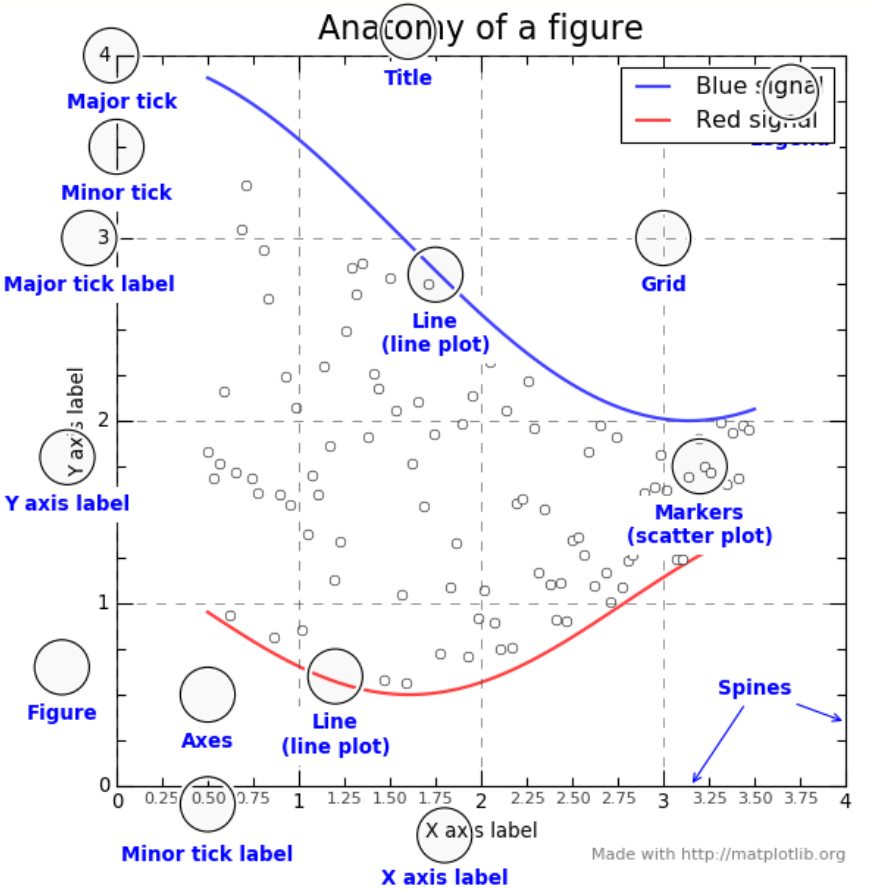
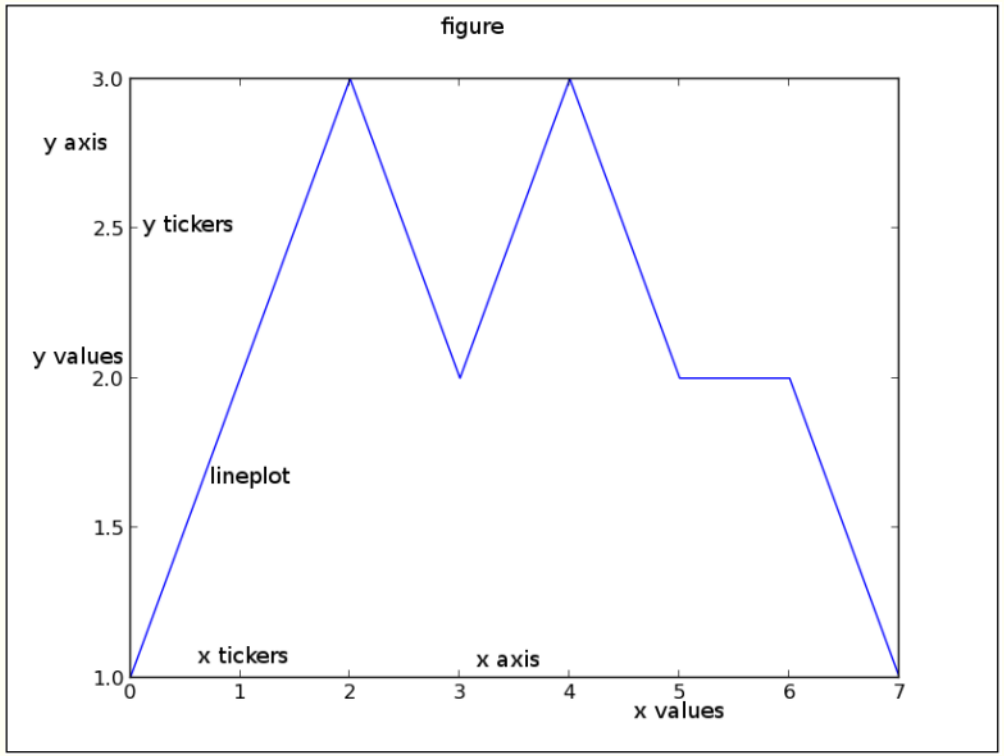
БІБЛІОТЕКА MATPLOTLIB ТА ВІЗУАЛІЗАЦІЯ ДАНИХ

Питання 10.3.

Анатомія графіків бібліотеки matplotlib



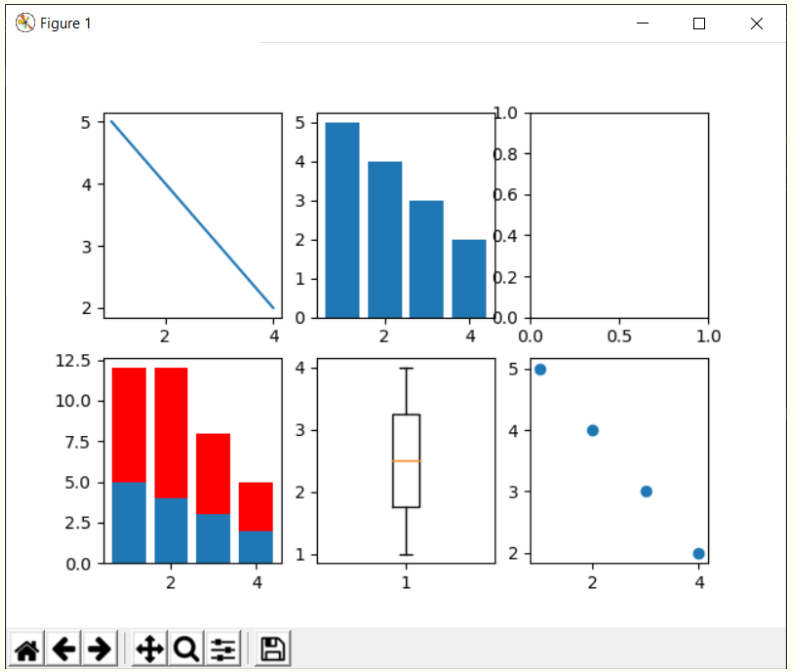
- Рисунок (Figure) – основний контейнер у matplotlib.
  - Вісі (Axes) представляють координатну систему.
  - Вона містить більшість елементів рисунка, наприклад, Axis, Line2D, Text.
  - Допускається багато Axes-об’єктів на рисунку (Figure-об’єкті).
  - Axis – контейнер для насічок (ticks) та написів (labels) для осей x та y на графіку (plot).



Демонстраційний код

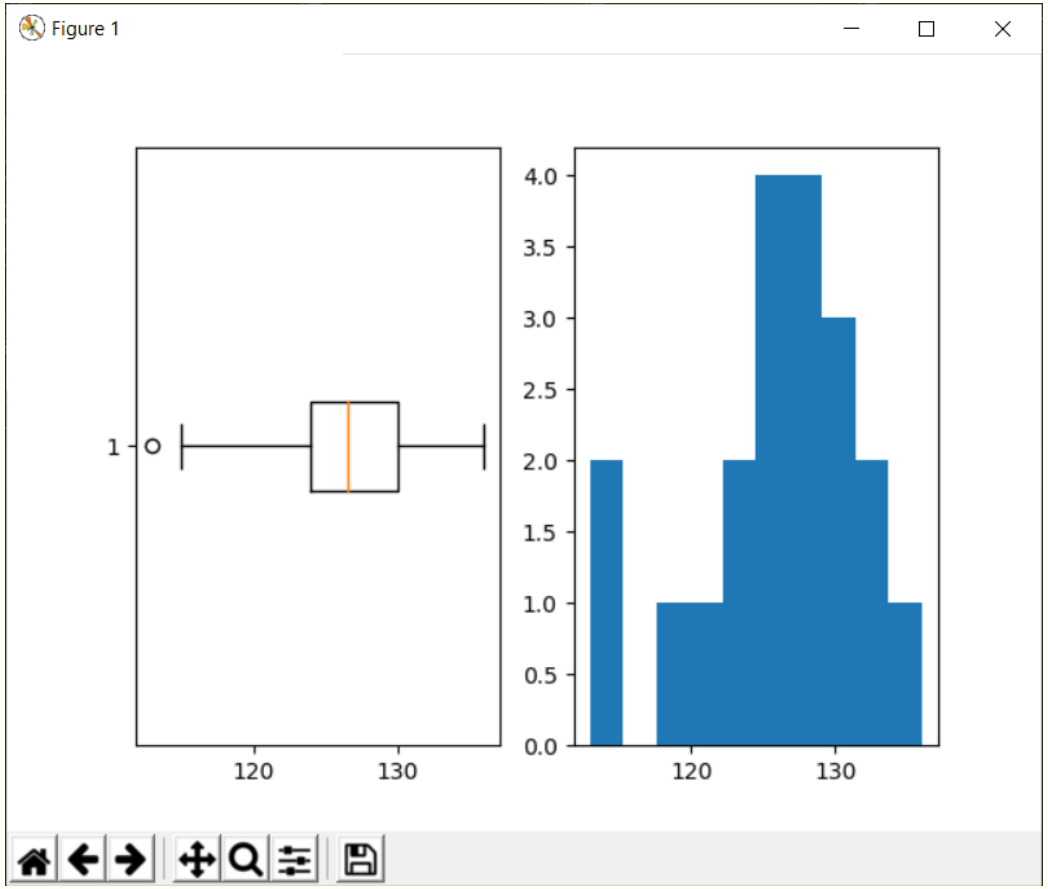
```
from matplotlib.pyplot import *
# some simple data
x = [1,2,3,4]
y = [5,4,3,2]

# create new figure
figure()
# divide subplots into 2 x 3 grid
# and select #1
subplot(231)
plot(x, y)
# select #2
subplot(232)
bar(x, y)
# horizontal bar-charts
subplot(233)
# create stacked bar charts
subplot(234)
bar(x, y)
# we need more data for stacked bar charts
y1 = [7,8,5,3]
bar(x, y1, bottom=y, color = 'r')
# box plot
subplot(235)
boxplot(x)
# scatter plot
subplot(236)
scatter(x,y)
show()
barh(x, y)
```

- За допомогою figure() створюємо новий рисунок.
  - Якщо передати рядковий аргумент, він стане backend title вікна.
  - Якщо викликати figure() з тим же параметром (це може бути й число), відповідний рисунок стане активним з подальшим рисуванням на ньому.
- Розіб’ємо рисунок сіткою 2x3 за допомогою виклику subplot(231).
  - Третій параметр позначає номер графіку по порядку.



Побудова гістограми

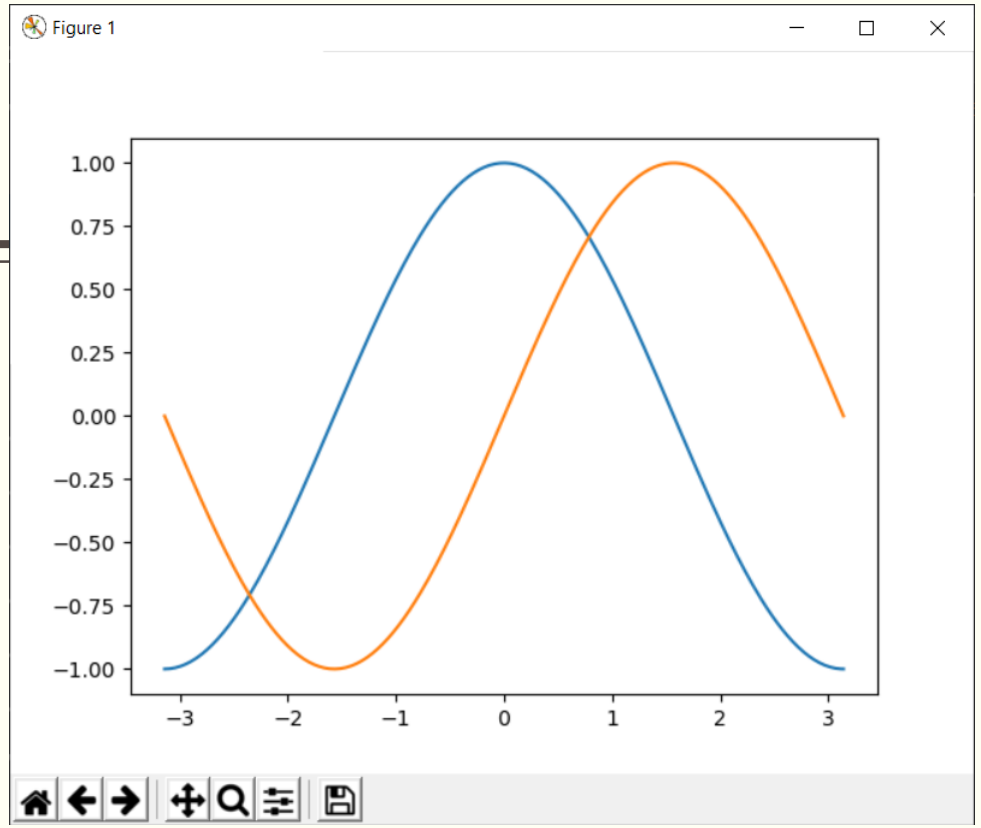
```
from pylab import *
dataset = [113, 115, 119, 121, 124,
124, 125, 126, 126, 126,
127, 127, 128, 129, 130,
130, 131, 132, 133, 136]
subplot(121)
boxplot(dataset, vert=False)
subplot(122)
hist(dataset)
show()
```



- Одні дані можна представити по-різному.
- На графіку можна представити 5 статистичних величин:
  - Мінімальне значення в наборі даних (dataset)
  - Другий квартиль: Below this the lower 25% набору даних lies
  - Медіана: медіанне значення датасету (половина чисел менша за нього)
  - Третій квартиль: Above this the upper 25 percent of the given dataset lies
  - Максимальне значення в наборі даних

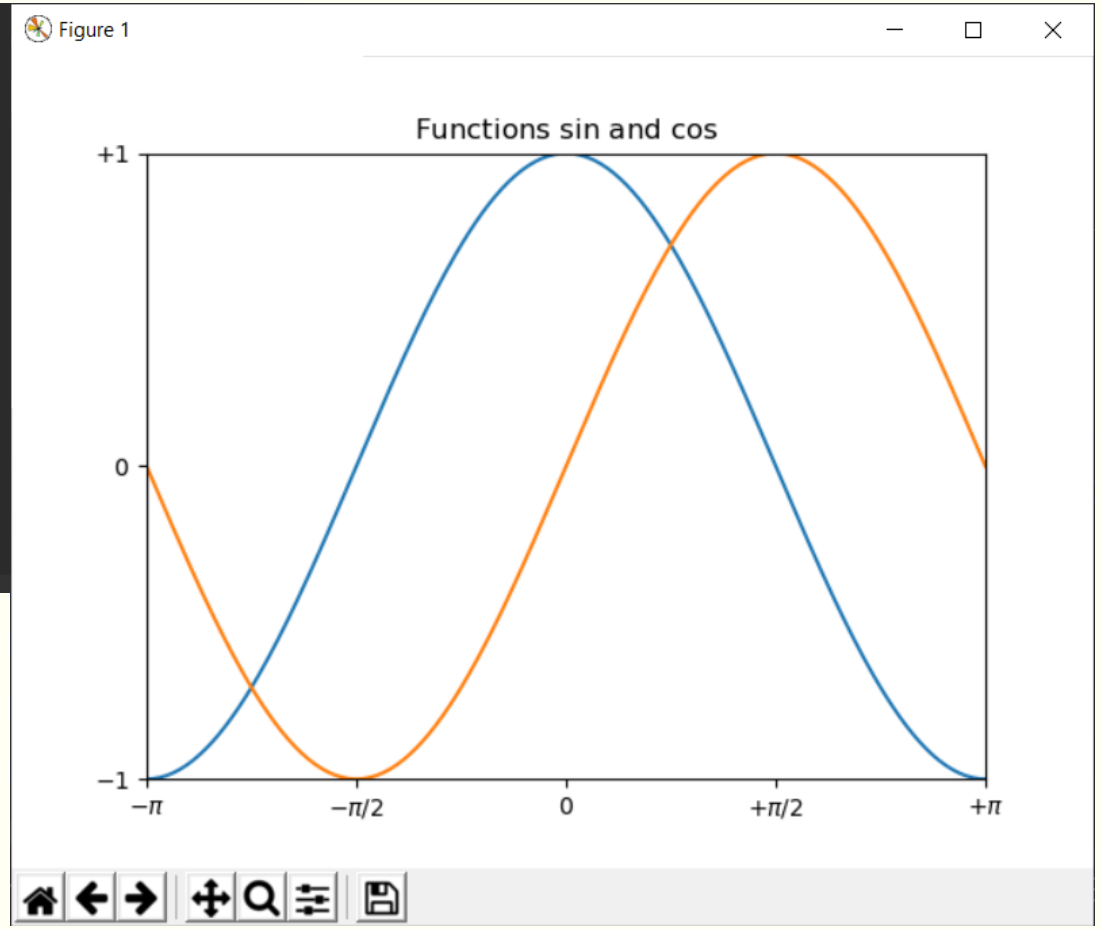
Рисування простих графіків синуса та косинуса

```python
import matplotlib.pyplot as pl
import numpy as np
x = np.linspace(-np.pi, np.pi, 256, endpoint=True)
y = np.cos(x)
y1 = np.sin(x)
pl.plot(x,y)
pl.plot(x, y1)
pl.show()
```



Додаткові налаштування для графіка

```
17  # define plot title
18  title("Functions $\sin$ and $\cos$")
19  # set x limit
20  xlim(-3.0, 3.0)
21  # set y limit
22  ylim(-1.0, 1.0)
23  # format ticks at specific values
24  xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
25  [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
26  yticks([-1, 0, +1],
27  [r'$-1$', r'$0$', r'$+1$'])
28  show()
```



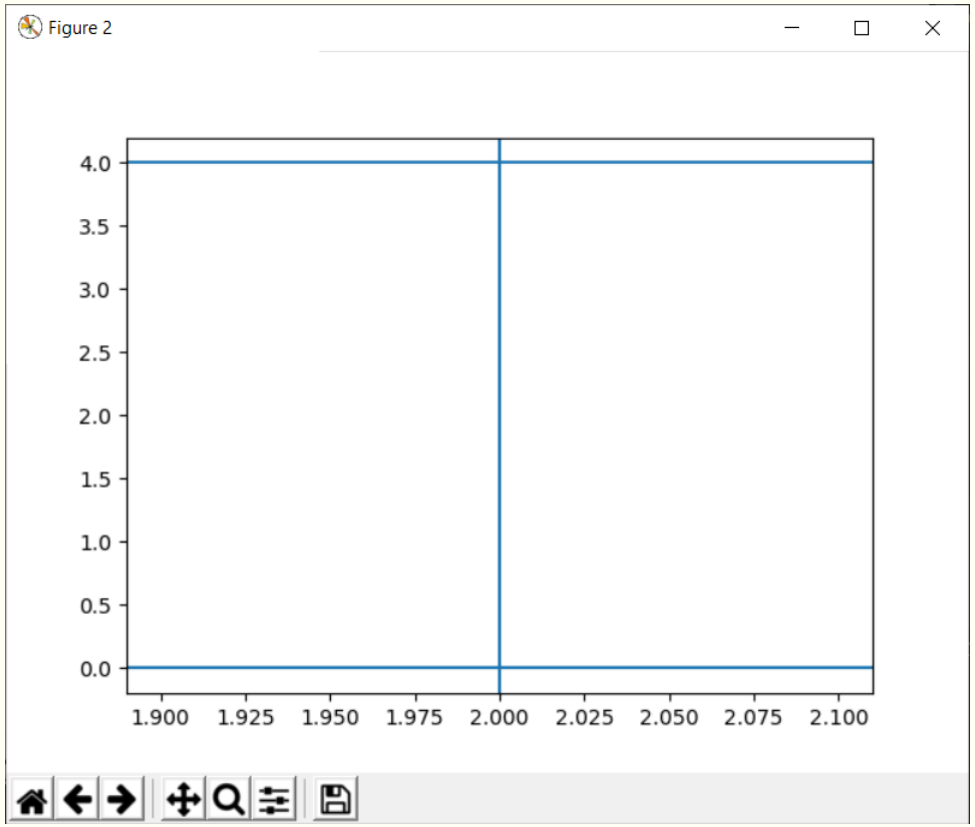
- Використовуються вирази на зразок $\sin$ або $-\pi$, що є синтаксисом LaTex.

Робота з сіткою та осями

- **`axis([-1, 1, -10, 10])`**
  - Визначає розміри осей
  - xmin, xmax, ymin, ymax
  - Можна викликати matplotlib.pyplot.autoscale(), яка підбере оптимальні розміри відповідно до даних.
- Для додавання нових осей на тому ж рисунку можна використати matplotlib.pyplot.axes().
  - Зазвичай до цього виклику додаються деякі властивості;
  - rect з атрибутами left, bottom, width та height у нормалізованих одиницях вимірювання (0, 1)
  - axisbg задає колір фону осей.
  - sharex/sharey приймає значення інших екземплярів осей та ділиться з ними поточними налаштуваннями для x/y.
  - polar – параметр, який визначає, чи потрібно використовувати полярну систему координат.
- Виклик matplotlib.pyplot.grid() перемикає видимість сітки на рисунку з параметрами:
  - which: визначає тип насічок для відрисовки (major, minor або обидва варіанти)
  - axis: визначає набір ліній сітки для відрисовки (x, y або обидві)
- Осі зазвичай управляються за допомогою matplotlib.pyplot.axis().
  - Всередині осі представляються кількома Python-класами:
    - matplotlib.axes.Axes містить більшість методів для операцій з осями.
    - matplotlib.axis.Axis описує окрему вісь за допомогою класів matplotlib.axis.XAxis та matplotlib.axis.YAxis.

```
figure(2)
axhline()
axhline(4)
axvline(2)

show()
```

- Якщо потрібно додати до поточного рисунку лише одну лінію, можна застосувати функції matplotlib.pyplot.axhline() або matplotlib.pyplot.axvline().
  - Вони рисують горизонтальні та вертикальні лінії вздовж осей x та y відповідно.
  - Найбільш важливі спільні параметри: y-координата, xmin та xmax для axhline(); x-координата, ymin та ymax для axvline().
- Подібні функції matplotlib.pyplot.axhspan() та matplotlib.pyplot.axspan() дозволяють додавати horizontal span (rectangle) вздовж осей.
  - axhspan() потребує параметрів ymin та ymax для визначення довжини horizontal span.
  - axvspan() потребує xmin та xmax для визначення довжини vertical span.



Визначення стилів ліній, властивостей та форматованих рядків на графіку

- Стиль ліній на графіку можна змінити різними методами:
  - 1) Найбільш поширений спосіб – передавати параметри у функції на зразок plot():
    - plot(x, y, linewidth=1.5)
  - 2) Оскільки plot() повертає екземпляр matplotlib.lines.Line2D, можна задати сеттери для цього екземпляру:
    - line, = plot(x, y)
    - line.set_linewidth(1.5)
  - 3) Спосіб, знайомий для користувачів MATLAB,—функція setp():
    - lines = plot(x, y)
    - setp(lines, 'linewidth', 1.5) або setp(lines, linewidth=1.5)

Linestyle	Description
'-'	Solid
'--'	Dashed
'-.'	Dash_dot
':'	Dotted
'None', ' ', ''	Draw nothing

Alias	Color
b	Blue
g	Green
r	Red
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

- Можна отримати всі кольори, які підтримує matplotlib, викликавши matplotlib.pyplot.colors()

Аргументи налаштування кольору

- Якщо базових кольорів не вистачає, можна застосувати шістнадцяткове HTML-значення, назву кольору або нормалізований RGB-кортеж (від 0 до 1):
  - `color = '#eeefff'`
  - `color = 'red', color = 'chartreuse'.`
  - `color = (0.3, 0.3, 0.4)`
- Аргумент color приймається багатьма функціями, зокрема title() та :
  - title('Title in a custom color', color='#123456’)
- Аргумент axisbg може визначати фоновий колір осі у таких функціях, як matplotlib.pyplot.axes() або matplotlib.pyplot.subplot():
  - subplot(111, axisbg=(0.1843, 0.3098, 0.3098))

Маркери на графіку

Marker	Description
'o'	Circle
'D'	Diamond
'h'	Hexagon1
'H'	Hexagon2
'_'	Horizontal line
'', 'None', ' ', None	Nothing
'8'	Octagon
'p'	Pentagon
','	Pixel
'+'	Plus
'.'	Point
's'	Square
'*'	Star

Marker	Description
'd'	Thin_diamond
'v'	Triangle_down
'<'	Triangle_left
'>'	Triangle_right
'^'	Triangle_up
'|'	Vertical line
'x'	X

Property	Value type	Description
alpha	float	Sets the alpha value used for blending; not supported on all backends.
color or c	Any matplotlib color	Sets the color of the line.
dashes	Sequence of on/off ink in points	Sets the dash sequence, the sequence of dashes with on/off ink in points. If seq is empty or if seq = (None, None), linestyle will be set to solid.
label	Any string	Sets the label to s for auto legend.
linestyle or ls	['-' | '–' | '-.' | ':' | 'steps' | ...]	Sets the linestyle of the line (also accepts drawstyles).
linewidth or lw	float value in points	Sets the line width in points.
marker	[7 | 4 | 5 | 6 | 'o' | 'D' | 'h' | 'H' | '_' | '' | 'None' | ' ' | None | '8' | 'p' | ',' | '+' | '.' | 's' | '*' | 'd' | 3 | 0 | 1 | 2 | '1' | '3' | '4' | '2' | 'v' | '<' | '>' | '^' | '|' | 'x' | '$...$' | tuple | Nx2 array]	Sets the line marker.

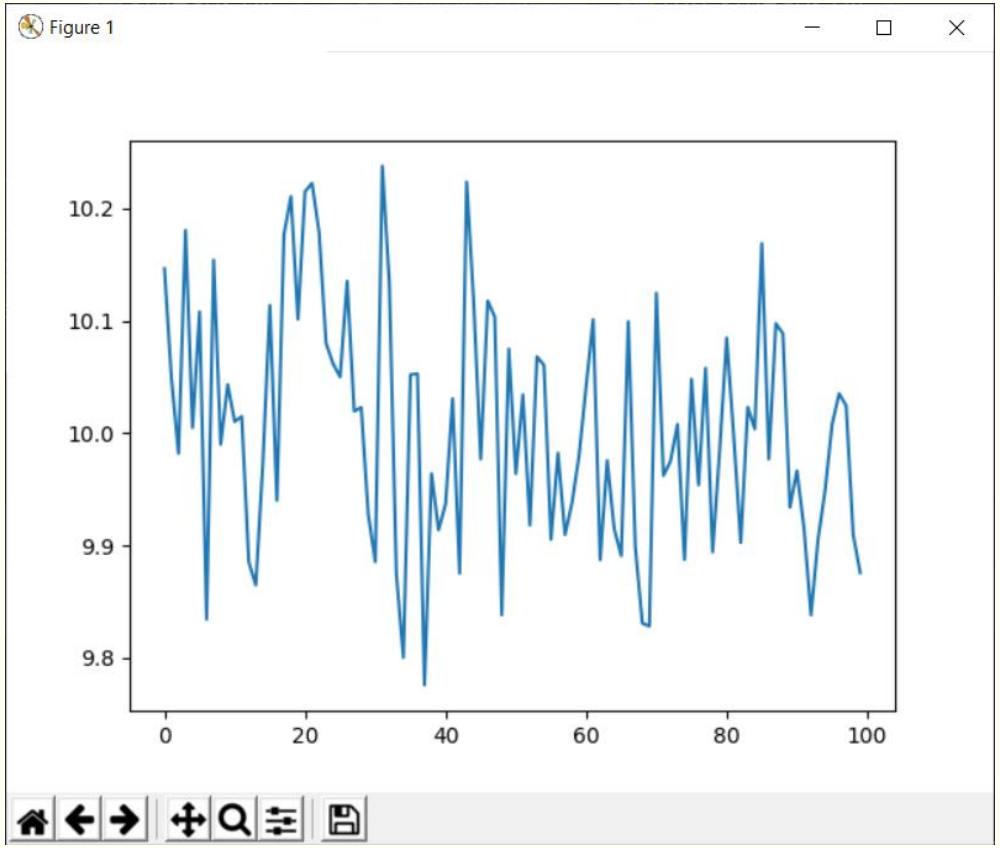
Property	Value type	Description
ydata	np.array	Set the data np.array for y.
Zorder	Any number	Set the z axis order for the artist. Artists with lower Zorder values are drawn first. If x and y are axes going horizontal to the right and vertical to the top of the screen, the z axis is the one extending toward the viewer. So 0 value would be at the screen, 1, one layer above, and so on.
markeredgecolor or mec	Any matplotlib color	Sets the marker edge color.
markeredgewidth or mew	float value in points	Sets the marker edge width in points.
markerfacecolor or mfc	Any matplotlib color	Set the marker face color.
markersize or ms	float	Set the marker size in points.
solid_capstyle	['butt' | 'round' | 'projecting']	Set the cap style for solid line styles.
solid_joinstyle	['miter' | 'round' | 'bevel']	Set the join style for solid line styles.
visible	[True | False]	Set the artist's visibility.
xdata	np.array	Set the data np.array for x.

Встановлення насічок, написів та сіток

- Насічки є частиною рисунку та складаються з локаторів (tick locators, розташування насічок) та форматтерів (tick formatters, визначають появу насічок).
  - Мінорні насічки (Minor ticks) за умовчанням невидимі.
  - Мажорні та мінорні насічки можна розташовувати та форматувати незалежно одні від одних.
- Для управління поведінкою локаторів застосовується функція matplotlib.pyplot.locator_params().
  - Хоч розташування насічок зазвичай визначається автоматично, кількістю насічок можна керувати.

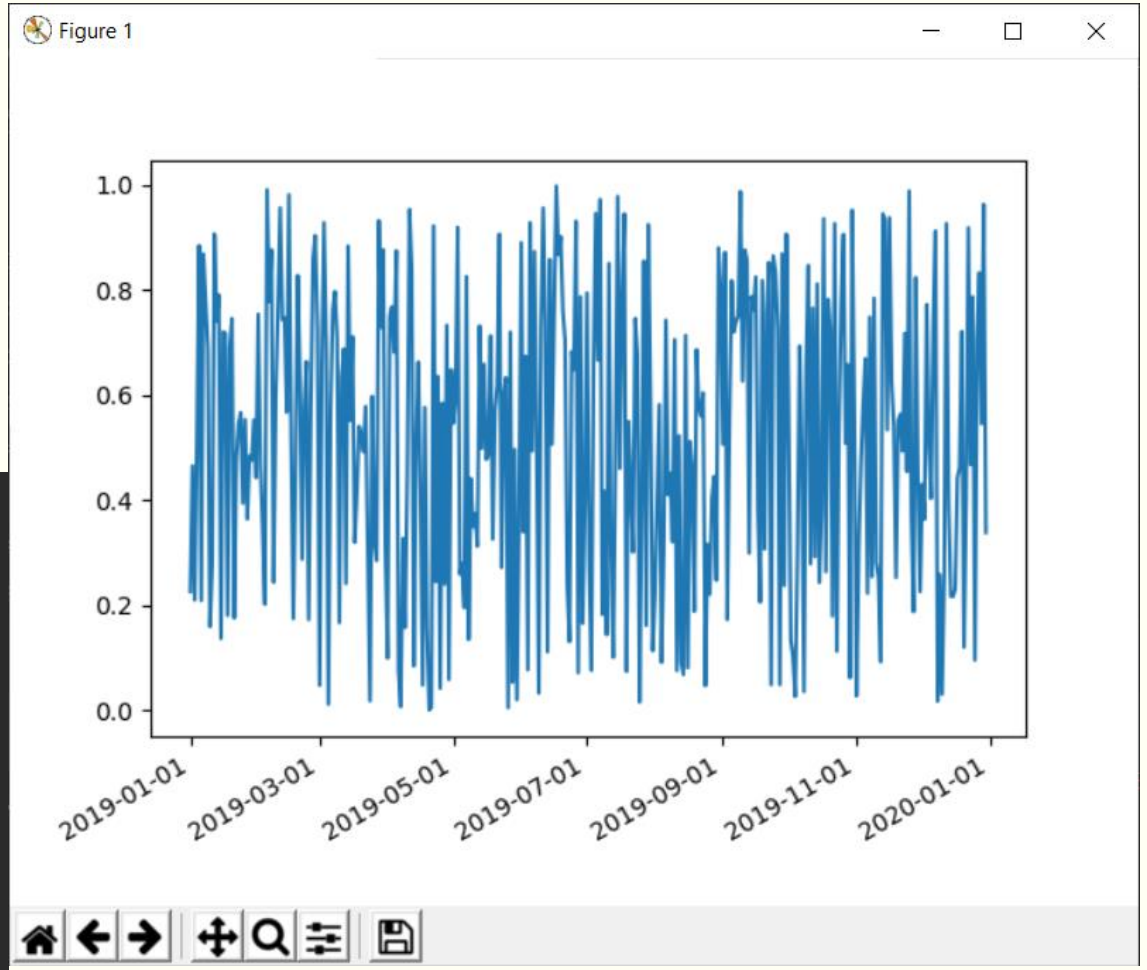
```
1  from pylab import *
2  # get current axis
3  ax = gca()
4  # set view to tight, and maximum number of tick intervals to 10
5  ax.locator_params(tight=True, nbins = 10)
6  # generate 100 normal distribution values
7  ax.plot(np.random.normal(10, .1, 100))
8  show()
```

- Такі ж налаштування можна здійснити за допомогою locator-класів.
  - ax.xaxis.set_major_locator(matplotlib.ticker.MultipleLocator(10))
- Аналогічно можна задати форматтери насічок.
  - Форматтери визначають, як значення (зазвичай числа) будуть відображатись.
  - Наприклад, matplotlib.ticker.FormatStrFormatter просто задає '%2.1f' або '%1.1f cm' to be used as the label for the ticker.



Приклад з використанням дат

```
from pylab import *
import matplotlib as mpl
import datetime
fig = figure()
# get current axis
ax = gca()
# set some daterange
start = datetime.datetime(2019, 1, 1)
stop = datetime.datetime(2019, 12, 31)
delta = datetime.timedelta(days = 1)
# convert dates for matplotlib
dates = mpl.dates.drange(start, stop, delta)
# generate some random values
values = np.random.rand(len(dates))
ax = gca()
# create plot with dates
ax.plot_date(dates, values, linestyle='-', marker='')
# specify formater
date_format = mpl.dates.DateFormatter('%Y-%m-%d')
# apply formater
ax.xaxis.set_major_formatter(date_format)
# autoformat date labels
# rotates labels by 30 degrees by default
# use rotate param to specify different rotation degree
# use bottom param to give more room to date labels
fig.autofmt_xdate()
show()
```



Додавання легенди та annotations

```
from matplotlib.pyplot import *
# generate different normal distributions
x1 = np.random.normal(30, 3, 100)
x2 = np.random.normal(20, 2, 100)
x3 = np.random.normal(10, 3, 100)
# plot them
plot(x1, label='plot')
plot(x2, label='2nd plot')
plot(x3, label='last plot')
# generate a legend box
legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
    ncol=3, mode="expand", borderaxespad=0.)
# annotate an important value
annotate("Important value", (55,20), xycoords='data'
xytext=(5, 38),
arrowprops=dict(arrowstyle='->'))
show()
```

String	Number value
upper right	1
upper left	2
lower left	3
lower right	4
right	5
center left	6
center right	7
lower center	8
upper center	9
center	10

- Задаємо розташування легенди за допомогою опційного параметру loc.
  - Для приховування легенди label = '_nolegend_'.
  - Для легенди визначили кількість стовпців ncol = 3 та задали розташування lower left.
  - Задано bounding box (bbox_to_anchor) з координатою верхнього лівого кута (0., 1.02), довжиною (width) 1 та шириною (height) 0.102. Координати нормалізовані.
  - Параметр mode може бути 'None' або 'expand' (horizontally filling the axis area).
  - Параметр borderaxespad визначає зміщення (padding) між осями та контуром легенди.

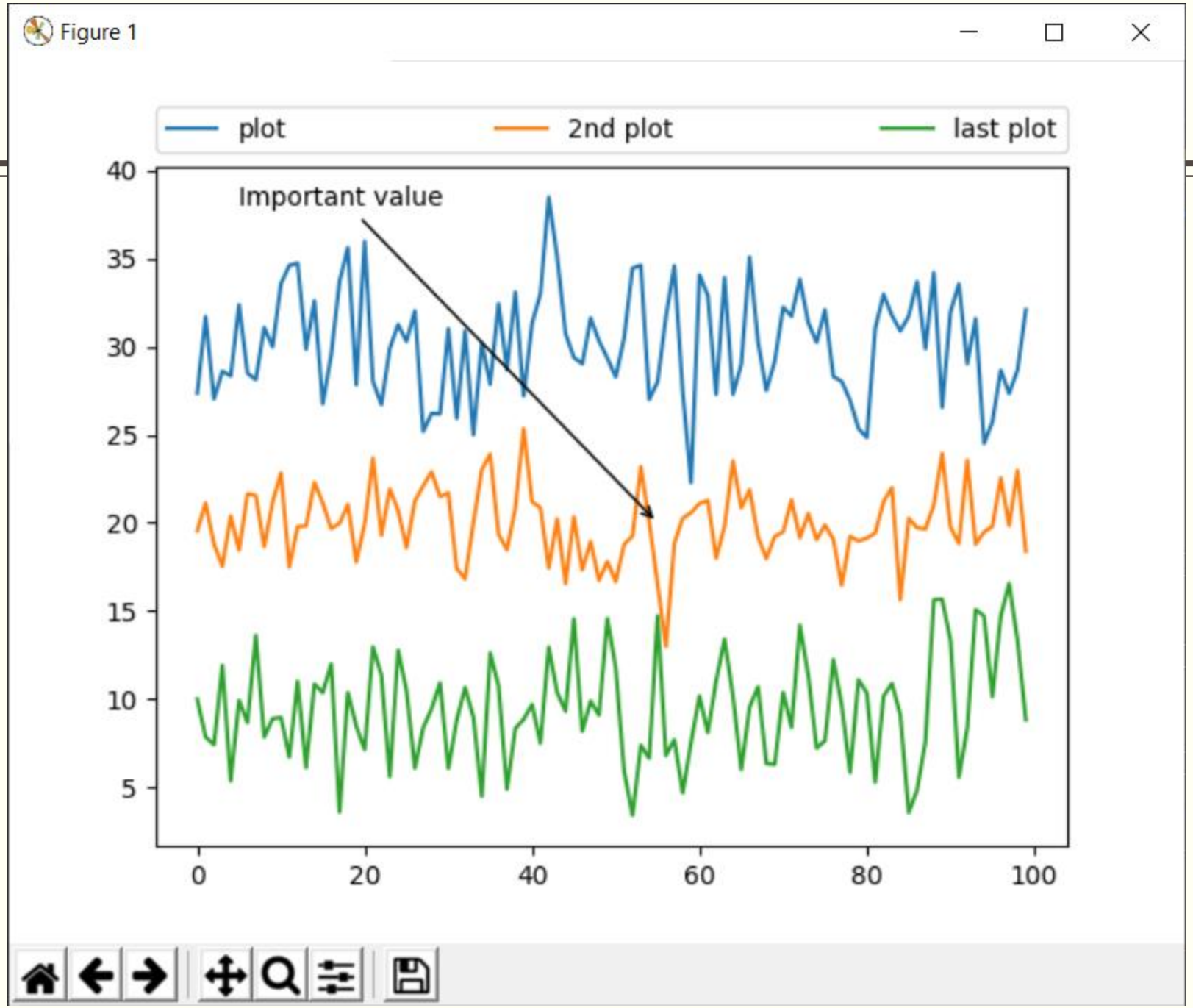
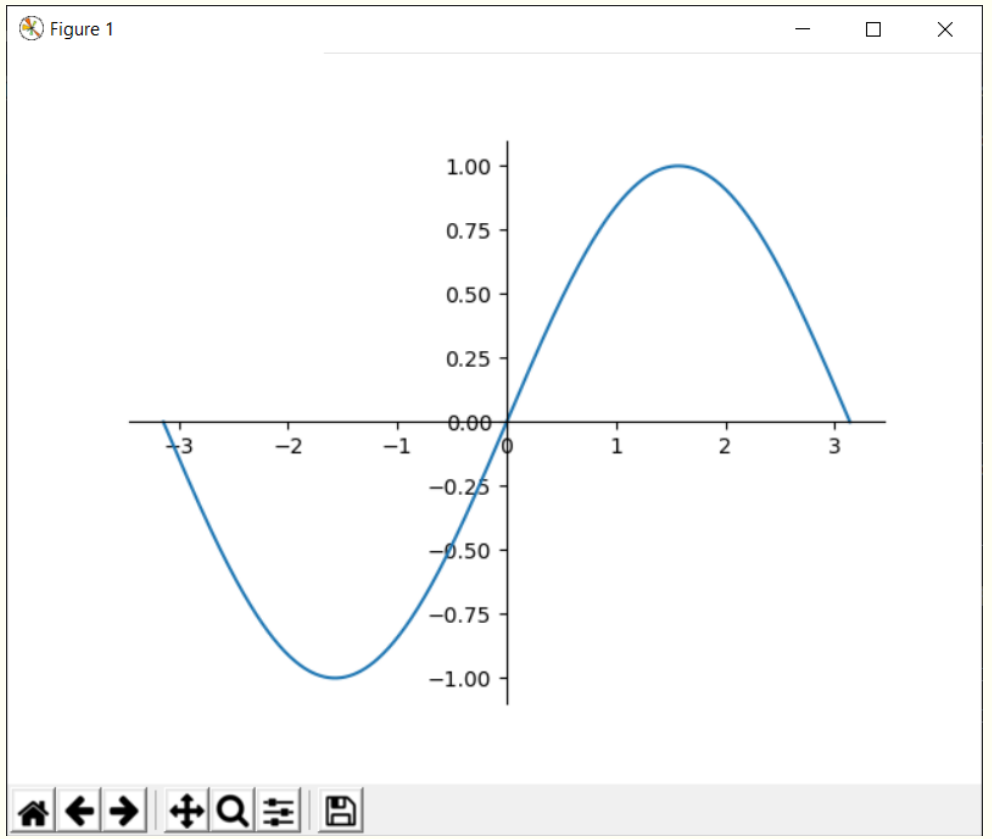


Figure 1
plot
2nd plot
last plot
Important value
40
35
30
25
20
15
10
5
0
20
40
60
80
100

Переміщення spines до центру

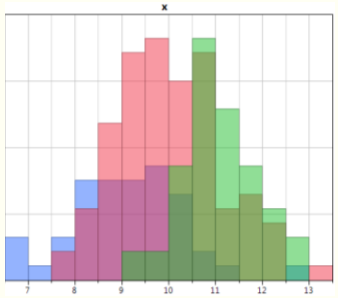
```python
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-np.pi, np.pi, 500, endpoint=True)
y = np.sin(x)
plt.plot(x, y)
ax = plt.gca()
# hide two spines
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
# move bottom and left spine to 0,0
ax.spines['bottom'].set_position(('data',0))
ax.spines['left'].set_position(('data',0))
# move ticks positions
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
plt.show()
```

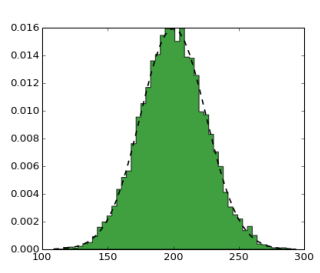
- Необхідно видалити 2 spines, приховуючи їх (присвоїти color значення ‘none’).
  - Після цього переміщаємо два інших в координати (0,0).
  - Координати задаються в data space coordinates.



Створення гістограм

- Виклик matplotlib.pyploy.hist() з набором параметрів. Найкорисніші:
  - ***bins:*** кількість бінів гістограми або послідовність, яка задає біни. За умовчанням – 10.
  - ***range:*** діапазон бінів, не використовується при послідовності бінів. За умовчанням None, Outliers ігноруються.
  - ***normed:*** якщо дорівнює True, значення гістограми нормалізуються та формують розподіл ймовірності. За умовчанням – False.

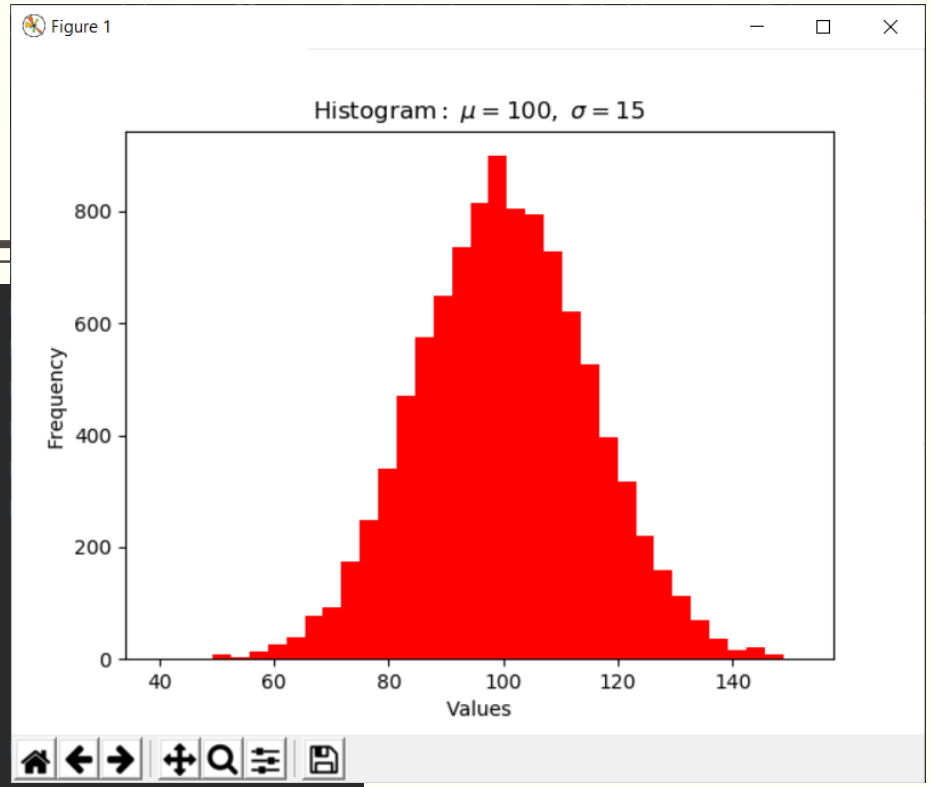




  - ***histtype:*** стандартний вигляд гістограми – прямокутниками. Інші опції:
    - barstacked: стекове представлення, як на рисунку зліва.
    - step: лінійна гістограма, яка left unfilled.
    - stepfilled: лінійна гістограма, яка залита за умовчанням. Елемент за умовчанням – прямокутник.
  - ***align:*** центрує прямокутники між границями бінів. За умовчаннням - mid. Інші значення - left і right.
  - ***color:*** задає колір гістограми. Одне значення або послідовність кольорів.
    - Якщо задано multiple datasets, послідовність кольорів буде застосована в тому ж порядку.
    - Якщо не задано, використовується default line color sequence.
  - ***orientation:*** дозволяє створення горизонтальних гістограм (‘horizontal’). За умовчанням – ‘vertical’.

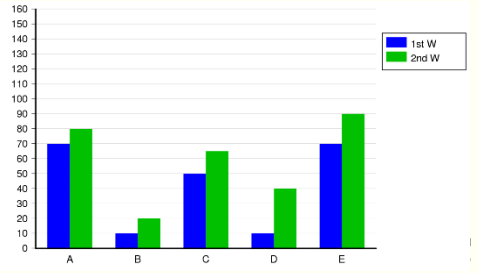
Приклад гістограми

```python
import numpy as np
import matplotlib.pyplot as plt
mu = 100
sigma = 15
x = np.random.normal(mu, sigma, 10000)
ax = plt.gca()
# the histogram of the data
ax.hist(x, bins=35, color='r')
ax.set_xlabel('Values')
ax.set_ylabel('Frequency')
ax.set_title(r'$\mathrm{Histogram:}\ \mu=%d,\ \sigma=%d$' % (mu, sigma))
plt.show()
```



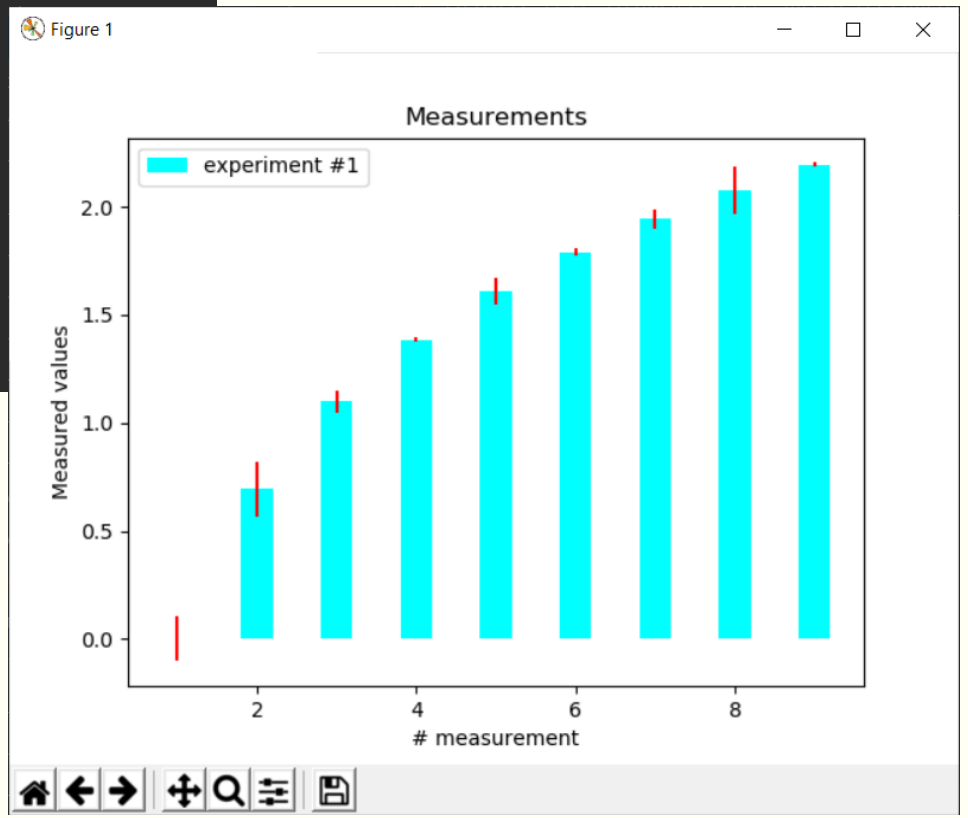
- Починаємо з генерування деяких нормально розподілених даних.
  - Гістограма будується із зданою кількістю бінів—35 та нормалізується (normed = True або 1);
  - колір – червоний (‘r’).
  - Далі встановлюємо написи та заголовок графіку.

Створення ствопчикової діаграми (bar charts)



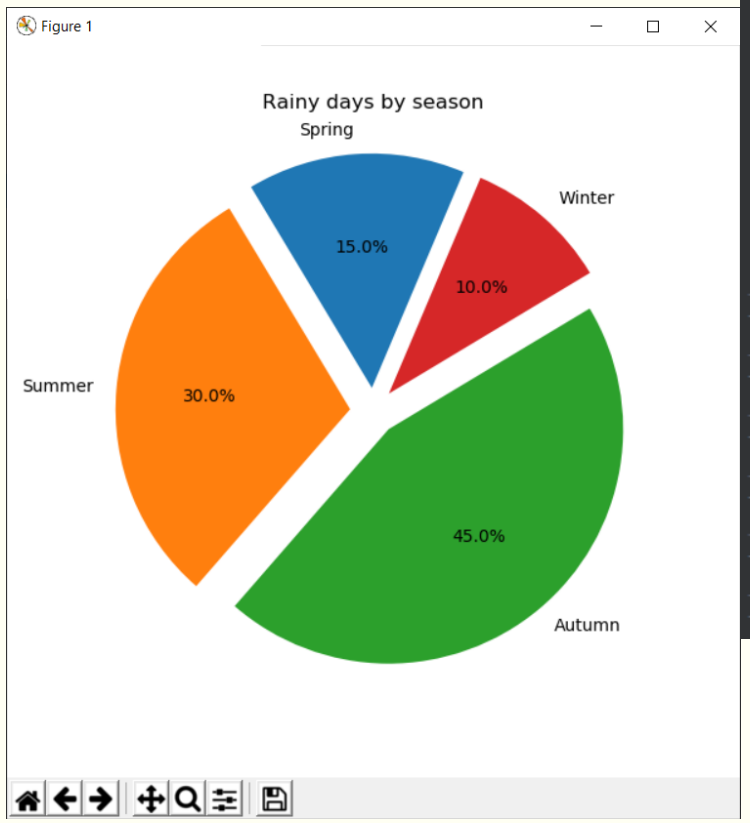
- Для візуалізації неточності вимірювання в наборі даних або для виявлення похибок можна використовувати error bars.
  - Вони можуть показати стандартне відхилення (standard deviation), стандартну похибку (standard error) або 95%-ий інтервал достовірності (confidence interval).
  - Це не стандартизовано, тому завжди явно перевіряйте, які значення (похибки) error bars показують.
- Разом з двома обов’язковими параметрами—left і height—можна використовувати й опційні:
  - ***width:*** ширина прямокутника. За умовчанням – 0.8.
  - ***bottom:*** якщо задане, це значення додається до height. За умовчанням – None.
  - ***edgecolor:*** задає колір контурів прямокутників.
  - ***ecolor:*** задає колір будь-якого error bar.
  - ***linewidth:*** задає ширину контуру прямокутника; спеціальні значення – None (за умовчанням) та 0 (коли контури не відображаються).
  - ***orientation:*** вертикальна або горизонтальна орієнтація.
  - ***xerr*** та ***yerr:*** використовуються для генерування error bars на стовпчиковій діаграмі.

```python
import numpy as np
import matplotlib.pyplot as plt
# generate number of measurements
x = np.arange(0, 10, 1)
# values computed from "measured"
y = np.log(x)
# add some error samples from standard normal distribution
xe = 0.1 * np.abs(np.random.randn(len(y)))
# draw and show errorbar
plt.bar(x, y, yerr=xe, width=0.4, align='center', ecolor='r',
    color='cyan', label='experiment #1');
# give some explainations
plt.xlabel('# measurement')
plt.ylabel('Measured values')
plt.title('Measurements')
plt.legend(loc='upper left')
plt.show()
```



- Another interesting option to use if we are preparing visualizations for a black-and-white medium is hatch; it can have the following values:
  - Якщо природа набору даних is such that errors are not the same in both directions (negative and positive), we can also specify them separately using asymmetrical error bars.
  - All we have to do differently is to specify xerr or yerr using a two-element list (such as a 2D array), where the first list contains values for negative errors and the second one for positive errors.

Hatch value	Description
/	Diagonal hatching
\	Back diagonal
|	Vertical hatching
-	Horizontal
+	Crossed
x	Crossed diagonal
o	Small circle
0	Large circle
.	Dot pattern
*	Star pattern

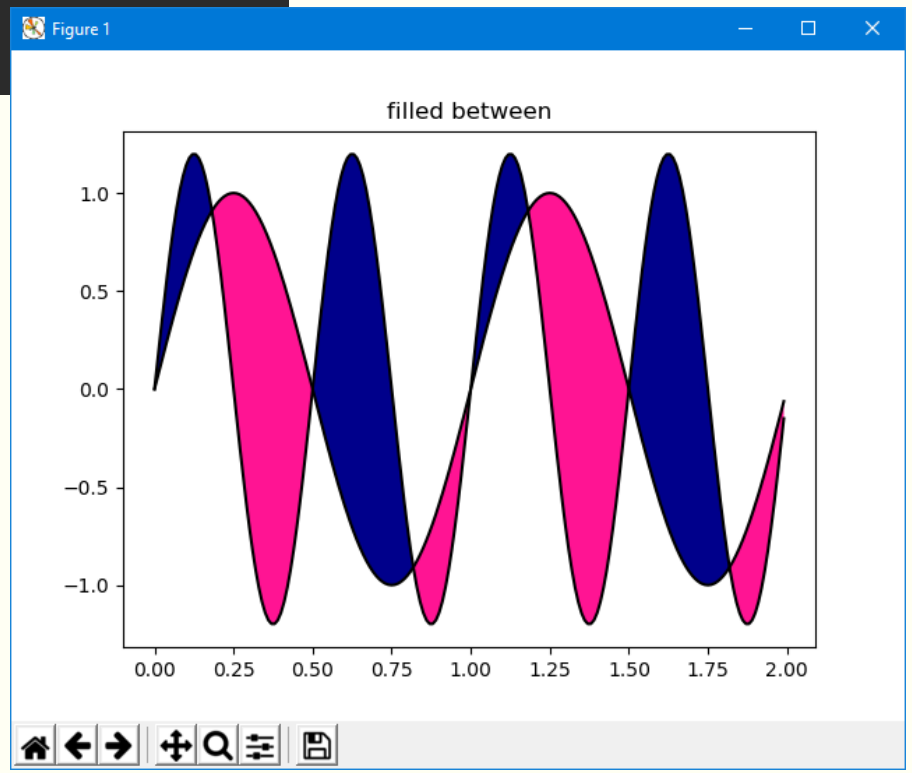


Кругова діаграма

```
1   from pylab import *
2   # make a square figure and axes
3   figure(1, figsize=(6,6))
4   ax = axes([0.1, 0.1, 0.8, 0.8])
5   # the slices will be ordered
6   # and plotted counter-clockwise.
7   labels = 'Spring', 'Summer', 'Autumn', 'Winter'
8   # fractions are either x/sum(x) or x if sum(x) <= 1
9   x = [15, 30, 45, 10]
10  # explode must be len(x) sequence or None
11  explode=(0.1, 0.1, 0.1, 0.1)
12  pie(x, explode=explode, labels=labels,
13  autopct='%1.1f%%', startangle=67)
14  title('Rainy days by season')
15  show()
```

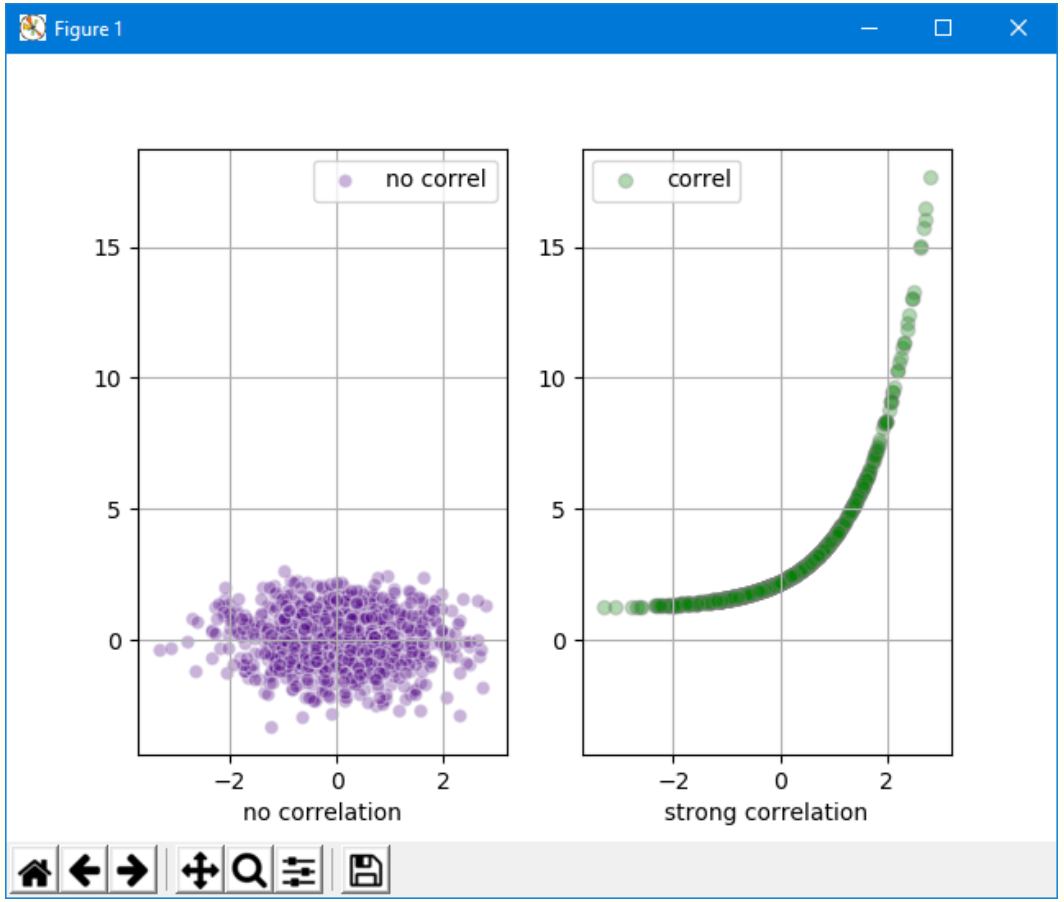
```python
from matplotlib.pyplot import figure, show, gca
import numpy as np
x = np.arange(0.0, 2, 0.01)
# two different signals are measured
y1 = np.sin(2*np.pi*x)
y2 = 1.2*np.sin(4*np.pi*x)
fig = figure()
ax = gca()
# plot and fill between y1 and y2 where a logical condition is met
ax.plot(x, y1, x, y2, color='black')
ax.fill_between(x, y1, y2, where=y2>=y1, facecolor='darkblue', interpolate=True)
ax.fill_between(x, y1, y2, where=y2<=y1, facecolor='deeppink', interpolate=True)
ax.set_title('filled between')
show()
```

- Бібліотека matplotlib дозволяє заповнювати проміжки між кривими - функція fill_between().



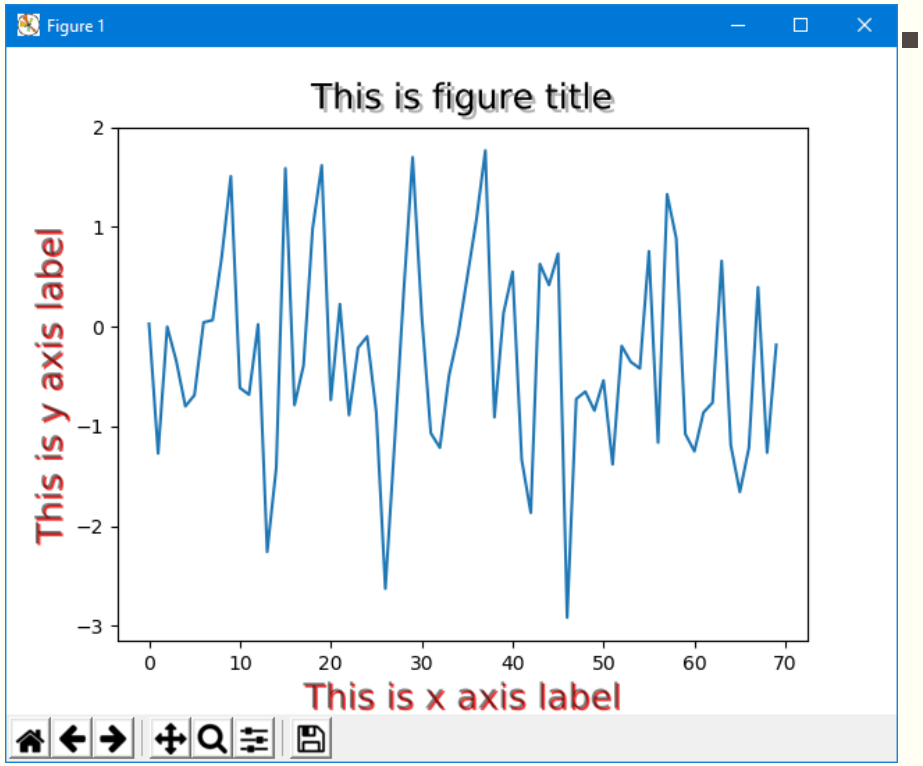
Малювання scatter plots з кольоровими маркерами

```python
import matplotlib.pyplot as plt
import numpy as np
# generate x values
x = np.random.randn(1000)
# random measurements, no correlation
y1 = np.random.randn(len(x))
# strong correlation
y2 = 1.2 + np.exp(x)
ax1 = plt.subplot(121)
plt.scatter(x, y1, color='indigo', alpha=0.3, edgecolors='white',
label='no correl')
plt.xlabel('no correlation')
plt.grid(True)
plt.legend()
ax2 = plt.subplot(122, sharey=ax1, sharex=ax1)
plt.scatter(x, y2, color='green', alpha=0.3, edgecolors='grey',
label='correl')
plt.xlabel('strong correlation')
plt.grid(True)
plt.legend()
plt.show()
```



- Scatter plots відображають значення з 2 наборів даних.
  - Візуалізація даних виконується як хмара нез’єднаних точок.

Налаштування прозорості та розміру підписів осей (axis labels)



- Підписи для осей описують, які дані представлено на рисунку.
  - 1. Створимо графік для деяких випадкових чисел.
  - 2. Додамо заголовок та підписи осей.
  - 3. Додамо налаштування прозорості (alpha).
  - 4. Додамо ефекти тіней до заголовку та підписів осей.

```python
import matplotlib.pyplot as plt
from matplotlib import patheffects
import numpy as np
data = np.random.randn(70)
fontsize = 18
plt.plot(data)
title = "This is figure title"
x_label = "This is x axis label"
y_label = "This is y axis label"
title_text_obj = plt.title(title, fontsize=fontsize, verticalalignment='bottom')
title_text_obj.set_path_effects([patheffects.withSimplePatchShadow()])
# offset_xy -- set the 'angle' of the shadow
# shadow_rgbFace -- set the color of the shadow
# patch_alpha -- setup the transparency of the shadow
offset_xy = (1, -1)
rgbRed = (1.0,0.0,0.0)
alpha = 0.8
```

Продовження коду

```
18  # customize shadow properties
19  pe = patheffects.withSimplePatchShadow(offset = offset_xy,
20                                         shadow_rgbFace = rgbRed,
21                                         alpha = alpha)
22  # apply them to the xaxis and yaxis labels
23  xlabel_obj = plt.xlabel(x_label, fontsize=fontsize, alpha=0.5)
24  xlabel_obj.set_path_effects([pe])
25  ylabel_obj = plt.ylabel(y_label, fontsize=fontsize, alpha=0.5)
26  ylabel_obj.set_path_effects([pe])
27  plt.show()
```

- Після усіх налаштувань інстанціюємо matplotlib.patheffects.withSimplePatch Shadow та зберігаємо посилання на нього у змінній pe з метою повторного використання потім.
  - Для накладання тіні потрібно мати доступ до об’єктів-підписів.
  - За допомогою matplotlib.pyplot.xlabel() повертаємо посилання на об’єкт (matplotlib.text.Text), який використаємо при виклику set_path_effects([pe]).

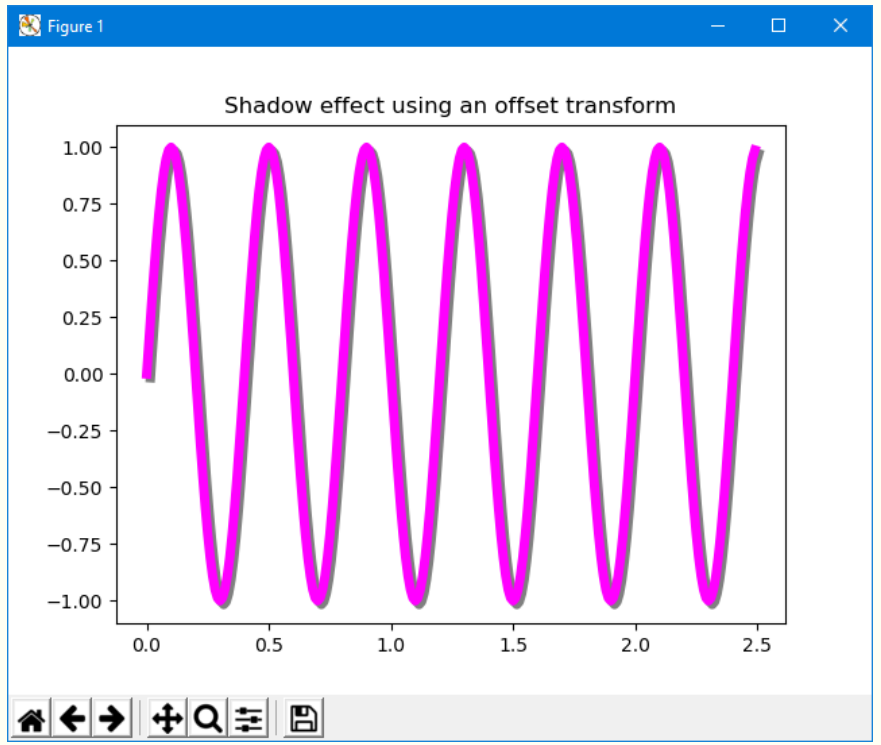
Додавання тіні до chart line

- Потрібно використовувати transformation framework з пакету matplotlib.transforms.
  - Перетворення (Transformations) знають, як конвертувати задані координати з їх координатної системи для відображення на екрані, і навпаки.

Coordinate system	Transformation object	Description
`Data`	`Axes.transData`	Represents the user's data coordinate system.
`Axes`	`Axes.transAxes`	Represents the `Axes` coordinate system, where (0,0) represents the bottom-left end of the axes and (1,1) represents the upper-right end of the axes.
`Figure`	`Figure.transFigure`	This is the `Figure` coordinate system, where (0,0) represents the bottom-left end of the figure and (1,1) represents the upper-right end of the figure.
`Display`	`None`	Represents the pixel coordinate system of the user display, where (0,0) represents the bottom-left of the display, and tuple (width, height) represents the upper-right of the display, where width and height are in pixels.

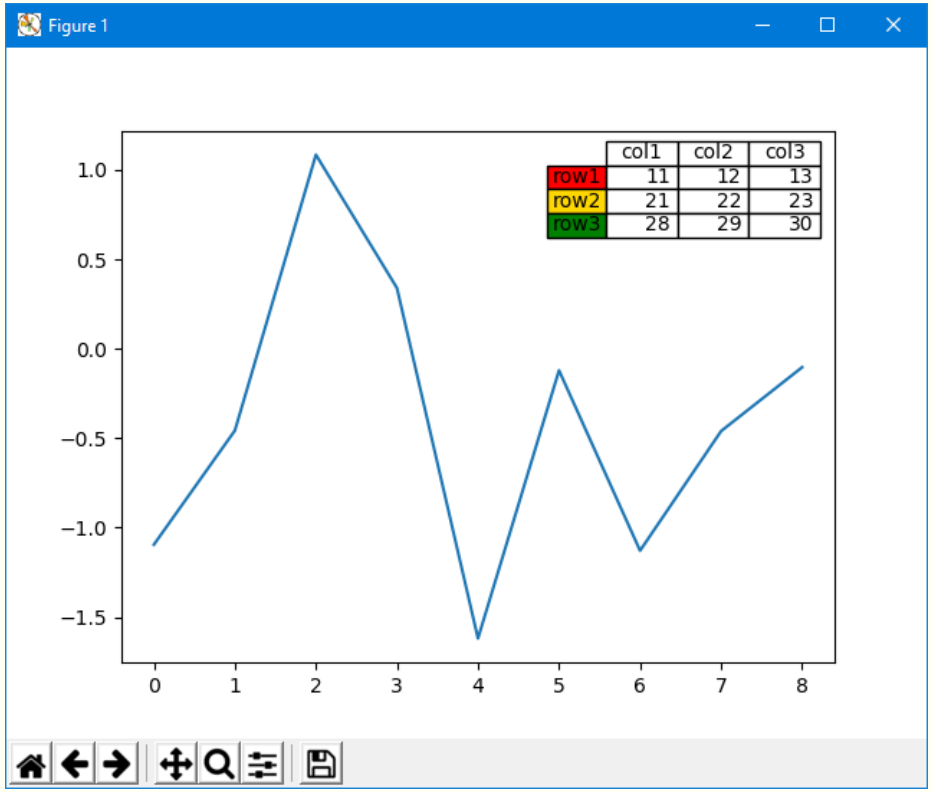
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.transforms as transforms
def setup(layout=None):
    assert layout is not None
    fig = plt.figure()
    ax = fig.add_subplot(layout)
    return fig, ax

def get_signal():
    t = np.arange(0., 2.5, 0.01)
    s = np.sin(5 * np.pi * t)
    return t, s

def plot_signal(t, s):
    line, = axes.plot(t, s, linewidth=5, color='magenta')
    return line,
def make_shadow(fig, axes, line, t, s):
    delta = 2 / 72. # how many points to move the shadow
    offset = transforms.ScaledTranslation(delta, -delta, fig.dpi_scale_trans)
    offset_transform = axes.transData + offset
    # We plot the same data, but now using offset transform
    # zorder -- to render it below the line
    axes.plot(t, s, linewidth=5, color='gray',
    transform=offset_transform,zorder=0.5 * line.get_zorder())

if __name__ == "__main__":
    fig, axes = setup(111)
    t, s = get_signal()
    line, = plot_signal(t, s)
    make_shadow(fig, axes, line, t, s)
    axes.set_title('Shadow effect using an offset transform')
    plt.show()
```

- Спочатку створимо рисунок у setup(); потім отримуємо сигнал (генерує дані—синусоїду).
  - Будуємо графік базового сигналу plot_signal().
  - Далі накладаємо тінь у make_shadow().
  - Використовуємо ефект зміщення (offset).



Додавання таблиці до рисунку

```python
import matplotlib.pylab as plt
import numpy as np
plt.figure()
ax = plt.gca()
y = np.random.randn(9)
col_labels = ['col1','col2','col3']
row_labels = ['row1','row2','row3']
table_vals = [[11, 12, 13], [21, 22, 23], [28, 29, 30]]
row_colors = ['red', 'gold', 'green']
my_table = plt.table(cellText=table_vals,
                     colWidths=[0.1] * 3,
                     rowLabels=row_labels,
                     colLabels=col_labels,
                     rowColours=row_colors,
                     loc='upper right')
plt.plot(y)
plt.show()
```



- Базова сигнатура функції:

```
table(cellText=None, cellColours=None,
      cellLoc='right', colWidths=None,
      rowLabels=None, rowColours=None, rowLoc='left',
      colLabels=None, colColours=None, colLoc='center',
      loc='bottom', bbox=None)
```

- Функція створює та повертає екземпляр matplotlib.table.Table.
- Клас Table дозволяє напряму налаштувати таблицю до її додавання на екземпляр осі за допомогою add_table().

Використання підграфіків (subplots)

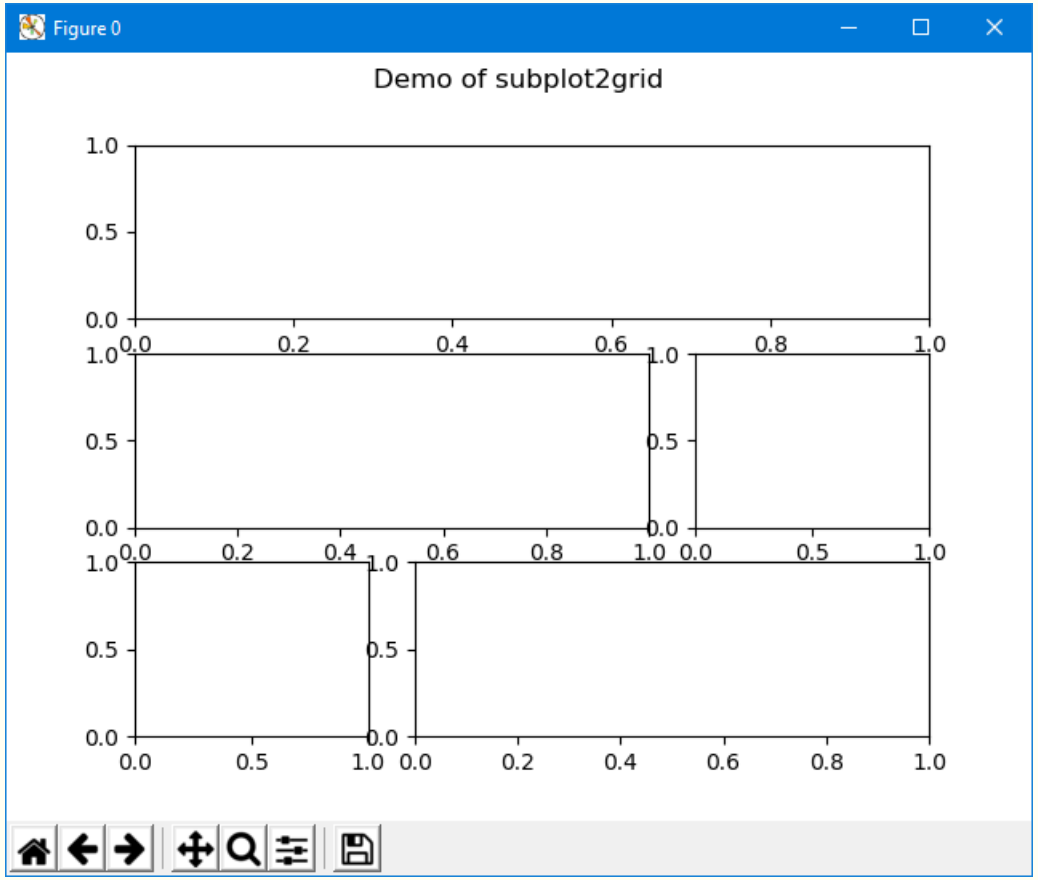
- Базовий клас для підграфіків – matplotlib.axes.SubplotBase.
  - Підграфіки є екземплярами matplotlib.axes.Axes, проте постачають допоміжні методи для генерування та оперування набором Axes всередині рисунку.
- Доступний клас matplotlib.figure.SubplotParams, який містить всі параметри підграфіка.
  - Одиниці вимірювання нормалізуються по ширині та висоті рисунку.
  - Як уже відомо, якщо не задати власних значень, вони зчитуються з параметрів rc.
- Скриптинговий прошарок (scripting layer, matplotlib.pyplot) містить кілька допоміжних методів для роботи з підграфіками.
  - matplotlib.pyplot.subplots використовується для простого створення поширених макетів підграфіків.
  - Можемо задати розмір сітки—кількість рядків та стовпців сітки підграфіку.

Можемо створити підграфіки, які мають спільні осі x або y

- Застосовуються аргументи sharex або sharey.
  - sharex = True, якщо вісь x спільна для всіх підграфіків. Написи для насічок будуть невидимими на всіх, крім останнього рядка графіків.
  - They can also be defined as String, with enumerated values of row, col, all, or none.
  - The value all is the same as True, and the value none is the same as False.
  - Якщо задається значення row, кожен підграфік у ряду має спільну вісь x.
  - Якщо задається значення col, кожний стовпчик підграфіків має спільну вісь x.
  - This helper returns tuple fig, ax where ax is either an axis instance or, if more than one subplot is created, an array of axis instances.
- matplotlib.pyplot.subplots_adjust використовуються для налаштування макету підграфіка.
  - Іменовані аргументи задають координати підграфіків всередині рисунку (left, right, bottom, top), нормалізовані до розміру рисунку.
  - Відступи між підграфіками можна задати за допомогою аргументів wspace та hspace.

Інша допоміжна функція в matplotlib—subplot2grid()

```python
import matplotlib.pyplot as plt
plt.figure(0)
axes1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
axes2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)
axes3 = plt.subplot2grid((3, 3), (1, 2))
axes4 = plt.subplot2grid((3, 3), (2, 0))
axes5 = plt.subplot2grid((3, 3), (2, 1), colspan=2)
# tidy up tick labels size
all_axes = plt.gcf().axes
for ax in all_axes:
    for ticklabel in ax.get_xticklabels() + ax.get_yticklabels():
        ticklabel.set_fontsize(10)

plt.suptitle("Demo of subplot2grid")
plt.show()
```

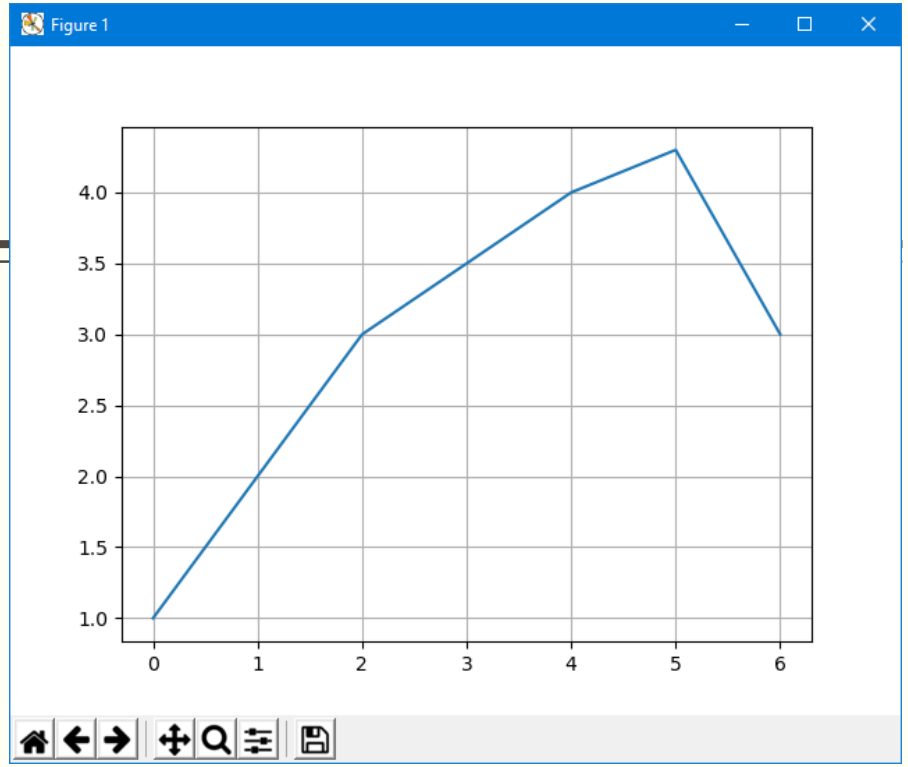


Альтернативи

- ```
  axes = fig.add_subplot(111)
  rectangle = axes.patch
  rectangle.set_facecolor('blue')
  ```
  - Кожен екземпляр axes містить поле patch, яке посилається на об’єкт-прямокутник, що представляє фон для екземпляру поточної осі; доступні властивості для оновлення кольору фону, завантаження зображення, додавання водяного знаку тощо.
- Також можливе створення patch, а потім його додавання до фону осі:
- ```
  fig = plt.figure()
  axes = fig.add_subplot(111)
  rect = matplotlib.patches.Rectangle((1,1), width=6, height=12)
  axes.add_patch(rect)
  # потрібно вручну запустити відрисовку рисунку
  axes.figure.canvas.draw()
  ```

Кастомізація сіток

```
1  import matplotlib.pyplot as plt
2  plt.figure(1)
3  plt.plot([1,2,3,3.5,4,4.3,3])
4  plt.grid()
5  plt.show()
```

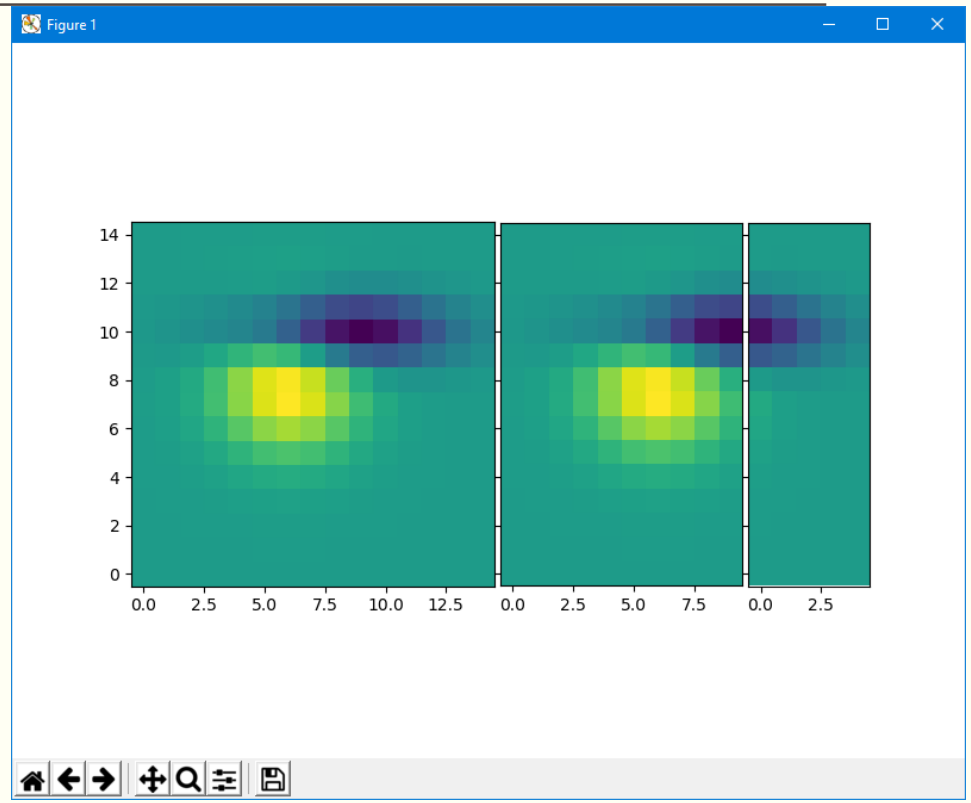


- Для можливості налаштування видимості, частоти відображення та стилю сітки зазвичай використовують matplotlib.pyplot.grid.
- Крім ввімкнення-вимкнення сітки, можна маніпулювати нею мажорними та/або мінорними насічками; hence, the value of function argument which can be 'major', 'minor', or 'both'.
- Усі інші властивості передаються за допомогою kwargs та представляють стандартний набір властивостей, прийнятних для екземпляру matplotlib.lines.Line2D, зокрема color, linestyle, linewidth:
  - `ax.grid(color='g', linestyle='--', linewidth=1)`

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
from matplotlib.cbook import get_sample_data

def get_demo_image():
    f = get_sample_data("axes_grid/bivariate_normal.npy", asfileobj=False)
    # z is a numpy array of 15x15
    Z = np.load(f)
    return Z, (-3, 4, -4, 3)

def get_grid(fig=None, layout=None, nrows_ncols=None):
    assert fig is not None
    assert layout is not None
    assert nrows_ncols is not None
    grid = ImageGrid(fig, layout, nrows_ncols=nrows_ncols,
                     axes_pad=0.05, add_all=True, label_mode="L")
    return grid
```

- У функції get_demo_image() завантажуємо дані з sample-папки з matplotlib.
  - Список grid містить axes grid (тут – ImageGrid).

```python
def load_images_to_grid(grid, Z, *images):
    min, max = Z.min(), Z.max()
    for i, image in enumerate(images):
        axes = grid[i]
        axes.imshow(image, origin="lower", vmin=min, vmax=max,
                    interpolation="nearest")

if __name__ == "__main__":
    fig = plt.figure(1, (8, 6))
    grid = get_grid(fig, 111, (1, 3))
    Z, extent = get_demo_image()
    # Slice image
    image1 = Z
    image2 = Z[:, :10]
    image3 = Z[:, 10:]
    load_images_to_grid(grid, Z, image1, image2, image3)
    plt.draw()
    plt.show()
```



- Змінні image1, image2 та image3 містять розрізані (sliced) дані з Z, які були розбиті по багатьох осях у списку grid.
  - Looping over all the grids, we are plotting data from im1, im2, and im3 using the standard imshow() call, while matplotlib takes care that everything is neatly rendered and aligned.

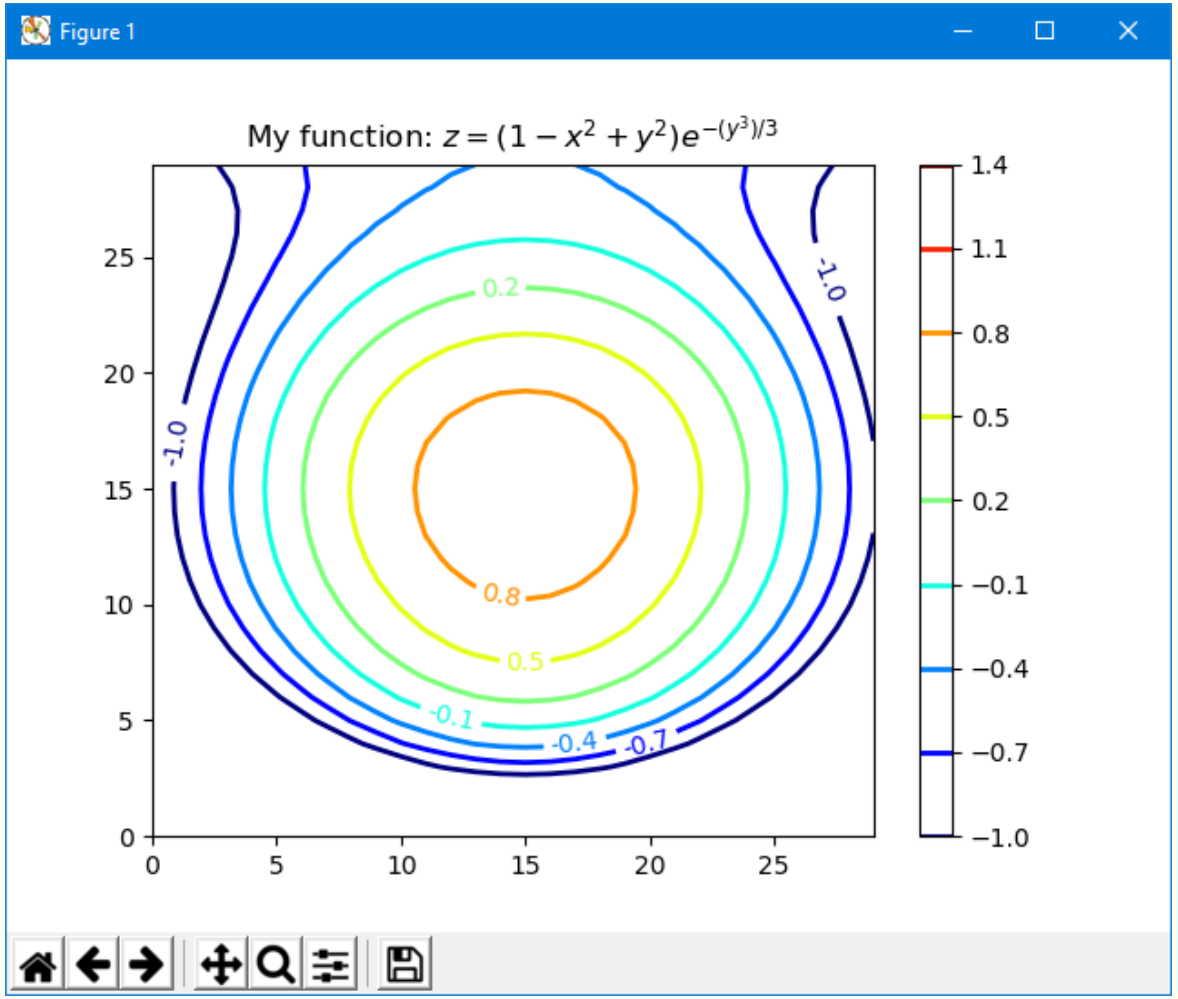
Створення контурних графіків

- Контурний графік відображає ізолінії – криві, в яких двопараметрична функція має однакове значення.
  - Контури представляються as a contour plot of matrix Z, де Z інтерпретується як висота над XY площиною.
- Проблема контурних графіків – їх представлення без позначення ізоліній досить даремне, оскільки неможливо декодувати висоту точок, знайти локальні мінімуми тощо.
  - Позначення (labeling) ізоліній можна здійснити або за допомогою написів (clabel()), або карт кольорів (colormaps).
- Інший ризик роботи з контурними графіками – вибір кількості ізоліній для побудови.
  - При надто великій кількості графік стає надто щільним для розбору, а при надто маленькій – неінформативним та неточним.
  - Функція contour() автоматично вгадує, скільки ізоліній будувати, проте можна й задавати власну кількість.
  - У matplotlib використовується matplotlib.pyplot.contour.
- Доступні подібні функції contour() та contourf() (контури з заливкою).
  - Будемо демонструвати лише contour(), оскільки більшість з цього застосовна і до contourf().
  - Вони працюють майже з однаковими аргументами.

Функція contour() може мати різні сигнатури виклику

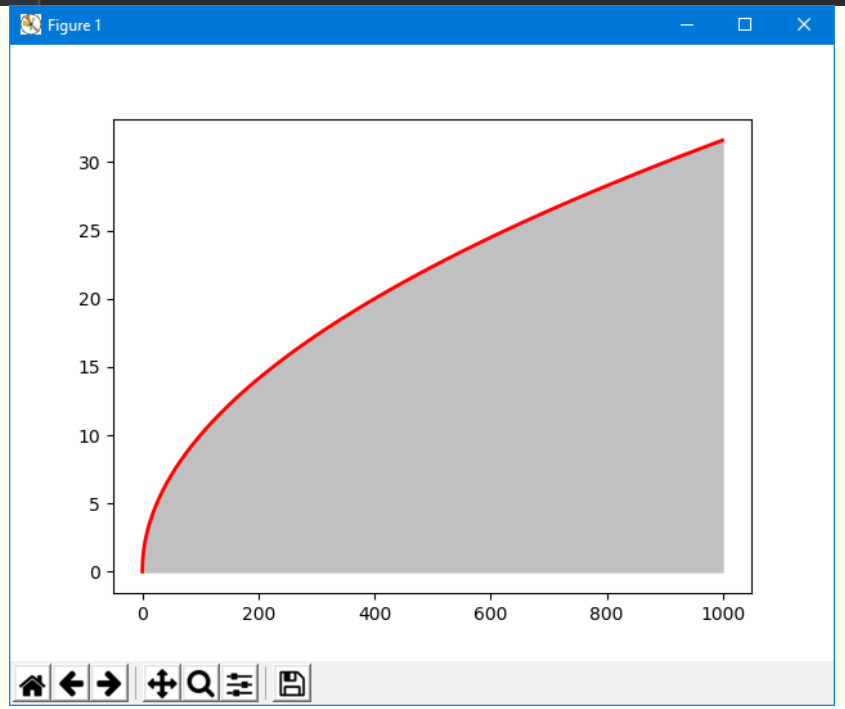
Call signature	Description
`contour(Z)`	Plots the contour of Z (array). The level values are chosen automatically.
`contour(X,Y,Z)`	Plots the contour of X, Y, and Z. The arrays `X` and `Y` are (x, y) surface coordinates.
`contour(Z,N)` `contour(X,Y,Z,N)`	Plots the contour of `Z`, where the number of levels is defined with `N`. The level values are automatically chosen.
`contour(Z,V)` `contour(X,Y,Z,V)`	Plots the contour lines with levels at the values specified in `V`.
`contourf(..., V)`	Fills the `len(V)-1` regions between the level values in sequence `V`.
`contour(Z, **kwargs)`	Uses keyword arguments to control common line properties (colors, line width, origin, color map, and so on).

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

def process_signals(x, y):
    return (1 - (x**2 + y**2)) * np.exp(-y**3 / 3)

x = np.arange(-1.5, 1.5, 0.1)
y = np.arange(-1.5, 1.5, 0.1)
# Make grids of points
X, Y = np.meshgrid(x, y)
Z = process_signals(X, Y)
# Number of isolines
N = np.arange(-1, 1.5, 0.3)
# adding the Contour lines with labels
CS = plt.contour(Z, N, linewidths=2, cmap=mpl.cm.jet)
plt.clabel(CS, inline=True, fmt='%1.1f', fontsize=10)
plt.colorbar(CS)
plt.title('My function: $z=(1-x^2+y^2) e^{-(y^3)/3}$')
plt.show()
```



Заливка under-plot області

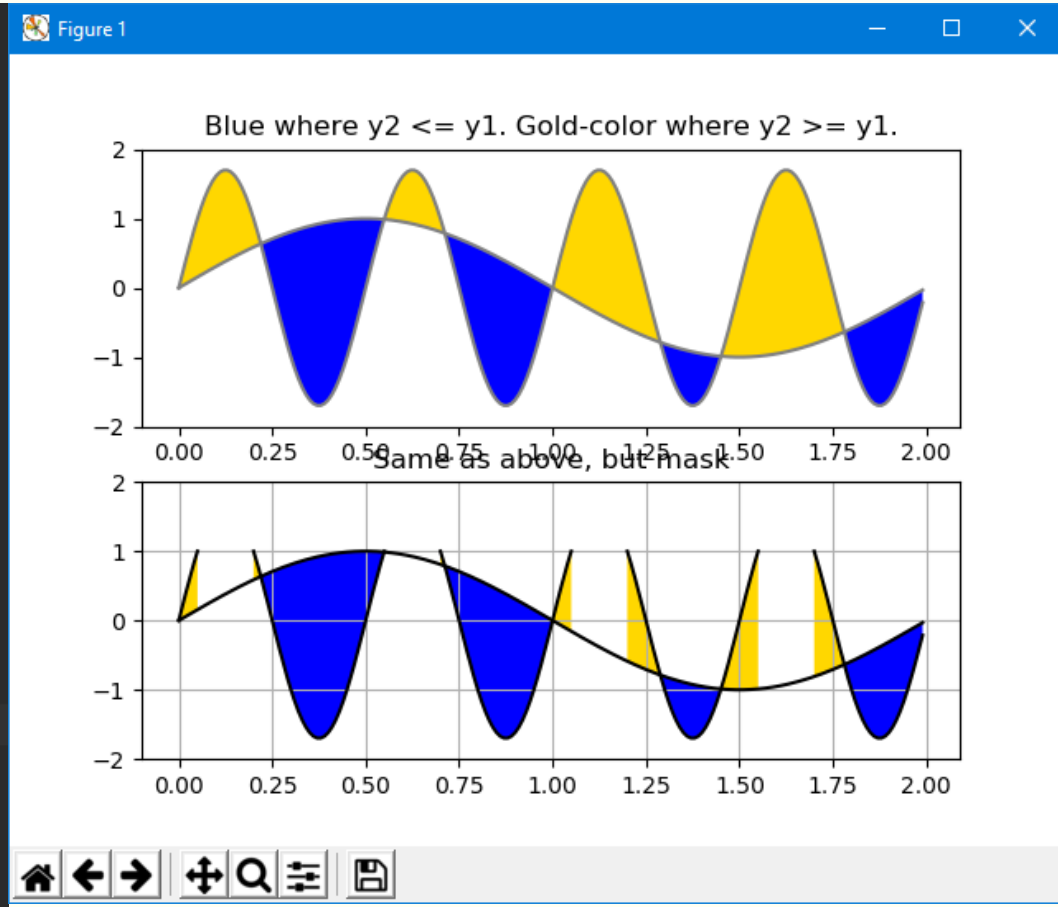
```
import matplotlib.pyplot as plt
from math import sqrt
t = range(1000)
y = [sqrt(i) for i in t]
plt.plot(t, y, color='red', lw=2)
plt.fill_between(t, y, color='silver')
plt.show()
```



- Базовий спосіб відрисовки залитого багатокутника в matplotlib – застосувати matplotlib.pyplot.fill().
  - Дана функція приймає подібні до matplotlib.pyplot.plot() аргументи—багато пар x та y та інші властивості Line2D.
  - Повертає список екземплярів Patch, які було додано.
- matplotlib постачає кілька функцій, які допомагають будувати залиті фігури, apart from plotting functions that are inherently plotting closed filled polygons, such as histogram(), of course.
- Також доступні функції matplotlib.pyplot.fill_between() та matplotlib.pyploy.fill_betweenx().
  - Вони заливають багатокутники між 2 кривими.
  - Основна відмінність між функціями - is that the latter fills between the x axis values, whereas the former fills between the y axis values.
  - Функція fill_between() приймає аргументи x, y1 та y2—масиви значень по осях.

Інший приклад: більше умов для функції заливки

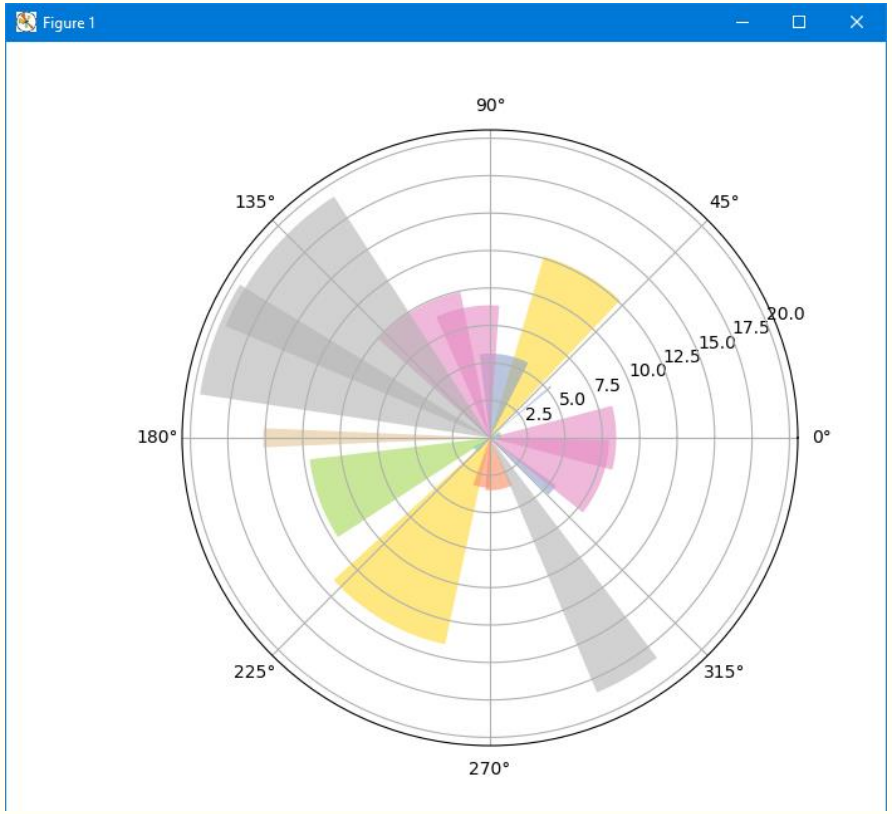
```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0.0, 2, 0.01)
y1 = np.sin(np.pi*x)
y2 = 1.7*np.sin(4*np.pi*x)
fig = plt.figure()
axes1 = fig.add_subplot(211)
axes1.plot(x, y1, x, y2, color='grey')
axes1.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue', interpolate=True)
axes1.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold', interpolate=True)
axes1.set_title('Blue where y2 <= y1. Gold-color where y2 >= y1.')
axes1.set_ylim(-2,2)
# Mask values in y2 with value greater than 1.0
y2 = np.ma.masked_greater(y2, 1.0)
axes2 = fig.add_subplot(212, sharex=axes1)
axes2.plot(x, y1, x, y2, color='black')
axes2.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue', interpolate=True)
axes2.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold', interpolate=True)
axes2.set_title('Same as above, but mask')
axes2.set_ylim(-2,2)
axes2.grid('on')
plt.show()
```



Відрисовка в полярних координатах

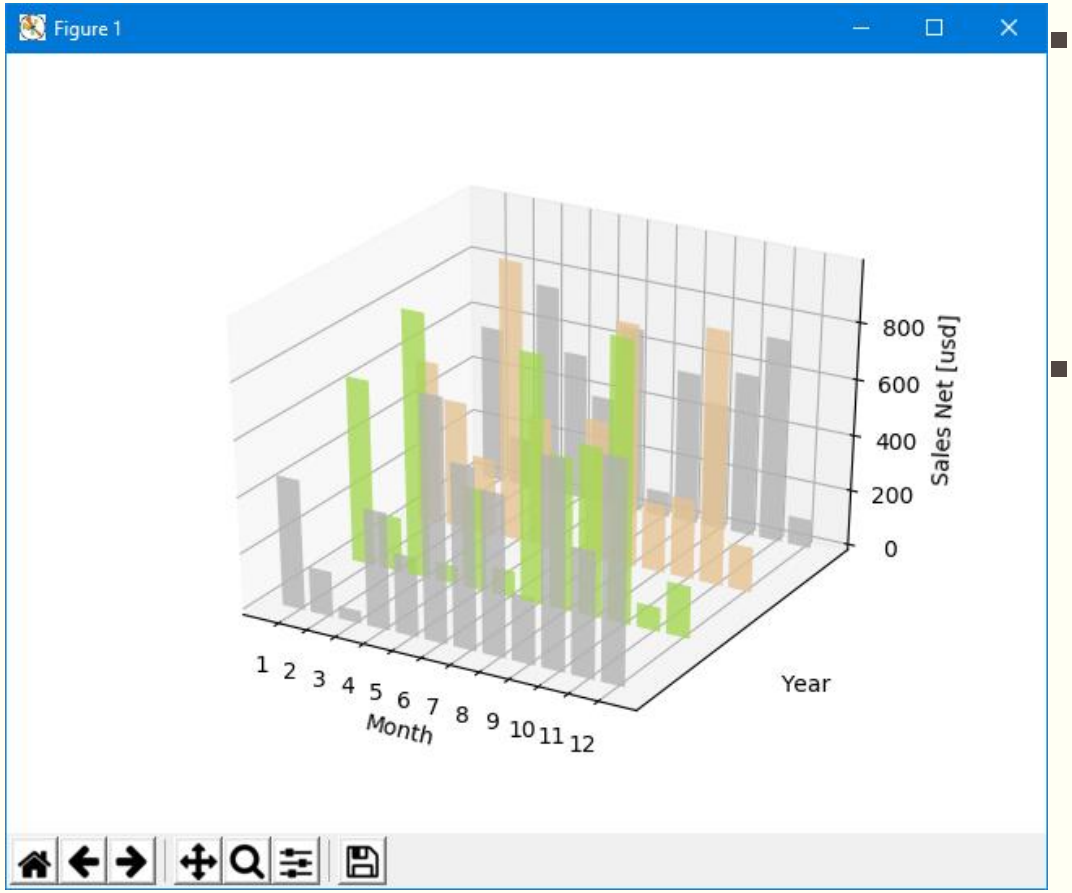
- У полярній системі координат точка описується радіус-вектором r та кутом theta.
  - Бібліотека matplotlib виконристовує градуси для позначення величини кутів.
  - Для відрисовки використовується функція polar(), яка приймає 2 масиви параметрів, theta та r. Також приймаються інші аргументи, як у функції plot().

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

figsize = 7
colormap = lambda r: cm.Set2(r / 20.)
N = 18 # number of bars
fig = plt.figure(figsize=(figsize,figsize))
ax = fig.add_axes([0.2, 0.2, 0.7, 0.7], polar=True)
theta = np.arange(0.0, 2*np.pi, 2*np.pi/N)
radii = 20*np.random.rand(N)
width = np.pi/4*np.random.rand(N)
bars = ax.bar(theta, radii, width=width, bottom=0.0)
for r, bar in zip(radii, bars):
    bar.set_facecolor(colormap(r))
    bar.set_alpha(0.6)
plt.show()
```



- We also need to tell matplotlib that we want axes in the polar coordinate system.
  - This is done by providing the polar=True argument to the add_axes or add_subplot functions.
  - Additionally, to set other properties on the figure, such as grids on radii or angles, we need to use matplotlib.pyplot.rgrids() to toggle radial grid visibility or to set up labels.
  - Similarly, we use matplotlib.pyplot.thetagrid() to configure angle ticks and labels.
- We then generate random values for a set of angles (theta) and a set of polar distances (radii).
  - Because we drew bars, we also needed a set of widths for each bar, so we also generated a set of widths.
  - Since maplotlib.axes.bar accepts an array of values (as almost all the drawing functions in matplotlib do) we don't have to loop over this generated dataset; we just need to call the bar once with all the arguments passed to it.

Створення 3D bars

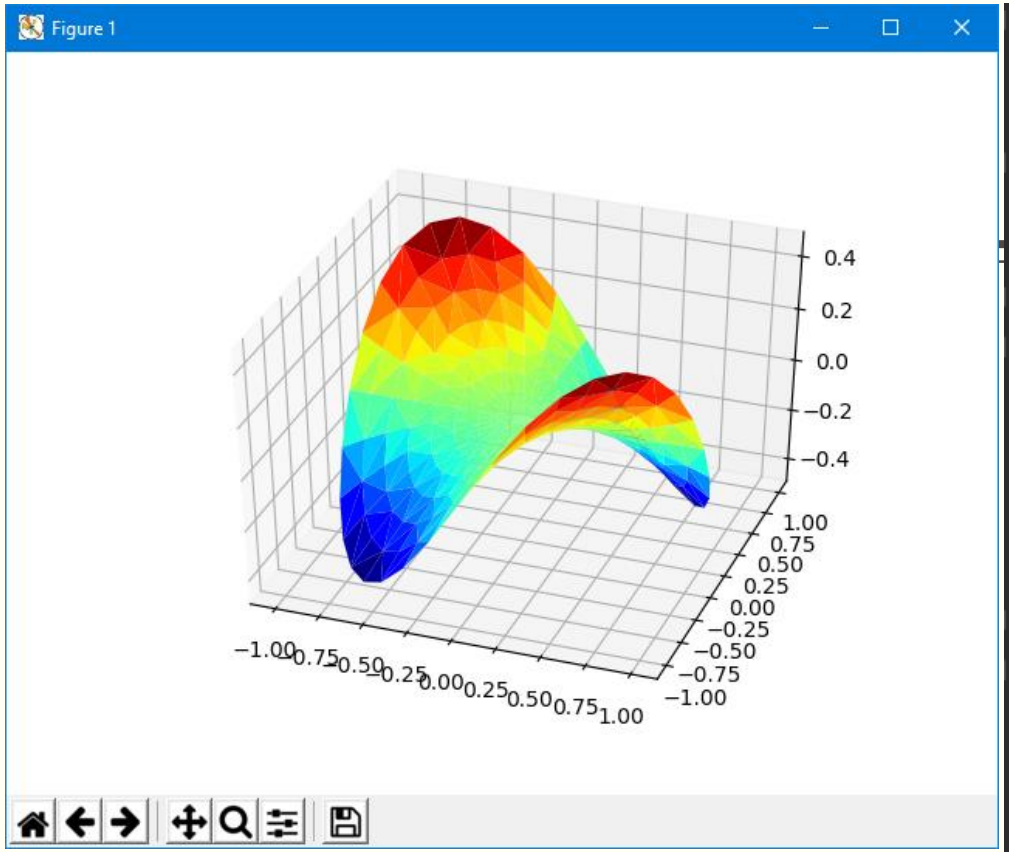


- Існують різні розширення (тулкіти – колекції специфічних функцій, сфокусованих на одному напрямку) для побудови 3D-графіків.
  - Популярні тулкіти: Basemap, GTK Tools, Excel Tools, Natgrid, AxesGrid та mplot3d.
- Розглянемо mplot3d у прикладі.
  - Тулкіт mpl_toolkits.mplot3d постачає деякі базові можливості побудови 3D-графіків.
  - Підтримувані графіки: scatter, surf, line та mesh.
  - Хоч це не найкраща бібліотека, вона включається в інтерфейс matplotlib.

Загалом, все ще потрібно створювати рисунок та додавати бажані осі

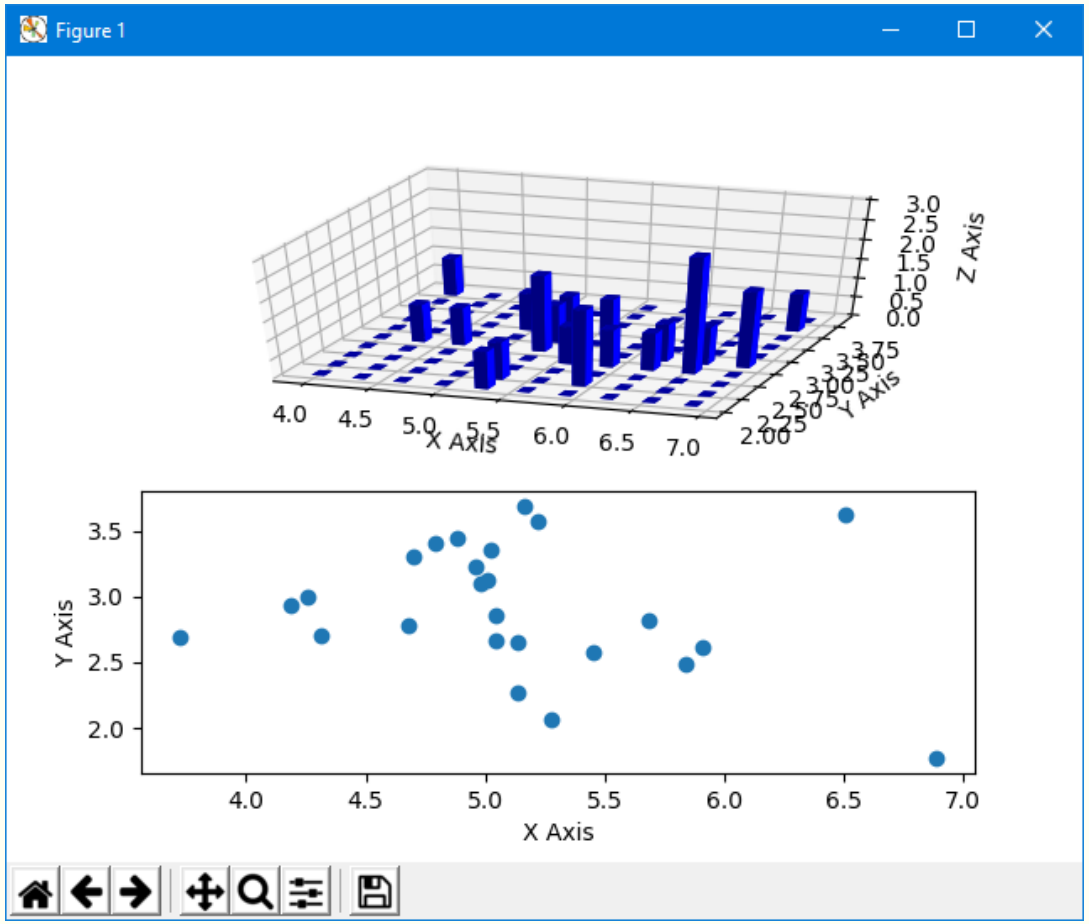
```
import random
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from mpl_toolkits.mplot3d import Axes3D

mpl.rcParams['font.size'] = 10
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for z in [2011, 2012, 2013, 2014]:
    xs = range(1,13)
    ys = 1000 * np.random.rand(12)
    color = plt.cm.Set2(random.choice(range(plt.cm.Set2.N)))
    ax.bar(xs, ys, zs=z, zdir='y', color=color, alpha=0.8)

ax.xaxis.set_major_locator(mpl.ticker.FixedLocator(xs))
ax.yaxis.set_major_locator(mpl.ticker.FixedLocator(ys))
ax.set_xlabel('Month')
ax.set_ylabel('Year')
ax.set_zlabel('Sales Net [usd]')
plt.show()
```

- Відмінність: задаємо 3D-проєкцію для рисунку, а осі додаються як Axes3D.
  - Наприклад, функція mpl_toolkits.mplot3d.Axes3D.plot() задає аргументи xs, ys, zs та zdir.
  - Решта аргументів передаються напряму в matplotlib.axes.Axes.plot().
- Специфічні аргументи:
  - xs та ys: координати по осях x та y.
  - zs: значення по осі z. Одне для всіх точок або для кожної з них.
  - zdir: обирає розмірність z-осі (зазвичай zs, проте можуть бути xs або ys)




```python
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
n_angles = 36
n_radii = 8
# An array of radii
# Does not include radius r=0, this is to eliminate duplicate points
radii = np.linspace(0.125, 1.0, n_radii)
# An array of angles
angles = np.linspace(0, 2*np.pi, n_angles, endpoint=False)
# Repeat all angles for each radius
angles = np.repeat(angles[...,np.newaxis], n_radii, axis=1)
# Convert polar (radii, angles) coords to cartesian (x, y) coords
# (0, 0) is added here. There are no duplicate points in the (x, y) plane
x = np.append(0, (radii*np.cos(angles)).flatten())
y = np.append(0, (radii*np.sin(angles)).flatten())
# Pringle surface
z = np.sin(-x*y)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
plt.show()
```

Створення 3D-гістограм




```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D

mpl.rcParams['font.size'] = 10
samples = 25
x = np.random.normal(5, 1, samples)
y = np.random.normal(3, .5, samples)
fig = plt.figure()
ax1 = fig.add_subplot(211, projection='3d')
# compute two-dimensional histogram
hist, xedges, yedges = np.histogram2d(x, y, bins=10)
# compute location of the x,y bar positions
elements = (len(xedges) - 1) * (len(yedges) - 1)
xpos, ypos = np.meshgrid(xedges[:-1]+.25, yedges[:-1]+.25)
xpos = xpos.flatten()
ypos = ypos.flatten()
zpos = np.zeros(elements)
# make every bar the same width in base
dx = .1 * np.ones_like(zpos)
dy = dx.copy()
```

- 1. Використовуємо NumPy, оскільки в ній доступна функція для обчислення гістограми двох змінних.
- 2. Генеруємо x та y з нормальних розподілів, проте з різними параметрами.
- 3. Будуємо scatter-графік того ж набору даних, щоб продемонструвати, наскільки різне представлення.

```
24      # this defines the height of the bar
25      dz = hist.flatten()
26      ax1.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', alpha=0.4)
27      ax1.set_xlabel('X Axis')
28      ax1.set_ylabel('Y Axis')
29      ax1.set_zlabel('Z Axis')
30      # plot the same x,y correlation in scatter plot
31      # for comparison
32      ax2 = fig.add_subplot(212)
33      ax2.scatter(x, y)
34      ax2.set_xlabel('X Axis')
35      ax2.set_ylabel('Y Axis')
36      plt.show()
```

- Гістограма готується за допомогою функції np.histogram2d(), яка повертає гістограму (hist) та межі бінів x та y.
  - Оскільки для функції bard3d() необхідні координати в просторі (x, y), потрібно обчислювати спільні матричні координати (функція np.meshgrid()), утворені комбінуванням векторів x та y в 2D-сітку.
  - Використовується для побудови паралелепіпедів на площині xy.
  - Змінні dx та dy представляють ширину (width) основи кожного паралелепіпеда, тут - 0.1pt для кожного положення на площині xy.
  - Значення по осі z (dz) і буде нашою гістограмою (у змінній hist).

ДЯКУЮ ЗА УВАГУ!

Наступна тема: Функціональне програмування в мові Python

Анімації в matplotlib

- A framework for animation is added to the standard matplotlib from version 1.1 and its main class is matplotlib.animation.Animation.
  - This class is a base class; it is to be subclassed for specific behavior as is the case with the already provided classes: TimedAnimation, ArtistAnimation, and FuncAnimation

Class name (parent class)	Description
`Animation(object)`	This class wraps the creation of an animation using matplotlib. It is only a base class that should be subclassed to provide the needed behavior.
`TimedAnimation(Animation)`	This animation subclass supports time-based animation, and drawing a new frame every interval* milliseconds.
`ArtistAnimation(TimedAnimation)`	Before calling this function, all plotting should have taken place and the relevant artists saved.
`FuncAnimation(TimedAnimation)`	This makes an animation by repeatedly calling a function, passing in (optional) arguments.