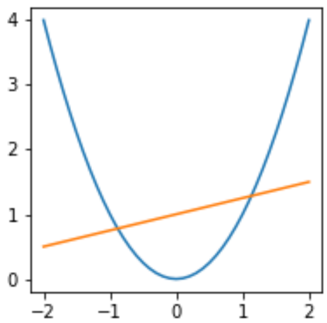
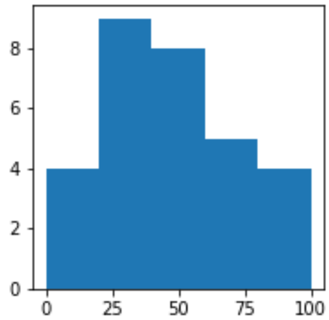
PW4: Matplotlib, function plotting, histograms and scatter plots

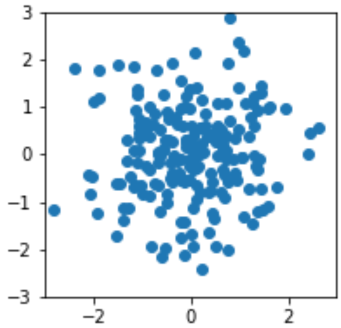
1 - Introduction

1.1 - Definition

- `matplotlib` is a "Python library" for drawing figures.
- It allows you to represent graphs of functions, histograms, or scatterplots such as:







- This library must be imported before it can be used. This is done with the `import` instruction.
- The complete documentation of `matplotlib` is online: [https://matplotlib.org (https://matplotlib.org)](https://matplotlib.org)

1.2 - `matplotlib.pyplot`

- We will not use the whole `matplotlib` library but only the sub-library `matplotlib.pyplot` sufficient for our needs.
- We assign an **alias** to this import with the command

```
import matplotlib.pyplot as plt
```

- This will make the python code easier to write.
- The complete documentation of `matplotlib.pyplot` is online: [https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)

1.3 - Exercise 1 :

Import the libraries `numpy` and `matplotlib.pyplot` with the appropriate aliases.

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
```

2 - Figures

2.1 - Creating a figure

- **Before doing any plotting**, you must create a figure with the `figure` function:

```
plt.figure()
```

- Parentheses are mandatory.
- The `figure` function accepts optional arguments which are then indicated between the parentheses. We give examples in the following.

In [3]:
```
plt.figure()
```

Out[3]:
```
<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>
```

2.2 - Plotting a curve

- To draw a curve, we use the `plot` function with two arguments `x` and `y` which are **arrays of the same length**, containing numerical values `x[i]` and `y[i]`.

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

- `x` and `y` can also be **lists** or **tuples**.
- The `plot` function draw the lines connecting the points of coordinates (`x[i]` , `y[i]`).
- The `plot` function accepts optional arguments. We give examples in the following.

2.3 - Plotting the graph of a function

- To plot the graph of a function `f` , we must first create the arrays `x` and `y` .
- We create the array `x` with the command :

```
x = np.arange(a,b,s)
```

- The array thus created contains the numbers `[a, a+s, a+2*s, etc. a+k*s]` .
- The number `s` is called the **step** of the series of numbers.
- The last term of the array is such that `a+ks < b ≤ a+(k+1)s` .
- For example, the command

```
x = np.arange(0,10,1)
```

  creates the list `x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
- We create the array `y` with the command

```
y = f(x)
```

  where `f` is the desired function that we have defined beforehand.

2.4 - Exercise 2 : graph of the Gauss function

- The function `gauss` whose code is given below returns the value of the function:

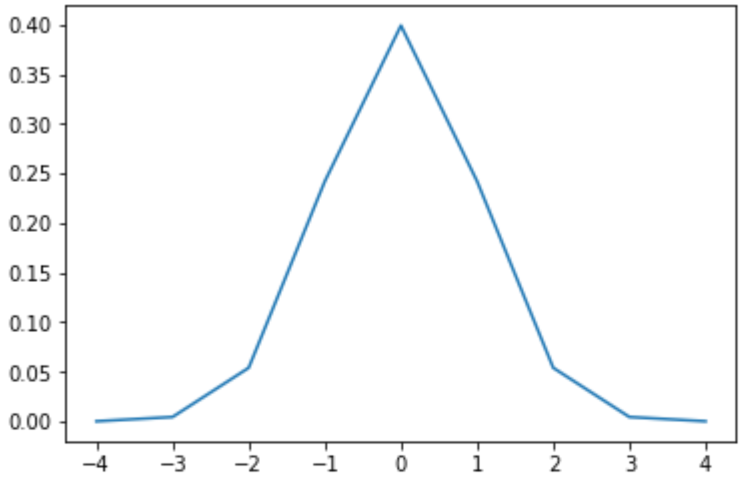
$$\mathrm{gauss}(x) = \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\pi}}$$

In [4]:
```
def gauss(x):
    return np.exp(-x**2/2)/np.sqrt(2*np.pi)
```

- Create an array `x` containing the integers between -4 and 4 (inclusive).
- Create an array `y` containing the values of `gauss` function applied to the elements of `x` .
- Display `x` and `y` to check the result (namely that `x` contains the desired values and that `x` and `y` have the desired length).
- Create a new figure and draw the graph of the function `gauss` .

In [9]:
```
x = np.arange(-4,5,1)
y = gauss(x)
print(x)
print(y)
fig = plt.plot(x,y)
```

```
[-4 -3 -2 -1  0  1  2  3  4]
[1.33830226e-04 4.43184841e-03 5.39909665e-02 2.41970725e-01
 3.98942280e-01 2.41970725e-01 5.39909665e-02 4.43184841e-03
 1.33830226e-04]
```



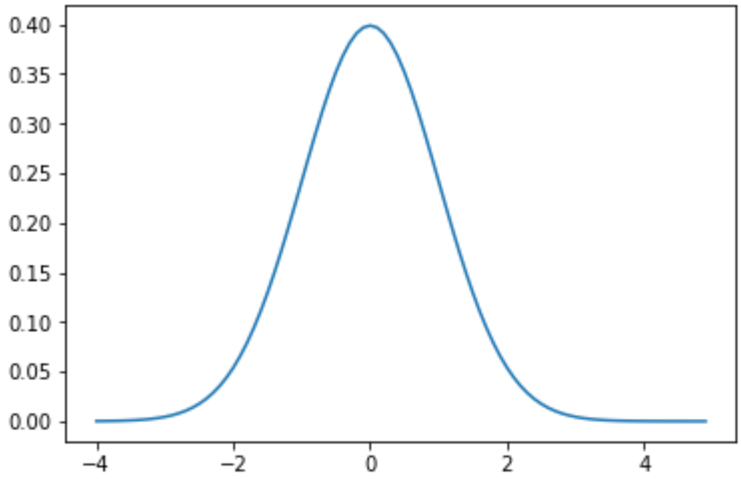
2.5 - Comments on exercise 2

- The function `plot` only connects the given points `(x[i],y[i])`.
- If the **step** is too big between the values of `x`, the curve will be not very representative, as it is the case above where the **step** is equal to **1**.
- The more values there are in the tables, the more precise the plot will be.

2.6 - Exercise 2 (continued)

- Repeat the above exercise, this time taking for `x` an array containing all the multiples of 0.1 between -4 and 4 (inclusive). (Do not display the values of `x` and `y`.)

In [10]:
```
x = np.arange(-4,5,0.1)
y = gauss(x)
fig = plt.plot(x,y)
```



2.7 - Drawing several graphs on the same figure

To draw several graphs on the same figure one must :

- create a figure using the `figure()` function
- execute the command `plot(...)` for each desired graph (without creating a new figure)

```
def function1(x)
  return ...
def function2(x)
  return ...
x = np.arange(...)
y1 = function1(x)
y2 = function2(x)
plt.figure()
plt.plot(x,y1)
plt.plot(x,y2)
```

- or, for curves that are not defined for the same values of `x` :

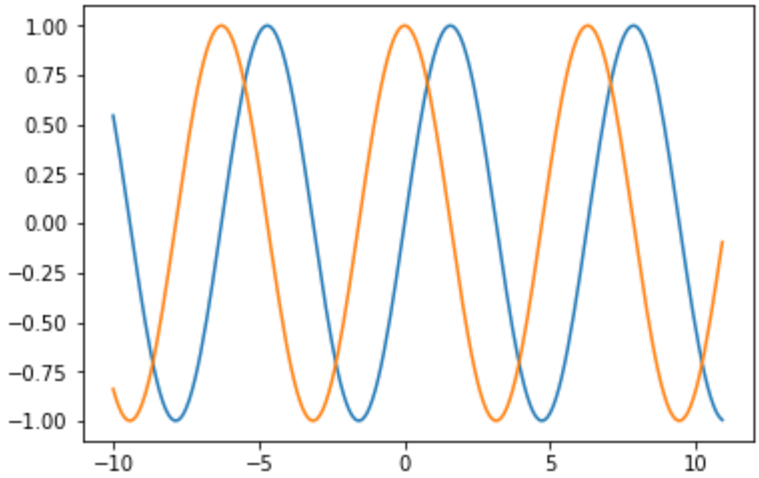
```
x1 = np.arange(...)
y1 = function1(x1)
x2 = np.arange(...)
y2 = function2(x2)
plt.figure()
plt.plot(x1,y1)
plt.plot(x2,y2)
```

2.8 - Exercise 3

- Draw the graphs of the functions `sin(x)` and `cos(x)` for x between -10 and 10 (inclusive) with a **step** of **0.1**

In [11]:
```
x = np.arange(-10,11,0.1)
plt.figure()
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

Out[11]: [<matplotlib.lines.Line2D at 0x7fcb4031cc88>]



2.9 - Comment on exercise 3

The `sin(x)` and `cos(x)` functions in the `numpy` library are defined for values of `x` expressed in **radians**.

- the right angle measures 90 degrees = π/2 radians ~ 1.57 radians
- the flat angle measures 180 degrees = π radians ~ 3.14 radians
- we have sin(π/2) = 1 and cos(π/2)=0

2.10 - Exercise 3 (continued)

Identify the graph of the function `sin(x)` and the graph of the function `cos(x)` in the figure above. **(Answer in a new cell)**

The red graph is the graph of the function cos(x) because cos(0) = 1. The blue graph is consequently the graph of the function sin(x).

3 - Customizing figures

3.1 - Defining the dimensions of the figure

- The optional argument `figsize=[width, height]` allows you to define the dimensions of the figure in inches (1 inch = 2.54 centimeters); for example:

```
plt.figure(figsize=[4, 4]) # 4 inches wide and 4 inches high
```

- To give the dimensions in centimeters one can write

```
plt.figure(figsize=[10/2.54, 10/2.54]) # 10 centimeters wide and 10
centimeters high
```

- An alternative solution is to give a name to the figure and then indicate its dimensions like this:

```
fig = plt.figure()
fig.set_size_inches(10/2.54, 10/2.54) # 10 centimeters wide and 10 c
entimeters high
```

3.2 - Setting display limits

- We can limit the displayed area with the commands :

```
plt.axis([xmin, xmax, ymin, ymax]) # BE CAREFUL with the order of th
e parameters
```

- Or separately:

```
plt.xlim([xmin, xmax])
plt.ylim([ymin, ymax])
```

3.3 - Adding a title

- It is recommended to add a title to figures with the command :

```
plt.title("figure title")
```

3.4 - Adding names to the axes

- We name the x- and y-axes with the following commands:

```
plt.xlabel("name of the x-axis")
plt.ylabel("name of the y-axis")
```

- We can simply write :

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

- But we can also put some arbitrary text that describes the nature of the data on the x and y-axes:

```
plt.xlabel("time")
plt.ylabel("pulse")
```

3.5 - Adding a legend

- When several curves are presented in the same graph, it is necessary to be able to identify them. This is done (for example) by adding a legend to the figure.
- We add a label to each graph thanks to the `label` option of the `plot` function:

```
plt.plot(x1 ,y1 ,label='plot name 1')
plt.plot(x2 ,y2 ,label='plot name 2')
```

- Then we display the legend with the `legend` function:

```
plt.legend()
```

3.6 - Adding free text

- We can add text to a figure with the command :

```
plt.text(a, b, "some text") # the text will be displayed to the right of the point of coordinates (a,b)
```

3.7 - Changing the appearance of a line

- The `linestyle` option of the `plot` function allows you to change the appearance of a line:

```
plt.plot(x1 ,y1 ,linestyle='solid') # default style
plt.plot(x2 ,y2 ,linestyle='dotted') # dotted
plt.plot(x2 ,y2 ,linestyle='dashed') # dashed
...
```

- List of accepted line styles: https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html (https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html)
- The `linewidth` option of the `plot` function allows you to change the thickness of a line:

```
plt.plot(x1 ,y1 ,linewidth=1.5) # default thickness
plt.plot(x2 ,y2 ,linestyle=2) # thicker
plt.plot(x2 ,y2 ,linestyle=1) # thinner
...
```

3.8 - Changing colors

- The option `color='color'` is accepted by all graphics commands

```
plt.title("figure title", color='green')
plt.plot(x, y, color='red')
plt.text(0, 0, "some text", color='blue')
```

- List of colors: https://matplotlib.org/stable/gallery/color/named_colors.html (https://matplotlib.org/stable/gallery/color/named_colors.html)

3.9 - Changing the position and size of text

- The `ha` (horizontal alignment) and `va` (vertical alignment) options change the position of a text relative to the display point.
- The `size` option changes the size of a text.
- These options are accepted by the `title` and `text` functions.

```
plt.text(a, b, "some text",ha='left') # the text will be left aligned on the point (a,b)
plt.text(a, b, "some text",ha='right') # the text will be right aligned on the point (a,b)
plt.text(a, b, "some text",ha='left') # the text will be centered horizontally on the point (a,b)

plt.text(a, b, "some text",va='center') # the text will be centered vertically on the point (a,b)

plt.text(a, b, "some text",size=10) # 10px is the default text size
plt.text(a, b, "some text",size=20) # the text size will be 20px

plt.title("figure title",size=14) # the size of the title will be 14 px
```

- text options: [https://matplotlib.org/stable/tutorials/text/text_props.html (https://matplotlib.org/stable/tutorials/text/text_props.html)](https://matplotlib.org/stable/tutorials/text/text_props.html)

3.10 - Saving a figure

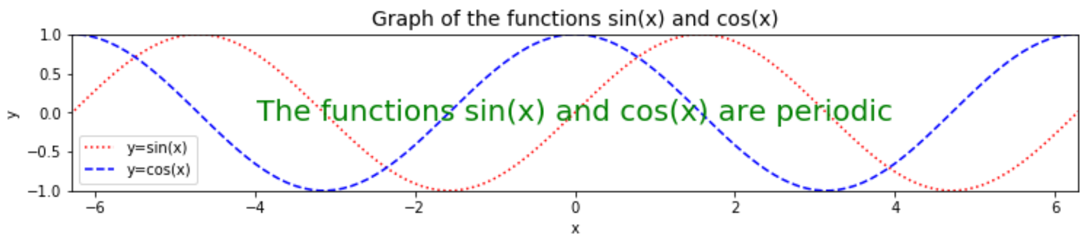
- The function `savefig("filename")` saves a figure as an image file.
- The file format is determined by the extension given to the file name (*.png* / *.tif* / *.jpeg* / etc.).
- The option `dpi=NNN` allows you to change the resolution of the saved image

```
plt.savefig("my_image.png") ### the default resolution is 72 dots per inch (72dpi)
plt.savefig("my_image.png",dpi=300) ### the resolution of the image will be 300 dpi
```

3.11 - Exercise 4

- Create a new figure with dimensions : width = 10*π cm / height = 5 cm (the number π~3.14 is obtained by entering `np.pi`)
- Draw the graphs of the functions:
  - `sin(x)` in RED DOTS with the label 'y=sin(x)'
  - `cos(x)` in BLUE DASHES with the label 'y=cos(x)'
- Do not forget to display the legend.
- Restrict the display area to x between -2*π and +2*π and y between -1 and 1
- Name the axes 'x' and 'y'
- Add the title 'Graph of the functions sin(x) and cos(x)' at the size of 14px
- Add the text 'The functions sin(x) and cos(x) are periodic', centered on the point of coordinates (0, 0), in GREEN and at the size of 20px
- Save the figure under the name 'sincos.png' with a resolution of 150dpi

In [12]:
```
plt.figure(figsize=[10 * np.pi / 2.54, 5 / 2.54])
plt.plot(x, np.sin(x), color = 'red', linestyle = 'dotted', labe
l = 'y=sin(x)')
plt.plot(x, np.cos(x), color = 'blue', linestyle = 'dashed', lab
el = 'y=cos(x)')
plt.legend()
plt.axis([-2 * np.pi, 2 * np.pi, -1, 1])
plt.xlabel("x")
plt.ylabel("y")
plt.title('Graph of the functions sin(x) and cos(x)', size = 14)
plt.text(0, 0, 'The functions sin(x) and cos(x) are periodic', v
a = 'center', ha = 'center', color = 'green', size = 20)
plt.savefig('sincos.png', dpi = 150)
```



4 - Histogram

4.1 - Generating random data

- For the purpose of this tutorial, we use Python to generate arrays of random numbers.
- There are several libraries for this. We will use the functions of the `numpy` library.
- The function `random.uniform` is used to generate an array of uniformly distributed numbers.
- The function `random.standard_normal` allows to generate an array of numbers distributed according to the *standard normal law*.
- The following code returns a number randomly drawn in the range [0,1].

```
x = np.random.uniform (0, 1)
```

- The following code generates an array of 10 numbers distributed uniformly in the range [0,1] and an array of 10 numbers distributed according to the *standard normal law*.

```
N_points = 10
array1 = np.random.uniform (0, 1 , N_points)
array2 = np.random.standard_normal (N_points)
```

- The documentation of the sub-library `numpy.random` is online: [https://numpy.org/doc/1.16/reference/routines.random.html (https://numpy.org/doc/1.16/reference/routines.random.html)](https://numpy.org/doc/1.16/reference/routines.random.html)

4.2 - Exercise 5

- Draw at random a number in the range [-1,1] and display it.
- Generate an array named `ran` of 20 numbers uniformly distributed in the range [-2,2] and display it.

In [14]:
```
x = np.random.uniform (-1, 1)
print(x)
ran = np.random.uniform (-2, 2 , 20)
print(ran)
```

```
0.5756701644793678
[ 0.29574152 -1.18756974  1.0721777   0.39394603 -1.64431938 -0.
55314942
 -1.89838518 -0.59675866 -1.25484244 -1.17471931  1.56366446 -1.
01860625
 -0.89581956  1.44264144 -0.90417362 -1.92562744 -0.28750379 -0.
20685785
 -0.3309035   0.38793217]
```

4.3 Plotting a Histogram

- A histogram is a graph which allows you to visualize the distribution of a table of numbers over a given range.
- To plot such a graph we use the `hist` command of the `matplotlib.pyplot` library.
- The following command plots a histogram, with 10 bands of equal width along the x-axis, that represents the distribution of the numbers in the array `t` over the range `[a,b]` ; the values along the y-axis correspond **to the number of elements** of the array `t` in each band:

```
plt.hist(t, bins=10, range=[a,b])
```

- The following command plots a histogram, with 10 bands of equal width along the x-axis, that represents the density of the numbers in the array `t` over the range `[a,b]` ; the values along the y-axis correspond **to the density** of the array `t` in each band; this representation is such that **the total area of the bands is equal to 1** :

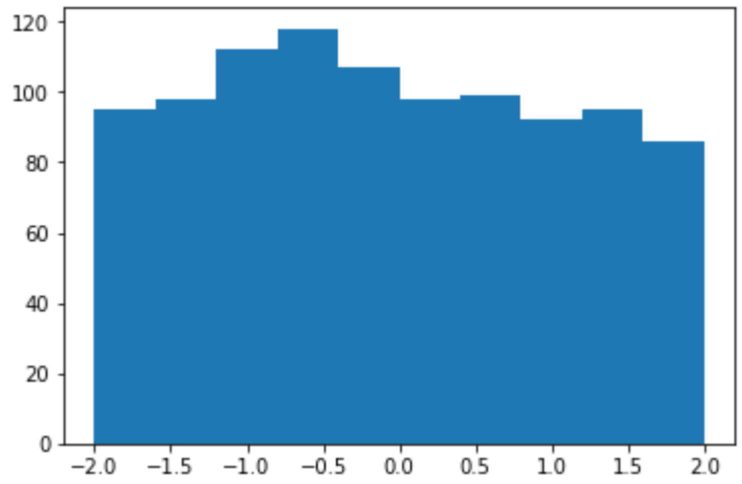
```
plt.hist(t, bins=10, range=[a,b], density=True) # True is written wi
th a capital T
plt.hist(t, bins=10, range=[a,b], density=1) # equivalent to the lin
e above
```

- The complete documentation of the `hist` command is online: [https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html)](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html)

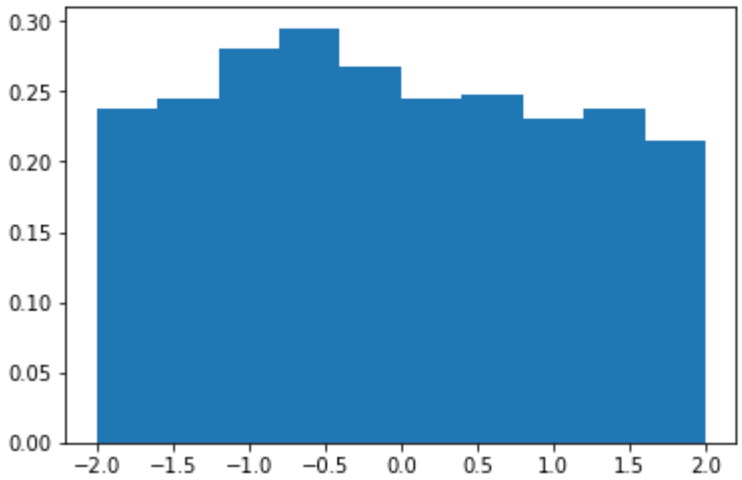
4.4 Exercise 6

- Generate a table of 1000 numbers distributed uniformly in the range [-2,2].
- Plot a 10-band histogram representing the **distribution** of these numbers over the range [-2,2].
- Save this figure under the name 'histogram_1.png'.
- Plot a second 10-band histogram representing the **density** of these numbers over the range [-2,2].
- Save this figure as 'histogram_2.png

In [16]:
```
plt.figure
t = np.random.uniform (-2, 2 , 1000)
plt.hist(t, bins=10, range=[-2,2])
plt.savefig("histogram_1.png")
```



In [18]:
```
plt.figure
plt.hist(t, bins=10, range=[-2,2], density=1)
plt.savefig("histogram_2.png")
```



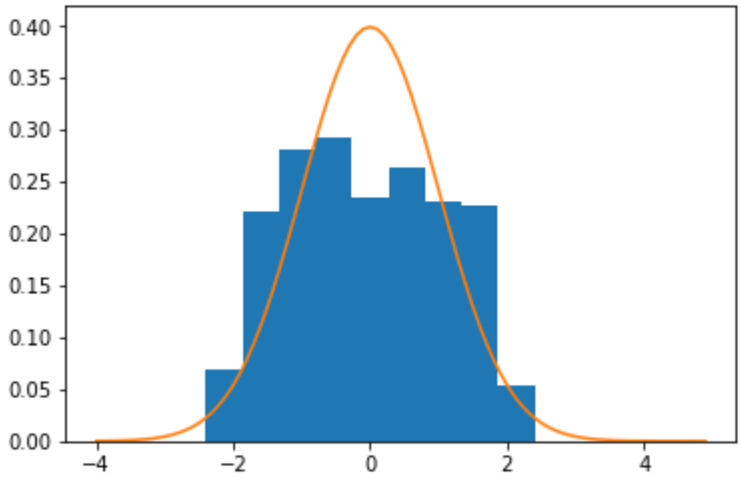
4.5 - Comment on exercise 6

- We notice that even with 1000 randomly drawn numbers, the distribution does not seem perfectly uniform.
- This impression disappears by increasing the randomly drawn numbers to 10000, but it seems more interesting to us to have *imperfect* data.

4.6 - Exercise 7

- Generate a table of 1000 numbers distributed according to the *standard normal distribution*.
- Draw a 15-band histogram representing the **density** of these numbers over the range [-4,4].
- On the same figure, draw the graph of the *Gaussian* function on the range [-4,4] (see §2.4 - Exercise 2 : graph of the Gauss function)
- Save this figure under the name 'histogram_3.png'

In [20]:
```
t2 = np.random.standard_normal (1000)
plt.figure
plt.hist(t, bins=15, range=[-4,4], density=1)
x = np.arange(-4,5,0.1)
y = gauss(x)
fig = plt.plot(x,y)
plt.savefig("histogram_3.png")
```



4.7 - Comment on exercise 7

- The two representations (the histogram of the random numbers and the graph of the Gaussian function) must correspond approximately.

5 - Scatter plots

5.1 Representing a scatterplot

- A scatterplot is given by **two arrays of numbers of equal length**, `x` and `y`, the first array being the x-coordinate of points `M[i]=(x[i],y[i])` and the second array being the y-coordinate. The data are thus of the same nature as for the plot of the graph of a function. The difference lies in the way the data are displayed in the graph.
- In order to draw the graph of a function, we join the points `(x[i],y[i]` with lines, and we wish to observe a regular line.
- In order to draw scatterplot, we only display the points, which will be represented by a cross, a circle, or another symbol, but we do not want to draw the lines joining those points.
- To achieve this, we use two options of the `plot` command: the `linestyle='none'` option suppresses the display of lines, the `marker='x'` option displays a cross for each point.
- This gives, for two arrays `x` and `y`, with different marker types:

```
plt.plot(x, y, linestyle='none', marker='x') # display a cross for e
ach point
plt.plot(x, y, linestyle='none', marker='+') # display a cross for e
ach point
plt.plot(x, y, linestyle='none', marker='.') # displays a dot for ea
ch point
plt.plot(x, y, linestyle='none', marker='o') # show a bigger dot for
each point
```

- You can also use the shorter version below:

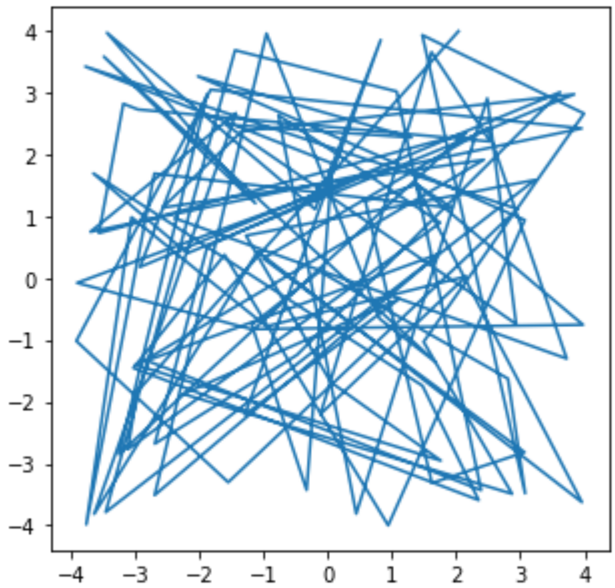
```
plt.plot(x, y, 'x') # display a cross for each point
plt.plot(x, y, '+') # show a cross for each point
plt.plot(x, y, '.') # display a dot for each point
plt.plot(x, y, 'o') # show a big dot for each point
```

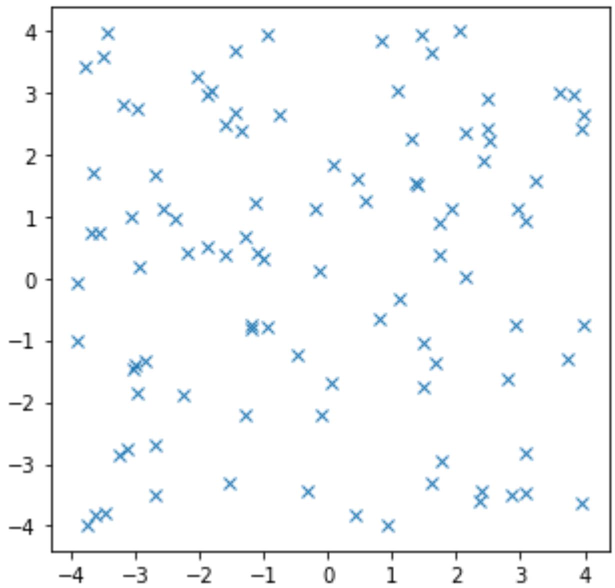
- List of accepted markers: [https://matplotlib.org/stable/api/markers_api.html (https://matplotlib.org/stable/api/markers_api.html)](https://matplotlib.org/stable/api/markers_api.html)
- Other options of the `plot` command are also accepted: `color`, `label`, etc.

5.2 - Exercise 8

- Generate a random array named `x` of 100 numbers distributed uniformly in the range [-4,4].
- Generate a second random array named `y` of 100 numbers distributed uniformly in the range [-4,4].
- Create a figure of dimension 5 inches in width and 5 inches in height and draw the curve joining the points of coordinates `(x[i],y[i])`.
- Create a second figure of the same dimensions and plot the scatterplot of the points of coordinates `(x[i],y[i])`, marked with an `x`.
- Save the second figure under the name 'scatterplot_1.png'.

In [25]:
```
x = np.random.uniform (-4, 4 , 100)
y = np.random.uniform (-4, 4 , 100)
plt.figure(figsize=[5, 5])
plt.plot(x, y)
plt.figure(figsize=[5, 5])
plt.plot(x, y, linestyle='none', marker='x')
plt.savefig('scatterplot_1.png')
```





5.3 - Comment on exercise 8

- The first representation is obviously not very interesting for random points.

5.4 - Interpretating scatter plots

- The figure above represents randomly drawn points. This property can be seen visually, because the points fill more or less the square.
- For data from experiments, the points rather respect a certain regularity.
- We will artificially create an array of points distributed around a curve and display the scatterplot and the curve.

Exercise 9

- The following code produces two arrays of numbers `x` and `y` . The points with coordinates `(x[i],y[i])` lie near the graph of the function `f(x) = exp(-x)`

In [28]:
```
def f(x):
    return np.exp(-x)
a = np.random.uniform(0,4,100)
b = []
for x in a:
    b.append( f(x)*np.random.uniform(0.5, 1.5) )
```

- Draw, on the same figure, the scatterplot of points with coordinates `(a[i],b[i])` , and the graph of the function `f(x)` on the range [0,4]
- Make sure that the scatterplot is close to the curve.
- Save this figure under the name `scatterplot_2.png`

In [30]:
```
plt.figure()
plt.plot(a, b)
plt.plot(a, b, linestyle='none', marker='x')
plt.savefig('scatterplot_2.png')
```