

Librería en Python para Matemáticas II (Álgebra Lineal)

<https://github.com/mbujosab/LibreriaDePythonParaMates2>

Marcos Bujosa

December 29, 2019

Índice

Declaración de intenciones	3
1 Código principal de la librería. Las clases Sistema, Vector, Matrix y T	4
1.1 La clase Sistema	4
1.2 La clase Vector	8
1.2.1 Implementación de los vectores en la clase Vector	9
1.3 La clase Matrix	11
1.3.1 Implementación de las matrices en la clase Matrix	12
1.4 Operadores selectores	14
1.4.1 Operador selector por la derecha para la clase Sistema	14
1.4.2 Operador selector por la derecha para la clase Vector	16
1.4.3 Operador selector por la izquierda para la clase Vector	17
1.4.4 Operador selector por la derecha para la clase Matrix	17
1.4.5 Operador transposición de una Matrix	18
1.4.6 Operador selector por la izquierda para la clase Matrix	20
1.5 Operaciones con vectores y matrices	22
1.5.1 Suma de Vectores	22
1.5.2 Producto de un Vector por un escalar a su izquierda	23
1.5.3 Producto de un Vector por un escalar, un Vector o una Matrix a su derecha	23
1.5.4 Igualdad entre vectores	25
1.5.5 Suma de matrices	25
1.5.6 Producto de una Matrix por un escalar a su izquierda	26
1.5.7 Producto de una Matrix por un escalar, un Vector o una Matrix a su derecha	27
1.5.8 Potencias de una Matrix cuadrada	29
1.5.9 Igualdad entre matrices	29
1.6 La clase transformación elemental T	31
1.6.1 Implementación	33
1.6.2 Transposición de transformaciones elementales	35
1.6.3 Inversión de transformaciones elementales	36
1.7 Transformaciones elementales de un Sistema	38
1.8 Transformaciones elementales de una Matrix	39
1.8.1 Transformaciones elementales de las columnas de una Matrix	39
1.8.2 Transformaciones elementales de las filas de una Matrix	39
1.9 La clase SubEspacio (de \mathbb{R}^m)	41
1.10 Librería completa	45
2 Algoritmos del curso	47
2.1 Escalonamiento de una matriz por eliminación Gaussiana	47
2.1.1 Variantes que guardan los pasos dados sobre la matriz original	50
2.2 Inversión de una matriz por eliminación Gaussiana	58
3 Otros trozos de código	62
3.1 Métodos de representación para el entorno Jupyter	62
3.2 Completando la clase Sistema	64
3.2.1 Representación de la clase Sistema	64
3.2.2 Otros métodos de la clase Sistema	65
3.3 Completando la clase Vector	67

3.3.1	Representación de la clase <code>Vector</code>	67
3.3.2	Otros métodos para la clase <code>Vector</code>	67
3.4	Completando la clase <code>Matrix</code>	68
3.4.1	Otras formas de instanciar una <code>Matrix</code>	68
3.4.2	Códigos que verifican que los argumentos son correctos	68
3.4.3	Representación de la clase <code>Matrix</code>	69
3.4.4	Otros métodos para la clase <code>Matrix</code>	69
3.5	Vectores y Matrices especiales	70
3.6	Completando la clase <code>T</code>	72
3.6.1	Otras formas de instanciar una <code>T</code>	72
3.6.2	Representación de la clase <code>T</code>	72
3.7	Completando la clase <code>SubEspacio</code>	74
3.7.1	Representación de la clase <code>SubEspacio</code>	74
3.8	La clase <code>BlockMatrix</code> . Matrices particionadas	74
3.8.1	Particionado de matrices	75
3.8.2	Representación de la clase <code>BlockMatrix</code>	79
A	Sobre este documento	80
A.1	Secciones de código	81

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 final¹ (véase la sección *Literate programming* en la [Página 80](#)). 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!**

¹aquellas 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 L^AT_EX, 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 clase, 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:

$$\left[\begin{smallmatrix} \tau \\ [(5)4] \end{smallmatrix}; \begin{smallmatrix} \tau \\ [(2)3] \end{smallmatrix}; \begin{smallmatrix} \tau \\ [1 \rightleftharpoons 2] \end{smallmatrix} \right]$$

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
```

¹Como 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 75b, 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 75b, Matrix 13, Sistema 7, and Vector 9b.

Un `Sistema` va a ser como una lista de Python, salvo por su modo de representación. Así que vamos a definir los métodos de selección y modificación de sus elementos, la concatenación y un método que cuente el número de elementos del `Sistema` de manera análoga 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. También 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 actúe 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 **Sistemas** 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 **Sistemas** son iguales con "==", y si son distintos con "!=".

6b *<Métodos de la clase Sistema para que actúe 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 66>  
    <Transformaciones elementales de los elementos de un Sistema 38b>  
    <Métodos de representación de la clase Sistema 65a>  
This code is used in chunk 45.  
Defines:  
  Sistema, used in chunks 4–6, 8, 9a, 11, 12, 14, 15, 38, 42, 43, 60b, 65, 66, 68b, and 75a.
```

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

$$\mathbf{v} = (v_1, \dots, v_n)$$

o bien en forma de columna

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

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 67b>
    <Opuesto de un Vector 68a>
    <Representación de la clase Vector 67a>
```

This code is used in chunk 45.
Defines:
`Vector`, used in chunks 4, 5, 8, 9a, 11, 12, 14, 16, 17c, 19–26, 28, 29a, 43, 47, 60b, and 65–70.

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 **Vectores** 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

$\mathbf{A} = [\mathbf{v}_1 \dots \mathbf{v}_n]$, donde las n columnas \mathbf{v}_i 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 75b, 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 Vectores 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 Vectores (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 Vectores con el mismo número de componentes, entonces `self.sis` es el Sistema compuesto por dicha lista de Vectores. (Esto pasa cuando `data` es una lista, tupla o Sistema de Vectores 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 lists, tuples o Sistemas 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 Vectores 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 <Iniciación 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 69a>
        self.sis = Sistema(data)
        <Creación del atributo sis cuando no tenemos una lista de Vectores 68b>

    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 70a>
        <Opuesto de una Matrix 70b>
        <Representación de la clase Matrix 69b>

This code is used in chunk 45.
Defines:
    Matrix, used in chunks 4, 5, 11, 12, 17–21, 24–29, 32, 34c, 35, 39, 42, 43, 48–60, 65b, 66, 69–71, 74, and 78.
```

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 Vectores** en el atributo `sis` de tres modos distintos (el código de los dos últimos se puede consultar en el Capítulo 3 de este documento):

1. Cuando se instancia con una lista, tupla o **Sistema de Vectores**, en el atributo `sis` se almacena el **Sistema** formado por dichos **Vectores**. *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 Sistemas**, 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 **Vectores** 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 $\mathbf{v} = (v_1, \dots, v_n)$ entonces ${}_i|\mathbf{v} = \mathbf{v}|_i = v_i$ para todo $i \in \{1, \dots, n\}$.
- Si $\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}$ entonces $\begin{cases} \mathbf{A}|_j = \begin{pmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{pmatrix} \text{ para todo } j \in \{1, \dots, m\} \\ {}_i|\mathbf{A} = (a_{i1}, \dots, a_{im}) \text{ para todo } i \in \{1, \dots, n\} \end{cases}$.

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

Notación en Mates 2

- ${}_{(i_1, \dots, i_r)}|\mathbf{v} = (\mathbf{v}_{i_1}, \dots, \mathbf{v}_{i_r}) = \mathbf{v}|_{(i_1, \dots, i_r)}$ (es un vector formado por elementos de \mathbf{v})
- ${}_{(i_1, \dots, i_r)}|\mathbf{A} = \begin{bmatrix} {}_{i_1}|\mathbf{A} & \cdots & {}_{i_r}|\mathbf{A} \end{bmatrix}^\top$ (es una matriz cuyas filas son filas de \mathbf{A})
- $\mathbf{A}|_{(j_1, \dots, j_r)} = \begin{bmatrix} \mathbf{A}|_{j_1} & \cdots & \mathbf{A}|_{j_r} \end{bmatrix}$ (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
$\mathbf{v} _i$	<code>v i</code>
${}_i \mathbf{v}$	<code>i v</code>
$\mathbf{A} _j$	<code>A j</code>
${}_i \mathbf{A}$	<code>i A</code>

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 75b, Matrix 13, and Vector 9b.

```

Implementación del operador selector por la derecha para la clase Matrix

Como el objeto `Matrix` es un sistema de `Vectores`, 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 76c>
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 `Vectores`, 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}^\top ; y es la matriz tal que $(\mathbf{A}^\top)_{lj} = {}_j\mathbf{A}$; $j = 1 : n$.

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 “ \top ”. 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 “ \sim ”, símbolo que además deberemos colocar a la izquierda de la matriz.

Mates II	Python
\mathbf{A}^\top	$\sim \mathbf{A}$

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 transposición. Basta instanciar `Matrix` con la lista de `Sistemas` correspondientes a los n `Vectores` 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 75b, 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 76b>

    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

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

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 Vectores 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 \mathbf{a} por un escalar x a su izquierda como el vector tal que

$$\boxed{(x\mathbf{a})_i = x(\mathbf{a}_i)} \quad \text{para } i = 1:n.$$

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 \mathbf{a} por un escalar es conmutativo. Por tanto,

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

cuya transcripción será

$$\mathbf{x} * \text{self}$$

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 **a** y **x** en \mathbb{R}^n es

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

cuya transcripción será

$$\text{sum}([(\text{self}[i])*(\mathbf{x}[i]) \text{ for } i \text{ in range}(1,\text{self}.n+1)])$$

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

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

$$\mathbf{a}\mathbf{X} = \mathbf{X}^T\mathbf{a}$$

cuya transcripción será

$$(\sim \mathbf{x}) * \text{self}$$

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 Vectores 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

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

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

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

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,

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

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 $\mathbf{A}_{m \times n}$ por un vector \mathbf{x} de \mathbb{R}^n a su derecha se define como

$$\mathbf{A}\mathbf{x} = x_1\mathbf{A}_{|_1} + \cdots + x_n\mathbf{A}_{|_n} = \sum_{j=1}^n x_j\mathbf{A}_{|_j} \quad \text{para } j = 1 : n.$$

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

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

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 70c, 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 gaussiano 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 58a>
    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: $\mathbf{A}_{[(\lambda)i+j]}$ suma λ veces la fila i a la fila j ($i \neq j$); $\mathbf{A}_{[(\lambda)i+j]}$ lo mismo con las columnas.

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

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

Comentario sobre la notación. Como una transformación 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* “ τ ” de la transformación elemental a derecha es como si multiplicáramos la matriz \mathbf{A} por la derecha por la correspondiente matriz elemental

$$\mathbf{A}_{\tau_1} = \mathbf{A} \mathbf{I}_{\tau_1} = \mathbf{A} \mathbf{E}_1 \quad \text{donde} \quad \mathbf{E}_1 = \mathbf{I}_{\tau_1} \quad \text{y donde la matriz } \mathbf{I} \text{ es de orden } n.$$

De manera similar, al poner la *abreviatura* “ τ ” de la transformación elemental a izquierda, es como si multiplicáramos la matriz \mathbf{A} por la izquierda por la correspondiente matriz elemental

$$\tau_2 \mathbf{A} = \tau_2 \mathbf{I} \mathbf{A} = \mathbf{E}_2 \mathbf{A} \quad \text{donde} \quad \mathbf{E}_2 = \tau_2 \mathbf{I} \quad \text{y donde la matriz } \mathbf{I} \text{ es de orden } m.$$

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
$\mathbf{A}_{\tau_{[i \rightleftharpoons j]}}$	<code>A & T({i,j})</code>	$\tau_{[i \rightleftharpoons j]} \mathbf{A}$	<code>T({i,j}) & A</code>
$\mathbf{A}_{\tau_{[(a)j]}}$	<code>A & T((a,j))</code>	$\tau_{[(a)j]} \mathbf{A}$	<code>T((a,i)) & A</code>
$\mathbf{A}_{\tau_{[(a)i+j]}}$	<code>A & T((a,i,j))</code>	$\tau_{[(a)i+j]} \mathbf{A}$	<code>T((a,i,j)) & A</code>

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.

²Fíjese 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 $\mathbf{A} \mathbf{E}$ o $\mathbf{E} \mathbf{A}$)

Empleando listas de abreviaturas extendemos la notación para expresar secuencias de transformaciones elementales, es decir, $\tau_1 \cdots \tau_k$. Así logramos la siguiente equivalencia entre expresiones

$$T(t_1) \& T(t_2) \& \cdots \& T(t_k) = T([t_1, t_2, \dots, t_k])$$

De esta manera

$$\mathbf{A}_{\tau_1 \cdots \tau_k} : \quad \mathbf{A} \& T(t_1) \& T(t_2) \& \cdots \& T(t_k) = \mathbf{A} \& T([t_1, t_2, \dots, t_k])$$

$$\tau_1 \cdots \tau_k \mathbf{A} : \quad T(t_1) \& T(t_2) \& \cdots \& T(t_k) \& \mathbf{A} = T([t_1, t_2, \dots, t_k]) \& \mathbf{A}.$$

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

$$\mathbf{A}_{\tau_1 \cdots \tau_k} = \mathbf{A} \mathbf{E}_1 \mathbf{E}_2 \cdots \mathbf{E}_k \quad \text{donde } \mathbf{E}_j = \mathbf{I}_{\tau_j} \text{ y donde } \mathbf{I} \text{ es de orden } n.$$

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

$$\tau_1 \cdots \tau_k \mathbf{A} = \mathbf{E}_1 \mathbf{E}_2 \cdots \mathbf{E}_k \mathbf{A} \quad \text{donde } \mathbf{E}_i = \tau_i \mathbf{I} \text{ y donde } \mathbf{I} \text{ es de orden } m.$$

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 concatenación de todas las abreviaturas

Ejemplos:
>>> # Intercambio entre vectores
>>> T( {1,2} )

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

>>> # Transformación 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

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

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 $\mathbf{A} \ \& \ T(t_1)$ es una nueva matriz con las columnas modificadas. Python no tiene problema en ejecutar $\mathbf{A} \ \& \ T(t_1) \ \& \ T(t_2) \ \& \cdots \ \& \ T(t_k)$ pues ejecutar de izquierda a derecha, es lo mismo que ejecutar $\left[\left[\left[\mathbf{A} \ \& \ T(t_1) \right] \ \& \ T(t_2) \right] \ \& \cdots \right] \ \& \ T(t_k)$ donde la expresión dentro de cada corchete es una `Matrix`, por lo que las operaciones están definidas. La dificultad aparece con

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

Lo primero que Python tratara de ejecutar es $T(t_1) \ \& \ T(t_2)$, pero ni $T(t_1)$ ni $T(t_2)$ 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 $T(t_i) \ \& \ T(t_j)$, 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 `Ts`) empleada para instanciar.

```

33  <Iniciación 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 72>
        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 transformación elemental')
    if isinstance(j,tuple) and (len(j) == 3) and (j[1] == j[2]):
        raise ValueError('T( (a, i, i) ) no es una transformación elemental')
    if isinstance(j,set) and (len(j) > 2) or not j:
        raise ValueError \
            ('El conjunto debe tener uno o dos índices para ser transformación 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 instanciar T con una lista de Ts 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 72.

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 $\mathbf{I}_{\tau_1 \dots \tau_k} = (\mathbf{E}_1 \dots \mathbf{E}_k)$ y puesto que el producto de matrices es asociativo, deducimos que la transpuesta de $\mathbf{I}_{\tau_1 \tau_2 \dots \tau_k}$ es

$$(\mathbf{I}_{\tau_1 \tau_2 \dots \tau_k})^T = (\mathbf{I} \mathbf{E}_1 \dots \mathbf{E}_k)^T = \mathbf{E}_k^T \dots \mathbf{E}_1^T \mathbf{I} = \mathbf{I}_{\tau_k \dots \tau_2 \tau_1}$$

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 sugiere denotar a la operación de invertir el orden de las transformaciones como una transposición:

$$(\tau_1 \dots \tau_k)^T = \tau_k \dots \tau_1;$$

así

$$(\mathbf{A}_{\tau_1 \dots \tau_k})^T = (\tau_1 \dots \tau_k)^T \mathbf{A}^T = \tau_k \dots \tau_1 \mathbf{A}^T$$

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.

³Recuerde 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 $\mathbf{I}_{\tau_1 \dots \tau_k}$ o de la forma $\tau_k \dots \tau_1 \mathbf{I}$ es invertible por ser producto de matrices elementales, en particular,

$$\left(\mathbf{I}_{\tau_1 \dots \tau_k} \right) \left(\mathbf{I}_{\tau_k^{-1} \dots \tau_1^{-1}} \right) = \mathbf{E}_1 \dots \mathbf{E}_k \cdot \mathbf{E}_k^{-1} \dots \mathbf{E}_1^{-1} = \mathbf{I}_{\tau_1 \dots \tau_k \cdot \tau_k^{-1} \dots \tau_1^{-1}} = \mathbf{I};$$

por lo que podemos denotar por $\left(\tau_1 \dots \tau_k \right)^{-1}$ a la sucesión de transformaciones $\tau_k^{-1} \dots \tau_1^{-1}$. De este modo

$$\left(\mathbf{I}_{\tau_1 \dots \tau_k} \right)^{-1} = \mathbf{I}_{(\tau_1 \dots \tau_k)^{-1}}.$$

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 Transformación 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 73>
```

This code is used in chunk 45.
Defines:
T, used in chunks 32, 34–40, 48, 49a, 51–60, 72, and 73.

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 transformaciones elementales de las columnas de una **Matrix**. Puesto que cada **Matrix** es un **Sistema** de vectores, en la librería vamos a comenzar 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: $\sim(\sim \text{self} \ \& \ t)$. Pero hay que recordar que las transformaciones más próximas a la matriz se ejecutan primero, puesto que $\tau_1 \dots \tau_k \mathbf{A} = \mathbf{E}_1 \mathbf{E}_2 \dots \mathbf{E}_k \mathbf{A}$. Con la función `reversed` aplicamos la sucesión de transformaciones en el orden inverso a como aparecen en la lista:

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

```

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

La clase `SubEspacio` 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*:

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

donde el vector \mathbf{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:

$$\{ \mathbf{v} \in \mathbb{R}^3 \mid [-3 \quad 0 \quad 2] \mathbf{v} = \mathbf{0} \}$$

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`.
- `Rm`: indica al espacio \mathbb{R}^m al que pertenece. En el ejemplo anterior `Rm=3` puesto que es un subespacio de \mathbb{R}^3 .
- `base`: una base del subespacio (un `Sistema` de `Vectores` de `Rm`).
- `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.

Implementación . La implementación requiere del uso del algoritmos que se describen en el capítulo siguiente. Por una parte, es necesario escalar una `Matrix` para encontrar un `Sistema` de `Vectores` linealmente independientes y formar con ellos un `Sistema` generador del `SubEspacio`. Lo haremos con el algoritmo `ECLsd` sencillamente para evitar fracciones, pero esta decisión es completamente arbitraria. Además, también necesitaremos resolver un sistema de ecuaciones homogéneo con el algoritmo `Homogenea`.

```

42 <Iniciación de la clase SubEspacio 42>≡
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          = ECLsd(A)
        self.dim   = L.rank
        self.Rm    = A.m
        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)
    if isinstance(data, Matrix):
        A          = data
        H          = Homogenea(A)
        self.sgen  = H.sgen
        self.dim   = 0 if H.determinado else len(self.sgen)
        self.Rm    = A.n
        self.base  = self.sgen if self.dim else Sistema([])
        self.cart  = ~Matrix(Homogenea(~Matrix(self.sgen)).sgen)

```

This code is used in chunk 44.

Defines:

SubEspacio, used in chunks 43, 44, and 74.

Uses ECLsd 53, Homogenea 60b, Matrix 13, Sistema 7, and V0 70c.

```

43 <Métodos de la clase SubEspacio 43>≡
    def verificacion(self, other):
        if not isinstance(other, SubEspacio) or not self.Rm == other.Rm:
            raise \
                ValueError('Ambos argumentos deben ser SubEspacios de un mismo espacio')

    def contenidoEn(self, other):
        """Indica si un subespacio de Rn contiene a otro"""
        self.verificacion(other)
        M = Matrix(BlockMatrix([[Matrix(other.sgen), Matrix( self.sgen)]]))
        return True if (ECL(M).rank == other.dim) else False

    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(\
                'Argumento: Vector con el número adecuado de componentes')
        return (self.cart*other==V0(self.cart.m))

    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)

    def __add__(self, other):
        """Devuelve la suma de subespacios de Rn"""
        self.verificacion(other)
        return SubEspacio(Sistema(self.sgen + other.sgen))

    def __and__(self, other):
        """Devuelve la intersección de subespacios"""
        self.verificacion(other)
        M = Matrix(BlockMatrix([ [ self.cart], [other.cart] ]))
        return SubEspacio(M)

    def __invert__(self):
        """Devuelve el complemento ortogonal"""
        return SubEspacio(Sistema((~self.cart).sis))

```

This code is used in chunk 44.

Uses BlockMatrix 75b, Matrix 13, Sistema 7, SubEspacio 42, V0 70c, and Vector 9b.

```
44  <La clase SubEspacio 44>≡  
    class SubEspacio:  
        <Inicialización de la clase SubEspacio 42>  
        <Métodos de la clase SubEspacio 43>  
        <Métodos de representación de la clase SubEspacio 74>
```

This code is used in chunk 45.

Uses SubEspacio 42.

1.10 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
```

```
45 <notacion.py 45>≡
    # coding=utf8

    from fractions import Fraction

    <Método html general 62a>
    <Método latex general 62b>
    <Métodos html y latex para fracciones 64>

    <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 75b>

    <Definición del método particion 76a>
    <Definición del procedimiento de generación del conjunto clave para particionar 78a>

    <Definición de vector nulo: V0 70c>
    <Definición de matriz nula: M0 71a>
    <Definición de la matriz identidad: I 71b>

    <Método pivote calcula el índice del primer coef. no nulo de un Vector a partir de cierta posición 47>
    <Escalonamiento de una matriz mediante eliminación por columnas 48>
    <Escalonamiento de una matriz mediante eliminación por filas 49b>
    <Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52>
    <Normalizando la diagonal principal para que sus componentes no nulos sean unos 57b>
    <Variantes del escalonamiento por eliminación por filas que guardan los pasos dados 56a>
    <Invirtiendo una matriz 58b>
    <La clase Sistema 7>
    <La clase SubEspacio 44>
    <Resolviendo un sistema homogéneo 60b>
    <normal 59c>
    <sistema 60a>

    Root chunk (not used in this document).
```

⁴el 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 Escalonamiento de una matriz por 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 definición de pivote tanto si programamos el método de eliminación por filas como por columnas.

Por conveniencia, el método `pivote` 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**).

47 `<Método pivote calcula el índice del primer coef. no nulo de un Vector a partir de cierta posición 47>≡`

```
def pivote(v, k=0):  
    """  
    Devuelve el primer índice(i) mayor que k de un coeficiente(c) no  
    nulo del Vector v. En caso de no existir devuelve 0  
    """  
    return ( [i for i,c in enumerate(v.sis, 1) if (c!=0 and i>k)] + [0] )[0]
```

This code is used in chunk 45.
Defines:
 `pivote`, used in chunks 48–50, 52–57, and 59c.
Uses **Vector** 9b.

En las notas de la asignatura decimos que la matriz **L** es *escalonada*, si toda columna que precede a una no nula $L_{|k}$ no es nula y su posición de pivote es anterior a la posición de pivote de $L_{|k}$. 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 demás nos indica el procedimiento para programar el método.

El procedimiento es iterativo. La variable `r` es un contador de pivotes encontrados en la matriz escalonada (inicialmente cero). Vamos proceder iterativamente comenzando por la primera fila. Primero buscamos la posición de pivote de la fila en que vamos a eliminar componentes.

- Si `p` es cero, quiere decir que todos los componentes son cero y hemos acabado con la eliminación para esta fila y pasamos a la siguiente.
- Si `p` es positivo, quiere decir que en dicha fila almenos hay un componente distinto de cero en una columna de índice mayor que `r`. Entonces hemos encontrado una nueva columna no nula y consecuentemente hemos encontrado un nuevo pivote. Así que debemos proceder a eliminar de izquierda a derecha hasta anular los componentes de la fila que están más a la izquierda de la posición `r`.

- El contador de pivotes aumenta en una unidad: $r+=1$
- Como en el cuarto caso de la demostración por inducción del teorema, *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).
- Como en el tercer caso de la demostración por inducción del teorema, una vez reordenadas las columnas, aplicamos la sucesión de transformaciones elementales Tipo I, $(\text{Fraction}(-(i|A|j), (i|A|r)), r, j)$:

$$\left[\begin{pmatrix} \tau \\ -a_{ij} \\ a_{ir} \end{pmatrix} r+j \right], \quad \text{con } j = r : n.$$

Tenga en cuenta que en el teorema sólo se indica el paso para la primera fila no nula (es decir, para $r=1$).¹ Pero en este algoritmo se continúa fila a fila con el mismo procedimiento hasta escalonar toda la matriz.

48 $\langle \text{Escalonamiento de una matriz mediante eliminación por columnas 48} \rangle \equiv$

```
class GaussCL(Matrix):
    def __init__(self, data):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
        A = Matrix(data)
        r = 0
        for i in range(1,A.m+1):
            p = pivote((i|A),r)
            if p > 0:
                r += 1
                A & T( {p, r} )
                A & T( [(Fraction(-(i|A|j), (i|A|r)), r, j) for j in range(r+1,A.n+1)] )

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

This definition is continued in chunks 49a and 50a.
 This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

Este procedimiento da lugar a operaciones con fracciones, ya que para eliminar el número b usando a , 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 . Pero para escalonar una matriz con componentes enteros no es necesario trabajar con fracciones. Podemos eliminar el número b usando a encadenando dos operaciones:

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

Es decir, multiplicamos b por a y luego restamos ba . Del modo análogo, podemos aplicar la sucesión de pares de transformaciones elementales Tipo II y Tipo I:

$$T([(-(i|A|r), j), ((i|A|j), r, j)]) \longrightarrow \left[\begin{pmatrix} \tau \\ (-a_{ir})j \end{pmatrix} \left[\begin{pmatrix} \tau \\ a_{ij} \end{pmatrix} r+j \right] \right], \quad \text{con } j = r : n.$$

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 los números de la matriz crecerían sin que ello fuera estrictamente necesario. Y 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 multiplicar b por el denominador de la fracción simplificada y a por el numerador. El siguiente código usa esta idea además del hecho de que si n es un número del tipo `Fraction`, es decir, de la forma $\frac{a}{b}$, entonces `n.numerator` nos da el numerador de la fracción simplificada y `n.denominator` es el denominador.

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

49a *<Escalonamiento de una matriz mediante eliminación por columnas 48>+≡*

```

class GaussCLsd(Matrix):
    def __init__(self, data):
        """Escalona una Matrix con eliminación por columnas (sin div.)"""
        A = Matrix(data)
        r = 0
        for i in range(1,A.m+1):
            p = pivote((i|A),r)
            if p > 0:
                r += 1
                A & T( {p, r} )
                A & T([ T([ Fraction((i|A|j),(i|A|r)).denominator, j), \
                        (-Fraction((i|A|j),(i|A|r)).numerator, r, j))] \
                      for j in range(r+1,A.n+1)])

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

```

This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

En la mayoría de los libros de Álgebra Lineal, la eliminación gaussiana consiste en lograr una matriz triangular superior mediante operaciones con las *filas* siguiendo el esquema (NO→SE); es decir, colocando el primer pivote en la primera fila y haciendo ceros por debajo. Pero esto lo podemos lograr con el algoritmo anterior. Basta transponer la matriz, escalonar operando con las columnas, y volver a transponer el resultado:

49b *<Escalonamiento de una matriz mediante eliminación por filas 49b>≡*

```

class GaussFU(Matrix):
    def __init__(self, data):
        """Escalona una Matrix con eliminación por filas (transf. Gauss)"""
        A = ~GaussCL(~Matrix(data))
        super(self.__class__ ,self).__init__(A.sis)

```

This code is used in chunk 45.
 Uses Matrix 13.

También es arbitrario el orden en que se triangula la matriz. Por ejemplo, podemos obtener una matriz triangular superior mediante eliminación por columnas si en lugar de “ir de arriba a abajo” y de “izquierda a derecha” (escalando desde la esquina Noroeste a la esquina Sureste: NO→SE), comenzamos por las filas inferiores y realizando la eliminación de derecha a izquierda (SE→NO).

Si aplicamos el método `reversed` de la clase `Matrix` sobre `A` obtenemos una matriz cuyas columnas son las de `A` pero en orden inverso (primero la última, luego la penúltima, etc.). Así que si aplicamos el método `reversed` sobre la transpuesta de `A` y volvemos a transponer, obtenemos una matriz cuyas filas aparecen en el orden inverso de las de `A`. Si aplicamos de nuevo `reversed` habremos invertido el orden tanto de las filas como de las columnas.

Por tanto, si invertimos el orden de las filas y columnas, escalonamos con `GaussCL` y volvemos a invertir el orden de las filas y las columnas, obtenemos el citado escalonamiento (SE→NO) que arroja una matriz triangular superior:

50a *⟨Escalonamiento de una matriz mediante eliminación por columnas 48⟩+≡*

```
class GaussCU(Matrix):
    def __init__(self, data):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
        A = reversed(~reversed(~ GaussCL( reversed(~reversed(~ Matrix(data) ))) ))
        super(self.__class__, self).__init__(A.sis)
```

This code is used in chunk 45.
Uses Matrix 13.

2.1.1 Variantes que guardan los pasos dados sobre la matriz original

Las variantes de más abajo tienen un código con la misma estructura: con dos trozos de código comunes a todas las variantes (uno al principio, *⟨Definición del método PasosYEscritura 51a⟩* y otro al final *⟨Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b⟩* que se describen un poco más abajo). Entre ambos trozos aparece el código que aplica las secuencias de transformaciones elementales necesarias en cada variante.

50b *⟨Estructura de las variantes de aplicación del método de Gauss 50b⟩≡*

```
class NombreMetodo(Matrix):
    def __init__(self, data, rep=0):
        """Descripción de lo que hace la variante particular"""

        ⟨Definición del método PasosYEscritura 51a⟩

        A = Matrix(data); pasos = []; r = 0

        for i in (indices_en_el_orden_en_el_que_se_recorren_las_filas_de_A):

            p = pivote(fila_en_el_orden_en_el_que_se_recorren_las_columnas, r)

            if p > 0:
                # si hay nuevo pivote en la fila
                r += 1
                # el contador cuenta un nuevo pivote y ...

                Se_definen_las_transf_necesarias,
                se_almacenan_en_la_variable_pasos,
                y_se_aplican_sobre_A

        ⟨Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b⟩
```

Root chunk (not used in this document).
Uses Matrix 13 and pivote 47.

Si queremos mostrar los pasos, es más legible mostrar solo los que realmente cambian 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).

Además, si hemos encadenado varios procedimientos de eliminación, deberíamos poder ver los pasos desde el principio hasta el final; añadiendo a la escritura en L^AT_EX de dicho pasos previos (TexPasosPrev), los pasos dados como argumento. Si TexPasosPrev es vacío, la escritura comienza con la representación de data. Todo esto es lo que hace el método PasosYEscritura:

51a *⟨Definición del método PasosYEscritura 51a⟩*≡

```
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 50b and 52–60.
Uses BlockMatrix 75b, Matrix 13, and T 37b.

Al representar los pasos de eliminación, consideramos si `data` fue obtenido mediante un proceso previo de eliminación, para mostrar el proceso completo. El atributo `tex` guarda el código \LaTeX que muestra el proceso completo. El atributo `rank` guarda el rango y `pasos` las listas de transformaciones elementales empleadas.

51b *⟨Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b⟩*≡

```
pasosPrevios = data.pasos if hasattr(data, 'pasos') and data.pasos else [[],[]]
TeXPasosPrev = data.tex if hasattr(data, 'tex') and data.tex else []
self.tex = PasosYEscritura(data, pasos, TeXPasosPrev)
if rep:
    from IPython.display import display, Math
    display(Math(self.tex))
self.rank = r
pasos[0] = pasos[0] + pasosPrevios[0]
pasos[1] = pasosPrevios[1] + pasos[1]
self.pasos = pasos
super(self.__class__,self).__init__(A.sis)
```

This code is used in chunks 50b and 52–57.

La primera variante hace lo mismo que GaussCL pero guarda los pasos dados en el atributo `pasos` y en el atributo `tex` el código L^AT_EX que permite representar el procedimiento de eliminación gaussiana paso a paso.

52 *<Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52>≡*

```

class ECL(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
        <Definición del método PasosYEscritura 51a>
        A = Matrix(data); pasos = [[],[]]; r = 0
        for i in range(1,A.m+1):
            p = pivote((i|A),r)
            if p > 0:
                r += 1
                Tr = T([ {p, r} ])
                pasos[1] += [Tr]
                A & T( Tr )
                Tr = T([(Fraction(-(i|A|j),(i|A|r)), r, j) for j in range(r+1,A.n+1)])
                pasos[1] += [Tr] if Tr.t else []
                A & T( Tr )
        <Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b>

```

This definition is continued in chunks 53–55.
 This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

La siguiente variante evita las divisiones

53 *<Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52>+≡*

```

class ECLsd(Matrix):
    def __init__(self, data, rep=0):
        """Escalona por eliminación por columnas (sin divisiones)"""
        <Definición del método PasosYEscritura 51a>
        A = Matrix(data); pasos = [[],[]]; r = 0
        for i in range(1,A.m+1):
            p = pivote((i|A),r)
            if p > 0:
                r += 1
                Tr = T( [ {p, r} ] )
                pasos[1] += [Tr]
                A & T( Tr )
                Tr = T( [ T( [ ( Fraction((i|A|j),(i|A|r)).denominator, j), \
                               (-Fraction((i|A|j),(i|A|r)).numerator, r, j) ] ) \
                          for j in range(r+1,A.n+1) ] )
                pasos[1] += [Tr] if Tr.t else []
                A & T( Tr )
        <Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b>

```

This code is used in chunk 45.
 Defines:
 ECLsd, used in chunks 42 and 60b.
 Uses Matrix 13, pivote 47, and T 37b.

La siguiente variante (parecida a la primera) asegura que los pivotes sean todos iguales a uno

54a \langle Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52 $\rangle + \equiv$

```

class ECLN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona por eliminación por columnas haciendo pivotes unitarios"""
         $\langle$ Definición del método PasosYEscritura 51a $\rangle$ 
        A = Matrix(data); pasos = [[],[]]; r = 0
        for i in range(1,A.m+1):
            p = pivote((i|A),r)
            if p > 0:
                r += 1
                Tr = T( [ {p, r} ] )
                pasos[1] += [Tr]
                A & T( Tr )
                Tr = T( [ (Fraction(1,(i|A|r)), r) ] )
                pasos[1] += [Tr]
                A & T( Tr )
                Tr = T( [(-i|A|j), r, j) for j in range(r+1,A.n+1)] )
                pasos[1] += [Tr] if Tr.t else []
                A & T( Tr )
         $\langle$ Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b $\rangle$ 

```

This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

La siguiente variante logra una matriz triangular superior eliminando de derecha a izquierda y de abajo a arriba.

54b \langle Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52 $\rangle + \equiv$

```

class ECU(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
         $\langle$ Definición del método PasosYEscritura 51a $\rangle$ 
        A = Matrix(data); pasos = [[],[]]; r = 0
        for i in reversed(range(1,A.m+1)):
            p = pivote(reversed(i|A), r)
            if p > 0:
                r += 1
                Tr = T( [ {A.n-p+1, A.n-r+1} ] )
                pasos[1] += [Tr]
                A & T( Tr );
                Tr = T([ (Fraction(-(i|A|j), (i|A|(A.n-r+1)))), A.n-r+1, j) \
                        for j in reversed(range(1,A.n-r+1)) ] )
                pasos[1] += [Tr] if Tr.t else []
                A & T( Tr )
         $\langle$ Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b $\rangle$ 

```

This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

La siguiente variante hace lo mismo que la anterior, pero asegurandose de que los pivotes sean unos.

55a \langle Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52 $\rangle + \equiv$

```

class ECUN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
         $\langle$ Definición del método PasosYEscritura 51a $\rangle$ 
        A = Matrix(data); pasos = [[], []]; r = 0
        for i in reversed(range(1, A.m+1)):
            p = pivote(reversed(i|A), r)
            if p > 0:
                r += 1
                Tr = T( [ {A.n-p+1, A.n-r+1} ] )
                pasos[1] += [Tr]
                A & T( Tr )
                Tr = T( [ (Fraction(1, (i|A|(A.n-r+1))), A.n-r+1 ) ] )
                pasos[1] += [Tr]
                A & T( Tr )
                Tr = T([ -(i|A|j), A.n-r+1, j) for j in reversed(range(1, A.n-r+1)) ] )
                pasos[1] += [Tr] if Tr.t else []
                A & T( Tr )
         $\langle$ Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b $\rangle$ 

```

This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

La siguiente variante también obtiene una forma escalonada triangular superior, pero evitando realizar divisiones.

55b \langle Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52 $\rangle + \equiv$

```

class ECUsd(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por columnas (transf. Gauss)"""
         $\langle$ Definición del método PasosYEscritura 51a $\rangle$ 
        A = Matrix(data); pasos = [[], []]; r = 0
        for i in reversed(range(1, A.m+1)):
            p = pivote(reversed(i|A), r)
            if p > 0:
                r += 1; Tr = T( [ {A.n-p+1, A.n-r+1} ] ); pasos[1] += [Tr]
                A & T( Tr )
                Tr = T( [ T( \
                    [ ( Fraction((i|A|j), (i|A|(A.n-r+1))).denominator, j), \
                      (-Fraction((i|A|j), (i|A|(A.n-r+1))).numerator, A.n-r+1, j) ] \
                    ) for j in reversed(range(1, A.n-r+1)) ] )
                pasos[1] += [Tr] if Tr.t else []
                A & T( Tr )
         $\langle$ Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b $\rangle$ 

```

This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

Al operar con las filas, el método PasosYEscritura cambia, pues hay que tener en cuenta que las transformaciones elementales actúan por la izquierda.

56a *<Variantes del escalonamiento por eliminación por filas que guardan los pasos dados 56a>≡*

```

class EFU(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por filas (transf. Gauss)"""
        <Definición del método PasosYEscritura 51a>
        A = Matrix(data); pasos = [[],[]]; r = 0
        for j in range(1,A.n+1):
            p = pivote((A|j),r)
            if p > 0:
                r += 1
                Tr = T( [ {p, r} ] ); pasos[0] += [Tr]
                T( Tr ) & A
                Tr = T( [(Fraction(-(i|A|j),(r|A|r)), r, i) for i in range(r+1,A.m+1)] )
                pasos[0] += list(reversed([Tr]))
                T( Tr ) & A
            pasos[0]=list(reversed(pasos[0]))
        <Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b>

```

This definition is continued in chunks 56b and 57a.
This code is used in chunk 45.
Uses Matrix 13, pivote 47, and T 37b.

La siguiente variante hace lo mismo que la anterior, pero asegurándose de que los pivotes sean unos.

56b *<Variantes del escalonamiento por eliminación por filas que guardan los pasos dados 56a>+≡*

```

class EFUN(Matrix):
    def __init__(self, data, rep=0):
        """Escalona con eliminación por filas haciendo pivotes unitarios"""
        <Definición del método PasosYEscritura 51a>
        A = Matrix(data); pasos = [[],[]]; r = 0
        for j in range(1,A.n+1):
            p = pivote((A|j),r)
            if p > 0:
                r += 1
                Tr = T( [ {p, r} ] ); pasos[0] += [Tr]
                T( Tr ) & A
                Tr = T( [ (Fraction(1,(j|A|r)), r) ] ); pasos[0] += [Tr]
                T( Tr ) & A
                Tr = T( [(-(i|A|j), r, i) for i in range(r+1,A.m+1)] )
                pasos[0] += list(reversed([Tr])) if Tr.t else []
                T( Tr ) & A
            pasos[0]=list(reversed(pasos[0]))
        <Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b>

```

This code is used in chunk 45.
Uses Matrix 13, pivote 47, and T 37b.

La siguiente variante obtiene una matriz triangulas inferior operando con las filas de “abajo a arriba” y de “derecha a izquierda”

57a *<Variantes del escalonamiento por eliminación por filas que guardan los pasos dados 56a>+=*

```

class EFL(Matrix):
    def __init__(self, data, rep=0):
        """Escalona una Matrix con eliminación por filas (L)"""
        <Definición del método PasosYEScritura 51a>
        A = Matrix(data); pasos = [[],[]]; r = 0
        for j in reversed(range(1,A.n+1)):
            p = pivote(reversed(A[j]),r)
            if p > 0:
                r += 1
                Tr = T( [ {A.m-p+1, A.m-r+1} ] );      pasos[0] += [Tr]
                T( Tr ) & A
                Tr = T([ (Fraction(-(i|A[j]), ((A.m-r+1)|A[j])), A.m-r+1, i) \
                        for i in (range(1,A.m-r+1)) ] )
                pasos[0] += [Tr] if Tr.t else []
                T( Tr ) & A
        pasos[0]=list(reversed(pasos[0]))
        <Se almacenan los atributos tex, pasos y rank; y se devuelve el resultado 51b>

```

This code is used in chunk 45.
 Uses Matrix 13, pivote 47, and T 37b.

Si hemos diagonalizado y solo nos falta que la diagonal esté compuesta por unos, lo podemos hacer dividiendo cada columna por el valor de elemento de la diagonal (si no es nulo).

57b *<Normalizando la diagonal principal para que sus componentes no nulos sean unos 57b>≡*

```

class NormDiag(Matrix):
    def __init__(self, data, rep=0):
        """Normaliza a uno los componentes no nulos de la diagonal principal"""
        <Definición del método PasosYEScritura 51a>
        A = Matrix(data); pasos = [[],[]];
        Tr = T([ (Fraction(1,(j|A[j])), j) for j in range(1,A.n+1) if (j|A[j])!=0 ] )
        pasos[1] = [Tr] if Tr.t else []
        A & T( [Tr] )
        PPrevios = data.pasos if hasattr(data, 'pasos') and data.pasos else [[],[]]
        TexPPrev = data.tex if hasattr(data, 'tex') and data.tex else []
        self.tex = PasosYEScritura(data, pasos, TexPPrev)
        if rep:
            from IPython.display import display, Math
            display(Math(self.tex))
        self.rank = data.rank if hasattr(data, 'rank') and data.rank else []
        pasos[1] = PPrevios[1] + pasos[1]
        self.pasos = pasos
        super(self.__class__,self).__init__(A.sis)

```

This code is used in chunk 45.
 Uses Matrix 13 and T 37b.

2.2 Inversión de una matriz por eliminación Gaussiana

58a *⟨Método auxiliar que calcula la inversa de una Matrix 58a⟩≡*

```
def MatrixInversa( self ):
    """Calculo de la inversa de una matriz"""
    L = ECL(self)

    if L.rank < L.n:
        raise ArithmeticError('Matrix singular')

    return Matrix( I(L.n) & T(ECUN(L).pasos[1]) )
```

This code is used in chunk 29b.
Uses 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

58b *⟨Invirtiendo una matriz 58b⟩≡*

```
class InvMat(Matrix):
    def __init__(self, data, rep=0):
        """Devuelve la matriz inversa y los pasos dados sobre las columnas"""
        ⟨Definición del método PasosYEscritura 51a⟩
        A = Matrix(data)

        if A.m != A.n:
            raise ValueError('Matrix no cuadrada')

        L = ECL(A)
        if L.rank < A.n:
            raise ArithmeticError('Matrix singular')

        M = ECUN(L)
        stack = BlockMatrix([[A],[I(A.n)]])
        self.tex = PasosYEscritura(stack, M.pasos)
        if rep:
            from IPython.display import display, Math
            display(Math(self.tex))

        Inv = I(A.n) & T(M.pasos[1])
        self.pasos = M.pasos
        super(self.__class__,self).__init__(Inv.sis)
```

This definition is continued in chunk 59.
This code is used in chunk 45.
Uses BlockMatrix 75b, 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

59a *⟨Invirtiendo una matriz 58b⟩*+≡

```
class InvMatF(Matrix):
    def __init__(self, data, rep=0):
        """Devuelve la matriz inversa y los pasos dados sobre las filas"""
        ⟨Definición del método PasosYEscritura 51a⟩
        A = Matrix(data)
        Id = EFUN(EFL(A))
        stack = BlockMatrix([[A,I(A.m)]])
        self.tex = PasosYEscritura(stack, Id.pasos)
        if rep:
            from IPython.display import display, Math
            display(Math(self.tex))

        Inv = T(Id.pasos[0]) & I(A.n)
        self.pasos = Id.pasos
        super(self.__class__,self).__init__(Inv.sis)
```

This code is used in chunk 45.

Uses BlockMatrix 75b, Matrix 13, and T 37b.

El siguiente código obtiene la inversa de una matriz operando primero sobre las filas hasta obtener una matriz triangular superior con pivotes iguales a uno y luego operando sobre las columnas hasta obtener la identidad.

59b *⟨Invirtiendo una matriz 58b⟩*+≡

```
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 51a⟩
        A = Matrix(data)
        Id = ECLN(EFU(A))
        stack = BlockMatrix([[A,I(A.m)], [I(A.n),M0(A.m,A.n)]])
        self.tex = PasosYEscritura(stack, Id.pasos)
        if rep:
            from IPython.display import display, Math
            display(Math(self.tex))

        Inv = ( I(A.n) & T(Id.pasos[1]) ) * ( T(Id.pasos[0]) & I(A.n) )
        self.pasos = Id.pasos
        super(self.__class__,self).__init__(Inv.sis)
```

This code is used in chunk 45.

Uses BlockMatrix 75b, Matrix 13, and T 37b.

59c *⟨normal 59c⟩*≡

```
class Normal(Matrix):
```

```

def __init__(self, data):
    """Escalona por Gauss obteniendo una matriz cuyos pivotes son unos"""
    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 45.

Uses Matrix 13, pivote 47, and T 37b.

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

60a

<sistema 60a>≡

```

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 45.

Uses BlockMatrix 75b and Matrix 13.

El siguiente código obtiene una base del espacio nulo de una matriz

60b

<Resolviendo un sistema homogéneo 60b>≡

```

class Homogenea:
    def __init__(self, data, rep=0):
        """Describe el espacio nulo de una matriz y los pasos para encontrarlo"""
        <Definición del método PasosYEScritura 51a>
        A = Matrix(data)
        L = ECLsd(A)
        E = I(A.n) & T(L.pasos[1])

        base = [Vector(E|j) for j in range(L.rank+1, L.n+1)]
        dim = len(base)
        self.determinado = (dim == 0)
        self.sgen = Sistema(base) if dim else Sistema([V0(A.n)])

        self.tex = PasosYEScritura(BlockMatrix([[A],[I(A.n)]]), L.pasos)
        if rep:
            from IPython.display import display, Math
            display(Math(self.tex))

    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 45.

Defines:

Homogenea, used in chunk 42.

Uses BlockMatrix 75b, ECLsd 53, Matrix 13, Sistema 7, T 37b, V0 70c, and Vector 9b.

Capítulo 3

Otros trozos de código

3.1 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 gráfica correspondiente.

62a *⟨Método html general 62a⟩≡*

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

This code is used in chunk 45.

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 truíqui recursivo para que trate de manera parecida la expresiones en `LATEX` y los tipos de datos que corresponden a números `int` o `float`).

62b *⟨Método latex general 62b⟩≡*

```
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 45.

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

¹resulta 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`).

²más 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 fracción $\frac{a}{b}$ 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`).

64 *<Métodos html y latex para fracciones 64>≡*

```

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 45.

3.2 Completando la clase Sistema

3.2.1 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 “;”

$$\mathbf{v} = [v_1; \dots; v_n]$$

Definimos tres representaciones distintas. Una para la línea de comandos de Python de manera que “abra” el corchete “[” 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 línea 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.

65a *<Métodos de representación de la clase Sistema 65a>=*

```
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.

3.2.2 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**).

65b *<Texto de ayuda para el operador producto por la derecha en la clase Sistema 65b>=*

```
"""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 66.
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 elementos 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.

```
66 <Producto de un Sistema por un Vector o una Matrix a su derecha 66>≡
def __mul__(self,x):
    <Texto de ayuda para el operador producto por la derecha en la clase Sistema 65b>
    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.
```

3.3 Completando la clase Vector

3.3.1 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

$$\mathbf{v} = (v_1, \dots, v_n)$$

o bien en forma de columna

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

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 línea 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.

```
67a <Representación de la clase Vector 67a>≡
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.

3.3.2 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.

```
67b <Reverso de un Vector 67b>≡
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 bien manejar el opuesto de un vector

68a *<Opuesto de un Vector 68a>≡*

```

def __neg__(self):
    """Devuelve el opuesto de un Vector"""
    return -1*self
This code is used in chunk 9b.
Uses Vector 9b.

```

3.4 Completando la clase Matrix

3.4.1 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.

68b *<Creación del atributo sis cuando no tenemos una lista de Vectores 68b>≡*

```

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 68c>≡
    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 75b, Sistema 7, and Vector 9b.

```

3.4.2 Códigos que verifican que los argumentos son correctos

68c *<Verificación de que todas las filas de la matriz tendrán la misma longitud 68c>≡*

```

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 68b.

69a *<Verificación de que todas las columnas de la matriz tienen la misma longitud 69a>≡*

```
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.

3.4.3 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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 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.

69b *<Representación de la clase `Matrix` 69b>≡*

```
def __repr__(self):
    """ Muestra una matriz en su representación python """
    return 'Matrix(' + repr(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 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.

3.4.4 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.

70a *<Reverso de una Matrix 70a>*≡

```
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 bien manejar el opuesto de una Matrix

70b *<Opuesto de una Matrix 70b>*≡

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

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

3.5 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**.

70c *<Definición de vector nulo: V0 70c>*≡

```
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 45.
Defines:
V0, used in chunks 29a, 42, 43, 60b, and 71a.
Uses Vector 9b.

Y lo mismo hacemos para matrices

71a

```

<Definición de matriz nula: M0 71a>≡
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 45.
 Uses Matrix 13 and V0 70c.

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

- \mathbf{I} (de orden n) es la matriz tal que ${}_i\mathbf{I}_j = \begin{cases} 1 & \text{si } j = i \\ 0 & \text{si } j \neq i \end{cases}$.

71b

```

<Definición de la matriz identidad: I 71b>≡
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 45.
 Uses Matrix 13.

3.6 Completando la clase T

3.6.1 Otras formas de instanciar una T

Si se instancia `T` usando otra Transformación elemental, sencillamente se copia el atributo `t`. Si se instancia `T` usando una lista (no vacía) de Transformaciones elementales, el atributo `t` será la lista de abreviaturas resultante de concatenar las abreviaturas de todas las Transformaciones elementales de la lista empleada en la instanciación.

72 \langle Creación del atributo `t` cuando se instancia con otra `T` o lista de `Ts` 72 $\rangle \equiv$

```

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.

3.6.2 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
<code>T({1, 5})</code>	$\tau_{[1 \rightleftharpoons 5]}$
<code>T((6, 2))</code>	$\tau_{[(6)2]}$
<code>T((-1, 2, 3))</code>	$\tau_{[(-1)2+3]}$

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
$\tau_{[i \rightleftharpoons j]} \mathbf{A}$	<code>T({i,j}) & A</code>	$\mathbf{A} \tau_{[i \rightleftharpoons j]}$	<code>A & T({i,j})</code>
$\tau_{[(a)i]} \mathbf{A}$	<code>T((a,i)) & A</code>	$\mathbf{A} \tau_{[(a)i]}$	<code>A & T((a,i))</code>
$\tau_{[(a)i+j]} \mathbf{A}$	<code>T((a,i,j)) & A</code>	$\mathbf{A} \tau_{[(a)i+j]}$	<code>A & T((a,i,j))</code>

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 transformación 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
$T([(2, 1), (1, 3, 2)])$	$\tau_{\begin{bmatrix} (2) \mathbf{1} \\ (1) \mathbf{3+2} \end{bmatrix}}$

```
73 <Representación de la clase T 73>≡
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 transformación 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.

3.7 Completando la clase SubEspacio

3.7.1 Representación de la clase SubEspacio

74 *<Métodos de representación de la clase SubEspacio 74>≡*

```
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.Rm) + \
        '\\ \\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\\}' + \
        '\\quad\\text{(ecuaciones paramétricas)}'

def EcCartesianas(self):
    """Representación cartesiana del SubEspacio"""
    return '\\left\\{ \\boldsymbol{v}\\in\\mathbb{R}^{' + \
        latex(self.Rm) + \
        '\\ \\left| ' + \
        latex(self.cart) + \
        '\\boldsymbol{v}=\\boldsymbol{0}\\right. \\right\\}' + \
        '\\quad\\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 44.
Uses Matrix 13 and SubEspacio 42.

3.8 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 **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.

75a **<Iniciación de la clase BlockMatrix 75a>**≡

```
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 75b.
Uses BlockMatrix 75b and Sistema 7.

La clase **BlockMatrix** junto con el listado de sus métodos aparece en el siguiente recuadro:

75b **<Definición de la clase BlockMatrix 75b>**≡

```
class BlockMatrix:
    <Iniciación de la clase BlockMatrix 75a>
    <Repartición de las columnas de una BlockMatrix 77a>
    <Repartición de las filas de una BlockMatrix 77b>
    <Representación de la clase BlockMatrix 79>
```

This code is used in chunk 45.

Defines:

BlockMatrix, used in chunks 4, 5, 11, 17c, 20, 43, 51a, 58–60, 68b, 75–77, and 79.

3.8.1 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 $p, p+1, \dots, q$, (es decir, a la lista ordenada de los números de $\{k \in \mathbb{N} | p \leq k \leq q\}$).
- Si $i_1, \dots, i_r \in \mathbb{N}$ con $i_1 < \dots < i_r \leq m$ donde m es el número de filas de \mathbf{A} , entonces $\{i_1, \dots, i_r\} \mathbf{A}$ es la matriz de bloques

$$\{i_1, \dots, i_r\} \mathbf{A} = \begin{bmatrix} (1:i_1) \mathbf{A} \\ (i_1+1:i_2) \mathbf{A} \\ \vdots \\ (i_{r-1}+1:i_r) \mathbf{A} \end{bmatrix}$$

- Si $j_1, \dots, j_s \in \mathbb{N}$ con $j_1 < \dots < j_s \leq n$ donde n es el número de columnas de \mathbf{A} , entonces $\mathbf{A}_{\{j_1, \dots, j_s\}}$ es la matriz de bloques

$$\mathbf{A}_{\{j_1, \dots, j_s\}} = \left[\mathbf{A}_{|(1:j_1)} \mid \mathbf{A}_{|(j_1+1:j_2)} \mid \dots \mid \mathbf{A}_{|(j_{s-1}+1:j_s)} \right]$$

Comencemos por la partición de índices a partir de un conjunto y un número (correspondiente al último índice).

76a

```

<Definición del método particion 76a>≡
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 = 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 45.

y ahora el método de partición por filas y por columnas resulta inmediato:

76b

```

<Partición de una matriz por filas de bloques 76b>≡
elif isinstance(i,set):
    return BlockMatrix ([ [a|self] for a in particion(i,self.m) ])

```

This code is used in chunk 21.
Uses BlockMatrix 75b.

76c

```

<Partición de una matriz por columnas de bloques 76c>≡
elif isinstance(j,set):
    return BlockMatrix ([ [self|a for a in particion(j,self.n)] ])

```



```
for a in particion(i,self.sis.lista[0][0].m))
```

⟨Caso general de repartición por filas 78c⟩

This code is used in chunk 75b.
Uses `BlockMatrix` 75b.

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:

78a *⟨Definición del procedimiento de generación del conjunto clave para particionar 78a⟩*≡

```
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 45.

Así, los casos generales consisten en reparticionar de nuevo:

78b *⟨Caso general de repartición por columnas 78b⟩*≡

```
elif self.n > 1:
    return (key(self.lm) | Matrix(self)) | j
```

This code is used in chunk 77a.
Uses `Matrix` 13.

78c *⟨Caso general de repartición por filas 78c⟩*≡

```
elif self.m > 1:
    return i | (Matrix(self) | key(self.ln))
```

This code is used in chunk 77b.
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 $\{1,2,3\} | A$.

3.8.2 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 líneas 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.

```
79 <Representación de la clase BlockMatrix 79>≡
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 75b.
Uses `BlockMatrix` 75b.

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 3. 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:

80a `<Chunk de ejemplo que define la lista a 80a>≡`
`a = ["Matemáticas II es mi asignatura preferida", "Python mola", 1492, "Noweb"]`
This code is used in chunk 80c.

y este otro chunk:

80b `<Segundo chunk de ejemplo que cambia el último elemento de la lista a 80b>≡`
`a[-1] = 10`
This code is used in chunk 80c.

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:

80c `<EjemploLiterateProgramming.py 80c>≡`
`<Chunk de ejemplo que define la lista a 80a>`
`<Segundo chunk de ejemplo que cambia el último elemento de la lista a 80b>`

`for indice, item in enumerate(a, 1):`
 `print (indice, item)`

*<Chunk final que indica qué tipo de objeto es **a** y hace unas sumas 83b>*
 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 83](#) 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

<Caso general de repartición por columnas 78b> [77a](#), [78b](#)
<Caso general de repartición por filas 78c> [77b](#), [78c](#)
*<Chunk de ejemplo que define la lista **a** 80a>* [80a](#), [80c](#)
*<Chunk final que indica qué tipo de objeto es **a** y hace unas sumas 83b>* [80c](#), [83b](#)
<Composición de Transformaciones Elementales o aplicación sobre las filas de una Matrix 35> [35](#), [37b](#)
<Copyright y licencia GPL 83a> [83a](#)
*<Creación del atributo **sis** cuando no tenemos una lista de Vectores 68b>* [12](#), [68b](#)
*<Creación del atributo **t** cuando se instancia con otra **T** o lista de **Ts** 72>* [33](#), [72](#)
<Definición de la clase BlockMatrix 75b> [45](#), [75b](#)
<Definición de la clase Matrix 13> [13](#), [45](#)
*<Definición de la clase **T** (Transformación Elemental) 37b>* [37b](#), [45](#)
<Definición de la clase Vector 9b> [9b](#), [45](#)
<Definición de la igualdad entre Vectores 25b> [9b](#), [25b](#)
<Definición de la igualdad entre dos Matrix 30> [13](#), [30](#)
*<Definición de la matriz identidad: **I** 71b>* [45](#), [71b](#)
*<Definición de matriz nula: **MO** 71a>* [45](#), [71a](#)
*<Definición de vector nulo: **VO** 70c>* [45](#), [70c](#)
*<Definición del método **particion** 76a>* [45](#), [76a](#)
<Definición del método PasosYEscritura 51a> [50b](#), [51a](#), [52](#), [53](#), [54a](#), [54b](#), [55a](#), [55b](#), [56a](#), [56b](#), [57a](#), [57b](#), [58b](#), [59a](#), [59b](#), [60b](#)
<Definición del procedimiento de generación del conjunto clave para particionar 78a> [45](#), [78a](#)
<EjemploLiterateProgramming.py 80c> [80c](#)
<Escalonamiento de una matriz mediante eliminación por columnas 48> [45](#), [48](#), [49a](#), [50a](#)
<Escalonamiento de una matriz mediante eliminación por filas 49b> [45](#), [49b](#)
<Estructura de las variantes de aplicación del método de Gauss 50b> [50b](#)
<Inicialización de la clase BlockMatrix 75a> [75a](#), [75b](#)

<Inicialización de la clase **Matrix** 12> [12](#), [13](#)
 <Inicialización de la clase **Sistema** 5> [5](#), [7](#)
 <Inicialización de la clase **SubEspacio** 42> [42](#), [44](#)
 <Inicialización de la clase **T (Transformación Elemental)** 33> [33](#), [37b](#)
 <Inicialización de la clase **Vector** 9a> [9a](#), [9b](#)
 <Invirtiendo una matriz 58b> [45](#), [58b](#), [59a](#), [59b](#)
 <La clase **Sistema** 7> [7](#), [45](#)
 <La clase **SubEspacio** 44> [44](#), [45](#)
 <Método **pivote** calcula el índice del primer coef. no nulo de un **Vector** a partir de cierta posición 47> [45](#), [47](#)
 <Método auxiliar **CreaLista** que devuelve listas de abreviaturas 34b> [33](#), [34b](#), [35](#), [37a](#)
 <Método auxiliar que calcula la inversa de una Transformación elemental 37a> [36b](#), [37a](#)
 <Método auxiliar que calcula la inversa de una **Matrix** 58a> [29b](#), [58a](#)
 <Método **html** general 62a> [45](#), [62a](#)
 <Método **latex** general 62b> [45](#), [62b](#)
 <Métodos de la clase **Sistema** para que actue como si fuera una lista 6a> [6a](#), [6b](#), [7](#)
 <Métodos de la clase **SubEspacio** 43> [43](#), [44](#)
 <Métodos de representación de la clase **Sistema** 65a> [7](#), [65a](#)
 <Métodos de representación de la clase **SubEspacio** 74> [44](#), [74](#)
 <Métodos **html** y **latex** para fracciones 64> [45](#), [64](#)
 <normal 59c> [45](#), [59c](#)
 <Normalizando la diagonal principal para que sus componentes no nulos sean unos 57b> [45](#), [57b](#)
 <notacion.py 45> [45](#)
 <Operador selector por la derecha para la clase **Matrix** 18> [13](#), [18](#)
 <Operador selector por la derecha para la clase **Sistema** 15> [7](#), [15](#)
 <Operador selector por la derecha para la clase **Vector** 16b> [9b](#), [16b](#)
 <Operador selector por la izquierda para la clase **Matrix** 21> [13](#), [21](#)
 <Operador selector por la izquierda para la clase **Vector** 17b> [9b](#), [17b](#)
 <Operador transposición para la clase **Matrix** 19b> [13](#), [19b](#)
 <Operador transposición para la clase **T** 36a> [36a](#), [37b](#)
 <Opuesto de un **Vector** 68a> [9b](#), [68a](#)
 <Opuesto de una **Matrix** 70b> [13](#), [70b](#)
 <Partición de una matriz por columnas de bloques 76c> [18](#), [76c](#)
 <Partición de una matriz por filas de bloques 76b> [21](#), [76b](#)
 <Potencia de una **Matrix** 29b> [13](#), [29b](#)
 <Potencia de una **T** 36b> [36b](#), [37b](#)
 <Producto de un **Sistema** por un **Vector** o una **Matrix** a su derecha 66> [7](#), [66](#)
 <Producto de un **Vector** por un escalar a su izquierda 23b> [9b](#), [23b](#)
 <Producto de un **Vector** por un escalar, **Vector**, o **Matrix** a su derecha 25a> [9b](#), [25a](#)
 <Producto de una **Matrix** por un escalar a su izquierda 27b> [13](#), [27b](#)
 <Producto de una **Matrix** por un escalar, un vector o una matriz a su derecha 29a> [13](#), [29a](#)
 <Repartición de las columnas de una **BlockMatrix** 77a> [75b](#), [77a](#)
 <Repartición de las filas de una **BlockMatrix** 77b> [75b](#), [77b](#)
 <Representación de la clase **BlockMatrix** 79> [75b](#), [79](#)
 <Representación de la clase **Matrix** 69b> [13](#), [69b](#)
 <Representación de la clase **T** 73> [37b](#), [73](#)
 <Representación de la clase **Vector** 67a> [9b](#), [67a](#)
 <Resolviendo un sistema homogéneo 60b> [45](#), [60b](#)
 <Reverso de un **Vector** 67b> [9b](#), [67b](#)
 <Reverso de una **Matrix** 70a> [13](#), [70a](#)
 <Se almacenan los atributos **tex**, **pasos** y **rank**; y se devuelve el resultado 51b> [50b](#), [51b](#), [52](#), [53](#), [54a](#), [54b](#), [55a](#), [55b](#),
[56a](#), [56b](#), [57a](#)
 <Segundo chunk de ejemplo que cambia el último elemento de la lista **a** 80b> [80b](#), [80c](#)
 <sistema 60a> [45](#), [60a](#)
 <Suma de **Matrix** 26b> [13](#), [26b](#)
 <Suma de **Vectores** 22b> [9b](#), [22b](#)
 <Texto de ayuda de la clase **Matrix** 11> [11](#), [13](#)
 <Texto de ayuda de la clase **Sistema** 4> [4](#), [7](#)

<Texto de ayuda de la clase **T** (Transformación Elemental) 32> [32](#), [37b](#)
 <Texto de ayuda de la clase **Vector** 8> [8](#), [9b](#)
 <Texto de ayuda de las transformaciones elementales de las columnas de una **Matrix** 39a> [39a](#), [39b](#)
 <Texto de ayuda de las transformaciones elementales de las filas de una **Matrix** 39c> [39c](#), [40](#)
 <Texto de ayuda de las transformaciones elementales de un **Sistema** 38a> [38a](#), [38b](#)
 <Texto de ayuda para el operador producto por la derecha en la clase **Matrix** 28> [28](#), [29a](#)
 <Texto de ayuda para el operador producto por la derecha en la clase **Sistema** 65b> [65b](#), [66](#)
 <Texto de ayuda para el operador producto por la derecha en la clase **Vector** 24> [24](#), [25a](#)
 <Texto de ayuda para el operador producto por la izquierda en la clase **Matrix** 27a> [27a](#), [27b](#)
 <Texto de ayuda para el operador producto por la izquierda en la clase **Vector** 23a> [23a](#), [23b](#)
 <Texto de ayuda para el operador selector por la derecha para la clase **Matrix** 17c> [17c](#), [18](#)
 <Texto de ayuda para el operador selector por la derecha para la clase **Sistema** 14> [14](#), [15](#)
 <Texto de ayuda para el operador selector por la derecha para la clase **Vector** 16a> [16a](#), [16b](#)
 <Texto de ayuda para el operador selector por la izquierda para la clase **Matrix** 20> [20](#), [21](#)
 <Texto de ayuda para el operador selector por la izquierda para la clase **Vector** 17a> [17a](#), [17b](#)
 <Texto de ayuda para el operador suma en la clase **Matrix** 26a> [26a](#), [26b](#)
 <Texto de ayuda para el operador suma en la clase **Vector** 22a> [22a](#), [22b](#)
 <Texto de ayuda para el operador transposición de la clase **Matrix** 19a> [19a](#), [19b](#)
 <Texto de ayuda para la composición de Transformaciones Elementales **T** 34c> [34c](#), [35](#)
 <Transformaciones elementales de las columnas de una **Matrix** 39b> [13](#), [39b](#)
 <Transformaciones elementales de las filas de una **Matrix** 40> [13](#), [40](#)
 <Transformaciones elementales de los elementos de un **Sistema** 38b> [7](#), [38b](#)
 <Variantes del escalonamiento por eliminación por columnas que guardan los pasos dados 52> [45](#), [52](#), [53](#), [54a](#), [54b](#),
[55a](#), [55b](#)
 <Variantes del escalonamiento por eliminación por filas que guardan los pasos dados 56a> [45](#), [56a](#), [56b](#), [57a](#)
 <Verificación de que las abreviaturas corresponden a transformaciones elementales 34a> [33](#), [34a](#)
 <Verificación de que todas las columnas de la matriz tienen la misma longitud 69a> [12](#), [69a](#)
 <Verificación de que todas las filas de la matriz tendrán la misma longitud 68c> [68b](#), [68c](#)

Licencia

83a

```

<Copyright y licencia GPL 83a>≡
# 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.

83b

```

<Chunk final que indica qué tipo de objeto es a y hace unas sumas 83b>≡
type(a)
2+2

```

3+20

This code is used in chunk 80c.