



Universidad Complutense de Madrid

---

Laboratorio de Computación Científica  
Laboratory for Scientific Computing  
Introducción a Numpy ※ Introduction to Numpy

Juan Jiménez  
Héctor García de Marina  
Lía García

2 de septiembre de 2024



El contenido de estos apuntes está bajo licencia Creative Commons Attribution-ShareAlike 4.0  
<http://creativecommons.org/licenses/by-sa/4.0/>

©Juan Jiménez

# Índice general

<b>1. Introducción a Numpy</b>	
<b>Introduction to Numpy</b>	<b>9</b>
1.1. Numpy: un paquete de Python para cálculo numérico . . . . .	9
1.1. Numpy: a Python library for scientific computing . . . . .	9
1.1.1. Vectores y matrices en Numpy. . . . .	12
1.1.1. Vectors and matrices in Numpy. . . . .	12
1.2. Operaciones matriciales . . . . .	20
1.2. Matrix Operations . . . . .	20
1.2.1. Funciones incluidas en Numpy. . . . .	41
1.2.1. Functions included in Numpy . . . . .	41
1.3. Operadores vectoriales . . . . .	41
1.3. Vectorial operators . . . . .	41
1.4. Tipos de matrices empleados frecuentemente. . . . .	54
1.4. Kinds of matrices frequently used. . . . .	54
1.5. Factorización de matrices . . . . .	56
1.5.1. Factorization LU . . . . .	56
1.5. Matrix factorization. . . . .	56
1.5.1. LU factorization. . . . .	56
1.5.2. Factorización de Cholesky . . . . .	59
1.5.2. Cholesky factorisation . . . . .	59
1.5.3. Diagonalización . . . . .	60
1.5.3. Diagonalisation . . . . .	60
1.5.4. Factorización QR . . . . .	67
1.5.4. QR Factorisation . . . . .	67
1.5.5. Factorización SVD . . . . .	68
1.5.5. SVD Factorisation . . . . .	68
1.6. Ejercicios . . . . .	70



# Índice de figuras

1.1. Representación gráfica de vectores en el plano . . . . .	11
1.1. Graphic of vectors on the plane . . . . .	11
1.2. Representación gráfica de vectores en el espacio 3D . . . . .	12
1.2. Graphic of vector in the 3D space . . . . .	12
1.3. interpretación geométrica de la norma de un vector . . . . .	30
1.3. Geometrical interpretation of the norm of a vector. . . . .	30
1.4. efecto del producto de un escalar por un vector . . . . .	42
1.4. effect of the product of a vector by a scalar . . . . .	42
1.5. Representación gráfica de los vectores $a = (1, 2)$ , $b = (-1, 1)$ y algunos vectores, combinación lineal de $a$ y $b$ . . . . .	43
1.5. A graphic representations of Vectors $a = (1, 2)$ , $b = (-1, 1)$ and some vectors obtained from a lineal combination of $a$ and $b$ . . . . .	43
1.6. Representación gráfica de los vectores $a = (1, -2, 1)$ , $b = (2, 0, -1)$ , $c = (-1, 1, 1)$ y del vector $a - b + c$ . . . . .	44
1.6. Vectors $a = (1, -2, 1)$ , $b = (2, 0 - 1)$ , $c = (-1, 1, 1)$ and vector $a - b + c$ ; a graphic representation . . . . .	44
1.7. Representación gráfica del vector $a$ , en la base canónica $\mathcal{C}$ y en la base $\mathcal{B}$ . . . . .	47
1.7. Graphic representation of vector $a$ , in the canonical basis $\mathcal{C}$ and in $\mathcal{B}$ basis. . . . .	47
1.8. Transformaciones lineales del vector $a = [1, 2]$ . $D$ , dilatación/contracción en un factor 1.5/0.5. $R_x$ , reflexión respecto al eje x. $R_y$ , reflexión respecto al eje y. $R_\theta$ rotaciones respecto al origen para ángulos $\theta = \pi/6$ y $\theta = \pi/3$ . . . . .	50
1.8. Linear transformation of vector $= [1, 1]$ . $D$ , factor 1.5/0.5 dilation/contraction. $R_y$ reflection on y-axis. $R_\theta$ $\theta = \pi/6$ and $\theta = \pi/3$ angles rotations around the origin. . . . .	50
1.9. Formas cuadráticas asociadas a las cuatro matrices diagonales: $ a_{11}  =  a_{22}  = 1$ , $a_{12} = a_{21} = 0$ . . . . .	53
1.9. Quadratic forms obtained using the four diagonal matrices: $ a_{11}  =  a_{22}  = 1$ , $a_{12} = a_{21} = 0$ . . . . .	53



# Índice de cuadros

1.1. Algunas funciones matemáticas en Numpy de uso frecuente . . . . .	40
1.1. frequently used mathematical functions included in Numpy . . . . .	40





## Capítulo/Chapter 1

# Introducción a Numpy Introduction to Numpy

When the going gets tough, the  
tough get going

---

Popular Witticism (US)

### 1.1. Numpy: un paquete de Python para cálculo nu- mérico

En el capítulo ??, vimos cómo importar módulos de python en un script o directamente en el terminal de Ipython, de modo que podamos reutilizar el código contenido en ellos. Numpy es un librería de python que contiene funciones y variables específicamente diseñadas para el cálculo numérico. Está estructurada en forma de módulos y submódulos de modo que para utilizarla en nuestros programas basta importarla en nuestros script, importar sus submodulos o importar sus funciones. La base de Numpy la constituyen objetos y operaciones algebraicas. En esta sección vamos a repasar algunos conceptos fundamentales de álgebra lineal y cómo pueden manejarse empleando Python. No daremos definiciones precisas ni tampoco demostraciones, ya que tanto unas como otras se verán en detalle en la asignatura de álgebra.

### 1.1. Numpy: a Python li- brary for scientific com- puting

In chapter ??, we have seen how to import Python modules to a script or the Ipython console. In this way, we can reuse the code available in the modules. Numpy is a Python library with functions and variables specifically intended to perform scientific computing. Numpy is structured in modules and submodules so that we can import the whole library, its submodules, or just any of its functions into our scripts. The background of Numpy is built of algebraic objects and operations. In this section, we will review some fundamental notions of linear algebra and how to deal with them using Numpy. We will not provide formal definitions or demonstrations because both will be studied in detail in algebra during the next term.

**Matrices** From a functional perspective, we will define a matrix as a table of numbers ordered by rows and columns,

**Matrices.** Desde un punto de vista funcional definiremos una matriz como una tabla bidimensional de números ordenados en filas y columnas,

$$A = \begin{pmatrix} 1 & \sqrt{2} & 3.5 & 0 \\ -2 & \pi & -4.6 & 4 \\ 7 & -19 & 2.8 & 0.6 \end{pmatrix}$$

Cada línea horizontal de números constituye una *fila* de la matriz y cada línea vertical una *columna* de la misma.

A una matriz con  $m$  filas y  $n$  columnas se la denomina matriz de orden  $m \times n$ .  $m$  y  $n$  son las dimensiones de la matriz y se dan siempre en el mismo orden: primero el número de filas y después el de columnas. Así, la matriz  $A$  del ejemplo anterior es una matriz  $3 \times 4$ , y como esta formada por números reales se dice que  $A \in \mathbb{R}^{3 \times 4}$ . El orden de una matriz expresa el tamaño de la matriz.

Dos matrices son iguales si tienen el mismo orden, y los elementos que ocupan en ambas matrices los mismo lugares son iguales.

Una matriz es cuadrada, si tiene el mismo número de filas que de columnas. Es decir es de orden  $n \times n$ .

Mientras no se diga expresamente lo contrario, emplearemos letras mayúsculas  $A, B, \dots$  para representar matrices. La expresión  $A_{m \times n}$  indica que la matriz  $A$  tiene dimensiones  $m \times n$ . Para denotar los elementos de una matriz, emplearemos la misma letra en minúsculas empleada para nombrar la matriz, indicando mediante subíndices, y siempre por este orden, la fila y la columna a la que pertenece el elemento. Así por ejemplo  $a_{ij}$  representa al elemento de la matriz  $A$ , que ocupa la fila  $i$  y la columna  $j$ .

Each horizontal line of numbers forms a matrix row, and each vertical line a matrix column. A matrix with  $m$  rows and  $n$  columns is denoted as a matrix of order  $m \times n$ .  $m$  and  $n$  are the matrix dimensions, and they are always defined in the same order: first, the number of rows and then the number of columns. So matrix  $A$  in the example above is a  $3 \times 4$  matrix, and as it is built of real numbers, we say that  $A \in \mathbb{R}^{3 \times 4}$ . The matrix order defines the size of the matrix.

Two matrices are equal if they have the same order and the entries located in both matrices in the same place are equal. A matrix is square whenever it has the same number of rows and columns. That is, it is an  $n \times n$  matrix.

From now on, we will use capital letters  $A, B, \dots$  to name matrices. The expression  $A_{m \times n}$  means that matrix  $A$  has dimensions  $m \times n$ . We will refer to the entries of a matrix using the same letter used to name the matrix but in lowercase and using subindexes to indicate the row and column the entry belongs to. The first subindex always represents the row, and the second is the column. For instance,  $a_{ij}$  Represents the matrix  $A$  entry that belongs to row  $i$  and to column  $j$ .

$$A = \begin{pmatrix} 1 & \sqrt{2} & 3.5 & 0 \\ -2 & \pi & -4.6 & 4 \\ 7 & -19 & 2.8 & 0.6 \end{pmatrix} \rightarrow a_{23} = -4.6$$

**vectores** A una matriz compuesta por una sola fila, la denominaremos vector fila. A una matriz compuesta por una sola columna la denominaremos vector columna. Siempre que hablemos de un vector, sin especificar más, entenderemos que se trata de un vector colum-

We call a matrix formed by a single row a row vector. We call a matrix formed by a single column a column vector. Whenever we speak of a vector without further specification,

na.<sup>1</sup> Para representar vectores, emplearemos letras minúsculas. Para representar sus elementos añadiremos a la letra que representa al vector un subíndice indicando la fila a la que pertenece el elemento.

<sup>1</sup>Esta identificación de los vectores como vectores columna no es general. La introducimos porque simplifica las explicaciones posteriores.

we consider it a column vector.<sup>1</sup> To name vectors, we will use lowercase letters. To name the entries of a vector, we will add a subindex to the letter representing the vector to show the row the entry belongs to.

<sup>1</sup>This identification of a vector with a column vector is by no means general. We introduce it because it simplifies the exposition.

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_i \\ \vdots \\ a_n \end{pmatrix}$$

Podemos asociar los puntos del plano con los vectores de dimensión dos. Para ello, usamos una representación cartesiana, en la que los elementos del vector son los valores de las coordenadas  $(x, y)$  del punto del plano que representan. Cada vector se representa gráficamente mediante una flecha que parte del origen de coordenadas y termina en el punto  $(x, y)$  representado por el vector. La figura 3.1 representa los vectores,

We can establish a relationship between the plane points and two-dimensional vectors. To do so, we use a Cartesian representation, in which the entries of a vector are the  $(x, y)$  coordinates of the point of the plane they represent. Each vector is drawn using an arrow that starts at the origin of the coordinates and ends at the point  $(x, y)$  represented by the vector. Figure 3.1 represents vectors,

$$a = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, b = \begin{pmatrix} 2 \\ -3 \end{pmatrix}, c = \begin{pmatrix} 0 \\ -2 \end{pmatrix}$$

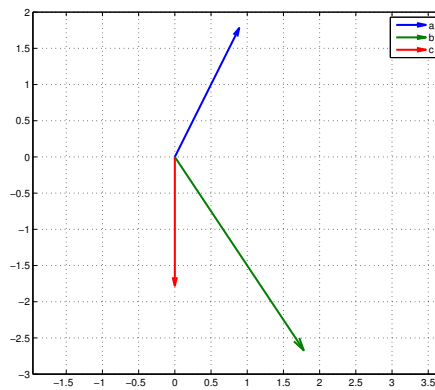


Figura 1.1: Representación gráfica de vectores en el plano  
Figure 1.1: Graphic of vectors on the plane

De modo análogo, podemos asociar vectores de dimensión tres con puntos en el espacio tridimensional. En este caso, los valores de los elementos del vector corresponden con la coordenadas  $(x, y, z)$  de los puntos en el espacio. La figura 3.2 muestra la representación gráfica en espacio tridimensional de los vectores,

Likewise, we can associate vectors of dimension three with points in the 3D space. In this case, the vector entries represent the coordinates  $(x, y, z)$  of the points in the space. Figure 3.2 shows a graphic representation of vectors,

$$a = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, b = \begin{pmatrix} 2 \\ -3 \\ -1 \end{pmatrix}, c = \begin{pmatrix} 0 \\ -2 \\ 1 \end{pmatrix}$$

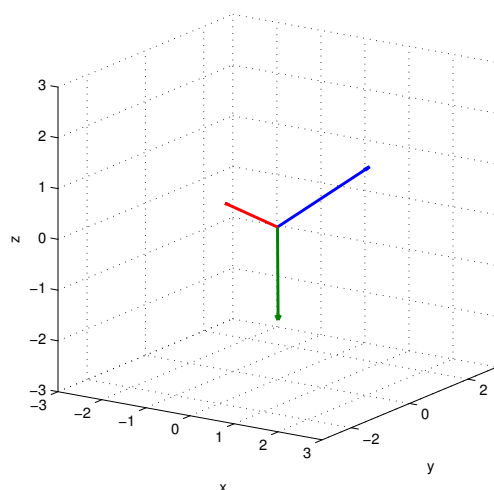


Figura 1.2: Representación gráfica de vectores en el espacio 3D  
Figure 1.2: Graphic of vector in the 3D space

Evidentemente para vectores de mayor dimensión, no es posible obtener una representación gráfica. Si embargo muchas de las propiedades geométricas, observables en los vectores bi y tridimensionales, pueden extrapolarse a vectores de cualquier dimensión.

Obviously, it is impossible to get a graphic representation for vectors of larger dimensions. Nevertheless, many geometrical properties owned by 2D and 3D vectors can be applied to vectors of whatever dimension.

### 1.1.1. Vectores y matrices en Numpy.

**Matrices.** Una de las característica más interesantes de Numpy, es la posibilidad de crear fácilmente matrices. Se pueden crear de diferentes maneras, la más elemental de todas ellas, emplea la función de Numpy `array` aplicada a una lista de filas de la matriz que se quiere construir. Cada fila debe ser a su vez

### 1.1.1. Vectors and matrices in Numpy.

**Matrices.** One interesting feature of numpy is that it allows us to create matrices easily. There are several methods to create them, but using the Numpy function `array` with a Python list with the rows of the matrix we want to build is probably the easiest method. Each row should be, in turn, a list of numbers, i.e., matrix entries. Of course, to build a matrix, all

una lista de números. Evidentemente, para que se pueda construir la matriz, todas las filas deben tener el mismo número de elementos. El siguiente ejemplo muestra como construir una matriz de dos filas y tres columnas,

rows should have the same number of entries. The following example shows how to build a matrix of two rows and three columns.

```
In [3]: import numpy as np

In [4]: A = np.array([[1,2,3],[4,5,6]])

In [5]: print(A)
[[1 2 3]
 [4 5 6]]

In [6]: L = [[1,2,3],[4,5,6]]

In [7]: B = np.array(L)

In [9]: print(L)
[[1, 2, 3], [4, 5, 6]]

In [10]: print(B)
[[1 2 3]
 [4 5 6]]
```

Lo primero que hacemos es importar Numpy, lo importamos usando como alias la abreviatura `np`, porque es más cómodo a la hora de llamar a funciones específicas de Numpy. Hemos construido dos matrices iguales `A` y `B`. En el primer caso hemos creado directamente dentro de la llamada a la función `array`, la lista a partir de la cual construimos la matriz. En el segundo caso, hemos construido primero una lista `L`, y luego hemos empleado dicha lista como variable de entrada de la función `array`. Si nos fijamos en las líneas [9] y [10], vemos como Python distingue la lista —todos sus elementos aparecen representados en la misma línea—, de la matriz en la que cada fila ocupa una línea distinta. Podemos construir matrices a partir de listas y variables ya definidas, siempre que seamos coherentes con el criterio de que cada fila se cree a partir de una lista y que todas las filas tengan los mismos elementos,

First, we import Numpy; we do it using `np` as an alias. The reason is that `np` is shorter than `numpy`, and it eases the calls to specific Numpy functions. We have built two identical matrices `a` and `B`. In the first case, we have straightforwardly created the list needed to make the matrix inside the call to the function `array`. In the second case, we first created a list `L`, and then we used it as an input variable to the function `array`. Focusing on lines [9] and [10], we see how Python can tell a list and a matrix apart. For a list, Python represents all entries in the same line. For a matrix, each row is written in a different line. We can build matrices from lists and variables already defined as long as we follow the criteria that each row be created from a list and all rows have the same number of entries,

```
In [23]: a =1; b= 2; c =3

In [24]: d =[4,5,6]
```

```

In [25]: C = np.array([[a,b,c],d])

In [26]: print(C)
[[1 2 3]
 [4 5 6]]

In [27]: C = np.array([[a,b,c],d,[7,8,9]])

In [28]: print(C)
[[1 2 3]
 [4 5 6]
 [7 8 9]]

```

**Indexación.** Al igual que se hace en álgebra, Numpy es capaz de referirse a un elemento cualquiera de una matriz empleando índices para determinar su posición (fila y columna) dentro de la matriz.

**Indexing** As in algebra, in Numpy is also possible to refer any entry of a matrix using indexes to indicate its position (row and column) in the matrix.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Sin embargo, el criterio para referirse a un elemento concreto de una matriz, en Numpy está heredado de las listas: se indica el nombre de la variable que contiene la matriz y a continuación, entre corchetes y separados por una coma, el índice de su fila y después el de su columna **pero empezando a contar desde 0**. Es decir, la primera fila de una matriz de dimensión  $m \times n$  es la fila 0 y la última es la fila  $m - 1$ . De modo análogo su primera columna es la 0 y su última columna es la  $n - 1$ ,

However, the Numpy criteria to refer to a specific entry inside a matrix has been borrowed from Python Lists: we write the name of the matrix followed by the row and column indexes of the entry we are interested in, enclosed in a square bracket and separated by a comma. The point is that we **count the rows and columns starting at 0**. Thus, the first row of a matrix with dimensions  $m \times n$  is the row 0, and the last is the row  $m - 1$ . Likewise, its first column is column 0, and the last one is column  $n - 1$ ,

```

In [31]: print(C)
[[1 2 3]
 [4 5 6]
 [7 8 9]]

In [32]: C[1,2]
Out[32]: 6

In [33]: C[0,0]
Out[33]: 1

```

Numpy puede seleccionar dentro de una matriz no solo elementos aislados, sino también submatrices completas. Para ello, emplea un símbolo reservado, el símbolo *dos puntos*

Inside a matrix, Numpy can select single entries and whole submatrices. To do so, it uses the colon : symbol as a special symbol. Using a fixed step, we use this symbol to co-

∴ Este símbolo se emplea para recorrer valores desde un valor inicial hasta un valor final, con un incremento o paso fijo. La sintaxis es: **inicio:fin:paso**. Es importante tener en cuenta que Numpy detendrá la cuenta en el valor **stop-step**. Además, si no indicamos el tamaño del paso, Numpy tomará por defecto un paso igual a uno. En este caso basta emplear **start:stop**

ver a set of values from a start (initial) value to a stop (final) one. The syntax is simple: **start:stop:step**. Beware! Numpy will stop the count in the step **stop-step**. Besides, if we leave apart the step size, Numpy will take a step equal to one. In this case, the expression is just **start:stop**

```
In [87]: D = numpy.array([[1.,2.,3.],[4.,5.,6.],[-2,-3,0],[3,2,1]])
```

```
In [88]: print(D)
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [-2. -3.  0.]
 [ 3.  2.  1.]]
```

```
In [90]: D[0:1,0:3]
Out[90]: array([[1., 2., 3.]])
```

```
In [91]: D[0:2,0:2]
Out[91]:
array([[1., 2.],
       [4., 5.]])
```

```
In [92]: D[2:3,1:2]
Out[92]: array([[ -3.]])
```

```
In [93]: D[3:4,0:3]
Out[93]: array([[3., 2., 1.]])
```

```
In [94]: D[0:4,0:3]
Out[94]:
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [-2., -3.,  0.],
       [ 3.,  2.,  1.]])
```

```
In [95]: D[0:4,2:3]
Out[95]:
array([[3.],
       [6.],
       [0.],
       [1.]])
```

La línea [90], extrae un vector fila con los elementos de la primera fila de la matriz D. La fila [91] extrae una matriz de dimension  $2 \times 2$  con los cuatro elementos de la esquina superior

Line [90] extracts a row vector from the first row of matrix D. Line [91] extracts a  $2 \times 2$  matrix using the four entries located on the left-up corner of the original, D, matrix. Line

izquierda de la matriz original. La fila [92] extrae un único elemento pero sigue siendo una matriz de dimensión  $1 \times 1$ , por tanto es distinto que si empleamos la indexación directa del elemento: `D[2,1]`. La línea [93] nos devuelve un vector fila con la última fila de la matriz. La línea [94], nos devuelve de nuevo la matriz entera. Por último la línea [95] nos devuelve un vector columna con la primera columna de la matriz.

Hemos dicho que la línea [95] nos devuelve un vector columna. Bueno, sí y no; vamos a verlo más despacio.

**Vectores.** Cuando introducimos los vectores, distinguimos entre vectores filas y columna, definiéndolos como matrices de una sola fila o una sola columna. Sin embargo, en Numpy, se sigue un criterio distinto que permite generalizar el concepto de matriz asociándolo con el de tensor. Sin entrar en detalles<sup>2</sup>, podemos decir que un tensor es un objeto algebraico caracterizado por dos parámetros; el orden y la dimensión. Así un escalar  $a \in \mathbb{R}$  es un tensor de orden cero. Un vector  $b \in \mathbb{R}^n$  es un tensor de orden 1 y dimensión  $n$ . Una matriz  $A \in \mathbb{R}^{m \times n}$  es un tensor de orden dos y dimensiones  $m, n$ . Un tensor de orden 3,  $T \in \mathbb{R}^{n \times m \times l}$ , etc. Podemos asociar el orden al número mínimo de índices que necesitamos para definir de forma unívoca los elementos de un Tensor: para un escalar, no nos hace falta ningún índice, solo tenemos un elemento que es el propio escalar, por tanto le asociamos orden cero. Para un vector es suficiente emplear un índice para definir sus elementos,  $b = (b_i)$ ,  $i = 1, \dots, n$ ,  $b \in \mathbb{R}^n$ . Para una matriz necesito dos índices,  $A = (a_{ij})$ ,  $i = 1, \dots, n, j = 1, \dots, m$ ,  $A \in \mathbb{R}^{n \times m}$ . Para un tensor de orden tres necesitaría tres índices,  $T = (T_{ijk})$ ,  $i = 1, \dots, n, j = 1, \dots, m, k = 1, \dots, l$ ,  $T \in \mathbb{R}^{n \times m \times l}$  y así sucesivamente. Numpy permite definir estructuras de cualquier orden y dimensión que queramos. Pero nosotros nos vamos a limitar a vectores (orden 1) y matrices (orden 2). ¿Tiene sentido entonces distinguir entre vectores fila y columna? So-

[92] extracts a single matrix entry, but notice that it is a  $1 \times 1$  matrix. Thus, this result is different than the result achieved when we directly use the indexes of the entry: `D[2,1]`. Line [93] returns a row vector with the matrix's last row entries. Lastly, line [95] returns a column vector with the matrix's last column entries.

We have said that line [95] returns a column vector. Well, yes and no; let's examine it in more detail.

**Vectors.** When we introduced vectors in the previous section, we distinguished between row and column vectors, defining them as matrices with a single row or column. Nevertheless, Numpy follows another criterion that allows us to generalise the idea of matrix, linking it with the concept of tensor. we do not get into details<sup>2</sup> and simply say that a tensor is an algebraic object defined by two parameters, its order, and its dimension. So, a scalar  $a \in \mathbb{R}$  is a tensor of order zero. A vector  $b \in \mathbb{R}^n$  is a tensor of order one and dimension  $n$ . A matrix  $A \in \mathbb{R}^{m \times n}$  is a tensor of order two and dimensions  $m, n$ . A third order tensor,  $T \in \mathbb{R}^{n \times m \times l}$ , etc. We can relate the order with the minimum number of indexes we need to univocally define the tensor entry: for a scalar, we don't need an index at all; we have only a single entry, the scalar itself. For this reason, we associate scalars with a zero-order tensor. For a vector, it is enough to use a single index to define its entries,  $b = (b_i)$ ,  $i = 1, \dots, n$ ,  $b \in \mathbb{R}^n$ . For a matrix, we need two indexes,  $A = (a_{ij})$ ,  $i = 1, \dots, n, j = 1, \dots, m$ ,  $A \in \mathbb{R}^{n \times m}$ . For third-order tensors, we need three indexes,  $T = (T_{ijk})$ ,  $i = 1, \dots, n, j = 1, \dots, m, k = 1, \dots, l$ ,  $T \in \mathbb{R}^{n \times m \times l}$  and so on. In Numpy, we can define structures of whatever order and dimensions we want, but we will only use vectors (order 1) and matrices (order 2). Has, then, any sense to distinguish between row and column vector? Only if we always consider vectors as matrices (order 2) and dimensions  $1 \times n$  (row vector) or  $n \times 1$  (column vector). For Numpy, however, vectors and matrices are

<sup>2</sup>Una definición formal del concepto de tensor, queda fuera del alcance de estos apuntes.

<sup>2</sup>A formal definition of tensors is far beyond the scope of these notes.



lo si consideramos siempre los vectores como matrices (orden 2) y dimensiones  $1 \times n$  (vector fila) ó  $n \times 1$  (vector columna). Para Numpy, sin embargo, vectores y matrices son estructuras de distinto orden. Veamos con algunos ejemplo cómo se diferencian. Para verlo mejor podemos emplear la propiedad **shape** de los arrays en numpy. Dicha propiedad nos devuelve una tupla con las dimensiones del array, el número de elementos que contine la tupla nos da el orden,

structures of different order. Let's see some examples of their differences. To appreciate it better, we can use the Numpy arrays attribute **shape**, which gives us a tuple containing the dimensions of the array. The number of entries of the tuple tells us the order of the array,

```
In [23]: D
Out[23]:
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [-2., -3.,  0.],
       [ 3.,  2.,  1.]])
```

```
In [24]: D.shape
Out[24]: (4, 3)
```

```
In [25]: D[1,1]
Out[25]: 5.0
```

```
In [26]: D[1,1].shape
Out[26]: ()
```

```
In [27]: D[1,1:2]
Out[27]: array([5.])
```

```
In [28]: D[1,1:2].shape
Out[28]: (1,)
```

```
In [29]: D[1:2,1:2]
Out[29]: array([[5.]])
```

```
In [30]: D[1:2,1:2].shape
Out[30]: (1, 1)
```

Empezamos con la matriz D de ejemplos anteriores. Para obtener sus dimensiones empleamos **D.shape**. El resultado es una tupla compuesta de dos elementos, puesto que es una matriz y, por tanto su orden es dos. El primer elemento no da la dimensión de sus columnas, es decir, el número de filas. El segundo elemento nos da la dimensión de sus filas, es decir el número de columnas.

En la línea [25] extraemos el elemento que ocupa la posición [1, 1]. En la [26] cuando tra-

We begin with the same matrix D we use in previous examples. To get its dimensions, we use **D.shape**. The result is a tuple of two entries because it is a matrix, and thus, its order is two. The first entry gives us its column dimension, that is, its number of rows. The second entry is its row dimension, that is, its number of columns.

In line [25] we extract the entry located at position [1, 1]. In line [26] when we try to get its dimension, Numpy returns an empty tu-

tamos de obtener sus dimensiones, nos da una tupla vacía, porque es un escalar y su orden es cero. En la línea [27] le hemos pedido a Numpy que nos de los elementos de la fila 1 de la matriz `D` que ocupan las columnas desde la 1 hasta la 1. Es decir, hacemos referencia al mismo elemento de la matriz, sin embargo el resultado no es exactamente el mismo. Nos ha devuelto un array con el elemento seleccionado. Cuando en la línea [28] pedimos sus dimensiones, obtenemos una tupla con un único elemento `(1,)`. Es decir, el orden del array es 1, se trata de un vector, y tiene de dimensión 1, el vector solo tiene un elemento.

Por último, en la línea [29] volvemos a pedir a Python que nos de los elementos de la matriz `D`, que ocupan las filas desde la 1 hasta la 1 y las columnas desde la 1 hasta la 1, el resultado es ahora una matriz, podemos ver en la línea `out` [29] que el número 5 aparece ahora encerrado entre dos pares de corchetes. Cuando en la línea [30], preguntamos por su `shape`, nos devuelve una tupla con dos elementos –el orden del array es 2, puesto que se trata de una matriz– y sus dimensiones son una fila y una columna, puesto que la matriz solo tiene un elemento.

Vamos a completar nuestro estudio de los arrays en Python, extrayendo ahora partes más grandes de la misma matriz `D`,

ple because it is a scalar and its order is zero. In line [27], we ask Numpy that retrieve the entries of row 1 of the matrix `D` that fill the columns 1 to 1. Thus, we are making reference to the same entry of the matrix as in the previous case. However, the result is not exactly the same. Numpy retrieves an array with the selected entry. When we ask for its dimensions in line [28] we get a tuple with a single entry `(1,)`. Then, the order of the array is one. It is a vector with a single entry.

Finally, in line [29], we ask Numpy to retrieve the entries of matrix `D` with fill the rows from 1 to 1 and the columns from 1 to 1. Thus, the result is now a matrix, We can see in line `out` [29] that the number 5 is enclosed in two pair of square brackets. When in line [30], we ask for its `shape`, we get a tuple with two entries –now the order of the array is two, because it is a matrix– and their dimension are one row and one column as far as the matrix has a single entry.

We are going to complete our study on Numpy arrays, extracting larger parts of the same matrix,

```
In [9]: D
Out[9]:
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [-2., -3.,  0.],
       [ 3.,  2.,  1.]])

In [10]: D[1,:]
Out[10]: array([4., 5., 6.])

In [11]: D[:,1]
Out[11]: array([ 2.,  5., -3.,  2.])

In [12]: D[1:2,:]
Out[12]: array([[4., 5., 6.]])
```

```
In [13]: D[:,1:2]
Out[13]:
array([[ 2.],
       [ 5.],
       [-3.],
       [ 2.]])
```

En el primer caso, línea [10], hemos extraído toda la segunda fila de la matriz D, el resultado es un vector, por tanto tiene orden 1 y dimensión 3. En la línea [11] hemos extraído la primera columna y el resultado es de nuevo un vector, por tanto tiene orden 1 y la dimensión esta vez 4. En la línea [12] extraemos todas las columnas de las filas que van desde la segunda hasta la segunda. El resultado es una matriz, ya que el orden del array extraído es 2, y la dimensiones será 1 para las filas y 3 para las columna. Es lo más parecido a un vector 'fila' que podemos obtener con Numpy. Por último, en la línea [13], extraemos todas las filas de las columnas que van desde la segunda hasta la segunda. El resultado es de nuevo una matriz, porque el array extraído tiene orden dos, pero las dimensiones son ahora 4 para las filas y 1 para las columnas, lo podemos identificar con un vector columna de los descritos antes.

Evidentemente, podemos también extraer de una matriz bloque (submatrices), indicando las filas y columnas que queremos extraer de la matriz original. por ejemplo,

In the first case, line [10] we get the whole second row of matrix D. The result is a vector and has order 1 and dimension 3. In line [11], we extract the first column of the matrix, and the result is again a vector. This time the order is 1, and the dimension is 4. In line [12] we extract all columns belonging to rows second to second. The result is a matrix because the order of the extracted array is two. The dimensions are 1 for the rows and 3 for the columns. It is the most similar to a 'row' vector we can get using Numpy. Lastly, in line [13], we extract all the rows belonging to columns second to second. the result is again a matrix because the array obtained has order two, but the dimensions are now 4 for the rows and 1 for the columns, we could identify it as a column vector of those described above.

Indeed, we can also extract a block matrix (submatrices), using indexes to define the rows and columns we want to obtain from the original matrix,

```
In [20]: D
Out[20]:
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [-2., -3.,  0.],
       [ 3.,  2.,  1.]])
```

```
In [21]: D[1:4,1:3]
Out[21]:
array([[ 5.,  6.],
       [-3.,  0.],
       [ 2.,  1.]])
```

```
In [22]: D[1:3,0:2]
Out[22]:
array([[ 4.,  5.],
       [-2., -3.]])
```

## 1.2. Operaciones matriciales

A continuación definiremos las operaciones matemáticas más comunes, definidas sobre matrices. Vamos a empezar por aquellas que se realizan elemento a elemento, entre aquellos elementos que ocupan la misma posición en las matrices que se operan,

**Suma.** La suma de dos matrices, se define como la matriz resultante de sumar los elementos que ocupan en ambas la misma posición. Solo está definida para matrices del mismo orden,

## 1.2. Matrix Operations

In this section we will define the most common mathematical operation for matrices. We are going to begin for those operation that are carried out, entry by entry, between those entries which occupy the same place in the operating matrices.

**Addition.** Addition of two matrix, the result is a new matrix obtained adding the entries which occupy the same position in both matrices. It is defined only for matrix of the same dimensions,

$$C = A + B$$

$$c_{ij} = a_{ij} + b_{ij}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 5 \\ 3 & 5 & 7 \\ 5 & 7 & 9 \end{pmatrix} + \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$$

La suma de matrices cumple las siguientes propiedades,

1. Asociativa:  $(A + B) + C = A + (B + C)$
2. Conmutativa:  $A + B = B + A$
3. Elemento neutro:  $O_{n \times m} + A_{n \times m} = A_{n \times m}$   
El elemento neutro  $O_{n \times m}$  de la suma de matrices de orden  $n \times m$  es la matriz nula de dicho orden, —compuesta exclusivamente por ceros— .
4. Elemento opuesto: La opuesta a una matriz se obtiene cambiando de signo todos sus elementos,  $A_{op} = -A$

En numpy el signo + se también utiliza para representar la suma de matrices, por lo que la suma de dos matrices puede obtenerse directamente como,

Matrix addition fulfil the following properties,

1. Associative:  $(A + B) + C = A + (B + C)$
2. Commutative:  $A + B = B + A$
3. Identity element:  $O_{n \times m} + A_{n \times m} = A_{n \times m}$   
The identity element  $O_{n \times m}$  of the addition of  $n \times m$  matrices is the null matrix of this dimensions, —an only zeros matrix— .
4. Inverse: we get the addition inverse of a matrix changing the signs of all its entries  $A_{inv} = -A$

In numpy, the symbol + also represents the matrix addition. Thus, you can use it to add two matrices in the same way you use it to add two numbers,

```

In [0]: import numpy as np
In [1]: A = np.array([[1,3,5],[2,4,6]])
In [2]: A
Out[2]:
array([[1, 3, 5],
       [2, 4, 6]])

In [3]: B = np.array([[3,-2,0],[1,-4,3]])

In [4]: A+B
Out[4]:
array([[4, 1, 5],
       [3, 0, 9]])

```

En numpy, podemos crear una matriz de cualquier orden, compuesta exclusivamente por ceros mediante el comando `numpy.zeros((m,n))`, donde  $m$  es el número de filas y  $n$  el de columnas de la matriz de ceros resultante. Si damos un único valor, `numpy.zeros(n)`, obtendremos un vector formado por  $n$  ceros.

We can use the Numpy command `numpy.zeros((m,n))` to create a matrix, of whatever dimension, with all its entries equal to zero.  $m$  stands for the number of rows and  $n$  for the number of columns of the zero matrix we want to create. If we use the command `numpy.zeros(n)` with a single value instead a tuple, then we will obtain a vector built up of  $n$  zeros..

```

In [375]: np.zeros(3)
Out[376]: array([0., 0., 0.])

In [377]: np.zeros((3,1))
Out[378]:
array([[0.],
       [0.],
       [0.]])

In [379]: np.zeros((1,3))
Out[380]: array([[0., 0., 0.]])

In [381]: np.zeros((3,3))
Out[382]:
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])

In [383]: Z = np.zeros((2,3))

In [384]: A+Z
Out[385]:
array([[1., 3., 5.],
       [2., 4., 6.]])

```

```
In [386]: Aop = -A
```

```
In [387]: A+Aop
```

```
Out[388]:
array([[0, 0, 0],
       [0, 0, 0]])
```

**Multiplicación elemento a elemento.** No es propiamente una operación matricial. Dadas dos matrices  $A$  y  $B$  de las mismas dimensiones, si las multiplicamos elemento a elemento, obtendremos una nueva matriz  $C$  tal que,  $c_{i,j} = a_{i,j}b_{i,j}$ . Al igual que la suma, el producto elemento a elemento es asociativo y conmutativo. En Numpy el símbolo para la multiplicación elemento a elemento es el asterisco `*`,

**entry-wise multiplication** This is not a proper matrix (algebraic) operation. You can get the entry-wise product of two matrices,  $A$  and  $B$ , with the same dimensions to obtain a new matrix  $C$  which entries are  $c_{i,j} = a_{i,j}b_{i,j}$ . This product, like the matrix addition, is commutative and asociative. In numpy the entry-wise multiplication symbol is the asterisk `*`,

```
In [49]: A
Out[49]:
array([[1, 2],
       [2, 0],
       [3, 5]])
```

```
In [50]: B
Out[50]:
array([[ -3,  1],
       [ 0,  2],
       [ 1, -4]])
```

```
In [51]: A*B
Out[51]:
array([[ -3,  2],
       [ 0,  0],
       [ 3, -20]])
```

**División elemento a elemento.** Es análoga a la multiplicación que acabamos de ver. Si dividimos elemento a elemento una matrix  $A$  entre otra matrix  $B$ , ambas de las mismas dimensiones, obtenemos una matrix  $C$  cuyos elementos cumplen  $c_{ij} = a_{ij}/b_{ij}$ . El símbolo que se emplea es el mismo de la división ordinaria entre números.

En el ejemplo siguiente, dividimos entre sí las dos matrices del ejemplo anterior, es interesante observar como Python nos advierte de la división entre cero, y asigna al elemento en que se produce el valor `inf`.

**entry-wise division.** It is verymuch alike the elemet-wise product. If we divide entry-wise a matrix  $A$  by another matrix  $B$ , both of the same dimensions, we get a new matrix  $C$  which entries are  $c_{ij} = a_{ij}/b_{ij}$ . For matrix entry-wise division, we use the same symbol as in the ordinamry number division.

In the following example, we divide the two matrix of the previous example, it is interesting to see how Python warnings us of a division by zero, and assigns to the resulting entry the value `inf`.

```
In [52]: A/B
/tmp/ipykernel_10098/713994841.py:1: RuntimeWarning: divide by zero encountered
in divide A/B
Out[52]:
array([[ -0.33333333,  2.          ],
       [          inf,  0.          ],
       [  3.          , -1.25        ]])
```

**Transposición.** Dada una matriz  $A$ , su transpuesta  $A^T$  se define como la matriz que se obtiene intercambiando sus filas con sus columnas.

**Transposition.** The transpose matrix  $A^T$  of a matrix  $A$  is the matrix we obtain interchanging its rows and columns.

$$A \rightarrow A^T$$

$$a_{ij} \rightarrow a_{ji}$$

$$A = \begin{pmatrix} 1 & -3 & 2 \\ 2 & 7 & -1 \end{pmatrix} \rightarrow A^T = \begin{pmatrix} 1 & 2 \\ -3 & 7 \\ 2 & -1 \end{pmatrix}$$

En numpy, la operación de transposición se indica mediante un punto y la letra T, A.T. Solo tiene sentido aplicarla para matrices; si transponemos un vector volvemos a obtener el mismo vector.

In Numpy the traspose matrix is obtained adding a point an the letter T to the matrix we wish to transpose, A.T. It only has sense for matrices. If we try to transpose a vector wi will optain the same vector again.

```
In [375]: A
Out[375]:
array([[1, 3, 5],
       [2, 4, 6]])
```

```
In [376]: A.T
Out[376]:
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
In [377]: B = np.array([1,2,3])
```

```
In [378]: B.T
Out[378]: array([1, 2, 3])
```

```
In [379]: C = np.array([[1,2,3]])
```

```
In [380]: C
Out[380]: array([[1, 2, 3]])
```

```
In [381]: C.T
Out[381]:
array([[1],
       [2],
       [3]])
```

Una matriz cuadrada se dice que es simétrica si coincide con su transpuesta,

A square matrix is antisymmetric if it is equal to its traspose matrix,

$$A = A^T$$

$$a_{ij} = a_{ji}$$

$$A = A^T = \begin{pmatrix} 1 & 3 & -3 \\ 3 & 0 & -2 \\ -3 & -2 & 4 \end{pmatrix}$$

Una matriz cuadrada es antisimétrica cuando cumple que  $A = -A^T$ . Cualquier matriz cuadrada se puede descomponer en la suma de una matriz simétrica más otra antisimétrica.

A square matrix is antisymmetric when it meets that  $A = A^T$ . Any square matrix can be split in the sum of two matrix, one simetric and another antisymmetric

The simetric part can be defined as,

La parte simétrica puede definirse como,

$$A_S = \frac{1}{2} (A + A^T)$$

y la parte antisimétrica como,

An the antisymmetric part as,

$$A_A = \frac{1}{2} (A - A^T)$$

Así, por ejemplo,

For instance,

$$A = A_S + A_A \rightarrow \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 5 \\ 3 & 5 & 7 \\ 5 & 7 & 9 \end{pmatrix} + \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$$

Por último, la transpuesta de la suma de matrices cumple,

The transpose of the addition of matrix meets,

$$(A + B)^T = A^T + B^T$$

### Producto de una matriz por un escalar.

El producto de una matriz  $A$  por un número  $b$  es una matriz del mismo orden que  $A$ , cuyos elementos se obtienen multiplicando los elementos de  $A$  por el número  $b$ ,

**Product of a matrix and a scalar.** The product of a matrix  $A$  for a number  $b$  is a matrix with the same dimensions as  $A$ . We obtain the entry of this product multiplying the elemetns of  $A$  by the number  $b$ ,

$$C = b \cdot A \rightarrow c_{ij} = b \cdot a_{ij}$$

$$3 \cdot \begin{pmatrix} 1 & -2 & 0 \\ 2 & 3 & -1 \end{pmatrix} = \begin{pmatrix} 3 & -6 & 0 \\ 6 & 9 & -3 \end{pmatrix}$$



En Python, el símbolo  $*$  se emplea también para representar el producto de una matriz por un número

In Python we also use the symbol  $*$  to represent the product of a matrix by a number.

```
In [391]: A
Out[391]:
array([[ 3, -5, -2,  1],
       [ 2,  3,  4,  5]])

In [394]: A*5
Out[394]:
array([[ 15, -25, -10,  5],
       [ 10,  15,  20,  25]])
```

**Producto escalar de dos vectores.** Dados dos vectores de la misma dimensión  $m$  se define su producto escalar como,

**Dot product of two vectors** For two vectors of the same dimension, we define the the dot product as,

$$a \cdot b = \sum_{i=1}^n a_i b_i$$

$$\begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix} = 1 \cdot 1 + 3 \cdot (-2) + 4 \cdot 0 = -5$$

El resultado de producto escalar de dos vectores, es siempre un número; se multiplican los entre sí los elementos de los vectores que ocupan idénticas posiciones y se suman los productos resultantes.

The dot product result is always a number; we multiply the entry which take the same place in both vectors and then, we sum the resulting products.

**Producto matricial** El producto de una matriz de orden  $n \times m$  por una matriz  $m \times l$ , es una nueva matriz de orden  $n \times l$ , cuyos elementos se obtiene de acuerdo con la siguiente expresión,

**Matrix product** The matrix product of a  $n \times m$  matrix by a  $m \times l$  matrix is a new matrix which dimensions are  $n \times l$ , We obtain the entries of the resulting matrix according with the following expression,

$$P = A \cdot B \rightarrow a_{ij} = \sum_{t=1}^m a_{it} b_{tj}$$

Por tanto, el elemento de la matriz producto que ocupa la fila  $i$  y la columna  $j$ , se obtiene multiplicando por orden los elementos de la fila  $i$  de la matriz  $A$  con los elementos correspondientes de la columna  $j$  de la matriz  $B$ , y sumando los productos resultantes

So, we get the entry of the product matrix, which occupies the row  $i$  and the column  $j$ , by multiplying in turn the entries of the row  $i$  of matrix  $A$  with the entries of the column  $j$  of matrix  $B$  and adding up the resulting products.

Para que dos matrices puedan multiplicarse es imprescindible que el número de columnas de la primera matriz coincida con el número

Two matrix can be multiplied only if the number of columns in the first matrix dimension is equal to the second matrix row dimension

ro de filas de la segunda.

Podemos entender la mecánica del producto de matrices de una manera más fácil si consideramos la primera matriz como un grupo de vectores fila,

$$\begin{aligned} A_1 &= (a_{11} \quad a_{12} \quad \cdots a_{1n}) \\ A_2 &= (a_{21} \quad a_{22} \quad \cdots a_{2n}) \\ &\vdots \\ A_m &= (a_{m1} \quad a_{m2} \quad \cdots a_{mn}) \end{aligned} \rightarrow A = \begin{pmatrix} a_{11} & a_{12} & \cdots a_{1n} \\ a_{21} & a_{22} & \cdots a_{2n} \\ \vdots & \vdots & \cdots \vdots \\ a_{m1} & a_{m2} & \cdots a_{mn} \end{pmatrix}$$

y la segunda matriz como un grupo de vectores columna,

sion.

We may understand better the matrix product mechanism if we consider the first matrix as a group of row vectors,

An the second matrix as a group of column vectors.

$$B_1 = \begin{pmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{pmatrix} B_2 = \begin{pmatrix} b_{12} \\ b_{22} \\ \vdots \\ b_{n2} \end{pmatrix} \cdots B_3 = \begin{pmatrix} b_{1m} \\ b_{2m} \\ \vdots b_{nm} \end{pmatrix} \rightarrow B = \begin{pmatrix} b_{11} & b_{12} & \cdots b_{1n} \\ b_{21} & b_{22} & \cdots b_{2n} \\ \vdots & \vdots & \cdots \vdots \\ b_{m1} & b_{m2} & \cdots b_{mn} \end{pmatrix}$$

Podemos ahora considerar cada elemento  $p_{ij}$  de la matriz producto  $P = A \cdot B$  como el producto escalar del vector fila  $A_i$  for el vector columna  $B_j$ ,  $p_{ij} = A_i \cdot B_j$ . Es ahora relativamente fácil, deducir algunas de las propiedad del producto matricial,

1. Para que dos matrices puedan multiplicarse, es preciso que el número de columnas de la primera coincida con el numero de filas de la segunda. Además la matriz producto tiene tantas filas como la primera matriz y tantas columnas como la segunda.
2. El producto matricial no es conmutativo. En general  $A \cdot B \neq B \cdot A$
3.  $(A \cdot B)^T = B^T \cdot A^T$

En Numpy se emplea el símbolo @ para representar el producto escalar, el producto de un vector por una matriz y el producto matricial. En todos los casos, es preciso que las dimensiones internas de los objetos que se multiplican coincidan. A continuación se muestran algunos ejemplos,

we may consider the consider each entry  $p_{ij}$  of the product matrix  $P = A \cdot B$  as a escalar product of the row vector  $A_i$  and the column vector  $B_j$ ,  $p_{ij} = A_i \cdot B_j$ . Now we can easily find out some interesting properties of the matrix product.

1. Two matrices can be multiplied only if the number of columns of the first matrix meet the number of columns of the second one. Besides, the product matrix has so meny row as the first matrix and so many columns as the second one.
2. Matrix produc it is not commutative. In general,  $A \cdot B \neq B \cdot A$
3.  $(A \cdot B)^T = B^T \cdot A^T$

Numpy uses the symbol @ to represent the scalar product, the product of a matrix and a vector and the matrix product. In any case, it is necessary that the inner dimensions of factor objects meet. Next we show some matrix multiplication examples,

```
In [382]: a = np.array([1,2,3,4])
```

```
In [383]: b = np.array([-1,2,0,-3])
```

```
In [384]: a@b
```

```
Out[384]: -9
```

```
In [386]: A = np.array([[3,-5,-2,1],[2,3,4,5]])
```

```
In [387]: A
```

```
Out[387]:
```

```
array([[ 3, -5, -2,  1],
       [ 2,  3,  4,  5]])
```

```
In [388]: A@b
```

```
Out[388]: array([-16, -11])
```

```
In [389]: b@A.T
```

```
Out[389]: array([-16, -11])
```

```
In [390]: b@A
```

```
Traceback (most recent call last):
```

```
Cell In[390], line 1
```

```
b@A
```

```
ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0,
with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 2 is different from 4)
```

```
In [398]: B
```

```
Out[398]:
```

```
array([[ 0, -1],
       [-1,  0],
       [ 2,  3],
       [ 3,  4]])
```

```
In [399]: A@B
```

```
Out[399]:
```

```
array([[ 4, -5],
       [20, 30]])
```

```
In [400]: B@A
```

```
Out[400]:
```

```
array([[-2, -3, -4, -5],
       [-3,  5,  2, -1],
       [12, -1,  8, 17],
       [17, -3, 10, 23]])
```

En la Línea In [384] se ha calculado el producto escalar de los vectores **a** y **b**. En este caso, la única condición requerida es que tengan la misma dimensión. En la línea In [388] se calcula el producto de la matriz **A** por el vector **b**. El requisito ahora es que la dimensión de la matriz ( $2 \times 4$ ), que corresponde al número de columnas, coincida con la única dimensión del vector (4). En la línea In [389] calculamos el producto del vector **b** por la transpuesta de la matriz **A**. La operación es posible porque la única dimensión del vector y coincide con la dimensión de la matriz transpuesta (4), que corresponde con a número de filas. Sin embargo, no es posible calcular el producto del vector **b** por la matriz **A**, ya que la dimensión del vector no coincide con la primera dimensión de la matriz ( $2 \times 4$ ). En la línea In [399] hemos multiplicado las matrices **A** ( $2 \times 4$ ) por la matriz **B** ( $4 \times 2$ ). Como coinciden las dimensiones “internas”, –numero de columnas de la primera matriz con número de filas de la segunda– La operación puede llevarse a cabo. Si invertimos el orden de las matrices, (línea In [400]) el producto también es posible, ya que en también coincidirían las dimensiones “internas”. Sin embargo, es fácil ver que los resultados son complementamente distintos.

**Matriz identidad** La matriz identidad de orden  $n \times n$  se define como:

$$I_n = \begin{cases} i_{ll} = 1 \\ i_{kj} = 0, \quad k \neq j \end{cases}$$

Es decir, una matriz en la que todos los elementos que no pertenecen a la diagonal principal son 0 y los elementos de la diagonal principal son 1. Por ejemplo,

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La matriz identidad  $I_n$  es el elemento neutro del producto de matrices cuadradas de orden  $n \times n$ ,

In line In[384] we calculated the scalar product of vectors **a** and **b**. In this case, the only requirement is that both vectors have the same dimension. In line In [388] we calculated the product of matrix **A** and the vector **b**. Now the requirement is the dimension of the matrix ( $2 \times 4$ ), corresponding to the number of columns, meets the the single dimension of the vector (4). In line [389], we multiplied vector **b** with the transpose of matrix **A**. This operation can be carried out because the single dimension of vector **b** is the same as the dimension of the transpose of the matrix ( $4 \times 2$ ), which corresponds with the number of rows of the matrix. However, we cannot multiply vector **b** by matrix **A** because the dimension of the vector does not meet the first dimension of the matrix ( $2 \times 4$ ). In line In [399] we multiplied matrix **A** ( $2 \times 4$ ) by matrix **B** ( $4 \times 2$ ). We can carry out the operation because the “inner” dimensions of the matrices –number of columns of the first matrix and number of rows of the second one–, meet. If we invert the order of the matrices (line In [400]), we can still multiply the matrix because, in this particular case, the new “inner” dimensions also meet. However, the result is entirely different

**The identity matrix** The identity matrix of dimension  $n \times n$  is defined as:

So, an identity matrix has any entry off the main diagonal equal to 0 and all entries on the main diagonal equal to 1, for example,

The identity matrix  $I_n$  is the identity element of the square  $n \times n$  matrices product.

$$A_{n \times n} \cdot I_n = I_n \cdot A_{n \times n}$$

Además,

Besides,

$$A_{n \times m} \cdot I_m = A_{n \times m}$$

$$I_n \cdot A_{n \times m} = A_{n \times m}$$

En Numpy se emplea el comando `eye(n)` para construir la matriz identidad de dimensiones  $n \times n$ ,

In Numpy, we can use the command `eye(n)` to build the identity matrix of dimension  $n \times n$ .

```
In [381]: np.eye(4)
Out[381]:
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Una matriz cuadrada se dice que es ortogonal si cumple,

An orthogonal matrix is a square matrix that fulfils,

$$A^T \cdot A = I$$

**Norma de un vector.** La longitud euclídea, módulo, norma 2 o simplemente norma de un vector se define como,

**Vector norm** The Euclidean length, module, norm two or just norm of a vector is defined as follows,

$$\|x\|_2 = \|x\| = \sqrt{x \cdot x} = \sqrt{x^T x} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} = \left( \sum_{i=1}^n x_i^2 \right)^{\frac{1}{2}}$$

Constituye la manera usual de medir la longitud de un vector. Tiene una interpretación geométrica inmediata a través del teorema de Pitágoras: nos da la longitud del segmento que representa al vector. La figura 3.3 muestra dicha interpretación, para un vector bidimensional.

The vector norm represents the usual method to measure its length. It has a direct geometrical interpretation using Pithagoras' theorem: the norm is the length of the segment that represents the vector. Figure 3.3 shows such an interpretation for a bi-dimensional vector.

La norma de un vector en Numpy se obtiene empleando el comando `norm` que pertenece al submódulo `linalg`,

```
In [422]: a
Out[422]: array([1, 2, 3, 4])
```

```
In [425]: np.linalg.norm(a)
Out[425]: 5.477225575051661
```

A partir de la norma de un vector es posible obtener una expresión alternativa para el producto escalar de dos vectores,

We can derive an alternative expression of the scalar product of two vectors, using the vector norm

$$a \cdot b = \|a\| \|b\| \cos \theta$$

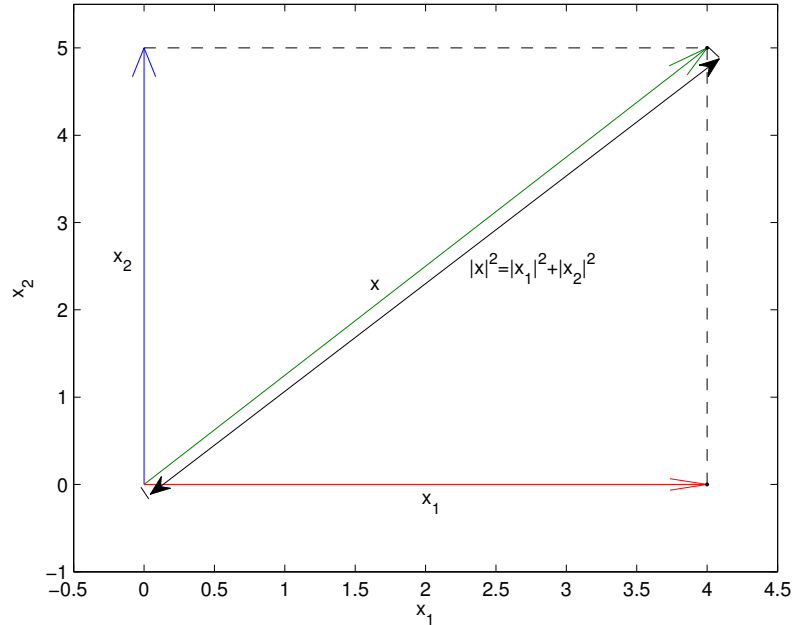
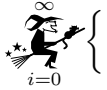


Figura 1.3: interpretación geométrica de la norma de un vector

Figure 1.3: Geometrical interpretation of the norm of a vector.

Donde  $\theta$  representa el ángulo formado por los dos vectores.

where  $\theta$  is the angle between the two vectors.



Aunque se trate de la manera más común de definir la norma de un vector, la norma 2 no es la única definición posible,

*Norma 1:* Se define como la suma de los valores absolutos de los elementos de un vector,

Being the norm 2 the most common way to define a vector norm is by no means the only possible definition,

*Norm 1:* defined as the sum of the absolute values of vector entries.

$$\|x\|_1 = |x_1| + |x_2| \cdots |x_n|$$

*Norma p:* Es una generalización de la norma 2,

*P-Norm:* It is a generalization of the norm 2

$$\|x\|_p = \sqrt[p]{|x_1|^p + |x_2|^p + \cdots |x_n|^p} = \left( \sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}}$$

*Norma  $\infty$ :* se define como el mayor ele-

$\infty$  *Norm:* The entry with the largest ab-

mento del vector valor absoluto, | solute value

$$\|x\|_{\infty} = \max \{|x_i|\}$$

*Norma*  $-\infty$ : el menor elemento del vector |  $-\infty$  *Norm*: The entry with the smallest  
en valor absoluto, | absolute value.

$$\|x\|_{-\infty} = \min \{|x_i|\}$$

En Numpy la norma de un vector puede obtenerse mediante el comando `norm(v,p)`. La primera variable de entrada debe ser un vector y la segunda el tipo de norma que se desea calcular. Si se omite la segunda variable de entrada, el comando devuelve la norma 2. Para las normas  $\infty$  y  $-\infty$  Se emplea el símbolo especial `inf` de Numpy. A continuación se incluyen varios ejemplo de utilización,

In Numpy the vector norm can be obtained using the command `norm(v,p)`. The first variable should be a vector and the second one represents the norm we want to calculate. If we leave out the second variable, the function returns norm 2 by default. For norms  $\infty$  y  $-\infty$  we use the symbol `inf` to represent the norm kind.

```
In [422]: a
Out[422]: array([1, 2, 3, 4])

In [423]: np.linalg.norm(a,1)
Out[423]: 10.0

In [424]: np.linalg.norm(a,2)
Out[424]: 5.477225575051661

In [425]: np.linalg.norm(a)
Out[425]: 5.477225575051661

In [426]: np.linalg.norm(a,4)
Out[426]: 4.337613136533361

In [427]: np.linalg.norm(a,np.inf)
Out[427]: 4.0

In [428]: np.linalg.norm(a,-np.inf)
Out[428]: 1.0
```

En general, una norma se define como una | In general, a norm can be define as fun-  
función  $\mathbb{R}^n \rightarrow \mathbb{R}$ , que cumple, | ctions  $\mathbb{R}^n \rightarrow \mathbb{R}$ , which satisfies,

$$\|x\| \geq 0, \|x\| = 0 \Rightarrow x = 0$$

$$\|x + y\| \leq \|x\| + \|y\|$$

$$\|\alpha x\| = |\alpha| \|x\|, \alpha \in \mathbb{R}$$



Llamaremos vectores unitarios  $u$ , a aquellos para los que se cumple que  $\|u\| = 1$ .

Dos vectores  $a$  y  $b$  son ortogonales si cumplen que su producto escalar es nulo,  $a^T b = 0 \Rightarrow a \perp b$ . Si además ambos vectores tienen módulo unidad, se dice entonces que los vectores son ortonormales. Desde el punto de vista de su representación geométrica, dos vectores ortogonales, forman entre sí un ángulo recto.

**Traza de una matriz.** La traza de una matriz cuadrada, se define como la suma de los elementos que ocupan su diagonal principal

Vectors  $u$  with  $\|u\| = 1$  are called unitary vectors.

Two vectors  $a$  and  $b$  are orthogonal if their scalar product is zero,  $a^T b = 0 \Rightarrow a \perp b$ . Besides, if the vectors have norm one them, they are called orthonormal vectors. From the point of view of the geometrical representation, the angle between two orthogonal vectors is a square angle.

**Matrix's trace.** The trace of a square matrix is the sum of the entries located in the matrix's main diagonal.

$$Tr(A) = \sum_{i=1}^n a_{ii}$$

$$Tr \left( \begin{pmatrix} 1 & 4 & 4 \\ 2 & -2 & 2 \\ 0 & 3 & 6 \end{pmatrix} \right) = 1 - 2 + 6 = 5$$

La traza de la suma de dos matrices cuadradas  $A$  y  $B$  del mismo orden, coincide con la suma de las trazas de  $A$  y  $B$ ,

The trace of the sum of two square, same dimension matrices  $A$  and  $B$  is equal to the sum of  $A$  trace plus  $B$  trace,

$$tr(A + B) = tr(A) + tr(B)$$

Dada una matriz  $A$  de dimensión  $m \times n$  y una matriz  $B$  de dimensión  $n \times m$ , se cumple que,

For two matrices  $A$  of dimensions  $m \times n$  and  $B$  of dimensions  $n \times m$  is always true that,

$$tr(AB) = tr(BA)$$

En Python, puede obtenerse directamente el valor de la traza de una matriz  $A$ , mediante la función `np.trace(A)`. En este caso, se trata de un método asociado a cualquier array por lo que también puede expresarse como `A.trace()`

In Numpy, the value of a matrix's trace can be calculated using the function `np.trace(A)`. In this case, the function is a method associated to any matrix and can be also expressed as `A.trace()`



```
In [197]: A = np.array([[1,2,3],[3,-2,3],[0,2,-1]])
```

```
In [198]: A
Out[198]:
array([[ 1,  2,  3],
       [ 3, -2,  3],
       [ 0,  2, -1]])
```

```
In [199]: np.trace(A)
Out[199]: -2
```

```
In [200]: A.trace()
Out[200]: -2
```

**Determinante de una matriz.** En Numpy, el determinante de una matriz se calcula empleando la función `det`, del submódulo `linalg`. Así, para calcular el determinante de la matriz  $A$  del ejemplo anterior,

**Matrix's determinant.** In Numpy, a matrix's determinant can be calculated using the function `det`, del submódulo `linalg`. We can calculate the determinant of matrix  $A$  of the previous example,

```
In [203]: np.linalg.det(A)
Out[203]: 20.000000000000007
```



El determinante de una matriz  $A$ , se representa habitualmente como  $|A|$  o, en ocasiones como  $\det(A)$ . Para poder definir el determinante de una matriz, necesitamos antes introducir una serie de conceptos previos. En primer lugar, si consideramos un escalar como una matriz de un solo elemento, el determinante sería precisamente el valor de ese único elemento,

The determinant of a matrix,  $A$  is usually represented as  $|A|$  and occasionally as  $\det(A)$ . To define the determinant of a matrix, we need first to establish some previous definitions. First, if we consider a scalar as a matrix of a single entry, the determinant of the matrix is this single entry.

$$A = (a_{11}) \rightarrow |A| = a_{11}$$

En álgebra lineal, se denomina menor primero,  $M_{ij}$  de una matriz  $A$ , al determinante de la matriz que resulta de eliminar de la matriz  $A$  la fila  $i$  y la columna  $j$ . Por ejemplo,

In linear algebra, a first minor  $M_{ij}$  of a square matrix  $A$  is the determinant of the matrix obtained by removing simultaneously row  $i$  and column  $j$  of  $A$ . For instance,

$$A = \begin{pmatrix} 1 & 0 & -2 \\ 3 & -2 & 3 \\ 0 & 6 & 5 \end{pmatrix}, M_{23} = \det \begin{pmatrix} 1 & 0 \\ 0 & 6 \end{pmatrix}$$

$$M_{32} = \det \begin{pmatrix} 1 & -2 \\ 3 & 3 \end{pmatrix}, M_{33} = \det \begin{pmatrix} 1 & 0 \\ 3 & -2 \end{pmatrix} \dots$$

El cofactor  $C_{ij}$  de un elemento  $a_{ij}$  de la matriz  $A$ , se define a partir del menor primero  $M_{ij}$ , que corresponde precisamente a la fila y columna que contiene a  $a_{ij}$ ,

We define the cofactor  $C_{ij}$  of a matrix  $A_{ij}$  entry,  $a_{ij}$ , using the first minor  $M_{ij}$  obtained removing the row and column  $a_{ij}$  belongs to.

$$C_{ij} = (-1)^{i+j} M_{ij}$$

Podemos ahora definir el determinante de una matriz  $A$  cuadrada de orden  $n$ , empleando la fórmula de Laplace,

Now, we can define the determinant of a  $n$  dimensional square matrix  $A$ , using the Laplace's formula,

$$|A| = \sum_{j=1}^n a_{ij} C_{ij}$$

o alternativamente,

or also as,

$$|A| = \sum_{i=1}^n a_{ij} C_{ij}$$

En el primer caso, se dice que se ha desarrollado el determinante a lo largo de la fila  $i$ . En el segundo caso, se dice que se ha desarrollado a lo largo de la columna  $j$ .

In the first case, we say that we develop the determinant along the row  $i$ . In the second one, we say that we have developed the determinant along the column  $j$ .

La fórmula de Laplace obtiene el determinante de una matriz de orden  $n \times n$  a partir del cálculo de los menores complementarios de los elementos de una fila o columna; los determinantes de  $n$  matrices de orden  $(n-1) \times (n-1)$ . A su vez, podríamos calcular cada menor complementario, aplicando la fórmula de Laplace y así sucesivamente hasta llegar a matrices de orden  $2 \times 2$ . Para una matriz  $2 \times 2$ , si desarrollamos por la primera fila obtenemos su determinante como,

Laplace's formula gets the determinant of a Matrix of dimensions  $n \times n$  using the first minors of a matrix row or column, i.e. the determinants of  $n$  matrix of order  $(n-1) \times (n-1)$ . In turn, we can calculate each minor using Laplace's formula and so on, till we arrive at matrices of dimension  $2 \times 2$ . We obtain the determinant of a  $2 \times 2$  matrix developing it along its first row,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$\begin{aligned} |A| &= \sum_{j=1}^2 a_{1j} C_{1j} = a_{11} C_{11} + a_{12} C_{12} \\ &= a_{11} (-1)^{1+1} |M_{11}| + a_{12} (-1)^{1+2} |M_{12}| \\ &= -a_{11} a_{22} + a_{12} a_{21} \end{aligned}$$

y si desarrollamos por la segunda columna,

And if we develop using the second column,

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$\begin{aligned} |A| &= \sum_{j=1}^2 a_{i2} C_{i2} = a_{12} C_{12} + a_{22} C_{22} \\ &= a_{12} (-1)^{1+2} |M_{12}| + a_{22} (-1)^{2+2} |M_{22}| \\ &= -a_{12} a_{21} + a_{22} a_{12} \end{aligned}$$

Para una matriz de dimensión arbitraria  $n \times n$ , el determinante se obtiene aplicando recursivamente la fórmula de Laplace,

For a matrix of arbitrary dimension we obtain the determinant using the Laplace's formula recursively,

$$\begin{aligned} |A| &= \sum_{j=1}^n a_{ij} C_{ij} = \sum_{j=1}^n a_{ij} (-1)^{i+j} |M_{ij}^{(n-1) \times (n-1)}| \\ |M_{ij}^{(n-1) \times (n-1)}| &= \sum_{k=1}^{n-1} m_{lk} C_{lk} = \sum_{k=1}^{n-1} m_{lk} (-1)^{l+k} |M_{lk}^{(n-2) \times (n-2)}| \\ &\vdots \\ |M_{st}^{1 \times 1}| &= (-1)^{s+t} m_{st} \end{aligned}$$

Así, por ejemplo, podemos calcular el determinante de la matriz,

For instance, we can calculate the determinant of the following matrix,

$$A = \begin{pmatrix} 1 & 0 & -2 \\ 3 & -2 & 3 \\ 0 & 6 & 5 \end{pmatrix}$$

desarrollándolo por los elementos de la primera columna, como,

Developing by the first column entries,

$$\begin{aligned} |A| &= 1 \cdot (-1)^2 \cdot \begin{vmatrix} -2 & 3 \\ 6 & 5 \end{vmatrix} + 3 \cdot (-1)^3 \cdot \begin{vmatrix} 0 & -2 \\ 0 & 5 \end{vmatrix} + 0 \cdot (-1)^4 \cdot \begin{vmatrix} 0 & -2 \\ -2 & -3 \end{vmatrix} \\ &= 1 \cdot (-1)^2 \cdot [(-2) \cdot 5 - 6 \cdot 3] + 3 \cdot (-1)^3 \cdot [0 \cdot 5 - 6 \cdot (-2)] + 0 \cdot (-1)^4 \cdot [0 \cdot 3 - (-2) \cdot (-2)] = -64 \end{aligned}$$

Podemos programar en Python una función recursiva<sup>3</sup> que calcule el determinante de una matriz de dimensiones  $n \times n$ .

We can now program a recursive function<sup>3</sup> to calculate the determinant of any  $n \times n$  dimensions.

<sup>3</sup>El método no es especialmente eficiente pero ilustra el uso de funciones recursivas.

<sup>3</sup>The method is not particularly efficient but it helps to show the use of recursive functions

---

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Thu May 23 14:50:48 2024
5  Este modulo implementa la funcion dumbdet que calcula el determinante de
6  una matriz empleando la formula de Laplace. La función es recursiva,
7  (se llama a si misma sucesivamente para calcular los cofactores necesarios).
8  Desarrolla siempre por los elementos de la primera columna.
9  (Es un prodigio de ineficiencia numerica, pero permite manejar bucles
10 y funciones recursivas, asi que supongo que puede ser útil para los que
11 empiezan a programar).
12 un posible ejercicio para ver lo malo que es el método, consiste ir
13  aumentando la dimension de la matriz y comparar lo que lo tarde en
14  calcular el determinante con lo que tarda la función de numpy.linalg.det...
15 this module implements the function poordet which calculates a matrix
16 determinant using the Laplace's formulae. It is a recursive function,

```

---

```

17  (it calls itself on and on to calculate the cofactor needed to get the
18  determinat). It always developed the formula using the elements of the matrix
19  first column.
20  (It is the most inefficient function ever written, but may be an example of loops
21  and recursive functions for beginners)
22  @author: juan
23  """
24  import numpy as np
25  def dumbdet(A):
26
27      #first we check the matrix is square
28      #primero comprobamos si la matriz es cuadrada
29      sz = A.shape[0]
30      d = 0 #variable to save the result/ Variable para guardar el resultado
31
32      if sz != A.shape[1]:
33          print('La matriz no es cuadrada')
34          print('The matrix is not square')
35          d = []
36      elif sz == 1:
37          return A[0,0]
38      else:
39
40          for i in range(0,sz):
41              N = np.delete(A,i,0)
42              d = (-1)**(i+2)*A[i,0]*dumbdet(N[:,1:sz])+d
43
44
45      return d

```

---



Entre las propiedades de los determinantes, destacaremos las siguientes,

1. El determinante del producto de un escalar  $a$  por una matriz  $A$  cumple,

$$|a \cdot A| = a^n \cdot |A|$$

2. El determinante de una matriz es igual al de su traspuesta,

$$|A| = |A^T|$$

3. El determinante del producto de dos matrices es igual al producto de los determinantes,

$$|A_{n \times n} \cdot B_{n \times n}| = |A_{n \times n}| \cdot |B_{n \times n}|$$

Among determinants' properties we will point out the following ones,

1. The determinant of a scalar  $a$  by a matrix  $A$  is,

$$|a \cdot A| = a^n \cdot |A|$$

2. The determinant of a matrix transpose equals the determinant of the matrix,

$$|A| = |A^T|$$

3. The determinant of a matrix product equals the product of their determinants,

$$|A_{n \times n} \cdot B_{n \times n}| = |A_{n \times n}| \cdot |B_{n \times n}|$$

Una matriz es singular si su determinante es cero.

El rango de una matriz  $A$  se define como el tamaño de la submatriz más grande dentro de  $A$ , cuyo determinante es distinto de cero. Así por ejemplo la matriz,

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \rightarrow |A| = 0$$

Es una matriz singular y su rango es dos,

$$\begin{vmatrix} 1 & 2 \\ 4 & 5 \end{vmatrix} = -3 \neq 0 \Rightarrow r(A) = 2$$

Para una matriz cuadrada no singular, su rango coincide con su dimensión.

En Numpy se puede obtener el rango de una matriz mediante el comando `matrix_rank`,

```
In [395]: A
Out[395]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
In [396]: np.linalg.matrix_rank(A)
Out[396]: 2
```

A singular matrix is a square matrix whose determinant is zero.

The rank of a matrix  $A$  is the size of the largest submatrix, inside  $A$ , whose determinant is non-zero. For instance,

is a singular matrix, and its rank is 2,

For a non-singular square matrix, its rank meets its dimension.

In Numpy, we can obtain the rank of a matrix using the comand `matrix_rank`,

**Inversión.** Dada una matriz cuadrada no singular  $A$  de dimension  $n$  existe una única matriz  $A^{-1}$  de dimension  $n$  que cumple,

$$A \cdot A^{-1} = I_{n \times n}$$

Donde  $I_{n \times n}$  es la matriz identidad de dimensión  $n$ . La matriz  $A^{-1}$  recibe el nombre de matriz inversa de  $A$ , y en numpy puede calcularse a partir de  $A$  como `np.linalg.inv(A)`, ó `np.linalg.matrix_power(A,-1)` En el segundo caso, estamos empleando la función, del submódulo linalg de numpy, `matrix.power(A,n)` que permite calcular el resultado de elevar una matriz a una potencia  $A^n$ .

**Inversion.** For a square non-singular matrix  $A$  of dimension  $n$  there is a unique matrix  $A^{-1}$  of dimension  $n$  that satisfies,

$$A \cdot A^{-1} = I_{n \times n}$$

Where  $I_{n \times n}$  is the identity matrix of dimension  $n$ . The matrix  $A^{-1}$  is called the inverse matrix of  $A$ , and we can calculate it in Numpy applying the comand `np.linalg.inv(A)`, ó `np.linalg.matrix_power(A,-1)` to the matrix  $A$ . In the second case, we are using the function `matrix.power(A,n)` which allows us to calculate the result to raise a matrix to a power  $A^n$ .

```

In [409]: B = np.linalg.inv(A)

In [410]: B
Out[410]:
array([[ 1.29166667, -0.58333333, -0.04166667],
       [ 0.58333333, -0.16666667, -0.08333333],
       [-0.48611111,  0.30555556,  0.06944444]])

In [411]: B @ A
Out[411]:
array([[ 1.00000000e+00, -5.55111512e-17, -1.11022302e-16],
       [ 0.00000000e+00,  1.00000000e+00, -1.11022302e-16],
       [ 0.00000000e+00,  0.00000000e+00,  1.00000000e+00]])

In [412]: np.linalg.matrix_power(A,-1)
Out[412]:
array([[ 1.29166667, -0.58333333, -0.04166667],
       [ 0.58333333, -0.16666667, -0.08333333],
       [-0.48611111,  0.30555556,  0.06944444]])

```



La inversa de una matriz puede obtenerse a partir de la expresión,

A matrix inverse can be calculate using the following expression,

$$A^{-1} = \frac{1}{|A|} [adj(A)]^T$$

Donde  $adj(A)$  es la matriz adjunta de  $A$ , que se obtiene sustituyendo cada elemento  $a_{ij}$  de  $A$ , por su cofactor  $C_{ij}$ . A continuación incluimos el código en Python de una función, **inver**, que calcula la inversa de una matriz. La función **inver** llama a su vez a la función **dumbdet** descrita más arriba, por lo que debemos importar el módulo **determinante** en el módulo en que vamos a crear la función **inver**.

Where  $adj(A)$  is the adjugate matrix of  $A$ , which can be obtained replacing each entry  $a_{ij}$  of  $A$  by its cofactor  $C_{ij}$ . Next, We include the Python code of a function, **inver**, to calculate the inverse of a matrix. The function **inver**, in turn, calls the function **dumbdet**. Therefore, we must need to import the module **determinante** in the module where we will create the funtion **inver**

---

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat May 25 17:47:28 2024
4  Este módulo implementa la funcion inver, que calcula la inversa
5  de una matriz cuadrada empleando la expresión clásica
6  A^-1 = [adj(A)]^T*det(A)
7  EL determinante de la matriz se obtiene empleando la función
8  dumbdet incluida en el módulo determinante
9  @author: abierto
10 """
11 import numpy as np

```

```

12 from determinante import dumbdet
13 def inver(A):
14     """
15     Esta función devuelve la inversa de una matriz cuadrada A
16     """
17     # primero comprobamos que la matriz suministrada
18     # es cuadrada:
19     f = A.shape[0]
20     # creamos una matriz de ceros del mismo tamaño
21     iA = np.zeros((f,f))
22     if f != A.shape[1]:
23         print('la matriz no es cuadrada, Campeón')
24         return()
25     else:
26         # calculamos el determinante de A, si es cero hemos terminado
27         dA=dumbdet(A)
28         if dA==0:
29             print('la matriz es singular, la pobre')
30             return()
31
32         if abs(dA) <= 10*np.finfo(float).eps:
33             print('cuidado determinante proximo a cero')
34
35
36         # Calculamos el cofactor de cada
37         # término de A mediante un doble bucle.
38         for i in range(f):
39             for j in range(f):
40                 # Construimos el menor correspondiente al elemento (i,j)
41                 m=np.delete(np.delete(A,i,0),j,1)
42
43                 # calculamos el cofactor llamando a la función determinante
44                 # lo ponemos ya en la posición que correspondería a la matriz
45                 # transpuesta de la adjunta.
46                 iA[j,i]=(-1)**(i+j)*dumbdet(m)
47
48
49         # Terminamos la operación dividiendo por el determinante de A
50         iA=iA/dA
51         return(iA)

```



Algunas propiedades relacionadas con la inversión de matrices son,

1. Inversa del producto de dos matrices,

$$(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$$

Some properties relate with matrix inversion are,

1. Inverse of the the producto of two ma-

2. Determinante de la inversa,	trices,
$ A^{-1}  =  A ^{-1}$	$(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$
3. Una matriz es ortogonal si su inversa coincide con su transpuesta,	2. Determinant of the inverse matrix,
$A^{-1} = A^T$	$ A^{-1}  =  A ^{-1}$
	3. A matrix is ortogonal if its transpose is equal to its inverse,
	$A^{-1} = A^T$

Tabla 1.1: Algunas funciones matemáticas en Numpy de uso frecuente  
Table 1.1: frequently used mathematical functions included in Numpy

Tipo Type	Nombre Name	variables variables	función matemática funcion matemática
Trigonómicas Trigonometric	cos	y=cos(x)	coseno de un ángulo en radianes Cosine of angle in radians
Trigonómicas Trigonometric	sin	y=sin(x)	seno de un ángulo en radianes Sine of an angle in radians
Trigonómicas Trigonometric	tan	y=tan(x)	tangente de un ángulo en radianes Tangent of an angle in radians
Trigonómicas Trigonometric	... arcsin	y=arc...(x) y=arcsin(x)	inversa de una función trigonométrica Inverse of a trigonometric function
Exponencial Exponential	exp	y=exp(x)	$e^x$
Exponencial Exponential	log	y=log(x)	logaritmo natural Natural logarithm
Exponencial Exponential	log10	log10(x)	logaritmo en base 10 Basis 10 logarithm
Exponencial Exponential	sqrt	y=sqrt(x)	$\sqrt{x}$
Redondeo Rounding	ceil	y=ceil(x)	redondeo hacia $+\infty$ rounding towards $+\infty$
Redondeo Rounding	floor	y=floor(x)	redondeo hacia $-\infty$ rounding towards $-\infty$
Redondeo Rounding	rint	y=rint(x)	redondeo al entero más próximo rounding towards the nearest integer
Redondeo Rounding	fix	y=fix(x)	redondeo hacia 0 rounding towards 0
Redondeo Rounding	mod	r=mod(x,y)	resto de la división entera de y entre x remainder after integer division
Módulos Norms	norm	y=norm(x)	módulo de un vector x Norm of a vector
Módulos Norms	abs	y=abs(x)	valor absoluto de x Absolute value of x
Módulos Norms	sign	y=sign(x)	función signo; 1 si $x > 0$ , -1 si $x < 0$ , 0 si $x=0$ sign function; 1 if $x > 0$ , -1 if $x < 0$ , 0 if $x=0$



### 1.2.1. Funciones incluidas en Numpy.

Numpy incluye un gran número de funciones. Muchas de ellas están pensadas para ser aplicadas a arrays o a variables numéricas indistintamente. En el primer caso, la función se aplica elemento a elemento y el resultado es un array con las mismas dimensiones que el original.

En la tabla 3.1, se incluyen algunos ejemplos de las funciones matemáticas más corrientes. No están todas. Para obtener una visión más completa de las funciones disponibles se aconseja acudir a la documentación de Numpy. El uso de estas funciones es directo, por ejemplo, el siguiente código calcula la exponencial de los elementos de una matriz,

```
In [1]: import numpy as np

In [2]: A = np.array([[1,2],[2,0],[3,5]])

In [3]: A
Out[3]:
array([[1, 2],
       [2, 0],
       [3, 5]])

In [4]: np.exp(A)
Out[4]:
array([[ 2.71828183,  7.3890561 ],
       [ 7.3890561 ,  1.          ],
       [20.08553692, 148.4131591 ]])
```



## 1.3. Operadores vectoriales

En esta sección vamos a estudiar el efecto de las operaciones matriciales, descritas en la sección anterior, sobre los vectores. Empecemos por considerar el producto por un escalar  $\alpha \cdot a$ . El efecto fundamental es modificar el módulo del vector,

$$\alpha \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \alpha a_1 \\ \alpha a_2 \\ \alpha a_3 \end{pmatrix} \rightarrow \|\alpha \cdot a\| = \sqrt{\alpha^2 a_1^2 + \alpha^2 a_2^2 + \alpha^2 a_3^2} = |\alpha| \sqrt{a_1^2 + a_2^2 + a_3^2} = |\alpha| \cdot \|a\|$$

### 1.2.1. Functions included in Numpy

Numpy includes a large number of mathematical functions. Many are intended to be applied to arrays and simple variables. In the first case, the function is applied entry-wise, and the result is an array with the same dimension as the original one.

Table 3.1 shows some examples of common mathematical functions. Not every function available in Numpy has been included. To achieve a complete view of available functions, it is advisable to read Numpy documentation. These functions can be used straightforwardly. For instance, the following code calculates the exponential values of a matrix entry,

## 1.3. Vectorial operators

In this section we will study the effect of matrix operation described in the previous section when they are applied to vectors. We start considering the effect if the product of a vector by a scalar  $\alpha \cdot a$ . The main effect is the change of the vector module,

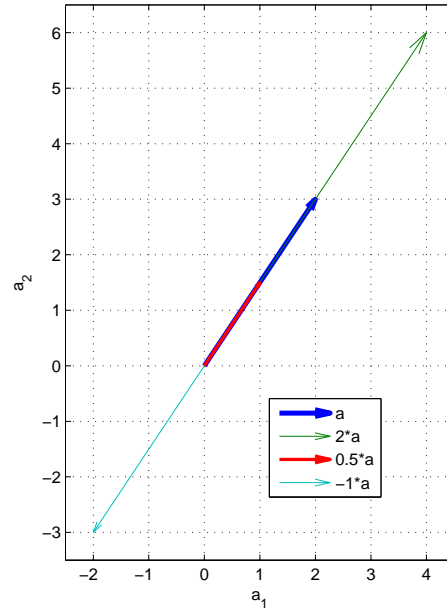


Figura 1.4: efecto del producto de un escalar por un vector  
 Figure 1.4: effect of the product of a vector by a scalar

Gráficamente, si  $\alpha$  es un número positivo y mayor que la unidad, el resultado del producto será un vector más largo que  $a$  con la misma dirección y sentido. Si por el contrario,  $\alpha$  es menor que la unidad, el vector resultante será más corto que  $a$ . Por último si se trata de un número negativo, a los resultados anteriores se añade el cambio de sentido con respecto a  $a$ . La figura 3.4 muestra gráficamente un ejemplo del producto de un vector por un escalar.

**Combinación lineal.** Combinando la suma de vectores, con el producto por un escalar, podemos generar nuevos vectores, a partir de otros, el proceso se conoce como combinación lineal,

If  $\alpha$  is a positive number and greater than one, we observe graphically that the result of the product will be a vector larger than  $a$  and with the same direction and sense. Conversely, if  $\alpha$  is positive but less than one, the resulting vector will be shorter than  $a$ . Lastly if  $\alpha$  is negative, a change of sense with respect to  $a$  is added to the previously obtained results. Figure 3.4 shows graphically and example of the product of a vector by a scalar.

**Linear combination** Combining vector addition with product by a scalar, we can use a set of vectors to generate new vectors. This process is called linear combination.

$$c = \alpha \cdot a + \beta \cdot b + \dots + \theta z$$

Así el vector  $c$  sería el resultado de una combinación lineal de los vectores  $a, b, \dots, z$ . Dado un conjunto de vectores, se dice que son linealmente independientes entre sí, si no es posible poner a unos como combinación lineal

So, the vector  $c$  is the result of linear combination of vectors  $a, b, \dots, z$ . A set of vectors are linearly independent if it is not possible to represent any of them as a combination of the others.

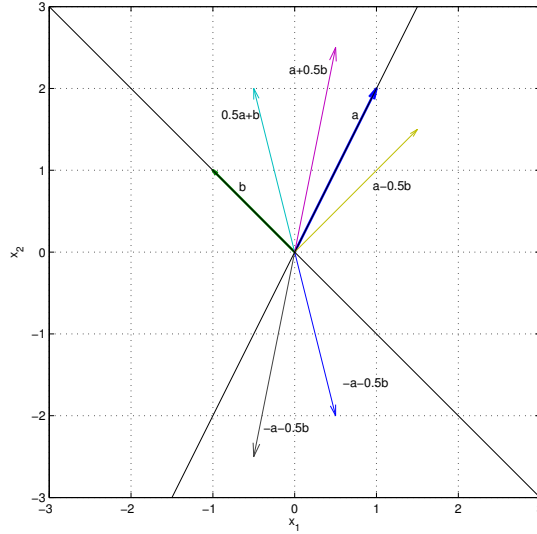


Figura 1.5: Representación gráfica de los vectores  $a = (1, 2)$ ,  $b = (-1, 1)$  y algunos vectores, combinación lineal de  $a$  y  $b$ .

Figure 1.5: A graphic representations of Vectors  $a = (1, 2)$ ,  $b = (-1, 1)$  and some vectors obtained from a lineal combination of  $a$  and  $b$ .

de otros,

$$\alpha \cdot a + \beta \cdot b + \cdots + \theta z = 0 \Rightarrow \alpha = \beta = \cdots = \theta = 0$$

Es posible expresar cualquier vector de dimensión  $n$  como una combinación lineal de  $n$  vectores linealmente independientes.

Supongamos  $n = 2$ , cualquier par de vectores que no estén alineados, pueden generar todos los vectores de dimensión 2 por ejemplo,

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \beta \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

La figura 3.5 muestra gráficamente estos dos vectores y algunos de los vectores resultantes de combinarlos linealmente.

Si tomamos como ejemplo  $n = 3$ , cualquier conjunto de vectores que no estén contenidos en el mismo plano, pueden generar cualquier otro vector de dimensión 3. Por ejemplo,

It is possible to represent any vector of dimensions  $n$  as a lineal combination of  $n$  vectors linearly independent.

Let's take  $n = 2$ , any pair of no collinear vectors can generate all vectors of dimension 2, for example,

Figure 3.5 shows these two vector and some other vectors obtained by combining them linearly.

If we take now  $n = 3$ , whatever three vectors not contained in the same plane can generate any other vector of dimension three. For example,

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} + \beta \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix} + \gamma \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$$

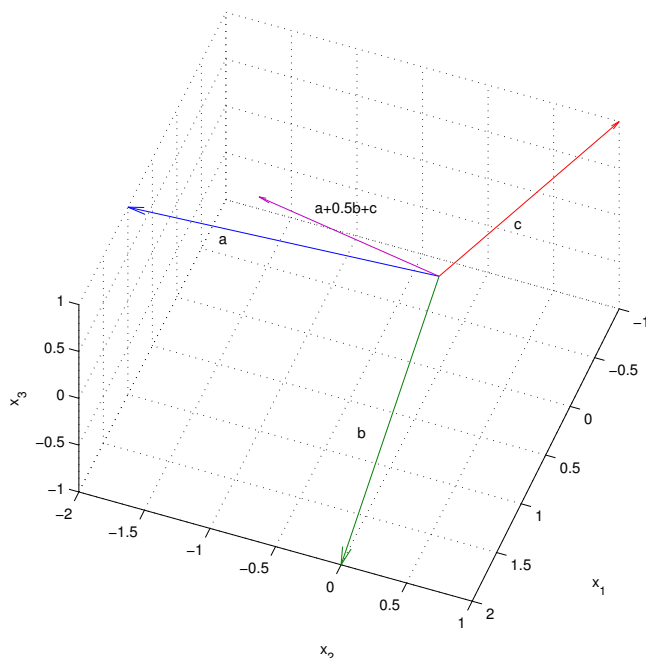


Figura 1.6: Representación gráfica de los vectores  $a = (1, -2, 1)$ ,  $b = (2, 0, -1)$ ,  $c = (-1, 1, 1)$  y del vector  $a - b + c$ .

Figure 1.6: Vectors  $a = (1, -2, 1)$ ,  $b = (2, 0, -1)$ ,  $c = (-1, 1, 1)$  and vector  $a - b + c$ ; a graphic representation

La figura 3.6 muestra gráficamente estos tres vectores y el vector resultante de su combinación lineal, con  $\alpha = 1$ ,  $\beta = -0.5$  y  $\gamma = 1$ . Es fácil ver a partir de la figura que cualquier otro vector de dimensión 3 que queramos construir puede obtenerse a partir de los vectores  $a$ ,  $b$  y  $c$ .

**Espacio vectorial y bases del espacio vectorial.** El conjunto de los vectores de dimensión  $n$ , junto con la suma vectorial y el producto por un escalar, constituye un *espacio vectorial* de dimensión  $n$ .

Como acabamos de ver, es posible obtener cualquier vector de dicho espacio vectorial a partir de  $n$  vectores linealmente independientes del mismo. Un conjunto de  $n$  vectores linealmente independientes de un espacio vectorial de dimensión  $n$  recibe el nombre de base del espacio vectorial. En principio es posible

Figure 3.6 shows these three vector and another one obtained by combining them linearly, taking  $\alpha = 1$ ,  $\beta = -0.5$  y  $\gamma = 1$ . It is easy to deduce using the figure that any other three-dimensional vector can be built using vectors  $a$ ,  $b$  y  $c$ .

**Vector spaces and vector space bases.** The set of all  $n$ -dimensional vectors with the vector addition and the product by scalars define a *vector space* of dimension  $n$ .

We have already seen that it is possible to obtain any vector of this  $n$ -dimensional space using  $n$  linearly independent vectors belonging to the space. A set of  $n$  linearly independent vector in a vector space of dimension  $n$ , is called a basis of the vector space. We can, in principle, find infinity different basis for an  $n$ -dimensional vector space. Some of these bases

encontrar infinitas bases distintas para un espacio vectorial de dimensión  $n$ . Hay algunas particularmente interesantes,

**Bases ortogonales.** Una base ortogonal es aquella en que todos sus vectores son ortogonales entre sí, es decir cumple que su producto escalar es  $b^i \cdot b^j = 0, i \neq j$ . Donde  $b^i$  representa el  $i$ -ésimo vector de la base,  $\mathcal{B} = \{b^1, b^2, \dots, b^n\}$ .

**Bases ortonormales.** Una base ortonormal, es una base ortogonal en la que, además, los vectores de la base tienen módulo 1. Es decir,  $b^i \cdot b^j = 0, i \neq j$  y  $b^i \cdot b^j = 1, i = j$ . Un caso particularmente útil de base ortonormal es la base canónica, formada por los vectores,

$$\mathcal{C} = \left\{ c^1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, c^2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \dots, c^{n-1} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix}, c^n = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \right\}$$

Podemos considerar las componentes de cualquier vector como los coeficientes de la combinación lineal de la base canónica que lo representa,

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix} = a_1 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + a_2 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \dots + a_{n-1} \cdot \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix} + a_n \cdot \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

Por extensión, podemos generalizar este resultado a cualquier otra base, es decir podemos agrupar en un vector los coeficientes de la combinación lineal de los vectores de la base que lo generan. Por ejemplo, si construimos, para los vectores de dimensión 3 la base,

$$\mathcal{B} = \left\{ \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \right\}$$

Podemos entonces representar un vector en la base  $\mathcal{B}$  como,

$$\alpha \cdot \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} + \beta \cdot \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + \gamma \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \rightarrow a^{\mathcal{B}} = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}$$

have properties that make them specially interesting,

**Ortogonal bases.** An orthogonal basis is built by vectors which are all orthogonal among them. That is, the scalar product is  $b^i \cdot b^j = 0, i \neq j$ . Where  $b^i$  is the  $i$  vector of the basis,  $\mathcal{B} = \{b^1, b^2, \dots, b^n\}$ .

**Ortonormal bases.** An orthogonal basis whose vectors, in addition, have norm 1 is called an orthonormal basis. That is,  $b^i \cdot b^j = 0, i \neq j$  and  $b^i \cdot b^j = 1, i = j$ . A particularly useful orthonormal basis is the canonical basis, composed by the following vectors,

We can consider the components of any vector as the coefficients of the linear combination of the canonical base vectors which represents the vector

We can also generalise and extend this result to any other basis, just grouping in a vector the coefficients of linear combination of basis vectors which generates it. For example, we can build for vector of dimension 3 the following basis,

We can then represent a vector in the basis  $\mathcal{B}$  as,

Donde estamos empleando el superíndice  $\mathcal{B}$ , para indicar que las componentes del vector  $a$  están definidas con respecto a la base  $\mathcal{B}$ .

Así por ejemplo el vector,

$$a^{\mathcal{B}} = \begin{pmatrix} 1.125 \\ 0.375 \\ 0.75 \end{pmatrix}$$

Tendría en la base canónica las componentes,

$$a^{\mathcal{B}} = \begin{pmatrix} 1.125 \\ 0.375 \\ 0.75 \end{pmatrix} \rightarrow a = 1.125 \cdot \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} + 0.375 \cdot \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix} + 0.75 \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \\ 1.5 \end{pmatrix}$$

La figura 3.7, muestra gráficamente la relación entre los vectores de la base canónica  $\mathcal{C}$ , los vectores de la base  $\mathcal{B}$ , y el vector  $a$ , cuyas componentes se han representado en ambas bases.

Podemos aprovechar el producto de matrices para obtener las componentes en la base canónica  $\mathcal{C}$  de un vector representado en una base cualquiera  $\mathcal{B}$ . Si agrupamos los vectores de la base  $\mathcal{B}$ , en una matriz  $B$ ,

Where the superscript  $\mathcal{B}$ , means that vector  $a$  components are defined in basis  $\mathcal{B}$ .  
So, for instance, vector,

Would have the following component in the canonical basis,

Figure 3.7, shows the relationship among the canonical basis vectors  $\mathcal{C}$ , the basis  $\mathcal{B}$  vectors, and vector  $a$ , whose component have been represented in both bases.

We can use the matrix product to obtain any vector components in the canonical basis  $\mathcal{C}$  from the vector components in any other basis  $\mathcal{B}$ . If we put together the basis  $\mathcal{B}$  vector in a single matrix  $B$ ,

$$\mathcal{B} = \left\{ b^1 = \begin{pmatrix} b_{11} \\ b_{21} \\ b_{31} \\ \vdots \\ b_{n1} \end{pmatrix}, b^2 = \begin{pmatrix} b_{12} \\ b_{22} \\ b_{32} \\ \vdots \\ b_{n2} \end{pmatrix}, \dots, b^n = \begin{pmatrix} b_{1n} \\ b_{2n} \\ \vdots \\ b_{(n-1)n} \\ b_{nn} \end{pmatrix} \right\} \rightarrow B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ b_{31} & b_{32} & \dots & \vdots \\ \vdots & \vdots & \dots & b_{(n-1)n} \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}$$

Supongamos que tenemos un vector  $a$  cuyas componentes en la base  $\mathcal{B}$  son,

$$a^{\mathcal{B}} = \begin{pmatrix} a_1^{\mathcal{B}} \\ a_2^{\mathcal{B}} \\ \vdots \\ a_n^{\mathcal{B}} \end{pmatrix}$$

Para obtener las componentes en la base canónica, basta entonces multiplicar la matriz  $B$ , por el vector  $a^{\mathcal{B}}$ . Así en el ejemplo que acabamos de ver,

Suppose now that we vector  $a$  has the following components in basis  $\mathcal{B}$ ,

we can obtain the components on the canonical base just multiplying matrix  $B$  for vector  $a^{\mathcal{B}}$ . So in the example we just have seen,

$$a = B \cdot a^{\mathcal{B}} \rightarrow a = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 0 & -1 \\ 0 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1.125 \\ 0.375 \\ 0.75 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \\ 1.5 \end{pmatrix}$$

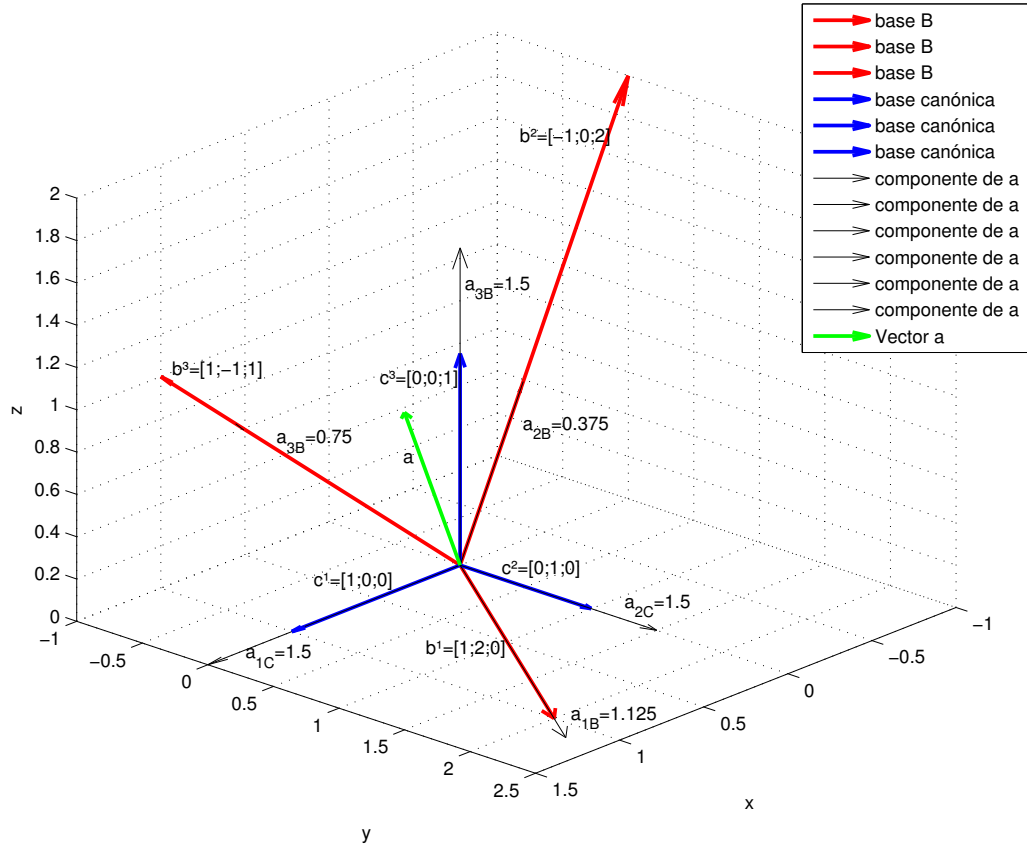


Figura 1.7: Representación gráfica del vector  $a$ , en la base canónica  $\mathcal{C}$  y en la base  $\mathcal{B}$ .  
 Figure 1.7: Graphic representation of vector  $a$ , in the canonical basis  $\mathcal{C}$  and in  $\mathcal{B}$  basis.

Por último, podemos combinar el producto de matrices y la matriz inversa, para obtener las componentes de un vector en una base cualquiera a partir de sus componentes en otra base. Supongamos que tenemos dos bases  $\mathcal{B}_1$  y  $\mathcal{B}_2$  y un vector  $a$ . Podemos obtener las componentes de  $a$  en la base canónica, a partir de las componentes en la base  $\mathcal{B}_1$  como,  $a = B_1 \cdot a^{\mathcal{B}_1}$  y a partir de sus componentes en la base  $\mathcal{B}_2$  como  $a = B_2 \cdot a^{\mathcal{B}_2}$ . Haciendo uso de la matriz inversa,

Eventually, we can combine the matrix product and the inverse matrix to obtain the vector components in any basis whatsoever, knowing the vector components in other arbitrary basis. Suppose we have two bases  $\mathcal{B}_1$  and  $\mathcal{B}_2$  and a vector  $a$ . We can get the components of  $a$  in the canonical basis from its components on basis  $\mathcal{B}_1$  as,  $a = B_1 \cdot a^{\mathcal{B}_1}$  and from basis  $\mathcal{B}_2$  as  $a = B_2 \cdot a^{\mathcal{B}_2}$ . Using the inverse matrix we obtain,

$$\begin{aligned} a &= B_1 \cdot a^{\mathcal{B}_1} \Rightarrow a^{\mathcal{B}_1} = B_1^{-1} \cdot a \\ a &= B_2 \cdot a^{\mathcal{B}_2} \Rightarrow a^{\mathcal{B}_2} = B_2^{-1} \cdot a \end{aligned}$$

y substituyendo obtenemos,

and, after replace, we obtain,

$$a^{B_1} = B_1^{-1} \cdot B_2 \cdot a^{B_2}$$

$$a^{B_2} = B_2^{-1} \cdot B_1 \cdot a^{B_1}$$

El siguiente código permite cambiar de base un vector y, si el vector pertenece a  $\mathbb{R}^3$ , representa gráficamente tanto el vector como las bases antigua y nueva.

The following code allows us to change a vector from one basis to another. Besides, if the vector is in  $\mathbb{R}^3$ , the program represents the vector and the old and the new bases.

cambio\_vb.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Fri Jun  7 15:40:36 2024
5  Vector basis change
6  aB1 : vector a coordinates represented in base b1
7  B1 Basis in which aB1 is represented. Matrix each column represents a vector of
8  the basis
9  B2 Basis in which we want to represent vector aB1
10 aB2: vector a coordinates in base B2
11 If base B2 is omitted the program takes B2 as canonical
12 @author: juan
13 """
14 import numpy as np
15 import matplotlib.pyplot as plt
16
17 eps = np.finfo(np.float64).eps
18
19 def pintavec(v,ax,col='b'):
20     '''just to draw a 3D vector'''
21
22
23     ax.quiver(0, 0, 0, v[0], v[1], v[2],color=col,length=0.1, normalize=True)
24     plt.axis('equal')
25 def basisch(aB1,B1,B2=np.array(0)):
26     '''Cambio de base de B1 a B2
27     Basis change form B1 to B2
28     '''
29     if B2.shape == ():
30         B2 = np.eye(B1.shape[0])
31     if np.linalg.det(B1) < eps or np.linalg.det(B2) < eps:
32         print('los vectores de al menos una de las bases no son linealmente independientes')
33         return([])
34     aB2 = np.linalg.inv(B2)@B1@aB1
35     if B1.shape[0] == 3:
36         #draw the vector
37         ax = plt.figure().add_subplot(projection='3d')
38         for i in B1.T:
39             pintavec(i,ax)
40         for i in B2.T:

```



```

41         pintavec(i,ax,'r')
42         pintavec(B1@aB1,ax,'k')
43         pintavec(B2@aB2,ax,'g')
44         ax.set_xlabel('x')
45         ax.set_ylabel('y')
46         ax.set_zlabel('z')
47     return(aB2)
48
49

```

**Operadores lineales.** A partir de los visto en las secciones anteriores, sabemos que el producto de una matriz de  $A$  de orden  $n \times n$  multiplicada por un vector  $b$  de dimension  $n$  da como resultado un nuevo vector  $c = A \cdot b$  de dimensión  $n$ . Podemos considerar cada matriz  $n \times n$  como un *operador lineal*, que transforma unos vectores en otros. Decimos que se trata de un operador lineal porque las componentes del vector resultante, están relacionadas linealmente con las del vector original, por ejemplo para  $n = 3$ ,

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \rightarrow \begin{matrix} y_1 = a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ y_2 = a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \\ y_3 = a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \end{matrix}$$

Entre los operadores lineales, es posible destacar aquellos que producen transformaciones geométricas sencillas. Veamos algunos ejemplos para vectores bidimensionales,

1. Dilatación: aumenta el módulo de un vector en un factor  $\alpha > 1$ . Contracción: disminuye el módulo de un vector en un factor  $0 < \alpha < 1$ . En ambos casos, se conserva la dirección y el sentido del vector original.

$$R = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \rightarrow R \cdot a = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \alpha \cdot a_1 \\ \alpha \cdot a_2 \end{pmatrix}$$

2. Reflexión de un vector respecto al eje x, conservando su módulo,

$$R_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \rightarrow R_x \cdot a = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} a_1 \\ -a_2 \end{pmatrix}$$

3. Reflexión de un vector respecto al eje y, conservando su módulo,

$$R_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \rightarrow R_y \cdot a = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -a_1 \\ a_2 \end{pmatrix}$$

**Linear operators.** In previous sections we saw that the product of a matrix  $A$  of dimensions  $n \times n$  by a vector  $b$  of dimension  $n$  cast a new vector  $c$  of dimension  $n$  as a result. We could consider every matrix  $n \times n$  as a *linear operator*, which transforms one vector into another. We define it as a linear operator because the components of the resulting vector are linearly related with the components of the operator (the original vector before the transformation). For example, in  $n = 3$ ,

Some linear operator are particular interesting because they generate simple geometrical transformations. Let's see some examples for bi-dimensional vectors:

1. Dilation: Spans the module of a vector in a factor  $\alpha > 1$ . Contraction: diminishes the module of a vector in a factor  $0 < \alpha < 1$ . In both cases, the direction and sense of the original vector is preserved.

2. Reflection of a vector with respect to the x-axis, preserving its norm,

3. reflection of a vector with respect to the y-axis, preserving its norm,

4. Reflexión respecto al origen: Invierte el sentido de un vector, conservando su módulo y dirección,

4. reflection with respect to the origin: Invert the sense of the vector, preserving its norm and direction.

$$R = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \rightarrow R \cdot a = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -a_1 \\ -a_2 \end{pmatrix}$$

Sería equivalente a aplicar una reflexión respecto al eje x y luego respecto al eje y o viceversa,

It is equivalent to apply a reflection on x-axis and then a reflection on y-axis or vice versa,

$$R = R_x \cdot R_y = R_y \cdot R_x.$$

5. Rotación en torno al origen un ángulo  $\theta$ ,

5. Rotation an angle  $\theta$  around the origin,

$$R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \rightarrow R_\theta \cdot a = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} a_1 \cos(\theta) - a_2 \sin(\theta) \\ a_1 \sin(\theta) + a_2 \cos(\theta) \end{pmatrix}$$

La figura 3.8 muestra los vectores resultantes de aplicar las transformaciones lineales que acabamos de describir al vector,  $a = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ ,

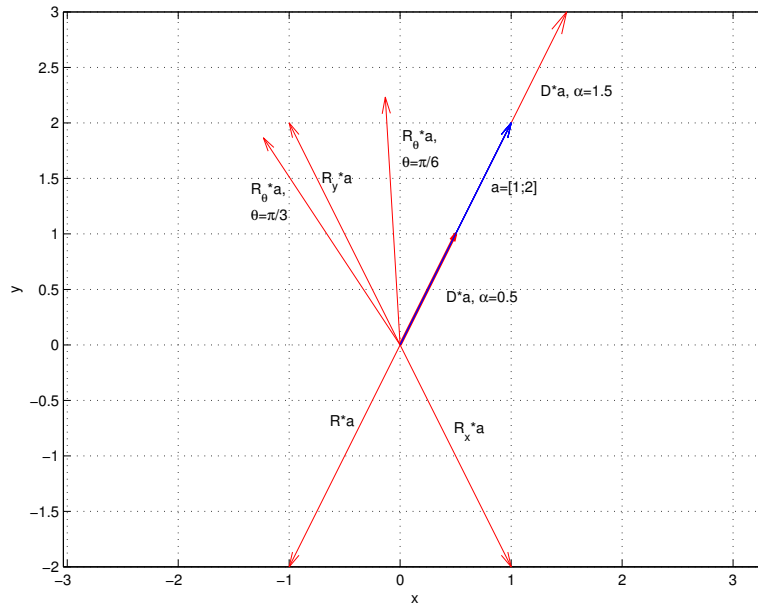


Figura 1.8: Transformaciones lineales del vector  $a = [1, 2]$ .  $D$ , dilatación/contracción en un factor 1.5/0.5.  $R_x$ , reflexión respecto al eje x.  $R_y$ , reflexión respecto al eje y.  $R_\theta$  rotaciones respecto al origen para ángulos  $\theta = \pi/6$  y  $\theta = \pi/3$

Figure 1.8: Linear transformation of vector  $a = [1, 2]$ .  $D$ , factor 1.5/0.5 dilation/contraction.  $R_y$  reflection on y-axis.  $R_\theta$   $\theta = \pi/6$  and  $\theta = \pi/3$  angles rotations around the origin.

**Norma de una matriz.** La norma de una matriz se puede definir a partir del efecto que produce al actuar, como un operador lineal, sobre un vector. En este caso, se les llama normas *inducidas*. Para una matriz  $A$  de orden  $m \times n$ ,  $y_{(m)} = A_{(m \times n)} x_{(n)}$ , La norma inducida de  $A$  se define en función de las normas de los vectores  $x$  de su dominio y de las normas de los vectores  $y$  de su rango como,

$$\|A\| = \max_{x \neq 0} \frac{\|y\|}{\|x\|} = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

Se puede interpretar como el factor máximo con que el que la matriz  $A$  puede *alargar* un vector cualquiera. Es posible definir la norma inducida en función de los vectores unitarios del dominio,

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{x \neq 0} \left\| A \frac{x}{\|x\|} \right\| = \max_{\|x\|=1} \|Ax\|$$

Junto a la norma inducida que acabamos de ver, se definen las siguientes normas,

1. Norma 1: Se suman los elementos de cada columna de la matriz, y se toma como norma el valor máximo de dichas sumas,

$$\|A_{m,n}\|_1 = \max_j \sum_{i=1}^m a_{ij}$$

2. Norma  $\infty$ : Se suman los elementos de cada fila y se toma como norma  $\infty$  el valor máximo de dichas sumas.

$$\|A_{m,n}\|_\infty = \max_i \sum_{j=1}^m a_{ij}$$

3. Norma 2: Se define como el mayor de los valores singulares de una matriz. (Ver sección 3.5.5).

$$\|A_{m,n}\|_2 = \sigma_1$$

4. Norma de Frobenius. Se define como la raíz cuadrada de la suma de los cuadrados de todos los elementos de la matriz,

$$\|A_{m,n}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^m a_{ij}^2}$$

**Matrix norm.** A matrix norm can be defined from the effect of the matrix when operates on a vector as a linear operator. In this case, we call these norms *induced* norms. For a matrix  $A$  of dimensions  $m \times n$ ,  $y_{(m)} = A_{m \times n} x_{(n)}$ . The induced norm of  $A$  is defined using the norms of the vectors  $x$  of its domain and the norms of the vectors  $y$  of its range, according to the following expression,

It can be interpreted as the maximum factor a matrix  $A$  can *enlarge* any vector. Some times the induced norm is defined using the unitary vector of its domain,

Besides the induced norm already described, it is possible to define the following norms,

1. Norm 1: we add the entries of each column up and then take the maximum value of these sums as a norm

2. Norm  $\infty$ : We add the entries of each row up and then take the maximum value of these sums as the  $\infty$  norm.

3. Norm 2: it is defined as the largest of the matrix singular values. (See section 3.5.5).

4. Frobenius' norm. It is defined as the square root of the sum of the squared values of the matrix entries,

Que también puede expresarse de forma mas directa como,

That can be also expressed in a more straightforward way as,

$$\|A_{m,n}\|_F = \sqrt{\text{tr}(A^T \cdot A)}$$

En Numpy, es posible calcular las distintas normas de una matriz, de modo análogo a como se calculan para el caso de vectores, mediante el comando `norm(A,p)`. Donde `A`, es ahora una matriz y `p` especifica el tipo de norma que se quiere calcular. En el caso de una matriz, el parámetro `p` solo puede tomar los valores, 1 (norma 1), 2 (norma 2), `inf` (norma  $\infty$ ), y `'fro'` (norma de Frobenius). Esta última es la norma por defecto, y es la que se obtiene si se no se define el tipo de norma. El siguiente ejemplo muestra el cálculo de las normas 1, 2,  $\infty$  y de Frobenius de la misma matriz.

In Numpy is possible to calculate the different matrix norms, similarly to how we calculate the norm of a vector, using the command `norm(A,p)`. Where `A` is now a matrix and `p` specifies the kind of norm we can calculate. In the case of a matrix, parameter `p` can take only the values, 1 (norm 1), 2 (norm 2), `inf`, (norm  $\infty$ ), and `'fro'` (Frobenius' norm). This last is the norm by default and, thus, if the norm Numpy calculates if parameter `p` is omitted. The following example shows the calculations of norms 1, 2,  $\infty$ , and Frobenius for the same matrix.

```
In [11]: A = np.array([[1,-1,3],[2,0,-2],[3,1,2]])
```

```
In [12]: A
Out[12]:
array([[ 1, -1,  3],
       [ 2,  0, -2],
       [ 3,  1,  2]])
```

```
In [13]: np.linalg.norm(A)
Out[13]: 5.744562646538029
```

```
In [14]: np.linalg.norm(A,'fro')
Out[14]: 5.744562646538029
```

```
In [15]: np.linalg.norm(A,1)
Out[15]: 7.0
```

```
In [17]: np.linalg.norm(A,inf)
Out[17]: 6.0
```

```
In [18]: np.linalg.norm(A,2)
Out[18]: 4.552861506620628
```

**Formas cuadráticas.** Se define como forma cuadrática a la siguiente operación entre una matriz cuadrada  $A$  de orden  $n \times n$  y un vector  $x$  de dimensión  $n$ ,

**Quadratic forms.** We define the following operation between a square matrix  $A$  of dimensions  $n \times n$  and a vector  $x$  as a quadratic form,

$$\alpha = x^T \cdot A \cdot x, \alpha \in \mathbb{R}$$

El resultado es un escalar. Así por ejemplo,

The result is a scalar. For instance,

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 0 & 2 \\ 3 & 2 & -2 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \rightarrow (1 \ 2 \ 3) \cdot \begin{pmatrix} 1 & 2 & -1 \\ 2 & 0 & 2 \\ 3 & 2 & -2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = 21$$

Para dimensión  $n = 2$ ,

In  $n = 2$  dimension,

$$\alpha = (x_1 \ x_2) \cdot \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow x_3 \equiv \alpha = a_{11}x_1^2 + (a_{12} + a_{21})x_1x_2 + a_{22}x_2^2$$

Lo que obtenemos, dependiendo de los signos de  $a_{11}$  y  $a_{12}$ , es la ecuación de un paraboloide o un hiperboloide. En la figura 3.9 Se muestra un ejemplo,

Depending of the sign of  $a_{11}$  y  $a_{12}$  we obtain a paraboloid or a hyperboloid equation. Figure 3.9 shows an example.

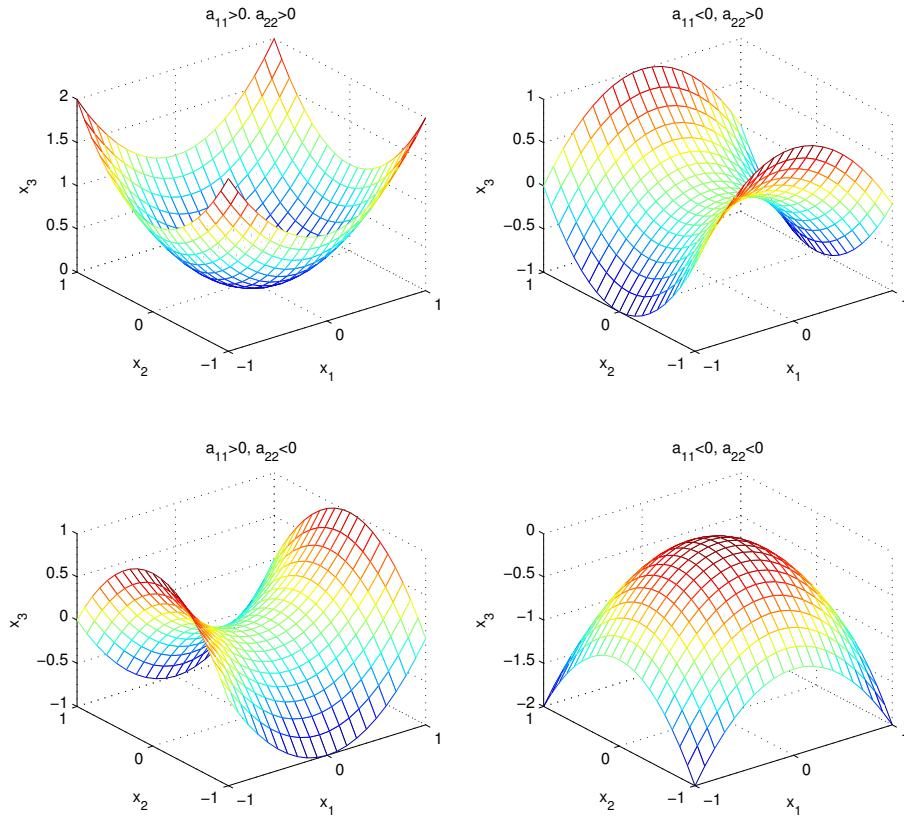


Figura 1.9: Formas cuadráticas asociadas a las cuatro matrices diagonales:  $|a_{11}| = |a_{22}| = 1$ ,  $a_{12} = a_{21} = 0$

Figure 1.9: Quadratic forms obtained using the four diagonal matrices:  $|a_{11}| = |a_{22}| = 1$ ,  $a_{12} = a_{21} = 0$

Veamos brevemente, algunas propiedades de las formas cuadráticas,

1. Una matriz  $A$  de orden  $n \times n$  se dice que es definida positiva si da lugar a una forma cuadrática que es siempre mayor que cero para cualquier vector no nulo,

$$x^T \cdot A \cdot x > 0, \forall x \neq 0$$

2. Una matriz *simétrica* es definida positiva si todos sus *valores propios* (ver sección 3.5.3) son positivos.

3. Una matriz no simétrica  $A$  es definida positiva si su parte simétrica  $A_s = (A + A^T)/2$  lo es.

Briefly we will revise some quadratic forms properties,

1. A matrix  $A$  of dimensions  $n \times n$  is positive definite if it generates a quadratic form always greater than zero for any non-null vector.

2. A *symmetric* matrix is positive definite if all its *eigenvalues* (see section 3.5.3) are positive.

3. A non-symmetric matrix  $A$  is positive definite if its symmetric part  $A_s = (A + A^T)/2$  is so.

$$x \cdot A_s \cdot x > 0, \forall x \neq 0 \Rightarrow x \cdot A \cdot x > 0, \forall x \neq 0$$



## 1.4. Tipos de matrices empleados frecuentemente.

Definimos a continuación algunos tipos de matrices frecuentemente empleados en álgebra, algunos ya han sido introducidos en secciones anteriores; los reunimos todos aquí para facilitar su consulta.

1 Matriz ortogonal: Una matriz  $A_{n \times n}$  es ortogonal cuando su inversa coincide con su traspuesta.

## 1.4. Kinds of matrices frequently used.

We define next some kinds of matrices frequently used in algebra, some of them have been already introduced in previous sections; we put them together here for ease reference.

1 Orthogonal matrix: A matrix  $A_{n \times n}$  is orthogonal whenever its inverse matrix is equal to its transpose.

$$A^T = A^{-1}$$

ejemplo,

$$A = \begin{pmatrix} 1/3 & 2/3 & 2/3 \\ 2/3 & -2/3 & 1/3 \\ 2/3 & 1/3 & -2/3 \end{pmatrix} \rightarrow A \cdot A^T = A^T \cdot A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2. Matriz simétrica: Una matriz  $A_{n \times n}$  es simétrica cuando es igual que su traspuesta,

2. Symmetric Matrix: A matrix  $n \times n$  is symmetric if it is equal to its transpose.

$$A = A^T \rightarrow a_{ij} = a_{ji}$$

ejemplo,

$$A = \begin{pmatrix} 1 & -2 & 3 \\ -2 & 4 & 0 \\ 3 & 0 & -5 \end{pmatrix}$$

3. Matriz Diagonal: Una matriz  $A$  es diagonal si solo son distintos de cero los elementos de su diagonal principal,

3. Diagonal matrix: A matrix  $A$  is diagonal if only the entries of its main diagonal are different from zero.

$$\begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix} \rightarrow a_{ij} = 0, \forall i \neq j$$

4. Matriz triangular superior: Una matriz cuadrada es triangular superior cuando todos los elementos situados por debajo de la diagonal son cero. Es estrictamente triangular superior si además los elementos de la diagonal también son cero,

4. Upper triangular matrix: a square matrix is upper triangular when every entry below its main diagonal is zero, and if it is strictly upper triangular in all the diagonal entries are also zero.

$$TRS \rightarrow a_{ij} = 0, \forall i \geq j$$

$$ETRS \rightarrow a_{ij} = 0, \forall i > j$$

ejemplos,

$$TRS = \begin{pmatrix} 1 & 3 & 7 \\ 0 & 2 & -1 \\ 0 & 0 & 4 \end{pmatrix}$$

$$ETRS = \begin{pmatrix} 0 & 3 & 7 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

5. Matriz triangular inferior: Una matriz es triangular inferior si todos los elementos por encima de su diagonal son cero. Es estrictamente triangular inferior si además los elementos de su diagonal son también cero,

5. Lower triangular matrix: A square matrix is lower triangular when every entry above its main diagonal are zero, and it is strictly lower triangular if all the diagonal entries are also zero.

$$TRI \rightarrow a_{ij} = 0, \forall i \leq j$$

$$ETRI \rightarrow a_{ij} = 0, \forall i < j$$

ejemplos,

$$TRI = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 2 & 0 \\ 7 & -1 & 4 \end{pmatrix}$$

$$ETRI = \begin{pmatrix} 0 & 0 & 0 \\ 3 & 0 & 0 \\ 7 & -1 & 0 \end{pmatrix}$$

6. Matriz definida Positiva. Una matriz  $A_{n \times n}$  es definida positiva si dado un vector  $x$  no nulo cumple,

$$x^T \cdot A \cdot x > 0, \forall x \neq 0,$$

si,

$$x^T \cdot A \cdot x \geq 0, \forall x \neq 0,$$

entonces la matriz  $A$  es semidefinida positiva.

7. Una matriz es (strictamente) diagonal dominante si cada uno de los elementos de la diagonal en valor absoluto es (mayor) mayor o igual que la suma de los valores absolutos de los elementos de la fila a la que pertenece.

6. Positive definite matrix: A matrix  $A_{n \times n}$  is positive definite if taking a non-null vector  $x$  satisfy,

if,

then matrix  $A$  is positive semi-definite.

7. A matrix is (strictly) diagonally dominant if, for every row of the matrix, the magnitude of the diagonal entry in a row is (greater) greater than or equal to the sum of the magnitudes of all the other (off-diagonal) entries in that row.

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \forall i$$

ejemplo,

$$A = \begin{pmatrix} 10 & 2 & 3 \\ 2 & -5 & 1 \\ 4 & -2 & 8 \end{pmatrix} \rightarrow \begin{cases} 10 > 2 + 3 \\ 5 > 2 + 1 \\ 8 > 4 + 2 \end{cases}$$

## 1.5. Factorización de matrices

La factorización de matrices, consiste en la descomposición de una matriz en el producto de dos o más matrices. Las matrices resultantes de la factorización se eligen de modo que simplifiquen, o hagan más robustas numéricamente determinadas operaciones matriciales: Cálculos de determinantes, inversas, etc. A continuación se describen las más comunes.

### 1.5.1. Factorization LU

Consiste en factorizar una matriz como el producto de una matriz triangular inferior  $L$  por una matriz triangular superior  $U$ ,  $A = L \cdot U$ . Por ejemplo,

## 1.5. Matrix factorization.

Factorizing a matrix consists on divide it into the product of two or more matrices. We choose the resulting factor matrices so that they simplify or make certain matrix operations more robust. For instance: matrix determinant calculation, obtaining inverse matrix, etc. Next, we will describe most common matrix factorizations.

### 1.5.1. LU factorization.

We factorize a matrix into the product of a lower triangular matrix  $L$  by an upper triangular one  $U$ ,  $A = L \cdot U$ . For instance,

$$\begin{pmatrix} 3 & 4 & 2 \\ 2 & 0 & 1 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1 & 3/4 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 & 4 & 2 \\ 0 & -8/3 & -1/3 \\ 0 & 0 & -3/4 \end{pmatrix}$$



Una aplicación inmediata, es el calculo del determinante. Puesto que el determinante de una matriz triangular, es directamente el producto de los elementos de la diagonal.

En el ejemplo anterior,

$$|A| = 6 \equiv |L| \cdot |U| = 1 \cdot 1 \cdot 1 \cdot 3 \cdot \left(-\frac{8}{3}\right) \cdot \left(-\frac{3}{4}\right) = 6$$

Describir un método para realizar la factorización LU de una matriz, queda fuera de los objetivos de estos apuntes. Sin embargo sí que utilizaremos el comando `lu` incluido en una librería de Python llamada Scipy para obtener directamente el resultado de una factorización LU. Es una librería parecida a Numpy y que, al igual que esta última contiene un gran numero de funciones matemáticas. De hecho Numpy y Scipy solapan un poco y hay determinadas funciones que pueden encontrarse en ambas librerías. El comando pertenece a un submodulo de Scipy llamado `linalg`. Admite como variable de entrada una matriz cuadrada  $A$  y nos devuelve como salida una matriz de permutación  $P$ , de la que hablaremos enseguida, y una matriz triangular inferior  $L$  y otra triangular superior  $U$  de modo que se cumple que  $A = PLU$ .

A veces obtener la factorización LU directa de una matriz, puede llevar a cálculos que son numéricamente inestables afectando a la precisión del resultado. Una manera de paliar este efecto es permutar entre sí algunas de las filas de la matriz que se quiere factorizar y factorizar la versión permutada.

La permutación de las filas de una matriz  $A$  de orden  $n \times m$ , se puede definir a partir del producto con las matrices de permutación de orden  $n \times n$ . Éstas se obtienen permutando directamente las filas de la matriz identidad  $I_{n \times n}$ . Si una matriz de permutación multiplica a otra matriz por la izquierda, permuta el orden de sus filas. Si la multiplica por la derecha, permuta el orden de sus columnas. Así por ejemplo, para matrices de orden  $3 \times n$ ,

A straightforward application is determinant calculation. Recall that a triangular matrix determinant is the product of their diagonal entries.

So, from the previous example we obtain,

The description of method to calculate a matrix LU factorization is far beyond the scope of this notes. However, we do use the command `lu`, include in a Python library called Scipy, to compute the result of a LU factorization. Scipy is a library very much alike to Numpy and similarly to it, Scipy contains a large number of mathematical functions. In fact, Numpy and Scipy overlaps a little and you can find certain functions in both libraries. The `lu` command belongs to Scipy submodule called `linalg` and takes as input a square matrix  $A$  of dimension  $n \times n$ , and returns as outputs a permutation matrix  $P$  of dimension  $n \times n$  we will describe later on, the lower triangular matrix  $L$ , and the upper triangular matrix  $U$ . The three output matrices satisfy  $A = PLU$ .

Sometimes obtaining directly the LU factorization of a matrix, leads to numerically instable calculations. This, in turn, affect the results precision. One way to overcome this problem is to permute some rows of the matrix we intend to factorize and then, to factorize the permuted version.

We can define the rows permutation of  $A$  matrix of dimensions  $n \times m$  multiplying the matrix by permutation matrices of dimensions  $n \times n$ . We obtain these last permuting the rows of the identity matrix  $I_{n \times n}$ . When a permutation matrix multiplies another matrix by its left side, it permutes the order of the matrix rows. By contrast, if the permutation matrix multiplies another matrix by its right, it permutes the order of the matrix columns. For instance, for matrix of dimensions  $3 \times n$ ,

$$I_{n \times n} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \rightarrow P_{1 \leftrightarrow 3} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Si multiplicamos  $P_{1 \leftrightarrow 3}$  con cualquier otra matriz  $A$  de orden  $3 \times n$ , El resultado es equivalente a intercambiar en la matriz  $A$  la fila 1 con la 3. Por ejemplo,

If we multiply  $P_{1 \leftrightarrow 3}$  by any other matrix  $A$  of dimensions  $3 \times n$ , the result is equivalent to interchange row 1 and 3. For example,

$$P_{1 \leftrightarrow 3} \cdot A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 5 & 3 \\ 4 & 2 & 3 & 0 \\ 3 & 6 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 6 & 2 & 1 \\ 4 & 2 & 3 & 0 \\ 1 & 2 & 5 & 3 \end{pmatrix} = A_{1 \leftrightarrow 3}$$

Volviendo a la factorización LU, vamos a ver como se calcularía para la siguiente matriz,

Coming back to LU factorization, we are going to calculate it for the following matrix,

$$A = \begin{pmatrix} 3 & 4 & 2 & 5 \\ 2 & 0 & 1 & -2 \\ 3 & 2 & 1 & 8 \\ 5 & 2 & 3 & 2 \end{pmatrix}$$

Scipy, calcula siempre la factorización LU, permutando las filas para obtener un resultado lo mas preciso posible,

Scipy always use row permutation, when calculating an LU factorization, to obtain a result as accurate as possible,

```
In [213]: A = np.array([[3,4,2,5],[2,0,1,-2],[3,2,1,8],[5,2,3,2]])
```

```
In [214]: A
```

```
Out[214]:
```

```
array([[ 3,  4,  2,  5],
       [ 2,  0,  1, -2],
       [ 3,  2,  1,  8],
       [ 5,  2,  3,  2]])
```

```
In [215]: import scipy as sp
```

```
In [216]: [P,L,U] = sp.linalg.lu(A)
```

```
In [217]: P
```

```
Out[217]:
```

```
array([[0., 1., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 1., 0.],
       [1., 0., 0., 0.]])
```

```
In [218]: L
```

```
Out[218]:
```

```
array([[ 1.          ,  0.          ,  0.          ,  0.          ],
       [ 0.6         ,  1.          ,  0.          ,  0.          ],
       [ 0.6         ,  0.28571429,  1.          ,  0.          ],
       [ 0.4         , -0.28571429,  0.16666667,  1.          ]])
```

```

In [219]: U
Out[219]:
array([[ 5.,          2.,          3.,          2.],
       [ 0.,          2.8,          0.2,          3.8],
       [ 0.,          0.,        -0.85714286,  5.71428571],
       [ 0.,          0.,          0.,        -2.66666667]])

In [220]: P @ L @ U
Out[220]:
array([[ 3.,  4.,  2.,  5.],
       [ 2.,  0.,  1., -2.],
       [ 3.,  2.,  1.,  8.],
       [ 5.,  2.,  3.,  2.]])

```

### 1.5.2. Factorización de Cholesky

Dada una matriz cuadrada, simétrica y definida positiva, es siempre posible factorizarla como  $A = L \cdot L^T$ . Donde  $L$  es una matriz triangular inferior,

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & \cdots & 0 \\ L_{21} & L_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n2} & \cdots & L_{nn} \end{pmatrix} \cdot \begin{pmatrix} L_{11} & L_{21} & \cdots & L_{n1} \\ 0 & L_{22} & \cdots & L_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L_{nn} \end{pmatrix}$$

Numpy tiene una función, en el submódulo `linalg`, llamada `cholesky`, que permite obtener la factorización de Cholesky.

### 1.5.2. Cholesky factorisation

Given a square symmetric positive definite matrix, it is always possible to factorize it as  $A = L \cdot L^T$ . Where  $L$  is a lower triangular matrix.

Numpy has a function, included in the `linalg` submodule, called `cholesky`, to calculate the cholesky factorization of a matrix.

```

In [224]: B
Out[224]:
array([[47, 28, 26, 45],
       [28, 24, 16, 40],
       [26, 16, 15, 22],
       [45, 40, 22, 97]])

```

```
In [225]: L = np.linalg.cholesky(B)
```

```

In [226]: L
Out[226]:
array([[ 6.8556546 ,  0.          ,  0.          ,  0.          ],
       [ 4.08421976,  2.70539257,  0.          ,  0.          ],
       [ 3.79248978,  0.18874832,  0.76249285,  0.          ],
       [ 6.56392462,  4.87599823, -5.00195311,  2.2627417 ]])

```

```
In [227]: L @ L.T
Out[227]:
array([[47., 28., 26., 45.],
       [28., 24., 16., 40.],
       [26., 16., 15., 22.],
       [45., 40., 22., 97.]])
```

Si la matriz no es definida positiva, la función `cholesky` da un mensaje de error.

If the matrix is not definite positive, then function `cholesky` returns an error message.

### 1.5.3. Diagonalización

**Autovectores y autovalores.** Dada una matriz  $A$  de orden  $n \times n$ , se define como autovector, o vector propio de dicha matriz al vector  $x$  que cumple,

$$A \cdot x = \lambda \cdot x, \quad x \neq 0, \quad \lambda \in \mathbb{C}$$

Es decir, el efecto de multiplicar la matriz  $A$  por el vector  $x$  es equivalente a multiplicar el vector  $x$  por un número  $\lambda$ . Tanto  $x$  como  $\lambda$  pueden ser reales o complejos.

$\lambda$  recibe el nombre de autovalor, o valor propio de la matriz  $A$  asociado al autovector  $x$ . Así por ejemplo,

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

tiene un autovalor  $\lambda = 3$  para el vector propio  $x = [0, -3, 3]^T$ ,

$$\begin{pmatrix} -2 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ -3 \\ 3 \end{pmatrix} = 3 \cdot \begin{pmatrix} 0 \\ -3 \\ 3 \end{pmatrix} = \begin{pmatrix} 0 \\ -9 \\ 9 \end{pmatrix}$$

El vector propio asociado a un valor propio no es único, si  $x$  es un vector propio de una matriz  $A$ , asociado a un valor propio  $\lambda$ , Es trivial comprobar que cualquier vector de la forma  $\alpha \cdot x$ ,  $\alpha \in \mathbb{C}$  también es un vector propio asociado a  $\lambda$ ,

$$A \cdot x = \lambda x \Rightarrow A \cdot \alpha x = \lambda \alpha x \quad (1.1)$$

El conjunto de todos los autovalores de una matriz  $A$  recibe el nombre de espectro de  $A$  y se representa como  $\Lambda(A)$ . Una descomposición en autovalores de una matriz  $A$  se define como,

### 1.5.3. Diagonalisation

**Eigenvectors and eigenvalues.** We define an eigenvector or characteristic vector  $x$  of the  $n \times n$  matrix  $A$  as,

Namely, the result of multiply matrix  $A$  for its eigenvector  $x$  is equal to multiply the number,  $\lambda$  for the eigen vector.  $x$  and  $\lambda$  could be real or complex.

$\lambda$  is call de eigenvalue or characteristic values of the matrix  $A$ , associated to the eigenvector  $x$ . For example,

It has an eigenvalue  $\lambda = 3$  for the eigenvector  $x = [0, -3, 3]^T$ ,

There is not a single eigenvector associated to an eigenvalue, if  $x$  is an eigenvector of a matrix  $A$ , associated to an eigenvalue  $\lambda$ , It is easy to check that any other vector that we build as  $\alpha \cdot x$ ,  $\alpha \in \mathbb{C}$  is also an eigenvector associated to  $\lambda$ ,

we define the set of all eigenvalues of a matrix  $A$  as the spectrum of  $A$  and we represented it as  $\Lambda(A)$ . The eigendecomposition of a matrix  $A$  is defined as,

$$A = X \cdot D \cdot X^{-1}$$

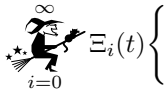
Donde  $D$  es una matriz diagonal formada por los autovalores de  $A$ .

Where  $D$  is a diagonal matrix built using the eigenvalues of  $A$ .

$$\begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & 0 \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

Esta descomposición no siempre existe.

The eigendescomposition does not always exist.



$$\Xi_i(t)$$

En general, si dos matrices  $A$  y  $B$  cumplen,

In general, if two matrices  $A$  and  $B$  satisfy,

$$A = X \cdot B \cdot X^{-1}$$

se dice de ellas que son similares. A la transformación que lleva a convertir  $B$  en  $A$  se le llama una transformación de semejanza. Una matriz es diagonalizable cuando admite una transformación de semejanza con una matriz diagonal de autovalores.

Their a similar matrices. The transformation that convert  $A$  into  $B$  is denoted a similarity transformation. A matrix is diagonalisable when there is a similarity transformation that converts it in a diagonal matrix formed by its eigenvalues.

Volviendo al ejemplo anterior podemos factorizar la matriz  $A$  como,

Coming back to the previous example we can factorise matrix  $A$  as,

$$A = \begin{pmatrix} -2 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & -3 \\ 0 & 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} -2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & -1 \\ 0 & 0 & 1/3 \end{pmatrix}$$

Por tanto en este ejemplo, los autovalores de  $A$  serían  $\lambda_1 = -2$ ,  $\lambda_2 = 2$  y  $\lambda_3 = 3$ . Para estudiar la composición de la matriz  $X$  Reescribimos la expresión de la relación de semejanza de la siguiente manera,

So, in this example the  $A$  eigenvectors would be  $\lambda_1 = -2$ ,  $\lambda_2 = 2$  and  $\lambda_3 = 3$ . To study the structure of matrix  $X$ , we rewrite the similarity relationship as follows,

$$A = X \cdot D \cdot X^{-1} \rightarrow A \cdot X = X \cdot D$$

Si consideramos ahora cada columna de la matriz  $X$  como un vector,

Now, we can consider each column of matrix  $X$  as a vector,

$$A \cdot X = X \cdot D \rightarrow A \cdot (x_1 | x_2 | \cdots | x_n) = (x_1 | x_2 | \cdots | x_n) \cdot \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & 0 \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

Es fácil comprobar que si consideramos el producto de la matriz  $A$ , por cada una de las columnas de la matriz  $X$  se cumple,

It is easy to check that the product of matrix  $A$  by each column of matrix  $X$  satisfy,

$$\begin{aligned} A \cdot x_1 &= \lambda_1 \cdot x_1 \\ A \cdot x_2 &= \lambda_2 \cdot x_2 \\ &\vdots \\ A \cdot x_n &= \lambda_n \cdot x_n \end{aligned}$$

Lo que nos lleva a que cada columna de la matriz  $X$  tiene que ser un autovector de  $A$  puesto que cumple,

$$A \cdot x_i = \lambda_i \cdot x_i$$

En realidad, cada valor propio expande un subespacio  $E_\lambda S$  de  $\mathbf{C}^n$ . Aplicar la matriz  $A$ , a un vector de  $E_\lambda S$  es equivalente a multiplicar el vector por  $\lambda$ . Cada subespacio asociado a un autovalor recibe el nombre de autosubespacio o subespacio propio.

**Polinomio característico.** El polinomio característico de una matriz  $A$  de dimensión  $n \times n$ , se define como,

$$P_A(z) = \det(zI - A)$$

Donde  $I$  es a matriz identidad de dimensión  $n \times n$ . Así por ejemplo para una matriz de dimensión  $2 \times 2$ ,

$$P_A(z) = \det \left( z \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \right) = \begin{vmatrix} z - a_{11} & -a_{12} \\ -a_{21} & z - a_{22} \end{vmatrix} =$$

$$= (z - a_{11}) \cdot (z - a_{22}) - a_{12} \cdot a_{21} = z^2 - (a_{11} + a_{22}) \cdot z + a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$$

Los autovalores  $\lambda$  de una matriz  $A$  son las raíces del polinomio característico de la matriz  $A$ ,

$$p_A(\lambda) = 0$$

Las raíces de un polinomio pueden ser simples o múltiples, es decir una misma raíz puede repetirse varias veces. Por tanto, los autovalores de una matriz, pueden también repetirse. Se llama multiplicidad algebraica de un autovalor al número de veces que se repite.

Además las raíces de un polinomio pueden ser reales o complejas. Para una matriz cuyos elementos son todos reales, si tiene autovalores complejos estos se dan siempre como pares de números complejos conjugados. Es decir si  $\lambda = a + bi$  es un autovalor entonces  $\lambda^* = a - bi$  también lo es.

Veamos algunos ejemplos:

La matriz,

$$A = \begin{pmatrix} 3 & 2 \\ -2 & -2 \end{pmatrix}$$

But then, each column of matrix  $x$  must be an eigenvector of  $A$  because they satisfy,

Actually, each eigenvalue expand a subspace  $E_\lambda S$  of  $\mathbf{C}^n$ . To apply matrix  $A$  to a vector in  $E_\lambda S$  is identical to multiply the eigenvector by  $\lambda$ . Each subspace associated to an eigenvector is called an eigensubspace.

**Characteristic polynomial.** The characteristic polynomial of a matrix  $A$  with dimensions  $n \times n$ , is defined as,

Where  $I$  is the  $n \times n$  identity matrix. For instance, for a matrix with dimensions  $2 \times 2$ ,

The eigenvalues  $\lambda$  of matrix  $A$  are the roots of the characteristic polynomial of matrix  $A$ .

The roots of a polynomial may be single or multiple that is the same root can be repeated several times. So, the autovalues of a matrix can also be repeated.

Besides, the roots of a polynomial may be real or complex. For a matrix with real elements, if it has complex eigenvalues their always pairs complex conjugated numbers. That is, if  $\lambda = a + bi$  is an eigenvalue then  $\lambda^* = a - bi$  is also an eigenvalue.

Let's have a look to some examples:

The matrix,

Tiene como polinomio característico,

has characteristic polynomial,

$$P_A(z) = \begin{vmatrix} z-3 & -2 \\ 2 & z+2 \end{vmatrix} = z^2 - z - 2$$

Igualando el polinomio característico a cero y obteniendo las raíces de la ecuación de segundo grado resultante, obtenemos los autovalores de la matriz  $A$ ,

If we equals the characteristic polynomial to zero and compute the roots of the resulting quadratic equation, we obtain the eigenvalues of matrix  $A$

$$\lambda^2 - \lambda - 2 = 0 \rightarrow \begin{cases} \lambda_1 = 2 \\ \lambda_2 = -1 \end{cases}$$

Un vector propio asociado a  $\lambda_1 = 2$  sería  $x_1 = [2, -1]^T$ ,

An associated eigenvector to  $\lambda_1 = 2$  would be  $x_1 = [2, -1]^T$ ,

$$\begin{pmatrix} 3 & 2 \\ -2 & -2 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix} = 2 \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 4 \\ -2 \end{pmatrix}$$

y un vector propio asociado a  $\lambda_2 = -1$  sería  $x_2 = [1, -2]^T$ ,

An an associated vector to  $\lambda_2 = -1$  would be  $x_2 = [1, -2]^T$ ,

$$\begin{pmatrix} 3 & 2 \\ -2 & -2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -2 \end{pmatrix} = -1 \cdot \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

La matriz,

the matrix,

$$B = \begin{pmatrix} 4 & -1 \\ 1 & 2 \end{pmatrix}$$

tiene como polinomio característico,

has charasteristic polynomial,

$$P_B(z) = \begin{vmatrix} z-4 & -1 \\ 1 & z-2 \end{vmatrix} = z^2 - 6z + 9$$

procediendo igual que en el caso anterior, obtenemos los autovalores de la matriz  $B$ ,

repeating the same procedure as in the previous example, we obtain the eigenvalues of matrix  $B$ ,

$$\lambda^2 - 6\lambda + 9 = 0 \rightarrow \begin{cases} \lambda_1 = 3 \\ \lambda_2 = 3 \end{cases}$$

En este caso, hemos obtenido para el polinomio característico una raíz doble, con lo que obtenemos un único autovalor  $\lambda = 3$  de multiplicidad algebraica 2.

In this case, we obtain a double root for the characteristic polynomial so, we get a single eigenvector with algebraic multiplicity 2.

La matriz,

Matrix,

$$C = \begin{pmatrix} 2 & -1 \\ 1 & 2 \end{pmatrix}$$

tiene como polinomio característico,

has characteristic polynomial,

$$P_C(z) = \begin{vmatrix} z-2 & 1 \\ -1 & z-2 \end{vmatrix} = z^2 - 4z + 5$$

Con lo que sus autovalores resultan ser dos números complejos conjugados,

So, its eigenvalues are two complex conjugate numbers.

$$\lambda^2 - 4\lambda + 5 = 0 \rightarrow \begin{cases} \lambda_1 = 2 + i \\ \lambda_2 = 2 - i \end{cases}$$

Para que una matriz  $A$  de orden  $n \times n$  sea diagonalizable es preciso que cada autovalor tenga asociado un número de autovectores linealmente independientes, igual a su multiplicidad algebraica. La matriz  $B$  del ejemplo mostrado más arriba tiene tan solo un autovector,  $x = [1, 1]^T$  asociado a su único autovalor de multiplicidad 2. No es posible encontrar otro linealmente independiente; por lo tanto  $B$  no es diagonalizable.

La matriz,

For an  $A$  matrix of dimension  $n$  to be diagonalisable it is necessary that any eigenvalue has a number of linearly independent eigenvectors that meet its algebraic multiplicity. Matrix  $B$  in the example showed above has a single eigenvector,  $x = [1, 1]^T$  associated to its single eigenvalue of multiplicity 2; thus, it is impossible to find another eigenvector linearly independent and  $B$  is not a diagonalisable matrix.

The matrix,

$$G = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Tiene un autovalor  $\lambda_1 = 3$ , de multiplicidad 2, y un autovalor  $\lambda_2 = 1$  de multiplicidad 1. Para el autovalor de multiplicidad 2 es posible encontrar dos autovectores linealmente independientes, por ejemplo:  $x_1 = [1, 0, 0]^T$  y  $x_2 = [0, 1, 0]^T$ . Para el segundo autovalor un posible autovector sería,  $x_3 = [-2, 0, 1]^T$ . Es posible por tanto diagonalizar la matriz  $G$ ,

has an eigenvalue  $\lambda_1 = 3$ , with multiplicity 2, and an eigenvalue  $\lambda_2 = 1$  with multiplicity 1. It is possible to find two linearly independent vectors for the eigenvalue of multiplicity 2, for example:  $x_1 = [1, 0, 0]^T$  and  $x_2 = [0, 1, 0]^T$ . A possible eigenvector for the second eigenvalue may be,  $x_3 = [-2, 0, 1]^T$ . Thus, it is possible to diagonalise matrix  $G$ ,

$$G = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} = X \cdot D \cdot X^{-1} = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$



**Propiedades asociadas a los autovalores.** Damos a continuación, sin demostración, algunas propiedades de los autovalores de una matriz,

1. La suma de los autovalores de una matriz, coincide con su traza,

**Eigenvalues associated properties** We include below, without proofs, some properties of a matrix eigenvalues

1. the addition of the eigenvalues of a matrix equals the matrix trace.

$$\text{tr}(A) = \sum_{i=1}^n \lambda_i$$



2. El producto de los autovalores de una matriz coincide con su determinante,

2. the product of the eigenvalues of a matrix equals the matrix determinant

$$|A| = \prod_{i=1}^n \lambda_i$$

3. Cuando los autovectores de una matriz  $A$  son ortogonales entre sí, entonces la matriz  $A$  es diagonalizable ortogonalmente,

3. When the eigenvector of a matrix  $A$  are orthogonal among them, then matrix  $a$  is orthogonally diagonalisable

$$A = Q \cdot D \cdot Q^{-1}; Q^{-1} = Q^T \Rightarrow A = Q \cdot A \cdot Q^T$$

Cualquier matriz simétrica posee autovalores reales y es diagonalizable ortogonalmente. En general, una matriz es diagonalizable ortogonalmente si es normal:  $A \cdot A^T = A^T \cdot A$

Any symmetric matrix has real eigenvalues and is orthogonally diagonalisable. IN general, a matrix is orthogonally diagonalizable if it is normal:  $A \cdot A^T = A^T \cdot A$

4. El mayor de los autovalores en valor absoluto, de una matriz  $A$  de orden  $n$ . recibe el nombre de *radio espectral* de dicha matriz,

4. The maximum absolute value of a  $n \times n$  matrix  $A$  eigenvalues is called the spectral radius of matrix  $A$ .

$$\rho(A) = \max_{i=1}^n |\lambda_i|$$

Numpy incluye en el submodulo `linalg` la función `eig` para calcular los autovectores y autovalores de una matriz. Esta función admite como variable de entrada una matriz cuadrada de dimensión arbitraria y devuelve como variable de salida un vector con los autovalores de la matriz de entrada y una matriz cuadrada en la que cada columna corresponde a un autovalor. A continuación, se muestran dos formas distintas de llamar a la función `eig`. En la línea In [281] se ha realizado la operación  $X \cdot D \cdot X^{-1}$  para comprobar que se cumple que coincide con la matriz  $A$ .

Numpy includes, in the submodule `linalg`, the function `eig` to calculate the eigenvalues and eigenvector of a matrix. Function `eig` takes a square matrix of arbitrary dimensions as input and returns a vector with the eigenvalues of the input matrix and a square matrix, each of its column is an eigenvector. We show next two different ways of calling function `eig`. In line In [281] we computed the operation  $X \cdot D \cdot X^{-1}$  to check that the results coincides with matrix  $A$ .

```
In [273]: A
```

```
Out[273]:
```

```
array([[ 3,  4,  2,  5],
       [ 2,  0,  1, -2],
       [ 3,  2,  1,  8],
       [ 5,  2,  3,  2]])
```

```
In [274]: [autovalores, autovectores] = np.linalg.eig(A)
```

```
In [275]: autovalores
```

```
Out[275]: array([10.71154715, -4.45608147,  0.70096017, -0.95642586])
```

```

In [276]: autovectores
Out[276]:
array([[ -0.5482369 , -0.40905141, -0.34168564,  0.53836293],
       [ -0.05940265,  0.51767323, -0.46625182, -0.17480032],
       [ -0.63104339, -0.60944455,  0.78709433, -0.82325502],
       [ -0.54561146,  0.43962336,  0.2152735 ,  0.04314365]])

In [277]: Results = np.linalg.eig(A)
In [278]: Results
Out[278]:
EigResult(eigenvalues=array([10.71154715, -4.45608147,  0.70096017,
-0.95642586]), eigenvectors=array([[ -0.5482369 , -0.40905141, -0.34168564,  0.53836293],
       [ -0.05940265,  0.51767323, -0.46625182, -0.17480032],
       [ -0.63104339, -0.60944455,  0.78709433, -0.82325502],
       [ -0.54561146,  0.43962336,  0.2152735 ,  0.04314365]]))

In [279]: Results.eigenvalues
Out[279]: array([10.71154715, -4.45608147,  0.70096017, -0.95642586])

In [280]: Results.eigenvectors
Out[280]:
array([[ -0.5482369 , -0.40905141, -0.34168564,  0.53836293],
       [ -0.05940265,  0.51767323, -0.46625182, -0.17480032],
       [ -0.63104339, -0.60944455,  0.78709433, -0.82325502],
       [ -0.54561146,  0.43962336,  0.2152735 ,  0.04314365]])

\begin{center}
\begin{minipage}{\textwidth}
\begin{minted}{python}
In [281]: autovectores@np.diag(autovalores)@np.linalg.inv(autovalores)
Out[281]:
array([[ 3.00000000e+00,  4.00000000e+00,  2.00000000e+00,
         5.00000000e+00],
       [ 2.00000000e+00, -2.33146835e-15,  1.00000000e+00,
        -2.00000000e+00],
       [ 3.00000000e+00,  2.00000000e+00,  1.00000000e+00,
         8.00000000e+00],
       [ 5.00000000e+00,  2.00000000e+00,  3.00000000e+00,
         2.00000000e+00]])

```

#### 1.5.4. Factorización QR

La idea es factorizar una matriz  $A$  en el producto de dos matrices; una matriz ortogonal  $Q$  y una matriz triangular superior  $R$ .

#### 1.5.4. QR Factorisation

The objective of QR factorisation is to divide a matrix  $A$  in the product of two matrices; one of them an orthogonal matrix  $Q$  and the other one an upper triangular matrix  $R$ .

$$A = Q \cdot R, \leftarrow Q \cdot Q^T = I$$

No vamos tampoco a ver en detalle como se calcula la factorización QR. Podemos calcularla empleando la función de Numpy, incluida en el submodulo `linalg`, `[Q,R]=qr(A)`,

We are not going to get into details on how to calculate a QR factorisation. We can use a Numpy function included in submodule `linalg`, `[Q,R]=qr(A)`,

```
In [287]: A = np.array([[1,-2,4],[2,5,3],[1,3,-3]])
```

```
In [288]: [Q,R] = np.linalg.qr(A)
```

```
In [289]: Q@Q.T
```

```
Out[289]:
```

```
array([[1.00000000e+00, 2.56739074e-16, 8.32667268e-17],
       [2.56739074e-16, 1.00000000e+00, 5.55111512e-17],
       [8.32667268e-17, 5.55111512e-17, 1.00000000e+00]])
```

```
In [290]: R
```

```
Out[290]:
```

```
array([[ -2.44948974, -4.4907312 , -2.85773803],
       [  0.          , -4.22295315,  3.51254982],
       [  0.          ,  0.          , -3.67359866]])
```

### 1.5.5. Factorización SVD

Dada una matriz cualquiera  $A$  de orden  $m \times n$  es posible factorizarla en el producto de tres matrices,

### 1.5.5. SVD Factorisation

[eng] Any matrix  $A$  of dimensions  $m \times n$  can be factorised as the product of three matrices,

$$A = U \cdot S \cdot V^T$$

Donde  $U$  es una matriz de orden  $m \times m$  ortogonal,  $V$  es una matriz de orden  $n \times n$  ortogonal y  $S$  es una matriz diagonal de orden  $m \times n$ . Además los elementos de  $S$  son positivos o cero y están ordenados en orden no creciente,

Where  $U$  is an ortogonal matrix of dimensions  $m \times m$ ,  $V$  is an ortogonal matrix of dimensions  $n \times n$  and  $S$  Is a diagonal matrix of dimensions  $m \times n$ . Besides, the entries of  $S$  are positive or zero and are ordered in non increasing order.

$$s = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_i \end{pmatrix}; \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_i; i = \min(m, n)$$

Los elementos de la diagonal de la matriz  $S$ ,  $(\sigma_1, \sigma_2, \cdots \sigma_i)$ , reciben el nombre de *valores singulares* de la matriz  $A$ . De ahí el nombre que recibe esta factorización; SVD son las siglas en inglés de *Singular Value Decomposition*.

The entries of the diagonal of matrix  $S$ ,  $(\sigma_1, \sigma_2, \cdots \sigma_i)$ , are known as the singular values of matrix  $A$ . This is the reason why this factorisation is named *Singular Value Decomposition*, SVD.

No vamos a describir ningún algoritmo para obtener la factorización SVD de una matriz. En Numpy existe la función `[U,s,VT]=svd(A)`,

We are not going to describe any algorithm to get the SVD factorisation of a matrix. In Numpy, the function `[U,s,VT]=svd(A)`, which belongs to the `linalg` submodule, allow

dentro del submodulo `linalg`, que permite obtener directamente la factorización SDV de una matriz  $A$  de dimensión arbitraria. Es importante destacar que Numpy devuelve un vector  $s$  en con los valores singulares y, directamente, la matriz  $V^T$ . Para reconstruir la matriz  $S$  es preciso crear un matriz de dimensión  $m \times n$  a partir del vector  $s$ , primero se crea una matriz diagonal a partir de  $s$  y después se le añaden las filas o columnas de ceros necesarias, hasta completar la mayor de las dimensiones. Si  $n > m$  debemos añadir filas, si  $n < m$  columnas. A continuación se incluyen un ejemplo de uso para una matrice  $A$  no cuadrada,

us to get straighforwardly the SVD matrix factorisation for a matrix  $A$  with arbitrary dimensions. It is important to notice that Numpy returns a vector  $s$  with the singular values and the matrix  $V^T$ , directly. To build the matrix  $S$  we need to create a matrix of dimensions  $m \times n$  from vector  $S$ . To do so, we first create a diagonal matrix with the entries of  $s$  and then, we pad the matrix with rows or columns of zeros till complete the larger of the matrix dimension. If  $n > m$  we have to add rows, if  $n < m$  columns. The following code shows an example for a non-square matrix

```
In [93]: A = np.array([[1,3,4],[2,3,2],[2,4,5],[3,2,3]])
```

```
In [94]: A
Out[94]:
array([[1, 3, 4],
       [2, 3, 2],
       [2, 4, 5],
       [3, 2, 3]])
```

```
In [102]: U,s,VT = np.linalg.svd(A)
```

```
In [105]: S[:s.shape[0],:s.shape[0]] = np.diag(s)
```

```
In [106]: S
Out[106]:
array([[10.25447551,  0.,  0.],
       [ 0.,  1.90114896,  0.],
       [ 0.,  0.,  1.10966868],
       [ 0.,  0.,  0.]])
```

```
In [107]: U@s@VT
Out[107]:
array([[1., 3., 4.],
       [2., 3., 2.],
       [2., 4., 5.],
       [3., 2., 3.]])
```

Como la matriz  $A$  tiene más filas que columnas, la matriz  $S$  resultante termina con una fila de ceros.

A continuación enunciamos sin demostración algunas propiedades de la factorización SVD.

1. El rango de una matriz  $A$  coincide con

Matrix  $A$  has more rows than columns. For this reason, matrix  $S$  ends with a row of zeros.

Next, we include without demonstration some SVD factorisation properties.

1. The rank of a matrix  $A$  coincides with the number of its non-zero singular values.

el número de sus valores singulares distintos de cero.

2. La norma-2 inducida de una matriz  $A$  coincide con su valor singular mayor  $\sigma_1$ . 3. La norma de Frobenius de una matriz  $A$  cumple:

$$\|A\|_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2}$$

4. Los valores singulares de una matriz  $A$  distintos de cero son iguales a la raíz cuadrada positiva de los autovalores distintos de cero de las matrices  $A \cdot A^T$  ó  $A^T \cdot A$ . (Los autovalores distintos de cero de estas dos matrices son iguales),

2. The induced 2-norm of a matrix  $A$  is equal to its largest singular value  $\sigma_1$ .

3. The Frobenius norm of matrix satisfies,

4. The non-zero singular values of a matrix  $A$  are equal to the positive square root of the non-zero eigen values of the matrices  $A \cdot A^T$  ó  $A^T \cdot A$ . (The non-zero eigenvalues of these matrix are equal by obvious reasons),

$$\sigma_i^2 = \lambda_i(A \cdot A^T) = \lambda_i(A^T \cdot A)$$

5. El valor absoluto del determinante de una matriz cuadrada  $A, n \times n$ , coincide con el producto de sus valores singulares,

5. The absolute value of a square  $n \times n$  matrix  $A$  determinant is equal to the product of the matrix singular values.

$$|\det(A)| = \prod_{i=1}^n \sigma_i$$

6. El número de condición de una matriz cuadrada  $A, n \times n$ , que se define como el producto de la norma-2 inducida de  $A$  por la norma-2 inducida de la inversa de  $A$ , puede expresarse como el cociente entre el valor singular mayor de  $A$  y su valor singular más pequeño,

6. The condition number of a square  $n \times n$  matrix  $A$ , which is defined as the product of the  $A$  induced 2-norm by the induced 2-norm of the inverse of  $A$ , may be expressed as the quotient between the largest singular value of  $A$  and its lower singular value,

$$k(A) = \|A\|_2 \cdot \|A^{-1}\|_2 = \sigma_1 \cdot \frac{1}{\sigma_n} = \frac{\sigma_1}{\sigma_n}$$

El número de condición de una matriz, es una propiedad importante que permite estimar cómo de estables serán los cálculos realizados empleando dicha matriz, en particular aquellos que involucran directa o indirectamente el cálculo de su inversa.

The condition number of a matrix, is very useful property that allows us to estimate how stable will be the operations performed using the matrix, in particular, those operations with involve directly or indirectly the computing of the matrix inverse.

## 1.6. Ejercicios

Para los siguientes ejercicios, es necesario importar Numpy. Se puede hacer de tres maneras distintas:

```
import numpy
import numpy as alias
from numpy import *
```

En el primer caso, hay que escribir las funciones precedidas del nombre del módulo: `numpy.fun`. EN el segundo caso, hay que precederlas del alias elegido, en los ejercicios que siguen es la opción elegida

y el alias empleado es np: `np.fun`. EN el tercer caso, importamos directamente las funciones y submódulos por lo que podríamos emplearlos usando directamente su nombre `fun`. Este último método tiene el riesgo de sobrescribir accidentalmente alguna función de Numpy.

1. Arrays en Numpy; matrices y vectores. Escribe, por orden, las siguientes expresiones en la *línea de comandos* de Ipython e interpreta los resultados que obtienes. En los casos en que Python devuelva un mensaje de error, trata de averiguar la razón. **Nota:** Es importante hacerlos por orden ya que algunas operaciones se apoyan en los resultados de operaciones anteriores.

<pre> 1 a =np.array([[1, 2, 3],[4, 5, 6]]) 2 b =np.array([[1, 2, 3],[4, 5]]) 3 c = np.array([1,2,3]) 4 d = np.array([[1,2,3]]) 5 e = np.array([[1],[2],[3]]) 6 f = np.zeros((3,3)) 7 f1 = np.zeros(3) 8 f2 = np.zeros((3,)) 9 f3 = np.zeros((3,1)) 10 g = np.eye(4) 11 h = np.eye(4,3) 12 i = np.eye(4,6) 13 j = np.identity(4) 14 q = np.arange(0,1,0.2) 15 k = np.diag(c) 16 l = np.diag(k) 17 r=np.diag(q,1) 18 r=np.diag(q,2) 19 r=np.diag(q,0) 20 r=np.diag(q,-1) 21 r=np.diag(q,-2)   * indexing and slicing 22 n = a[0,1] 23 du = a[0] 24 duf = a[:,1] 25 duc = a[1,: ] 26 dd = a[0:2,1:3]   * Operadores aplicados a matrices 27 ash = np.shape(a) </pre>	<pre> 28 fa,ca =a.shape 29 fc,cc = np.shape(c) 30 shc = c.shape 31 a1= a.shape[0] 32 a2= c.shape[1] 33 o1=ones(a.shape[1]) 34 o2=ones(c.shape[0]) 35 o3=ones((c.shape[0],a.shape[0])) 36 tt=eye(1,3) 37 p1=a*c 38 p2=c*a 39 p3 = a*e 40 p4 = a@c 41 p5 = c@a 42 p6 = a@e 43 p7 = e@a 44 p10 = dd@duf 45 p11 = duf@dd 46 p8 = a*3 47 b = np.array([[3,5,6],[5,6,7]]) 48 bt=b.T 49 aT=a.T 50 r=a-b 51 z=a-a 52 j1=a+b 53 j2=a.T+b 54 j3 = a.T+b.T 55 d=(a+b)@b.T 56 f=(a+b)*b.T 57 f=(a-b).T*b.T 58 p=a**2 59 r=a**2-a*a </pre>
---	---

<pre> 60 A = np.append(a, [[3,-2,1]],axis=0) 61 B = np.append(b, [[1,0,-1]],axis=0)     * Encadenando operaciones 62 w1 = A@np.linalg.inv(B) 63 w2 = A@np.linalg.matrix_power(B,-1) 64 w3 = A/B 65 w4 = B/A     * importamos linalg / we import       linalg 66 from numpy import linalg as la </pre>	<pre> 67 w5 = A@la.inv(B)-A@la.matrix_power(B,-1) 68 w6 = A@la.matrix_power(B,-1)       -la.solve(la.inv(A),la.inv(B)) 69 C = np.append(A, [[4,0,-3],[2,2,4]],axis=0) 70 C =np.append(C,array([[4,3,2,1,0]]).T,axis       = 1) 71 C1 = C[0:4:2,3] 72 C2=C[0:3,: ] 73 C3 = C[1:3,1:3] </pre>
---	---

2. Define en un terminal de Ipython las matriz  $M$  y  $N$  e interpreta el resultado de las condiciones lógicas impuestas.

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, N = \begin{pmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

nota & es equivalente a **and**

```

y1=M<2
y2=M<=2
y3=M>2
y4=M>=2
y5=M==2
y6=M!=2
y7=any(M<2)
y8=any(M<2,0)
y9=any(M<2,1)
y10=any(M==2)
y11=all(M>=2)
y12=all(M!=0)
y13=all(any(M<2))
y14=any(all(M<2))
index=np.argwhere(M>2)
y15=any(M<2)&any(N>4)
y16=any(M<2)&any(N>10)
y17=any(M<2)&any(N>6)
y18=any(M<2)&any(N>7)
y19=any(M<2)&any(N>3)
y20=any(M<2)&any(N>2)
y21=any(M>2)&any(N>2)
y22=any(M>4)&any(N>2)
y23=any(M>4,0)&any(N>2)
y24=any(M>4,1).T&any(N>2)

```

Repíte el estudio de las condiciones lógicas cambiando & por |, (equivalente a **or**)

3. Obtén empleando el terminal de Ipython el resultado de las siguientes expresiones:

$\frac{\frac{5+3x^2}{6y}}{\frac{1+4\sqrt{2x}}{6}y}$ $\frac{-x + \sqrt{x^2 - 4xy}}{2x}$	$\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(x-1)^2}$ $\sin^2(2x) + \cos^2(2x)$ $\arctan(\infty)$ $\arctan\left(\frac{y}{x}\right)$
--	--

Emplea para  $x$  e  $y$  tanto valores escalares como matrices.

4. Dada una matriz  $A \in \mathbb{R}^{n \times n}$  y un vector columna  $v \in \mathbb{R}^n - \mathbf{0}$ . Se dice que el vector  $v$  pertenece al *Kernel* de  $A$  si se cumple que  $Av = \mathbf{0}$ .  
Crea una función que tome como variables de entrada una matriz  $A$  cuadrada de cualquier dimension  $n \times n$  y un vector columna de dimensión  $n$  y compruebe si pertenece o no al *kernel* de  $A$ . **Nota:** Considera que la condición se cumple si todos los elementos del vector  $Av$  son menores que  $1e-12$ . Hazlo sin emplear el operador @.
5. Escribe —empleando bucles una función que reciba como argumentos de entrada dos matrices  $X$  e  $Y$  y devuelva el valor de su suma  $S = X + Y$ . El programa debe comprobar si es posible la operación indicada y, caso de no serlo, interrumpir la ejecución y mostrar un mensaje de error.
6. Implementa una función SumoDiag.m que tome como entrada una matriz cuadrada  $A$  y que devuelva la suma de todos los elementos de las 2 diagonales principales de dicha matriz. No utilizar la función diag().
7. Escribe —empleando bucles y sin hacer uso del operador @ — que reciba como argumentos de entrada dos matrices  $X$  e  $Y$  y devuelva el valor del producto  $P = XY$ . El programa debe comprobar si es posible la operación indicada y, caso de no serlo, interrumpir la ejecución y mostrar un mensaje de error.



# Índice alfabético

Factorizacion SVD, 67	en Numpy, 12
Funciones	
Funciones incluidas en Numpy, 41	Numpy, 9
Functions	indexación, 14
Functions included in Numpy, 41	SVD Factorisation, 67
Matrices	Vectores
Operaciones Matriciales, 20	en Numpy, 12



# Alphabetic Index

Matrices  
    in Numpy, 12  
    Matrix Operations, 20  
Numpy, 9

Indexing, 14  
Vectors  
    in Numpy, 12