boldsymbol{j}]}{\boldsymbol{\tau}}$	`A & T( {i,j} )`	$\underset{[\boldsymbol{i}\rightleftharpoons\boldsymbol{j}]}{\boldsymbol{\tau}}\mathbf{A}$	`T( {i,j } ) & A`
$\mathbf{A}\underset{[(a)\boldsymbol{j}]}{\boldsymbol{\tau}}$	`A & T( (a,j) )`	$\underset{[(a)\boldsymbol{i}]}{\boldsymbol{\tau}}\mathbf{A}$	`T( (a,i) ) & A`
$\mathbf{A}\underset{[(a)\boldsymbol{i}+\boldsymbol{j}]}{\boldsymbol{\tau}}$	`A & T( (a,i,j) )`	$\underset{[(a)\boldsymbol{i}+\boldsymbol{j}]}{\boldsymbol{\tau}}\mathbf{A}$	`T( (a,i,j) ) & A`

Vemos que:

1. Representar el intercambio con un conjunto, permite admitir la repetición del índice $\{i, i\} = \{i\}$ como un caso especial en el que la matriz no cambia. Esto simplificará el método de Gauss.

2. Tanto para los pares (a, i) como las ternas (a, i, j)
   (a) La columna (fila) que cambia es la del índice que aparece en última posición.
   (b) El escalar aparece en la primera posición y multiplica a la columna (fila) del siguiente índice.

[2]Fíjese que la notación usada en las notas de la asignatura para las matrices elementales $\mathbf{E}$, no las describe completamente (se deja al lector la deducción de cuál es el orden adecuado para poder realizar el producto $\mathbf{AE}$ o $\mathbf{EA}$)

Empleando listas de abreviaturas extendemos la notación para expresar secuencias de transformaciones elementales, es decir, $\boldsymbol{\tau}_1 \cdots \boldsymbol{\tau}_k$. Así logramos la siguiente equivalencia entre expresiones

$$\texttt{T(}\ t_1\texttt{)}\ \texttt{\&}\ \texttt{T(}\ t_2\ \texttt{)}\ \texttt{\&}\cdots\texttt{\&}\ \texttt{T(}\ t_k\texttt{)} \quad = \quad \texttt{T(}\ \texttt{[}t_1\texttt{,}t_2\texttt{,}\ldots\texttt{,}t_k\texttt{]}\ \texttt{)}$$

De esta manera

$$\mathbf{A}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}: \qquad \mathbf{A}\ \texttt{\&}\ \texttt{T(}\ t_1\texttt{)}\ \texttt{\&}\ \texttt{T(}\ t_2\ \texttt{)}\ \texttt{\&}\cdots\texttt{\&}\ \texttt{T(}\ t_k\texttt{)} = \mathbf{A}\ \texttt{\&}\ \texttt{T(}\ \texttt{[}t_1\texttt{,}t_2\texttt{,}\ldots\texttt{,}t_k\texttt{]}\ \texttt{)}$$

$${}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}\mathbf{A}: \qquad \texttt{T(}\ t_1\texttt{)}\ \texttt{\&}\ \texttt{T(}\ t_2\ \texttt{)}\ \texttt{\&}\cdots\texttt{\&}\ \texttt{T(}\ t_k\texttt{)}\ \texttt{\&}\ \mathbf{A} = \texttt{T(}\ \texttt{[}t_1\texttt{,}t_2\texttt{,}\ldots\texttt{,}t_k\texttt{]}\ \texttt{)}\ \texttt{\&}\ \mathbf{A}.$$

Si $\mathbf{A}$ es de orden $m \times n$, el primer caso es equivalente a escribir el producto de matrices

$$\mathbf{A}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k} = \mathbf{A}\mathbf{E}_1\mathbf{E}_2\cdots\mathbf{E}_k \qquad \text{donde } \mathbf{E}_j = \mathbf{I}_{\boldsymbol{\tau}_j} \text{ y donde } \mathbf{I} \text{ es de orden } n.$$

Y el segundo caso es equivalente a escribir el producto de matrices

$${}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}\mathbf{A} = \mathbf{E}_1\mathbf{E}_2\cdots\mathbf{E}_k\mathbf{A} \qquad \text{donde } \mathbf{E}_i = {}_{\boldsymbol{\tau}_i}\mathbf{I} \text{ y donde } \mathbf{I} \text{ es de orden } m.$$

32 ⟨*Texto de ayuda de la clase* `T` *(Transformación Elemental)* 32⟩≡

```
"""Clase T

T ("Transformación elemental") guarda en su atributo 't' una abreviatura
(o una secuencia de abreviaturas) de transformaciones elementales. Con
el método __and__ actúa sobre otra T para crear una T que es composición
de transformaciones elementales (la lista de abreviaturas), o bien actúa
sobre una Matrix (para transformar sus filas)

Atributos:
    t (set)  : {índice, índice}. Abrev. de un intercambio entre los
                 vectores correspondientes a dichos índices
      (tuple): (índice, número). Abrev. transf. Tipo II que multiplica
                 el vector correspondiente al índice por el número
             : (índice1, índice2, número). Abrev. transformación Tipo I
                 que suma al vector correspondiente al índice1 el vector
                 correspondiente al índice2 multiplicado por el número
      (list) : Lista de conjuntos y tuplas. Secuencia de abrev. de
                 transformaciones como las anteriores.
      (T)    : Transformación elemental. Genera una T cuyo atributo t es
                 una copia del atributo t de la transformación dada
      (list) : Lista de transformaciones elementales. Genera una T cuyo
                 atributo es la concatenanción de todas las abreviaturas
Ejemplos:
>>> # Intercambio entre vectores
>>> T( {1,2} )

>>> # Trasformación Tipo II (multiplica por 5 el segundo vector)
>>> T( (5,2) )

>>> # Trasformación Tipo I (resta el tercer vector al primero)
>>> T( (-1,3,1) )

>>> # Secuencia de las tres transformaciones anteriores
>>> T( [{1,2}, (5,2), (-1,3,1)] )

>>> # T de una T
>>> T( T( (5,5) ) )
```

```
  T( (5,2) )

  >>> # T de una lista de T's
  >>> T( [T([(-8, 2), (2, 1, 2)]), T([(-8, 3), (3, 1, 3)]) ] )

  T( [(-8, 2), (2, 1, 2), (-8, 3), (3, 1, 3)] )
  """
```

This code is used in chunk 37b.
Uses Matrix 13 and T 37b.

1.6.1 Implementación

Python ejecuta las órdenes de izquierda a derecha. Fijándonos en la expresión

$$\mathbf{A}\ \ \&\ \mathtt{T(}\ t_1\mathtt{)}\ \&\ \mathtt{T(}\ t_2\ \mathtt{)}\ \&\cdots\&\ \mathtt{T(}\ t_k\mathtt{)}$$

podríamos pensar que podemos implementar la transformación elemental como un método de la clase `Matrix`. Así, al definir el método `__and__` por la derecha de la matriz podemos indicar que $\mathbf{A}\ \&\ \mathtt{T(}\ t_1\mathtt{)}$ es una nueva matriz con las columnas modificadas. Python no tiene problema en ejecutar $\mathbf{A}\ \&\ \mathtt{T(}\ t_1\mathtt{)}\ \&\ \mathtt{T(}\ t_2\ \mathtt{)}\ \&\cdots\&\ \mathtt{T(}\ t_k\mathtt{)}$ pues ejecutar de izquierda a derecha, es lo mismo que ejecutar $\Big[\big[[\mathbf{A}\ \&\ \mathtt{T(}\ t_1\mathtt{)}]\&\ \mathtt{T(}\ t_2\ \mathtt{)}\big]\&\cdots\Big]\&\ \mathtt{T(}\ t_k\mathtt{)}$ donde la expresión dentro de cada corchete es una `Matrix`, por lo que las operaciones están definidas. La dificultad aparece con

$$\mathtt{T(}\ t_1\mathtt{)}\ \&\ \mathtt{T(}\ t_2\ \mathtt{)}\ \&\cdots\&\ \mathtt{T(}\ t_k\mathtt{)}\ \&\ \mathbf{A}$$

Lo primero que Python tratara de ejecutar es $\mathtt{T(}\ t_1\ \mathtt{)}\ \&\ \mathtt{T(}\ t_2\mathtt{)}$, pero ni $\mathtt{T(}\ t_1\mathtt{)}$ ni $\mathtt{T(}\ t_2\mathtt{)}$ son matrices, por lo que esto no puede ser programado como un método de la clase `Matrix`.

Por tanto, necesitamos definir una nueva clase que almacene las *abreviaturas* "t_i" de las operaciones elementales, de manera que podamos definir $\mathtt{T(}\ t_i\ \mathtt{)}\ \&\ \mathtt{T(}\ t_j\mathtt{)}$, como un método que "compone" dos transformaciones elementales para formar una secuencia de abreviaturas (de operaciones a ejecutar sobre una `Matrix`).

Así pues, definimos un nuevo tipo de objeto: `T` ("transformación elemental") que nos permitirá encadenar transformaciones elementales (es decir, almacenar una lista de abreviaturas). El siguiente código inicializa la clase. El atributo `t` almacenará la abreviatura (o lista de abreviaturas) dada al instanciar `T` o bien creará la lista de abreviaturas a partir de otra `T` (o de una lista de `T`s) empleada para instanciar.

33 ⟨*Inicialización de la clase* `T` *(Transformación Elemental)* 33⟩≡

```
  def __init__(self, t):
      """Inicializa una transformación elemental"""
      ⟨Método auxiliar CreaLista que devuelve listas de abreviaturas 34b⟩
      ⟨Creación del atributo t cuando se instancia con otra T o lista de Ts 85⟩
      else:
          self.t = t
      ⟨Verificación de que las abreviaturas corresponden a transformaciones elementales 34a⟩
```

This code is used in chunk 37b.

34a ⟨*Verificación de que las abreviaturas corresponden a transformaciones elementales* 34a⟩≡

```
for j in CreaLista(self.t):
    if isinstance(j,tuple) and (len(j) == 2) and j[0]==0:
        raise ValueError('T( (0, i) ) no es una trasformación elemental')
    if isinstance(j,tuple) and (len(j) == 3) and (j[1] == j[2]):
        raise ValueError('T( (a, i, i) ) no es una trasformación elemental')
    if isinstance(j,set) and (len(j) > 2) or not j:
        raise ValueError \
        ('El conjunto debe tener uno o dos índices para ser trasformación elemental')
```

This code is used in chunk 33.
Uses `CreaLista` 34b.

Con algunas operaciones, como la composición de transformaciones elementales requeriremos operar con listas de abreviaturas. El siguiente procedimiento *crea la lista* `[t]` que contiene a `t` (cuando `t` no es una lista), si `t` es una lista, el procedimiento no hace nada. Se usa al instaciar `T` con una lista de `T`s y también al componer transformaciones elementales.

34b ⟨*Método auxiliar* `CreaLista` *que devuelve listas de abreviaturas* 34b⟩≡

```
def CreaLista(t):
    """Devuelve t si t es una lista; si no devuelve la lista [t]"""
    return t if isinstance(t, list) else [t]
```

This code is used in chunks 33, 35, and 37a.
Defines:
`CreaLista`, used in chunks 34a, 35, 37a, and 85.

Composición de `Transf.` element. o llamada al método de transformación de filas de una `Matrix`

34c ⟨*Texto de ayuda para la composición de Transformaciones Elementales* `T` 34c⟩≡

```
"""Composición de transformaciones elementales (o transformación filas)

Crea una T con una lista de abreviaturas de transformaciones elementales
(o llama al método que modifica las filas de una Matrix)

Parámetros:
    (T): Crea la abreviatura de la composición de transformaciones, es
         decir, una lista de abreviaturas
    (Matrix): Llama al método de la clase Matrix que modifica las filas
         de Matrix
Ejemplos:
>>> # Composición de dos Transformaciones elementales
>>> T( {1, 2} ) & T( (2, 4) )
```

```
T( [{1,2}, (2,4)] )

>>> # Composición de dos Transformaciones elementales
>>> T( {1, 2} ) & T( [(2, 4), (2, 1), {3, 1}] )

T( [{1, 2}, (2, 4), (2, 1), {3, 1}] )

>>> # Transformación de las filas de una Matrix
>>> T( [{1,2}, (4,2)] ) & A # multiplica por 4 la segunda fila de A y
                            # luego intercambia las dos primeras filas
"""
```

This code is used in chunk 35.
Uses `Matrix` 13 and `T` 37b.

Describimos la composición de transformaciones `T(`t_1`) & T(`t_2`)` creando una lista de abreviaturas `[`t_1, t_2`]` (mediante la concatenación de listas)[3]. Si el atributo del método `__and__` de la clase `T` es una `Matrix`, llama al método `__rand__` de la clase `Matrix` (que transforma las filas de la matriz y que veremos un poco más abajo).

35 ⟨*Composición de Transformaciones Elementales o aplicación sobre las filas de una* `Matrix` 35⟩≡

```
def __and__(self, other):
    ⟨Texto de ayuda para la composición de Transformaciones Elementales T 34c⟩
    ⟨Método auxiliar CreaLista que devuelve listas de abreviaturas 34b⟩
    if isinstance(other, T):
        return T(CreaLista(self.t) + CreaLista(other.t))

    if isinstance(other, Matrix):
        return other.__rand__(self)
```

This code is used in chunk 37b.
Uses `CreaLista` 34b, `Matrix` 13, and `T` 37b.

1.6.2 Transposición de transformaciones elementales

Puesto que $\mathbf{I}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k} = (\mathbf{E}_1\cdots\mathbf{E}_k)$ y puesto que el producto de matrices es asociativo, deducimos que la transpuesta de $\mathbf{I}_{\boldsymbol{\tau}_1\boldsymbol{\tau}_2\cdots\boldsymbol{\tau}_k}$ es

$$\left(\mathbf{I}_{\boldsymbol{\tau}_1\boldsymbol{\tau}_2\cdots\boldsymbol{\tau}_k}\right)^{\intercal} = (\mathbf{I}\mathbf{E}_1\cdots\mathbf{E}_k)^{\intercal} = \mathbf{E}_k^{\intercal}\cdots\mathbf{E}_1^{\intercal}\mathbf{I} = {}_{\boldsymbol{\tau}_k\cdots\boldsymbol{\tau}_2\boldsymbol{\tau}_1}\mathbf{I}$$

Nótese cómo al transponer no solo cambiamos de lado los subíndices, sino también invertimos el orden de la secuencia de transformaciones (de la misma manera que también cambia el orden en el que se multiplican las matrices elementales). Esto sugierie denotar a la operación de invertir el orden de las transformaciones como una transposición:

$$(\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k)^{\intercal} = \boldsymbol{\tau}_k\cdots\boldsymbol{\tau}_1;$$

así

$$\left(\mathbf{A}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}\right)^{\intercal} = {}_{(\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k)^{\intercal}}\mathbf{A}^{\intercal} \qquad = \qquad {}_{\boldsymbol{\tau}_k\cdots\boldsymbol{\tau}_1}\mathbf{A}^{\intercal}$$

El siguiente procedimiento invierte el orden de la lista cuando `t` es una lista de abreviaturas. Cuando `t` es una única abreviatura, no hace nada.

[3]Recuerde que la suma de listas (`list + list`) concatena las listas

36a ⟨*Operador transposición para la clase* T 36a⟩≡
```
def __invert__(self):
    """Transpone la lista de abreviaturas (invierte su orden)"""
    return T( list(reversed(self.t)) ) if isinstance(self.t, list) else self
```
This code is used in chunk 37b.
Uses T 37b.

1.6.3 Inversión de transformaciones elementales

Cualquier matriz de la forma $\mathbf{I}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}$ o de la forma ${}_{\boldsymbol{\tau}_k\cdots\boldsymbol{\tau}_1}\mathbf{I}$ es invertible por ser producto de matrices elementales, en particular,

$$\left(\mathbf{I}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}\right)\left(\mathbf{I}_{\boldsymbol{\tau}_k^{-1}\cdots\boldsymbol{\tau}_1^{-1}}\right)=\mathbf{E}_1\cdots\mathbf{E}_k\cdot\mathbf{E}_k^{-1}\cdots\mathbf{E}_1^{-1}=\mathbf{I}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k\cdot\boldsymbol{\tau}_k^{-1}\cdots\boldsymbol{\tau}_1^{-1}}=\mathbf{I};$$

por lo que podemos denotar por $\left(\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k\right)^{-1}$ a la sucesión de transformaciones $\boldsymbol{\tau}_k^{-1}\cdots\boldsymbol{\tau}_1^{-1}$. De este modo

$$\left(\mathbf{I}_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}\right)^{-1}=\mathbf{I}_{(\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k)^{-1}}.$$

El siguiente método devuelve la potencia `n`-esima de una transformación elemental. Si `n` es -1, calcula la inversa. Por ejemplo,

```
T([ (1,2,3), (2,3), {1,2} ]) **(-1)
```

es

```
T([ {1,2}, (Fraction(1,2),3), (-1,2,3) ])
```

36b ⟨*Potencia de una* T 36b⟩≡
```
def __pow__(self,n):
    """Calcula potencias de una T (incluida la inversa)"""
    ⟨Método auxiliar que calcula la inversa de una Transformación elemental 37a⟩
    if not isinstance(n,int):
        raise ValueError('La potencia no es un entero')
    t = self

    for i in range(1,abs(n)):
        t = t & self

    if n < 0:
        t = Tinversa(t)

    return t
```
This code is used in chunk 37b.
Uses T 37b.

37a ⟨*Método auxiliar que calcula la inversa de una* `T`*ransformación elemental* 37a⟩≡

```
  def Tinversa ( self ):
      """Calculo de la inversa de una transformación elemental"""
      ⟨Método auxiliar CreaLista que devuelve listas de abreviaturas 34b⟩

      listaT = [ ( -j[0], j[1], j[2] )    if (isinstance(j,tuple) and len(j)==3) else \
                 (Fraction(1,j[0]), j[1]) if (isinstance(j,tuple) and len(j)==2) else \
                 j                                             for j in CreaLista(self.t) ]

      return ~T( listaT )
```

This code is used in chunk 36b.
Uses `CreaLista` 34b and `T` 37b.

La clase `T` junto con el listado de sus métodos aparece en el siguiente recuadro:

37b ⟨*Definición de la clase* `T` *(Transformación Elemental)* 37b⟩≡

```
  class T:
      ⟨Texto de ayuda de la clase T (Transformación Elemental) 32⟩
      ⟨Inicialización de la clase T (Transformación Elemental) 33⟩
      ⟨Composición de Transformaciones Elementales o aplicación sobre las filas de una Matrix 35⟩
      ⟨Operador transposición para la clase T 36a⟩
      ⟨Potencia de una T 36b⟩
      ⟨Representación de la clase T 86⟩
```

This code is used in chunk 41.
Defines:
`T`, used in chunks 32, 34–40, 46–48, 59–63, 66b, 85, 86, and 88.

1.7 Transformaciones elementales de un Sistema

En el segundo Tema de las notas de la asignatura, se definen las transformaciones elementales sobre Sistemas como una generalización a las trasnformaciones elementales de las columnas de una Matrix. Puesto que cada Matrix es un Sistema de vectores, en la librería vamos a comezar con las transformaciones elementales de un Sistema.

38a ⟨*Texto de ayuda de las transformaciones elementales de un* Sistema 38a⟩≡

```
"""Transforma los elementos de un Sistema S

Atributos:
    t (T): transformaciones a aplicar sobre un Sistema S
Ejemplos:
>>>  S & T({1,3})             # Intercambia los elementos 1º y 3º
>>>  S & T((5,1))             # Multiplica por 5 el primer elemento
>>>  S & T((5,2,1))           # Suma 5 veces el elem. 1º al elem. 2º
>>>  S & T([{1,3},(5,1),(5,2,1)])# Aplica la secuencia de transformac.
             # sobre los elementos de S y en el orden de la lista
"""
```

This code is used in chunk 38b.
Uses Sistema 7 and T 37b.

Implementación de la aplicación de las transformaciones elementales sobre los elementos de un Sistema (nótese que hemos incluido el intercambio, aunque usted ya sabe que es una composición de los otros dos tipos de transf.)

38b ⟨*Transformaciones elementales de los elementos de un* Sistema 38b⟩≡

```
def __and__(self,t):
    ⟨Texto de ayuda de las transformaciones elementales de un Sistema 38a⟩
    if isinstance(t.t,set):
        self.lista = Sistema([(self|max(t.t)) if k==min(t.t) else \
                              (self|min(t.t)) if k==max(t.t) else \
                              (self|k) for k in range(1,len(self)+1)] ).lista.copy()

    elif isinstance(t.t,tuple) and (len(t.t) == 2):
        self.lista = Sistema([ t.t[0]*(self|k) if k==t.t[1] else  \
                              (self|k) for k in range(1,len(self)+1)] ).lista.copy()

    elif isinstance(t.t,tuple) and (len(t.t) == 3):
        self.lista = Sistema([ t.t[0]*(self|t.t[1]) + (self|k) if k==t.t[2] else \
                              (self|k) for k in range(1,len(self)+1)] ).lista.copy()
    elif isinstance(t.t,list):
        for k in t.t:
            self & T(k)
    return self
```

This code is used in chunk 7.
Uses Sistema 7 and T 37b.

Observación 1. Al actuar sobre `self.lista`, las transformaciones elementales modifican el Sistema.

1.8 Transformaciones elementales de una Matrix

1.8.1 Transformaciones elementales de las columnas de una Matrix

Ahora ya podemos transformar de manera sencilla las columnas de una Matrix

39a ⟨*Texto de ayuda de las transformaciones elementales de las columnas de una* Matrix 39a⟩≡

```
"""Transforma las columnas de una Matrix

Atributos:
    t (T): transformaciones a aplicar sobre las columnas de Matrix

Ejemplos:
>>>  A & T({1,3})                # Intercambia las columnas 1 y 3
>>>  A & T((5,1))                # Multiplica la columna 1 por 5
>>>  A & T((5,2,1))              # suma 5 veces la col. 2 a la col. 1
>>>  A & T([{1,3},(5,1),(5,2,1)])# Aplica la secuencia de transformac.
             # sobre las columnas de A y en el mismo orden de la lista
"""
```

This code is used in chunk 39b.
Uses Matrix 13 and T 37b.

Implementación de la aplicación de las transformaciones elementales sobre las columnas de una Matrix. Basta transformar el correspondiente Sistema de columnas.

39b ⟨*Transformaciones elementales de las columnas de una* Matrix 39b⟩≡

```
def __and__(self,t):
    ⟨Texto de ayuda de las transformaciones elementales de las columnas de una Matrix 39a⟩
    self.sis = (self.sis & t).copy()
    return self
```

This code is used in chunk 13.

Observación 2. Al actuar sobre `self.sis`, las transformaciones elementales modifican la matriz.

1.8.2 Transformaciones elementales de las filas de una Matrix

39c ⟨*Texto de ayuda de las transformaciones elementales de las filas de una* Matrix 39c⟩≡

```
"""Transforma las filas de una Matrix

Atributos:
    t (T): transformaciones a aplicar sobre las filas de Matrix

Ejemplos:
>>>  T({1,3})   & A              # Intercambia las filas 1 y 3
```

```
>>>  T((5,1))   & A              # Multiplica por 5 la fila 1
>>>  T((5,2,1)) & A              # Suma 5 veces la fila 2 a la fila 1
>>>  T([(5,2,1),(5,1),{1,3}]) & A # Aplica la secuencia de transformac.
            # sobre las filas de A y en el orden inverso al de la lista
"""
```

This code is used in chunk 40.
Uses Matrix 13 and T 37b.

Para implementar las transformaciones elementales de las filas usamos el truco de aplicar las operaciones sobre las columnas de la transpuesta y de nuevo transponer el resultado: `~(~self & t)`. Pero hay que recordar que las transformaciones más próximas a la matriz se ejecutan primero, puesto que $_{\boldsymbol{\tau}_1\cdots\boldsymbol{\tau}_k}\mathbf{A} = \mathbf{E}_1\mathbf{E}_2\cdots\mathbf{E}_k\mathbf{A}$. Con la función `reversed` aplicamos la sucesión de transformaciones en el orden inverso a como aparecen en la lista:

$$\texttt{T([}t_1\texttt{,}t_2\texttt{,}\ldots\texttt{,}t_k\texttt{]) \& }\mathbf{A} \quad = \quad \texttt{T(}t_1\texttt{) \&}\cdots\texttt{\& T(}t_{k-1}\texttt{) \& T(}t_k\texttt{) \& }\mathbf{A}$$

40 ⟨*Transformaciones elementales de las filas de una* Matrix 40⟩≡

```
def __rand__(self,t):
    ⟨Texto de ayuda de las transformaciones elementales de las filas de una Matrix 39c⟩
    if isinstance(t.t,set) | isinstance(t.t,tuple):
        self.sis = (~(~self & t)).sis.copy()

    elif isinstance(t.t,list):
        for k in reversed(t.t):
            T(k) & self

    return self
```

This code is used in chunk 13.
Uses T 37b.

Observación 3. Al actuar sobre `self.lista`, las transformaciones elementales modifican la matriz.

1.9 Librería completa

Finalmente creamos la librería `notacion.py` concatenando los trozos de código que se describen en este fichero de documentación (recuerde que el número que aparece detrás de nombre de cada trozo de código indica la página donde encontrar el código en este documento).

Pero antes de todo, y para que los vectores funcionen como un espacio vectorial, es esencial importar la clase `Fraction` de la librería `fractions` [4]. Así pues, antes de nada, importamos la clase Fraction de números fraccionarios con el código:

```
from fractions import Fraction
```

41 ⟨*notacion.py* 41⟩≡

```
# coding=utf8

from fractions import Fraction

⟨Método html general 75a⟩
⟨Método latex general 75b⟩
⟨Métodos html y latex para fracciones 77⟩

⟨Definición de la clase Vector 9b⟩
⟨Definición de la clase Matrix 13⟩
⟨Definición de la clase T (Transformación Elemental) 37b⟩
⟨Definición de la clase BlockMatrix 92b⟩

⟨Definición del método particion 93a⟩
⟨Definición del procedimiento de generación del conjunto clave para particionar 95a⟩

⟨Definición de vector nulo: V0 83c⟩
⟨Definición de matriz nula: M0 84a⟩
⟨Definición de la matriz identidad: I 84b⟩

⟨Variantes que reducen una matriz por eliminación por columnas y guardan los pasos dados 50a⟩
⟨Variantes que escalonan una matriz por eliminación por columnas y guardan los pasos dados 54b⟩
⟨Variantes que reducen una matriz por eliminación por filas y guardan los pasos dados 57b⟩
⟨Variantes que escalonan una matriz por eliminación por filas y guardan los pasos dados 58⟩
⟨Invirtiendo una matriz 59b⟩
⟨La clase Sistema 7⟩
⟨La clase SubEspacio 69c⟩
⟨La clase EAfin 70b⟩
⟨Resolviendo un sistema homogeneo 61a⟩
⟨Resolviendo un Sistema de Ecuaciones Lineales 62⟩
⟨normal 63a⟩
⟨sistema 63b⟩
```

Root chunk (not used in this document).

[4] el tipo de datos `float` no tiene estructura de cuerpo. Por ello vamos a emplear números fraccionarios del tipo $\frac{a}{b}$, donde a y b son enteros. Así pues, en el fondo estaremos trabajando con vectores de $\mathbb{Q}^n$ (en lugar de vectores de $\mathbb{R}^n$). Más adelante intentaré emplear un cuerpo de números más grande, que incluya números reales tales como $\sqrt{2}$... y también el cuerpo de fracciones de polinomios (para obtener el polinomio característico vía eliminación gaussiana, en lugar de vía determinantes).

Tutorial previo en un Jupyter notebook

Consulte el Notebook sobre el **uso de nuestra librería para Mates 2** en la carpeta "TutorialPython" en
`https://mybinder.org/v2/gh/mbujosab/LibreriaDePythonParaMates2/master`

Para empaquetar esta librería en el futuro: `https://packaging.python.org/tutorials/packaging-projects/`

Capítulo 2

Algoritmos del curso

2.1 Eliminación Gaussiana

En las notas de clase llamamos *pivote* (de una columna no nula) a su primer componente no nulo; y *posición de pivote* al índice de la fila en la que está el pivote. Vamos a generalizar esta definición y decir sencillamente que llamamos *pivote* de un `Vector` (no nulo) a su primer componente no nulo; y *posición de pivote* al índice de dicho componente (así podremos usar la definicón de pivote tanto si programamos el método de eliminación por filas como por columnas).

Por conveniencia, el método `ppivote` nos indicará el *primer índice* mayor que `k` de un componente no nulo del `Vector`. Como por defecto `k=0`, si no especificamos el valor de `k`, entonces nos devuelve la *posición de pivote* de un `Vector`. Si no hay ningún componente no nulo de índice mayor que `k`, pivote nos devuelve el valor cero. Así, si $\boldsymbol{a} = (0, 5, 0, 5)$, entonces

`ppivote(a)=2;`   `ppivote(a,0)=2;`   `ppivote(a,2)=4;`   `ppivote(a,4)=0`.

Por convenencia también vamos a permitir la búsqueda de componentes no nulos en sentido *inverso* (comenzando por el final). Si fijamos un valor distinto a `'Normal'` para la variable `sentido`, entonces el vector se lee en sentido contrario. Así,

`ppivote(a, sentido='c')=4;`   `ppivote(a,4, sentido='c')=2;`   `ppivote(a,2, sentido='c')=0`.

es decir, el último componente no nulo es el cuarto, el anterior es el segundo y antes del segundo componente solo hay ceros. Esto nos permite diseñar algoritmos alternativos de eliminación que "barren" las filas de derecha a izquierda.

43 ⟨ppivote*: cálculo del índice del primer coef. no nulo de un* Vector *a partir de la posición* r 43⟩≡

```
def ppivote(v, k=0, sentido='Normal'):
    """Devuelve el primer índice(i) mayor que k de un coeficiente(c) no
    nulo del Vector v. En caso de no existir devuelve 0. Si el vector
    se recorre en sentido normal, devuelve el último índice(i) menor
    que k de v."""
    if sentido=='Normal':
        return ( [i for i,c in enumerate(v.sis, 1) if (c!=0 and i>k)] + [0] )[0]
    else:
        k = k-1 if k else len(v.sis)
        return ( [i[0] for i in reversed(list(enumerate(v.sis,1))) \
                                           if (i[1]!=0 and i[0]<=k) ] + [0] )[0]
```

This code is used in chunk 45.
Defines:
`ppivote`, used in chunk 45.
Uses `Vector` 9b.

Decimos que la matriz L *es escalonada*, si toda columna que precede a una no nula $\mathsf{L}_{|k}$ no es nula y su posición de pivote es anterior a la posición de pivote de $\mathsf{L}_{|k}$. Por tanto, en una matriz escalonada por columnas el primer pivote corresponde a la primera columna no nula, el segundo pivote a la segunda columna no nula, etc. El Teorema del final de las secciones de referencia de la Lección 4 demuestra que toda matriz es escalonable. Y la demostración nos indica el procedimiento para programar el método.

La variable `r` es un contador de los pivotes encontrados en las columnas de la matriz escalonada (inicialmente `r=0`), y **se procede iterativamente** comenzando por la primera fila (`i=1`). En cada iteración buscamos la posición de pivote `p` de la fila `(i|A)` en la que vamos a eliminar componentes. Caben dos posibilidades

- Si `p` es cero, quiere decir que todos los componentes de la fila `(i|A)` son cero. Así que ya hemos acabado con la eliminación para dicha fila, por lo que pasamos a la siguiente iteración (sobre la siguiente fila `i`-ésima).

- Si `p` es positivo, hemos encontrado un nuevo pivote en la posición `p` (donde `p>r`).

  - El contador de pivotes aumenta en una unidad: `r+=1`
  - *Como estamos escalonando, intercambiamos la columna* `p`*-ésima donde hemos encontrado el pivote con la columna* `r`*-ésima.* Es decir:

    como en el cuarto caso de la demostración por inducción del teorema de la Lección 4 del curso, *intercambiamos la posición de las columnas* `p`*-ésima y* `r`*-ésima*: `A & T( {p, r} )` . Nótese que la posición de pivote `p` es necesariamente mayor o igual al número de pivotes encontrado `r` (cuando `p==r` este paso no hace nada, véase la definición de intercambio).
  - *A continuación debemos proceder a eliminar de izquierda a derecha hasta anular los componentes de la fila que están más a la izquierda del pivote, que ahora está en la posición* `r`*.* Es decir:

    como en el tercer caso de la demostración por inducción del teorema de la Lección 4 del curso, aplicamos la sucesión de tranformaciones elementales Tipo I,

    ```
    (Fraction(-(i|A|j),(i|A|r)), r, j) for j in range(r+1,A.n+1)
    ```

    es decir

    $$\underset{\left[\left(\frac{-a_{ij}}{a_{ir}}\right)\boldsymbol{r}+\boldsymbol{j}\right]}{\boldsymbol{\tau}}, \quad \text{con} \quad j = r : n.$$

  Como hemos anulado todos los componentes a la derecha del pivote de la fila `(i|A)`, ya hemos acabado con la eliminación para dicha fila y pasamos a la siguiente iteración (sobre la siguiente fila `i`-ésima).

Fíjese que en el teorema sólo se indica el paso para la primera fila no nula (es decir, para `r=1`).[1] Pero en este algorimto se continúa fila a fila con el mismo procedimiento hasta escalonar toda la matriz.

Este algoritmo aparece tal cual en el Notebook de Jupyter de la Lección 4 (Sección *Escalonando una matriz por el método de eliminación gaussiana*). De manera que por ejemplo, si $\mathsf{A} = \begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}$, le logra el siguiente escalonamiento (en el que por claridad indico todos y cada una de las trasformaciones empleadas):

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix} \xrightarrow[]{\substack{\boldsymbol{\tau}\\ [\mathbf{1}\rightleftharpoons\mathbf{2}]\\ [(0)\mathbf{1}+\mathbf{2}]\\ [(-1)\mathbf{1}+\mathbf{3}]\\ [\mathbf{2}\rightleftharpoons\mathbf{2}]\\ [(\frac{-3}{7})\mathbf{2}+\mathbf{3}]\\ [\mathbf{3}\rightleftharpoons\mathbf{3}]}} \begin{bmatrix}1&0&0\\-3&7&0\\4&6&\frac{-32}{7}\end{bmatrix}.$$

Aunque aquí muestro todos los pasos dados por el algoritmo cuyo código aparece en el Notebook de Jupyter de la Lección 4 (Sección *Escalonando una matriz por el método de eliminación gaussiana*), lo cierto es que el algoritmo se limita a efectuar las transformaciones necesarias y mostrar la forma escalonada final (pero no nos indica qué transformaciones ha empleado).

Aquí vamos dar mayor operatividad al agoritmo en dos sentidos. Por una parte vamos a guardar el listado de abreviaturas de las transformaciones para poder aplicar la misma sucesión de transformaciones a otras matrices

[1] pues por hipótesis de inducción se supone que la submatriz B es escalonable

(además podremos representar el procedimiento de eliminación paso a paso); y por otra vamos a permitir otras formas de eliminación Gaussiana (eliminación por filas o por columnas, reducir una matriz sin llegar a escalonarla, realizar la eliminación de derecha a izquierda o eliminar componentes tanto a la izquierda como a la derecha de cada pivote, evitar las fracciones (en la medida de lo posible), o normalizar los pivotes para que todos sean iguales a 1).

2.1.1 Variantes de eliminación gaussiana

1. Todos los algoritmos los vamos a programar mediante operaciones por columnas. Más adelante combinaremos estos algoritmos de eliminación con la transposición para lograr la eliminación por filas.
2. Llamaremos eliminación *Gaussiana* a la eliminación de izquierda a derecha; es decir, cuando anulamos todo lo que queda a la derecha de cada pivote.
3. Llamaremos eliminación *Gauss-Jordan* cuando anulamos todo lo que queda a la derecha y la izquierda de cada pivote.
4. Llamaremos eliminación *Gauss-Opuesto* a la eliminación de derecha a izquierda; es decir, cuando anulamos todo lo que queda a la izquierda de cada pivote.

Búsqueda de nuevos pivotes

Cada pivote debe estar situado en una columna diferente. Para lograr que en todos los casos sea así, generamos el conjunto `columnaOcupada` que contiene los índices de todas las columnas en la que ya hemos encontrado un pivote. Inicalmente `columnaOcupada` es un conjunto vacío; y cada vez que encontremos una columna con pivote, su correspondiente índice será incluido en este conjunto. De esta manera, para cada fila buscaremos (con `ppivote`) un componente no nulo, y si dicho componente se encuentra en una columna con pivote, buscaremos el siguiente componente no nulo de la fila que esté en una columna no ocupada por un pivote encontrado anteriormente. Si esto no es posible, `ppivote` devolverá el valor `0` que no corresponde a ningún índice de columna, por lo que habremos terminado de buscar. El `Sentido` de la búsqueda es el `'Normal'` (de izquierda a derecha) en la eliminación *Gaussiana* o *Gauss-Jordan*. Y el sentido opuesto en la eliminación *Gauss-Opuesto* (no tengo claro que lo vaya a usar, pero lo dejo por el momento...).

45 ⟨*Método auxiliar de búsqueda de un nuevo pivote* 45⟩≡

```
def BuscaNuevoPivote(self, r=0, Sentido='Normal'):
    ⟨ppivote: cálculo del índice del primer coef. no nulo de un Vector a partir de la posición r 43⟩
    p = ppivote(self, r, Sentido)
    while p in columnaOcupada:
        p = ppivote(self, p, Sentido)
    return p
```

This code is used in chunks 50–52 and 54–56.
Defines:
`BuscaNuevoPivote`, used in chunks 50–52 and 54–56.
Uses `ppivote` 43.

En la eliminación eliminación *Gaussiana* o *Gauss-Jordan* se comienza la búsqueda de los pivotes por la primera fila, luego la segunda, etc. Por ello, el cada pivote encontrado estará situado por encima del siguiente que encontremos.

En la eliminación *Gauss-Opuesto* se comienza la búsqueda de los pivotes por la última fila, luego la penúltima, etc. Por ello, el cada pivote encontrado estará situado por debajo del siguiente que encontremos.

Pasos a dar cada vez que encontramos un pivote

A parte de las variantes de eliminación *Gaussiana*, *Gauss-Jordan* y *Gauss-Opuesto*, vamos a contemplar otras variantes que tienen que ver con la naturaleza de los pasos dados en la eliminación. Contemplamos tres tipos de pasos (dos de ellos tiene a su vez dos variantes):

Para cada fila: si se ha encontrado una componente no nula (que actuará como pivote de columna), se pueden dar estos tres tipos de pasos:

1. **(Intercambio)** Si lo que se busca es escalonar la matriz, se han de ordenar las columnas con pivote para que los pivotes describan una *escalera descendente* cuando se recorre la matriz de izquierda a derecha..

   - En la eliminación eliminación *Gaussiana* o *Gauss-Jordan*: Las columnas con pivote se colocan en el orden en que vamos encontrando los correspondientes pivotes (las columnas nulas quedan al final de matriz, es decir, al lado derecho). Por ejemplo:
   $$\begin{bmatrix} 1 & 0 & 0 \\ 1 & -3 & 0 \\ -2 & 3 & 0 \end{bmatrix}$$
   - En la eliminación eliminación *Gauss-Opuesto*: La columna cuya posición de pivote está más abajo se coloca la última, y sucesivamente se van colocando las columnas con pivote para describir una escalera descendente (las columnas nulas quedan al principio de matriz, es decir, en el lado izquierdo). Por ejemplo:
   $$\begin{bmatrix} 0 & 3 & -2 \\ 0 & -3 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

   Esto supone que la posición de cada columna con pivote está en función del orden en que se han ido encontrando los pivotes (en el primer caso, los pivotes más altos son los primeros que se encuentran y los que se situan más a la izquierda. En el segundo caso (eliminación *Gauss-Opuesto*), los primeros pivotes encontrados son los de posición más baja, y se colocan más a la derecha).

   Así pues, si queremos escalonar una matriz, probablemente necesitemos intercambiar columnas para reordenar su posición en función del orden en que hemos encontrado los pivotes. Si llamamos `p` al índice de la columna donde hemos encontrado el pivote, y `pc{r}` la posición que debe ocupar la columna que contiene el pivote en función del número `r` de pivotes encontrados hasta ese momento.

   Por **un paso necesario para escalonar la matriz es realizar los intercambios** requeridos entre las columnas `p`-ésima y `pc{r}`-ésima, e indicar que una vez realizado el intercambio, el nuevo índice de la columna con pivote es `pc{r}`.

   46 ⟨*Intercambio de vectores para escalonar los pivotes* 46⟩≡
   ```
   Tr = T([ {p, pc(r)} ])
   p  = pc(r)
   ⟨Apuntamos las transformaciones Tr de las columnas y las aplicamos 48b⟩
   ```
   This code is used in chunks 54–56.
   Uses `T` 37b.

   Las dos variantes de este paso tiene qe ver con la definición de la función `pc()`

   - Eliminación *Gaussiana* o *Gauss-Jordan*: `pc(x) = x`
     Es decir, el primer pivote encontrado se coloca en la primera columna, el segundo en la segunda, etc.
   - Eliminación *Gauss-Opuesto*: `pc(x) = A.n+1-x`
     Es decir, el primer pivote encontrado se coloca en la última columna, el segundo en la penúltima, etc.

2. **(Normalización)**. En ocasiones será conveniente que los pivotes estén normalizados a 1. Para ello, debemos dividir la columna con pivote por el valor del pivote (que estará situado en la fila `i` y columna `p`),

47a ⟨*Normalizamos el pivote para que sea igual a uno* 47a⟩≡
```
Tr = T([ (Fraction(1, i|A|p), p) ])
```
⟨*Apuntamos las transformaciones* `Tr` *de las columnas y las aplicamos* 48b⟩
This code is used in chunks 52b and 56a.
Uses `T` 37b.

3. **(Eliminación)**: Una vez identificado un pivote, se deben eliminar todas las componentes situadas a su derecha (eliminación *Gaussiana*); o tanto las de la derecha como las de izquierda (eliminación *Gauss-Jordan*): o solo las componentes situadas a su izquierda (eliminación *Gauss-Opuesto*).

   Con la funcion auxiliar `elim` indicaremos qué componentes queremos elimnar

   - indicaremos que solo queremos elimnar los componentes a la derecha de `p` con:

     `elim = lambda x: x > p`

   - indicaremos que que queremos elimnar el resto de componentes (izda + dcha) con:

     `elim = lambda x: x != p`

   - indicaremos que solo queremos elimnar los componentes a la izquierda de `p` con:

     `elim = lambda x: x < p`

   `filter( elim , range(1,A.n+1) )` contiene los índices que queremos eliminar (indicados con la función `elim`).

Eliminación solo con transformaciones Tipo I : Ahora consideremos que queremos escalonar una matriz mediante eliminación gaussiana, por ejemplo

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}\xrightarrow[\boldsymbol{[1\rightleftharpoons2]}]{\boldsymbol{\tau}}\begin{bmatrix}1&0&1\\-3&7&0\\4&6&2\end{bmatrix}\xrightarrow[{[(-1)\mathbf{1}+\mathbf{3}]}]{\boldsymbol{\tau}}\begin{bmatrix}1&0&0\\-3&7&3\\4&6&-2\end{bmatrix}\xrightarrow[{[(\frac{-3}{7})\mathbf{2}+\mathbf{3}]}]{\boldsymbol{\tau}}\begin{bmatrix}1&0&0\\-3&7&0\\4&6&\frac{-32}{7}\end{bmatrix}$$

Si se fija en el último paso, se dará cuenta que este procedimiento da lugar a operaciones con fracciones, ya que para eliminar el número b usando el pivote $a\neq 0$, la estrategia seguida es:

$$b-\left(\frac{b}{a}\right)a=0;$$

así, en un solo paso se elimina b restándole un múltiplo de a.

47b ⟨*Uso del pivote para eliminar componentes con trasformaciones Tipo I* 47b⟩≡
```
Tr = T([(-Fraction(i|A|j, i|A|p), p, j) for j in filter(elim,range(1,A.n+1))])
```
⟨*Apuntamos las transformaciones* `Tr` *de las columnas y las aplicamos* 48b⟩
This code is used in chunks 50b, 52b, 54a, and 56a.
Uses `T` 37b.

Eliminación sin fraciones (sin dividir) : Pero para escalonar una matriz cuyos componentes son números enteros no es necesario trabajar con fracciones. Podemos eliminar el número b encadenando dos operaciones:

$$(-a)b + ba = 0.$$

Es decir, multiplicamos b por $-a$ y luego sumamos ba. Con esta idea podemos aplicar la sucesión de pares de tranformaciones elementales Tipo II y Tipo I:

```
( -(i|A|p),    j)   # Tipo II
(  (i|A|j), p, j)   # Tipo I
```

es decir, $\underset{[(-a_{ip})\mathbf{j}]}{\boldsymbol{\tau}}\;\underset{[(a_{ij})\mathbf{p}+\mathbf{j}]}{\boldsymbol{\tau}}$. El problema de esta solución tan simple, es que si $a = 3$ y $b = 3$, en realidad basta con restar $b - a$, pero la solución de arriba calcularía $(3 \times 3) - (3 \times 3)$, por lo que todos los números de las columnas `r` y `j` se multiplicarían por tres sin que ello fuera estrictamente necesario. Esto hace que podamos terminar con matrices escalonadas con números muy grandes.

No obstante, fíjese que si, por ejemplo, $a = 6$ y $b = 4$, entonces $\frac{b}{a} = \frac{2}{3}$, así que para eliminar b basta con la operación $b \times 3 - a \times 2$; es decir, basta con simplificar la fracción $\frac{b}{a}$ y multiplicar b por el denominador y a por el numerador de la fracción simplificada. El siguiente código usa está idea, teniendo en cuenta que si `n` es un número del tipo `Fraction`, entonces `n.numerator` nos da el numerador de la fracción simplificada y `n.denominator` el denominador.

48a

```
⟨Uso del pivote para eliminar componentes evitando dividir 48a⟩≡
  Tr = T( [ T( [ ( Fraction((i|A|j),(i|A|p)).denominator,   j) ,    \
                 (-Fraction((i|A|j),(i|A|p)).numerator,  p, j)  ] ) \
                              for j in filter(elim,range(1,A.n+1)) ] )
  ⟨Apuntamos las transformaciones Tr de las columnas y las aplicamos 48b⟩
```

This code is used in chunks 51b and 55a.
Uses T 37b.

Siguiendo esta estrategia, ahora los pasos para escalonar por eliminación gaussian serán

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix} \xrightarrow[{[\mathbf{1}\rightleftharpoons\mathbf{2}]}]{\boldsymbol{\tau}} \begin{bmatrix}1&0&1\\-3&7&0\\4&6&2\end{bmatrix} \xrightarrow[{[(-1)\mathbf{1}+\mathbf{3}]}]{\boldsymbol{\tau}} \begin{bmatrix}1&0&0\\-3&7&3\\4&6&-2\end{bmatrix} \xrightarrow[{[(-3)\mathbf{2}+\mathbf{3}]}]{\underset{[(7)\mathbf{3}]}{\boldsymbol{\tau}}} \begin{bmatrix}1&0&0\\-3&7&0\\4&6&-32\end{bmatrix}$$

Hemos visto que son tres los pasos que podemos dar cada vez que encontramos un pivote: **Intercambio**, **normalización** y **eliminación** (bien permitiendo divisiones, o bien sin divisiones).

Se anotan las transformaciones de cada paso y se aplican a las columnas.

Tras cada uno de los pasos hemos definido una serie de transformaciones `Tr` que vamos a apuntar (o concatenar) en una lista de `transformaciones` que más adelante guardaremos como un atributo. Si no se ha indicado ninguna transformación, se concatena una lista vacía. Por último, antes de dar un nuevo paso se aplica `Tr` sobre las *columnas* de la `Matrix`.

48b

```
⟨Apuntamos las transformaciones Tr de las columnas y las aplicamos 48b⟩≡
  transformaciones += [Tr]  if Tr.t else []
  A & T( Tr )
```

This code is used in chunks 46–48.
Uses T 37b.

2.1.2 Reducción de una matriz

La variante más sencilla de eliminación es la reducción de una matriz. El único paso que daremos en cada fila es la eliminación (si hemos encontrado un pivote). No hay normalizacion, ni tampoco se ordenan las columnas, por lo que la matriz final no está escalonada. Dispondremos de tres variantes

- Reducción por eliminación *Gaussiana*. El `Sentido` de la búsqueda de pivotes es de derecha a izquierda (`S = 'Normal'`) y consecuentemente las filas se recorren de arriba a abajo (`sentido = lambda x: x`). Además, solo se elimiman los componentes a la derecha del pivote (`elim = lambda x: x > p`)

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\[(-1)\mathbf{2}+\mathbf{3}]}}\begin{bmatrix}0&1&0\\7&-3&3\\6&4&-2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\[(\frac{3}{7})\mathbf{1}+\mathbf{2}]\\[(\frac{-3}{7})\mathbf{1}+\mathbf{3}]}}\begin{bmatrix}0&1&0\\7&0&0\\6&\frac{46}{7}&\frac{-32}{7}\end{bmatrix}$$

- Reducción por eliminación *Gauss-Jordan*. El `Sentido` de la búsqueda de pivotes es de derecha a izquierda (`S = 'Normal'`) y consecuentemente las filas se recorren de arriba a abajo (`sentido = lambda x: x`). Se elimiman todos los componentes a la derecha e izquierda del pivote (`elim = lambda x: x != p`)

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\[(-1)\mathbf{2}+\mathbf{3}]}}\begin{bmatrix}0&1&0\\7&-3&3\\6&4&-2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\[(\frac{3}{7})\mathbf{1}+\mathbf{2}]\\[(\frac{-3}{7})\mathbf{1}+\mathbf{3}]}}\begin{bmatrix}0&1&0\\7&0&0\\6&\frac{46}{7}&\frac{-32}{7}\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\[(\frac{21}{16})\mathbf{3}+\mathbf{1}]\\[(\frac{23}{16})\mathbf{3}+\mathbf{2}]}}\begin{bmatrix}0&1&0\\7&0&0\\0&0&\frac{-32}{7}\end{bmatrix}$$

- Reducción por eliminación *Gauss-Opuesto*. El `Sentido` de la búsqueda de pivotes es el izquierda a derecha (`S = 'Opuesto'`) y consecuentemente las filas se recorren de abajo a arriba (`sentido = lambda x: reversed(x)`). Solo se elimiman los componentes a la izquierda del pivote (`elim = lambda x: x < p`)

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\[(-3)\mathbf{3}+\mathbf{1}]\\[(-2)\mathbf{3}+\mathbf{2}]}}\begin{bmatrix}-3&-1&1\\7&-3&0\\0&0&2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\[(\frac{7}{3})\mathbf{2}+\mathbf{1}]}}\begin{bmatrix}\frac{-16}{3}&-1&1\\0&-3&0\\0&0&2\end{bmatrix}$$

Cada variante estabece el valor de la variable `S` y el método auxiliar `elim`. Una vez hecho esto, los tres procedimentos usan el mismo código.

Todas las variantes poseen idéntica estructura:

- `A` es la matriz sobre la que se aplican las transformaciones elementales.
- `r` es el contador de pivotes encontrados al escalonar la matriz (inicialemente `0`).
- `transformaciones` es una lista que almacenará las abreviaturas de las transformaciones aplicadas sobre las columnas de la matriz (inicialmente vacía).
- `columnaOcupada` es el conjunto de índices de columnas con pivote en la matriz reducida (inicialmente vacío).

En función de si `S` toma el valor ['Normal'] u otro distinto se fijan los métodos auxiliares `sentido` que indicará si las filas se leen de arriba a abajo o en sentido contrario.

El bucle recorre las filas. El orden en el que se recorren las filas está determinado por el método auxiliar `sentido`. En cada iteración del bucle se busca la posición `p` de un nuevo pivote. Si se encuentra un pivote en la fila sobre la que se está trabajando (si `p` es distinto de `0]`), se aumenta en una unidad el contador `r` y se dan los pasos de eliminación necesarios. Se apuntan en `transformaciones`, se aplican a las columnas de `A` y se añade el índice `p` al conjunto `columnaOcupada`.

Como todas las transformaciones han sido sobre las columnas se guarda la lista `transformaciones` como segunda lista de `pasos` (la primera lista queda reservada para las operaciones sobre las filas y en este caso queda vacía: `[]`).

Por último se guardan los atributos `tex` (que permite la representación del proceso en Jupyter), `pasos` y `rank` que almacena el número de pivotes de la matriz reducida.

Reducción usando solo transformaciones Tipo I

50a
```
⟨Variantes que reducen una matriz por eliminación por columnas y guardan los pasos dados 50a⟩≡
  class RGC(Matrix):
      def __init__(self, data, rep=0):
          """Reduce una Matrix con eliminación por columnas (transf. Gauss)"""
          S       = 'Normal'
          elim    = lambda x: x > p
          ⟨Reducción de una matriz mediante transformaciones de las columnas 50b⟩

  class RGJC(Matrix):
      def __init__(self, data, rep=0):
          """Reduce una Matrix con eliminación por columnas (transf. Gauss)"""
          S       = 'Normal'
          elim    = lambda x: x != p
          ⟨Reducción de una matriz mediante transformaciones de las columnas 50b⟩

  class RGOC(Matrix):
      def __init__(self, data, rep=0):
          """Reduce una Matrix con eliminación por columnas (transf. Gauss)"""
          S       = 'Opuesto'
          elim    = lambda x: x < p
          ⟨Reducción de una matriz mediante transformaciones de las columnas 50b⟩
```

This definition is continued in chunks 51a and 52a.
This code is used in chunk 41.
Defines:
  `RGC`, used in chunks 57b and 66b.
  `RGJC`, never used.
  `RGOC`, never used.
Uses `Matrix` 13.

50b
```
⟨Reducción de una matriz mediante transformaciones de las columnas 50b⟩≡
  ⟨Método auxiliar de búsqueda de un nuevo pivote 45⟩
  A = Matrix(data);  r = 0;  transformaciones = [];  columnaOcupada = set()
  sentido = lambda x: x if S=='Normal' else reversed(x)
  for i in sentido(range(1,A.m+1)):
      p = BuscaNuevoPivote(i|A , Sentido=S);
      if p:
          r += 1
          ⟨Uso del pivote para eliminar componentes con trasformaciones Tipo I 47b⟩
          columnaOcupada.add(p)
  pasos = [[], transformaciones]
  ⟨Se guardan los atributos tex, pasos y rank (y se muestran los pasos si se pide) 87a⟩
  super(self.__class__ ,self).__init__(A.sis)
```
This code is used in chunk 50a.
Uses `BuscaNuevoPivote` 45 and `Matrix` 13.

Reducción evitando divisiones

51a ⟨*Variantes que reducen una matriz por eliminación por columnas y guardan los pasos dados* 50a⟩+≡

```
class RGCsd(Matrix):
    def __init__(self, data, rep=0):
        """Reduce una Matrix con eliminación por columnas (sin divisiones)"""
        S       = 'Normal'
        elim    = lambda x: x > p
        ⟨Reducción de una matriz mediante transformaciones de las columnas sin divisiones 51b⟩

class RGJCsd(Matrix):
    def __init__(self, data, rep=0):
        """Reduce una Matrix con eliminación por columnas (sin divisiones)"""
        S       = 'Normal'
        elim    = lambda x: x != p
        ⟨Reducción de una matriz mediante transformaciones de las columnas sin divisiones 51b⟩

class RGOCsd(Matrix):
    def __init__(self, data, rep=0):
        """Reduce una Matrix con eliminación por columnas (sin divisiones)"""
        S       = 'Opuesto'
        elim    = lambda x: x < p
        ⟨Reducción de una matriz mediante transformaciones de las columnas sin divisiones 51b⟩
```

This code is used in chunk 41.
Defines:
`RGCsd`, used in chunks 57b, 62, and 70a.
`RGJCsd`, never used.
`RGOCsd`, never used.
Uses `Matrix` 13.

51b ⟨*Reducción de una matriz mediante transformaciones de las columnas sin divisiones* 51b⟩≡

```
⟨Método auxiliar de búsqueda de un nuevo pivote 45⟩
A = Matrix(data);  r = 0;  transformaciones = [];  columnaOcupada = set()
sentido = lambda x: x if S=='Normal' else reversed(x)
for i in sentido(range(1,A.m+1)):
    p = BuscaNuevoPivote(i|A , Sentido=S);
    if p:
        r += 1
        ⟨Uso del pivote para eliminar componentes evitando dividir 48a⟩
        columnaOcupada.add(p)
pasos = [[], transformaciones]
⟨Se guardan los atributos tex, pasos y rank (y se muestran los pasos si se pide) 87a⟩
super(self.__class__ ,self).__init__(A.sis)
```

This code is used in chunk 51a.
Uses `BuscaNuevoPivote` 45 and `Matrix` 13.

Reducción normalizando pivotes antes de eliminar componentes

52a ⟨*Variantes que reducen una matriz por eliminación por columnas y guardan los pasos dados* 50a⟩+≡

```
class RGCN(Matrix):
    def __init__(self, data, rep=0):
        """Reduce una Matrix con eliminación por columnas (pivotes normalizados)"""
        S       = 'Normal'
        elim    = lambda x: x > p
        ⟨Reducción normalizada de una matriz mediante transformaciones de las columnas 52b⟩

class RGJCN(Matrix):
    def __init__(self, data, rep=0):
        """Reduce una Matrix con eliminación por columnas (pivotes normalizados)"""
        S       = 'Normal'
        elim    = lambda x: x != p
        ⟨Reducción normalizada de una matriz mediante transformaciones de las columnas 52b⟩

class RGOCN(Matrix):
    def __init__(self, data, rep=0):
        """Reduce una Matrix con eliminación por columnas (pivotes normalizados)"""
        S       = 'Opuesto'
        elim    = lambda x: x < p
        ⟨Reducción normalizada de una matriz mediante transformaciones de las columnas 52b⟩
```

This code is used in chunk 41.
Defines:
`RGJCN`, used in chunk 57b.
`RGNC`, never used.
`RGOCN`, never used.
Uses `Matrix` 13.

52b ⟨*Reducción normalizada de una matriz mediante transformaciones de las columnas* 52b⟩≡

```
⟨Método auxiliar de búsqueda de un nuevo pivote 45⟩
A = Matrix(data);  r = 0;  transformaciones = [];  columnaOcupada = set()
sentido = lambda x: x if S=='Normal' else reversed(x)
for i in sentido(range(1,A.m+1)):
    p = BuscaNuevoPivote(i|A , Sentido=S);
    if p:
        r += 1
        ⟨Normalizamos el pivote para que sea igual a uno 47a⟩
        ⟨Uso del pivote para eliminar componentes con trasformaciones Tipo I 47b⟩
        columnaOcupada.add(p)
pasos = [[], transformaciones]
⟨Se guardan los atributos tex, pasos y rank (y se muestran los pasos si se pide) 87a⟩
super(self.__class__ ,self).__init__(A.sis)
```

This code is used in chunk 52a.
Uses `BuscaNuevoPivote` 45 and `Matrix` 13.

2.1.3 Escalonamiento de una matriz

El escalonamiento requiere un paso adicional con respecto a la reducción, pues antes de eliminar (o de normalizar pivotes y eliminar) se coloca cada columna pivote (mediante un intercambio) en la posición adecuada para que la matriz describa una escalera descendentre con sus pivotes (⟨*Intercambio de vectores para escalonar los pivotes* 46⟩). La regla para colocar las columnas que contienen los pivotes es ligeramente distinta en el caso de la eliminación *Gauss-Opuesto*. Lo que requiere especificar el método auxiliar `pc` que determina la posición de la columna que contiene cada pivote.

- Escalonamiento por eliminación *Gaussiana*. El `Sentido` de la búsqueda de pivotes es de derecha a izquierda (`S = 'Normal'`) y consecuentemente las filas se recorren de arriba a abajo (`sentido = lambda x: x`). Además, solo se elimiman los componentes a la derecha del pivote (`elim = lambda x: x > p`)

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}\xrightarrow{\overset{\boldsymbol{\tau}}{[\mathbf{1}\rightleftharpoons\mathbf{2}]}}\begin{bmatrix}1&0&1\\-3&7&0\\4&6&2\end{bmatrix}\xrightarrow{\overset{\boldsymbol{\tau}}{[(-1)\mathbf{1}+\mathbf{3}]}}\begin{bmatrix}1&0&0\\-3&7&3\\4&6&-2\end{bmatrix}\xrightarrow{\overset{\boldsymbol{\tau}}{[(\frac{-3}{7})\mathbf{2}+\mathbf{3}]}}\begin{bmatrix}1&0&0\\-3&7&0\\4&6&\frac{-32}{7}\end{bmatrix}$$

- Escalonamiento por eliminación *Gauss-Jordan*. El `Sentido` de la búsqueda de pivotes es de derecha a izquierda (`S = 'Normal'`) y consecuentemente las filas se recorren de arriba a abajo (`sentido = lambda x: x`). Se elimiman todos los componentes a la derecha e izquierda del pivote (`elim = lambda x: x != p`)

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}\xrightarrow{\overset{\boldsymbol{\tau}}{[\mathbf{1}\rightleftharpoons\mathbf{2}]}}\begin{bmatrix}1&0&1\\-3&7&0\\4&6&2\end{bmatrix}\xrightarrow{\overset{\boldsymbol{\tau}}{[(-1)\mathbf{1}+\mathbf{3}]}}\begin{bmatrix}1&0&0\\-3&7&3\\4&6&-2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\ [(\frac{3}{7})\mathbf{2}+\mathbf{1}]\\ [(\frac{-3}{7})\mathbf{2}+\mathbf{3}]}}\begin{bmatrix}1&0&0\\0&7&0\\\frac{46}{7}&6&\frac{-32}{7}\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\ [(\frac{23}{16})\mathbf{3}+\mathbf{1}]\\ [(\frac{21}{16})\mathbf{3}+\mathbf{2}]}}\begin{bmatrix}1&0&0\\0&7&0\\0&0&\frac{-32}{7}\end{bmatrix}$$

- Escalonamiento por eliminación *Gauss-Opuesto*. El `Sentido` de la búsqueda de pivotes es el izquierda a derecha (`S = 'Opuesto'`) y consecuentemente las filas se recorren de abajo a arriba (`sentido = lambda x: reversed(x)`). Solo se elimiman los componentes a la izquierda del pivote (`elim = lambda x: x < p`)

$$\begin{bmatrix}0&1&1\\7&-3&0\\6&4&2\end{bmatrix}\xrightarrow{\substack{\boldsymbol{\tau}\\ [(-3)\mathbf{3}+\mathbf{1}]\\ [(-2)\mathbf{3}+\mathbf{2}]}}\begin{bmatrix}-3&-1&1\\7&-3&0\\0&0&2\end{bmatrix}\xrightarrow{\overset{\boldsymbol{\tau}}{[(\frac{7}{3})\mathbf{2}+\mathbf{1}]}}\begin{bmatrix}\frac{-16}{3}&-1&1\\0&-3&0\\0&0&2\end{bmatrix}$$

Como en el caso de la reducción, cada variante de escalonamiento estabece el valor de la variable `S` y el método auxiliar `elim`. Una vez hecho esto, los tres procedimentos usan el mismo código.

Escalonamiento usando solo transformaciones Tipo I

54a ⟨*Escalonamiento de una matriz mediante transformaciones de las columnas* 54a⟩≡

```
⟨Método auxiliar de búsqueda de un nuevo pivote 45⟩
A = Matrix(data);  r = 0;  transformaciones = [];  columnaOcupada = set()
sentido = lambda x: x if S=='Normal' else reversed(x)
pc      = lambda x: x if S=='Normal' else (A.n+1)-x
for i in sentido(range(1,A.m+1)):
    p = BuscaNuevoPivote(i|A , Sentido=S);
    if p:
        r += 1
        ⟨Intercambio de vectores para escalonar los pivotes 46⟩
        ⟨Uso del pivote para eliminar componentes con trasformaciones Tipo I 47b⟩
        columnaOcupada.add(p)
pasos = [[], transformaciones]
⟨Se guardan los atributos tex, pasos y rank (y se muestran los pasos si se pide) 87a⟩
super(self.__class__ ,self).__init__(A.sis)
```

This code is used in chunk 54b.
Uses `BuscaNuevoPivote` 45 and `Matrix` 13.

54b ⟨*Variantes que escalonan una matriz por eliminación por columnas y guardan los pasos dados* 54b⟩≡

```
class EGC(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
        S       = 'Normal'
        elim    = lambda x: x > p
        ⟨Escalonamiento de una matriz mediante transformaciones de las columnas 54a⟩

class EGJC(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
        S       = 'Normal'
        elim    = lambda x: x != p
        ⟨Escalonamiento de una matriz mediante transformaciones de las columnas 54a⟩

class EGOC(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
        S       = 'Opuesto'
        elim    = lambda x: x < p
        ⟨Escalonamiento de una matriz mediante transformaciones de las columnas 54a⟩
```

This definition is continued in chunks 55b and 56b.
This code is used in chunk 41.
Defines:
`EGC`, used in chunk 58.
`EGJC`, never used.
`EGOC`, used in chunk 58.
Uses `Matrix` 13.

Escalonamiento evitando divisiones

55a ⟨*Escalonamiento de una matriz mediante transformaciones de las columnas sin divisiones* 55a⟩≡

```
⟨Método auxiliar de búsqueda de un nuevo pivote 45⟩
A = Matrix(data);  r = 0;  transformaciones = [];  columnaOcupada = set()
sentido = lambda x: x if S=='Normal' else reversed(x)
pc      = lambda x: x if S=='Normal' else (A.n+1)-x
for i in sentido(range(1,A.m+1)):
    p = BuscaNuevoPivote(i|A , Sentido=S);
    if p:
        r += 1
        ⟨Intercambio de vectores para escalonar los pivotes 46⟩
        ⟨Uso del pivote para eliminar componentes evitando dividir 48a⟩
        columnaOcupada.add(p)
pasos = [[], transformaciones]
⟨Se guardan los atributos tex, pasos y rank (y se muestran los pasos si se pide) 87a⟩
super(self.__class__ ,self).__init__(A.sis)
```

This code is used in chunk 55b.
Uses `BuscaNuevoPivote` 45 and `Matrix` 13.

55b ⟨*Variantes que escalonan una matriz por eliminación por columnas y guardan los pasos dados* 54b⟩+≡

```
class EGCsd(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (sin dividir)"""
        S       = 'Normal'
        elim    = lambda x: x > p
        ⟨Escalonamiento de una matriz mediante transformaciones de las columnas sin divisiones 55a⟩

class EGJCsd(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (sin dividir)"""
        S       = 'Normal'
        elim    = lambda x: x != p
        ⟨Escalonamiento de una matriz mediante transformaciones de las columnas sin divisiones 55a⟩

class EGOCsd(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (sin dividir)"""
        S       = 'Opuesto'
        elim    = lambda x: x < p
        ⟨Escalonamiento de una matriz mediante transformaciones de las columnas sin divisiones 55a⟩
```

This code is used in chunk 41.
Defines:
`EGCsd`, used in chunks 58, 61b, and 65–67.
`EGJCsd`, never used.
`EGOCsd`, never used.
Uses `Matrix` 13.

Escalonamiento normalizando pivotes antes de eliminar componentes

56a ⟨*Escalonamiento normalizado de una matriz mediante transformaciones de las columnas* 56a⟩≡

```
⟨Método auxiliar de búsqueda de un nuevo pivote 45⟩
A = Matrix(data);  r = 0;  transformaciones = [];  columnaOcupada = set()
sentido = lambda x: x if S=='Normal' else reversed(x)
pc      = lambda x: x if S=='Normal' else (A.n+1)-x
for i in sentido(range(1,A.m+1)):
    p = BuscaNuevoPivote(i|A , Sentido=S);
    if p:
        r += 1
        ⟨Intercambio de vectores para escalonar los pivotes 46⟩

        ⟨Normalizamos el pivote para que sea igual a uno 47a⟩
        ⟨Uso del pivote para eliminar componentes con trasformaciones Tipo I 47b⟩
        columnaOcupada.add(p)
pasos = [[], transformaciones]
⟨Se guardan los atributos tex, pasos y rank (y se muestran los pasos si se pide) 87a⟩
super(self.__class__ ,self).__init__(A.sis)
```

This code is used in chunk 56b.
Uses `BuscaNuevoPivote` 45 and `Matrix` 13.

56b ⟨*Variantes que escalonan una matriz por eliminación por columnas y guardan los pasos dados* 54b⟩+≡

```
class EGCN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (pivotes normalizados)"""
        S       = 'Normal'
        elim    = lambda x: x > p
        ⟨Escalonamiento normalizado de una matriz mediante transformaciones de las columnas 56a⟩

class EGJCN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (pivotes normalizados)"""
        S       = 'Normal'
        elim    = lambda x: x != p
        ⟨Escalonamiento normalizado de una matriz mediante transformaciones de las columnas 56a⟩

class EGOCN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (pivotes normalizados)"""
        S       = 'Opuesto'
        elim    = lambda x: x < p
        ⟨Escalonamiento normalizado de una matriz mediante transformaciones de las columnas 56a⟩
```

This code is used in chunk 41.
Defines:
- `EGCN`, used in chunks 58 and 60b.
- `EGJCN`, used in chunks 58 and 59.
- `EGOCN`, never used.

Uses `Matrix` 13.

2.1.4 Reducción o escalonamiento operando con las filas

Para aplicar la eliminación sobre las filas, no necesitamos reprogramar cada uno de los algoritmos anteriores, basta con usarlos con las columnas de la matriz transpuesta (`~Matrix(data)`).

Una vez hecho esto, basta recuperar el rango (`r = A.rank`) y almacenar correctamente las transformaciones aplicadas: la lista de `pasos[1]` dados sobre las columnas de la transpuesta se debe almacenar en el orden inverso como *primera* lista de `pasos`. Por último de devuelve la transpuesta: `(~A).sis`.

57a
```
⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩≡
  r = A.rank
  pasos = [ list(reversed(A.pasos[1])) , [] ]
  ⟨Se guardan los atributos tex, pasos y rank (y se muestran los pasos si se pide) 87a⟩
  super(self.__class__ ,self).__init__((~A).sis)
```
This code is used in chunks 57b and 58.

Reducción de una matriz por eliminación por filas

57b
```
⟨Variantes que reducen una matriz por eliminación por filas y guardan los pasos dados 57b⟩≡
  class RGF(Matrix):
      def __init__(self, data, rep=0):
          """Reduce una Matrix con eliminación por filas (transf. Gauss)"""
          A = RGC(~Matrix(data))

  class RGFsd(Matrix):
      def __init__(self, data, rep=0):
          """Reduce una Matrix con eliminación por filas (sin dividir)"""
          A = RGCsd(~Matrix(data))
          ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩

  class RGFN(Matrix):
      def __init__(self, data, rep=0):
          """Reduce una Matrix con eliminación por filas (pivotes normalizados)"""
          A = RGCN(~Matrix(data))
          ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩

  class RGJFN(Matrix):
      def __init__(self, data, rep=0):
          """Reduce una Matrix con eliminación por filas (pivotes normalizados)"""
          A = RGJCN(~Matrix(data))
          ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩
```
This code is used in chunk 41.
Defines:
- `RGF`, never used.
- `RGFN`, never used.
- `RGFsd`, never used.
- `RGJFN`, never used.

Uses `Matrix` 13, `RGC` 50a, `RGCsd` 51a, and `RGJCN` 52a.

Escalonamiento de una matriz por eliminación por filas

58 ⟨*Variantes que escalonan una matriz por eliminación por filas y guardan los pasos dados* 58⟩≡

```
class EGF(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por filas (transf. Gauss)"""
        A = EGC(~Matrix(data))
        ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩

class EGFsd(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por filas (sin dividir)"""
        A = EGCsd(~Matrix(data))
        ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩

class EGFN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por filas (pivotes normalizados)"""
        A = EGCN(~Matrix(data))
        ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩

class EGJFN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por filas (pivotes normalizados)"""
        A = EGJCN(~Matrix(data))
        ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩

class EGOF(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por filas (transf. Gauss)"""
        A = EGOC(~Matrix(data))
        ⟨Traducción de los pasos dados para verlos como eliminación por filas 57a⟩
```

This code is used in chunk 41.
Defines:
`EGF`, used in chunk 60b.
`EGFN`, never used.
`EGFsd`, never used.
`EGJFN`, used in chunk 60a.
`EGOF`, never used.
Uses `EGC` 54b, `EGCN` 56b, `EGCsd` 55b, `EGJCN` 56b, `EGOC` 54b, and `Matrix` 13.

2.2 Inversión de una matriz por eliminación Gaussiana

Para invertir una matriz basta aplicar a una matriz *Gauss-Jordan* normalizando los pivotes. Los pasos dados aplicados sobre la matriz identidad nos dan la inversa de la matriz.

59a
```
⟨Método auxiliar que calcula la inversa de una Matrix 59a⟩≡
  def MatrixInversa( self ):
      """Calculo de la inversa de una matriz"""
      L = EGJCN(self)
      if L.rank < L.n:  raise ArithmeticError('Matrix singular')
      return Matrix( I(L.n) & T(L.pasos[1]) )
```

This code is used in chunk 29b.
Uses EGJCN 56b, Matrix 13, and T 37b.

El siguiente código obtiene la inversa de una matriz cuadrada operando sobre las columnas, y muestra los pasos dados hasta llegar a ella, o hasta llegar a una matriz singular

59b
```
⟨Invirtiendo una matriz 59b⟩≡
  class InvMat(Matrix):
      def __init__(self, data, rep=0):
          """Devuelve la matriz inversa y los pasos dados sobre las columnas"""
          ⟨Método auxiliar que crea el atributo tex y muestra los pasos en Jupyter 87b⟩
          A           = Matrix(data)
          if A.m != A.n:
              raise ValueError('Matrix no cuadrada')
          M           = EGJCN(A)
          if M.rank < A.n:
              raise ArithmeticError('Matrix singular')
          Inv         = I(A.n) & T(M.pasos[1])
          self.pasos = M.pasos
          self.tex    = tex( BlockMatrix([ [A], [I(A.n)] ]) , self.pasos)
          super(self.__class__ ,self).__init__(Inv.sis)
```

This definition is continued in chunk 60.
This code is used in chunk 41.
Uses BlockMatrix 92b, EGJCN 56b, Matrix 13, and T 37b.

El siguiente código obtiene la inversa de una matriz operando sobre las filas, y muestra los pasos dados hasta llegar a ella

60a
```
⟨Invirtiendo una matriz 59b⟩+≡
  class InvMatF(Matrix):
      def __init__(self, data, rep=0):
          """Devuelve la matriz inversa y los pasos dados sobre las filas"""
          ⟨Método auxiliar que crea el atributo tex y muestra los pasos en Jupyter 87b⟩
          A          = Matrix(data)
          if A.m != A.n:
              raise ValueError('Matrix no cuadrada')
          M          = EGJFN(A)
          if M.rank < A.n:
              raise ArithmeticError('Matrix singular')
          Inv        = T(M.pasos[0]) & I(A.n)
          self.pasos = M.pasos
          self.tex   = tex( BlockMatrix([ [A,I(A.m)] ]) , self.pasos)
          super(self.__class__ ,self).__init__(Inv.sis)
```

This code is used in chunk 41.
Uses `BlockMatrix` 92b, `EGJFN` 58, `Matrix` 13, and `T` 37b.

El siguiente código obtiene la inversa de una matriz operando primero sobre las filas hasta obtener una matriz trianguar superior y luego operando sobre las columnas hasta obtener la identidad.

60b
```
⟨Invirtiendo una matriz 59b⟩+≡
  class InvMatFC(Matrix):
      def __init__(self, data, rep=0):
          """Devuelve la matriz inversa y los pasos dados sobre las filas y columnas"""
          ⟨Definición del método PasosYEscritura 88⟩
          ⟨Método auxiliar que crea el atributo tex y muestra los pasos en Jupyter 87b⟩
          A          = Matrix(data)
          if A.m != A.n:
              raise ValueError('Matrix no cuadrada')
          M          = EGCN(EGF(A))
          if M.rank < A.n:
              raise ArithmeticError('Matrix singular')
          Inv        = ( I(A.n) & T(M.pasos[1]) ) * ( T(M.pasos[0]) & I(A.n) )
          self.pasos = M.pasos
          self.tex = tex(BlockMatrix([ [A,I(A.m)], [I(A.n),M0(A.m,A.n)] ]),self.pasos)
          super(self.__class__ ,self).__init__(Inv.sis)
```

This code is used in chunk 41.
Uses `BlockMatrix` 92b, `EGCN` 56b, `EGF` 58, `Matrix` 13, and `T` 37b.

2.3 Resolución de un sistema de ecuaciones homogeneo por eliminación Gaussiana

El siguiente código devuelve el conjunto de soluciones de un sistema homogéneo $\mathbf{A}\boldsymbol{x} = \mathbf{0}$. Descripción de los atributos:

- `sgen` es un sistema generador del espacio nulo $\mathcal{N}(\mathbf{A})$.
- `determinado` indica si es cierto que el sistema es determinado (una única solución)
- `tex` es la cadena de texto LaTeX que permite representar los pasos dados para resolver el sistema.

61a

```
⟨Resolviendo un sistema homogeneo 61a⟩≡
  class Homogenea:
      def __init__(self, data, rep=0):
          """Resuelve un Sistema de Ecuaciones Lineales Homogeneo

          y muestra los pasos para encontrarlo"""
          ⟨Método auxiliar que crea el atributo tex y muestra los pasos en Jupyter 87b⟩

          A      = Matrix(data)
          ⟨Cálculo de L y de una base y dimensión (dim) del espacio nulo de A 61b⟩

          self.sgen        = Sistema(base) if dim else Sistema([V0(A.n)])
          self.determinado = (dim == 0)
          self.pasos       = L.pasos
          self.tex         = tex( BlockMatrix([[A],[I(A.n)]]), self.pasos)
          self.enulo       = SubEspacio(self.sgen)

      ⟨Métodos de representación de la clase Homogenea 89⟩
```

This code is used in chunk 41.
Defines:
`Homogenea`, used in chunk 65.
Uses `BlockMatrix` 92b, `Matrix` 13, `Sistema` 7, `SubEspacio` 66a, and `V0` 83c.

61b

```
⟨Cálculo de L y de una base y dimensión (dim) del espacio nulo de A 61b⟩≡
  L      = EGCsd( A )
  E      = I(A.n) & T(L.pasos[1])
  base   = [Vector(E|j) for j in range(1, L.n+1) if Vector(L|j).esNulo()]
  dim    = len(base)
```

This code is used in chunks 61a and 62.
Uses `EGCsd` 55b, `T` 37b, and `Vector` 9b.

62 ⟨*Resolviendo un Sistema de Ecuaciones Lineales* 62⟩≡

```
class SEL:
    def __init__(self, A, b, rep=0):
        """Resuelve un Sistema de Ecuaciones Lineales

        y muestra los pasos para encontrarlo"""
        ⟨Método auxiliar que crea el atributo tex y muestra los pasos en Jupyter 87b⟩

        ⟨Cálculo de L y de una base y dimensión (dim) del espacio nulo de A 61b⟩
        self.sgen        = Sistema(base) if dim else Sistema([V0(A.n)])
        self.determinado = (dim == 0)

        MA = Matrix( BlockMatrix([ [A, -Matrix([b])] ]) )
        LA = RGCsd ( Matrix(MA) & T(L.pasos[1]) )

        if not (LA|0).esNulo():
            BM       = Matrix(BlockMatrix([[MA],[I(MA.n)]]))
            self.tex = tex( {A.m,A.m+A.n}|BM|{A.n} , [[], L.pasos[1]+LA.pasos[1]] )
            raise ArithmeticError('No hay solución: Sistema incompatible')

        EA               = I(MA.n) & T(L.pasos[1]) & T(LA.pasos[1])
        Normalizacion    = T([(Fraction(1, 0|EA|0 ), MA.n)])
        EA               = EA & Normalizacion
        self.solP        = list(range(1,A.n+1))|EA|0

        self.pasos = [ [], L.pasos[1] + LA.pasos[1] + [Normalizacion] ]
        BM         = Matrix(BlockMatrix([ [MA],[I(MA.n)] ]))
        self.tex   = tex({A.m,A.m+A.n}|BM|{A.n}, self.pasos)
        self.eafin = EAfin(self.sgen,self.solP)

    ⟨Métodos de representación de la clase SEL 90⟩
```

This code is used in chunk 41.
Defines:
`SEL`, used in chunk 72a.
Uses `BlockMatrix` 92b, `EAfin` 70a, `Matrix` 13, `RGCsd` 51a, `Sistema` 7, `T` 37b, and `V0` 83c.

63a ⟨*normal* 63a⟩≡

```
class Normal(Matrix):
    def __init__(self, data):
        """Escalona por Gauss obteniendo una matriz cuyos pivotes son unos"""
        pivote=lambda v,k=0:([i for i,c in enumerate(v.sis,1)if(c!=0 and i>k)]+[0])[0]

        A = Matrix(data); r = 0
        self.rank = []
        for i in range(1,A.n+1):
            p = pivote((i|A),r)
            if p > 0:
                r += 1
                A & T( {p, r} )
                A & T( (1/Fraction(i|A|r), r) )
                A & T( [ (-(i|A|k), r, k) for k in range(r+1,A.n+1)] )

            self.rank+=[r]

        super(self.__class__ ,self).__init__(A.sis)
```

This code is used in chunk 41.
Uses `Matrix` 13 and `T` 37b.

Con este código ya podemos hacer muchísimas cosas. Por ejemplo, ¡eliminación gaussiana para encontrar el espacio nulo de una matriz!

63b ⟨*sistema* 63b⟩≡

```
def homogenea(A):
     """Devuelve una BlockMatriz con la solución del problema homogéneo"""
     stack=Matrix(BlockMatrix([[A],[I(A.n)]]))
     soluc=Normal(stack)
     col=soluc.rank[A.m-1]
     return {A.m} | soluc | {col}
```

This code is used in chunk 41.
Uses `BlockMatrix` 92b and `Matrix` 13.

Capítulo 3

Las clases `SubEspacio` y `EAfin`

El conjunto de vectores $\boldsymbol{x}$ que resuelven el sistema $\mathbf{A}\boldsymbol{x} = \mathbf{0}$ es un subespacio de $\mathbb{R}^n$; y el conjunto de vectores $\boldsymbol{x}$ que resuelven el sistema $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ con $\boldsymbol{b} \neq \mathbf{0}$ es un espacio afín de $\mathbb{R}^n$. En este capítulo vamos a definir objetos que que representen estos subconjuntos de $\mathbb{R}^n$.

3.1 La clase `SubEspacio` (de $\mathbb{R}^m$)

La clase `SubEspacio` se puede instanciar tanto con un `Sistema` de `Vector`es de $\mathbb{R}^m$ como con una `Matrix` de orden m por n. Dado un `Sistema` de vectores, por ejemplo

$$S = \left[\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} ; \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} ; \begin{pmatrix} 2 \\ 10 \\ 3 \end{pmatrix} \right]$$

`SubEspacio( S )` corresponde al conjunto de combinaciones lineales de los `Vector`es de dicho `Sistema`, representado por las siguientes ecuaciones *paramétricas*:

$$\left\{ \boldsymbol{v} \in \mathbb{R}^3 \;\middle|\; \exists \boldsymbol{p} \in \mathbb{R}^2 \text{ tal que } \boldsymbol{v} = \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 3 & 0 \end{bmatrix} \boldsymbol{p} \right\}$$

donde el vector $\boldsymbol{p}$ es el vector de parámetros.

Y dada una `Matrix`, por ejemplo $\mathbf{M} = \begin{bmatrix} -3 & 0 & 2 \\ 6 & 0 & -4 \end{bmatrix}$, El obejto `SubEspacio( M )` corresponde al conjunto de `Vector`es que son solución al sistema de ecuaciones cartesianas:

$$\left\{ \boldsymbol{v} \in \mathbb{R}^3 \;\middle|\; \begin{bmatrix} -3 & 0 & 2 \end{bmatrix} \boldsymbol{v} = \mathbf{0} \right\}$$

En el caso de estos dos ejemplos, ambos conjuntos son el mismo subespacio de $\mathbb{R}^3$; y, de hecho, la representación de `SubEspacio` muestra ambasrepresentaciones, tanto con ecuaciones paramétricas, como con cartesianas. `Subespacio` tiene varios atributos.

- `dim`: dimensión del subespacio. En el ejemplo `dim=2`.
- `Rm`: indica el espacio vectorial $\mathbb{R}^m$ al que pertenece `SubEspacio(S)`. En el ejemplo anterior `Rm=3` puesto que es un subespacio de $\mathbb{R}^3$.
- `base`: una base del subespacio (un `Sistema` de `Vector`es de `Rm`). Cuando `dim==0` base es un `Sistema` vacío.
- `sgen`: Un `Sistema` de `Vector`es generador del subespacio. En particular será el sistema de vectores correspondiente a la `Matrix` de coeficientes empleada en la representación con ecuaciones paramétricas. En el ejemplo

$$\left[\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} ; \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right].$$

- `cart`: `Matrix` de coeficientes empleada en la representación mediante un sistema de ecuaciones cartesianas. En el ejemplo

$$\begin{bmatrix} -3 & 0 & 2 \end{bmatrix}.$$

Implementación . La implementación requiere encontrar un Sistema de Vectores linealmente independientes y formar con ellos un Sistema generador del SubEspacio. Lo haremos escalonando una Matrix con el algoritmo ECLsd sencillamente para evitar fracciones y obtener una matriz triangular, pero esta decisión es completamente arbitraria. Además, también necesitaremos resolver un sistema de ecuaciones homogéneo. Lo haremos con el algorítmo Homogenea.

65 ⟨*Inicialización de la clase* SubEspacio *Viejo* 65⟩≡

```
def __init__(self,data):
    """Inicializa un SubEspacio de Rn"""
    if not isinstance(data, (Sistema, Matrix)):
        raise ValueError(' Argumento debe ser un Sistema o Matrix ')
    if isinstance(data, Sistema):
        A           = Matrix(data)
        L           = EGCsd(A)
        self.dim    = L.rank
        self.base   = Sistema([L|j for j in range(1,L.rank+1)])
        self.sgen   = self.base if L.rank else Sistema([V0(A.m)])
        self.cart   = ~Matrix(Homogenea(~A).sgen)
        self.Rn     = A.m
    if isinstance(data, Matrix):
        A           = data
        H           = Homogenea(A)
        self.sgen   = H.sgen
        self.dim    = 0 if H.determinado else len(self.sgen)
        self.base   = self.sgen if self.dim else Sistema([])
        self.cart   = ~Matrix(Homogenea(~Matrix(self.sgen)).sgen)
        self.Rn     = A.n
```

Root chunk (not used in this document).
Uses EGCsd 55b, Homogenea 61a, Matrix 13, Sistema 7, SubEspacio 66a, and V0 83c.

66a ⟨*Inicialización de la clase* SubEspacio 66a⟩≡
```
def __init__(self,data):
    """Inicializa un SubEspacio de Rn"""
    ⟨Método auxiliar SGenENulo que Encuentra un sistema generador del Espacio Nulo de A 66b⟩
    if not isinstance(data, (Sistema, Matrix)):
        raise ValueError(' Argumento debe ser un Sistema o Matrix ')
    if isinstance(data, Sistema):
        A           = Matrix(data)
        L           = EGCsd(A)
        self.dim    = L.rank
        self.base   = Sistema([L|j for j in range(1,L.rank+1)])
        self.sgen   = self.base if L.rank else Sistema([V0(A.m)])
        self.cart   = ~Matrix(SGenENulo(~A))
        self.Rn     = A.m
    if isinstance(data, Matrix):
        A           = data
        self.sgen   = SGenENulo(A)
        self.dim    = 0 if self.sgen.lista[0].esNulo() else len(self.sgen)
        self.base   = self.sgen if self.dim else Sistema([])
        self.cart   = ~Matrix(SGenENulo(~Matrix(self.sgen)))
        self.Rn     = A.n
```
This code is used in chunk 69c.
Defines:
`SubEspacio`, used in chunks 61a, 65, 67–73, 90, and 91.
Uses `EGCsd` 55b, `Matrix` 13, `Sistema` 7, and `V0` 83c.

66b ⟨*Método auxiliar* SGenENulo *que Encuentra un sistema generador del Espacio Nulo de A* 66b⟩≡
```
def SGenENulo(A):
    """Encuentra un sistema generador del Espacio Nulo de A"""
    R = RGC(A)
    S = Sistema([(I(A.n) & T(R.pasos[1]))|j for j in range(1,A.n+1) \
                                                  if (R|j).esNulo() ])
    return Sistema([V0(A.n)]) if R.rank==A.n else S
```
This code is used in chunk 66a.
Uses `RGC` 50a, `Sistema` 7, `T` 37b, and `V0` 83c.

Definimos un método que nos indique si es cierto que un `SubEspacio` está contenido en otro (`contenidoEn`). Si `A` y `B` son `SubEspacio`s, la siguiente expresión

`A.contenidoEn(B)`

nos dirá si es cierto que `A` es un `SubEspacio` de `B` (fíjese que como "`contenidoEn`" no es un "Método Mágico" de Python,, se debe invocar escribiendo `A.contenidoEn()`, donde `A` es un `SubEspacio`.

Para comprobar si está contenido, basta con concatenar las matrices formadas por el sistema generador de `A` y el sistema generador de `B` respectivamente, y escalonar dicha matriz. Si su rango resulta ser igual a la dimensión de `B`, entonces significa que los vectores de `A` pertenecen a `B`.

67a ⟨*Métodos de la clase* SubEspacio 67a⟩≡

```
  def contenidoEn(self, other):
      """Indica si este SubEspacio está contenido en other"""
      self.verificacion(other)
      if isinstance(other, SubEspacio):
          #return all ([(other.cart*w).esNulo() for w in self.sgen])
          M = Matrix(BlockMatrix([[Matrix(other.sgen), Matrix( self.sgen)]]))
          return True if (EGCsd(M).rank == other.dim) else False
      else:
          return other.v.esNulo() and self.contenidoEn(other.S)
```

This definition is continued in chunks 67–69.
This code is used in chunk 69c.
Uses `BlockMatrix` 92b, `EGCsd` 55b, `Matrix` 13, and `SubEspacio` 66a.

También definimos dos métodos (mágicos) que nos indican

- si dos SubEspacios son iguales (`__eq__`), es decir, que `A` esta contenido en `B` y viceversa; o
- si son distintos (`__ne__`), es decir, que no son iguales.

Así podemos usar las siguientes expresiones boleanas

`A == B` y `A != B`

67b ⟨*Métodos de la clase* SubEspacio 67a⟩+≡

```
  def __eq__(self, other):
      """Indica si un subespacio de Rn es igual a otro"""
      self.verificacion(other)
      return self.contenidoEn(other) and other.contenidoEn(self)

  def __ne__(self, other):
      """Indica si un subespacio de Rn es distinto de otro"""
      self.verificacion(other)
      return not (self == other)
```

This code is used in chunk 69c.

Para que estos tres métodos funcionen es necesario un método auxiliar que realice la `verificacion` de que los dos argumentos son SubEspacios del mismo espacio vectorial $\mathbb{R}^m$ (como este método tampoco es mágico, se invoca con `self.verificacion()`).

68a ⟨*Métodos de la clase* SubEspacio 67a⟩+≡
```
def verificacion(self,other):
    if not isinstance(other, (SubEspacio, EAfin)) or  not self.Rn == other.Rn:
        raise \
         ValueError('Ambos argumentos deben ser subconjuntos de en un mismo espacio')
```

This code is used in chunk 69c.
Uses EAfin 70a and SubEspacio 66a.

También definimos un método que nos devuelva la suma de dos SubEspacios de $\mathbb{R}^m$: `A + B`

68b ⟨*Métodos de la clase* SubEspacio 67a⟩+≡
```
def __add__(self, other):
    """Devuelve la suma de subespacios de Rn"""
    self.verificacion(other)
    return SubEspacio(Sistema(self.sgen + other.sgen))
```

This code is used in chunk 69c.
Uses Sistema 7 and SubEspacio 66a.

y otro que nos devuelva la intersección: `A & B`

68c ⟨*Métodos de la clase* SubEspacio 67a⟩+≡
```
def __and__(self, other):
    """Devuelve la intersección de subespacios"""
    self.verificacion(other)
    M = Matrix(BlockMatrix([ [ self.cart], [other.cart] ]))
    return SubEspacio(M)
```

This code is used in chunk 69c.
Uses BlockMatrix 92b, Matrix 13, and SubEspacio 66a.

Con `~A` obtendremos el complemento ortogonal del SubEspacio `A`

69a
```
⟨Métodos de la clase SubEspacio 67a⟩+≡
  def __invert__(self):
      """Devuelve el complemento ortogonal"""
      return SubEspacio(Sistema((~self.cart).sis))
```
This code is used in chunk 69c.
Uses Sistema 7 and SubEspacio 66a.

y por último definimos un método que nos indique si un `Vector` `x` pertence a un `SubEspacio` `A`, es decir, que indique si es cierta o no la siguiente expresión boleana

`x in A`

69b
```
⟨Métodos de la clase SubEspacio 67a⟩+≡
  def __contains__(self, other):
      """Indica si un Vector está pertenece a un SubEspacio"""
      if not isinstance(other, Vector) or other.n != self.cart.n:
          raise ValueError\
                ('Es necesario un Vector con el número adecuado de componentes')
      return (self.cart*other == V0(self.cart.m))
```
This code is used in chunk 69c.
Uses SubEspacio 66a, V0 83c, and Vector 9b.

69c
```
⟨La clase SubEspacio 69c⟩≡
  class SubEspacio:
      ⟨Inicialización de la clase SubEspacio 66a⟩
      ⟨Métodos de la clase SubEspacio 67a⟩
      ⟨Métodos de representación de la clase SubEspacio 91⟩
```
This code is used in chunk 41.
Uses SubEspacio 66a.

3.2 La clase `EAfin` (de $\mathbb{R}^m$)

El coinjunto de soluciones de un sistema de ecuaciones homoéneo $\mathbf{A}\boldsymbol{x} = \mathbf{0}$ forma un subespacio (que llamamos espacio nulo $\mathcal{N}(\mathbf{A})$), pero el conjunto de soluciones de $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ cuando $\boldsymbol{b} \neq \mathbf{0}$ es un *espacio afín*.

Vamos a crear la clase `EAfin`. La definiremos como un par $(\boldsymbol{v}, \mathcal{S})$ cuyo primer elemento, $\boldsymbol{v}$, sea un vector del espacio afín (una solución particular de $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$) y el segundo elemento, $\mathcal{S}$, sea un `SubEspacio` (el conjunto de soluciones a $\mathbf{A}\boldsymbol{x} = \mathbf{0}$). En el atributo `v` guardaremos el `Vector` y en el atributo `S` el `SubEspacio`. Así, pues, para instanciar un `EAfin` usaremos dos argumentos: el primero será un `Sistema` o `Matrix` con la que formar el `SubEspacio`, y elsegundo será un `Vector`.

Cuando $\boldsymbol{v} \in \mathcal{S}$, el espacio afín es un subespacio (que por tanto contiene al vector nulo). Así que si $\boldsymbol{v} \in \mathcal{S}$ en el atributo `v` guardaremos el vector nulo, en lugar del vector dado como argumento.

70a ⟨*Inicialización de la clase* `EAfin` 70a⟩≡

```
def __init__(self,data,v):
    """Inicializa un Espacio Afín de Rn"""
    self.S  = data if isinstance(data, SubEspacio) else SubEspacio(data)
    if not isinstance(v, Vector) or v.n != self.S.Rn:
         raise ValueError('v y SubEspacio deben estar en el mismo espacio vectorial')
    MA      = Matrix( BlockMatrix([ [ Matrix(self.S.sgen), Matrix([v]) ] ]) )
    self.v  = RGCsd( MA )|0
    self.Rn = self.S.Rn
```

This code is used in chunk 70b.
Defines:
`EAfin`, used in chunks 62, 68a, and 70–72.
Uses `BlockMatrix` 92b, `Matrix` 13, `RGCsd` 51a, `SubEspacio` 66a, and `Vector` 9b.

70b ⟨*La clase EAfin* 70b⟩≡

```
class EAfin:
    ⟨Inicialización de la clase EAfin 70a⟩
    ⟨Métodos de la clase EAfin 70c⟩
    ⟨Métodos de representación de la clase EAfin 73⟩
```

This code is used in chunk 41.
Uses `EAfin` 70a.

Un vector $\boldsymbol{x}$ pertenece al espacio afín $\mathcal{S}$ si verifica las ecuaciones cartesianas, cuya matriz de coeficientes es `self.S.cart`, y cuyo vector del lado derecho es `(self.S.cart)*self.v`. Así pues

70c ⟨*Métodos de la clase* `EAfin` 70c⟩≡

```
def __contains__(self, other):
    """Indica si un Vector pertenece a un EAfin"""
    if not isinstance(other, Vector) or other.n != self.S.cart.n:
        raise ValueError('Vector con un número inadecuado de componentes')
    return (self.S.cart)*other == (self.S.cart)*self.v
```

This definition is continued in chunks 71 and 72.
This code is used in chunk 70b.
Uses `EAfin` 70a and `Vector` 9b.

71a ⟨*Métodos de la clase* EAfin 70c⟩+≡
```
def contenidoEn(self, other):
    """Indica si este EAfin está contenido en other"""
    self.verificacion(other)
    if isinstance(other, SubEspacio):
         return self.v in other and self.S.contenidoEn(other)
    else:
         return self.v in other and self.S.contenidoEn(other.S)
```

This code is used in chunk 70b.
Uses `EAfin` 70a and `SubEspacio` 66a.

71b ⟨*Métodos de la clase* EAfin 70c⟩+≡
```
def __eq__(self, other):
    """Indica si un EAfin de Rn es igual a other"""
    self.verificacion(other)
    return self.contenidoEn(other) and other.contenidoEn(self)

def __ne__(self, other):
    """Indica si un subespacio de Rn es distinto de other"""
    self.verificacion(other)
    return not (self == other)
```

This code is used in chunk 70b.
Uses `EAfin` 70a.

71c ⟨*Métodos de la clase* EAfin 70c⟩+≡
```
def verificacion(self,other):
    if not isinstance(other, (SubEspacio, EAfin)) or  not self.Rn == other.Rn:
        raise \
         ValueError('Ambos argumentos deben ser subconjuntos de en un mismo espacio')
```

This code is used in chunk 70b.
Uses `EAfin` 70a and `SubEspacio` 66a.

72a
```
⟨Métodos de la clase EAfin 70c⟩+≡
  def __and__(self, other):
      """Devuelve la intersección de este EAfin con other"""
      self.verificacion(other)
      if isinstance(other, EAfin):
          M = Matrix(BlockMatrix([ [ self.S.cart], [other.S.cart] ]))
          w = Vector((self.S.cart*self.v).sis+(other.S.cart*other.v).sis)
      elif isinstance(other, SubEspacio):
          M = Matrix(BlockMatrix([ [ self.S.cart], [other.cart] ]))
          w = Vector((self.S.cart*self.v).sis+(other.S.cart*V0(S.Rn)).sis)
      try:
          S=SEL(M,w)
      except:
          print('Intersección vacía')
          return Sistema([])
      else:
          return S.eafin
```

This code is used in chunk 70b.
Uses `BlockMatrix` 92b, `EAfin` 70a, `Matrix` 13, `SEL` 62, `Sistema` 7, `SubEspacio` 66a, `V0` 83c, and `Vector` 9b.

Con `~A` obtendremos el complemento ortogonal del `SubEspacio` `A`

72b
```
⟨Métodos de la clase EAfin 70c⟩+≡
  def __invert__(self):
      """Devuelve el mayor SubEspacio perpendicular a self"""
      return SubEspacio(Sistema((~self.S.cart).sis))
```

This code is used in chunk 70b.
Uses `Sistema` 7 and `SubEspacio` 66a.

73 ⟨*Métodos de representación de la clase* `EAfin` 73⟩≡
```
  def _repr_html_(self):
      """Construye la representación para el entorno jupyter notebook"""
      return html(self.latex())

  def EcParametricas(self):
      """Representación paramétrica del SubEspacio"""
      return '\\left\\{ \\boldsymbol{v}\\in\\mathbb{R}^' \
        + latex(self.S.Rn) \
        + '\ \\left|\ \\exists\\boldsymbol{p}\\in\\mathbb{R}^' \
        + latex(max(self.S.dim,1)) \
        + '\ \\text{tal que}\ \\boldsymbol{v}= '\
        + latex(self.v) + '+' \
        + latex(Matrix(self.S.sgen.lista)) \
        + '\\boldsymbol{p}\\right. \\right\\}' \
        #+ '\qquad\\text{(ecuaciones paramétricas)}'

  def EcCartesianas(self):
      """Representación cartesiana del SubEspacio"""
      return '\\left\\{ \\boldsymbol{v}\\in\\mathbb{R}^' \
        + latex(self.S.Rn) \
        + '\ \\left|\ ' \
        + latex(self.S.cart) \
        + '\\boldsymbol{v}=' \
        + latex(self.S.cart*self.v) \
        + '\\right.\\right\\}' \
        #+ '\qquad\\text{(ecuaciones cartesianas)}'

  def latex(self):
      """ Construye el comando LaTeX para un SubEspacio de Rn"""
      if self.v != 0*self.v:
           return self.EcParametricas() + '\; = \;' + self.EcCartesianas()
      else:
           return latex(self.S)
```

This code is used in chunk 70b.
Uses `Matrix` 13 and `SubEspacio` 66a.

se puede instanciar tanto con un `Sistema` de `Vectores` de $\mathbb{R}^m$ como con una `Matrix` de orden m por n. Dado un `Sistema` de vectores, por ejemplo

$$S = \left[\begin{pmatrix}0\\1\\0\end{pmatrix};\ \begin{pmatrix}2\\1\\3\end{pmatrix};\ \begin{pmatrix}2\\10\\3\end{pmatrix} \right]$$

`SubEspacio( S )` representa el conjunto de combinaciones lineales de los `Vectores` de dicho `Sistema`, representado por las siguientes ecuaciones *paramétricas*:

$$\left\{ \boldsymbol{v}\in\mathbb{R}^3 \ \middle|\ \exists\boldsymbol{p}\in\mathbb{R}^2 \text{ tal que } \boldsymbol{v}= \begin{bmatrix}2&0\\1&1\\3&0\end{bmatrix}\boldsymbol{p} \right\}$$

donde el vector $\boldsymbol{p}$ es el vector de parámetros.

Y dada una `Matrix`, por ejemplo $\mathbf{M} = \begin{bmatrix}-3&0&2\\6&0&-4\end{bmatrix}$, `SubEspacio( M )` representa el conjunto de `Vectores` que son solución al sistema de ecuaciones cartesianas:

$$\left\{\boldsymbol{v}\in\mathbb{R}^3 \;\middle|\; \begin{bmatrix}-3 & 0 & 2\end{bmatrix}\boldsymbol{v}=\boldsymbol{0}\right\}$$

Ambos ejemplos representan un mismo subespacio de $\mathbb{R}^3$; y la representación de `SubEspacio` muestra la representación tanto con ecuaciones paramétricas, como con cartesianas. `Subespacio` tiene varios atributos.

- `dim`: dimensión del subespacio. En el ejemplo `dim=2`.
- `Rn`: indica al espacio $\mathbb{R}^n$ al que pertenece. En el ejemplo anterior `Rn=3` puesto que es un subespacio de $\mathbb{R}^3$.
- `base`: una base del subespacio (un `Sistema` de `Vector`es de `Rn`).
- `sgen`: `Matrix` de coeficientes empleada en la representación mediante un sistema de ecuaciones paramétricas.
- `cart`: `Matrix` de coeficientes empleada en la representación mediante un sistema de ecuaciones cartesianas.

Capítulo 4

Otros trozos de código

4.1 Métodos de representación para el entorno Jupyter

El método `html`, escribe el inicio y el final de un párrafo en html y en medio del párrafo escribirá la cadena `TeX`; que contendrá el código LaTeX de las expresiones matemáticas que queremos que se muestren en pantalla cuando usamos Jupyter Notebook. En el navegador, la librería MathJax de Javascript se encargará de convertir la expresión LaTeX en la grafía correspondiente.

75a ⟨*Método html general* 75a⟩≡

```
def html(TeX):
    """ Plantilla HTML para insertar comandos LaTeX """
    return "<p style=\"text-align:center;\">$" + TeX + "$</p>"
```

This code is used in chunk 41.

El método `latex`, convertirá en cadena de caracteres los inputs de tipo `str`, `float` o `int` [1], y en el resto de casos llamara al método `latex` de la clase desde la que se invocó a este método (es un truqui recursivo para que trate de manera parecida la expresiones en LaTeX y los tipos de datos que corresponden a números `int` o `float`).

75b ⟨*Método latex general* 75b⟩≡

```
def latex(a):
     if isinstance(a,float) | isinstance(a,int) | isinstance(a,str):
         return str(a)
     else:
         return a.latex()
```

This code is used in chunk 41.

Si el objeto a representar no es un número de coma flotante (`float`) ni tampoco un entero (`int`), el método general `latex` llamará el método `latex` de la clase correspondiente. Por tanto, si `a` es un `Vector`, una `Matrix`, o una transformación elemental (`T`), se llama al método `a.latex` definido en la clase correspondiente a dicho objeto `a`.[2] Sin

[1] resulta que para los tipos de datos `int` y `float` no es posible definir un nuevo método de representación. Afortunadamente la cadena de caracteres que representa el número nos vale perfectamente en ambos casos (tanto para los números enteros, `int`, como los números con decimales, `float`).

[2] más adelante tendré que incluir métodos de representación para otros tipos de datos, por ejemplo para poder representar vectores o matrices con polinomios.

embargo, la clase `Fraction` no tiene definidos los métodos de representación `html` o `latex`. Así pues, para representar fracciones necesitamos incorporar estos métodos en la preexistente clase `Fraction` que hemos importado desde la librería `fractions`. Primero definimos el método `_repr_html_fraction` (que sencillamente llamara al método `latex`) y luego definimos el método `latex_fraction` (en el nombre de ambos métodos he incluido la coletilla `fraction` para recordar que son los métodos que usaremos para la clase `fraction`). Si en LaTeX queremos representar la fración $\frac{a}{b}$ escribimos el código: `\frac{a}{b}`. Pero cuando el denominador es $b = 1$, no nos gusta escribir $\frac{a}{1}$, preferimos mostrar solamente el numerador a. Esto es precisamente lo que hace el método `latex_fraction` de más abajo.

Finalmente, con la función `setattr`, añadimos a la clase `Fraction` un método que se llamará '`_repr_html_`' (y que hace lo que hemos indicado al definir `_repr_html_fraction`), y un método que se llamará '`latex`' (y que hace los que hemos indicado al definir `latex_fraction`).

77 ⟨*Métodos html y latex para fracciones* 77⟩≡

```
def _repr_html_fraction(self):
    return html(self.latex())

def latex_fraction(self):
    if self.denominator == 1:
         return repr(self.numerator)
    else:
         return "\\frac{"+repr(self.numerator)+"}{"+repr(self.denominator)+"}"

setattr(Fraction, '_repr_html_', _repr_html_fraction)
setattr(Fraction, 'latex', latex_fraction)
```

This code is used in chunk 41.

4.2 Completando la clase `Sistema`

4.2.1 Representación de la clase `Sistema`

Necesitamos indicar a Python cómo representar los objetos de tipo `Sistema`.

Los sistemas, son secuencias finitas de objetos que representaremos con corchetes, separando los elementos por ";"

$$\boldsymbol{v} = [v_1;\ \ldots;\ v_n]$$

Definimos tres representaciones distintas. Una para la línea de comandos de Python de manera que "abra" el cochete "`[`" y a continuación muestre `self.lista` (la lista de objetos) separados por puntos y comas y se "cierre" el corchete "`]`". Por ejemplo, si la lista es `[a,b,c]`, Python nos mostrará en la linea de comandos: `[a; b; c]`.

La representación en LaTeX sigue el mismo esquema, pero los elementos son mostrados en su representación LaTeX (si la tienen) y es usada a su vez por la representación html usada por el entorno Jupyter.

78a ⟨*Métodos de representación de la clase* Sistema 78a⟩≡

```
def __repr__(self):
    """ Muestra un Sistema en su representación python """
    return '[' + \
        '; '.join(  repr (self.lista[i]) for i in range(0,len(self.lista)) ) + \
        ']'

def _repr_html_(self):
    """ Construye la representación para el entorno jupyter notebook """
    return html(self.latex())

def latex(self):
    """ Construye el comando LaTeX para representar un Sistema """
    return '\\left[' + \
        ';\;'.join( latex(self.lista[i]) for i in range(0,len(self.lista)) ) + \
        '\\right]'
```

This code is used in chunk 7.
Uses Sistema 7.

4.2.2 Otros métodos de la clase Sistema

Tal como se indica en las notas de la asignatura, definimos el producto de un Sistema por un Vector o Matrix a su derecha. Esto nos permite generalizar las combinaciones lineales a los elementos de un Sistema, si dichos elementos pertenecen a un espacio vectorial (Sistema*Vector), o bien, generar un nuevo Sistema cuyos elementos son combinaciones lineales de un Sistema dado de vectores de un espacio vectorial (Sistema*Matrix).

78b ⟨*Texto de ayuda para el operador producto por la derecha en la clase* Sistema 78b⟩≡

```
"""Multiplica un Sistema por un  Vector o una Matrix a su derecha

Parámetros:
    x (Vector): Vector con tantos componentes como elementos tiene el
                Sistema
      (Matrix): con tantas filas como elementos tiene el Sistema

Resultado:
    Combinación de los elementos del Sistema: Si x es Vector, devuelve
       una combinación lineal de los componentes del Sistema, si dicha
       operación está definida para ellos (los componentes del Vector
       son los coeficientes de la combinación)
    Matrix: Si x es Matrix, devuelve un Sistema si esa definida la
       operación combinación lineal entre los objetos del Sistema

Ejemplos:
>>> # Producto por un Vector
>>> Sistema([Vector([1, 3]), Vector([2, 4])]) * Vector([1, 1])

Vector([3, 7])
>>> # Producto por una Matrix
```

```
  >>> Sistema([Vector([1, 3]), Vector([2, 4])]) * Matrix([Vector([1,1])]))

  [Vector([3, 7])]
  """
```

This code is used in chunk 79.
Uses `Matrix` 13, `Sistema` 7, and `Vector` 9b.

Al implementar `Sistema` por `Vector` usamos la función **`sum`**. La función **`sum`** de Python tiene dos argumentos: el primero es la lista de objetos a sumar, y el segundo es el primer objeto de la suma (por defecto es el número 0). Como sumar cero más un elemento del `Sistema` puede no tener sentido, haremos el siguiente truco. La lista de elento a sumar va desde el segundo sumando en adelante, y como segundo argumento usamos el primer elemento de la lista que queremos sumar, así sumamos la lista completa.

79 ⟨*Producto de un* `Sistema` *por un* `Vector` *o una* `Matrix` *a su derecha* 79⟩≡

```
  def __mul__(self,x):
      ⟨Texto de ayuda para el operador producto por la derecha en la clase Sistema 78b⟩
      if isinstance(x, Vector):
          if len(self) != x.n:
              raise ValueError('Vector y Sistema incompatibles')
          return sum([(x|j)*(self|j) for j in range(1,len(self)+1)][1:], (x|1)*(self|1))

      elif isinstance(x, Matrix):
          if len(self) != x.m:
              raise ValueError('Matrix y Sistema incompatibles')
          return Sistema( [ self*(x|j) for j in range(1,x.n+1)] )
```

This code is used in chunk 7.
Uses `Matrix` 13, `Sistema` 7, and `Vector` 9b.

4.3 Completando la clase `Vector`

4.3.1 Representación de la clase `Vector`

Necesitamos indicar a Python cómo representar los objetos de tipo `Vector`.

Los vectores, son secuencias finitas de números que representaremos con paréntesis, bien en forma de fila

$$\boldsymbol{v} = (v_1, \ldots, v_n)$$

o bien en forma de columna

$$\boldsymbol{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}$$

Definimos tres representaciones distintas. Una para la línea de comandos de Python de manera que escriba `Vector` y a continuación encierre la representación de `self.sis` (el sistema de números), entre paréntesis. Por ejemplo, si la lista es `[a,b,c]`, Python nos mostrará en la linea de comandos: `Vector([a,b,c])`.

La representación en LaTeX encierra un vector (en forma de fila o de columna) entre paréntesis; y es usada a su vez por la representación html usada por el entorno Jupyter.

80a ⟨*Representación de la clase* `Vector` 80a⟩≡

```
def __repr__(self):
    """ Muestra el vector en su representación python """
    return 'Vector(' + repr(self.sis.lista) + ')'

def _repr_html_(self):
    """ Construye la representación para el entorno jupyter notebook """
    return html(self.latex())

def latex(self):
    """ Construye el comando LaTeX para representar un Vector"""
    if self.rpr == 'fila':
        return '\\begin{pmatrix}' + \
               ',&'.join([latex(self|i)   for i in range(1,self.n+1)]) + \
               '\\end{pmatrix}'
    else:
        return '\\begin{pmatrix}' + \
               '\\\\'.join([latex(self|i) for i in range(1,self.n+1)]) + \
               '\\end{pmatrix}'
```

This code is used in chunk 9b.
Uses `Vector` 9b.

4.3.2 Otros métodos para la clase `Vector`

Para jugar con el método de Gauss nos vendrá bien poder hacer el "reverso" de un `Vector`, es decir, obtener el `Vector` cuyas componentes están ordenadas en sentido inverso al original: la primera componente es la última, la segunda es la penúltima, etc.

80b ⟨*Reverso de un* `Vector` 80b⟩≡

```
def __reversed__(self):
```

```
        """Devuelve el reverso de un Vector"""
        return Vector(self.sis.lista[::-1])
```

This code is used in chunk 9b.
Uses `Vector` 9b.

También nos viene viene bien manejar el opuesto de un vector

81a ⟨*Opuesto de un* `Vector` 81a⟩≡

```
    def __neg__(self):
        """Devuelve el opuesto de un Vector"""
        return -1*self
```

This code is used in chunk 9b.
Uses `Vector` 9b.

81b ⟨*Comprobación de que un* `Vector` *es nulo* 81b⟩≡

```
    def esNulo(self):
        """Indica si es cierto que el vector es nulo"""
        return self==self*0
```

This code is used in chunk 9b.

4.4 Completando la clase `Matrix`

4.4.1 Otras formas de instanciar una `Matrix`

Si se introduce una lista (tupla) de listas o tuplas, creamos una matriz fila a fila. Si se introduce una `Matrix` creamos una copia de la matriz. Si se introduce una `BlockMatrix` se elimina el particionado y que crea una única matriz. Si el argumento no es correcto se informa con un error.

81c ⟨*Creación del atributo* `sis` *cuando no tenemos una* `lista` *de* `Vector`*es* 81c⟩≡

```

    elif isinstance(data, BlockMatrix):
        self.sis  =  Sistema([ Vector([ lista[i][j]|k|s  \
                                for i in range(data.m) for s in range(1,(data.lm[i])+1) ])\
                                for j in range(data.n) for k in range(1,(data.ln[j])+1) ])

    elif isinstance(lista[0], (list, tuple, Sistema)):
        ⟨Verificación de que todas las filas de la matriz tendrán la misma longitud 82a⟩
        self.sis  =  Sistema([ Vector([ lista[i][j] for i in range(len(lista   )) ]) \
                                                     for j in range(len(lista[0])) ])
```

This code is used in chunk 12.
Uses `BlockMatrix` 92b, `Sistema` 7, and `Vector` 9b.

4.4.2 Códigos que verifican que los argumentos son correctos

82a ⟨*Verificación de que todas las filas de la matriz tendrán la misma longitud* 82a⟩≡
```
if not all (type(lista[0])==type(v) and len(lista[0])==len(v) for v in iter(lista)):
    raise ValueError('no todas son listas o no tienen la misma longitud!')
```

This code is used in chunk 81c.

82b ⟨*Verificación de que todas las columnas de la matriz tienen la misma longitud* 82b⟩≡
```
if not all ( isinstance(v, Vector) and (lista[0].n == v.n) for v in iter(lista)):
    raise ValueError('no todos son vectores, o no tienen la misma longitud!')
```

This code is used in chunk 12.
Uses `Vector` 9b.

4.4.3 Representación de la clase `Matrix`

Y como en el caso de los vectores, construimos los dos métodos de presentación. Uno para la consola de comandos que escribe `Matrix` y entre paréntesis la lista de listas (es decir la lista de filas); y otro para el entorno Jupyter (que a su vez usa la representación LaTeX que representa las matrices entre corchetes como en las notas de la asignatura). Si self.lista es una lista vacía, se representa una matriz vacía.

82c ⟨*Representación de la clase* `Matrix` 82c⟩≡
```
def __repr__(self):
    """ Muestra una matriz en su representación python """
    return 'Matrix(' + repr(self.sis) + ')'

def _repr_html_(self):
    """ Construye la representación para el  entorno jupyter notebook """
    return html(self.latex())

def latex(self):
    """ Construye el comando LaTeX para representar una Matrix """
    return '\\begin{bmatrix}' + \
            '\\\\'.join(['&'.join([latex(i|self|j) for j in range(1,self.n+1) ]) \
                                                   for i in range(1,self.m+1) ]) + \
           '\\end{bmatrix}'
```

This code is used in chunk 13.
Uses `Matrix` 13.

4.4.4 Otros métodos para la clase Matrix

Para jugar con el método de Gauss nos vendrá bien poder hacer el "reverso" de una Matrix, es decir, obtener la Matrix cuyas columnas están ordenadas en sentido inverso al original.

83a ⟨*Reverso de una* Matrix 83a⟩≡
```
def __reversed__(self):
    """Devuelve el reverso de una Matrix"""
    return Matrix(reversed(self.sis))
```
This code is used in chunk 13.
Uses Matrix 13.

También nos viene viene bien manejar el opuesto de una Matrix

83b ⟨*Opuesto de una* Matrix 83b⟩≡
```
def __neg__(self):
    """Devuelve el opuesto de una Matrix"""
    return -1*self
```
This code is used in chunk 13.
Uses Matrix 13.

4.5 Vectores y Matrices especiales

Notación en Mates 2

Los vectores cero **0** y las matrices cero **0** se pueden implementar como subclases de la clase Vector y Matrix (pero tenga en cuenta que Python necesita conocer el número de componentes del vector y el orden de la matriz):

V0 es una subclase de Vector (por tanto hereda los atributos de la clase Vector), pero el código inicia (y devuelve) un objeto de su superclase, es decir, inicia y devuelve un Vector.

83c ⟨*Definición de vector nulo:* V0 83c⟩≡
```
class V0(Vector):
    def __init__(self, n ,rpr = 'columna'):
        """ Inicializa el vector nulo de n componentes"""
        super(self.__class__ ,self).__init__([0 for i in range(n)], rpr)
```
This code is used in chunk 41.
Defines:
V0, used in chunks 29a, 61a, 62, 65, 66, 69b, 72a, and 84a.
Uses Vector 9b.

Y lo mismo hacemos para matrices

84a ⟨*Definición de matriz nula:* `M0` 84a⟩≡

```
class M0(Matrix):
    def __init__(self, m, n=None):
        """ Inicializa una matriz nula de orden n """
        n = m if n is None else n

        super(self.__class__ ,self).__init__([ V0(m) for j in range(n)])
```

This code is used in chunk 41.
Uses `Matrix` 13 and `V0` 83c.

También debemos definir la matriz identidad de orden n (y sus filas y columnas). En los apuntes de clase no solemos indicar expresamente el orden de la matriz identidad (pues normalmente se sobrentiende por el contexto). Pero esta habitual imprecisión no nos la podemos permitir con el ordenador.

Notación en Mates 2

- I (de orden n) es la matriz tal que ${}_{i|}\mathsf{I}_{|j} = \begin{cases} 1 & \text{si } j = i \\ 0 & \text{si } j \neq i \end{cases}$.

84b ⟨*Definición de la matriz identidad:* `I` 84b⟩≡

```
class I(Matrix):
    def __init__(self, n):
        """ Inicializa la matriz identidad de tamaño n """
        super(self.__class__ ,self).__init__(\
                      [[(i==j)*1 for i in range(n)] for j in range(n)])
```

This code is used in chunk 41.
Uses `Matrix` 13.

4.6 Completando la clase T

4.6.1 Otras formas de instanciar una T

Si se instancia T usando otra Transfomación elemental, sencillamente se copia el atributo `t`. Si se instancia T usando una lista (no vacía) de Transfomaciones elementales, el atributo `t` será la lista de abreviaturas resultante de concatenar las abreviaturas de todas las Transfomaciones elementales de la lista empleada en la instanciación.

85 ⟨*Creación del atributo* `t` *cuando se instancia con otra* `T` *o lista de* `T`*s* 85⟩≡

```
if isinstance(t, T):
    self.t = t.t

elif isinstance(t, list) and t and isinstance(t[0], T):
        self.t = [val for sublist in [x.t for x in t] for val in CreaLista(sublist)]
```

This code is used in chunk 33.
Uses `CreaLista` 34b and `T` 37b.

4.6.2 Representación de la clase T

De nuevo construimos los dos métodos de presentación. Uno para la consola de comandos que escribe `T` y entre paréntesis la abreviatura (una tupla o un conjunto) que representa la transformación. Así,

- `T( {1, 5} )` : intercambio entre los vectores primero y quinto.
- `T( (6, 2) )` : multiplica por seis el segundo vector.
- `T( (-1, 2, 3) )` : resta el segundo vector al tercero.

La otra representación es para el entorno Jupyter y replica la notación usada en los apuntes de la asignatura:

Python	Representación en Jupyter
`T( {1, 5} )`	$\underset{[\mathbf{1}\rightleftharpoons\mathbf{5}]}{\boldsymbol{\tau}}$
`T( (6, 2) )`	$\underset{[(6)\mathbf{2}]}{\boldsymbol{\tau}}$
`T( (-1, 2, 3) )`	$\underset{[(-1)\mathbf{2}+\mathbf{3}]}{\boldsymbol{\tau}}$

Los apuntes de la asignatura usan una notación matricial, y por tanto es una notación que discrimina entre operaciones sobre las filas o las columnas, situando los operadores a la izquierda o a la derecha de la matriz. En este sentido, nuestra notación en Python hace lo mismo. Así, en la siguiente tabla, la columna de la izquierda corresponde a operaciones sobre las filas, y la columna de la derecha a las operaciones sobre las columnas:

Mates II	Python	Mates II	Python
$\underset{[\boldsymbol{i}\rightleftharpoons\boldsymbol{j}]}{\boldsymbol{\tau}}\mathbf{A}$	`T( {i,j} ) & A`	$\mathbf{A}\underset{[\boldsymbol{i}\rightleftharpoons\boldsymbol{j}]}{\boldsymbol{\tau}}$	`A & T( {i,j} )`
$\underset{[(a)\boldsymbol{i}]}{\boldsymbol{\tau}}\mathbf{A}$	`T( (a,i) ) & A`	$\mathbf{A}\underset{[(a)\boldsymbol{j}]}{\boldsymbol{\tau}}$	`A & T( (a,j) )`
$\underset{[(a)\boldsymbol{i}+\boldsymbol{j}]}{\boldsymbol{\tau}}\mathbf{A}$	`T( (a,i,j) ) & A`	$\mathbf{A}\underset{[(a)\boldsymbol{i}+\boldsymbol{j}]}{\boldsymbol{\tau}}$	`A & T( (a,i,j) )`

Secuencias de transformaciones

Considere las siguientes transformaciones

- multiplicar por 2 el primer vector, cuya abreviatura es: `(2, 1)`
- intercambiar el tercer vector por cuarto, cuya abreviatura es: `{3, 4}`

Para indicar una secuencia que contiene ambas transformaciones, usaremos una lista de abreviaturas: `[(2,1), {3,4}]`. De esta manera, cuando componemos ambas operaciones: `T( (2, 1) ) & T( {3, 4} )`, nuestra librería nos devuelve la trasformación composición de las dos operaciones **en el orden en el que han sido escritas**:

al escribir `T( (2, 1) ) & T( {3, 4} )` Python nos devuelve `T( [ (1, 2), {3, 4} ] )`

Por tanto, si queremos realizar dichas operaciones sobre las columnas de la matriz **A**, podemos hacerlo de dos formas:

- `A & T( (2, 1) ) & T( {3, 4} )` (indicando las transformaciones de una en una)
- `A & T( [(2, 1), {3, 4}] )` (usando la transformación composición de todas ellas)

y si queremos operar sobre la filas hacemos exactamente igual, pero a la izquierda de la matriz

- `T( (2, 1) ) & T( {3, 4} ) & A`
- `T( [(2, 1), {3, 4}] ) & A`

Representación de una secuencia de transformaciones.

Representación en la consola de Python	Representación en Jupyter
`T( [ (2, 1), (1, 3, 2) ] )`	$\underset{\begin{subarray}{c} \left[\left(2\right)\mathbf{1}\right] \\ \left[\left(1\right)\mathbf{3}+\mathbf{2}\right] \end{subarray}}{\boldsymbol{\tau}}$

86 ⟨*Representación de la clase* T 86⟩≡
```
def __repr__(self):
    """ Muestra T en su representación python """
    return 'T(' + repr(self.t) + ')'

def _repr_html_(self):
    """ Construye la representación para el entorno jupyter notebook """
    return html(self.latex())

def latex(self):
    """ Construye el comando LaTeX para representar una Trans. Elem. """
    def simbolo(t):
        """Escribe el símbolo que denota una trasformación elemental particular"""
        if isinstance(t,set):
            return '\\left[\\mathbf{' + latex(min(t)) + \
              '}\\rightleftharpoons\\mathbf{' + latex(max(t)) + '}\\right]'
        if isinstance(t,tuple) and len(t) == 2:
            return '\\left[\\left(' + \
              latex(t[0]) + '\\right)\\mathbf{'+ latex(t[1]) + '}\\right]'
        if isinstance(t,tuple) and len(t) == 3:
            return '\\left[\\left(' + latex(t[0]) + '\\right)\\mathbf{' + \
              latex(t[1]) + '}' + '+\\mathbf{' + latex(t[2]) + '} \\right]'

    if isinstance(self.t, (set, tuple) ):
        return '\\underset{' + simbolo(self.t) + '}{\\mathbf{\\tau}}'

    elif isinstance(self.t, list):
        return '\\underset{\\begin{subarray}{c} ' + \
              '\\\\'.join([simbolo(i) for i in self.t])  + \
              '\\end{subarray}}{\\mathbf{\\tau}}'
```

This code is used in chunk 37b.
Uses T 37b.

4.7 Representación de los procesos de eliminación gaussiana

Cuando hemos encadenado varios procedimientos de eliminación, deberíamos poder ver los pasos desde el princio hasta el final. Para ello comprobamos si `data` fue obtenido mediante un proceso previo de eliminación. El modo de saberlo es comprobar si `data` posee el atributo `pasos`. El atributo `tex` guarda el código LaTeX que muestra el proceso completo, y se construye aplicando el método `PasosYEscritura`. Si `rep` es distinto de cero se muestran los pasos en el entorno Jupyter. El atributo `rank` guarda el rango y `pasos` las listas de transformaciones elementales empleadas.

87a ⟨*Se guardan los atributos* `tex`, `pasos` *y* `rank` *(y se muestran los pasos si se pide)* 87a⟩≡

```
⟨Definición del método PasosYEscritura 88⟩
⟨Método auxiliar que crea el atributo tex y muestra los pasos en Jupyter 87b⟩
pasosPrevios = data.pasos if hasattr(data, 'pasos') and data.pasos else [[],[]]
TexPasosPrev = data.tex   if hasattr(data, 'tex')   and data.tex   else []
self.tex = tex(data, pasos, TexPasosPrev)
self.rank  = r
pasos[0] = pasos[0] + pasosPrevios[0]
pasos[1] = pasosPrevios[1] + pasos[1]
self.pasos = pasos
```

This code is used in chunks 50–52 and 54–57.

87b ⟨*Método auxiliar que crea el atributo* `tex` *y muestra los pasos en Jupyter* 87b⟩≡

```
def tex(data, pasos, TexPasosPrev=[]):
    ⟨Definición del método PasosYEscritura 88⟩
    tex     = PasosYEscritura(data, pasos, TexPasosPrev)
    if rep:
        from IPython.display import display, Math
        display(Math(tex))
    return tex
```

This code is used in chunks 59–62 and 87a.

Cuando mostramos los pasos, es más legible mostrar únicamente los que modifican la matriz (omitiendo sustituciones de una columna por ella misma, productos de una columna por 1, o sumas de un vector nulo a una columna). Esto es lo que se hace con la ⟨*Definición del método* `PasosYEscritura` 88⟩

El atributo `tex` guardará el código LaTeX que muestra el proceso completo. Si ha habido transformaciones previas, la cadena de LaTeX que permite su representación en el entorno Jupyter estará guardada en la variable (`TexPasosPrev`), y a dicha cadena hay que añadir la correspondiente cadena de LaTeX que permita representar los nuevos `pasos` dados como argumento de este método. Si `TexPasosPrev` es vacio, la escritura comienza con la representación de `data`. A la hora de representar los pasos hay que tener en cuenta si se dan sobre las filas (l==0) o sobre las columnas (l==1). Todo esto es lo que hace el método `PasosYEscritura`:

88 ⟨*Definición del método* PasosYEscritura 88⟩≡

```
def PasosYEscritura(data,pasos,TexPasosPrev=[]):
    """Escribe en LaTeX los pasos efectivos dados"""
    A   = Matrix(data);  p   = [[],[]]
    tex = latex(data) if len(TexPasosPrev)==0 else TexPasosPrev
    for l in range(0,2):
        p[l] = [ T([j for j in pasos[l][i].t if (isinstance(j,set) and len(j)>1)   \
                            or (isinstance(j,tuple) and len(j)==3 and j[0]!=0)     \
                            or (isinstance(j,tuple) and len(j)==2 and j[0]!=1) ])  \
                                                for i in range(0,len(pasos[l])) ]
        p[l]   = [ t for t in p[l] if len(t.t)!=0]  # quitamos abreviaturas vacías
        if l==0:
            for i in reversed(range(0,len(p[l]))):
                tex += '\\xrightarrow[' + latex(p[l][i]) + ']{}'
                if isinstance (data, Matrix):
                             tex += latex( p[l][i] & A )
                elif isinstance (data, BlockMatrix):
                             tex += latex( key(data.lm)|(p[l][i] & A)|key(data.ln) )
        if l==1:
            for i in range(0,len(p[l])):
                tex += '\\xrightarrow{' + latex(p[l][i]) + '}'
                if isinstance (data, Matrix):
                             tex += latex( A & p[l][i] )
                elif isinstance (data, BlockMatrix):
                             tex += latex( key(data.lm)|(A & p[l][i])|key(data.ln) )
    return tex
```

This code is used in chunks 60b and 87.
Uses `BlockMatrix` 92b, `Matrix` 13, and `T` 37b.

4.8 Representación de la resolución de sistemas de ecuaciones

89 ⟨*Métodos de representación de la clase* Homogenea 89⟩≡

```
def __repr__(self):
    """Muestra el Espacio Nulo de una matriz en su representación python"""
    return 'Combinaciones lineales de (' + repr(self.sgen) + ')'

def _repr_html_(self):
    """Construye la representación para el entorno jupyter notebook"""
    return html(self.latex())

def latex(self):
    """ Construye el comando LaTeX para la solución de un Sistema Homogéneo"""
    if self.determinado:
        return '\\text{La única solución es el vector cero: }' + \
                    latex(self.sgen.lista[0])
    else:
        return '\\text{Conjunto de combinaciones lineales de }' + \
         ',\;'.join([latex(self.sgen.lista[i]) for i in range(0,len(self.sgen))])
```

This code is used in chunk 61a.
Uses Sistema 7.

90 ⟨*Métodos de representación de la clase* SEL 90⟩≡

```
def EcParametricas(self):
    """Representación paramétrica del SubEspacio"""
    return '\\left\\{ \\boldsymbol{x}\\in\\mathbb{R}^' \
      + latex(self.eafin.Rn) \
      + '\ \\left|\ \\exists\\boldsymbol{p}\\in\\mathbb{R}^' \
      + latex(len(self.sgen)) \
      + '\ \\text{tal que}\ \\boldsymbol{x}= '\
      + latex(self.solP) + '+' \
      + latex(Matrix(self.sgen)) \
      + '\\boldsymbol{p}\\right. \\right\\}' \

def __repr__(self):
    """Muestra el Espacio Nulo de una matriz en su representación python"""
    return repr(self.solP) + ' + Combinaciones lineales de (' + repr(self.sgen) + ')'

def _repr_html_(self):
    """Construye la representación para el entorno jupyter notebook"""
    return html(self.latex())

def latex(self):
    """ Construye el comando LaTeX para la solución de un Sistema Homogéneo"""
    if self.determinado:
        return '\\text{Tiene solución única:  }\\boldsymbol{x}=' + latex(self.solP)
    else:
        return '\\text{Conjunto de vectores: }' + self.EcParametricas()
```

This code is used in chunk 62.
Uses `Matrix` 13, `Sistema` 7, and `SubEspacio` 66a.

4.9 Completando la clase `SubEspacio`

4.9.1 Representación de la clase `SubEspacio`

91

```
⟨Métodos de representación de la clase SubEspacio 91⟩≡
  def _repr_html_(self):
      """Construye la representación para el entorno jupyter notebook"""
      return html(self.latex())

  def EcParametricas(self):
      """Representación paramétrica del SubEspacio"""
      return '\\left\\{ \\boldsymbol{v}\\in\\mathbb{R}^' \
        + latex(self.Rn) \
        + '\ \\left|\ \\exists\\boldsymbol{p}\\in\\mathbb{R}^' \
        + latex(max(self.dim,1)) \
        + '\ \\text{tal que}\ \\boldsymbol{v}= '\
        + latex(Matrix(self.sgen.lista)) \
        + '\\boldsymbol{p}\\right. \\right\\}' \
        #+ '\qquad\\text{(ecuaciones paramétricas)}'

  def EcCartesianas(self):
      """Representación cartesiana del SubEspacio"""
      return '\\left\\{ \\boldsymbol{v}\\in\\mathbb{R}^' \
        + latex(self.Rn) \
        + '\ \\left|\ ' \
        + latex(self.cart) \
        + '\\boldsymbol{v}=\\boldsymbol{0}\\right.\\right\\}' \
        #+ '\qquad\\text{(ecuaciones cartesianas)}'

  def latex(self):
      """ Construye el comando LaTeX para un SubEspacio de Rn"""
      return self.EcParametricas() + '\; = \;' + self.EcCartesianas()
```

This code is used in chunk 69c.
Uses `Matrix` 13 and `SubEspacio` 66a.

4.10 La clase `BlockMatrix`. Matrices particionadas

Las matrices particionadas no son tan importantes para seguir el curso, aunque si se usan en esta librería. Piense que cuando invierte una matriz o resuelve un sistema de ecuaciones, usa una matriz particionada (con dos bloques: una matriz arriba, y la matriz identidad con idéntico número de columnas debajo). Como esta librería replica lo que se ve en clase, es necesario definir las matrices particionadas. Si quiere, ***puede saltarse esta sección****: el modo de particionar una matriz es sencillo y se puede aprender rápidamente con el siguiente Notebook*

Tutorial previo en un Jupyter notebook

Este Notebook es un ejemplo sobre el **uso de nuestra librería para Mates 2** en la carpeta "TutorialPython" en
`https://mybinder.org/v2/gh/mbujosab/LibreriaDePythonParaMates2/master`

Las matrices por bloques o cajas $\boxed{\mathbf{A}}$ son tablas de matrices de modo que todas las matrices de una misma fila comparten el mismo número de filas, y todas las matrices de una misma columna comparten el mismo número de columnas. Por ello al "pegar" todas ellas obtenemos una gran matriz.

El argumento de inicialización `sis` es una lista (o tupla) de listas de matrices, cada una de las listas de matrices es una fila de bloques (o submatrices con el mismo número de filas).

El atributo `self.m` contiene el número de filas (de bloques o submatrices) y `self.n` contiene el número de columnas (de bloques o submatrices). Añadimos el atributo `self.ln`, que es una lista con el número de filas que tienen las submatrices de cada fila, y `self.lm` con el número de columnas de las submatrices de cada columna.

92a ⟨*Inicialización de la clase* `BlockMatrix` 92a⟩≡

```
def __init__(self, data):
    """Inicializa una BlockMatrix con una lista de listas de matrices"""
    self.sis    = Sistema([Sistema(data[i]) for i in range(0,len(data))])
    self.m      = len(data)
    self.n      = len(data[0])
    self.lm     = [fila[0].m for fila in data]
    self.ln     = [c.n for c in data[0]]
```

This code is used in chunk 92b.
Uses `BlockMatrix` 92b and `Sistema` 7.

La clase `BlockMatrix` junto con el listado de sus métodos aparece en el siguiente recuadro:

92b ⟨*Definición de la clase* `BlockMatrix` 92b⟩≡

```
class BlockMatrix:
    ⟨Inicialización de la clase BlockMatrix 92a⟩
    ⟨Repartición de las columnas de una BlockMatrix 94a⟩
    ⟨Repartición de las filas de una BlockMatrix 94b⟩
    ⟨Representación de la clase BlockMatrix 96⟩
```

This code is used in chunk 41.
Defines:
`BlockMatrix`, used in chunks 4, 5, 11, 17c, 20, 59–63, 67a, 68c, 70a, 72a, 81c, 88, 92–94, and 96.

4.10.1 Particionado de matrices

Vamos a completar las capacidades de los operadores "`i|`" y "`|j`" sobre matrices. Hasta ahora, si los argumentos `i` o `j` eran *enteros* (`int`), se seleccionaba una fila o una columna respectivamente; y si los argumentos `i` o `j` eran *listas o tuplas* de índices, se generaba una submatriz con las filas o las columnas indicadas.

Aquí, si los argumentos `i` o `j` son conjuntos de enteros, asumimos que dicho números enteros indican las filas o columnas por las que se debe particionar una `Matrix` según el siguiente cuadro explicativo:

Notación en Mates 2

- Si $p \leq q \in \mathbb{N}$ denotaremos con $(p:q)$ a la secuencia $p, p+1, \ldots, q$, (es decir, a la lista ordenada de los números de $\{k \in \mathbb{N} | p \leq k \leq q\}$).
- Si $i_1, \ldots, i_r \in \mathbb{N}$ con $i_1 < \ldots < i_r \leq m$ donde m es el número de filas de $\mathbf{A}$, entonces ${}_{\{i_1,\ldots,i_r\}|}\mathbf{A}$ es la matriz de bloques
$$ {}_{\{i_1,\ldots,i_r\}|}\mathbf{A} = \left[\begin{array}{c} {}_{(1:i_1)|}\mathbf{A} \\ \hline {}_{(i_1+1:i_2)|}\mathbf{A} \\ \hline \vdots \\ \hline {}_{(i_r+1:m)|}\mathbf{A} \end{array}\right]$$
- Si $j_1, \ldots, j_s \in \mathbb{N}$ con $j_1 < \ldots < j_s \leq n$ donde n es el número de columnas de $\mathbf{A}$, entonces $\mathbf{A}_{|\{j_1,\ldots,j_s\}}$ es la matriz de bloques
$$\mathbf{A}_{|\{j_1,\ldots,j_s\}} = \left[\begin{array}{c|c|c|c} \mathbf{A}_{|(1:j_1)} & \mathbf{A}_{|(j_1+1:j_2)} & \cdots & \mathbf{A}_{|(j_s+1:n)} \end{array}\right]$$

Comencemos por la partición de índices a partir de un conjunto y un número (correspondiente al último índice).

93a ⟨*Definición del método* `particion` 93a⟩≡
```
def particion(s,n):
    """ genera la lista de particionamiento a partir de un conjunto y un número
    >>> particion({1,3,5},7)

    [[1], [2, 3], [4, 5], [6, 7]]
    """
    p = sorted(list(s | set([0,n])))
    return [ list(range(p[k]+1,p[k+1]+1)) for k in range(len(p)-1) ]
```
This code is used in chunk 41.

y ahora el método de partición por filas y por columnas resulta inmediato:

93b ⟨*Partición de una matriz por filas de bloques* 93b⟩≡
```
elif isinstance(i,set):
    return BlockMatrix ([ [a|self] for a in particion(i,self.m) ])
```
This code is used in chunk 21.
Uses `BlockMatrix` 92b.

93c ⟨*Partición de una matriz por columnas de bloques* 93c⟩≡
```
elif isinstance(j,set):
    return BlockMatrix ([ [self|a for a in particion(j,self.n)] ])
```

This code is used in chunk 18.
Uses `BlockMatrix` 92b.

Pero aún nos falta algo:

Notación en Mates 2

- Si $i_1, \ldots, i_r \in \mathbb{N}$ con $i_1 < \ldots < i_r \leq m$ donde m es el número de filas de $\mathbf{A}$ y $j_1, \ldots, j_s \in \mathbb{N}$ con $j_1 < \ldots < j_s \leq n$ donde n es el número de columnas de $\mathbf{A}$ entonces

$$
{}_{\{i_1,\ldots,i_y\}|}\mathbf{A}_{|\{j_1,\ldots,j_s\}} = \left[\begin{array}{c|c|c|c}
{}_{(1:i_1)|}\mathbf{A}_{|(1:j_1)} & {}_{(1:i_1)|}\mathbf{A}_{|(j_1+1:j_2)} & \cdots & {}_{(1:i_1)|}\mathbf{A}_{|(j_s+1:n)} \\ \hline
{}_{(i_1+1:i_2)|}\mathbf{A}_{|(1:j_1)} & {}_{(i_1+1:i_2)|}\mathbf{A}_{|(j_1+1:j_2)} & \cdots & {}_{(i_1+1:i_2)|}\mathbf{A}_{|(j_s+1:n)} \\ \hline
\vdots & \vdots & \cdots & \vdots \\ \hline
{}_{(i_k+1:m)|}\mathbf{A}_{|(1:j_1)} & {}_{(i_k+1:m)|}\mathbf{A}_{|(j_1+1:j_2)} & \cdots & {}_{(i_k+1:m)|}\mathbf{A}_{|(j_s+1:n)}
\end{array}\right]
$$

es decir, queremos poder particionar una `BlockMatrix`. Los casos interesantes son cuando particionamos por el lado contrario por el que se particionó la matriz de partida, es decir,

$$
{}_{\{i_1,\ldots,i_r\}|}\left(\mathbf{A}_{|\{j_1,\ldots,j_s\}}\right) \qquad \text{y} \qquad \left({}_{\{i_1,\ldots,i_r\}|}\mathbf{A}\right)_{|\{j_1,\ldots,j_s\}}
$$

que, por supuesto, debe dar el mismo resultado. Para dichos casos, programamos el siguiente código que particiona una `BlockMatrix`. Primero el procedimiento para particionar por columnas cuando hay una única columna de matrices (`self.n == 1`). El caso general se verá más tarde:

94a ⟨*Repartición de las columnas de una* `BlockMatrix` 94a⟩≡

```
  def __or__(self,j):
      """ Reparticiona por columna una matriz por cajas """
      if isinstance(j,set):
          if self.n == 1:
              return BlockMatrix([ [ self.sis.lista[i][0]|a  \
                                      for a in particion(j,self.sis.lista[0][0].n)] \
                                      for i in range(self.m) ])

          ⟨Caso general de repartición por columnas 95b⟩
```

This code is used in chunk 92b.
Uses `BlockMatrix` 92b.

y hacemos lo mismo para particionar por filas cuando `self.m == 1` (la matriz por bloques tiene una única fila):

94b ⟨*Repartición de las filas de una* `BlockMatrix` 94b⟩≡

```
      def __ror__(self,i):
          """ Reparticiona por filas una matriz por cajas """
          if isinstance(i,set):
              if self.m == 1:
                  return BlockMatrix([[ a|self.sis.lista[0][j]  \
                                         for j in range(self.n) ] \
```

```
                                    for a in particion(i,self.sis.lista[0][0].m)])

                 ⟨Caso general de repartición por filas 95c⟩
```

This code is used in chunk 92b.
Uses `BlockMatrix` 92b.

Falta implementar el caso general. Debemos decidir el significado de reparticionar una matriz por el mismo lado por el que ya ha sido particionada. Seguiremos un criterio práctico. . . eliminar el anterior particionado y aplicar el nuevo:

$$
\begin{aligned}
{}_{\{i'_1,\dots,i'_r\}|}\left({}_{\{i_1,\dots,i_k\}|}\mathbf{A}_{|\{j_1,\dots,j_s\}}\right) &= {}_{\{i'_1,\dots,i'_r\}|}\mathbf{A}_{|\{j_1,\dots,j_s\}}\\
\left({}_{\{i_1,\dots,i_k\}|}\mathbf{A}_{|\{j_1,\dots,j_s\}}\right)_{|\{j'_1,\dots,j'_r\}} &= {}_{\{i_1,\dots,i_k\}|}\mathbf{A}_{|\{j'_1,\dots,j'_r\}}
\end{aligned}
$$

Para ello nos viene bien extraer el conjunto selector a partir del resultado:

95a ⟨*Definición del procedimiento de generación del conjunto clave para particionar* 95a⟩≡

```
def key(L):
    """Genera el conjunto clave a partir de una secuencia de tamaños
    número
    >>> key([1,2,1])

    {1, 3, 4}
    """
    return set([ sum(L[0:i]) for i in range(1,len(L)+1) ])
```

This code is used in chunk 41.

Así, los casos generales consisten en reparticionar de nuevo:

95b ⟨*Caso general de repartición por columnas* 95b⟩≡

```
elif self.n > 1:
     return (key(self.lm) | Matrix(self)) | j
```

This code is used in chunk 94a.
Uses `Matrix` 13.

95c ⟨*Caso general de repartición por filas* 95c⟩≡

```
elif self.m > 1:
    return i | (Matrix(self) | key(self.ln))
```

This code is used in chunk 94b.
Uses `Matrix` 13.

Observación 4. El método `__or__` está definido para conjuntos ... realiza la unión. Por tanto si `A` es una matriz, la orden `{1,2}|({3}|A)` no da igual que `({1,2}|{3})|A`. La primera es igual da `{1,2}|A`, mientras que la segunda da `{1,2,3}|A`.

4.10.2 Representación de la clase `BlockMatrix`

A continuación definimos las reglas de representación para las matrices por bloques. `Matrix` y `BlockMatrix` son objetos distintos. Los bloques se separan con lineas verticales y horizontales; pero si hay un único bloque, no habrá ninguna línea vertical u horizontal por medio de la representación de la `BlockMatrix`. Así, si una matriz por bloques tienen un único bloque, pintaremos una caja alrededor para distinguirla de una matriz ordinaria.

96 ⟨*Representación de la clase* `BlockMatrix` 96⟩≡

```
def __repr__(self):
    """ Muestra una matriz en su representación Python """
    return 'BlockMatrix(' + repr(self.sis) + ')'

def _repr_html_(self):
    """ Construye la representación para el  entorno Jupyter Notebook """
    return html(self.latex())

def latex(self):
    """ Escribe el código de LaTeX para representar una BlockMatrix """
    if self.m == self.n == 1:
        return \
          '\\begin{array}{|c|}' + \
          '\\hline ' + \
          '\\\\ \\hline '.join( \
                ['\\\\'.join( \
                ['&'.join( \
                [latex(self.sis.lista[0][0]) ]) ]) ])  + \
          '\\\\ \\hline ' + \
          '\\end{array}'
    else:
        return \
          '\\left[' + \
          '\\begin{array}{' + '|'.join([n*'c' for n in self.ln])  + '}' + \
          '\\\\ \\hline '.join( \
                ['\\\\'.join( \
                ['&'.join( \
                [latex(self.sis.lista[i][j]|k|s) \
                for j in range(self.n) for k in range(1,self.ln[j]+1) ]) \
                for s in range(1,self.lm[i]+1) ]) for i in range(self.m) ])  + \
          '\\\\' + \
          '\\end{array}' + \
          '\\right]'
```

This code is used in chunk 92b.
Uses `BlockMatrix` 92b.

Appendix A

Sobre este documento

Con ánimo de que esta documentación sea más didáctica, en el Capítulo 1 muestro las partes más didácticas del código, y relego las otras al Capítulo 4. Así puedo destacar cómo la librería de Python es una implementación literal de las definiciones dadas en mis notas de la asignatura de Mates II. Para lograr presentar el código en un orden distinto del que realmente tiene en la librería uso la herramienta noweb. Una breve explicación aparece en la siguiente sección…

Literate programming con noweb

Este documento está escrito usando noweb. Es una herramienta que permite escribir a la vez tanto código como su documentación. El código se escribe a trozos o "chunks" como por ejemplo este:

97a
```
⟨Chunk de ejemplo que define la lista a 97a⟩≡
  a = ["Matemáticas II es mi asignatura preferida", "Python mola", 1492, "Noweb"]
This code is used in chunk 97c.
```

y este otro chunk:

97b
```
⟨Segundo chunk de ejemplo que cambia el último elemento de la lista a 97b⟩≡
  a[-1] = 10
This code is used in chunk 97c.
```

Cada chunk recibe un nombre (que yo uso para describir lo que hace el código dentro del chunk). Lo maravilloso de este modo de programar es que dentro de un chunk se pueden insertar otros chunks. Así, podemos programar el siguiente guión de Python (`EjemploLiterateProgramming.py`) que enumera los elementos de una tupla y después hace unas sumas:

97c
```
⟨EjemploLiterateProgramming.py 97c⟩≡
  ⟨Chunk de ejemplo que define la lista a 97a⟩
  ⟨Segundo chunk de ejemplo que cambia el último elemento de la lista a 97b⟩

  for indice, item in enumerate(a, 1):
      print (indice, item)
```

⟨*Chunk final que indica qué tipo de objeto es* `a` *y hace unas sumas* 101⟩
Root chunk (not used in this document).

Este modo de escribir el código permite destacar unas partes y pasar por alto otras. Por ejemplo, *del chunk del recuadro de arriba me interesa que se vea el código del bucle que permite enumerar los elementos de una lista*. Lo demás es accesorio y se puede consultar en los correspondientes chunks. Como el nombre de dichos chunks es auto-explicativo, mirando el recuadro anterior es fácil hacerse una idea de que hace el programa "`EjemploLiterateProgramming.py`" en su conjunto.

Fíjese que el número al final del nombre de cada chunk corresponde a la página donde se puede consultar su código. Por ejemplo, el último chunk de este ejemplo se encuentra en la Página 101 de este documento.

El código completo del ejemplo usado para explicar cómo funciona el "Literate Programming" queda así:

```
a = ["Matemáticas II es mi asignatura preferida", "Python mola", 1492, "Noweb"]
a[-1] = 10

for indice, item in enumerate(a, 1):
    print (indice, item)

type(a)
2+2
3+20
```

A.1 Secciones de código

Licencia

100 ⟨*Copyright y licencia GPL* 100⟩≡

```
# Copyright (C) 2019  Marcos Bujosa

# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
```

```
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/
```

Root chunk (not used in this document).

Último chunk del ejemplo de Literate Programming

Este es uno de los trozos de código del ejemplo.

101 ⟨*Chunk final que indica qué tipo de objeto es* `a` *y hace unas sumas* 101⟩≡

```
type(a)
2+2
3+20
```

This code is used in chunk 97c.