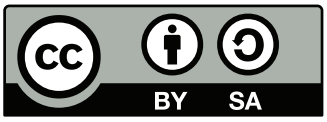
Universidad Complutense de Madrid

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

Juan Jiménez
Héctor García de Marina
Lía García

2 de septiembre de 2024



Índice general

Índice de figuras

Índice de cuadros

Capítulo/Chapter 1

Introducción a matplotlib
Introduction to matplotlib

> El punto geométrico es invisible. De modo que lo debemos definir como un ente abstracto. Si pensamos en él materialmente, el punto se asemeja a un cero. Cero que, sin embargo, oculta diversas propiedades «humanas». Para nuestra percepción este cero —el punto geométrico—está ligado a la mayor concisión. Habla, sin duda, pero con la mayor reserva.
>
> Vasili Kandinski, Punto y línea sobre plano

Visualizar los datos es fundamental para poder comprender y transmitir ideas e información en ciencias e ingeniería. Python tiene numerosas funciones gráficas que para mostrar muchos tipos de gráficos. Una de las librerías para visualizar datos de uso más extendido es matplotlib a la que dedicamos este capítulo.

Importaremos la biblioteca matplotlib de la siguiente manera

```
import matplotlib as mpl
```

matplotlib.pyplot es una colección de funciones que hacen que matplotlib funcione como MATLAB. Cada función de pyplot realiza algún cambio en una figura: por ejemplo, crea

Visualising data is essential for understanding and conveying ideas and information in science and engineering. Python has numerous graphics functions for displaying many types of graphics. One of the most widely used data visualisation libraries is matplotlib to which we dedicate this chapter.

We can import matplotlib as follows:

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: for example, it creates a figure, creates a

una figura, crea un área de trazado en una figura, traza algunas líneas en un área de trazado, decora el trazado con etiquetas, etc.

En matplotlib.pyplot la figura se conserva a través de las llamadas a funciones, de modo que las funciones de trazado se ejecutan sobre los ejes actuales.

Para crear gráficas usando Matplotlib necesitaremos una figura. Cada figura tiene un par (o más) de ejes y un área que es donde se dibujarán los puntos en el sistema de coordenadas que decidamos. Además podremos poner títulos, leyendas etc...

La manera más sencilla de crear una figura es empleando el método `figure` que creará una figura en la que posteriormente podremos añadir ejes, títulos y más cosas. Simplemente llamando a `figure` creamos un objeto de tipo figura, aunque también podemos darle de manera opcional parámetros como el tamaño, el color de fondo y más.

plot area in a figure, draws some lines in a plot area, decorates the plot with labels, etc.

In matplotlib.pyplot the figure is preserved across function calls, so that the plotting functions are executed on the current axes.

To create graphs using Matplotlib we will need a figure. Each figure has a pair (or more) of axes and an area which is where the points will be drawn in the coordinate system of your choice. In addition we can put titles, legends etc...

The easiest way to create a figure is to use the `figure` method which will create a figure to which we can later add axes, titles and more. Simply by calling `figure` we create an object of type figure, although we can also optionally give it parameters such as size, background colour and more.

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(2, 2), facecolor='lightskyblue',
                 layout='constrained')
fig =plt.figure
```

1.1. Dibujar en 2D

La manera más sencilla de dibujar usando matplotlib.pyplot es mediante el método `plot`. A este método le pasaremos dos vectores con las coordenadas x e y de los puntos que queremos representar (evidentemente tendremos que asegurarnos de que tienen la misma longitud).

Por ejemplo si generamos unos vectores de coordenadas como los siguientes $x = (0, 2, -1, -2)$ e $y = (0, 3, 2, -4)$, y se los pasamos al método `plot` pyplot dibujará los puntos (x, y) y los unirá con rectas, como puede verse en la figura **??**.

The easiest way to draw using matplotlib.pyplot is by using the `plot` method. To this method we will pass two vectors with the x and y coordinates of the points we want to represent (obviously we will have to make sure that they have the same length).

For example, if we generate coordinate vectors such as $x = (0, 2, -1, -2)$ and $y = (0, 3, 2, -4)$, and pass them to the pyplot method, it will draw the points (x, y) and join them with straight lines, as can be seen in the figure **??**.

```
import numpy as np
import matplotlib.pyplot as plt

x=np.array([0, 2, -1, -2])
y=np.array([0, 3, 2, -4])
plt.figure
plt.plot(x,y)
```

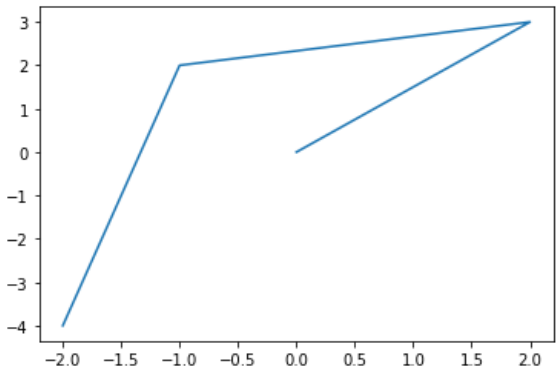


Figura 1.1: Ejemplo sencillo de pyplot.
Figure 1.1: Simple pyplot example.

Símbolo	**Color**
'b'	azul
'r'	rojo
'g'	verde
'm'	magenta
'c'	cian
'y'	amarillo
'k'	negro
'w'	blanco

Tabla 1.1: Colores de los puntos

Symbol	**Color**
'b'	blue
'r'	red
'g'	green
'm'	magenta
'c'	cyan
'y'	yellow
'k'	black
'w'	white

Tabla 1.1: Point colors

Además de las coordenadas de los puntos (x, y) al método `pyplot.plot` podemos darle un tercer argumento indicando el formato (color y forma) con el que queremos que se grafiquen los puntos y opcionalmente la línea que los una.

Algunos de los colores y formatos admitidos están en las tablas **??**, **??** y **??**. Para más información consulta la ayuda de `pyplot.plot`

Se puede combinar un símbolo de cada tipo en un mismo `plot`. Así por ejemplo si queremos representar los datos anteriores unidos mediante una línea de puntos,

```
plt.plot(x,y,':')
```

Si queremos que pinte solo los puntos sin unirlos con líneas y en color rojo,

```
plt.plot(x,y,'.r')
```

Si queremos que pinte los puntos representados por triángulos con el vértice hacia arri-

In addition to the coordinates of the points (x, y), we can give the `pyplot.plot` method a third argument indicating the format (colour and shape) in which we want the points to be plotted and optionally the line that joins them.

Some of the colours and formats supported are in the tables **??**, **??** and **??**. For more information, see the help for `pyplot.plot`.

It is possible to combine one symbol of each type in the same text. So for example if we want to represent the above data joined by a dotted line,

```
plt.plot(x,y,':')
```

If we want it to paint only the dots without joining them with lines and in red,

```
plt.plot(x,y,'.r')
```

If we want it to paint the points represented by triangles with the vertex upwards, joined by a continuous line and in black,

Símbolo	Formato punto
'.'	punto
','	píxel
'o'	círculo
's'	cuadrado
'v'	triángulo hacia abajo
'^'	triángulo hacia arriba
'p'	pentágono
'*'	asterisco

Tabla 1.2: Algunos de los marcadores de puntos

Símbolo	Tipo de línea
'-'	sólida
'_'	discontínua
'-.'	punto-raya
':'	de puntos

Tabla 1.3: Tipos de línea

ba, unidos mediante una línea continua y en color negro,

```
plt.plot(x,y,'-^k')
```

La figura **??** muestra los resultados de las combinaciones de símbolos que acabamos de describir.

Scatter. También podemos dibujar los puntos x, y, por separado sin unirlos usando el método `scatter(x,y)`.

1.1.1. Trabajar con múltiples figuras

Con `matplotlib.pyplot` se pueden dibujar diferentes gráficas en una misma figura usando el método `subplot`. Con este méotod crearemos una figura donde las distintas gráficas estarán dispuestas a modo de tabla. A este método le indicaremos mediante parámetros de entrada el número de filas, el número de columnas y la posición donde se va a dibujar la siguiente gráfica.

En el siguiente ejemplo se crean dos gráficas dentro de la misma figura en dos filas y una columna. En la primera se dibuja la gráfi-

Symbol	Point format
'.'	point
','	pixel
'o'	circle
's'	square
'v'	down triangle
'^'	up triangle
'p'	pentagon
'*'	star

Tabla 1.2: Some point markers

Symbol	Line type
'-'	solid
'_'	dashed
'-.'	dash and dot
':'	dotted

Tabla 1.3: Line types

```
plt.plot(x,y,'-^k')
```

Figure **??** shows the results of the symbol combinations just described.

Scatter. We can also draw the points x, y, separately without joining them using the method `scatter(x,y)`.

1.1.1. Working with multiple figures

With `matplotlib.pyplot` you can draw different graphs in the same figure using the `subplot` method. With this method we will create a figure where the different graphs will be arranged as a table. To this method we will indicate by means of input parameters the number of rows, the number of columns and the position where the next graph is going to be drawn.

In the following example, two graphs are created within the same figure in two rows and one column. In the first one the cosine graph

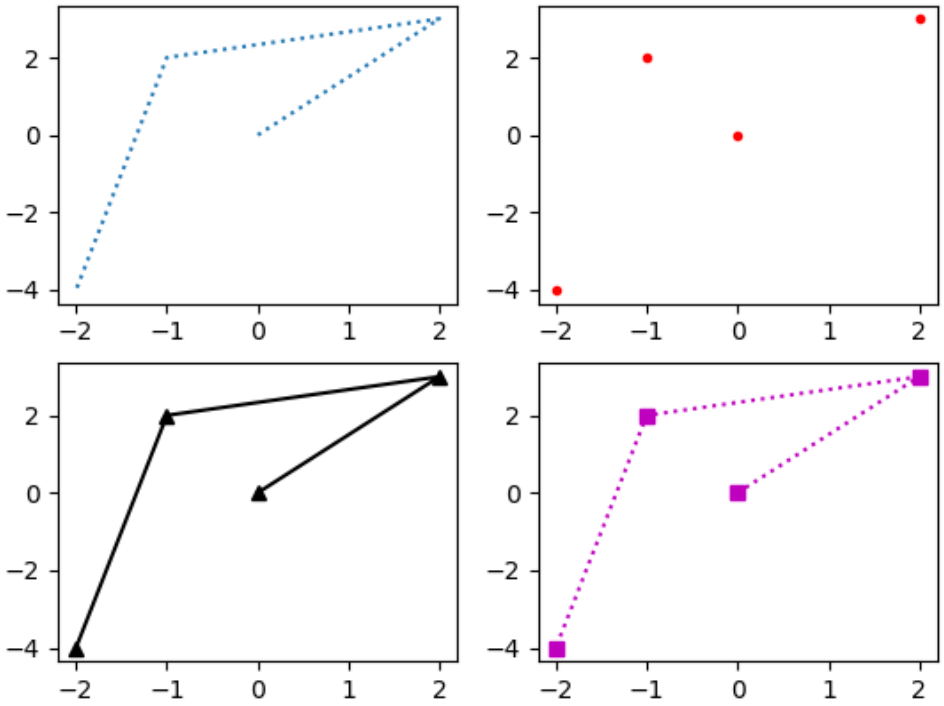


Figura 1.2: Ejemplos de dibujos con diferente formato
Figure 1.2: Different uses of plot

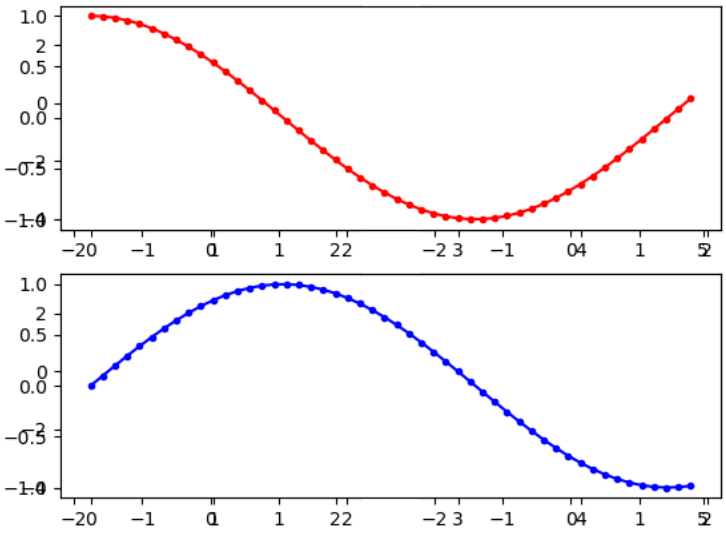


Figura 1.3: Dos gráficas en una misma figura usando `subplot`
Figure 1.3: Two graphs in one figure using `subplot`

ca del coseno y en la segunda la del seno. El resultado puede verse en la figura **??**.

```
import numpy as np
import matplotlib.pyplot as plt
t1 = np.arange(0.0, 5.0, 0.1)
plt.subplot(2,1,1)
plt.plot(t1,np.cos(t1),'r.-')
plt.subplot(2,1,2)
plt.plot(t1,np.sin(t1),'b.-')
```

Se puede añadir texto en cualquier parte del dibujo mediante el método `text` indicando las coordenadas donde se quiere poner el texto y el texto deseado. También se puede poner un texto como título de la figura, usando `title`, o etiquetando los ejes con `xlabel`, `ylabel`.

Por ejemplo el siguiente código da lugar a la figura **??**.

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure()
t=np.arange(0.0,5,0.01)
y=np.exp(-t) * np.cos(2*np.pi*t)
plt.plot(t,y)
```

is drawn and in the second one the sine graph is drawn. The result can be seen in figure **??**.

```
import numpy as np
import matplotlib.pyplot as plt
t1 = np.arange(0.0, 5.0, 0.1)
plt.subplot(2,1,1)
plt.plot(t1,np.cos(t1),'r.-')
plt.subplot(2,1,2)
plt.plot(t1,np.sin(t1),'b.-')
```

Text can be added anywhere on the drawing using the `text` method, indicating the coordinates where the text is to be placed and the desired text. You can also put a text as the title of the figure, using `title`, or by labelling the axes with `xlabel`, `ylabel`.

For example, the following code results in the following figure **??**.

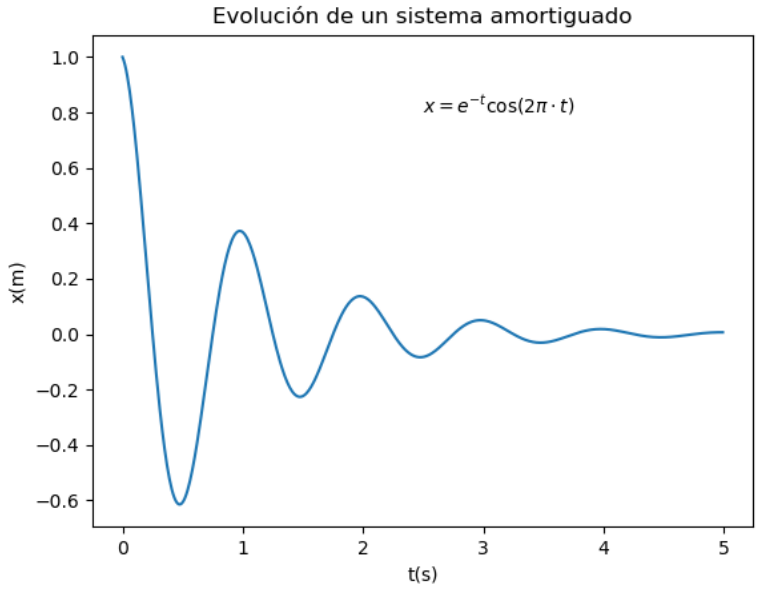


Figura 1.4: Figura con texto en etiquetas
Figure 1.4: Figure with text and labels

```
plt.title("Evolución de un sistema amortiguado")
plt.xlabel('t(s)')
plt.ylabel('x(m)')
plt.text(2.5,0.8,'$x=e^{-t}\cos(2\pi \cdot t)$')
```

1.1.2. Ejes no lineales

`Matplotlib.pytplot` cuenta con un método para definir la escala de los ejes, son los métodos `xscale` e `yscale`. Estos métodos pueden recibir como argumento el tipo de escala del eje, a elegir entre los siguientes:

- `"linear"` para definir una escala lineal
- `"log"` para definir una escala logarítmica de eje positivo
- `"symlog"` para definir una escala logarítmica con tramo positivo y negativo centrado en el cero (como los valores del logaritmo en torno al cero tienden a infinito hay una zona en torno al cero donde considera la escala lineal)
- `"logit"` escala logarítmica entre 0 y 1

1.1.2. Nonlinear axis

`Matplotlib.pytplot` has a method to define the scale of the axes, these are the methods `xscale` and `yscale`. These methods can take as an argument the scale type of the axis, choose from the following:

- 'linear' to define a linear scale
- `'log'` to define a positive-axis logarithmic scale
- `'symlog'` to define a logarithmic scale with a positive and negative span centred at zero (as the values of the logarithm around zero tend to infinity there is a zone around zero where the linear scale is considered)
- 'logit' logarithmic scale between 0 and 1.

Tanto `xscale` como `yscale` permiten definir la escala mediante una función que le definamos nosotros.

En el siguiente ejemplo de código puede verse un ejemplo de uso de los métodos `xscale` e `yscale` para generar la figura **??**.

Como se ve en la figura **??** las divisiones del eje logarítmico (eje y en la segunda figura y ambos en la tercera) aparecen marcadas como potencias de 10. Como estamos representando empleando el logaritmo decimal de la variable, las divisiones se corresponden con el exponente de la potencia de 10 de cada división, $\log_{10}(10^n) = n$.

En segundo lugar hemos empleado un nuevo método: `grid()`. Este método añade una retícula al gráfico de modo que sea más fácil ver los valores que toman las variables en cada punto de la gráfica. Si se invoca el método `grid()` sin parámetros cambia la retícula de visible a invisible y viceversa.

Both `xscale` and `yscale` allow the scale to be defined by a function that we define.

The following code example shows an example of using the `xscale` and `yscale` methods to generate the **??** figure.

As can be seen in the figure **??** the divisions of the logarithmic axis (y-axis in the second figure and both in the third figure) are marked as powers of 10. As we are representing using the decimal logarithm of the variable, the divisions correspond to the exponent of the power of 10 of each division, $\log_{10}(10^n) = n$.

Secondly, we have employed a new method: grid(). This method adds a grid to the graph so that it is easier to see the values taken by the variables at each point on the graph. Calling the `grid()` method without parameters toggles the visibility of the grid.

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0,10,100)
y=np.exp(x)
plt.figure()

plt.subplot(3,1,1)
plt.plot(x,y)
plt.grid()
plt.xlabel("Escala lineal")
plt.ylabel("Escala lineal")
plt.title("$y=e^x$")

plt.subplot(3,1,2)
plt.plot(x,y)
plt.yscale('log')
plt.xlabel("Escala lineal")
plt.ylabel("Escala logarítmica")
plt.grid()


plt.subplot(3,1,3)
plt.plot(x,y)
plt.xscale('log')
plt.yscale('log')
plt.xlabel("Escala logarítmica")
plt.ylabel("Escala logarítmica")
plt.grid()
```

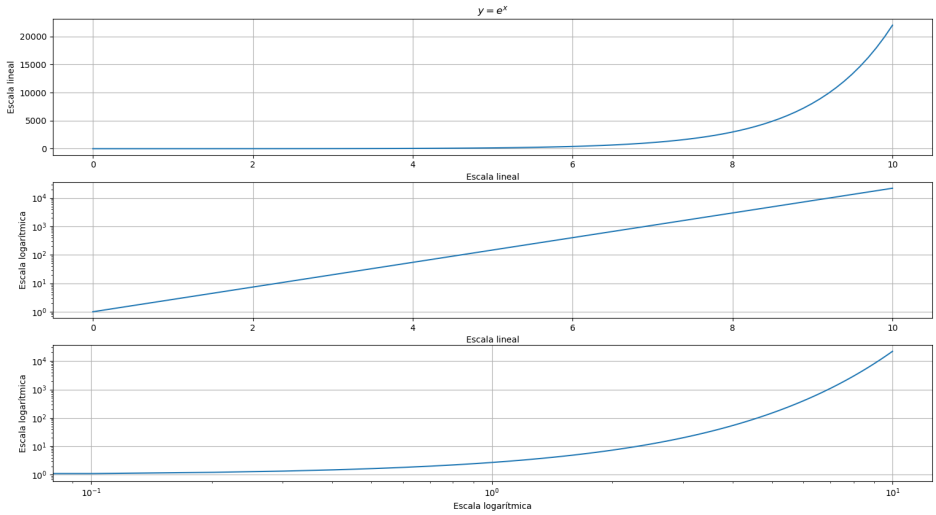


Figura 1.5: Gráfica de la función $y = e^x$ con ejes en escala lineal y logarítmica
Figure 1.5: Graph of the function $y = e^x$ with axes on linear and logarithmic scale

1.1.3. Gráfica en coordenadas polares

Usando el método `polar` podemos representar funciones en coordenadas polares. El primer argumento es un ángulo en radianes y el segundo el correspondiente radio. La figura **??** muestra la espiral,

$$r = 2 \cdot \sqrt{\theta}$$

Para el intervalo angular $[0, 8\pi]$.

```
import numpy as np
import matplotlib.pyplot as plt

theta=np.linspace(0,8*np.pi,100)
radio=2*theta
plt.figure()
plt.polar(theta,radio,'b.-')
plt.grid(visible=True)
plt.title("Espiral en polares")
```

1.1.3. Polar coordinates graph

Using the `polar` method we can plot functions in polar coordinates. The first argument is an angle in radians and the second the corresponding radius. The figure **??** shows the spiral,

$$r = 2 \cdot \sqrt{\theta} \tag{1.1}$$

For the angular interval $[0, 8\pi]$.

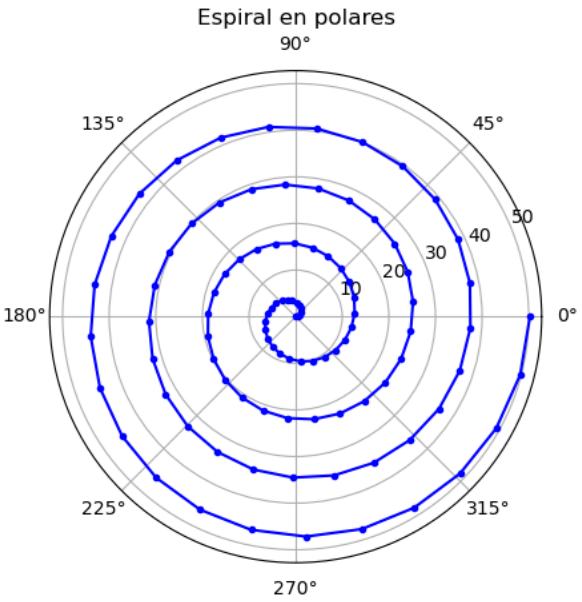


Figura 1.6: Espiral en coordenadas polares
Figure 1.6: Spiral plot using polar coordinates

1.1.4. Otras representaciones gráficas

Con `matplotlib.pyplot` tenemos más opciones para hacer otro tipo de representaciones gráficas.

hist . Este método permite dibujar el histograma de una colección de datos. El histograma representa cuántos datos de una colección caen dentro de un intervalo dado. El método `hist` necesita como parámetro de entrada un vector de datos. Si no se le indica otra cosa los datos se agrupan en 10 intervalos. Si se quiere cambiar el número de intervalos se puede indicar con el parámetro `bin`. En el siguiente ejemplo se dibuja el histograma de los datos de coches por cada 1000 habitantes usando el método `hist` con el número de intervalos por defecto, con 5 intervalos y con 10. Los datos se cargan de un fichero llamado *coches.dat* y la figura **??** es la figura resultante.

```
import numpy as np
import matplotlib.pyplot as plt
```

1.1.4. Other graphical representations

With `matplotlib.pyplot` we have more options to make other types of graphical representations.

hist . This method allows you to draw the histogram of a collection of data. The histogram represents how much data in a collection falls within a given interval. The `hist` method requires a vector of data as input parameter. Unless otherwise specified, the data is grouped into 10 intervals. If you want to change the number of intervals you can specify it with the parameter `bin`. In the following example the histogram of the car data per 1000 inhabitants is drawn using the `hist` method with the default number of intervals, with 5 intervals and with 10 intervals. The data is loaded from a file called *cars.dat* and the resulting figure **??** is the resulting figure.

```
# Abrir el archivo en modo lectura
with open('coches.dat', 'r') as file:
    lineas=file.readlines()
    f=len(lineas)
    n=np.zeros([f])
    i=0
    for row in lineas:
        n[i]=float(row)
        i+=1


plt.figure()

plt.subplot(1,3,1)
plt.hist(n)
plt.grid(visible=True)
plt.title("10 intervalos")
plt.xlabel("Coches por cada 1000 habitantes")
plt.ylabel("Número de paises")
plt.subplot(1,3,2)
plt.hist(n, bins=5)
plt.grid(visible=True)
plt.title("5 intervalos")
plt.xlabel("Coches por cada 1000 habitantes")
plt.ylabel("Número de paises")
plt.subplot(1,3,3)
plt.hist(n,bins=20)
plt.grid(visible=True)
plt.title("20 intervalos")
plt.xlabel("Coches por cada 1000 habitantes")
plt.ylabel("Número de paises")
```

1.2. Dibujar en 3D

En tres dimensiones es posible representar dos tipos de gráficos: puntos y curvas, análogos a los representados en dos dimensiones y además superficies en el espacio.

subplots Este método encapsula todo lo necesario para crear una figura y un conjunto de subfiguras contenidas en ella. Podemos indicarle muchos parámetros como el número de filas y de columnas de la rejilla de subfiguras, si se comparten o no los ejes entre ellas etc. También podemos indicar, mediante un diccionario qué tipo de proyección queremos usar en los ejes creados. Concretamen-

1.2. 3D plots

In three dimensions it is possible to represent two types of graphics: points and curves, analogous to those represented in two dimensions, and also surfaces in space.

subplots This method encapsulates everything needed to create a figure and a set of subfigures contained in it. We can indicate many parameters such as the number of rows and columns of the grid of subfigures, whether or not the axes are shared between them, etc. We can also indicate, by means of a dictionary, which type of projection we want to use in the axes created. Concretely, by means of

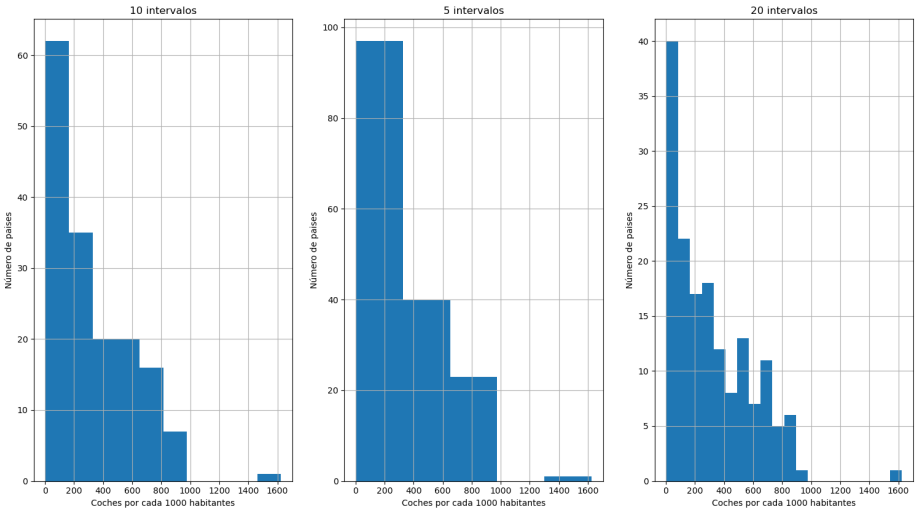


Figura 1.7: Histogramas del número de automóviles por cada 1000 habitantes
Figure 1.7:

te mediante el par clave-valor siguiente estamos indicando que queremos ejes en 3d *subplot_kw="projection":"3d"*.

the following key-value pair, we are indicating that we want 3d axes *subplot_kw='projection': '3d'*.

```
matplotlib.pyplot.subplots(nrows=1, ncols=1, *, sharex=False,
sharey=False, squeeze=True, width_ratios=None, height_ratios=None,
subplot_kw=None, gridspec_kw=None, **fig_kw)
```

Por ejemplo, podemos representar la curva,

$$y = \sin(2\pi x)$$
$$z = \cos(2\pi x)$$

Para ello, seleccionamos un intervalo de valores para $x \in (0, 2)$, y calculamos los correspondientes valores de y y z. Los representamos usando el método *scatter*, figura **??**.

For example, we can represent the curve,

$$y = \sin(2\pi x)$$
$$z = \cos(2\pi x)$$

To do this, we select an interval of values for $x \in (0, 2)$, and calculate the corresponding values of y and z. Then, we plot them using the *scatter* method, Figure **??**.

```
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(0,2,100)
y=np.sin(2*np.pi*x)
z=np.cos(2*np.pi*x)

fig,ax=plt.subplots(subplot_kw={"projection": "3d"})
```

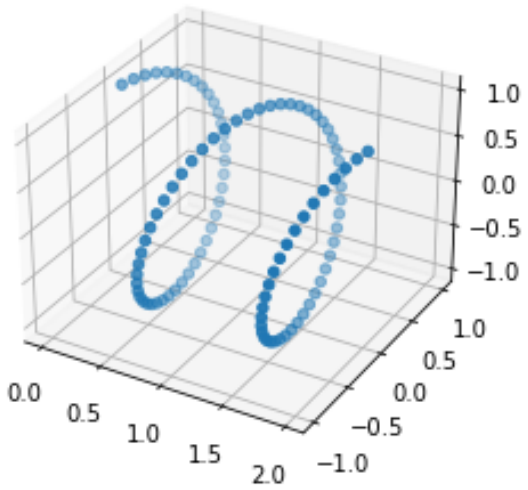


Figura 1.8: Representación en 3 dimensiones usando *scatter*
Figure 1.8: 3D plot using *scatter*

```
ax.scatter(x,y,z)
```

1.2.1. Superfícies

Para dibujar una superfície emplearemos el método *plot_surface*. Igual que en el caso del método *scatter* tendremos que crear previamente una figura y unos ejes en 3 dimensiones usando *subplots* e indicando que se trata de una proyección en 3D.

Para poder dibujar una superfície con *plot_surface* hay que generar una rejilla (X_m, Y_m) y en cada uno de los puntos de esa rejilla calcular el valor de la superfície.

Para definir dicha retícula se necesitan dos matrices. Una de ellas X_m contiene las coordenadas x de los nodos de la retícula y la otra Y_m las coordenadas y. Los elementos que ocupan la misma posición en ambas matrices, representan —juntos— un punto en el plano.

Matplotlib emplea dichas matrices como matrices de *adyacencia*. Cada nodo, $(x_m(i,j), y_m(i,j)$, aparecerá en la gráfica conectado por una arista a cada uno de sus cuatros puntos vecinos, $(x_m(i-1,j), y_m(i-1,j)$, $(x_m(i,j-1), y_m(i,j-1)$, $(x_m(i+1,j), y_m(i+1,j)$, $(x_m(i,j+1), y_m(i,j+1)$. Supongamos que empleamos las siguientes

To draw a surface we will use the *plot-surface* method. As in the case of the *scatter* method, we will have to previously create a figure and axes in 3 dimensions using *subplots* and indicating that it is a 3D projection.

In order to draw a surface with *plot_surface* we must generate a grid (X_m, Y_m) and at each of the points of this grid calculate the value of the surface.

To define such a grid, two matrices are needed. One of them X_m contains the x coordinates of the grid nodes and the other Y_m the y coordinates. Elements occupying the same position in both matrices represent - together - a point in the plane.

Matplotlib uses these matrices as adjacency matrices. Each node, $(x_m(i,j), y_m(i,j)$, will appear in the graph connected by an edge to each of its four neighbouring points, $(x_m(i-1,j), y_m(i-1,j)$, $(x_m(i,j-1), y_m(i,j-1)$, $(x_m(i+1,j), y_m(i+1,j)$, $(x_m(i,j+1), y_m(i,j+1)$. Suppose we use the following matrices, X_m and Y_m to define a lattice on which to draw a surface,z.

matrices, X_m y Y_m para definir una retícula sobre la que dibujar una superficie,

$$X_m = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{pmatrix}, Y_m \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{pmatrix} \xrightarrow[nodos]{posiciones} \begin{matrix} (0,0) & - & (1,0) & - & (2,0) & - & (3,0) \\ | & & | & & | & & | \\ (0,1) & - & (1,1) & - & (2,1) & - & (3,1) \\ | & & | & & | & & | \\ (0,2) & - & (1,2) & - & (2,2) & - & (3,2) \\ | & & | & & | & & | \\ (0,3) & - & (1,3) & - & (2,3) & - & (3,3) \end{matrix}$$

```
Axes3D.plot_surface(X, Y, Z, *, norm=None, vmin=None, vmax=None,
lightsource=None, **kwargs)
```

La estructura de las matrices X_m e Y_m del ejemplo anterior, es la típica de las matrices de adyacencia de una retícula cuadrada; la matriz X_m tiene la filas repetidas y la matriz Y_m tiene repetidas la columnas. En el ejemplo las matrices son cuadradas y definen una retícula de 4×4 nodos. En general, podemos definir una retícula rectangular de $m \times n$ nodos. En este caso las matrices empleadas para definir la retícula tendrían dimensión $m \times n$.

Para dibujar con Matplotlib superficies podemos en primer lugar definir la retícula a partir de dos vectores de coordenadas empleando el método de numpy `meshgrid`. En el ejemplo que acabamos de ver, hemos empleado una retícula que cubre el intervalo, $x \in [0, 3]$ e $y \in [0, 3]$. para definirlo creamos los vectores,

```
x=np.arange(0,4)

x
Out[23]: array([0, 1, 2, 3])

y=np.arange(0,4)

y
Out[25]: array([0, 1, 2, 3])
```

A continuación empleamos el método `meshgrid` para construir las dos matrices de adyacencia. Numpy se encargará de repetir las filas y columnas necesarias,

```
    [Xm,Ym]=np.meshgrid(x,y)
Xm
Out[20]:
array([[0, 1, 2, 3],
```

The structure of the X_m and Y_m matrices in the previous example is typical of the adjacency matrices of a square lattice; the X_m matrix has repeated rows and the Y_m matrix has repeated columns. In the example the matrices are square and define a lattice of 4×4 nodes. In general, we can define a rectangular lattice of *timesn* nodes. In this case the matrices used to define the lattice would have dimension $m \times n$.

To draw with Matplotlib surfaces we can first define the grid from two coordinate vectors using the numpy method `meshgrid`. In the example we have just seen, we have used a grid covering the interval, $x \in [0, 3]$ and $y \in [0, 3]$. To define it we create the vectors,

```
x=np.arange(0,4)

x
Out[23]: array([0, 1, 2, 3])

y=np.arange(0,4)

y
Out[25]: array([0, 1, 2, 3])
```

Next, we use the method `meshgrid` to construct the two adjacency matrices. Numpy will take care of repeating the necessary rows and columns,

```
    [Xm,Ym]=np.meshgrid(x,y)
Xm
Out[20]:
array([[0, 1, 2, 3],
       [0, 1, 2, 3],
```

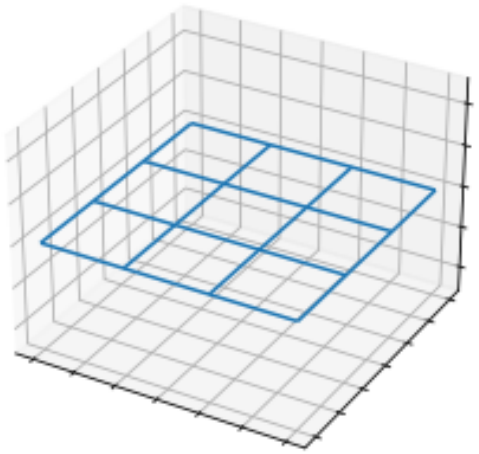


Figura 1.9: Retícula plana
Figure 1.9: Flat grid

```
       [0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3]])

Ym
Out[21]:
array([[0, 0, 0, 0],
       [1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 3, 3, 3]])
```

Una vez construidas las matrices de adyacencia, solo necesitamos una matriz de valores para Z_m. Si definimos por ejemplo,

```
Zm=np.zeros(np.shape(Xm))

Zm
Out[35]:
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

Podríamos representar la retícula plana de la figura **??**, empleando por ejemplo el comando `plot_wireframe(Xm, Ym, Zm)`.

Una vez que hemos visto como construir

```
       [0, 1, 2, 3],
       [0, 1, 2, 3]])

Ym
Out[21]:
array([[0, 0, 0, 0],
       [1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 3, 3, 3]])
```

Once the adjacency matrices are constructed, we only need a matrix of values for Z_m. If we define for example,

```
Zm=np.zeros(np.shape(Xm))

Zm
Out[35]:
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

We could represent the flat grid of the figure **??**, using for example the command `plot_wireframe(Xm, Ym, Zm)`.

Now that we have seen how to construct

una retícula rectangular sobre la que construir una superficie, veamos como dibujarla con un ejemplo. Supongamos que queremos dibujar la superficie,

$$z = x^3 + y^2$$

En la región del plano, $x \in [-1.5, 1.5]$, $y \in [-2, 2]$.

Igual que en el ejemplo inicial, lo primero que debemos hacer es construirnos una matrices de adyacencia que definan una retícula en la región de interés,

```
x=np.linspace(-1.5,1.5,25)
y=np.linspace(-2,2,50)
[Xm,Ym]=np.meshgrid(x,y)
```

Es interesante notar que la región de interés no es cuadrada y que las matrices de adyacencia tampoco los son (50×25). Además los puntos no están espaciados igual en los dos ejes.

A continuación obtenemos la matriz de coordenadas z, aplicando la función a los puntos de la retícula,

```
Zm=Xm**3+Ym**2
```

Podemos representar la superfície usando el método *wireframe* o el método *surface*. En el primer caso se dibuja la superfície como una rejilla, figura **??** y en el segundo como las caras rellenas de color de la retícula, figura **??**.

a rectangular grid on which to build a surface, let's see how to draw it with an example. Suppose we want to draw the surface,

$$z = x^3 + y^2$$

In the region of the plane, $x \in [-1.5, 1.5]$, $y \in [-2, 2]$.

As in the initial example, the first thing to do is to construct adjacency matrices that define a grid in the region of interest,

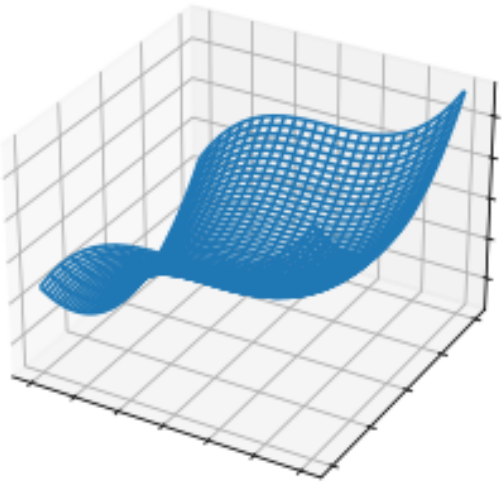
```
x=np.linspace(-1.5,1.5,25)
y=np.linspace(-2,2,50)
[Xm,Ym]=np.meshgrid(x,y)
```

It is interesting to note that the region of interest is not square and that the adjacency matrices are not square either (50×25). Also the points are not equally spaced on the two axes.

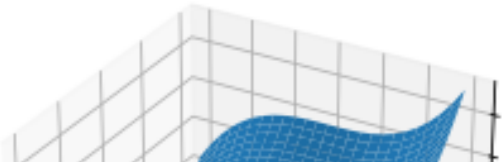
Next we obtain the z-coordinate matrix by applying the function to the grid points,

```
Zm=Xm**3+Ym**2
```

We can represent the surface using the *wireframe* method or the *surface* method. In the first case the surface is drawn as a grid, figure **??**, and in the second as the colour-filled faces of the grid, figure **??**.



(a) Función $z = x^3 + y^2$ representada con `wireframe`



1.3. Animaciones

Matplotlib también proporciona una interfaz para generar animaciones utilizando el módulo *animation*. Una animación es una secuencia de fotogramas en la que cada fotograma corresponde a un trazado sobre una figura.

Para crear una animación en python usando *matplotlib.animation* seguiremos los siguientes pasos:

1. Inicializar los meta datos para la animación.
2. Inicializar el fondo de la animación.
3. Definir los objetos que van a ir cambiando en cada fotograma.

Inicializar los metadatos . Inicializar los meta datos para la animación creando un diccionario con ellos y pasándoselos al método que va a crear la animación *writers*. Le indicamos tanto los metadatos como el número de frames por segundo con el parámetro `fps`.

1.3. Animations

Based on its plotting functionality, Matplotlib also provides an interface to generate animations using the *animation* module. An animation is a sequence of frames where each frame corresponds to a plot on a figure.

To create an animation in python using *matplotlib.animation* we will follow the following steps:

1. Initialise the meta data for the animation.
2. Initialise the background of the animation.
3. Define the objects that are going to change in each frame.

Initialize the meta data. Initialise the meta data for the animation by creating a dictionary of it and passing it to the method that will create the animation *writers*.We indicate both the metadata and the number of frames per second with the parameter `fps`.

```
FFMpegWriter=manimation.writers["ffmpeg"]
metadata = dict(title="GIF Test", artist ="Me",
comment="A red circle following a blue sine wave")
writer = FFMpegWriter(fps=15, metadata=metadata)
```

Inicializar el fondo de la animación , es decir aquellos elementos que no van a cambiar. Lo haremos definiendo una figura y todo aquello que va a permanecer constante durante la animación, por eso lo llamamos el fondo. En este caso es la curva del seno. También inicializamos el punto vacío rojo, al que le damos unas coordendas vacías, para que Python sepa que esas coordenadas se actualizarán posteriormente. Como las etiquetas no cambian también las ponemos aquí.

Initialise the background of the animation , i.e. those elements that are not going to change. We will do this by defining a figure and everything that will remain constant during the animation, which is why we call it the background. In this case it is the sine curve. We also initialise the red empty point, to which we give empty coordinates, so that Python knows that these coordinates will be updated later. Since the labels don't change, we also put them here.

```
fig =plt.figure()
n=1000
x=np.linspace(0, 6*np.pi,n)
y=np.sin(x)
sine_line, =plt.plot(x,y,"b")
red_circle,=plt.plot([],[],"ro",markersize=10)
```

```
plt.xlabel("x")
plt.ylabel("sin(x)")
```

Definir los objetos que van a ir cambiando en cada fotograma . Salvaremos un fichero, indicando qué figura vamos a salvar, el nombre del fichero (`movie_test.mpg`) y la resolución de la figura en puntos por pulgada (dpi). En el bucle *for* se actualiza repetidamente la figura para crear el movimiento, en cada iteración se actualiza la posición del punto rojo (usando `set_data`). El método `grab_frame` captura estos cambios y los muestra con los frames por segundo especificados anteriormente.

Define the objects that are going to change in each frame. We will save a file, indicating which figure we are going to save, the name of the file (`movie_test.mpg`) and the resolution of the figure in dots per inch (dpi). In the *for* loop the figure is repeatedly updated to create the movement, in each iteration the position of the red dot is updated (using `set_data`). The `grab_frame` method captures these changes and displays them at the frames per second specified above.

```
with writer.saving(fig,"movie_test.mp4",100):
    for i in range(n):
        x0=x[i]
        y0=y[i]
        red_circle.set_data([x0],[y0])
        writer.grab_frame()
```

1.4. Ejercicios

1. 1. Escribe una función que muestre paso a paso en una figura la trayectoria del tiro parabólico para $v_0 = 40$ (m/s) $\theta = 45$ grados y para $v_0 = 60$ (m/s) $\theta = 60$ grados. En cada instante se mostrará con un círculo rojo la posición (x, y).

$$x = v_0 \cdot \cos(\theta) \cdot t$$
$$y = v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

   Añade el nombre de cada eje y el título de la figura.

2. Usa únicamente una sola figura para representar en diferentes paneles las siguientes funciones.

   - $y = e^{-x^2}$ en el intervalo $[-2, 2]$ empleando 100 datos equiespaciados
   - $y = e^{-((\frac{x}{2})^2))}$
   - $y = e^{-((2x)^2)}$
   - La función $y = e^{-((2x)^2)}$ pero en escala logarítmica

3. Crear una función dibujatriangul que tenga como argumento de entrada los 3 vértices de un triángulo y devuelva una figura con el triángulo dibujado.

4. Crea una función *dibujaMUC* que dibuje la trayectoria circular de una partícula conociendo su velocidad angular ω y el radio de giro r. Las ecuaciones cartesianas son:

$$x = r\cos(\omega \cdot t)$$
$$y = r\sin(\omega \cdot t)$$

   - Añadir a la trayectoria en cada punto el vector velocidad con el comando quiver mediante las ecuaciones de la velocidad:

$$v_x = -r \cdot \omega \cdot \sin(\omega \cdot t)$$
$$v_x = r \cdot \omega \cdot \cos(\omega \cdot t)$$

5. Crea una función que tenga como parámetros de entrada el radio r y la velocidad angular ω y dibuje la trayectoria en 3 dimensiones de una partícula que describe un MCU en el plano XY moviéndose en el eje z con una velocidad constante v_z.

6. Crea una función que represente el tiro parabólico en 3D sabiendo que la función tiene como entrada los ángulos ψ y θ , de acuerdo con las siguientes expresiones:

$$x = v_0 \cdot \cos(\psi) \cdot \cos(\theta) \cdot t$$
$$y = v_0 \cdot \sin(\psi) \cdot \cos(\theta) \cdot t$$
$$z = v_0 \cdot \sin(\theta) \cdot t - \frac{1}{2} \cdot g \cdot t^2$$

7. Usando el Comando meshgrid, crea una retícula cuadrada en el intervalo $x = [-11]$ e $y = [-22]$ emplea para ello un paso de malla de valor 0.1. Crea una matriz de ceros del tamaño de las matrices que definen la retícula. Representa, empleando el comando mesh, la matriz de ceros creada sobre la retícula. Representa gráficamente la superficie $z = e^{-(x^2+y^2)}$.

8. Genera 1000 números aleatorios distribuidos normalmente (usa para ello la función *np.random.randn*. Usa la función *plt.hist* para dibujar el histograma de los números aleatorios generados.
   - Sepáralos en 10 contenedores.
   - Crea un diagrama de barras de los datos del apartado anterior usando *plt.bar*.
   - ¿Crees que la función *np.random.randn* es una buena aproximación de una distribución normal?
9. Crea una función *my_poly_plot(n,x)* que dibuje los polinomios $p(x) = x^k$ para $k = 1, ..., n$.
10. Supón que tienes tres puntos que forman las esquinas de un triángulo equilátero $P_1 = (0, 0)$, $P_2 = (0.5, \frac{\sqrt{2}}{2})$ y $P_3 = (1, 0)$. Crea una función que:
    - Genere un conjunto de n puntos $p_i = (x_i, y_i)$ tales que $p_1 = (0, 0)$ y p_{i+1} es el punto medio entre p_i y P_1 con un 33 % de probabilidad, el punto medio entre p_i y P_2 con un 33 % de probabilidad y el punto medio entre p_i y P_3 con un 33 % de probabilidad.
    - Dibuje los puntos obtenidos en el plano.
    - Prueba la función para $n = 100$ y $n = 1000$ puntos.

1.5. Test del curso 2020/21

1. El movimiento de un cuerpo en el plano viene descrito por las siguientes ecuaciones,

$$x(t) = \sin(2\pi\,\omega_1 t) \tag{1.1}$$
$$y(t) = \cos(2\pi\,\omega_2 t) \tag{1.2}$$
$$z(t) = a\cos(2\pi\,\omega_1 t) \tag{1.3}$$

donde x, y, z representan las coordenadas del cuerpo en el instante de tiempo t medidas en metros, w_1 y w_2 son frecuencias fijas medidas en $rad \cdot s^{-1}$ y a es una amplitud fija medida en metros.

a) (**1.5 puntos**) Escribe una función que:

1) Tome como variables de entrada las frecuencias w_1, w_2, la amplitud a .
2) Haga el cálculo de la trayectoria en tres dimensiones descrita por el cuerpo en el intervalo de tiempo $[0, 1]$. Considera incrementos de tiempo de $0.01s$.
3) Dibuje en una gráfica la trayectoria descrita por el cuerpo en el espacio. Cada eje del gráfico deberá llevar una etiqueta indicando de qué variable se trata x, y ó z.
4) La salida serán tres vectores con los valores de x, y y z calculados para cada instante de tiempo.

b) (**1.5 puntos**) Añade a la función creada en el apartado anterior el código necesario para que:

1) **Solo si** se le dan dos variables de entrada en lugar de tres, entonces calcule la trayectoria que describiría el cuerpo en el plano si se eliminara la ecuación (**??**) de las ecuaciones del sistema. (NOTA: La función debe seguir funcionando igual que en el apartado anterior si se le dan tres entradas. Emplea el comando nargin).

2) El resultado deberá también representarse gráficamente, pero solo en dos dimensiones, y devolver la variable z como un vector vacío.

c) (**1.5 puntos**) Genera dos vectores de frecuencias: `W1` y `W2` de modo que el primero contenga los números impares comprendidos entre 1 y 7 y el segundo los números pares comprendidos entres 2 y 8. Emplea la función creada en el apartado anterior para calcular el valor de las trayectorias obtenidas tomando como entradas para ω_1, ω_2, todos los pares posibles de la forma: `W1(i)` y `W2(j)`. Supón que no hay tercera entrada a. Realiza los cálculos empleando bucles.

d) (**1 punto**) Añade a tu programa el código necesario para que dibuje cada resultado en un *subplot* de modo que la figura resultante tenga 4×4 *subplots* (ver figura al dorso). Debes crear los *subplots* empleando los mismos bucles del apartados anterior.

e) (**0.5 puntos**) Repite los cálculos de los apartados c) y d) pero tomando ahora $a = 1$.

2. Una matriz cuadrada A, de dimensión arbitraria $(n \times n)$, cuyos elementos $a_{ij} \in \mathbb{Z}$, se define como *buenrollista* cuando las suma de los valores pares de cada fila es mayor o igual que la suma de los valores impares de la columna correspondiente, i.e., satisface la siguiente relación:

$$\sum_{j=1}^{n} a_{ij(\text{par})} \geq \sum_{j=1}^{n} a_{ji(\text{impar})}, \quad \forall i \in \{1, ..., n\} \tag{1.4}$$

a) (**1.5 puntos**) Escribe una función que tome como variable de entrada una matriz cuadrada de cualquier dimensión y calcule un vector con las sumas de los valores pares de los elementos de cada una de su filas y otro vector con las sumas de los elementos impares de cada una de sus columnas.

b) (**1.5 puntos**) Añade a la función anterior el código necesario para que compruebe, empleando los vectores obtenidos en el apartado anterior, si la función es *buenrollista* y muestre un mensaje por pantalla indicando si lo es o no.

c) (**1 punto**) Aplica la función desarrollada a la siguiente matriz:

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

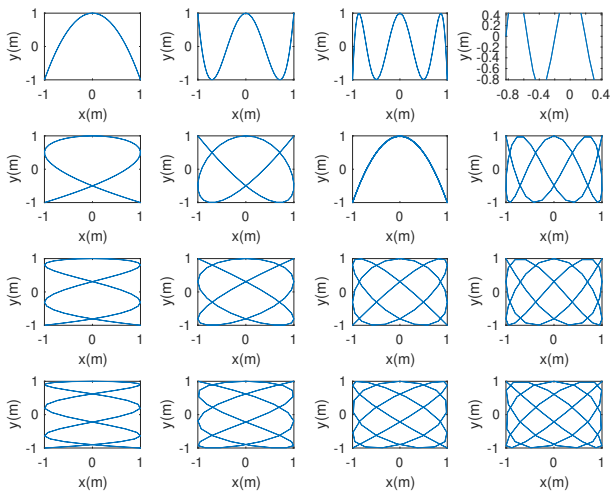


Figura 1.11: vista de la figura con los ocho subplots