

Notación Asociativa para un Curso de Álgebra Lineal (NAcAL)

(Versión: 0.1.29)

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

Marcos Bujosa

December 2, 2022

Índice

Declaración de intenciones	3
1 Código principal del módulo NAcAL	4
1.1 La clase Sistema	4
1.1.1 Ejemplos de uso	7
1.1.2 La subclase Vector	10
1.1.3 La subclase clase Matrix	13
1.2 Operadores selectores	17
1.2.1 Operador selector por la derecha para la clase Sistema	17
1.2.2 Operador selector por la izquierda para la clase Vector	19
1.2.3 Operador selector por la derecha para la clase Matrix	19
1.2.4 Operador transposición de una Matrix	21
1.2.5 Operador selector por la izquierda para la clase Matrix	22
1.2.6 Ejemplos de uso	23
1.3 Operaciones con Sistemas	24
1.3.1 Suma de Sistemas	24
1.3.2 Producto de un Sistema por un escalar a su izquierda	25
1.3.3 Producto de un Sistema por un escalar, un Vector o una Matrix a su derecha	27
1.4 La clase transformación elemental T	30
1.4.1 Implementación	32
1.4.2 Transposición de transformaciones elementales	34
1.4.3 Potencias e inversa de transformaciones elementales	35
1.4.4 Transformaciones elementales “espejo”	36
1.4.5 Sustitución de variables simbólicas	36
1.4.6 Igualdad entre transformaciones elementales	37
1.4.7 T es iterable	37
1.5 Transformaciones elementales de un Sistema	39
1.5.1 Transformaciones elementales de las filas de una Matrix	40
1.5.2 Transformaciones elementales por la izquierda de un Vector	41
1.6 Librería completa	42
2 Algoritmos del curso	44
2.1 Operaciones empleadas las distintas variantes de eliminación	45
2.1.1 La operación de eliminación de componentes	45
2.1.2 La operación de intercambio de columnas	47
2.1.3 La operación de normalización de los pivotes	47
2.1.4 Se anotan las transformaciones de cada operación y se aplican a las columnas.	47
2.2 Eliminación “de izquierda a derecha”, Gaussiana y Gauss-Jordan	48
2.2.1 Primero evitando las fracciones... en la medida de lo posible	48
2.2.2 Si no evitamos las fracciones realizamos menos operaciones	52
2.2.3 Eliminación por filas	54
2.3 Inversión de una matriz por eliminación Gaussiana	59
2.4 Resolución de un sistema de ecuaciones homogéneo	61
2.5 Resolución de un sistema de ecuaciones	63
2.6 Cálculo del determinante por eliminación Gaussiana	65
2.7 Diagonalizando en bloques triangulares una matriz cuadrada por semejanza (Dentado)	67
2.7.1 Diagonalización ortogonal de una matriz simétrica	70

2.8	Diagonalización por congruencia	71
3	Las clases SubEspacio y EAfin	75
3.1	La clase SubEspacio (de \mathbb{R}^m)	75
3.2	La clase EAfin (de \mathbb{R}^m)	80
4	Otros trozos de código	84
4.1	Métodos de representación para el entorno Jupyter	84
4.2	Completando la clase Sistema	85
4.2.1	Representación de la clase Sistema	85
4.2.2	Otros métodos de la clase Sistema	87
4.3	Completando la clase Vector	89
4.3.1	Representación de la clase Vector	89
4.3.2	Otros métodos para la clase Vector	89
4.4	Completando la clase Matrix	90
4.4.1	Otras formas de instanciar una Matrix	90
4.4.2	Códigos que verifican que los argumentos son correctos	91
4.4.3	Representación de la clase Matrix	91
4.4.4	Otros métodos para la clase Matrix	92
4.4.5	Otros métodos de la clase Matrix que usan la eliminación	96
4.4.6	Otros métodos de la clase Matrix específicos de las matrices cuadradas	97
4.5	Vectores y Matrices especiales	101
4.6	Completando la clase T	103
4.6.1	Otras formas de instanciar una T	103
4.6.2	Representación de la clase T	103
4.7	Representación de los procesos de eliminación Gaussiana	105
4.8	Representación de la resolución de sistemas de ecuaciones	109
4.9	Completando la clase EAfin	110
4.9.1	Representación de la clase EAfin	110
4.10	Completando la clase SubEspacio	110
4.10.1	Representación de la clase SubEspacio	110
4.11	La clase BlockM. Matrices particionadas	111
4.11.1	La clase SisMat. Sistema de Matrices	111
4.11.2	La clase BlockM. Matrices particionadas (o matrices por bloques)	114
4.11.3	Particionado de matrices	116
4.11.4	Representación de la clase BlockM	120
5	Sobre este documento	121
5.1	Secciones de código	122

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).

Así, este ejercicio 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 no hemos escrito las ordenes de manera correcta (lo que supone que también hemos escrito incorrectamente las expresiones matemáticas).

Con esta idea en mente:

1. La notación del **Curso de Álgebra Lineal** pretende ser operativa; es decir, su uso debe ser directamente traducible a operaciones a realizar por un ordenador. Para lograr una mayor simplificación, la notación explota de manera intensiva la asociatividad.
2. Muchas de las demostraciones del **libro** son algorítmicas. En particular aquellas las relacionadas con la eliminación. De esta manera las demostraciones describen literalmente la programación de los correspondientes algoritmos.

Un módulo de Python específico para la asignatura

Aunque Python dispone de módulos para operar con vectores y matrices, *he decidido escribir un módulo propio*. De este modo logro que la notación del **libro** y las expresiones empleadas en el módulo se parezcan lo más posible.

ESTE DOCUMENTO DESCRIBE TANTO EL USO DEL MÓDULO NAcAL COMO SU CÓDIGO; PERO TENGA EN CUENTA QUE ESTO NO ES UN TUTORIAL DE PYTHON.

No obstante, he escrito unos Notebooks de Jupyter que ofrecen unas breves nociones de programación en Python (aunque 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 subrayar el paralelismo entre las definiciones dadas en el **libro** y los objetos (o **Clases**) definidos en **NAcAL**, las partes auxiliares del código se relegan al final¹ (véase la sección *Literate programming* en la Página 121). Destacar las partes centrales del código permitirá apreciar cómo las definiciones del **libro** son implementadas de manera literal en la librería de Python.

Tutorial previo en un Jupyter notebook

Antes de seguir, repase el Notebook **“Listas y tuplas”** en la carpeta **“TutorialPython”** en <https://github.com/mbujosab/nacallib/tree/master/doc/Notebooks/TutorialPython>

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

Marcos Bujosa

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. La clase Sistema. Las subclases Vector y Matrix, y la clase T

Tutorial previo en un Jupyter notebook

Antes de seguir, mírese el Notebook referente a “Clases” en la carpeta “TutorialPython” en <https://github.com/mbujosab/nacallib/tree/master/doc/Notebooks/TutorialPython>

Con lo visto en el Notebook anterior, definimos una *clase* para los *sistemas* (listas ordenadas) con el nombre de **Sistema**. También definiremos una subclase para los *vectores* y otra para las *matrices*. Además definiremos una clase para las *transformaciones elementales* y otra para las *matrices por bloques* (o matrices particionadas) ¹. Cada vez que definamos una nueva clase, especificaremos su modo de representación para que los Notebooks de Jupyter muestren representaciones semejantes a las empleadas en las notas de la asignatura.

1.1 La clase Sistema

En las notas de la asignatura se dice que

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

Aunque Python ya posee “listas”, vamos a crear nuestra propia clase denominada **Sistema**. En las notas de la asignatura los sistemas genéricos se muestran entre *corchetes* y con los elementos de la lista *seguidos de “;”*. ² Así,

```
Sistema( [ Vector([1,2,3]), I(2), 1492, T({1,2}) ] )
```

será representado en los Notebooks de Jupyter como: $\left[\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}; \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; 1492; \begin{matrix} \tau \\ [1 \rightleftharpoons 2] \end{matrix}; \right]$

La clase **Sistema** solo posee un atributo llamado **lista**, que es una **list** de Python con la lista de objetos que componen el **Sistema**. Además, *los elementos del Sistema serán mostrados con su propia representación especial en los notebooks de Jupyter* (como en el ejemplo anterior donde vemos un vector de \mathbb{R}^3 , la matriz identidad 2 por 2, un número y una transformación intercambio entre los vectores 1 y 2).

Los **Sistemas** y las **listas** de Python se diferencian en dos cosas: la representación de los objetos contenidos en las listas, y en el modo de concatenar las listas: las **listas** de Python se concatenan con “+” y los **Sistemas** con el método **concatena()**. Así reservamos el símbolo “+” para sumar **Sistemas** tal como se hace en Álgebra Lineal. Con ello buscamos que lo que veamos y escribamos en un Notebook de Jupyter sea lo más parecido posible a lo que vemos y escribimos en la asignatura de Álgebra Lineal.

El texto de ayuda de la clase **Sistema** es auto-explicativo y será lo que Python nos muestre cuando tecleemos **help(Sistema)**:

¹Más adelante definiremos nuevas clases para los subespacios vectoriales, los espacios afines, etc.

²Aunque los vectores y las matrices también son sistemas, emplean representaciones particulares.

5a `<Texto de ayuda de la clase Sistema 5a>≡`

```

"""Clase Sistema

Un Sistema es una lista ordenada de objetos. Los Sistemas se instancian
con una lista, tupla u otro Sistema.

Parámetros:
    data (list, tuple, Sistema): lista, tupla o Sistema de objetos.

Atributos:
    lista (list): lista de objetos.

Ejemplos:
>>> # Crea un nuevo Sistema a partir de una lista, tupla o Sistema

>>> Sistema( [ 10, 'hola', T({1,2}) ] )      # con lista
>>> Sistema( ( 10, 'hola', T({1,2}) ) )      # con tupla
>>> Sistema( Sistema( [ 10, 'hola', T({1,2}) ] ) ) # con Sistema

[10; 'hola'; T({1, 2})]
"""

```

This code is used in chunk 9.
Uses Sistema 9 and T 38.

Implementación de los sistemas (o listas ordenadas) en la clase Sistema

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

- `data` es el único argumento (o parámetro) de la clase `Sistema`. Puede ser una lista, tupla o `Sistema`.
- Añadimos un breve texto de ayuda sobre el método `__init__` que Python mostrará con: `help Sistema.__init__`.
- Cuando `data` es una lista, tupla o `Sistema`, el atributo `self.lista` guarda una copia de la lista, o la tupla convertida en lista, o una copia del atributo `lista` del `Sistema` dado.
- Cuando `data` no es una lista, tupla, o `Sistema` se devuelve un mensaje de error.

5b `<Inicialización de la clase Sistema 5b>≡`

```

def __init__(self, data):
    """Inicializa un Sistema con una lista, tupla o Sistema"""
    if isinstance(data, (list, tuple, Sistema)):

        self.lista = list(data)

    else:
        raise ValueError(' El argumento debe ser una lista, tupla, o Sistema.')
```

This code is used in chunk 9.
Uses Sistema 9.

Un `Sistema` será como una `list` de Python (salvo por la representación y el modo de concatenar).

Para que un `Sistema` sea iterable necesitamos los procedimientos “mágicos” `__getitem__`, que permite seleccionar componentes del sistema, y `__setitem__`, que permite modificar componentes del `Sistema`³.

6a *<Métodos de la clase Sistema para que actúe como si fuera una list de Python 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 9.
Uses Sistema 9.

Con `len` contamos el número de elementos del `Sistema`. Con `copy` podemos hacer una copia, por ejemplo `Z=Y.copy()` hace una copia del `Sistema Y` (aunque lograremos el mismo resultado con `Z=Sistema(Y)`). Por otra parte, comprobaremos si dos `Sistemas` son iguales con `“==”`, y si son distintos con `“!=”`. Por último, con el método `reversed` obtenemos el `Sistema` cuyos elementos aparecen en el orden inverso.

6b *<Métodos de la clase Sistema para que actúe como si fuera una list de Python 6a>+≡*

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

def copy(self):
    """ Copia la lista de otro Sistema"""
    return type(self)(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 type(self)(list(reversed(self.lista)))
```

This code is used in chunk 9.
Uses Sistema 9.

Por último (y a diferencia de las `list` de Python) concatenamos dos `Sistemas` con el método `concatena()`. Así, `A.concatena(B)` añade al final de la `lista` del sistema A los elementos de la `lista` del sistema B.

³Recuerde que los índices de las listas comienzan en 0! Mantendremos este “pythonesco” modo de indexar los `Sistemas`, aunque luego añadiremos un modo de seleccionar componentes similar al empleado en las notas de la asignatura empleando el operador selector `“|”`.

```

7  <Método de la clase Sistema para concatenar dos Sistemas 7>≡
    def concatena(self, other, c=0):
        """ Concatena dos Sistemas """
        if not isinstance(other, Sistema):
            raise ValueError('Un Sistema solo se puede concatenar a otro Sistema')
        S = type(self)(self.lista + other.lista)
        if isinstance(other, Matrix) and c:
            S.cF, S.cC = self.cF, self.cC
            S.cC.update({self.n})
        return S

```

This code is used in chunk 9.

Defines:

`concatena`, used in chunks 60, 63b, 70, 71b, 78c, 80b, 83a, 88, and 95.

Uses `Matrix` 15b and `Sistema` 9.

...de este modo reservamos el símbolo “+” para las sumas elemento a elemento entre dos sistemas (por ejemplo para sumar dos `Vectores`).

1.1.1 Ejemplos de uso

Un sistema es esencialmente una lista. Sus elementos pueden ser cualquier cosa. Por ejemplo, vamos a crear dos sistemas cuya primera componente es un `Vector`, la segunda un número, la tercera un `Sistema` y la cuarta una cadena de caracteres:

```

>>> A = Sistema([Vector([1, 2]), 5, Sistema([10,20,30]), 'Sis' ]);
>>> B = Sistema([Vector([1,-2]), 1, Sistema([ 1,20, 3]), 'tema']);

```

Una diferencia entre los `Sistemas` y las listas de Python es que Jupyter muestra la representación $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ de los elementos de un `Sistema` (si es que dicha representación está definida). Así, en Jupyter el `Sistema` A se vería así

$$\left[\begin{pmatrix} 1 \\ 2 \end{pmatrix}; 5; [10; 20; 30]; \text{Sis}; \right]$$

y el B así

$$\left[\begin{pmatrix} 1 \\ -2 \end{pmatrix}; 1; [1; 20; 3]; \text{tema}; \right]$$

Podemos preguntar por la longitud del `Sistema` (su número de componentes)

```

>>> len(A)

```

4

Podemos generar una copia

```

>>> C=A.copy()
>>> C
Sistema([Vector([1, 2]); 5; Sistema([10; 20; 30]); 'Sis'];])

```

Otra forma de generar una copia del sistema A es así

```

>>> C=Sistema(A)

```

Podemos preguntar si dos `Sistemas` son iguales

```

>>> A == B
False

```


o si son distintos

```
>>> A != B
True
```

también podemos invertir el orden de sus elementos

```
>>> D=reversed(A)
```

$$\left[\text{Sis}; [10; 20; 30;]; 5; \begin{pmatrix} 1 \\ 2 \end{pmatrix}; \right]$$

La otra diferencia entre los **Sistemas** y las **list** es el modo de concatenar. Las listas se concatenan con el procedimiento “+”, pero dicho procedimiento lo hemos reservado en los **Sistemas** para sumar, así que concatenamos dos sistemas del siguiente modo:

```
>>> E=A.concatena(B)
```

$$\left[\begin{pmatrix} 1 \\ 2 \end{pmatrix}; 5; [10; 20; 30;]; \text{Sis}; \begin{pmatrix} 1 \\ -2 \end{pmatrix}; 1; [1; 20; 3;]; \text{tema}; \right]$$

que crea un sistema con los elementos de A seguidos de los elementos de B.

Podemos acceder a los elementos de un **Sistema** del mismo modo a los elementos de una lista

```
>>> A[0]
Vector([1, 2])
```

```
>>> B[-1]
'tema'
```

O incluso cambiar los elementos

```
>>> A[0] = Matrix([ [1, 2], [3, 4] ])
```

de manera que ahora el **Sistema** A es

$$\left[\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}; 5; [10; 20; 30;]; \text{Sis}; \right].$$

Sin embargo su copia (el **Sistema** C) no ha cambiado:

$$\left[\begin{pmatrix} 1 \\ 2 \end{pmatrix}; 5; [10; 20; 30;]; \text{Sis}; \right].$$

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

```

9  <Definición de la clase Sistema 9>≡
    class Sistema:
        <Texto de ayuda de la clase Sistema 5a>
        <Inicialización de la clase Sistema 5b>
        <Métodos de la clase Sistema para que actúe como si fuera una list de Python 6a>
        <Método de la clase Sistema para concatenar dos Sistemas 7>
        <Método para recuperar el Sistema de cualquier subclase de Sistema 87a>
        <Operador selector por la derecha para la clase Sistema 18>
        <Suma y resta de Sistemas 25b>
        <Opuesto de un Sistema 27>
        <Producto de un Sistema por un escalar a su izquierda 26b>
        <Producto de un Sistema por un escalar, un Vector o una Matrix a su derecha 29>
        <Transformaciones elementales de los elementos de un Sistema 40a>
        <Métodos de representación de la clase Sistema 86a>
        <Comprobación de que todos los elementos de un Sistema son del mismo tipo 87c>
        <Comprobación de que un Sistema es nulo 87b>
        <Junta una lista de Sistemas en un único Sistema 88>
        <Sustitución de un símbolo por otro símbolo o valor en un Sistema 87d>
This code is used in chunk 42.
Defines:
    Sistema, used in chunks 5–7, 10–13, 15, 17, 24–28, 39, 61, 63, 66b, 76–80, 83, 85–87, 90b, 109, 112a, and 114–16.

```

Resumen

Los `Sistemas` almacenan una lista de objetos en su atributo `lista`. Los sistemas se comportan como las `list` de Python salvo por su representación y el modo de concatenar.

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

1.1.2 La subclase 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:

$$\mathbf{v} = (v_1, \quad v_2, \quad v_3),$$

o bien en forma de columna:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}.$$

Así pues, vamos a definir una nueva clase de objeto en Python: la clase *Vector* será una subclase de *Sistema*. De esta manera la subclase *Vector* hereda todas las propiedades de la clase madre *Sistema*. La *Representación de la clase Vector 89a* se redefine más adelante, así los *Vectores* no serán representados como *Sistemas* genéricos, sino a la manera de los vectores.

El texto de ayuda de la clase *Vector* es auto-explicativo; y Python nos lo mostrará cuando tecleemos `help(Vector)`:

```
10 <Texto de ayuda de la clase Vector 10>≡
    """Clase Vector(Sistema)

    Vector es un Sistema de números u objetos del módulo Sympy. Se puede
    instanciar con una lista, tupla o Sistema. 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) : Lista, tupla o Sistema de
        objetos de tipo int, float o sympy.Basic, o bien otro Vector.
        rpr (str) : Representación en Jupyter ('columna' por defecto).
        Si rpr='fila' el vector se representa en forma de fila.

    Atributos:
        n      (int)      : número de elementos de la lista.
        rpr    (str)      : modo de representación en Jupyter.

    Atributos heredados de la clase Sistema:
        lista (list)      : list con los elementos.

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

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

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

Implementación de los vectores en la clase Vector

Método de inicialización: `def __init__(self, data, rpr='columna').`

- La clase `Vector` emplea dos argumentos. El primero (`data`) es una lista, tupla o `Sistema` de objetos tipo `int`, `float` o `sympy.Basic`. El segundo argumento (`rpr`) es opcional e indica 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'`). Cuando `data` es un `Vector` (es decir, un `Sistema` de números) se obtiene una copia.
- Añadimos un breve texto de ayuda sobre el método `__init__` que Python mostrará con: `help Vector.__init__`
- Con `super().__init__(data)` la subclase `Vector` hereda los métodos y atributos de la clase madre `Sistema` (por tanto, `Vector` tendrá un atributo `lista`, así como todos los métodos definidos para la clase `Sistema`).
- Se verifica que los elementos del atributo `lista` son de tipo `int`, `float` o `sympy.Basic`.
- Se definen dos atributos para la subclase `Vector`: los atributos `rpr` y `n`.
 - `self.rpr` indica si el vector ha de ser representado como fila o como columna en el entorno `Jupyter`.
 - `self.n` es el número de elementos de la lista del `Sistema`.

¡Ya tenemos traducido al lenguaje Python la definición de vector de \mathbb{R}^n !

11a *<Inicialización de la clase Vector 11a>≡*

```
def __init__(self, data, rpr='columna'):
    """Inicializa Vector con una lista, tupla o Sistema"""

    super().__init__(data)

    <Verificación de que todos los elementos de la lista son números o de tipo sympy.Basic 11b>

    self.rpr = rpr
    self.n   = len(self)
```

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

Los vectores de \mathbb{R}^n son sistemas de n números reales. Con el siguiente código verificamos que los componentes del `Vector` son enteros o números de coma flotante; pero también cualquier objeto de la librería `Sympy` (`sympy.Basic`). Así es posible incluir números racionales `sympy.Rational`, números irracionales como $\sqrt{2}$, o incluso variables simbólicas o polinomios. Al ampliar el tipo de objetos que pueden aparecer en la lista de los `Vectores`, ya no nos limitamos a implementar vectores de \mathbb{R}^n ; a cambio podremos extender las aplicaciones del módulo.

11b *<Verificación de que todos los elementos de la lista son números o de tipo sympy.Basic 11b>≡*

```
if not all( [isinstance(e, (int, float, sympy.Basic)) for e in self] ):
    raise ValueError('no todos los elementos son números o parámetros!')
```

This code is used in chunk 11a.

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

```
12 <Definición de la clase Vector 12>≡
    class Vector(Sistema):
        <Texto de ayuda de la clase Vector 10>
        <Inicialización de la clase Vector 11a>
        <Operador selector por la izquierda para la clase Vector 19b>
        <Transformaciones elementales por la izquierda de un Vector 41b>
        <Creación de una Matrix diagonal a partir de un Vector 89b>
        <Normalización de un Vector 90a>
        <Representación de la clase Vector 89a>

This code is used in chunk 42.
Defines:
    Vector, used in chunks 10, 11a, 13, 15a, 17, 19c, 21–25, 28, 29, 34, 71b, 79–81, 89–91, 93b, and 101a.
Uses Sistema 9.
```

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

Resumen

Los **vectores** son una subclase de la clase “padre” `Sistema`. Como `Sistemas` almacenan una lista de números en su atributo `lista`. Además, heredan los métodos definidos en la clase `Sistema`. Aquellos métodos de la clase `Sistema` que no son adecuados para la subclase `Vector` han de ser redefinidos dentro de la subclase (por ejemplo, el método de `<Representación de la clase Vector 89a>`).

La subclase `Vector` es un `Sistema` con algunos atributos propios: `self.rpr` indica el modo de representación en el entorno Jupyter; y `self.n` muestra el número de elementos de la `lista`.

1. Cuando se instancia un `Vector` con otro `Vector`, se obtiene una copia del `Vector`.
2. Asociados a los `Vectores` hay una serie de métodos que se describirán más adelante.

1.1.3 La subclase 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 .

Al representar las matrices, mostramos la lista de vectores entre corchetes y sin signos de puntuación:

$$\mathbf{A} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \dots \mathbf{v}_n], \quad \text{donde las } n \text{ columnas } \mathbf{v}_i \text{ son vectores de } \mathbb{R}^m.$$

Vamos a crear una nueva subclase de `Sistema` que denominaremos `Matrix`. El atributo `lista` de `Matrix` será la lista de `Vectores` (todos con el mismo número de componentes) que constituyen las “columnas” de la matriz.

El texto de ayuda de la clase `Matrix` es auto-explicativo; y Python lo mostrará si se teclea `help(Matrix)`.

```
13 <Texto de ayuda de la clase Matrix 13>≡
    """Clase Matrix

    Es un Sistema de Vectores con el mismo número de componentes. Una Matrix
    se puede construir con: 1) una lista, tupla o Sistema de Vectores con el
    mismo número de componentes (serán las columnas); 2) una lista, tupla o
    Sistema de listas, tuplas o Sistemas con el mismo número de componentes
    (serán las filas de la matriz); 3) una Matrix (se obtendrá una copia);
    4) una BlockM (se obtendrá la Matrix que resulta de unir los bloques).

    Parámetros:
        data (list, tuple, Sistema, Matrix, BlockM): Lista, tupla o Sistema
        de Vectores (columnas con mismo núm. de componentes); o de listas,
        tuplas o Sistemas (filas de misma longitud); o una Matrix o BlockM.

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

    Ejemplos:
    >>> # Crear 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]) ])
    >>> # Crear una Matrix a partir de una lista de listas de números
    >>> A = Matrix( [ [1,1,9], [2,0,2] ] )

    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 BlockM
    >>> Matrix( {1}|A|{2} )

    Matrix([ Vector([1, 2]); Vector([1, 0]); Vector([9, 2]) ]) """
This code is used in chunk 15b.
Uses BlockM 116c, Matrix 15b, Sistema 9, and Vector 12.
```

Implementación de las matrices en la clase Matrix

Método de inicialización: `def __init__(self, sis).`

- **Matrix** se instancia con el argumento **data**; que puede ser *una lista, tupla o Sistema*:
 - de **Vectores** con el mismo número de componentes (serán las columnas de la matriz)
 - o de listas, tuplas o **Sistemas** de la misma longitud (serán las filas). Los elementos han de ser de tipo `int`, `float` o `sympy.Basic` (como en el caso de los **Vectores**).
- o bien **data** puede ser otra **Matrix**; o una **BlockM**.
- Añadimos un breve texto de ayuda del método `__init__`
- Con `super().__init__(self.sis)` se heredan los atributos y métodos de la clase **Sistema** (en particular el atributo `self.lista` en el que guardaremos la lista de **Vectores**).
- La variable `lista` es igual al atributo `lista` del **Sistema** generado con **data**.
- Cuando el primer elemento de `lista` es un **Vector**, se comprueba si también el resto de elementos son **Vectores** con el mismo número de componentes. Si es así, en el atributo `self.lista` se guarda la lista de **Vectores** (que serán las columnas de la matriz). Así, cuando **data** es una **Matrix** se obtiene una copia.
- Cuando el primer elemento de `lista` NO es un **Vector**, la correspondiente `self.lista` de **Vectores** (columnas) se genera de manera diferente según qué tipo de objeto es **data**:
 - cuando los elementos de **data** son listas, tuplas o **Sistemas** de la misma longitud: se entiende que **data** es la “lista de filas” de la matriz. Entonces se reconstruye la `self.lista` de columnas correspondiente.
 - cuando **data** es una **BlockM**: se obtiene la **Matrix** resultante de eliminar la partición.

El trozo de código *⟨Creación del atributo lista cuando no tenemos una lista de Vectores 90b⟩* muestra cómo.

- Por conveniencia definimos dos atributos adicionales: `self.m` muestra el número de filas de la matriz (longitud del primer elemento del sistema); y `self.n` muestra el número de columnas (longitud del propio sistema).
- Así como los atributos `self.cF` y `self.cC` que se usan en la representación en Jupyter. Almacenan los índices de las filas y columnas detrás de las que se deben pintar las rectas horizontales y verticales. Por defecto ninguna, por lo que inicialmente los atributos son un conjunto que solo contiene el cero,

...; *Ya tenemos traducido al lenguaje Python la definición de matriz!*

15a *<Iniciación de la clase Matrix 15a>≡*

```
def __init__(self, data):
    """Inicializa una Matrix"""
    super().__init__(data)

    lista = Sistema(data).lista

    if isinstance(lista[0], Vector):
        <Verificación de que todas las columnas de la matriz tienen la misma longitud 91b>
        self.lista = lista.copy()

    <Creación del atributo lista cuando no tenemos una lista de Vectores 90b>

    self.m = len(self|1)
    self.n = len(self)
    self.cC = {0}
    self.cF = {0}
```

This code is used in chunk 15b.
Uses Matrix 15b, Sistema 9, and Vector 12.

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

15b *<Definición de la clase Matrix 15b>≡*

```
class Matrix(Sistema):
    <Texto de ayuda de la clase Matrix 13>
    <Iniciación de la clase Matrix 15a>
    <Operador selector por la derecha para la clase Matrix 20>
    <Operador transposición para la clase Matrix 22a>
    <Operador selector por la izquierda para la clase Matrix 23>
    <Transformaciones elementales de las filas de una Matrix 41a>
    <Métodos útiles para la clase Matrix 92b>
    <Comprobación de que una Matrix es singular 94b>
    <Métodos de Matrix que usan la eliminación 96>
    <Métodos de Matrix para obtener los subespacios asociados 97a>
    <Potencia de una Matrix 97b>
    <Determinante mediante la expansión de Laplace 98a>
    <Método Gram-Schmidt para ortogonalizar un sistema de Vectores 100a>
    <Representación de la clase Matrix 92a>
```

This code is used in chunk 42.

Defines:

`Matrix`, used in chunks 7, 13, 15a, 19c, 21–25, 28, 29, 31, 33c, 34, 40b, 49–53, 55–57, 59–61, 63, 65b, 68–74, 76, 87d, 89b, 90b, 92–102, 106, 107, 110, 112, 113b, 116, 119b, and 120.

Uses Sistema 9.

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

Resumen

Las **matrices** son una subclase de la clase “padre” **Sistema**. Como **Sistemas** almacenan una lista de **Vectores** en su atributo **lista**, además de heredar los métodos definidos en la clase **Sistema**. Aquellos métodos definidos para la clase **Sistema** que no son adecuados para la subclase **Matrix** han de ser redefinidos (por ejemplo, el método de *Representación de la clase Matrix 92a*)).

La subclase **Matrix** es un **Sistema** con dos atributos propios: **self.m** es el número de filas de **Matrix**; y **self.n** el número de columnas.

1. Cuando se instancia con una lista, tupla o **Sistema** de **Vectores**, el atributo **self.lista** almacena dicha lista de **Vectores**. *Ésta es la forma de crear una matriz a partir de sus columnas*. Consecuentemente, si se instancia una **Matrix** con otra **Matrix** se obtiene una copia.
2. Por comodidad, también es posible instanciar con una lista, tupla o **Sistema** de listas, tuplas o **Sistemas** que, en este caso, se interpreta como la lista de filas de la matriz. Internamente se dan los pasos necesarios para almacenar la matriz en forma de **Sistema** de columnas. (Esta forma de instanciar una **Matrix** se usará para programar la **transposición** en la *Página 21*).
3. Cuando se instancia con una **BlockM** se obtiene la **Matrix** resultante de unificar los bloques en una sola matriz.
4. Asociados a las **Matrix** hay una serie de métodos específicos que veremos más adelante.

¡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 más adelante nos permitirá definir las operaciones de suma, producto, etc...

1.2 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})

Pues bien, queremos manejar una notación similar en Python; así que debemos definir un operador selector. Nos conviene hacerlo con un método de Python asociado un símbolo.

Tutorial previo en un Jupyter notebook

Si no recuerda a qué me refiero con “símbolos asociados a métodos”, repase la sección “Métodos especiales con símbolos asociados” del Notebook “Clases” en la carpeta “TutorialPython” en <https://github.com/mbujosab/nacallib/tree/master/doc/Notebooks/TutorialPython>

Como los métodos `--or--` y `--ror--` tienen asociados la barra vertical a derecha e izquierda, usaremos el convenio:

Mates II	Python	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>

Recuerde que en Python los índices de objetos iterables comienzan en cero, pero en la notación empleada en Matemáticas 2 el índice del primer elemento es 1.

1.2.1 Operador selector por la derecha para la clase Sistema.

Tal como se hace en el Tema 2 de las notas de la asignatura, seleccionaremos elementos de un `Sistema` con el operador “|” actuando por la derecha (¡ojo, para la clase genérica `Sistema` solo lo hacemos por la derecha!).

```
17 <Texto de ayuda para el operador selector por la derecha para la clase Sistema 17>≡
    """
    Extrae el j-ésimo componente del Sistema; o crea un Sistema con los
    elementos indicados (los índices comienzan por el número 1)

    Parámetros:
        j (int, list, tuple, slice): Índice (o lista de índices) del
```

```

        elementos (o elementos) a seleccionar

Resultado:
    ?: Si j es int, devuelve el elemento j-ésimo del Sistema.
    Sistema: Si j es list, tuple o slice devuelve el Sistema formado por
        los elementos indicados en la lista, tupla o slice 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])]
>>> # Sistema formado por los elementos indicados en el slice
>>> Sistema([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | slice(1,3,2)

[Vector([1, 0]), Vector([3, 0])] """
This code is used in chunk 18.
Uses Sistema 9 and Vector 12.

```

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

Cuando el argumento `j` es un número entero (`int`), seleccionamos el elemento jésimo del atributo `lista` (como en Python los índices de objetos iterables comienzan en cero, para seleccionar el elemento jésimo de `lista`, escribimos `lista[j-1]`; así `a|1` selecciona el primer elemento del atributo `lista`, es decir `a.lista[0]`).

Luego usamos el método (`self|a`) (siendo `a` un `int`) para definir el operador cuando `j` es una lista o tupla (`list`, `tuple`) de índices y generar así un sistema con las componentes indicadas. El sistema obtenido será del mismo tipo que `self`, es decir, o un `Sistema`, o un `Vector`, o etc... dependiendo de a qué objeto se aplica el selector.

Cuando el argumento `j` es del tipo `slice(start,stop,step)`, se seleccionan varios componentes, comenzando por aquél cuyo índice es `start`, y seleccionando de `step` en `step` componentes hasta llegar al de índice `stop`. Dicho sistema será del mismo tipo que `self`. Si el primer argumento de `slice` es `None` se seleccionan los componentes empezando por el primero. Si el segundo argumento de `slice` es `None` se recorren todos los índices hasta llegar al último componente. Si se omite el tercer argumento de `slice` (o si el tercer argumento es `None`) entonces `step` es igual a uno. Así, `slice(None, None)` selecciona todos los componentes; `slice(2, None, 2)` selecciona los componentes pares hasta el final; y `slice(4, 11, 3)` selecciona un componente de cada tres comenzando por el cuarto y hasta llegar al undécimo (es decir, los índices 4, 7 y 10).

```

18 <Operador selector por la derecha para la clase Sistema 18>≡
    def __or__(self, j):
        <Texto de ayuda para el operador selector por la derecha para la clase Sistema 17>
        <Operador selector por la derecha cuando el argumento es entero, lista o slice 19a>

This code is used in chunk 9.

```

Recuerde que el operador selector por la derecha funcionará de la misma manera para la clase “padre” `Sistema` como para cualquiera de sus subclases (siempre y cuando dicho método no sea redefinido dentro de la subclase).

19a

<Operador selector por la derecha cuando el argumento es entero, lista o slice 19a>≡

```

if isinstance(j, int):
    return self[j-1]

elif isinstance(j, (list,tuple) ):
    return type(self) ([ self|a for a in j ])

elif isinstance(j, slice):
    start = None if j.start is None else j.start-1
    stop  = None if j.stop  is None else (j.stop if j.stop>0 else j.stop-1)
    step  = j.step  or 1
    return type(self) (self[slice(start,stop,step)])

```

This code is used in chunks 18, 20, 113a, and 118b.

1.2.2 Operador selector por la izquierda para la clase Vector.

En las notas de la asignatura hemos admitido la selección de elementos de un vector por la izquierda, ${}_i\mathbf{v} = \mathbf{v}|_i$. Así que aquí haremos lo mismo; y además ahora es muy sencillo... Como el selector por la izquierda hace lo mismo que el selector por la derecha, basta con llamar al selector por la derecha: `self|i`

19b

<Operador selector por la izquierda para la clase Vector 19b>≡

```

def __ror__(self,i):
    """Hace exactamente lo mismo que el método __or__ por la derecha."""
    return self | i

```

This code is used in chunk 12.

1.2.3 Operador selector por la derecha para la clase Matrix.

Necesitamos reescribir el método de selección por la derecha para `Matrix` ya que también emplearemos el operador `__or__` para particionar matrices. Es decir, con la clase `Matrix` usaremos el selector “|” tanto para seleccionar columnas como para obtener una matriz particionada por columnas (matriz por bloques, `BlockM`) si el argumento del operador es un conjunto (`set`). El siguiente texto de ayuda es auto-explicativo:

19c

<Texto de ayuda para el operador selector por la derecha para la clase Matrix 19c>≡

```

"""
Extrae la i-ésima columna de Matrix; o crea una Matrix con las columnas
indicadas; o crea una BlockM particionando una Matrix por las
columnas indicadas (los índices comienzan por la posición 1)

Parámetros:
    j (int, list, tuple, slice): Índice (o lista de índices) de la
                                columna (o 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, tuple o slice, devuelve la Matrix formada
            por las columnas indicadas en la lista o tupla de índices.
    BlockM: Si j es un set, devuelve la BlockM resultante de particionar
            la matriz a la derecha de 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])] )
>>> # BlockM correspondiente a la partición por la segunda columna
>>> Matrix([Vector([1,0]), Vector([0,2]), Vector([3,0])]) | {2}

BlockM([SisMat([Matrix([Vector([1, 0]), Vector([0, 2])])),
         SisMat([Matrix([Vector([3, 0])])])])
"""
This code is used in chunk 20.
Uses BlockM 116c, Matrix 15b, SisMat 114b, and Vector 12.

```

Implementación del operador selector por la derecha para la clase Matrix. Cuando el argumento es un entero, una lista o un slice todo es exactamente igual en la clase genérica `Sistema`. El modo de particionar una `Matrix` cuando el argumento es un conjunto (`set`) se verá en la sección de la clase `BlockM`.

```

20  <Operador selector por la derecha para la clase Matrix 20>≡
    def __or__(self,j):
        <Texto de ayuda para el operador selector por la derecha para la clase Matrix 19c>
        <Operador selector por la derecha cuando el argumento es entero, lista o slice 19a>
        <Partición de una matriz por columnas de bloques 118a>

This code is used in chunk 15b.

```

1.2.4 Operador transposición de una Matrix.

Implementar el operador selector por la izquierda para la clase `Matrix` es algo más complicado; pues en este caso no es lo mismo operar por la derecha que por la izquierda de una matriz. Vamos a definir primero el operador transposición, que usaremos después para implementar el operador selector por la izquierda (selección de filas mediante la selección de las columnas de la transpuesta).

Notación en Mates 2

Denotamos la *transpuesta* de \mathbf{A} con: \mathbf{A}^T ; que es la matriz tal que $(\mathbf{A}^T)_{ij} = a_{ji}$; $j = 1 : n$.

```
21 <Texto de ayuda para el operador transposición de la clase Matrix 21>≡
    """
    Devuelve la traspuesta de una matriz.

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

    Matrix([ Vector([1, 2, 3]) ])
    """
    This code is used in chunk 22a.
    Uses Matrix 15b and Vector 12.
```

Implementación del operador transposición.

Desgraciadamente Python no dispone del símbolo “ τ ”. 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 refiero con “símbolos asociados a métodos”, repase la sección “Métodos especiales con símbolos asociados” del Notebook “Clases” en la carpeta “TutorialPython” en <https://github.com/mbujosab/nacallib/tree/master/doc/Notebooks/TutorialPython>

Para implementar la transposición haremos uso del método `__invert__` que tiene asociado el símbolo del la tilde “ \sim ”. Desgraciadamente deberemos colocar el símbolo a la izquierda de la matriz, por lo que son dos las diferencias respecto a la notación usada en las notas de la asignatura: el símbolo en sí, y su posición respecto de la matriz:

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

Ahora recuerde que con la segunda forma de instanciar una `Matrix` (véase el resumen de la página 16) 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 los atributos `lista` de los n `Vectores` columna.

(Recuerde que `range(1, self.m+1)` recorre los números: $1, 2, \dots, m$).

22a *<Operador transposición para la clase Matrix 22a>≡*

```
def __invert__(self):
    <Texto de ayuda para el operador transposición de la clase Matrix 21>
    M = Matrix ([ c.lista for c in self ])
    M.cF, M.cC = self.cC, self.cF
    return M
```

This code is used in chunk 15b.
Uses Matrix 15b.

1.2.5 Operador selector por la izquierda para la clase Matrix.

22b *<Texto de ayuda para el operador selector por la izquierda para la clase Matrix 22b>≡*

```
"""Operador selector por la izquierda

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

Parámetros:
    i (int, list, tuple): Índice (o 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.
    BlockM: Cuando i es un set, particiona la matriz por debajo de 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])])
>>> # BlockM correspondiente a la partición por la primera fila
>>> {1} | Matrix([Vector([1,0]), Vector([0,2])])

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

This code is used in chunk 23.
 Uses BlockM 116c, Matrix 15b, SisMat 114b, and Vector 12.

Implementación del operador por la izquierda para la clase Matrix.

Es inmediato implementar el selector por la izquierda (selección de filas) con el operador selector de columnas y la transposición:

`(~self)|i`

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

(la partición en bloques de filas de matrices se verá en la sección de la clase `BlockM`).

```
23  <Operador selector por la izquierda para la clase Matrix 23>≡
    def __ror__(self,i):
        <Texto de ayuda para el operador selector por la izquierda para la clase Matrix 22b>
        if isinstance(i,int):
            return Vector( (~self)|i , rpr='fila' )

        elif isinstance(i, (list,tuple,slice)):
            return ~Matrix( (~self)|i )

        <Partición de una matriz por filas de bloques 117b>
```

This code is used in chunk 15b.
 Uses Matrix 15b and Vector 12.

1.2.6 Ejemplos de uso

Resumen

¡Ahora también hemos implementado en Python el operador “|” (por la derecha y 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...

1.3 Operaciones con Sistemas

Con la definición de la clase `Sistema` y el operador selector “|” por la derecha, ya podemos definir las operaciones de suma de dos sistemas y de producto de un sistema por un escalar. Fíjese que las definiciones de las operaciones en Python (usando el operador “|”) son idénticas a las empleadas en las notas de la asignatura:

1.3.1 Suma de Sistemas

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

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

y la suma de matrices como la matriz tal que

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

Ambas son casos particulares de sumas elemento a elemento entre dos `Sistemas` A y B, de n elementos cada uno:

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

Usando el operador selector podemos “literalmente” transcribir esta definición

```
Sistema ([ (self|i) + (other|i) for i in range(1,len(self)+1) ])
```

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

Hay que tener en cuenta que cuando el `Sistema` es un `Vector` el resultado es un `Vector` y cuando el `Sistema` es una `Matrix` el resultado es una `Matrix`. Es decir, el código debe devolver un objeto del mismo tipo que `self`. Esto lo logramos sustituyendo “Sistema” por “`type(self)`”. Así, la implementación final es:

```
type(self) ([ (self|i) + (other|i) for i in range(1,len(self)+1) ])
```

Por último, nótese que para que la implementación funcione es necesario que los elementos $A_{|i}$ y $B_{|i}$ sean sumables, es decir, es necesario que la operación

$$(self|i) + (other|i)$$

esté definida para cada i .

```
24 <Texto de ayuda para el operador suma en la clase Sistema 24>=
    """Devuelve el Sistema resultante de sumar dos Sistemas

    Parámetros:
        other (Sistema): Otro sistema del mismo tipo y misma longitud

    Ejemplos:
    >>> Sistema([10, 20, 30]) + Sistema([-1, 1, 1])

    Sistema([9, 21, 31])
    >>> Vector([10, 20, 30]) + Vector([-1, 1, 1])

    Vector([9, 21, 31])
    >>> Matrix([[1,5],[5,1]]) + Matrix([[1,0],[0,1]])

    Matrix([Vector([2, 5]); Vector([5, 2])]) """
    This code is used in chunk 25b.
    Uses Matrix 15b, Sistema 9, and Vector 12.
```

De manera análoga definimos diferencia entre sistemas.

25a *<Texto de ayuda para el operador resta en la clase Sistema 25a>*≡

```

    """Devuelve el Sistema resultante de restar dos Sistemas

    Parámetros:
        other (Sistema): Otro sistema del mismo tipo y misma longitud

    Ejemplos:
    >>> Sistema([10, 20, 30]) - Sistema([1, 1, -1])

    Sistema([9, 19, 31])
    >>> Vector([10, 20, 30]) - Vector([1, 1, -1])

    Vector([9, 19, 31])
    >>> Matrix([[1,5],[5,1]]) - Matrix([[1,0],[0,1]])

    Matrix([Vector([0, 5]); Vector([5, 0])])
    """
    This code is used in chunk 25b.
    Uses Matrix 15b, Sistema 9, and Vector 12.

```

Implementación

25b *<Suma y resta de Sistemas 25b>*≡

```

def __add__(self, other):
    <Texto de ayuda para el operador suma en la clase Sistema 24>
    if not type(self)==type(other) or not len(self)==len(other):
        raise ValueError ('Solo se suman Sistemas del mismo tipo y misma longitud')

    return type(self) ([ (self|i) + (other|i) for i in range(1,len(self)+1) ])

def __sub__(self, other):
    <Texto de ayuda para el operador resta en la clase Sistema 25a>
    if not type(self)==type(other) or not len(self)==len(other):
        raise ValueError ('Solo se restan Sistemas del mismo tipo y misma longitud')

    return type(self) ([ (self|i) - (other|i) for i in range(1,len(self)+1) ])

```

This code is used in chunk 9.

1.3.2 Producto de un Sistema por un escalar a su izquierda

El producto de un sistema A por un escalar x a su izquierda es el *sistema*

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

cuya transcripción literal sería

```
Sistema ( [ x*(self[i] for i in range(1,len(self)+1) ] )
```

donde x es un número (`int`, `float`) o un objeto del módulo Sympy (`sympy.Basic`) y donde `self` es `A`.

Como casos particulares tenemos el producto de un *vector* \mathbf{a} por un escalar x a su izquierda, que es el *vector*:

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

Y el producto de una *matriz* \mathbf{A} por un escalar x a su izquierda, que es la *matriz*:

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

Como en los casos particulares se obtienen *sistemas* de tipos particulares (*vectores* en el primer caso y *matrices* en el segundo), debemos sustituir `Sistema` por `type(self)` para obtener sistemas del mismo tipo que `self`:

```
type(self) ( [ x*(self[i] for i in range(1,len(self)+1) ] )
```

26a *<Texto de ayuda para el operador producto por la izquierda en la clase Sistema 26a>*≡

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

Parámetros:
    x (int, float o sympy.Basic): Escalar por el que se multiplica
Resultado:
    Sistema resultante de multiplicar cada componente por x
Ejemplo:
>>> 3 * Sistema([10, 20, 30])

Sistema([30, 60, 90])
"""

This code is used in chunk 26b.
Uses Sistema 9.
```

Implementación

26b *<Producto de un Sistema por un escalar a su izquierda 26b>*≡

```
def __rmul__(self, x):
    <Texto de ayuda para el operador producto por la izquierda en la clase Sistema 26a>
    if isinstance(x, (int, float, sympy.Basic)):
        return type(self)( [ x*(self[i] for i in range(1,len(self)+1) ] )

This code is used in chunk 9.
```

También nos viene bien manejar el opuesto de un `Sistema`: $-\mathbf{A} = -1(\mathbf{A})$.

```

27 <Opuesto de un Sistema 27>≡
    def __neg__(self):
        """Devuelve el opuesto de un Sistema"""
        return -1*self

```

This code is used in chunk 9.
Uses Sistema 9.

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

- En las notas de la asignatura se acepta que el producto de un **Sistema** por un escalar es conmutativo. Por tanto,

$$Ax = xA$$

cuya transcripción será

```
x * self
```

donde **self** es el **Sistema** y **x** es un número entero, u objeto del módulo Sympy (**int**, **float**, **sympy.Basic**).

- El producto de **A**, de **n** componentes, por un vector **x** de \mathbb{R}^n a su derecha se define como

$$Ax = (A_{|1})x_1 + \cdots + (A_{|n})x_n = \sum_{j=1}^n (A_{|j})x_j \quad \text{para } j = 1 : n.$$

cuya transcripción será

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

donde **self** es un **Sistema** y **x** es un (**Vector**).

Fíjese que el *producto punto* (o producto escalar usual en \mathbb{R}^n) de dos vectores **a** y **x** en \mathbb{R}^n es un caso particular en el que el sistema **A** es un vector **a**.

- El producto del sistema **A** de **p** componentes por una matriz **X** de \mathbb{R}^n a su derecha se define como el sistema tal que

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

cuya transcripción será

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

donde **self** es el **Sistema** y **x** es una **Matrix**.

Fíjese que el *producto de matrices* es un caso particular en el que el sistema **A** es una matriz **A**.

Además, sabemos por las notas de la asignatura que en el caso particular de que el sistema **A** sea un vector, el resultado es una combinación lineal de las filas de la matriz **X** (es decir, el resultado es un vector). Para recordar que el vector resultante es una combinación lineal de las filas, lo representaremos en forma de fila.

```

28 <Texto de ayuda para el operador producto por la derecha en la clase Sistema 28>≡
    """Multiplica un Sistema por un número, Vector o una Matrix a su derecha

    Parámetros:
        x (int, float o sympy.Basic): Escalar por el que se multiplica
        (Vector): con tantos componentes como el Sistema
        (Matrix): con tantas filas como componentes tiene el Sistema

    Resultado:
        Sistema del mismo tipo: Si x es int, float o sympy.Basic, devuelve
        el Sistema que resulta de multiplicar cada componente por x
        Objeto del mismo tipo de los componentes del Sistema: Si x es Vector,
        devuelve una combinación lineal de los componentes del Sistema,
        donde los componentes de x son los coeficientes de la combinación.
        Sistema del mismo tipo: Si x es Matrix, devuelve un Sistema cuyas
        componentes son combinación lineal de las componentes originales.

    Ejemplos:
    >>> # Producto por un número
    >>> Vector([10, 20, 30]) * 3

    Vector([30, 60, 90])
    >>> Matrix([[1,2],[3,4]]) * 10

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

    60
    >>> Matrix([Vector([1, 3]), Vector([2, 4])]) * Vector([1, 1])

    Vector([3, 7])
    >>> # Producto por una Matrix
    >>> Vector([1,1,1])*Matrix( ( [1,1,1], [2,4,8], [3,-1,0] ) )

    Vector([6, 4, 9])
    >>> Matrix([Vector([1, 3]), Vector([2, 4])]) * Matrix([Vector([1,1])])

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

This code is used in chunk 29.
Uses Matrix 15b, Sistema 9, and Vector 12.

```

Implementación

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 el número cero a un elemento del `Sistema` puede no tener sentido, haremos el siguiente truco: el primer objeto de la suma será el primer elemento de la lista multiplicado por el número cero.

29 *<Producto de un Sistema por un escalar, un Vector o una Matrix a su derecha 29>≡*

```

def __mul__(self,x):
    <Texto de ayuda para el operador producto por la derecha en la clase Sistema 28>
    if isinstance(x, (int, float, sympy.Basic)):
        return x*self

    elif isinstance(x, Vector):
        if len(self) != x.n: raise ValueError('Sistema y Vector incompatibles')
        return sum([(self|j)*(x|j) for j in range(1,len(self)+1)], 0*self|1)

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

```

This code is used in chunk 9.
 Uses Matrix 15b and Vector 12.

1.4 La clase transformación elemental T

Notación en Mates 2

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

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

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

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

Comentario sobre la notación. Como una transformación elemental se puede lograr mediante un producto con una matriz elemental, **la notación empleada busca parecerse a la notación del producto matricial**:

Al poner la *abreviatura* “ τ ” de la transformación elemental a derecha es como si multiplicáramos la matriz $\mathbf{A}_{m \times n}$ por la derecha por la correspondiente matriz elemental

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

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

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

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

Fíjese que 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 qué componente que no coincide con los de la matriz \mathbf{I} de orden n .⁴

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

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

Mates II	Python	Mates II	Python
$\mathbf{A}_{\tau_{[i \rightleftharpoons j]}}$	<code>A & T({i,j})</code>	${}_{\tau} \mathbf{A}_{[i \rightleftharpoons j]}$	<code>T({i,j}) & A</code>
$\mathbf{A}_{\tau_{[(a)j]}}$	<code>A & T((a,j))</code>	${}_{\tau} \mathbf{A}_{[(a)j]}$	<code>T((a,j)) & A</code>
$\mathbf{A}_{\tau_{[(a)i+j]}}$	<code>A & T((a,i,j))</code>	${}_{\tau} \mathbf{A}_{[(a)i+j]}$	<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 para las ternas (a, i, j) :
 - (a) La columna (fila) que cambia es la del índice que aparece en última posición.
 - (b) El escalar de la primera posición multiplica a la columna (fila) correspondiente al índice que le precede.

⁴Fíjese 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 \mathbf{E} adecuado para poder realizar el producto $\mathbf{A} \mathbf{E}$ o el producto $\mathbf{E} \mathbf{A}$)

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

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

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

Así, usando abreviaturas y 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

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

... ¡Pero gracias a las abreviaturas no hemos necesitado indicar el orden de las matrices elementales en ningún momento!

```
31 <Texto de ayuda de la clase T (Transformación Elemental) 31>≡
    """Clase T

    T ("Transformación elemental") guarda en su atributo 't' una abreviatura
    (o una secuencia de abreviaturas) de transformaciones elementales. El
    método __and__ actúa sobre otra T para crear una T que es composición de
    transformaciones elementales (una 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): (escalar, índice). Abrev. transf. Tipo II que multiplica
                  el vector correspondiente al índice por el escalar
                  : (escalar, índice1, índice2). Abrev. transformación Tipo I
                    que suma al vector correspondiente al índice2 el vector
                    correspondiente al índice1 multiplicado por el escalar
        (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( (5,2) ) )

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 38.
Uses I 102, Matrix 15b, and T 38.

```

1.4.1 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 bien 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`.

Así pues, definiremos una nueva clase que almacene las *abreviaturas* “ t_i ” 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 (que en última instancia será una secuencia de operaciones a ejecutar sobre una `Matrix`).

El nuevo objeto, `T` (“transformación elemental”), 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 lista de `Ts`) empleada para instanciar.

```

32  <Iniciación de la clase T (Transformación Elemental) 32>≡
    def __init__(self, t, rpr='v'):
        """Inicializa una transformación elemental"""
        <Método auxiliar CreaLista que devuelve listas 33b>
        <Creación del atributo t cuando se instancia con otra T o lista de Ts 103>
        else:
            self.t = t
            <Verificación de que las abreviaturas corresponden a transformaciones elementales 33a>
            self.rpr = rpr

This code is used in chunk 38.

```

Una transformación elemental no puede multiplicar por cero, ni sumar a un elemento un múltiplo de sí mismo. Además, un intercambio solo tiene sentido a lo sumo entre dos elementos.

33a *<Verificación de que las abreviaturas corresponden a transformaciones elementales 33a>≡*

```

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 un intercambio')

```

This code is used in chunk 32.
 Uses CreaLista 33b.

Con 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. Lo usaremos al instanciar T con una lista de Ts; y al componer transformaciones elementales.

33b *<Método auxiliar CreaLista que devuelve listas 33b>≡*

```

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 32, 34, 36, 87d, and 95b.
 Defines:
 CreaLista, used in chunks 33a, 34, 36, 87d, 95b, and 103.

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

33c *<Texto de ayuda para la composición de Transformaciones Elementales T 33c>≡*

```

"""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 sus filas

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 34.
Uses Matrix 15b and T 38.

```

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)⁵. Si `other` es un `Vector` o una `Matrix`, se llama al método `__rand__` de la clase `other` (que transformará los elementos del vector en el primer caso, y las filas de la matriz en el segundo; y que veremos más adelante).

```

34  <Composición de Transformaciones Elementales o aplicación sobre las filas de una Matrix 34>≡
    def __and__(self, other):
        <Texto de ayuda para la composición de Transformaciones Elementales T 33c>
        <Método auxiliar CreaLista que devuelve listas 33b>
        if isinstance(other, T):
            return T(CreaLista(self.t) + CreaLista(other.t), self.rpr)

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

    This code is used in chunk 38.
    Uses CreaLista 33b, Matrix 15b, T 38, and Vector 12.

```

1.4.2 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

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

¡Fíjese como efectivamente hemos logrado que la notación con abreviaturas se comporte como la notación matricial!

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.

⁵Recuerde que la suma de listas (`list + list`) concatena las listas

35a \langle Operador transposición para la clase T 35a $\rangle \equiv$

```
def __invert__(self):
    """Transpone la lista de abreviaturas (invierte su orden)"""
    return T( list(reversed(self.t)), self.rpr) if isinstance(self.t, list) else self
```

This code is used in chunk 38.
Uses T 38.

1.4.3 Potencias e inversa 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:

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

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-ésima de una transformación elemental. Si n es -1, calcula la inversa:

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

que en Jupyter veríamos como $\tau_{\begin{smallmatrix} [1 \Rightarrow 2] \\ [(3)2] \\ [(-1)2+3] \end{smallmatrix}}$. Al implementar el método, definimos la potencia de manera recursiva (con una función auxiliar `lambda`). Además, si n es cero, devolveremos una transformación que no haga nada (identidad); por ejemplo $\tau_{[1 \Rightarrow 1]}$.

35b \langle Potencia de una T 35b $\rangle \equiv$

```
def __pow__(self,n):
    """Calcula potencias de una T (incluida la inversa)"""
     $\langle$ Método auxiliar que calcula la inversa de una Transformación elemental 36a $\rangle$ 
    if not isinstance(n,int):
        raise ValueError('La potencia no es un entero')

    potencia = lambda x, n: x if n==1 else x & potencia(x, n-1)
    t = potencia(self,abs(n)) if n!=0 else T({1})

    return t if n>0 else Tinversa(t)
```

This code is used in chunk 38.
Uses T 38.

36a *⟨Método auxiliar que calcula la inversa de una Transformación elemental 36a⟩≡*

```
def Tinversa ( self ):
    """Calculo de la inversa de una transformación elemental"""
    ⟨Método auxiliar CreaLista que devuelve listas 33b⟩

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

    return ~T( listaT, self.rpr)
```

This code is used in chunk 35b.
Uses CreaLista 33b, fracc 43, and T 38.

1.4.4 Transformaciones elementales “espejo”

Al diagonalizar por semejanza, y aplicar transformaciones elementales por la derecha, que es lo mismo que multiplicar por una matriz invertible por la derecha, necesitaremos expresar la correspondiente matriz inversa mediante una secuencia de transformaciones elementales de la filas de la matriz identidad. Esto se logra con el método *espejo*.⁶

36b *⟨Transformación elemental espejo de una T 36b⟩≡*

```
def espejo ( self ):
    """Calculo de la transformación elemental espejo de otra"""
    ⟨Método auxiliar CreaLista que devuelve listas 33b⟩
    return T([(j[0],j[2],j[1]) if len(j)==3 else j for j in CreaLista(self.t)],self.rpr)
```

This code is used in chunk 38.
Defines:
 espejo, used in chunk 69.
Uses CreaLista 33b and T 38.

1.4.5 Sustitución de variables simbólicas

Sustituye la variable *c* por *v*, donde *v* puede ser un valor, u otra variable simbólica.

36c *⟨Sustitución de variables simbólicas en una Transformación elemental 36c⟩≡*

```
def subs(self, s=[]):
    ⟨Método auxiliar CreaLista que devuelve listas 33b⟩
    t = [sympy.S(item).subs(CreaLista(s)) for item in CreaLista(self.t)]
    self.t = [set(i) if isinstance(i, sympy.sets.sets.FiniteSet) else i for i in t]
    return self
```

⁶Al no encontrar ningún nombre en los manuales de Álgebra Lineal para este concepto, he adoptado este descriptivo nombre.

This code is used in chunk 38.
 Uses `CreaLista` 33b.

1.4.6 Igualdad entre transformaciones elementales

37a *<Otros métodos de la clase Transformación elemental 37a>≡*

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

This definition is continued in chunk 37b.
 This code is used in chunk 38.

1.4.7 T es iterable

Haremos que `T` sea iterable con los procedimientos “mágicos” `__getitem__`, que permite seleccionar pasos de `T`, y `__setitem__`, que permite modificar pasos de `T`. Con `__len__` podremos contar el número de pasos de `T`.

37b *<Otros métodos de la clase Transformación elemental 37a>+≡*

```
def __getitem__(self,i):
    """ Devuelve las transformaciones elementales del i-ésimo paso """
    return T(self.t[i])

def __setitem__(self,i,value):
    """ Modifica las transformaciones elementales del i-ésimo paso """
    self.t[i]=value

def __len__(self):
    """Número de pasos de T """
    return len(self.t)
```

This code is used in chunk 38.
 Uses `T` 38.

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

```

38  <Definición de la clase T (Transformación Elemental) 38>≡
    class T:
        <Texto de ayuda de la clase T (Transformación Elemental) 31>
        <Inicialización de la clase T (Transformación Elemental) 32>
        <Composición de Transformaciones Elementales o aplicación sobre las filas de una Matrix 34>
        <Operador transposición para la clase T 35a>
        <Potencia de una T 35b>
        <Transformación elemental espejo de una T 36b>
        <Sustitución de variables simbólicas en una Transformación elemental 36c>
        <Otros métodos de la clase Transformación elemental 37a>
        <Representación de la clase T 105>

This code is used in chunk 42.
Defines:
    T, used in chunks 5a, 31, 33–37, 39–41, 45–47, 50, 51, 53, 58–60, 62, 63c, 65b, 66a, 68–70, 72–74, 77a, 100a, 103,
    and 105–107.

```

1.5 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**.

```
39  <Texto de ayuda de las transformaciones elementales de un Sistema 39>≡
    """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 40a.
    Uses Sistema 9 and T 38.
```

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.)

40a \langle Transformaciones elementales de los elementos de un Sistema 40a $\rangle \equiv$

```
def __and__(self,t):
     $\langle$ Texto de ayuda de las transformaciones elementales de un Sistema 39 $\rangle$ 
    if isinstance(t.t,set):
        self.lista = [ (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)].copy()

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

    elif isinstance(t.t,tuple) and (len(t.t) == 3):
        self.lista = [ (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)].copy()

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

    return self
```

This code is used in chunk 9.
Uses T 38.

Observación 1. Al actuar sobre `self.lista`, las transformaciones elementales modifican los `Sistemas`.

1.5.1 Transformaciones elementales de las filas de una Matrix

40b \langle Texto de ayuda de las transformaciones elementales de las filas de una Matrix 40b $\rangle \equiv$

```
"""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 41a.
Uses Matrix 15b and T 38.

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:

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

41a *<Transformaciones elementales de las filas de una Matrix 41a>≡*

```
def __rand__(self,t):
    <Texto de ayuda de las transformaciones elementales de las filas de una Matrix 40b>

    if isinstance(t.t, (set, tuple) ):
        self.lista = (~(self & t)).lista.copy()

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

    return self
```

This code is used in chunk 15b.
Uses T 38.

Observación 2. Al actuar sobre `self.lista`, las transformaciones elementales modifican la *Matrix*.

1.5.2 Transformaciones elementales por la izquierda de un Vector

Hacen lo mismo que por la derecha (como ocurre con el operador selector)

41b *<Transformaciones elementales por la izquierda de un Vector 41b>≡*

```
def __rand__(self,t):
    """Hace exactamente lo mismo que el método __and__ por la derecha."""
    return self & t
```

This code is used in chunk 12.

Observación 3. Las transformaciones elementales modifican el *Vector*.

1.6 Librería completa

Finalmente creamos la librería `nacal.py` concatenando los trozos de código que se describen en este fichero de documentación.

Para que los **Vectores** funcionen como un espacio vectorial, importamos la librería **Sympy** con el código:

```
import sympy
```

Así podremos usar números racionales e irracionales (incluso el cuerpo de polinomios). Como queremos que la librería emplee números racionales siempre que sea posible, definimos tres métodos auxiliares: `fracc(a/b)` es la fracción $\frac{a}{b}$; `numer(a,b)`; y `denom(a,b)` (véase la página siguiente). Para usar el número racional $\frac{1}{3}$ escribiremos `fracc(1,3)`, y para usar un número irracional como $\sqrt{2}$ escribiremos `sympy.sqrt(2)`. La librería **Sympy** ya se ocupa de que Jupyter represente adecuadamente estos objetos (incluso simplificando expresiones, de manera que si escribimos el número irracional `fracc(2,sympy.sqrt(2))`, es decir $\frac{2}{\sqrt{2}}$, Jupyter lo simplificara, representándolo como $\sqrt{2}$).

```
42 <nacal.py 42>≡
# coding=utf8
import sympy
from IPython.display import display, Math, display_png
import tempfile
from os.path import join
<Métodos auxiliares para usar coeficientes racionales cuando sea posible 43>

<Método html general 84a>
<Método latex general 84b>
<Simplificación de expresiones simbólicas 85a>
<Filtrado de secuencias de transformaciones 58>
<Pinta un objeto en Jupyter 85b>

<Definición de la clase Sistema 9>
<Definición de la clase Vector 12>
<Definición de la clase Matrix 15b>
<Definición de la clase T (Transformación Elemental) 38>

<Definición del vector nulo: V0 101a>
<Definición de la matriz nula: M0 101b>
<Definición de la matriz identidad: I 102>

<Definición del método particion 117a>
<Definición del procedimiento de generación del conjunto clave para particionar 119a>
<Definición de la clase SisMat 114b>
<Definición de la clase BlockM 116c>

<Tres métodos de eliminación por columnas 49>
<Tres métodos de eliminación por filas 55>
<Representación de un proceso de eliminación 106c>
<Invirtiendo una matriz 59>
<La clase SubEspacio 80a>
<La clase EAfin 81a>
<Resolviendo un sistema homogéneo 61>
<Resolviendo un Sistema de Ecuaciones Lineales 63b>
<Calculando el determinante 65b>
<Diagonalizando una matriz por bloques triangulares (por Semejanza) 68>
<Diagonalizando Ortogonalmente una matriz simétrica 70>
<Diagonalizando una matriz por Congruencia 72>
```

Root chunk (not used in this document).

43 *⟨Métodos auxiliares para usar coeficientes racionales cuando sea posible 43⟩≡*

```
def fracc(a,b):
    """Transforma la fracción a/b en un número racional si ello es posible"""
    if all( [ isinstance(i, (int, float, sympy.Rational) ) for i in (a,b) ] ):
        return sympy.Rational(a, b)
    else:
        return a/b

def numer(a,b):
    """Devuelve el numerador de a/b si la fracción es un número racional,
    si no devuelve a/b"""
    if all( [ isinstance(i, (int, float, sympy.Rational) ) for i in (a,b) ] ):
        return fracc(a,b).p
    else:
        return a/b

def denom(a,b):
    """Devuelve el denominador de a/b si la fracción es un número
    racional, si no devuelve 1"""
    if all( [ isinstance(i, (int, float, sympy.Rational) ) for i in (a,b) ] ):
        return fracc(a,b).q
    else:
        return 1
```

This code is used in chunk 42.

Defines:

`denom`, used in chunk 46.
`fracc`, used in chunks 36a, 45b, 47b, 63c, 66a, 68, 90a, and 100a.
`numer`, used in chunk 46.

Notebooks de Jupyter que muestran el uso del módulo `nacal`

Consulte el Notebook sobre el **uso del módulo `nacal`** en la carpeta “Notebooks” en <https://github.com/mbujosab/nacallib/tree/master/doc/Notebooks>.

Notebooks de Jupyter que muestran el uso del módulo `nacal`

En el siguiente enlace:

<https://github.com/mbujosab/nacal-Jupyter-Notebooks>

puede ver Notebooks correspondientes a las distintas lecciones de mi curso de Álgebra Lineal (Matemáticas II), y los puede ejecutar online en mybinder.org

Capítulo 2

Algoritmos del curso

En el curso usamos tres métodos de eliminación. La *eliminación* por columnas de izquierda a derecha encuentra una matriz pre-escalada (todos los componentes a la derecha de los pivotes son cero), la *eliminación Gaussiana* por columnas nos da una forma escalada **L** (al reordenar las columnas de la matriz pre-escalada), y la *eliminación Gauss-Jordan* por columnas nos da la forma escalada reducida por columnas **R**, (los componentes a derecha e izquierda de los pivotes son cero y cada pivote es igual a 1). Es decir, los últimos métodos modifican las matrices obtenidas con los métodos anteriores:

- La eliminación encuentra los pivotes (matriz pre-escalada).
- La eliminación Gaussiana reordena las columnas de la matriz pre-escalada para obtener una escalada.
- La eliminación Gauss-Jordan reduce la matriz escalada.

Pero antes veamos la operación de búsqueda de pivotes

Búsqueda de pivotes

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 `ppivote` nos indicará el *primer índice* mayor que `k` de un componente no nulo del **Vector**. Como por defecto `k=0`, si no especificamos el valor de `k`, entonces `ppivote` nos devuelve la *posición de pivote* de un **Vector**. Si todos los componentes de índice mayor que `k` son nulos, `ppivote` nos devuelve el valor cero. Así, si $\mathbf{a} = (0, 5, 0, 5)$, entonces

`ppivote(a)=2;` `ppivote(a,1)=2;` `ppivote(a,2)=4;` `ppivote(a,4)=0.`

Lo programaremos con una función auxiliar `lambda`.¹

Búsqueda de nuevos pivotes

Cada pivote debe estar situado en una columna diferente. Para lograr que en todos los casos sea así, generamos el conjunto `colExcluida` que contiene los índices de todas las columnas en las que ya hemos encontrado un pivote. Inicialmente `colExcluida` es un conjunto vacío; y cada vez que encontremos una columna con pivote, su correspondiente índice `p` será incluido en este conjunto con `colExcluida.add(p)`. De esta manera, para cada fila buscaremos (con `ppivote`) un componente no nulo, y si dicho componente se encuentra en una columna con pivote, buscaremos el siguiente componente no nulo de la fila que esté en una columna no ocupada por un pivote encontrado anteriormente. Si esto no es posible, `ppivote` devolverá el valor 0 que no corresponde a ningún índice de columna, por lo que habremos terminado de buscar.

¹véase documentación de Python.

45a *⟨Definición del método auxiliar BuscaNuevoPivote 45a⟩*≡

```
def BuscaNuevoPivote(self, sust=[], r=0):
    ppivote = lambda v, k=0:\
        ( [i for i,c in enumerate(v.subs(sust), 1) if (c!=0 and i>k)] + [0] )[0]
    pp = ppivote(self, r)
    while pp in colExcluida:
        pp = ppivote(self, pp)
    return pp
```

This code is used in chunks 49–53 and 68.
 Defines:
 BuscaNuevoPivote, used in chunks 49–53 and 68.

2.1 Operaciones empleadas las distintas variantes de eliminación

2.1.1 La operación de eliminación de componentes

La **eliminación** usa cada pivote para anular las componentes de su fila situadas a su derecha. La eliminación Gauss-Jordan también anula las componentes de su izquierda.

Indicaremos los componentes a eliminar con la función `celim` (que definimos como una función auxiliar `lambda`):

- Para eliminar los componentes cuyo índice j es mayor que p (derecha del pivote): `celim = lambda j: j > p`
- Para eliminar los componentes cuyo índice j es menor que p (izquierda del pivote): `celim = lambda j: j < p`

Así, `filter(celim, range(1,A.n+1))` contendrá los índices de las columnas sobre las que queremos operar.

Usando únicamente transformaciones Tipo I

(Véase la demostración de que toda matriz se puede pre-escalonar en las notas de la asignatura). El siguiente ejemplo muestra las operaciones para una matriz concreta:

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

Con solo tres transformaciones elementales hemos pre-escalonado la matriz. El coste de este método ha sido el uso de operaciones con fracciones, ya que para eliminar el número b usando el pivote $a \neq 0$, la estrategia empleada ha sido:

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

En un paso se elimina b restándole un múltiplo de a . El método consiste en repetir, fila a fila el siguiente código:

45b *⟨Uso del pivote para eliminar componentes con trasformaciones Tipo I 45b⟩*≡

```
Tr = T([ (-fracc(i|A|j, i|A|p), p, j) for j in filter(celim, range(1,A.n+1)) ])
```

This code is used in chunks 52, 53b, and 74.
 Uses `fracc` 43 and `T` 38.

donde i es el índice de la fila en la que se está trabajando para eliminar componentes, p es el índice de la columna donde se encuentra el pivote y j recorre la lista de índices correspondientes a las columnas de los componentes a eliminar (tal como se ha descrito la función `celim`), de manera que se define la sucesión de transformaciones

$$\left[\left(\frac{\tau}{a_{ip}} \right) \mathbf{p+j} \right], \quad \text{con } \mathbf{j} \text{ recorriendo las columnas a la derecha del pivote (si eliminamos de izda a dcha)}$$

donde $i|A|j$ es el componente a_{ij} , a eliminar y $i|A|p$ es el pivote a_{ip} . Tras definir las transformaciones elementales Tr , se apuntan y se aplican sobre las columnas.

Usando transformaciones Tipo I y II para evitar las fracciones cuando sea posible.

Para escalonar una matriz cuyos componentes son números enteros no es necesario trabajar con fracciones. Por ejemplo

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

Con este procedimiento, aunque se realizan más transformaciones elementales, se evita el uso de fracciones (siempre y cuando la matriz sea entera). La estrategia consiste en eliminar el número b usando el pivote $a \neq 0$ encadenando dos operaciones:

$$-a(b) + b(a).$$

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

$$\begin{array}{ll} (- (i|A|p), & j) \quad \# \text{ Tipo II} \\ ((i|A|j), p, j) \quad \# \text{ Tipo I} \end{array} \quad \text{es decir, } \left[\frac{\tau}{(-a_{ip})} \mathbf{j} \right] \left[\frac{\tau}{(a_{ij})} \mathbf{p+j} \right].$$

El problema de esta solución es que si $a = 3$ y $b = 3$, bastaría con restar $b - a$; pero la solución de arriba calcularía $-3(b) + 3(a)$, por lo que todos los números de las columnas \mathbf{r} y \mathbf{j} se multiplicarían por tres sin que ello sea realmente necesario, así que podemos terminar con matrices pre-escalonadas de números innecesariamente grandes.

Considere $a = 6$ y $b = 4$, como $\frac{b}{a} = \frac{2}{3}$, para eliminar b basta con la operación $-3(b) + 2(a)$; es decir, basta con simplificar la fracción $\frac{b}{a}$ y multiplicar b por el denominador (cambiado de signo) y a por el numerador de la fracción simplificada. El siguiente código usa esta idea (donde si \mathbf{n} es un número racional, entonces `numer` nos da el numerador de la fracción simplificada y `denom` el denominador... y si no es racional, `numer` nos da $\frac{b}{a}$ y `denom` es igual a 1. Véase *⟨Métodos auxiliares para usar coeficientes racionales cuando sea posible 43⟩*).

46 *⟨Uso del pivote para eliminar componentes evitando dividir 46⟩*≡
`Tr = T([T([(denom((i|A|j),(i|A|p)), j), \`
`(-numer((i|A|j),(i|A|p)), p, j)]) \`
`for j in filter(celim, range(1,A.n+1))])`
 This code is used in chunks 49, 51, and 72.
 Uses `denom` 43, `numer` 43, and `T` 38.

Pero si la matriz tiene componentes irracionales, no hay más remedio que aplicar la estrategia

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

donde a y/o b son números irracionales (...o polinomios...o variables simbólicas).

2.1.2 La operación de intercambio de columnas

La eliminación Gaussiana reordena las columnas para obtener una matriz escalonada. Para ello necesitamos el **intercambio** de columnas. Escalonar la matriz supone que el orden de las columnas depende de la posición de pivote de cada una de ellas (dejando las columnas nulas al final). Así, el pivote más alto (que es el primero que hemos encontrado, pues recorremos las filas de arriba a abajo) se sitúa en la primera columna, el segundo en la segunda columna, etc. Llamamos p al índice de la columna donde encontramos el pivote, y r a la posición que debería ocupar dicha columna en la matriz escalonada (y que coincide con el número de pivotes encontrados hasta ese momento). Así, cuando se encuentra el primer pivote ($r==1$) la correspondiente columna se coloca en la primera posición, cuando se encuentra el segundo ($r==2$) la correspondiente columna se coloca en la segunda posición, etc.

47a *<Intercambio de columnas para escalonar 47a>≡*
`Tr = T([{p, r}])`
 This code is used in chunks 50 and 53a.
 Uses T 38.

2.1.3 La operación de normalización de los pivotes

La eliminación Gauss-Jordan también elimina las componentes a la izquierda de los pivotes (de manera similar a lo descrito en la Sección 2.1.1) y normaliza los pivotes para que todos sean iguales a “1”. Para ello divide cada columna no nula por el valor de su pivote. De nuevo i es el índice de la fila en que se está trabajando, y p el índice de la columna donde se encuentra el pivote, por lo que $i|A|p$ es el pivote:

47b *<Normalización del pivote para que sea igual a uno 47b>≡*
`Tr = T([(frac(1, i|A|p), p)])`
 This code is used in chunks 51 and 53b.
 Uses frac 43 and T 38.

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

Tras realizar cualquiera de las tres operaciones descritas más arriba, se ha generado una sucesión de transformaciones Tr , cuya lista concatenamos a una lista de **transformaciones** que más adelante se guardan como un atributo de la correspondiente clase (si no se hubiera definido ninguna transformación, se concatena una lista vacía). Por último, antes de pasar a la siguiente fila de la matriz, se aplica Tr sobre las *columnas* de la *Matrix*.

47c *<Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>≡*
`transformaciones += [Tr] if Tr.t else []`
`A & T(Tr)`
 This code is used in chunks 49–53.
 Uses T 38.

2.2 Eliminación “de izquierda a derecha”, Gaussiana y Gauss-Jordan

2.2.1 Primero evitando las fracciones... en la medida de lo posible

Como a los estudiantes no les suele agradar el uso de fracciones, la librería da preferencia a este modo de operar.

El método de eliminación evitando divisiones corresponde a la clase `Elim`. En él se define la función `celim` para que se anulen los componentes a la derecha de cada pivote. Como no hay reordenamiento, una vez encontrado un pivote, el índice `p` de su columna se incorpora al conjunto `colExcluida`, para no buscar nuevos pivotes en dicha columna. La operación empleada corresponde al *⟨Uso del pivote para eliminar componentes evitando dividir 46⟩*.

El argumento `data` es una `Matrix`, o algo que permita generar una `Matrix`, pues `A = Matrix(data)`. Si el argumento `rep` es distinto de cero, Jupyter representará los pasos de eliminación (por defecto `rep=0`).

El algoritmo recorre las filas de la matriz `A` (índice `i`). Para cada fila, busca un nuevo pivote que esté situado en alguna columna no ocupada por otros pivotes; `p` es el índice de la columna donde se ha encontrado un pivote (si no se encuentra ninguno, `p` vale cero). Cuando `p` es distinto de cero (i.e, cuando se ha encontrado un pivote en la fila `i`):

- el contador de pivotes `r` suma uno más.
- Se aplica la eliminación de los componentes indicados con `celim`
- Se añade el índice `p` al conjunto `colExcluida`

Una vez se han recorrido todas las filas, se guarda la lista de transformaciones aplicadas a las columnas como segundo componente de la lista `pasos` (el segundo componente corresponderá a las transformaciones de las columnas). Para finalizar, *⟨Se guardan los atributos `tex` y `pasos` (y se muestran los pasos si se pide) 106a⟩*,² se guarda el atributo `rango` y se devuelve la matriz transformada.

²por comodidad incluyo dos atributos más: `TrF` es la `T`transformación aplicada a las filas y `TrC` la aplicada a las columnas.

49 *<Tres métodos de eliminación por columnas 49>≡*

```

class Elim(Matrix):
    def __init__(self, data, rep=0, sust=[], repsust=0):
        """Devuelve una forma pre-escalada de Matrix(data)

        operando con las columnas (y evitando operar con fracciones).
        Si rep es no nulo, se muestran en Jupyter los pasos dados"""
        <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
        <Definición del método auxiliar BuscaNuevoPivote 45a>
        celim = lambda x: x > p
        A = Matrix(data.subs(sust)); A.TrC = A.TrF = []; A.pasos=[A.TrF, A.TrC];
        r = 0; transformaciones = []; colExcluida = set()
        for i in range(1,A.m+1):
            p = BuscaNuevoPivote(i|A, sust);
            if p:
                r += 1
                <Uso del pivote para eliminar componentes evitando dividir 46>
                <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>
                A = A.subs(sust);
                colExcluida.add(p)
        pasos = [[], transformaciones]
        pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
        self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
        self.rango = r
        super(self.__class__,self).__init__(A)
        self.__class__ = Matrix

        if isinstance (data, Matrix):
            self.cF, self.cC = data.cF, data.cC

```

This definition is continued in chunks 50–53.

This code is used in chunk 42.

Defines:

Elim, used in chunks 50, 55, 62, 63b, 76, 77a, and 96.

Uses BuscaNuevoPivote 45a, filtradopasos 58, and Matrix 15b.

La clase `ElimG` corresponde a la eliminación Gaussiana y su código es parecido al anterior. Las diferencias son:

- La matriz `A` está pre-escalada: `A = Elim(data)` .
- No es necesario definir la función `celim`, pues ahora no se anulan componentes (solo se intercambian columnas)
- La operación empleada es el *Intercambio de columnas para escalar* 47a)
- Puesto que el primer pivote ocupa la primera columna, el segundo la segunda, etc.; ahora es el número de pivotes `r` encontrados el que se incluye en el conjunto `colExcluida`
- Los pasos totales dados son la concatenación de los dados sobre las columnas al pre-escalar (`A.pasos[1]`) y los intercambios de las columnas (`T(transformaciones)`)

Todo lo demás es idéntico.

```

50 <Tres métodos de eliminación por columnas 49>+≡
    class ElimG(Matrix):
        def __init__(self, data, rep=0, sust=[], repsust=0):
            """Devuelve una forma escalonada de Matrix(data)

            operando con las columnas (y evitando operar con fracciones).
            Si rep es no nulo, se muestran en Jupyter los pasos dados"""
            <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
            <Definición del método auxiliar BuscaNuevoPivote 45a>
            A = Elim(data, 0, sust);
            r = 0; transformaciones = []; colExcluida = set()
            for i in range(1,A.m+1):
                p = BuscaNuevoPivote(i|A, sust);
                if p:
                    r += 1
                    <Intercambio de columnas para escalar 47a>
                    <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>
                    colExcluida.add(r)
            pasos = [ [], A.pasos[1]+[T(transformaciones)] ]
            pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
            self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
            self.rango = r
            super(self.__class__,self).__init__(A)
            self.__class__ = Matrix

            if isinstance (data, Matrix):
                self.cF, self.cC = data.cF, data.cC

This code is used in chunk 42.
Defines:
    ElimG, used in chunks 51, 56, 68, 70, and 96.
Uses BuscaNuevoPivote 45a, Elim 49, filtradopasos 58, Matrix 15b, and T 38.

```

La clase `ElimGJ` corresponde a la eliminación Gauss-Jordan y su código es algo más complicado (no mucho). Las diferencias con los anteriores son:

- La matriz `A` está escalada: `A = ElimG(data)`
- La función `celim`, indica que se deben eliminar las componentes a la izquierda de cada pivote.
- Se emplean dos operaciones.

1. Primero se recorren todas las filas de la matriz haciendo *(Uso del pivote para eliminar componentes evitando dividir 46)*
(después se guarda la lista de transformaciones en la variable `transElimIzda`)
 2. A continuación se “resetea” las variables `r`, `transformaciones` y `columnOcupada` y, por segunda vez, se recorren todas las filas para la aplicar la *(Normalización del pivote para que sea igual a uno 47b)*
(¡con esta operación inevitablemente aparecen las fracciones!...aunque se han demorado hasta el último momento.)
- Los pasos totales dados son la concatenación de los dados sobre las columnas al escalonar (`A.pasos[1]`), las eliminaciones de los componentes a la izquierda de los pivotes (`transElimIzda`) y las transformaciones para normalizar los pivotes (`T(transformaciones)`).

Todo lo demás es idéntico.

```

51 <Tres métodos de eliminación por columnas 49>+=
class ElimGJ(Matrix):
    def __init__(self, data, rep=0, sust=[], repsust=0):
        """Devuelve una forma escalonada reducida de Matrix(data)

        operando con las columnas (y evitando operar con fracciones
        hasta el último momento). Si rep es no nulo, se muestran en
        Jupyter los pasos dados"""
        <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
        <Definición del método auxiliar BuscaNuevoPivote 45a>
        celim = lambda x: x < p
        A = ElimG(data, 0, sust);
        r = 0; transformaciones = []; colExcluida = set()
        for i in range(1,A.m+1):
            p = BuscaNuevoPivote(i|A, sust);
            if p:
                r += 1
                <Uso del pivote para eliminar componentes evitando dividir 46>
                <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>
                colExcluida.add(p)

        transElimIzda = transformaciones

        r = 0; transformaciones = []; colExcluida = set()
        for i in range(1,A.m+1):
            p = BuscaNuevoPivote(i|A, sust);
            if p:
                r += 1
                <Normalización del pivote para que sea igual a uno 47b>
                <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>
                colExcluida.add(p)

        pasos = [ [], A.pasos[1] + transElimIzda + [T(transformaciones)] ]
        pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
        self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
        self.rango = r
        super(self.__class__,self).__init__(A)
        self.__class__ = Matrix

        if isinstance (data, Matrix):
            self.cF, self.cC = data.cF, data.cC

```

This code is used in chunk 42.

Defines:

ElimGJ, used in chunks 57, 59, 60b, and 96.

Uses BuscaNuevoPivote 45a, ElimG 50, filtradopasos 58, Matrix 15b, and T 38.

2.2.2 Si no evitamos las fracciones realizamos menos operaciones

Las clases `Elimr`, `ElimrG` y `ElimrGJ` hacen lo mismo, pero haciendo *⟨Uso del pivote para eliminar componentes con transformaciones Tipo I 45b⟩*

Muestro su código sin más explicaciones...

52 *⟨Tres métodos de eliminación por columnas 49⟩+≡*

```
class Elimr(Matrix):
    def __init__(self, data, rep=0, sust=[], repsust=0):
        """Devuelve una forma pre-escalada de Matrix(data)

        operando con las columnas. Si rep es no nulo, se muestran en
        Jupyter los pasos dados"""
        ⟨Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b⟩
        ⟨Definición del método auxiliar BuscaNuevoPivote 45a⟩
        celim = lambda x: x > p
        A = Matrix(data.subs(sust)); A.TrC = A.TrF = []; A.pasos=[A.TrF, A.TrC];
        r = 0; transformaciones = []; colExcluida = set()
        for i in range(1,A.m+1):
            p = BuscaNuevoPivote(i|A, sust);
            if p:
                r += 1
                ⟨Uso del pivote para eliminar componentes con transformaciones Tipo I 45b⟩
                ⟨Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c⟩
                A = A.subs(sust);
                colExcluida.add(p)
        pasos = [[], transformaciones]
        pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
        self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
        self.rango = r
        super(self.__class__,self).__init__(A)
        self.__class__ = Matrix

        if isinstance(data, Matrix):
            self.cF, self.cC = data.cF, data.cC
```

This code is used in chunk 42.

Defines:

Elimr, used in chunks 53a and 80b.

Uses BuscaNuevoPivote 45a, filtradopasos 58, and Matrix 15b.

53a *<Tres métodos de eliminación por columnas 49>+≡*

```

class ElimrG(Matrix):
    def __init__(self, data, rep=0, sust=[], repsust=0):
        """Devuelve una forma escalonada de Matrix(data)

        operando con las columnas. Si rep es no nulo, se muestran en
        Jupyter los pasos dados"""
        <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
        <Definición del método auxiliar BuscaNuevoPivote 45a>
        A = Elimr(data, 0, sust); r = 0; transformaciones = []; colExcluida = set()
        for i in range(1,A.m+1):
            p = BuscaNuevoPivote(i|A, sust);
            if p:
                r += 1
                <Intercambio de columnas para escalar 47a>
                <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>
                colExcluida.add(r)
        pasos = [ [], A.pasos[1]+[T(transformaciones)] ]
        pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
        self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
        self.rango = r
        super(self.__class__,self).__init__(A)
        self.__class__ = Matrix

        if isinstance (data, Matrix):
            self.cF, self.cC = data.cF, data.cC

```

This code is used in chunk 42.

Defines:

ElimrG, used in chunk 53b.

Uses BuscaNuevoPivote 45a, Elimr 52, filtradopasos 58, Matrix 15b, and T 38.

53b *<Tres métodos de eliminación por columnas 49>+≡*

```

class ElimrGJ(Matrix):
    def __init__(self, data, rep=0, sust=[], repsust=0):
        """Devuelve una forma escalonada reducida de Matrix(data)

        operando con las columnas. Si rep es no nulo, se muestran en
        Jupyter los pasos dados"""
        <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
        <Definición del método auxiliar BuscaNuevoPivote 45a>
        celim = lambda x: x < p
        A = ElimrG(data, 0, sust);
        r = 0; transformaciones = []; colExcluida = set()
        for i in range(1,A.m+1):
            p = BuscaNuevoPivote(i|A, sust);
            if p:
                r += 1

```

```

        <Uso del pivote para eliminar componentes con transformaciones Tipo I 45b>
        <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>
        colExcluida.add(p)
    transElimIzda = transformaciones
    r = 0; transformaciones = []; colExcluida = set()
    for i in range(1,A.m+1):
        p = BuscaNuevoPivote(i|A, sust);
        if p:
            r += 1
            <Normalización del pivote para que sea igual a uno 47b>
            <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c>
            colExcluida.add(p)
    pasos = [ [], A.pasos[1] + transElimIzda + [T(transformaciones)] ]
    pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
    self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
    self.rango = r
    super(self.__class__,self).__init__(A)
    self.__class__ = Matrix

    if isinstance (data, Matrix):
        self.cF, self.cC = data.cF, data.cC

```

This code is used in chunk 42.

Defines:

ElimrGJ, never used.

Uses BuscaNuevoPivote 45a, ElimrG 53a, filtradopasos 58, Matrix 15b, and T 38.

2.2.3 Variante de los métodos de eliminación evitando usar fracciones y operando con las filas (como en la mayoría de manuales de Álgebra Lineal)

Para esto no es necesario programar nuevos algoritmos, basta operar con las columnas de la matriz transpuesta y transponer el resultado. La única dificultad está en la representación de los pasos, pues queremos ver operaciones sobre las filas de la matriz, y no sobre las columnas de su transpuesta. Para ello escribimos la secuencia de transformaciones en el orden inverso (lo que requiere transponer algunas sub-secuencias de transformaciones elementales, $\tilde{\mathbf{t}}$); y las guardamos como elemento de la lista `pasos` (pues el primer elemento corresponde a las transformaciones de las filas). El siguientes recuadros muestran los tres métodos: Eliminación, Eliminación Gaussina y Eliminación Gauss-Jordan.

```

55 <Tres métodos de eliminación por filas 55>≡
    class ElimF(Matrix):
        def __init__(self, data, rep=0, sust=[], repsust=0):
            """Devuelve una forma pre-escalada de Matrix(data)

            operando con las filas (y evitando operar con fracciones).
            Si rep es no nulo, se muestran en Jupyter los pasos dados"""
            <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
            A = Elim(~Matrix(data), 0, sust);    r = A.rango
            pasos = [ list(reversed([ ~t for t in A.pasos[1] ])), [] ]
            pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
            self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
            self.rango = r
            super(self.__class__,self).__init__(~A)
            self.__class__ = Matrix

            if isinstance (data, Matrix):
                self.cF, self.cC = data.cF, data.cC

```

This definition is continued in chunks 56 and 57.

This code is used in chunk 42.

Defines:

ElimF, never used.

Uses Elim 49, filtradopasos 58, and Matrix 15b.

56 *<Tres métodos de eliminación por filas 55>+≡*

```

class ElimGF(Matrix):
    def __init__(self, data, rep=0, sust=[], repsust=0):
        """Devuelve una forma escalonada de Matrix(data)

        operando con las filas (y evitando operar con fracciones).
        Si rep es no nulo, se muestran en Jupyter los pasos dados"""
        <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
        A = ElimG(~Matrix(data), 0, sust);    r = A.rango
        pasos = [ list(reversed([ ~t for t in A.pasos[1] ])), [] ]
        pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
        self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
        self.rango = r
        super(self.__class__,self).__init__(~A)
        self.__class__ = Matrix

        if isinstance (data, Matrix):
            self.cF, self.cC = data.cF, data.cC

```

This code is used in chunk 42.

Defines:

ElimGF, used in chunks 60b and 96.

Uses ElimG 50, filtradopasos 58, and Matrix 15b.

57 *<Tres métodos de eliminación por filas 55>+≡*

```

class ElimGJF(Matrix):
    def __init__(self, data, rep=0, sust=[], repsust=0):
        """Devuelve una forma escalonada reducida de Matrix(data)

        operando con las columnas (y evitando operar con fracciones
        hasta el último momento). Si rep es no nulo, se muestran en
        Jupyter los pasos dados"""
        <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>
        A = ElimGJ(~Matrix(data), 0, sust); r = A.rango
        pasos = [ list(reversed([ ~t for t in A.pasos[1] ])), [] ]
        pasos = [ filtradopasos(pasos[i]) for i in (0,1) ]
        self.tex, self.pasos, self.TrF, self.TrC = texYpasos(data, rep, sust, repsust)
        self.rango = r
        super(self.__class__,self).__init__(~A)
        self.__class__ = Matrix

        if isinstance (data, Matrix):
            self.cF, self.cC = data.cF, data.cC

```

This code is used in chunk 42.
 Defines:
 ElimGJF, used in chunks 60a and 96.
 Uses ElimGJ 51, filtradopasos 58, and Matrix 15b.

En el proceso de eliminación, muchas transformaciones elementales realmente son identidades (sumar 0 veces otro vector y multiplicar un vector por 1).

A la hora de representar los pasos de eliminación, normalmente es mejor “filtrar” estos pasos innecesarios. Definimos un procedimiento general que quita de una lista de abreviaturas aquellas que son innecesarias en la representación. Si como argumento se le da una lista de abreviaturas, devuelve una lista filtrada. Si como argumento se le da una Transformación, devuelve una Transformación cuya lista de abreviaturas está filtrada.

58 *⟨Filtrado de secuencias de transformaciones 58⟩≡*

```
def filtradopasos(pasos):
    abv = pasos.t if isinstance(pasos,T) else pasos

    p = [T([j for j in T([abv[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(abv)) ]

    abv = [ t for t in p if t.t] # quitamos abreviaturas vacías de la lista

    return T(abv) if isinstance(pasos,T) else abv
```

This code is used in chunk 42.

Defines:

 filtradopasos, used in chunks 49–53, 55–57, 68, and 72–74.

Uses T 38.

2.3 Inversión de una matriz por eliminación Gaussiana

Para invertir una matriz basta aplicar la eliminación *Gauss-Jordan* sobre las columnas de una matriz cuadrada (`ElimrGJ`). Si la matriz resulta ser de rango completo (Si `R.rango == R.n`), entonces los pasos dados en la eliminación *Gauss-Jordan* aplicados sobre la matriz identidad del mismo orden nos dan la inversa.

El siguiente código obtiene la inversa de una matriz siguiendo el procedimiento anterior (pero demorando las operaciones con fracciones hasta el último momento, `ElimGJ`), y muestra los pasos dados hasta llegar a ella, o hasta llegar a una matriz singular

```
59 <Invirtiendo una matriz 59>≡
class InvMat(Matrix):
    def __init__(self, data, rep=0, sust=[]):
        """Devuelve la matriz inversa y los pasos dados sobre las columnas"""
        A = Matrix(data.subs(sust))
        if not A.es_cuadrada(): raise ValueError('Matrix no cuadrada')
        R = ElimGJ(A, 0, sust)
        self.pasos = R.pasos
        self.TrF = R.TrF
        self.TrC = R.TrC
        self.tex = rprElim( A.apila( I(A.n), 1 ) , self.pasos, [], sust)
        if rep:
            display(Math(self.tex))
        if R.es_singular(): raise ArithmeticError('Matrix singular')
        Inversa = I(A.n) & T(R.pasos[1])
        super(self.__class__,self).__init__(Inversa)
        self.__class__ = Matrix
```

This definition is continued in chunk 60.

This code is used in chunk 42.

Uses `apila` 95a, `ElimGJ` 51, `es_cuadrada` 93a, `es_singular` 94b 94c, `I` 102, `Matrix` 15b, `rprElim` 106c, and `T` 38.

El siguiente código obtiene la inversa de una matriz pero operando con las filas.

60a

```

<Invirtiendo una matriz 59>+=
class InvMatF(Matrix):
    def __init__(self, data, rep=0, sust=[]):
        """Devuelve la matriz inversa y los pasos dados sobre las filas"""
        A = Matrix(data.subs(sust))
        if not A.es_cuadrada(): raise ValueError('Matrix no cuadrada')
        M = ElimGJF(A, 0, sust)
        self.pasos = M.pasos
        self.TrF = M.TrF
        self.TrC = M.TrC
        self.tex = rprElim( A.concatena(I(A.m),1), self.pasos, [], sust )
        if rep:
            display(Math(self.tex))
        if M.es_singular(): raise ArithmeticError('Matrix singular')
        Inversa = T(M.pasos[0]) & I(A.n)
        super(self.__class__,self).__init__(Inversa)
        self.__class__ = Matrix

```

This code is used in chunk 42.

Uses `concatena` 7, `ElimGJF` 57, `es_cuadrada` 93a, `es_singular` 94b 94c, `I` 102, `Matrix` 15b, `rprElim` 106c, and `T` 38.

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

60b

```

<Invirtiendo una matriz 59>+=
class InvMatFC(Matrix):
    def __init__(self, data, rep=0, sust=[]):
        """Devuelve la matriz inversa y los pasos dados sobre las filas y columnas"""
        A = Matrix(data.subs(sust))
        if not A.es_cuadrada(): raise ValueError('Matrix no cuadrada')
        M = ElimGJ(ElimGF(A,0,sust),0,sust)
        self.pasos = M.pasos
        self.TrF = M.TrF
        self.TrC = M.TrC
        Aext = A.apila(I(A.n),1).concatena(I(A.n).apila(M0(A.n,A.n),1),1)
        self.tex = rprElim( Aext, self.pasos, [], sust )
        if rep:
            display(Math(self.tex))
        if M.es_singular(): raise ArithmeticError('Matrix singular')
        Inversa = ( I(A.n) & T(M.pasos[1]) ) * ( T(M.pasos[0]) & I(A.n) )
        super(self.__class__,self).__init__(Inversa)
        self.__class__ = Matrix

```

This code is used in chunk 42.

Uses `apila` 95a, `concatena` 7, `ElimGF` 56, `ElimGJ` 51, `es_cuadrada` 93a, `es_singular` 94b 94c, `I` 102, `M0` 101b, `Matrix` 15b, `rprElim` 106c, and `T` 38.

2.4 Resolución de un sistema de ecuaciones homogéneo

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

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

```

61  <Resolviendo un sistema homogéneo 61>≡
    class Homogenea:
        def __init__(self, data, rep=0, sust=[], repsust=0):
            """Resuelve un Sistema de Ecuaciones Lineales Homogéneo

            y muestra los pasos para encontrarlo"""

            A      = Matrix(data.subs(sust))
            <Cálculo de L y de una base del espacio nulo de A 62>

            self.sgen      = Sistema(base) if base else Sistema([ V0(A.n) ])
            self.determinado = (len(base) == 0)
            self.pasos      = L.pasos;
            self.TrF        = L.TrF
            self.TrC        = L.TrC

            if repsust:
                self.tex      = rprElim( A.apila( I(A.n) ,1 ) , self.pasos, [], sust)
            else:
                self.tex      = rprElim( A.apila( I(A.n) ,1 ) , self.pasos)

            self.enulo      = SubEspacio(self.sgen)

            if rep:
                display(Math(self.tex))

    <Métodos de representación de la clase Homogenea 109a>

```

This code is used in chunk 42.

Defines:

Homogenea, never used.

Uses `apila` 95a, `I` 102, `Matrix` 15b, `rprElim` 106c, `Sistema` 9, `SubEspacio` 76, and `V0` 101a.

La base la constituyen los vectores \mathbf{v} de E que corresponden a los vectores nulos de L :

62 $\langle \text{Cálculo de } L \text{ y de una base del espacio nulo de } A \text{ 62} \rangle \equiv$

```
L      = Elim( A, 0, sust )
E      = I(A.n) & T(L.pasos[1])
base   = [ v.subs(sust) for j, v in enumerate(E, 1) if (L[j].es_nulo(sust) ]
```

This code is used in chunk 61.
Uses [Elim 49](#), [I 102](#), and [T 38](#).

2.5 Resolución de un sistema de ecuaciones

63a *<Texto de ayuda de la clase SEL 63a>≡*
 """Resuelve un Sistema de Ecuaciones Lineales
 mediante eliminación por columnas en la matriz ampliada y muestra
 los pasos dados"""
 This code is used in chunk 63b.
 Uses Sistema 9.

63b *<Resolviendo un Sistema de Ecuaciones Lineales 63b>≡*
 class SEL:
 def __init__(self, A, b, rep=0, sust=[], repsust=0):
<Texto de ayuda de la clase SEL 63a>
 A = Matrix(A.subs(sust))
 MA = A.concatena(Matrix([-b])).apila(I(A.n+1))
 MA.cfil({A.m, A.m+A.n}).ccol({A.n})

 L = Elim(slice(1,A.m)|MA, 0, sust)

<Aplicamos los pasos de eliminación sobre la matriz ampliada y obtenemos la solución 63c>
<Métodos de representación de la clase SEL 109b>

 This code is used in chunk 42.
 Defines:
 SEL, used in chunk 83a.
 Uses apila 95a, concatena 7, Elim 49, I 102, and Matrix 15b.

Aplicamos los pasos sobre toda la matriz ampliada (más bien “super ampliada”, pues tiene una matriz identidad por debajo). Si el último elemento de la última columna es

63c *<Aplicamos los pasos de eliminación sobre la matriz ampliada y obtenemos la solución 63c>≡*
 EA = (Matrix(MA) & T(L.pasos[1])).subs(sust)
 Normaliza = T([]) if (0|EA|0)==1 else T([(fracc(1,0|EA|0), EA.n)])
 EA = (EA & Normaliza).subs(sust)

 K = slice(1,A.m)|EA|slice(1,A.n);
 E = slice(A.m+1,A.m+A.n)|EA|slice(1,A.n)
 S = slice(A.m+1,A.m+A.n)|EA|slice(A.n+1,None)

 self.base = Sistema([v for j, v in enumerate(E,1) if (K|j).es_nulo(sust)])
 self.sgen = self.base if self.base else Sistema([V0(A.n)])
 self.determinado = (len(self.base) == 0)

 if (L|0).no_es_nulo():


```

        self.solP = set()
        self.eafin = set()
    else:
        self.solP = S|1
        self.eafin = EAfin(self.sgen, self.solP, 1)

    self.pasos = [[], L.pasos[1]+[Normaliza] ] if Normaliza.t else [[], L.pasos[1]]
    self.TrF = T(self.pasos[0])
    self.TrC = T(self.pasos[1])

    if repsust:
        self.tex = rprElim( MA, self.pasos, [], sust )
    else:
        self.tex = rprElim( MA, self.pasos)

    if rep:
        display(Math(self.tex))

```

This code is used in chunk 63b.
 Uses EAfin 80b, fracc 43, Matrix 15b, rprElim 106c, Sistema 9, T 38, and VO 101a.

2.6 Cálculo del determinante por eliminación Gaussiana

65a *<Texto de ayuda de la clase Determinante 65a>*≡

```

    """Calcula el determinante

    mediante eliminación Gaussiana por columnas y muestra los pasos dados"""

```

This code is used in chunk 65b.
Uses `determinante` 98b.

65b *<Calculando el determinante 65b>*≡

```

class Determinante:
    def __init__(self, data, disp=0, sust=[]):
        <Texto de ayuda de la clase Determinante 65a>
        <Cálculo del determinante y representación de los pasos en Jupyter 66a>

        A = Matrix(data.subs(sust))

        if not A.es_cuadrada(): raise ValueError('Matrix no cuadrada')

        self.tex, self.valor, self.pasos = calculoDet( A.subs(sust) )

        self.TrF = T(self.pasos[0])
        self.TrC = T(self.pasos[1])

        if disp:
            display(Math(self.tex))

<Métodos de representación de la clase Determinante 66b>

```

This code is used in chunk 42.

Defines:

`Determinante`, used in chunk 98b.

Uses `calculoDet` 66a, `es_cuadrada` 93a, `Matrix` 15b, and `T` 38.

66a *⟨Cálculo del determinante y representación de los pasos en Jupyter 66a⟩≡*

```
def calculoDet(A, sust=[]):
    producto = lambda x: 1 if not x else x[0] * producto(x[1:])

    pc = (A.L(0,sust).pasos[1])
    ME = A.extDiag(I(1),1)
    tex = latex(ME)
    pasos = [[],[]]

    for i in range(len(pc)):
        S = [ tr for tr in filter( lambda x: len(x)==2, T(pc[i]).t ) ]
        m = [-1 if isinstance(tr,set) else tr[0] for tr in S]
        pf = [T([ ( fracc(1, producto(m)) , A.n+1 ) ]) if producto(m)!=1 else T([])]

        tex = rprElimFyC(ME,[pf,[pc[i]]],tex)

        T(pf) & ME & T(pc[i])

        pasos[0] = pf + pasos[0]
        pasos[1] = pasos[1] + [pc[i]]

    Det = simplify( producto( ME.diag() ) )

    return [tex, Det, pasos]
```

This code is used in chunk 65b.

Defines:

`calculoDet`, used in chunk 65b.

Uses `extDiag` 95b, `fracc` 43, `I` 102, `rprElimFyC` 107a, and `T` 38.

66b *⟨Métodos de representación de la clase Determinante 66b⟩≡*

```
def __repr__(self):
    """ Muestra un Sistema en su representación Python """
    return 'Valor del determinante: ' + repr (self.valor)

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 latex(self.valor)
```

This code is used in chunk 65b.

Uses `determinante` 98b and `Sistema` 9.

2.7 Diagonalizando en bloques triangulares una matriz cuadrada por semejanza (Dentado)

```

68 <Diagonalizando una matriz por bloques triangulares (por Semejanza) 68>≡
class DiagonalizaS(Matrix):
    def __init__(self, A, espectro, Rep=0, sust=[]):
        <Texto de ayuda para la clase DiagonalizaS 69a>
        <Definición del método auxiliar BuscaNuevoPivote 45a>
        if not A.es_cuadrada: raise ValueError('Matrix no es cuadrada')
        if not isinstance(espectro, list):
            raise ValueError('espectro no es una lista')
        if len(espectro)!=A.n:
            raise ValueError('número inadecuado de autovalores en la lista espectro')
        if not all([(A-l*I(A.n)).es_singular() for l in espectro]):
            raise ValueError('alguno de los valores proporcionados no es autovalor')

        D = Matrix(A.subs(sust))
        espectro = [sympy.S(l).subs(sust) for l in espectro]

        S = I(D.n)
        Tex = latex( D.apila(S,1) )
        pasosPrevios = [[],[]]
        selecc = list(range(1,D.n+1))
        for lamda in espectro:
            m = selecc[-1]
            <Restamos λI 69c>
            TrCol = filtradopasos(ElimG(selecc|D|selecc,0,sust).pasos[1])
            <Aplicación de las transformaciones y sus inversas "espejo" 69b>
            if m < D.n:
                transf = []; colExcluida = set(selecc)
                for i in range(m,D.n+1):
                    p = BuscaNuevoPivote(i|D, sust);
                    if p:
                        TrCol = filtradopasos([ T([(-fracc(i|D|m, i|D|p), p, m))] )])
                        <Aplicación de las transformaciones y sus inversas "espejo" 69b>
                        colExcluida.add(p)
                <Sumamos λI 69d>

            selecc.pop()

        if Rep:
            display(Math(Tex))

        self.espectro = espectro[::-1]
        self.tex = Tex
        self.S = S
        self.TrF = T(pasosPrevios[0])
        self.TrC = T(pasosPrevios[1])
        self.pasos = pasosPrevios
        super(self.__class__,self).__init__(D)
        self.__class__ = Matrix

```

This code is used in chunk 42.

Defines:

DiagonalizaS, used in chunk 99b.

Uses apila 95a, BuscaNuevoPivote 45a, ElimG 50, es_cuadrada 93a, es_singular 94b 94c, filtradopasos 58, fracc 43, I 102, Matrix 15b, and T 38.

69a $\langle \text{Texto de ayuda para la clase DiagonalizaS } 69a \rangle \equiv$
 """Diagonaliza por bloques triangulares una Matrix cuadrada

Encuentra una matriz diagonal semejante mediante transformaciones de sus columnas y las correspondientes transformaciones inversas `espejo` de las filas. Requiere una lista de autovalores (espectro), que deben aparecer en dicha lista tantas veces como sus respectivas multiplicidades algebraicas. Los autovalores aparecen en la diagonal principal de la matriz diagonal. El atributo `S` de dicha matriz diagonal es una matriz cuyas columnas son autovectores de los correspondientes autovalores.

"""

This code is used in chunks 68 and 99b.
 Uses `espejo` 36b and `Matrix` 15b.

69b $\langle \text{Aplicación de las transformaciones y sus inversas "espejo" } 69b \rangle \equiv$
`pasos` = [[], `TrCol`]
`pasosPrevios[1]` = `pasosPrevios[1]` + `pasos[1]`

`Tex` = `rprElim`(`D.apila`(`S`,1), `pasos`, `Tex`, `sust`) if `TrCol` else `Tex`
`D` = (`D` & `T`(`pasos[1]`)).`subs`(`sust`)
`S` = (`S` & `T`(`pasos[1]`)).`subs`(`sust`)

`pasos` = [[`T`(`pasos[1]`).`espejo`()**(-1)], []]
`pasosPrevios[0]` = `pasos[0]` + `pasosPrevios[0]`

`Tex` = `rprElim`(`D.apila`(`S`,1), `pasos`, `Tex`, `sust`) if `TrCol` else `Tex`
`D` = (`T`(`pasos[0]`) & `D`).`subs`(`sust`)

This code is used in chunk 68.
 Uses `apila` 95a, `espejo` 36b, `rprElim` 106c, and `T` 38.

69c $\langle \text{Restamos } \lambda I \text{ } 69c \rangle \equiv$
`D` = (`D`-(`lamda`*`I`(`D.n`))).`subs`(`sust`)
`Tex` += '`\xrightarrow`' + `latex`(`lamda`) + '`\mathbf{I}`' + `latex`((`D.apila`(`S`,1)).`subs`(`sust`))

This code is used in chunk 68.
 Uses `apila` 95a and `I` 102.

69d $\langle \text{Sumamos } \lambda I \text{ } 69d \rangle \equiv$
`D` = (`D`+(`lamda`*`I`(`D.n`))).`subs`(`sust`)
`Tex` += '`\xrightarrow`' + `latex`(`lamda`) + '`\mathbf{I}`' + `latex`((`D.apila`(`S`,1)).`subs`(`sust`))

This code is used in chunk 68.
 Uses `apila` 95a and `I` 102.

2.7.1 Diagonalización ortogonal de una matriz simétrica

70 *<Diagonalizando Ortogonalmente una matriz simétrica 70>≡*

```

class Diagonaliza0(Matrix):
    def __init__(self, A, espectro, Rep=0, sust=[]):
        <Texto de ayuda para la clase Diagonaliza0 71a>
        <Método auxiliar para creación de una base ortonormal donde q es el último vector 71b>

        if not A.es_simetrica:
            raise ValueError('La matriz no es simétrica')
        if not isinstance(espectro, list):
            raise ValueError('espectro no es una lista')
        if len(espectro)!=A.n:
            raise ValueError('número inadecuado de autovalores en la lista espectro')
        if not all([(A-l*I(A.n)).es_singular() for l in espectro]):
            raise ValueError('alguno de los valores proporcionados no es autovalor')

        D      = Matrix(A.subs(sust))
        S      = I(A.n)
        espectro = [sympy.S(l).subs(sust) for l in espectro];
        selecc  = list(range(1,D.n+1))
        for l in espectro:
            D = (D - l*I(D.n)).subs(sust)
            TrCol = ElimG(selecc|D|selecc,0,sust).pasos[1]
            D = (D + l*I(D.n)).subs(sust)
            k      = len(selecc)
            nmenosk = (D.n)-k
            selecc.pop()

            q = ( (I(k) & T(TrCol)).subs(sust) )|0
            q = (sympy.sqrt(q*q)) * q

            Q = BaseOrtNor(q).concatena(M0(k,nmenosk)).apila( \
                M0(nmenosk,k).concatena(I(nmenosk))) if nmenosk else BaseOrtNor(q)

            S = (S*Q).subs(sust)
            D = (~Q*D*Q).subs(sust)

        self.Q = S
        self.espectro = espectro[::-1]
        super(self.__class__,self).__init__(D)
        self.__class__ = Matrix

```

This code is used in chunk 42.

Defines:

Diagonaliza0, used in chunk 99c.

Uses apila 95a, concatena 7, ElimG 50, es_simetrica 93a, es_singular 94b 94c, I 102, M0 101b, Matrix 15b, and T 38.

71a *<Texto de ayuda para la clase DiagonalizaO 71a>≡*
 """ Diagonaliza ortogonalmente una *Matrix* simétrica

Encuentra una matriz diagonal por semejanza empleando una matriz ortogonal Q a la derecha y su inversa (transpuesta) por la izquierda. Requiere una lista de autovalores (espectro), que deben aparecer tantas veces como sus respectivas multiplicidades algebraicas. Los autovalores aparecen en la diagonal principal de la matriz diagonal. El atributo Q de la matriz diagonal es la matriz ortogonal cuyas columnas son autovectores de los correspondientes autovalores. """

This code is used in chunks 70 and 99c.
 Uses *Matrix* 15b.

Creamos una base ortonormal de \mathbb{R}^n con $\mathbf{q} \in \mathbb{R}^n$ como último vector (Corolario 2.1 de la lección)

71b *<Método auxiliar para creación de una base ortonormal donde q es el último vector 71b>≡*
 def BaseOrtNor(q):
 "Crea una base ortonormal cuyo último vector es 'q'"
 if not isinstance(q, Vector): raise ValueError('El argumento debe ser un Vector')
 M = *Matrix*([q]).concatena(I(q.n)).GS()
 l = [j for j, v in enumerate(M, 1) if v.no_es_nulo()]
 l = l[1:len(l)]+[l[0]]
 return (M[l]).normalizada()

This code is used in chunk 70.
 Uses *concatena* 7, *I* 102, *Matrix* 15b, *normalizada* 94a, and *Vector* 12.

2.8 Diagonalización por congruencia

71c *<Texto de ayuda para la clase DiagonalizaC 71c>≡*
 """ Diagonaliza por congruencia una *Matrix* simétrica (evitando dividir)

Encuentra una matriz diagonal por congruencia empleando una matriz B invertible (y entera si es posible) por la derecha y su transpuesta por la izquierda. No emplea los autovalores. En general los elementos en la diagonal principal no son autovalores, pero hay tantos elementos positivos en la diagonal como autovalores positivos, tantos negativos como autovalores negativos, y tantos ceros como autovalores nulos. """

This code is used in chunks 72 and 100b.
 Uses *Matrix* 15b.


```

72  <Diagonalizando una matriz por Congruencia 72>≡
    class DiagonalizaC(Matrix):
        def __init__(self, data, Rep=0, sust=[]):
            <Texto de ayuda para la clase DiagonalizaC 71c>
            <Definición del método auxiliar BuscaPrimerNoNuloEnLaDiagonal 73a>
            A = Matrix(data.subs(sust)); colExcluida = set()
            celim = lambda x: x > p; pasosPrevios = [ [], [] ]
            for i in range(1,A.n):
                p = BuscaPrimerNoNuloEnLaDiagonal(A,i)
                j = [k for k,col in enumerate(A[slice(i,None),i]) if (i|col and not k|col)]
                if not (i|A[i]):
                    if p:
                        Tr = T( {i, p} )
                        p = i
                        <Aplicación de las transformaciones a las columnas y a las filas 73b>
                    elif j:
                        Tr = T( (1, j[0], i) )
                        p = i
                        <Aplicación de las transformaciones a las columnas y a las filas 73b>
                if p:
                    <Uso del pivote para eliminar componentes evitando dividir 46>
                    <Aplicación de las transformaciones a las columnas y a las filas 73b>
                colExcluida.add(i)

            self.pasos = pasosPrevios
            self.tex = rprElimCF(Matrix(data.subs(sust)), self.pasos, [], sust)
            self.TrF = filtradopasos(T(self.pasos[0]))
            self.TrC = filtradopasos(T(self.pasos[1]))
            self.B = I(A.n) & self.TrC

            if Rep:
                display(Math(self.tex))

            super(self.__class__,self).__init__(A)
            self.__class__ = Matrix

```

This definition is continued in chunk 74.

This code is used in chunk 42.

Defines:

DiagonalizaC, used in chunk 100b.

Uses BuscaPrimerNoNuloEnLaDiagonal 73a, filtradopasos 58, I 102, Matrix 15b, rprElimCF 107b, and T 38.

73a

⟨Definición del método auxiliar BuscaPrimerNoNuloEnLaDiagonal 73a⟩≡

```
def BuscaPrimerNoNuloEnLaDiagonal(self, i=1):
    """Indica el índice de la primera componente no nula de a diagonal
    desde de la posición i en adelante. Si son todas nulas devuelve 0"""
    d = (slice(i,None)|self|slice(i,None)).diag().sis()
    return next((pos for pos, x in enumerate(d) if x), -i) + i
```

This code is used in chunks 72 and 74.

Defines:

BuscaPrimerNoNuloEnLaDiagonal, used in chunks 72 and 74.

73b

⟨Aplicación de las transformaciones a las columnas y a las filas 73b⟩≡

```
pasos = [ [], filtradopasos([Tr]) ]
pasosPrevios[1] = pasosPrevios[1] + pasos[1]
A = (A & T(pasos[1])).subs(sust)

pasos = [ filtradopasos([~Tr]) , [] ]
pasosPrevios[0] = pasos[0] + pasosPrevios[0]
A = (T(pasos[0]) & A).subs(sust)
```

This code is used in chunks 72 and 74.

Uses filtradopasos 58 and T 38.

73c

⟨Texto de ayuda para la clase DiagonalizaCr 73c⟩≡

```
""" Diagonaliza por congruencia una Matrix simétrica
```

```
Encuentra una matriz diagonal congruente multiplicando por una matriz
invertible B a la derecha y por la transpuesta de B por la izquierda.
No requiere conocer los autovalores. En general los elementos en la
diagonal principal de la matriz diagonal no son autovalores, pero hay
tantos elementos positivos en la diagonal como autovalores positivos
(incluyendo la multiplicidad de cada uno), tantos negativos como
autovalores negativos (incluyendo la multiplicidad de cada uno), y tantos
ceros como la multiplicidad algebraica del autovalor cero. """
```

This code is used in chunks 74 and 100c.

Uses Matrix 15b.

```

74  <Diagonalizando una matriz por Congruencia 72>+≡
    class DiagonalizaCr(Matrix):
        def __init__(self, data, Rep=0, sust=[]):
            <Texto de ayuda para la clase DiagonalizaCr 73c>
            <Definición del método auxiliar BuscaPrimerNoNuloEnLaDiagonal 73a>
            A = Matrix(data.subs(sust)); colExcluida = set()
            celim = lambda x: x > p; pasosPrevios = [ [], [] ]
            for i in range(1,A.n):
                p = BuscaPrimerNoNuloEnLaDiagonal(A,i)
                j = [k for k,col in enumerate(A[slice(i,None),i]) if (i|col and not k|col)]
                if not (i|A|i):
                    if p:
                        Tr = T( {i, p} )
                        p = i
                        <Aplicación de las transformaciones a las columnas y a las filas 73b>
                    elif j:
                        Tr = T( (1, j[0], i) )
                        p = i
                        <Aplicación de las transformaciones a las columnas y a las filas 73b>
                if p:
                    <Uso del pivote para eliminar componentes con trasformaciones Tipo I 45b>
                    <Aplicación de las transformaciones a las columnas y a las filas 73b>
                colExcluida.add(i)

            self.pasos = pasosPrevios
            self.tex = rprElimCF(Matrix(data.subs(sust)), self.pasos, [], sust)
            self.TrF = filtradopasos(T(self.pasos[0]))
            self.TrC = filtradopasos(T(self.pasos[1]))
            self.B = I(A.n) & self.TrC

            if Rep:
                display(Math(self.tex))

            super(self.__class__,self).__init__(A)
            self.__class__ = Matrix

```

This code is used in chunk 42.

Defines:

DiagonalizaCr, used in chunk 100c.

Uses BuscaPrimerNoNuloEnLaDiagonal 73a, filtradopasos 58, I 102, Matrix 15b, rprElimCF 107b, and T 38.

Capítulo 3

Las clases SubEspacio y EAfin

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

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

La clase `SubEspacio` se puede instanciar tanto con un `Sistema` de `Vectores` como con una `Matrix`.

En el primer caso, dado un `Sistema` de vectores, por ejemplo

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

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

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

donde el vector \mathbf{p} es el vector de parámetros. En el segundo caso, dada una `Matrix`, por ejemplo

$$\mathbf{M} = \begin{bmatrix} -3 & 0 & 2 \\ 6 & 0 & -4 \end{bmatrix},$$

`SubEspacio(M)` corresponde al conjunto de `Vectores` que son solución al sistema de ecuaciones $\mathbf{Mv} = \mathbf{0}$; y que se puede representar con el sistema de ecuaciones *cartesianas*:

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

En ambos ejemplos corresponden al mismo subespacio de \mathbb{R}^3 ; y, de hecho, la librería muestra ambos tipos de representación para cada `SubEspacio`: las ecuaciones paramétricas a la izquierda y las cartesianas a la derecha.

$$\left\{ \mathbf{v} \in \mathbb{R}^3 \mid \exists \mathbf{p} \in \mathbb{R}^2, \mathbf{v} = \begin{bmatrix} 0 & 2 \\ 1 & 0 \\ 0 & 3 \end{bmatrix} \mathbf{p} \right\} = \{ \mathbf{v} \in \mathbb{R}^3 \mid [-3 \quad 0 \quad 2] \mathbf{v} = \mathbf{0} \}$$

`SubEspacio` tiene varios atributos.

- `dim`: dimensión del subespacio. En el ejemplo `dim=2`.
- `Rm`: indica el espacio vectorial \mathbb{R}^m al que pertenece `SubEspacio(S)`. En el ejemplo anterior `Rm=3` puesto que es un subespacio de \mathbb{R}^3 .
- `base`: una base del subespacio (un `Sistema` de `Vectores` de `Rm`). Cuando `dim==0` base es un `Sistema` vacío.

- **sgen**: Un **Sistema** de **Vectores** generador del subespacio. En particular será el sistema de vectores correspondiente a la **Matrix** de coeficientes empleada en la representación con ecuaciones paramétricas. En el ejemplo Cuando $\text{dim}=0$, **sgen** contiene un vector nulo de R_m componentes.

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

- **cart**: **Matrix** de coeficientes empleada en la representación con las ecuaciones cartesianas. En el ejemplo

$$[-3 \ 0 \ 2].$$

La implementación requiere encontrar un **Sistema** base del **SubEspacio** columna de una **Matrix** **A**. Lo haremos pre-escalando una **A** con **Elim** (así evitamos las fracciones en la medida de lo posible). También necesitaremos encontrar un sistema generador del un espacio nulo de **A**. Lo haremos con el método auxiliar **SGenENulo**.

```
76 <Iniciación de la clase SubEspacio 76>≡
def __init__(self,data):
    """Inicializa un SubEspacio de Rn"""
    <Método auxiliar SGenENulo que Encuentra un sistema generador del Espacio Nulo de A 77a>
    if not isinstance(data, (Sistema, Matrix)):
        raise ValueError(' Argumento debe ser un Sistema o Matrix ')
    if isinstance(data, Sistema):
        A = Matrix(data)
        self.base = Sistema([A[j] for j,v in enumerate(Elim(A),1) if v.no_es_nulo()])
        self.dim = len(self.base)
        self.sgen = self.base if self.base else Sistema([ V0(A.m) ])
        self.cart = ~Matrix(SGenENulo(~A))
        self.Rn = A.m
    if isinstance(data, Matrix):
        A = data
        self.sgen = SGenENulo(A)
        self.dim = 0 if self.sgen.es_nulo() else len(self.sgen)
        self.base = self.sgen if self.dim else Sistema([])
        self.cart = ~Matrix(SGenENulo(~Matrix(self.sgen)))
        self.Rn = A.n
```

This code is used in chunk 80a.

Defines:

SubEspacio, used in chunks 61, 77–83, 97a, and 111.

Uses **Elim** 49, **Matrix** 15b, **Sistema** 9, and **V0** 101a.

Una base del espacio nulo está formada por los vectores de **E** correspondientes a los vectores nulos de **K**

77a *⟨Método auxiliar SGenENulo que Encuentra un sistema generador del Espacio Nulo de A 77a⟩*≡

```
def SGenENulo(A):
    """Encuentra un sistema generador del Espacio Nulo de A"""
    K = Elim(A);    E = I(A.n) & T(K.pasos[1])
    S = Sistema([ v for j, v in enumerate(E,1) if (K|j).es_nulo() ])
    return S if S else Sistema([VO(A.n)])
```

This code is used in chunk 76.
Uses Elim 49, I 102, Sistema 9, T 38, and VO 101a.

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

A.contenido_en(B)

nos dirá si es cierto que A es un **SubEspacio** de B (fíjese que como “**contenido_en**” no es un “Método Mágico” de Python, se debe invocar escribiendo **A.contenido_en()**, donde A es un **SubEspacio**).

Para comprobar si un **SubEspacio** A está contenido en un **SubEspacio** B, basta verificar si todos los vectores del sistema generador de A son solución de las ecuaciones cartesianas de B. Si B es un **EAFin**, entonces B.v debe ser nulo y A debe estar contenido en B.S.

77b *⟨Métodos de la clase SubEspacio 77b⟩*≡

```
def contenido_en(self, other):
    """Indica si este SubEspacio está contenido en other"""
    self.verificacion(other)
    if isinstance(other, SubEspacio):
        return all ([ (other.cart*v).es_nulo() for v in self.sgen ])
    elif isinstance(other, EAFin):
        return other.v.es_nulo() and self.contenido_en(other.S)
```

This definition is continued in chunks 78 and 79.
This code is used in chunk 80a.
Uses EAFin 80b and SubEspacio 76.

También definimos dos métodos (mágicos) que nos indican

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

Así podemos usar las siguientes expresiones booleanas

A == B y A != B

78a

```

<Métodos de la clase SubEspacio 77b>+≡
def __eq__(self, other):
    """Indica si un subespacio de Rn es igual a otro"""
    self.verificacion(other)
    return self.contenido_en(other) and other.contenido_en(self)

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

```

This code is used in chunk 80a.

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

78b

```

<Métodos de la clase SubEspacio 77b>+≡
def verificacion(self, other):
    if not isinstance(other, (SubEspacio, EAfin)) or not self.Rn == other.Rn:
        raise \
            ValueError('Ambos argumentos deben ser subconjuntos de en un mismo espacio')

```

This code is used in chunk 80a.

Uses `EAfin` 80b and `SubEspacio` 76.

También definimos un método que nos devuelva la suma de dos `SubEspacios` de \mathbb{R}^m : $A + B$. Para ello basta concatenar los `Sistemas` en uno solo.

78c

```

<Métodos de la clase SubEspacio 77b>+≡
def __add__(self, other):
    """Devuelve la suma de subespacios de Rn"""
    self.verificacion(other)
    return SubEspacio(Sistema(self.sgen.concatena(other.sgen)))

```

This code is used in chunk 80a.

Uses `concatena` 7, `Sistema` 9, and `SubEspacio` 76.

y definimos otro método que nos devuelva la intersección: $A \& B$. Para ello apilamos las matrices de las ecuaciones cartesianas en una sola `Matrix` y obtenemos el `SubEspacio` correspondiente. Si `other` es un `EAfin` llamamos al método de la intersección entre un `EAfin` y un `SubEspacio`.

79a *<Métodos de la clase SubEspacio 77b>+≡*

```
def __and__(self, other):
    """Devuelve la intersección de subespacios"""
    self.verificacion(other)
    if isinstance(other, SubEspacio):
        return SubEspacio( self.cart.apila(other.cart) )
    elif isinstance(other, EAFin):
        return other & self
```

This code is used in chunk 80a.
Uses [apila 95a](#), [EAFin 80b](#), and [SubEspacio 76](#).

Con $\sim A$ obtendremos el complemento ortogonal del [SubEspacio A](#), es decir, el [Sistema](#) formado por las filas (las columnas de la transpuesta) de `self.cart`

79b *<Métodos de la clase SubEspacio 77b>+≡*

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

This code is used in chunk 80a.
Uses [Sistema 9](#) and [SubEspacio 76](#).

y por último definimos un método que nos indique si un [Vector x](#) pertenece a un [SubEspacio A](#), es decir, que indique si es cierta o no la siguiente expresión booleana

$$x \text{ in } A$$

Basta verificar que el vector es solución del sistema de ecuaciones cartesianas.

79c *<Métodos de la clase SubEspacio 77b>+≡*

```
def __contains__(self, other):
    """Indica si un Vector pertenece a un SubEspacio"""
    if not isinstance(other, Vector) or other.n != self.cart.n:
        raise ValueError\
            ('El Vector no tiene el número adecuado de componentes')
    return (self.cart*other == V0(self.cart.m))
```

This code is used in chunk 80a.
Uses [SubEspacio 76](#), [V0 101a](#), and [Vector 12](#).

80a \langle La clase *SubEspacio* 80a $\rangle \equiv$

```
class SubEspacio:
     $\langle$ Inicialización de la clase SubEspacio 76 $\rangle$ 
     $\langle$ Métodos de la clase SubEspacio 77b $\rangle$ 
     $\langle$ Métodos de representación de la clase SubEspacio 111 $\rangle$ 
```

This code is used in chunk 42.
Uses *SubEspacio* 76.

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

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

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

Cuando $\mathbf{v} \in \mathcal{S}$, el espacio afín es un subespacio (que por tanto contiene al vector nulo). Así que si $\mathbf{v} \in \mathcal{S}$ en el atributo *v* guardaremos el vector nulo. Así, si consideramos el sistema “ampliado” que contiene los vectores del sistema generador de *S* primero, y *v* como último vector *v*, y aplicamos el método de eliminación de izquierda a derecha; el último vector tras la eliminación pertenece al espacio afín, y será cero si $\mathbf{v} \in \mathcal{S}$. Si *vi* es cero (su valor por defecto), en *selv.v* se guardará el vector resultante tras la eliminación, en caso contrario se guardará el vector indicado (*vi*) como argumento.

80b \langle Inicialización de la clase *EAfin* 80b $\rangle \equiv$

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

This code is used in chunk 81a.
Defines:
 EAfin, used in chunks 63c, 77–79, 81–83, 110, and 111.
Uses *concatena* 7, *Elimr* 52, *Sistema* 9, *SubEspacio* 76, and *Vector* 12.

81a *<La clase EAfin 81a>≡*

```
class EAfin:
    <Inicialización de la clase EAfin 80b>
    <Métodos de la clase EAfin 81b>
    <Métodos de representación de la clase EAfin 110>
```

This code is used in chunk 42.
Uses EAfin 80b.

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

81b *<Métodos de la clase EAfin 81b>≡*

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

This definition is continued in chunks 81-83.
This code is used in chunk 81a.
Uses EAfin 80b and Vector 12.

81c *<Métodos de la clase EAfin 81b>+≡*

```
def contenido_en(self, other):
    """Indica si este EAfin está contenido en other"""
    self.verificacion(other)
    if isinstance(other, SubEspacio):
        return self.v in other and self.S.contenido_en(other)
    elif isinstance(other, EAfin):
        return self.v in other and self.S.contenido_en(other.S)
```

This code is used in chunk 81a.
Uses EAfin 80b and SubEspacio 76.

82a

```

<Métodos de la clase EAfin 81b>+≡
def __eq__(self, other):
    """Indica si un EAfin de Rn es igual a other"""
    self.verificacion(other)
    return self.contenido_en(other) and other.contenido_en(self)

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

```

This code is used in chunk 81a.
 Uses EAfin 80b.

82b

```

<Métodos de la clase EAfin 81b>+≡
def verificacion(self, other):
    if not isinstance(other, (SubEspacio, EAfin)) or not self.Rn == other.Rn:
        raise \
            ValueError('Ambos argumentos deben ser subconjuntos de en un mismo espacio')

```

This code is used in chunk 81a.
 Uses EAfin 80b and SubEspacio 76.

La intersección es el conjunto de soluciones a ambos sistemas de ecuaciones cartesianas. El modo más sencillo es unificar ambos sistemas en uno solo: apilando las matrices de coeficientes por un lado y concatenando los vectores del lado derecho por el otro.

83a

```

<Métodos de la clase EAfin 81b>+≡
def __and__(self, other):
    """Devuelve la intersección de este EAfin con other"""
    self.verificacion(other)
    if isinstance(other, EAfin):
        M = self.S.cart.apila( other.S.cart )
        w = (self.S.cart*self.v).concatena( other.S.cart*other.v )
    elif isinstance(other, SubEspacio):
        M = self.S.cart.apila( other.cart )
        w = (self.S.cart*self.v).concatena( V0(other.cart.m) )
    try:
        S=SEL(M,w)
    except:
        print('Intersección vacía')
        return Sistema([])
    else:
        return S.eafin

```

This code is used in chunk 81a.

Uses `apila` 95a, `concatena` 7, `EAfin` 80b, `SEL` 63b, `Sistema` 9, `SubEspacio` 76, and `V0` 101a.

Con $\sim A$ obtendremos el mayor `SubEspacio` perpendicular a `A`.

83b

```

<Métodos de la clase EAfin 81b>+≡
def __invert__(self):
    """Devuelve el mayor SubEspacio perpendicular a self"""
    return SubEspacio( Sistema( ~(self.S.cart) ) )

```

This code is used in chunk 81a.

Uses `Sistema` 9 and `SubEspacio` 76.

Capítulo 4

Otros trozos de código

4.1 Métodos de representación para el entorno Jupyter

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

84a *⟨Método html general 84a⟩*≡

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

This code is used in chunk 42.

El método `latex` general, intentará llamar al método `latex` particular del objeto que se quiere representar (para todos los objetos de esta librería se define su método de representación). Si el objeto no tiene definido el método `latex`, entonces se emplea el método `latex` del módulo **Sympy** (`sympy.latex()`). Además, se pide que el en tal caso, antes de representar el objeto se simplifiquen las expresiones si ello es posible. Así,

```
Vector([ sympy.Rational(1.5, 3), fracc(2.4, 1.2), fracc(2, sympy.sqrt(2)) ], rpr='fila')
```

será representado como

$$\left(\frac{1}{2}, 2, \sqrt{2}\right)$$

84b *⟨Método latex general 84b⟩*≡

```
def latex(a):  
    try:  
        return a.latex()  
    except:  
        return sympy.latex(simplify(a))
```

This code is used in chunk 42.

Con el método `simplifica` se devuelven las expresiones simplificadas. Si el objeto es una tupla, lista o `Sistema`, se devuelve un objeto del mismo tipo, pero cuyos elementos han sido simplificados. Si `self` es otro tipo de objeto, entonces se “sympyfica” con `sympy.simplify(self)`, es decir, se transforma un objeto del módulo `Sympy`, para así poder ser simplificado con el método `sympy.simplify()`.

85a

```

<Simplificación de expresiones simbólicas 85a>≡
def simplifica(self):
    """Devuelve las expresiones simplificadas"""
    if isinstance(self, (list, tuple, Sistema)):
        return type(self)([ simplifica(e) for e in self ])
    else:
        return (sympy.simplify(self)).simplify()

def simplify(self):
    """Devuelve las expresiones simplificadas"""
    if isinstance(self, (list, tuple, Sistema)):
        return type(self)([ simplify(e) for e in self ])
    else:
        return (sympy.simplify(self)).simplify()

def factor(self):
    """Devuelve las expresiones factorizadas"""
    if isinstance(self, (list, tuple, Sistema)):
        return type(self)([ factor(e) for e in self ])
    else:
        return sympy.factor(self)

```

This code is used in chunk 42.

Defines:

`simplifica`, never used.

Uses `Sistema` 9.

85b

```

<Pinta un objeto en Jupyter 85b>≡
def pinta(data):
    display(Math(latex(simplify(data))))

```

This code is used in chunk 42.

Defines:

`pinta`, never used.

4.2 Completando la clase Sistema

4.2.1 Representación de la clase Sistema

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

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

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

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.

86a

```

<Métodos de representación de la clase Sistema 86a>≡
def __repr__(self):
    """ Muestra un Sistema en su representación python """
    pc = ';' if len(self.lista) else ''
    return 'Sistema([' + \
        ';' .join( repr (e) for e in self ) + \
        pc + '])'

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 """
    pc = ';' if len(self.lista) else r'\ '
    return r'\begin{bmatrix}' + \
        r';&' .join([latex(e) for e in self]) + pc + \
        r'\end{bmatrix}'

```

This definition is continued in chunk 86b.

This code is used in chunk 9.

Uses Sistema 9.

Añado dos métodos más para escribir notebooks dentro de scimax-jupyter

86b

```

<Métodos de representación de la clase Sistema 86a>+≡
def _repr_latex_(self):
    return '$'+self.latex()+'$'

def _repr_png_(self):
    expr = '$'+self.latex()+'$'
    workdir = tempfile.mkdtemp()
    with open(join(workdir, 'borrame.png'), 'wb') as outputfile:
        sympy.preview(expr, viewer='BytesIO', outputbuffer=outputfile)
    return open(join(workdir, 'borrame.png'),'rb').read()

```

This code is used in chunk 9.

4.2.2 Otros métodos de la clase Sistema

Con el método `sis` obtendremos el `Sistema` correspondiente a cualquier `Sistema` o subclase de `Sistema`. Así, si `A` es una `Matrix`, con `A.sis()` obtenemos el `Sistema` de Vectores (columnas) asociado.

87a *<Método para recuperar el Sistema de cualquier subclase de Sistema 87a>≡*

```
def sis(self):
    return Sistema(self.lista)
```

This code is used in chunk 9.
Uses `Sistema` 9.

87b *<Comprobación de que un Sistema es nulo 87b>≡*

```
def es_nulo(self, sust=[]):
    """Indica si es cierto que el Sistema es nulo"""
    return self.subs(sust) == self*0

def no_es_nulo(self, sust=[]):
    """Indica si es cierto que el Sistema no es nulo"""
    return self.subs(sust) != self*0
```

This code is used in chunk 9.
Uses `Sistema` 9.

87c *<Comprobación de que todos los elementos de un Sistema son del mismo tipo 87c>≡*

```
def de_composicion_uniforme(self):
    """Indica si es cierto que todos los elementos son del mismo tipo"""
    return all(type(e)==type(self[0]) for e in self)
```

This code is used in chunk 9.

87d *<Sustitución de un símbolo por otro símbolo o valor en un Sistema 87d>≡*

```
def subs(self, s=[]):
    <Método auxiliar CreaLista que devuelve listas 33b>
    if isinstance(self, sympy.Basic):
        return sympy.S(self).subs(CreaLista(s))
    elif isinstance(self, Sistema):
        Sis = type(self)([ sympy.S(e).subs(2*CreaLista(s)) for e in self ])
        if isinstance(self, Matrix):
            Sis.cF, Sis.cC = self.cF, self.cC
        return Sis
```



```

def simplify(self):
    if isinstance(self, sympy.Basic):
        return sympy.S(self).simplify()
    elif isinstance(self, Sistema):
        return type(self)([ sympy.S(e).simplify() for e in self ])

def factor(self):
    if isinstance(self, sympy.Basic):
        return sympy.S(self).factor()
    elif isinstance(self, Sistema):
        return type(self)([ sympy.S(e).factor() for e in self ])

```

This code is used in chunk 9.

Uses [CreaLista 33b](#), [Matrix 15b](#), and [Sistema 9](#).

88 *<Junta una lista de Sistemas en un único Sistema 88>≡*

```

def junta(self, l):
    """Junta una lista o tupla de Sistemas en uno solo concatenando las
    correspondientes listas de los distintos Sistemas"""
    l = l if isinstance(l, list) else [l]

    junta_dos = lambda x, other: x.concatena(other)
    reune      = lambda x: x[0] if len(x)==1 else junta_dos( reune(x[0:-1]), x[-1] )

    return reune([self] + [s for s in l])

```

This code is used in chunk 9.

Defines:

[junta](#), used in chunk 90b.

Uses [concatena 7](#).

4.3 Completando la clase Vector

4.3.1 Representación de la clase Vector

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

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

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

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.

```
89a <Representación de la clase Vector 89a>≡
def __repr__(self):
    """ Muestra el vector en su representación Python """
    return 'Vector(' + 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 un Vector"""
    if self.rpr == 'fila' or self.n==1:
        return r'\begin{pmatrix}' + \
            ',& '.join([latex(e) for e in self]) + \
            r',\end{pmatrix}'
    else:
        return r'\begin{pmatrix}' + \
            r'\\ '.join([latex(e) for e in self]) + \
            r'\end{pmatrix}'
```

This code is used in chunk 12.

Uses `Vector` 12.

4.3.2 Otros métodos para la clase Vector

```
89b <Creación de una Matrix diagonal a partir de un Vector 89b>≡
def diag(self):
    """Crea una Matrix diagonal cuya diagonal es self"""
    return Matrix([a*(I(self.n)|j) for j,a in enumerate(self, 1)])
```

This code is used in chunk 12.
 Uses `I 102` and `Matrix 15b`.

90a

```
<Normalización de un Vector 90a>≡
def norma(self):
    """Devuelve un múltiplo de un vector (no nulo) pero norma uno"""
    return sympy.sqrt(self*self)

def normalizado(self):
    """Devuelve un múltiplo de un vector (no nulo) pero norma uno"""
    if self.es_nulo(): raise ValueError('Un vector nulo no se puede normalizar')

    return self * fracc(1,self.norma())
```

This code is used in chunk 12.
 Defines:
 `normalizado`, used in chunk 94a.
 Uses `fracc 43`.

4.4 Completando la clase Matrix

4.4.1 Otras formas de instanciar una Matrix

Si se introduce una lista (tupla) de listas o tuplas, creamos una matriz fila a fila. Si se introduce un `SisMat`, se apilan sus matrices concatenando las columnas para obtener una única matriz. Si se introduce una `BlockM` se elimina el particionado y que crea una única matriz.

90b

```
<Creación del atributo lista cuando no tenemos una lista de Vectores 90b>≡
elif isinstance(data, SisMat):
    self.lista = (data|1).apila(data.lista[1:]).lista.copy() if len(data)>1 \
        else (data|1).lista.copy()

elif isinstance(data, BlockM):
    listmat = [Matrix(sismat) for sismat in data]
    self.lista = (listmat[0]).junta(listmat[1:]).lista.copy()

elif isinstance(lista[0], (list, tuple, Sistema)):
    <Verificación de que todas las filas de la matriz tendrán la misma longitud 91a>
    self.lista = [ Vector( [ lista[i][j] for i in range(len(lista)) ] ) \
        for j in range(len(lista[0])) ].copy()
```

This code is used in chunk 15a.
 Uses `apila 95a`, `BlockM 116c`, `junta 88`, `Matrix 15b`, `SisMat 114b`, `Sistema 9`, and `Vector 12`.

4.4.2 Códigos que verifican que los argumentos son correctos

91a \langle Verificación de que todas las filas de la matriz tendrán la misma longitud 91a $\rangle \equiv$

```
if not all ( type(lista[0])==type(v) and len(lista[0])==len(v) for v in lista ):
    raise ValueError('no todo son listas, o no tienen la misma longitud!')
```

This code is used in chunk 90b.

91b \langle Verificación de que todas las columnas de la matriz tienen la misma longitud 91b $\rangle \equiv$

```
if not all ( isinstance(v, Vector) and lista[0].n==v.n for v in lista ):
    raise ValueError('no todo son vectores, o no tienen la misma longitud!')
```

This code is used in chunk 15a.
Uses `Vector` 12.

4.4.3 Representación de la clase `Matrix`

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

92a

```

<Representación de la clase Matrix 92a>≡
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 cfil(self,conjuntoIndices):
    """ Añade el atributo cfilas para insertar líneas horizontales """
    self.cF = set(conjuntoIndices) if conjuntoIndices else {}
    return self

def ccol(self,conjuntoIndices):
    """ Añade el atributo cfilas para insertar líneas horizontales """
    self.cC = set(conjuntoIndices) if conjuntoIndices else {}
    return self

def latex(self):
    """ Construye el comando LaTeX para representar una Matrix """
    ln = [len(n) for n in particion(self.cC,self.n)]
    return \
        '\\left[ \\begin{array}{'+ '|'.join([n*'c' for n in ln]) + '}' + \
        '\\\\ \\hline '.join(['\\\\'.join(['&'.join([latex(e) for e in f.lista]) \
            for f in (~M).lista]) \
            for M in [ i|self for i in particion(self.cF,self.m)]]) + \
        '\\\\ \\end{array} \\right]'

```

This code is used in chunk 15b.
 Uses Matrix 15b and particion 117a.

4.4.4 Otros métodos para la clase Matrix

92b

```

<Métodos útiles para la clase Matrix 92b>≡
<Disposición de los elementos de una Matrix 93a>
<Creación de un Vector a partir de la diagonal de una Matrix 93b>
<Normalizado de las columnas (o filas) de una matriz 94a>
<Apila una lista de Matrix con el mismo número de columnas en una única Matrix 95a>
<Extiende una Matrix a lo largo de la diagonal con una lista de Matrix 95b>
<Transforma una Matrix más una lista de Matrix en una BlockM diagonal 95c>

```

This code is used in chunk 15b.

93a

⟨Disposición de los elementos de una Matrix 93a⟩≡

```

def es_cuadrada(self):
    """Indica si es cierto que la Matrix es cuadrada"""
    return self.m==self.n

def es_simetrica(self):
    """Indica si es cierto que la Matrix es simétrica"""
    return self == ~self

def es_triangularSup(self):
    """Indica si es cierto que la Matrix es triangular superior"""
    return not any(sum([[i|self|j for i in range(j+1,self.m+1)] \
                        for j in range(1 ,self.n+1)], []))

def es_triangularInf(self):
    """Indica si es cierto que la Matrix es triangular inferior"""
    return (~self).es_triangularSup()

def es_triangular(self):
    """Indica si es cierto que la Matrix es triangular inferior"""
    return self.es_triangularSup() | self.es_triangularInf()

def es_diagonal(self):
    """Indica si es cierto que la Matrix es diagonal"""
    return self.es_triangularSup() & self.es_triangularInf()

```

This code is used in chunk 92b.

Defines:

- es_cuadrada, used in chunks 59, 60, 65b, 68, 94, 97b, and 98a.
- es_diagonal, never used.
- es_simetrica, used in chunk 70.
- es_triangular, never used.
- es_triangularInf, never used.
- es_triangularSup, never used.

Uses Matrix 15b.

93b

⟨Creación de un Vector a partir de la diagonal de una Matrix 93b⟩≡

```

def diag(self):
    """Crea un Vector a partir de la diagonal de self"""
    return Vector([ (j|self|j) for j in range(1,min(self.m,self.n)+1)])

```

This code is used in chunk 92b.

Uses Vector 12.

94a *⟨Normalizado de las columnas (o filas) de una matriz 94a⟩≡*

```
def normalizada(self,o='Columnas'):
    if o == 'Columnas':
        if any( v.es_nulo() for v in self):
            raise ValueError('algún vector es nulo')
        return Matrix([ v.normalizado() for v in self])
    else:
        return ~(~self.normalizada())
```

This code is used in chunk 92b.
 Defines:
 normalizada, used in chunk 71b.
 Uses *Matrix* 15b and *normalizado* 90a.

94b *⟨Comprobación de que una Matrix es singular 94b⟩≡*

```
def es_singular(self):
    if not self.es_cuadrada():
        raise ValueError('La matriz no es cuadrada')
    return self.rg()<self.n
```

This code is used in chunk 15b.
 Defines:
 es_singular, used in chunks 59, 60, 68, and 70.
 Uses *es_cuadrada* 93a.

94c *⟨Comprobación de que una Matrix es invertible 94c⟩≡*

```
def es_invertible(self):
    if not self.es_cuadrada():
        raise ValueError('La matriz no es cuadrada')
    return self.rg()==self.n
```

Root chunk (not used in this document).
 Defines:
 es_singular, used in chunks 59, 60, 68, and 70.
 Uses *es_cuadrada* 93a.

95a *<Apila una lista de Matrix con el mismo número de columnas en una única Matrix 95a>≡*

```
def apila(self, l, c=0):
    """Apila una lista o tupla de Matrix con el mismo número de columnas
    en una única Matrix concatenando las respectivas columnas"""
    l = l if isinstance(l, list) else [l]
    apila_dos = lambda x, other, c=0: ~((~x).concatena(~other,c))
    apila = lambda x: x[0] if len(x)==1 else apila_dos( apila(x[0:-1]), x[-1] , c )

    return apila([self] + [s for s in l])
```

This code is used in chunk 92b.

Defines:

apila, used in chunks 59–61, 63b, 68–70, 79a, 83a, 90b, and 95b.

Uses concatena 7 and Matrix 15b.

95b *<Extiende una Matrix a lo largo de la diagonal con una lista de Matrix 95b>≡*

```
def ExtiendeDiag(self,lista):
    if not all(isinstance(m,Matrix) for m in lista):
        return ValueError('No es una lista de matrices')
    Ext_dos = lambda x, y: Matrix(BlockM([[x,M0(x.m,y.n)], [M0(y.m,x.n),y] ]))
    ExtDiag = lambda x: x[0] if len(x)==1 else Ext_dos( ExtDiag(x[0:-1]), x[-1] )
    return ExtDiag([self]+lista)

def extDiag(self,lista,c=0):
    <Método auxiliar CreaLista que devuelve listas 33b>
    lista = CreaLista(lista)
    if not all(isinstance(m,Matrix) for m in lista):
        return ValueError('No es una lista de matrices')
    Ext_dos = lambda x, y: x.apila(M0(y.m,x.n),c).concatena(M0(x.m,y.n).apila(y,c),c)
    ExtDiag = lambda x: x[0] if len(x)==1 else Ext_dos( ExtDiag(x[0:-1]), x[-1] )
    return ExtDiag([self]+lista)
```

This code is used in chunk 92b.

Defines:

extDiag, used in chunk 66a.

ExtiendeDiag, used in chunk 95c.

Uses apila 95a, BlockM 116c, concatena 7, CreaLista 33b, M0 101b, and Matrix 15b.

95c *<Transforma una Matrix más una lista de Matrix en una BlockM diagonal 95c>≡*

```
def BlockDiag(self,lista):
    if not all(isinstance(m,Matrix) for m in lista):
        return ValueError('No es una lista de matrices')

    lm = [e.m for e in [self]+lista]
```



```
ln = [e.n for e in [self]+lista]
return key(lm)|self.ExtiendeDiag(lista)|key(ln)
```

This code is used in chunk 92b.

Defines:

BlockDiag, never used.

Uses ExtiendeDiag 95b and Matrix 15b.

4.4.5 Otros métodos de la clase Matrix que usan la eliminación

Para ver la implementación de la eliminación, véase el Capítulo 2.

Formas pre-escalonada (**K**), escalonada (**L**), y escalonada reducida (**R**) y rango

96

(Métodos de Matrix que usan la eliminación 96)≡

```
def K(self,rep=0, sust=[], repsust=0):
    """Una forma pre-escalonada por columnas (K) de una Matrix"""
    return Elim(self, rep, sust, repsust)

def L(self,rep=0, sust=[], repsust=0):
    """Una forma escalonada por columnas (L) de una Matrix"""
    return ElimG(self, rep, sust, repsust)

def R(self,rep=0, sust=[], repsust=0):
    """Forma escalonada reducida por columnas (R) de una Matrix"""
    return ElimGJ(self, rep, sust, repsust)

def U(self,rep=0, sust=[], repsust=0):
    """Una forma escalonada por filas (U) de una Matrix"""
    return ElimGF(self, rep, sust, repsust)

def UR(self,rep=0, sust=[], repsust=0):
    """Una forma escalonada reducida por filas (U) de una Matrix"""
    return ElimGJF(self, rep, sust, repsust)

def rg(self):
    """Rango de una Matrix"""
    return self.K().rango
```

This definition is continued in chunks 98–100.

This code is used in chunk 15b.

Uses Elim 49, ElimG 50, ElimGF 56, ElimGJ 51, ElimGJF 57, and Matrix 15b.

Cuatro subespacios fundamentales de una matriz

97a *<Métodos de Matrix para obtener los subespacios asociados 97a>≡*

```
def enulo(self):
    """Rango de una Matrix"""
    return SubEspacio( self )

def enuloIzda(self):
    """Rango de una Matrix"""
    return SubEspacio( ~self )

def ecol(self):
    """Rango de una Matrix"""
    return SubEspacio( self.sis() )

def efila(self):
    """Rango de una Matrix"""
    return SubEspacio( (~self).sis() )
```

This code is used in chunk 15b.
Uses Matrix 15b and SubEspacio 76.

4.4.6 Otros métodos de la clase Matrix específicos de las matrices cuadradas

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; para ello usará el método de eliminación Gaussiano que se describirá en el Capítulo 2.

97b *<Potencia de una Matrix 97b>≡*

```
def __pow__(self,n):
    """Calcula la n-ésima potencia de una Matrix"""
    if not isinstance(n,int): raise ValueError('La potencia no es un entero')
    if not self.es_cuadrada: raise ValueError('Matrix no es cuadrada')

    M = self if n >= 0 else I(self.n)
    for i in range(1,abs(n)):
        M = M * self

    return M.inversa() if n < 0 else M
```

This code is used in chunk 15b.
Uses *es_cuadrada* 93a, *I* 102, and *Matrix* 15b.

Cálculo del determinante mediante la expansión de Laplace

(este método no usa la eliminación)

98a *⟨Determinante mediante la expansión de Laplace 98a⟩*≡

```
def det(self, sust=[]):
    """Calculo del determinate mediante la expansión de Laplace"""
    if not self.es_cuadrada(): raise ValueError('Matrix no cuadrada')

    def cof(self,f,c):
        """Cofactor de la fila f y columna c"""
        excl = lambda k: tuple(i for i in range(1,self.m+1) if i!=k)
        return (-1)**(f+c)*(excl(f)|self|excl(c)).det()

    if self.m == 1:
        return 1|self|1

    A = Matrix(self.subs(sust))
    # expansión por la 1ª columna
    return simplify(sum([(f|A|1)*cof(A,f,1)).subs(sust) for f in range(1,A.m+1)]))
```

This code is used in chunk 15b.
Uses `es_cuadrada` 93a and `Matrix` 15b.

Determinante por eliminación

Cálculo del determinante por eliminación.

98b *⟨Métodos de Matrix que usan la eliminación 96⟩*+≡

```
def determinante(self, rep=0):
    """Devuelve el valor del det. de Matrix"""
    return Determinante(self,rep).valor
```

This code is used in chunk 15b.
Defines:
 `determinante`, used in chunks 65a and 66b.
Uses `Determinante` 65b and `Matrix` 15b.

Inversa de una Matrix

99a `<Métodos de Matrix que usan la eliminación 96>+≡`
`def inversa(self, rep=0):`
 `"""Inversa de Matrix"""`
 `return InvMat(self,rep)`

This code is used in chunk 15b.
 Uses Matrix 15b.

Método de diagonalización por semejanza

99b `<Métodos de Matrix que usan la eliminación 96>+≡`
`def diagonalizaS(self, espectro, rep=0):`
 `<Texto de ayuda para la clase DiagonalizaS 69a>`
 `return DiagonalizaS(self, espectro, rep)`

This code is used in chunk 15b.
 Defines:
 `diagonalizaS`, never used.
 Uses DiagonalizaS 68.

Método de diagonalización ortogonal

99c `<Métodos de Matrix que usan la eliminación 96>+≡`
`def diagonaliza0(self, espectro, rep=0):`
 `<Texto de ayuda para la clase Diagonaliza0 71a>`
 `return Diagonaliza0(self, espectro)`

This code is used in chunk 15b.
 Defines:
 `diagonaliza0`, never used.
 Uses Diagonaliza0 70.

Método de Gram-Schmidt

100a *⟨Método Gram-Schmidt para ortogonalizar un sistema de Vectores 100a⟩≡*

```
def GS(self):
    """Devuelve una Matrix equivalente cuyas columnas son ortogonales

    Emplea el método de Gram-Schmidt"""
    A = Matrix(self)
    for n in range(2,A.n+1):
        A & T([ (-frac((A|n)*(A|j),(A|j)*(A|j))), j, n) \
              for j in range(1,n) if (A|j).no_es_nulo() ])
    return A
```

This code is used in chunk 15b.
Uses `frac` 43, `Matrix` 15b, and `T` 38.

Método de diagonalización por congruencia

Evitando dividir si es posible... entonces la matriz **B** será entera...

100b *⟨Métodos de Matrix que usan la eliminación 96⟩+≡*

```
def diagonalizaC(self, rep=0):
    ⟨Texto de ayuda para la clase DiagonalizaC 71c⟩
    return DiagonalizaC(self, rep)
```

This code is used in chunk 15b.
Defines:
 `diagonalizaC`, never used.
Uses `DiagonalizaC` 72.

O con menos operaciones, a coste de que aparezcan fracciones...

100c *⟨Métodos de Matrix que usan la eliminación 96⟩+≡*

```
def diagonalizaCr(self, rep=0):
    ⟨Texto de ayuda para la clase DiagonalizaCr 73c⟩
    return DiagonalizaCr(self, rep)
```

This code is used in chunk 15b.
Defines:
 `diagonalizaCr`, never used.
Uses `DiagonalizaCr` 74.

4.5 Vectores y Matrices especiales

Notación en Mates 2

Los vectores cero **0** y las matrices cero **0** se pueden implementar como subclases de la clase **Vector** y **Matrix** (pero tenga en cuenta que Python necesita conocer el número de componentes del vector y el orden de la matriz):

V0 es una subclase de **Vector** (por tanto hereda los atributos de la clase **Vector**), pero el código inicia (y devuelve) un objeto de su superclase, es decir, inicia y devuelve un **Vector**.

101a *<Definición del vector nulo: V0 101a>*≡

```
class V0(Vector):
    def __init__(self, n, rpr = 'columna'):
        """ Inicializa el vector nulo de n componentes"""
        super().__init__([0 for i in range(n)], rpr)
        self.__class__ = Vector

class V1(Vector):
    def __init__(self, n, rpr = 'columna'):
        """ Inicializa el vector nulo de n componentes"""
        super().__init__([1 for i in range(n)], rpr)
        self.__class__ = Vector
```

This code is used in chunk 42.

Defines:

V0, used in chunks 61, 63c, 76, 77a, 79c, 83a, and 101b.

V1, used in chunk 101b.

Uses **Vector** 12.

Y lo mismo hacemos para matrices

101b *<Definición de la matriz nula: M0 101b>*≡

```
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().__init__([ V0(m) for j in range(n)])
        self.__class__ = Matrix

class M1(Matrix):
    def __init__(self, m, n=None):
        """ Inicializa una matriz nula de orden n """
        n = m if n is None else n

        super().__init__([ V1(m) for j in range(n)])
        self.__class__ = Matrix
```

This code is used in chunk 42.

Defines:

M0, used in chunks 60b, 70, and 95b.
 M1, never used.
 Uses Matrix 15b, V0 101a, and V1 101a.

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 $i|_j = \begin{cases} 1 & \text{si } j = i \\ 0 & \text{si } j \neq i \end{cases}$.

102

<Definición de la matriz identidad: I 102>≡

```
class I(Matrix):
    def __init__(self, n):
        """ Inicializa la matriz identidad de tamaño n """
        super().__init__([[ (i==j)*1 for i in range(n)] for j in range(n)])
        self.__class__ = Matrix
```

This code is used in chunk 42.

Defines:

I, used in chunks 31, 59–63, 66a, 68–72, 74, 77a, 89b, and 97b.
 Uses Matrix 15b.

4.6 Completando la clase T

4.6.1 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.

103 `<Creación del atributo t cuando se instancia con otra T o lista de Ts 103>≡`

```

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 32.
Uses `CreaLista` 33b and `T` 38.

4.6.2 Representación de la clase T

De nuevo construimos los dos métodos de presentación. Uno para la consola de comandos que escribe `T` y entre paréntesis la abreviatura (una tupla o un conjunto) que representa la transformación. Así,

- `T({1, 5})` : intercambio entre los vectores primero y quinto.
- `T((6, 2))` : multiplica por seis el segundo vector.
- `T((-1, 2, 3))` : resta el segundo vector al tercero.

La otra representación es para el entorno Jupyter y replica la notación usada en los apuntes de la asignatura:

Python	Representación en Jupyter
<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>	$\mathbf{A} \tau_{[(a)j]}$	<code>A & T((a,j))</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
$T([(2, 1), (1, 3, 2)])$	τ $[(2)\mathbf{1}]$ $[(1)\mathbf{3+2}]$

Representación de transformaciones identidad. Si las transformaciones multiplican un vector por 1, y suman un vector nulo a otro vector, dichas transformaciones no cambian el sistema de vectores. Lo habitual es que si un paso no modifica nada, que no se represente, por ello se filtran los pasos con el procedimiento *Filtrado de secuencias de transformaciones* 58; si, a resultados del filtrado, la lista de abreviaturas es vacía entonces la representación en L^AT_EX es una cadena vacía (no se pinta ningún símbolo en Jupyter). Si el atributo **rpr** es distinto de 'v' la representación en Jupiter se realiza en horizontal.

```

105 <Representación de la clase T 105>≡
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, sympy.sets.sets.FiniteSet)):
            return '\\left[\\mathbf{' + latex(list(t)[0]) + \
                '\\rightleftharpoons\\mathbf{' + latex(list(t)[1]) + '}\\right]'
        if isinstance(t, (tuple, sympy.core.containers.Tuple)) and len(t) == 2:
            return '\\left[\\left(' + \
                latex(t[0]) + '\\right)\\mathbf{' + latex(t[1]) + '}\\right]'
        if isinstance(t, (tuple, sympy.core.containers.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) + '}{\\pmb{\\tau}}'

    elif self.t == []:
        return ' '

    elif isinstance(self.t, list) and self.rpr=='v':
        return '\\underset{\\begin{subarray}{c} ' + \
            '\\\\'.join([simbolo(i) for i in self.t]) + \
            '\\end{subarray}}{\\pmb{\\tau}}'

    elif isinstance(self.t, list):
        return '\\underset{' + \
            '\\pmb{\\tau}}\\underset{' + \
            '\\pmb{\\tau}}'.join([simbolo(i) for i in self.t]) + \
            '\\pmb{\\tau}'

```

This code is used in chunk 38.
 Uses T 38.

4.7 Representación de los procesos de eliminación Gaussiana

Cuando hemos encadenado varios procedimientos de eliminación, deberíamos poder ver los pasos desde el principio hasta el final. Para ello comprobamos si `data` fue obtenido mediante un proceso previo de eliminación. El modo de saberlo es comprobar si `data` posee el atributo `pasos`. El atributo `tex` guarda el código \LaTeX que muestra el proceso completo, y se construye aplicando el método `PasosYEscritura`. El atributo `pasos` guarda las listas de abreviaturas de las transformaciones elementales empleadas. Por comodidad añadimos dos atributos más: `TrF` es la \LaTeX transformación aplicada a las filas y `TrC` es la \LaTeX transformación aplicada a las columnas.

```

106a <Se guardan los atributos tex y pasos (y se muestran los pasos si se pide) 106a>≡
    pasosPrevios = data.pasos if hasattr(data, 'pasos') and data.pasos else [], []
    TexPasosPrev = data.tex if hasattr(data, 'tex') and data.tex else []
    if repsust:
        self.tex = rprElim(data.subs(sust), pasos, TexPasosPrev, sust)
    else:
        self.tex = rprElim(data.subs(sust), pasos, TexPasosPrev)
    pasos[0] = pasos[0] + pasosPrevios[0]
    pasos[1] = pasosPrevios[1] + pasos[1]
    self.pasos = pasos
    self.TrF = T(pasos[0])
    self.TrC = T(pasos[1])
    if rep:
        display(Math(self.tex))
Root chunk (not used in this document).
Uses rprElim 106c and T 38.

106b <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b>≡
    def texYpasos(data, rep=0, sust=[], repsust=0):
        pasosPrevios = data.pasos if hasattr(data, 'pasos') and data.pasos else [], []
        TexPasosPrev = data.tex if hasattr(data, 'tex') and data.tex else []
        if repsust:
            tex = rprElim(data, pasos, TexPasosPrev, sust)
        else:
            tex = rprElim(data, pasos, TexPasosPrev)
        pasos[0] = pasos[0] + pasosPrevios[0]
        pasos[1] = pasosPrevios[1] + pasos[1]

        TrF = T(pasos[0])
        TrC = T(pasos[1])

        if rep:
            display(Math(tex))

        return [tex, pasos, TrF, TrC]

This code is used in chunks 49–53 and 55–57.
Uses rprElim 106c and T 38.

```

Cuando mostramos los pasos, es más legible mostrar únicamente los que modifican la matriz (omitiendo sustituciones de una columna por ella misma, productos de una columna por 1, o sumas de un vector nulo a una columna).

El atributo `tex` guardará el código \LaTeX que muestra el proceso completo. Si ha habido transformaciones previas, la cadena de \LaTeX que permite su representación en el entorno Jupyter estará guardada en la variable (`TexPasosPrev`), y a dicha cadena hay que añadir la correspondiente cadena de \LaTeX que permita representar los nuevos `pasos` dados como argumento de este método. Si `TexPasosPrev` es vacío, la escritura comienza con la representación de `data`. A la hora de representar los pasos hay que tener en cuenta si se dan sobre las filas (`l==0`) o sobre las columnas (`l==1`).

```

106c <Representación de un proceso de eliminación 106c>≡
    def rprElim(data, pasos, TexPasosPrev=[], sust=[]):
        """Escribe en LaTeX los pasos efectivos y las sucesivas matrices"""

```

```

A = data.copy().subs(sust)
if isinstance (data, Matrix):
    A.cF, A.cC = data.cF, data.cC

tex = latex(A) if not TexPasosPrev else TexPasosPrev
for l in 0,1:
    if l==0:
        for i in reversed(range(len(pasos[l]))):
            tex += '\\xrightarrow{' + latex(pasos[l][i]) + '}'
            tex += latex( (pasos[l][i] & A).subs(sust) )
    if l==1:
        for i in range(len(pasos[l])):
            tex += '\\xrightarrow{' + latex(pasos[l][i]) + '}'
            tex += latex( (A & pasos[l][i]).subs(sust) )

return tex

```

This definition is continued in chunks 107 and 108.

This code is used in chunk 42.

Defines:

`rprElim`, used in chunks 59–61, 63c, 69b, 106, and 108.

Uses `Matrix` 15b.

107a

⟨Representación de un proceso de eliminación 106c⟩+≡

```

def rprElimFyC(data, pasos, TexPasosPrev=[], sust=[]):
    """Escribe en LaTeX los pasos efectivos y las sucesivas matrices"""
    A = data.copy().subs(sust)
    if isinstance (data, Matrix):
        A.cF, A.cC = data.cF, data.cC

    tex = latex(data) if not TexPasosPrev else TexPasosPrev
    for i in range(len(pasos[1])):
        tex += '\\xrightarrow' \
            + '[' + latex(T(pasos[0][-i-1])) + ']' \
            + '{' + latex(T(pasos[1][i])) + '}'
        tex += latex( (pasos[0][-i-1] & A & pasos[1][i]).subs(sust) )

    return tex

```

This code is used in chunk 42.

Defines:

`rprElimFyC`, used in chunks 66a and 108.

Uses `Matrix` 15b and `T` 38.

107b

⟨Representación de un proceso de eliminación 106c⟩+≡

```

def rprElimCF(data, pasos, TexPasosPrev=[], sust=[]):
    """Escribe en LaTeX los pasos efectivos y las sucesivas matrices"""

```

```

A = data.copy().subs(sust)
if isinstance (data, Matrix):
    A.cF, A.cC = data.cF, data.cC

tex = latex(data) if not TexPasosPrev else TexPasosPrev
for i in range(len(pasos[1])):
    tex += '\\xrightarrow[' + latex(T(pasos[1][i])) + ']'
    tex += latex( (A & pasos[1][i]).subs(sust) )
    tex += '\\xrightarrow[' + latex(T(pasos[0][-i-1])) + ']{ }'
    tex += latex( (pasos[0][-i-1] & A).subs(sust) )

return tex

```

This code is used in chunk 42.

Defines:

`rprElimCF`, used in chunks 72, 74, and 108.

Uses `Matrix` 15b and `T` 38.

Este procedimiento añadido es para “pintar” los pasos de eliminación.

108

```

<Representación de un proceso de eliminación 106c>+≡
def dispElim(self, pasos, TexPasosPrev=[]):
    display(Math(rprElim(self, pasos, TexPasosPrev)))

def dispElimFyC(self, pasos, TexPasosPrev=[]):
    display(Math(rprElimFyC(self, pasos, TexPasosPrev)))

def dispElimCF(self, pasos, TexPasosPrev=[]):
    display(Math(rprElimCF(self, pasos, TexPasosPrev)))

```

This code is used in chunk 42.

Defines:

`dispElim`, never used.

`dispElimCF`, never used.

`dispElimFyC`, never used.

Uses `rprElim` 106c, `rprElimCF` 107b, and `rprElimFyC` 107a.

4.8 Representación de la resolución de sistemas de ecuaciones

109a *⟨Métodos de representación de la clase Homogenea 109a⟩*≡

```
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 '\\left\\{\\ ' + latex(self.sgen[1]) + '\\ \\right\\}'
    else:
        return '\\mathcal{L}\\left(\\ ' + latex(self.sgen) + '\\ \\right)'
```

This code is used in chunk 61.
Uses Sistema 9.

109b *⟨Métodos de representación de la clase SEL 109b⟩*≡

```
def __repr__(self):
    """Muestra el Espacio Nulo de una matriz en su representación Python"""
    return repr(self.solP) + ' + Combinaciones lineales de (' + repr(self.sgen) + ')',

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

def latex(self):
    """ Construye el comando LaTeX para la solución de un Sistema Homogéneo"""
    if self.determinado and self.solP:
        return '\\left\\{\\ ' + latex(self.solP) + '\\ \\right\\}'
    else:
        return self.eafin.EcParametricas() if self.solP else latex(set())
```

This code is used in chunk 63b.
Uses EcParametricas 110 and Sistema 9.

4.9 Completando la clase EAfin

4.9.1 Representación de la clase EAfin

```

110 <Métodos de representación de la clase EAfin 110>≡
    def _repr_html_(self):
        """Construye la representación para el entorno Jupyter Notebook"""
        return html(self.latex())

    def EcParametricas(self, d=0):
        """Representación paramétrica de EAfin"""
        punto = latex(self.v) + '+' if (self.v != 0*self.v) else ''
        if d: display(Math(self.EcParametricas()))
        return '\\left\\{ \\boldsymbol{v}\\in\\mathbb{R}^{' + \
            + latex(self.S.Rn) + \
            + '\\ \\left|\\ \\exists\\boldsymbol{p}\\in\\mathbb{R}^{' + \
            + latex(max(self.S.dim,1)) + \
            + ',\\; \\boldsymbol{v}= ' + \
            + punto + \
            + latex(Matrix(self.S.sgen)) + \
            + '\\boldsymbol{p}\\right. \\right\\}' + \

    def EcCartesianas(self, d=0):
        """Representación cartesiana de EAfin"""
        if d: display(Math(self.EcCartesianas()))
        return '\\left\\{ \\boldsymbol{v}\\in\\mathbb{R}^{' + \
            + latex(self.S.Rn) + \
            + '\\ \\left| ' + \
            + latex(self.S.cart) + \
            + '\\boldsymbol{v}= ' + \
            + latex(self.S.cart*self.v) + \
            + '\\right.\\right\\}' + \

    def latex(self):
        """ Construye el comando LaTeX para un EAfin de Rn"""
        return self.EcParametricas() + '\\; = \\;' + self.EcCartesianas()

```

This code is used in chunk 81a.

Defines:

EcCartesianas, used in chunk 111.

EcParametricas, used in chunks 109b and 111.

Uses EAfin 80b and Matrix 15b.

4.10 Completando la clase SubEspacio

4.10.1 Representación de la clase SubEspacio

```

111 <Métodos de representación de la clase SubEspacio 111>≡
    def _repr_html_(self):
        """Construye la representación para el entorno Jupyter Notebook"""
        return html(self.latex())

    def EcParametricas(self, d=0):
        """Representación paramétrica del SubEspacio"""
        if d: display(Math(self.EcParametricas()))
        return EAfin(self.sgen,self.sgen|1).EcParametricas()

    def EcCartesianas(self, d=0):
        """Representación cartesiana del SubEspacio"""
        if d: display(Math(self.EcCartesianas()))
        return EAfin(self.sgen,self.sgen|1).EcCartesianas()

    def latex(self):
        """ Construye el comando LaTeX para un SubEspacio de Rn"""
        return EAfin(self.sgen,self.sgen|1).latex()

```

This code is used in chunk 80a.

Uses EAfin 80b, EcCartesianas 110, EcParametricas 110, and SubEspacio 76.

4.11 La clase BlockM. 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

Consulte el Notebook sobre el **uso del módulo nacal** en la carpeta “Notebooks” en <https://github.com/mbujosab/nacallib/tree/master/doc/Notebooks>.

4.11.1 La clase SisMat. Sistema de Matrices

De manera auxiliar y para ayudar a una representación más clara de las **BlockM** y del funcionamiento del operador selector de un **Sistema** en el caso de las **BlockM**, vamos a definir una nueva subclase auxiliar de **Sistema**, compuesto por matrices con el mismo número de columnas. Lo denominaremos **SisMat** (sistema de matrices) y lo representaremos entre paréntesis y con las **Matrix** dispuestas unas encima de otras y separadas por líneas horizontales (algo así como un vector en forma de columna cuyos elementos son matrices):

$$\left(\begin{array}{ccc} 1 & 2 & 1 \\ 7 & -3 & 0 \\ 6 & 4 & 2 \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right)$$

Y si el sistema `SisMat` contiene una única `Matrix`, también pintamos los corchetes de la matriz para no confundir un `SisMat` con una `Matrix`:

$$\left(\begin{bmatrix} 1 & 2 & 1 \\ 7 & -3 & 0 \\ 6 & 4 & 2 \end{bmatrix} \right)$$

Es decir, si está encerrado entre paréntesis es un `SisMat`.

Para mayor comodidad, `Matrix(B)` donde `B` es una `SisMat` apila las matrices de `B` en una única `Matrix`.

Una vez definidos los `SisMat`, definiremos las matrices por bloques `BlockM` como listas de `SisMat` (cada `SisMat` será una columna de matrices). Así, si `A` es una `BlockM`, entonces `A|1` selecciona un `SisMat` con las matrices de la primera columna de `A` (con `1|A` generamos una `BlockM` cuyas `SisMats` contienen únicamente las `Matrix` de la primera fila de `BlockM`).

112a

```
<Iniciación de la clase SisMat 112a>≡
def __init__(self, data):
    """Inicializa un SisMat con una lista, tupla o Sistema de Matrix,
    (todas con el mismo número de columnas).
    """
    super().__init__(data)

    lista = Sistema(data).lista.copy()

    if isinstance(data[0], Matrix):

        <Verificamos que todas las Matrix tienen el mismo número de columnas 112b>
        self.lista = lista.copy()

    self.n      = len(self)

    self.ln     = (self|1).n
    self.lm     = [matriz.m for matriz in self]
```

This code is used in chunk 114b.

Uses `Matrix` 15b, `SisMat` 114b, and `Sistema` 9.

112b

```
<Verificamos que todas las Matrix tienen el mismo número de columnas 112b>≡
if not all( isinstance(m,Matrix) and m.n==lista[0].n for m in self):
    raise ValueError('0 no todo son Matrix o no tienen el mismo número de columnas!')
```

This code is used in chunk 112a.

Uses `Matrix` 15b.

Cuando el argumento del operador selector por la derecha es entero, lista o slice, funciona como con cualquier `Sistema` genérico. Pero cuando el argumento es un *conjunto*, se particiona el sistema, de manera que se devuelve la `BlockM` resultante de particionar las matrices por la derecha de las columnas cuyos índices aparecen en el conjunto (manteniendo las particiones horizontales)

```

113a  <Operador selector por la derecha para la clase SisMat 113a>≡
      def __or__(self, j):
          <Operador selector por la derecha cuando el argumento es entero, lista o slice 19a>
          elif isinstance(j, set):
              return BlockM([ SisMat( [Mat|list(c) for Mat in self] ) \
                                   for c in particion(j, self.ln) ])

```

This code is used in chunk 114b.

Uses BlockM 116c, particion 117a, and SisMat 114b.

Cuando el argumento del operador selector por la izquierda es entero, lista o slice, funciona como por la derecha. Pero cuando el argumento es un *conjunto*, se reparticiona *SisMat* por debajo de las filas cuyos índices aparecen en el conjunto (manteniendo las particiones horizontales). Para ello se unifica el *SisMat* en una única *Matrix*, que se trocea en submatrices por las filas indicadas en el argumento *i*.

```

113b  <Operador selector por la izquierda para la clase SisMat 113b>≡
      def __ror__(self,i):
          """Hace exactamente lo mismo que el método __or__ por la derecha
          cuando es el argumento es int, list, tuple o slice. Cuando el
          argumento es un conjunto se reparticiona por las filas indicadas por
          el conjunto"""
          if isinstance(i, (int, list, tuple, slice)):
              return self | i
          elif isinstance(i, set):
              return SisMat([ list(f)|Matrix(self) for f in particion(i, Matrix(self).m) ])

```

This code is used in chunk 114b.

Uses Matrix 15b, particion 117a, and SisMat 114b.

```

113c  <Representación de la clase SisMat 113c>≡
      def __repr__(self):
          """ Muestra un SisMat en su representación Python """
          return 'SisMat(' + repr(self.lista) + ')'

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

```

This definition is continued in chunk 114a.

This code is used in chunk 114b.

Uses SisMat 114b.

114a *<Representación de la clase SisMat 113c>+≡*

```
def latex(self):
    """ Escribe el código de LaTeX para representar una SisMat """
    if self.n == 1:
        return '\\begin{pmatrix}' + latex(self|1) + '\\end{pmatrix}'

    else:
        return \
            '\\left(\\!\\!\\!\\!\\!\\!\\!\\!\\left(' + \
            '\\begin{array}{'+ self.ln*'c' + '}' + \
            '\\\\ \\hline\\hline'.join( ['\\\\'.join( ['&'.join( [latex(e) \
                for e in fila ]) for fila in ~Mat ]) for Mat in self ]) + \
            '\\\\' + \
            '\\end{array}' + \
            '\\right)\\!\\!\\!\\!\\!\\!\\!\\!\\right)'
```

This code is used in chunk 114b.
Uses SisMat 114b.

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

114b *<Definición de la clase SisMat 114b>≡*

```
class SisMat(Sistema):
    <Inicialización de la clase SisMat 112a>
    <Operador selector por la derecha para la clase SisMat 113a>
    <Operador selector por la izquierda para la clase SisMat 113b>
    <Representación de la clase SisMat 113c>
```

This code is used in chunk 42.
Defines:
 SisMat, used in chunks 19c, 22b, 90b, and 112–15.
Uses *Sistema* 9.

4.11.2 La clase BlockM. Matrices particionadas (o matrices por bloques)

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.

Aquí implementamos las matrices particionadas en la clase `BlockM`, pero lo hacemos de un modo ligeramente diferente: las `BlockM` serán sistemas de `SisMat` con el mismo número de matrices, y tales que las matrices *i*ésimas de dichos `SisMat` tengan el mismo número de filas.

El argumento de inicialización es una lista de `Sistemas` de matrices (o de `SisMats`), cada elemento de la lista será una columna de bloques (o submatrices con el mismo número de columnas).

Alternativamente, el argumento de inicialización puede ser una lista de listas de matrices con el mismo número de filas. Cada lista de matrices será una fila de bloques (o submatrices con el mismo número de filas).

El atributo `self.n` contiene el número de `SisMats` (columnas de bloques o submatrices) y `self.m` contiene el número de matrices de dichos `SisMat` (filas 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.

```

115  <Iniciación de la clase BlockM 115>≡
      def __init__(self, data):
          """Inicializa una BlockM con una lista, tupla, o Sistema: de SisMats
            (serán las columnas de matrices) o bien de listas o tuplas de
            matrices (filas de matrices)
            """
          super().__init__(data)

          lista = Sistema(data).lista

          if isinstance(lista[0], Sistema):
              <Verificación de que todos son Sistemas de matrices y de la misma longitud 116a>
              self.lista = [ SisMat(e) for e in lista ].copy()

          elif isinstance(data[0], (list, tuple)):
              <Verificación de que todas son listas de matrices y de la misma longitud 116b>
              self.lista = [ SisMat([ lista[j][i] for j in range(len(lista)) ] \
                                     for i in range(len(lista[0])) ).copy()

          self.m      = len(self|1)
          self.n      = len(self)

          self.lm     = (self|1).lm
          self.ln     = [sm.ln for sm in self]

```

This code is used in chunk 116c.

Uses BlockM 116c, SisMat 114b, and Sistema 9.

```

116a <Verificación de que todos son Sistemas de matrices y de la misma longitud 116a>≡
      if not all ( isinstance(s, Sistema) and s.de_composicion_uniforme() and \
                  isinstance(s[1, Matrix) and len(s)==len(lista[0]) for s in lista ):
          raise ValueError('no son Sistemas de matrices, o no tienen la misma longitud!')

This code is used in chunk 115.
Uses Matrix 15b and Sistema 9.

116b <Verificación de que todas son listas de matrices y de la misma longitud 116b>≡
      if not all ( isinstance(s, (list,tuple)) and isinstance(s[0], Matrix) and \
                  all(type(e)==type(lista[0]) for e in lista) and \
                  len(s)==len(lista[0]) for s in lista ):
          raise ValueError('no son listas de matrices, o no tienen la misma longitud!')

This code is used in chunk 115.
Uses Matrix 15b.

```

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

```

116c <Definición de la clase BlockM 116c>≡
      class BlockM(Sistema):
          <Inicialización de la clase BlockM 115>
          <Operador selector por la derecha para la clase BlockM 118b>
          <Operador selector por la izquierda para la clase BlockM 119c>
          <Representación de la clase BlockM 120>

This code is used in chunk 42.
Defines:
    BlockM, used in chunks 13, 19c, 22b, 90b, 95b, 113a, 115, and 117–20.
Uses Sistema 9.

```

4.11.3 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 dichos 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

$$\{i_1, \dots, i_r\} \mathbf{A} = \left[\begin{array}{c} (1:i_1) \mathbf{A} \\ (i_1+1:i_2) \mathbf{A} \\ \vdots \\ (i_{r-1}+1:i_r) \mathbf{A} \end{array} \right]$$

- 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

$$\mathbf{A}_{\{j_1, \dots, j_s\}} = \left[\begin{array}{c|c|c|c} \mathbf{A}_{(1:j_1)} & \mathbf{A}_{(j_1+1:j_2)} & \cdots & \mathbf{A}_{(j_{s-1}+1:j_s)} \end{array} \right]$$

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

117a

```

<Definición del método particion 117a>≡
def particion(s,n):
    """ genera la lista de particionamiento a partir de un conjunto y un número
    >>> particion({1,3,5},7)

    [[1], [2, 3], [4, 5], [6, 7]]
    """
    p = sorted(list(s | set([0,n])))
    return [ list(range(p[k]+1,p[k+1]+1)) for k in range(len(p)-1) ]

```

This code is used in chunk 42.

Defines:

`particion`, used in chunks 92a, 113, 117b, and 118a.

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

117b

```

<Partición de una matriz por filas de bloques 117b>≡
elif isinstance(i,set):
    return BlockM ([ [a|self] for a in particion(i,self.m) ])

```

This code is used in chunk 23.

Uses `BlockM` 116c and `particion` 117a.

118a *⟨Partición de una matriz por columnas de bloques 118a⟩*≡

```

elif isinstance(j, set):
    return BlockM ([ [self|a for a in particion(j, self.n)] ])

```

This code is used in chunk 20.
 Uses BlockM 116c and particion 117a.

Pero aún nos falta algo:

Notación en Mates 2

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

$$\{i_1, \dots, i_r\} | \mathbf{A} | \{j_1, \dots, j_s\} = \left[\begin{array}{c|c|c|c} (1:i_1) | \mathbf{A} | (1:j_1) & (1:i_1) | \mathbf{A} | (j_1+1:j_2) & \cdots & (1:i_1) | \mathbf{A} | (j_s+1:n) \\ \hline (i_1+1:i_2) | \mathbf{A} | (1:j_1) & (i_1+1:i_2) | \mathbf{A} | (j_1+1:j_2) & \cdots & (i_1+1:i_2) | \mathbf{A} | (j_s+1:n) \\ \hline \vdots & \vdots & \cdots & \vdots \\ \hline (i_k+1:m) | \mathbf{A} | (1:j_1) & (i_k+1:m) | \mathbf{A} | (j_1+1:j_2) & \cdots & (i_k+1:m) | \mathbf{A} | (j_s+1:n) \end{array} \right]$$

es decir, queremos poder particionar una BlockM. Los casos interesantes son cuando particionamos por el lado contrario por el que se particionó la matriz inicial, es decir,

$$\{i_1, \dots, i_r\} | \left(\mathbf{A} | \{j_1, \dots, j_s\} \right) \quad \text{y} \quad \left(\{i_1, \dots, i_r\} | \mathbf{A} \right) | \{j_1, \dots, j_s\}$$

que, por supuesto, debe dar el mismo resultado.

Cuando el argumento del operador selector por la derecha es entero, lista o slice, funciona como con cualquier Sistema genérico. Pero cuando el argumento es un conjunto y hay una única columna de matrices (`self.n == 1`), es decir, cuando BlockM contiene un único SisMat, se particiona dicho SisMat para obtener la BlockM correspondiente. El caso general (cuando el sistema contiene más de un SisMat) se verá más tarde:

118b *⟨Operador selector por la derecha para la clase BlockM 118b⟩*≡

```

def __or__(self, j):
    ⟨Operador selector por la derecha cuando el argumento es entero, lista o slice 19a⟩
    elif isinstance(j, set):
        if self.n == 1:
            return (self|1)|j

    ⟨Caso general de repartición por columnas 119b⟩

```

This code is used in chunk 116c.

y hacemos lo mismo para particionar por filas cuando `self.m == 1` (la matriz por bloques tiene una única fila):

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:

119a *<Definición del procedimiento de generación del conjunto clave para particionar 119a>≡*

```
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 42.

Así, los casos generales consisten en reparticionar de nuevo:

119b *<Caso general de repartición por columnas 119b>≡*

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

This code is used in chunk 118b.
Uses Matrix 15b.

El operador selector por la izquierda es más sencillo, pues usa el operador selector por la izquierda de las *SisMat*:

119c *<Operador selector por la izquierda para la clase BlockM 119c>≡*

```
def __ror__(self,i):
    if isinstance(i, (int)):
        return BlockM( [ [i|sm for sm in self] ] )

    if isinstance(i, (list,tuple,slice,set) ):
        return BlockM( [i|sm for sm in self] )
```

This code is used in chunk 116c.
Uses BlockM 116c.

Observación 4. El método `__or__` está definido para conjuntos ... y da como resultado la unión de conjuntos. Por tanto si A es una matriz, el resultado de $\{1,2\} | (\{3\} | A)$ es distinto del obtenido con $(\{1,2\} | \{3\}) | A$. El primero da lo mismo que $\{1,2\} | A$, mientras que el segundo nos da $\{1,2,3\} | A$.

4.11.4 Representación de la clase BlockM

A continuación definimos las reglas de representación para las matrices por bloques. `Matrix` y `BlockM` 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 `BlockM`. Así, si una matriz por bloques tienen un único bloque, pintaremos una caja alrededor para distinguirla de una matriz ordinaria:

$$\begin{bmatrix} 1 & 2 & 1 \\ 7 & -3 & 0 \end{bmatrix} \qquad \boxed{\begin{bmatrix} 1 & 2 & 1 \\ 7 & -3 & 0 \end{bmatrix}}$$

```
120 <Representación de la clase BlockM 120>≡
def __repr__(self):
    """ Muestra una BlockM en su representación Python """
    return 'BlockM(' + repr(self.lista) + ')'

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 BlockM """
    Neg = '\\!\\!\\!' if Matrix(self).m > 1 else '\\!'
    Neg2 = '\\!' if len(self.ln) > 1 and Matrix(self).m == 2 else ''
    Neg3 = '\\!' if Matrix(self).m > 2 else ''
    Pos = '\\,' if Matrix(self).m > 2 else ''
    return \
        '\\,\\,\\,\\left[' + Neg + Neg2 + Neg3 + '\\left[' + Pos + \
        '\\begin{array}{'} + '|'.join([n*'c' for n in self.ln]) + '}' + \
        '\\\\ \\hline '.join( ['\\\\'.join( ['&'.join( \
            [latex(e) for e in fila]) for fila in ~Mat]) for Mat in (self[{0}|1])]) + \
        '\\\\' + \
        '\\end{array}' + Pos + \
        '\\right]' + Neg + Neg2 + Neg3 + '\\right]\\,\\,\\,'
```

This code is used in chunk 116c.
Uses `BlockM` 116c and `Matrix` 15b.

Capítulo 5

Sobre este documento

Con ánimo de que esta documentación sea más didáctica, en el Capítulo 1 muestro las partes más didácticas del código, y relego las otras al Capítulo 4. Así puedo destacar cómo la librería de Python es una implementación literal de las definiciones dadas en mis notas de la asignatura de Mates II. Para lograr presentar el código en un orden distinto del que realmente tiene en la librería uso la herramienta `noweb`. Una breve explicación aparece en la siguiente sección...

Literate programming con `noweb`

Este documento está escrito usando `noweb`. Es una herramienta que permite escribir a la vez tanto código como su documentación. El código se escribe a trozos o “chunks” como por ejemplo este:

121a *<Chunk de ejemplo que define la lista a 121a>≡*
a = ["Matemáticas II es mi asignatura preferida", "Python mola", 1492, "Noweb"]
This code is used in chunk 122.

y este otro chunk:

121b *<Segundo chunk de ejemplo que cambia el último elemento de la lista a 121b>≡*
a[-1] = 10
This code is used in chunk 122.

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:

122

```

<EjemploLiterateProgramming.py 122>≡
  <Chunk de ejemplo que define la lista a 121a>
  <Segundo chunk de ejemplo que cambia el último elemento de la lista a 121b>

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

  <Chunk final que indica qué tipo de objeto es a y hace unas sumas 125b>
  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 125 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

```

5.1 Secciones de código

<Transformación elemental espejo de una T 36b> 36b, 38
 <Apila una lista de Matrix con el mismo número de columnas en una única Matrix 95a> 92b, 95a
 <Aplicación de las transformaciones a las columnas y a las filas 73b> 72, 73b, 74
 <Aplicación de las transformaciones y sus inversas "espejo" 69b> 68, 69b
 <Aplicamos los pasos de eliminación sobre la matriz ampliada y obtenemos la solución 63c> 63b, 63c
 <Apuntamos las transformaciones Tr y las aplicamos sobre las columnas 47c> 47c, 49, 50, 51, 52, 53a, 53b
 <Calculando el determinante 65b> 42, 65b
 <Caso general de repartición por columnas 119b> 118b, 119b
 <Chunk de ejemplo que define la lista a 121a> 121a, 122
 <Chunk final que indica qué tipo de objeto es a y hace unas sumas 125b> 122, 125b
 <Composición de Transformaciones Elementales o aplicación sobre las filas de una Matrix 34> 34, 38
 <Comprobación de que todos los elementos de un Sistema son del mismo tipo 87c> 9, 87c
 <Comprobación de que un Sistema es nulo 87b> 9, 87b
 <Comprobación de que una Matrix es invertible 94c> 94c
 <Comprobación de que una Matrix es singular 94b> 15b, 94b
 <Copyright y licencia GPL 125a> 125a

<Creación de un Vector a partir de la diagonal de una Matrix 93b> 92b, 93b
 <Creación de una Matrix diagonal a partir de un Vector 89b> 12, 89b
 <Creación del atributo lista cuando no tenemos una lista de Vectores 90b> 15a, 90b
 <Creación del atributo t cuando se instancia con otra T o lista de Ts 103> 32, 103
 <Cálculo de L y de una base del espacio nulo de A 62> 61, 62
 <Cálculo del determinante y representación de los pasos en Jupyter 66a> 65b, 66a
 <Definición de la clase BlockM 116c> 42, 116c
 <Definición de la clase Matrix 15b> 15b, 42
 <Definición de la clase SisMat 114b> 42, 114b
 <Definición de la clase Sistema 9> 9, 42
 <Definición de la clase T (Transformación Elemental) 38> 38, 42
 <Definición de la clase Vector 12> 12, 42
 <Definición de la matriz identidad: I 102> 42, 102
 <Definición de la matriz nula: MO 101b> 42, 101b
 <Definición del método particion 117a> 42, 117a
 <Definición del método auxiliar BuscaNuevoPivote 45a> 45a, 49, 50, 51, 52, 53a, 53b, 68
 <Definición del método auxiliar BuscaPrimerNoNuloEnLaDiagonal 73a> 72, 73a, 74
 <Definición del procedimiento de generación del conjunto clave para particionar 119a> 42, 119a
 <Definición del vector nulo: VO 101a> 42, 101a
 <Determinante mediante la expansión de Laplace 98a> 15b, 98a
 <Diagonalizando Ortogonalmente una matriz simétrica 70> 42, 70
 <Diagonalizando una matriz por Congruencia 72> 42, 72, 74
 <Diagonalizando una matriz por bloques triangulares (por Semejanza) 68> 42, 68
 <Disposición de los elementos de una Matrix 93a> 92b, 93a
 <EjemploLiterateProgramming.py 122> 122
 <Extiende una Matrix a lo largo de la diagonal con una lista de Matrix 95b> 92b, 95b
 <Filtrado de secuencias de transformaciones 58> 42, 58
 <Inicialización de la clase BlockM 115> 115, 116c
 <Inicialización de la clase EAfin 80b> 80b, 81a
 <Inicialización de la clase Matrix 15a> 15a, 15b
 <Inicialización de la clase SisMat 112a> 112a, 114b
 <Inicialización de la clase Sistema 5b> 5b, 9
 <Inicialización de la clase SubEspacio 76> 76, 80a
 <Inicialización de la clase T (Transformación Elemental) 32> 32, 38
 <Inicialización de la clase Vector 11a> 11a, 12
 <Intercambio de columnas para escalonar 47a> 47a, 50, 53a
 <Invirtiendo una matriz 59> 42, 59, 60a, 60b
 <Junta una lista de Sistemas en un único Sistema 88> 9, 88
 <La clase EAfin 81a> 42, 81a
 <La clase SubEspacio 80a> 42, 80a
 <Método auxiliar CreaLista que devuelve listas 33b> 32, 33b, 34, 36a, 36b, 36c, 87d, 95b
 <Método auxiliar SGenENulo que Encuentra un sistema generador del Espacio Nulo de A 77a> 76, 77a
 <Método auxiliar para creación de una base ortonormal donde q es el último vector 71b> 70, 71b
 <Método auxiliar que calcula la inversa de una Transformación elemental 36a> 35b, 36a
 <Método de la clase Sistema para concatenar dos Sistemas 7> 7, 9
 <Método Gram-Schmidt para ortogonalizar un sistema de Vectores 100a> 15b, 100a
 <Método html general 84a> 42, 84a
 <Método latex general 84b> 42, 84b
 <Método para recuperar el Sistema de cualquier subclase de Sistema 87a> 9, 87a
 <Método que define los atributos tex y pasos (y muestra los pasos si se pide) 106b> 49, 50, 51, 52, 53a, 53b, 55, 56, 57, 106b
 <Métodos auxiliares para usar coeficientes racionales cuando sea posible 43> 42, 43
 <Métodos de Matrix para obtener los subespacios asociados 97a> 15b, 97a
 <Métodos de Matrix que usan la eliminación 96> 15b, 96, 98b, 99a, 99b, 99c, 100b, 100c
 <Métodos de la clase EAfin 81b> 81a, 81b, 81c, 82a, 82b, 83a, 83b
 <Métodos de la clase Sistema para que actúe como si fuera una list de Python 6a> 6a, 6b, 9
 <Métodos de la clase SubEspacio 77b> 77b, 78a, 78b, 78c, 79a, 79b, 79c, 80a

<Métodos de representación de la clase **Determinante** 66b> 65b, 66b
 <Métodos de representación de la clase **EAfin** 110> 81a, 110
 <Métodos de representación de la clase **Homogenea** 109a> 61, 109a
 <Métodos de representación de la clase **SEL** 109b> 63b, 109b
 <Métodos de representación de la clase **Sistema** 86a> 9, 86a, 86b
 <Métodos de representación de la clase **SubEspacio** 111> 80a, 111
 <Métodos útiles para la clase **Matrix** 92b> 15b, 92b
 <nacal.py 42> 42
 <Normalización de un **Vector** 90a> 12, 90a
 <Normalización del pivote para que sea igual a uno 47b> 47b, 51, 53b
 <Normalizado de las columnas (o filas) de una matriz 94a> 92b, 94a
 <Operador selector por la derecha cuando el argumento es entero, lista o slice 19a> 18, 19a, 20, 113a, 118b
 <Operador selector por la derecha para la clase **BlockM** 118b> 116c, 118b
 <Operador selector por la derecha para la clase **Matrix** 20> 15b, 20
 <Operador selector por la derecha para la clase **SisMat** 113a> 113a, 114b
 <Operador selector por la derecha para la clase **Sistema** 18> 9, 18
 <Operador selector por la izquierda para la clase **BlockM** 119c> 116c, 119c
 <Operador selector por la izquierda para la clase **Matrix** 23> 15b, 23
 <Operador selector por la izquierda para la clase **SisMat** 113b> 113b, 114b
 <Operador selector por la izquierda para la clase **Vector** 19b> 12, 19b
 <Operador transposición para la clase **Matrix** 22a> 15b, 22a
 <Operador transposición para la clase **T** 35a> 35a, 38
 <Opuesto de un **Sistema** 27> 9, 27
 <Otros métodos de la clase **Transformación elemental** 37a> 37a, 37b, 38
 <Partición de una matriz por columnas de bloques 118a> 20, 118a
 <Partición de una matriz por filas de bloques 117b> 23, 117b
 <Pinta un objeto en Jupyter 85b> 42, 85b
 <Potencia de una **Matrix** 97b> 15b, 97b
 <Potencia de una **T** 35b> 35b, 38
 <Producto de un **Sistema** por un escalar a su izquierda 26b> 9, 26b
 <Producto de un **Sistema** por un escalar, un **Vector** o una **Matrix** a su derecha 29> 9, 29
 <Representación de la clase **BlockM** 120> 116c, 120
 <Representación de la clase **Matrix** 92a> 15b, 92a
 <Representación de la clase **SisMat** 113c> 113c, 114a, 114b
 <Representación de la clase **T** 105> 38, 105
 <Representación de la clase **Vector** 89a> 12, 89a
 <Representación de un proceso de eliminación 106c> 42, 106c, 107a, 107b, 108
 <Resolviendo un **Sistema** de Ecuaciones Lineales 63b> 42, 63b
 <Resolviendo un sistema homogéneo 61> 42, 61
 <Restamos **AI** 69c> 68, 69c
 <Se guardan los atributos **tex** y **pasos** (y se muestran los pasos si se pide) 106a> 106a
 <Segundo chunk de ejemplo que cambia el último elemento de la lista **a** 121b> 121b, 122
 <Simplificación de expresiones simbólicas 85a> 42, 85a
 <Suma y resta de **Sistemas** 25b> 9, 25b
 <Sumamos **AI** 69d> 68, 69d
 <Sustitución de un símbolo por otro símbolo o valor en un **Sistema** 87d> 9, 87d
 <Sustitución de variables simbólicas en una **Transformación elemental** 36c> 36c, 38
 <Texto de ayuda de la clase **Determinante** 65a> 65a, 65b
 <Texto de ayuda de la clase **Matrix** 13> 13, 15b
 <Texto de ayuda de la clase **SEL** 63a> 63a, 63b
 <Texto de ayuda de la clase **Sistema** 5a> 5a, 9
 <Texto de ayuda de la clase **T** (**Transformación Elemental**) 31> 31, 38
 <Texto de ayuda de la clase **Vector** 10> 10, 12
 <Texto de ayuda de las transformaciones elementales de las filas de una **Matrix** 40b> 40b, 41a
 <Texto de ayuda de las transformaciones elementales de un **Sistema** 39> 39, 40a
 <Texto de ayuda para el operador producto por la derecha en la clase **Sistema** 28> 28, 29
 <Texto de ayuda para el operador producto por la izquierda en la clase **Sistema** 26a> 26a, 26b

<Texto de ayuda para el operador resta en la clase Sistema 25a> [25a](#), [25b](#)
 <Texto de ayuda para el operador selector por la derecha para la clase Matrix 19c> [19c](#), [20](#)
 <Texto de ayuda para el operador selector por la derecha para la clase Sistema 17> [17](#), [18](#)
 <Texto de ayuda para el operador selector por la izquierda para la clase Matrix 22b> [22b](#), [23](#)
 <Texto de ayuda para el operador suma en la clase Sistema 24> [24](#), [25b](#)
 <Texto de ayuda para el operador transposición de la clase Matrix 21> [21](#), [22a](#)
 <Texto de ayuda para la clase DiagonalizaC 71c> [71c](#), [72](#), [100b](#)
 <Texto de ayuda para la clase DiagonalizaCr 73c> [73c](#), [74](#), [100c](#)
 <Texto de ayuda para la clase DiagonalizaO 71a> [70](#), [71a](#), [99c](#)
 <Texto de ayuda para la clase DiagonalizaS 69a> [68](#), [69a](#), [99b](#)
 <Texto de ayuda para la composición de Transformaciones Elementales T 33c> [33c](#), [34](#)
 <Transforma una Matrix más una lista de Matrix en una BlockM diagonal 95c> [92b](#), [95c](#)
 <Transformaciones elementales de las filas de una Matrix 41a> [15b](#), [41a](#)
 <Transformaciones elementales de los elementos de un Sistema 40a> [9](#), [40a](#)
 <Transformaciones elementales por la izquierda de un Vector 41b> [12](#), [41b](#)
 <Tres métodos de eliminación por columnas 49> [42](#), [49](#), [50](#), [51](#), [52](#), [53a](#), [53b](#)
 <Tres métodos de eliminación por filas 55> [42](#), [55](#), [56](#), [57](#)
 <Uso del pivote para eliminar componentes con transformaciones Tipo I 45b> [45b](#), [52](#), [53b](#), [74](#)
 <Uso del pivote para eliminar componentes evitando dividir 46> [46](#), [49](#), [51](#), [72](#)
 <Verificación de que las abreviaturas corresponden a transformaciones elementales 33a> [32](#), [33a](#)
 <Verificación de que todas las columnas de la matriz tienen la misma longitud 91b> [15a](#), [91b](#)
 <Verificación de que todas las filas de la matriz tendrán la misma longitud 91a> [90b](#), [91a](#)
 <Verificación de que todas son listas de matrices y de la misma longitud 116b> [115](#), [116b](#)
 <Verificación de que todos los elementos de la lista son números o de tipo sympy.Basic 11b> [11a](#), [11b](#)
 <Verificación de que todos son Sistemas de matrices y de la misma longitud 116a> [115](#), [116a](#)
 <Verificamos que todas las Matrix tienen el mismo número de columnas 112b> [112a](#), [112b](#)

Licencia

125a <Copyright y licencia GPL 125a>≡
 # Copyright (C) 2019 - 2020 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.

125b <Chunk final que indica qué tipo de objeto es a y hace unas sumas 125b>≡
 type(a)
 2+2
 3+20
 This code is used in chunk 122.

