Neural Networks: Python libraries

Andrzej Kordecki

Neural Networks for Classification and Identification (ML.EM05): Exercise 05
Division of Theory of Machines and Robots
Institute of Aeronautics and Applied Mechanics
Faculty of Power and Aeronautical Engineering
Warsaw University of Technology

Table of Contents

Reading and Writing Files

Reading and Writing Files

A file is a contiguous set of bytes used to store data and organized in a specific format. Example of 4 lines text format file "data.txt":

```
All the world's a stage,
and all the men and women merely players.
They have their exits and their entrances;
And one man in his time plays many parts.
```

Reading and Writing Files

File operation can start with the function open(filename, mode) used with two arguments:

- filename - string containing the filename,
- mode - options: 'r' when the file will only be read, 'w' for only writing, and 'a' opens the file for appending, 'r+' opens the file for both reading and writing.

Normally, files are opened in text mode, that means, you read and write strings from and to the file, which are encoded in a specific encoding. The default when reading is to convert line endings to just /n.

Reading and Writing Files

Example of reading a file:

```
f = open("data.txt", "r")

read_data = f.read() # Read whole file data
print(read_data) # Print the content of txt file

f.close()
```

The f.close() will close the file and immediately free up any system resources used by it.

Reading and Writing Files

Methods of File Objects:

- f.read() - read entire contents of the file
- f.readline(size) - reads one line of a file or reads at most size number of characters from the line. At the end of operation set position on character.
- readlines() - reads the remaining lines from the file object and returns them as a list.
- f.write(string) - writes the contents of string to the file.

Reading and Writing Files

Example of reading a file:

```
f = open("data.txt", "r")

print(f.readline(5)) Print "All t"
print(f.readlines()) Print ["he world's a stage, \n",

f = open("data.txt", "a") # append mode
f.write("\nThis is important\n") # We add at the end o
f.close()
```

Reading and Writing Files

You can use the remove() method to delete files:

```
# Check if the file exists
if os.path.exists("data.txt"):
    os.remove("data.txt")
    print("The file has been deleted")
else:
    print("The file does not exist")

# Prints "The file has been deleted"
```

Logger

Logger give you a means of tracking events that happen when program runs.

- We adds logging calls to code to indicate that certain events have occurred.
- An event is described by a descriptive message which can optionally contain variable data (e.g. error) or important information about program progress.
- The messages have different level of importance or severity of events.
- Not computationally demanding and easy access to logger file.

Standard Python library - do not need installation. Importing Logger library into Python code:

```
import logging
```

Logger

There are 5 standard levels indicating the severity of events, in order of increasing severity:

1. DEBUG
2. INFO
3. WARNING
4. ERROR
5. CRITICAL

Each has a corresponding method that can be used to log events at that level of severity.

Logger

The function basicConfig(**kwargs) allows to configure the logger module. Some of the commonly used parameters are as follow:

- level: The root logger will be set to the specified severity level.
- filemode: If filename is given, the file is opened in this mode. The mode specify method of writing information to file as 'a' - append or 'w' - overwrite.
- format: This is the format of the log message.

It should be noted that calling basicConfig() to configure the root logger works only if the root logger has not been configured before.

Logger

Recording logging events in a file:

```
logging.basicConfig(filename='program.log')
logging.debug('Debug message')
logging.info('Info message')
logging.warning('Warning message')
```

Result in 'program.log':

```
WARNING:root:Warning message
```

Notice that the debug and info messages didn't get logged.

Logger

We can chang the logging module logs the messages default a severity level of WARNING to severity level of DEBUG:

```
logging.basicConfig(filename='program.log',
                    level=logging.DEBUG,
                    filemode='w')
logging.debug('Debug message')
logging.info('Info message')
logging.warning('Warning message')
```

Result in 'program.log':

```
DEBUG:root:Debug message
INFO:root:Info message
WARNING:root:Warning message
```

Logger

We can change message output "format" by setting logger parameters in basicConfig():

- $\%(process)d$ - process number,
- $\%(levelname)s$ - severity of event,
- $\%(message)s$ - message to send,
- $\%(asctime)s$ - data of occurrence (day + hour) - it is possible to set own date format e.g. $datefmt =' \%d - \%b - \%y \%H : \%M : \%S'$ (12-Dec-19 14:45:01)

Logger

Example of custom message:

```
logging.basicConfig(filename='program.log',
    format='%(asctime)s-%(process)d-%(levelname)s
    -%(message)s')
logging.warning('Warning message')
```

Result in 'program.log':

```
2019-10-24 22:09:08,058-24852-WARNING-Warning message
```

Graphs, plots and Images

Matplotlib

Matplotlib is Python data visualization library.

- supports a very wide variety of graphs and plots: histogram, bar charts, power spectra, error charts etc.
- the module named pyplot allows to control features of the plot: line styles, font properties, formatting axes, etc.
- Matplotlib is used along with NumPy.

Importing library into Python code:

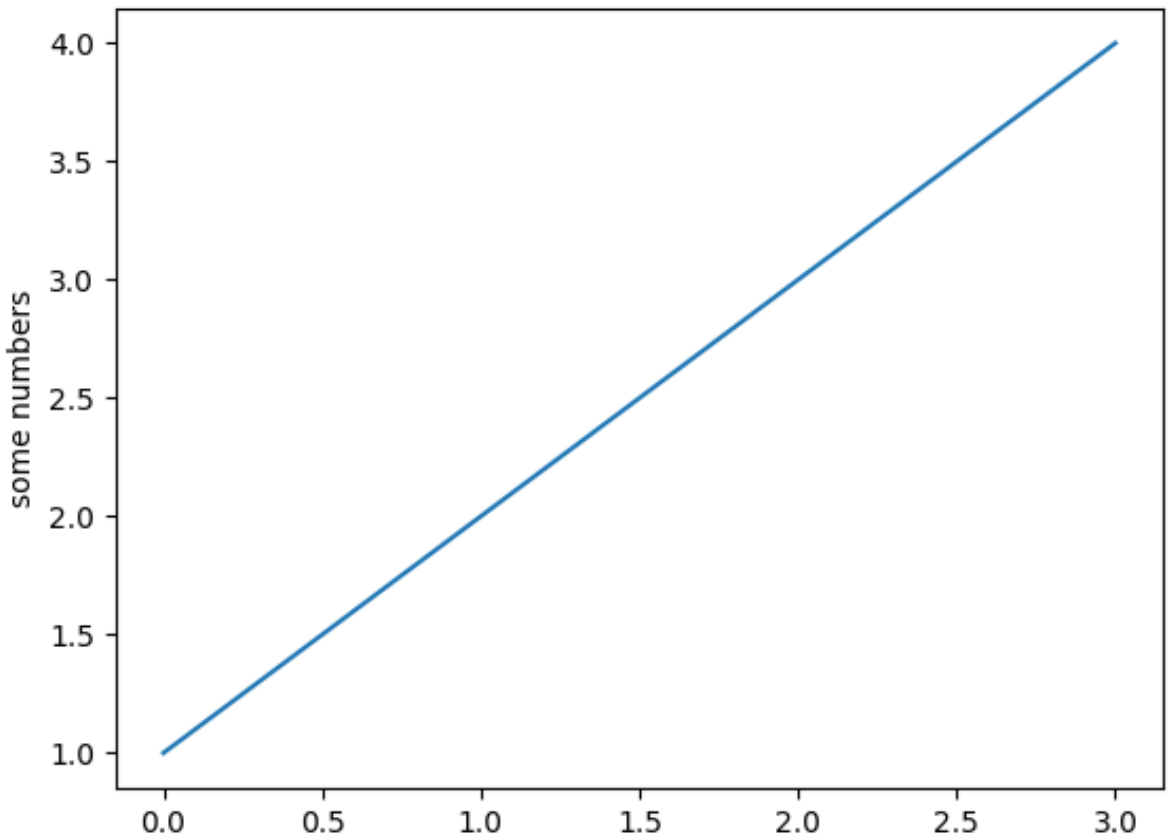
```
import numpy as np
import matplotlib.pyplot as plt
```

Plotting

Pyplot in Matplotlib is a collection of command style functions that make matplotlib work like MATLAB in showing data. The most important function in matplotlib is plot, which allows you to plot 2D data.

```
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

Plotting

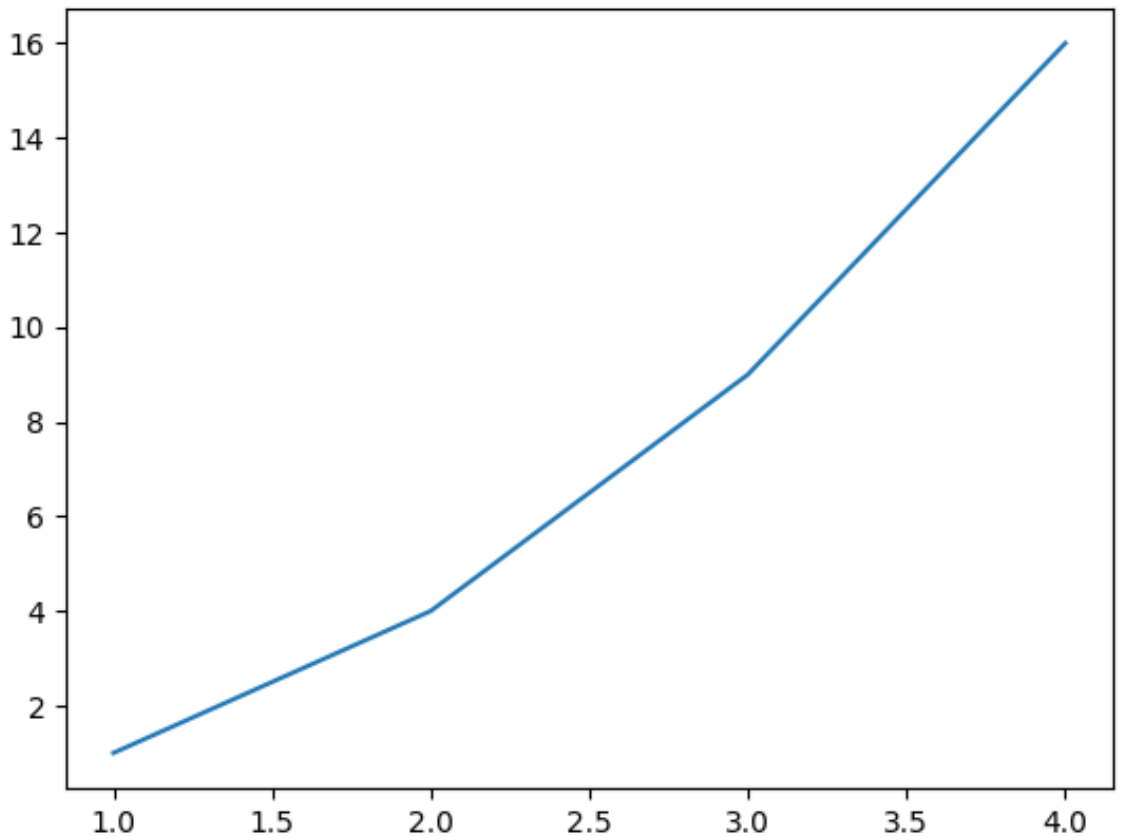


Plotting

Function plot() can take an arbitrary number of arguments, e.g. plot x versus y:

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.ylabel('some numbers')
plt.show()
```

Plotting

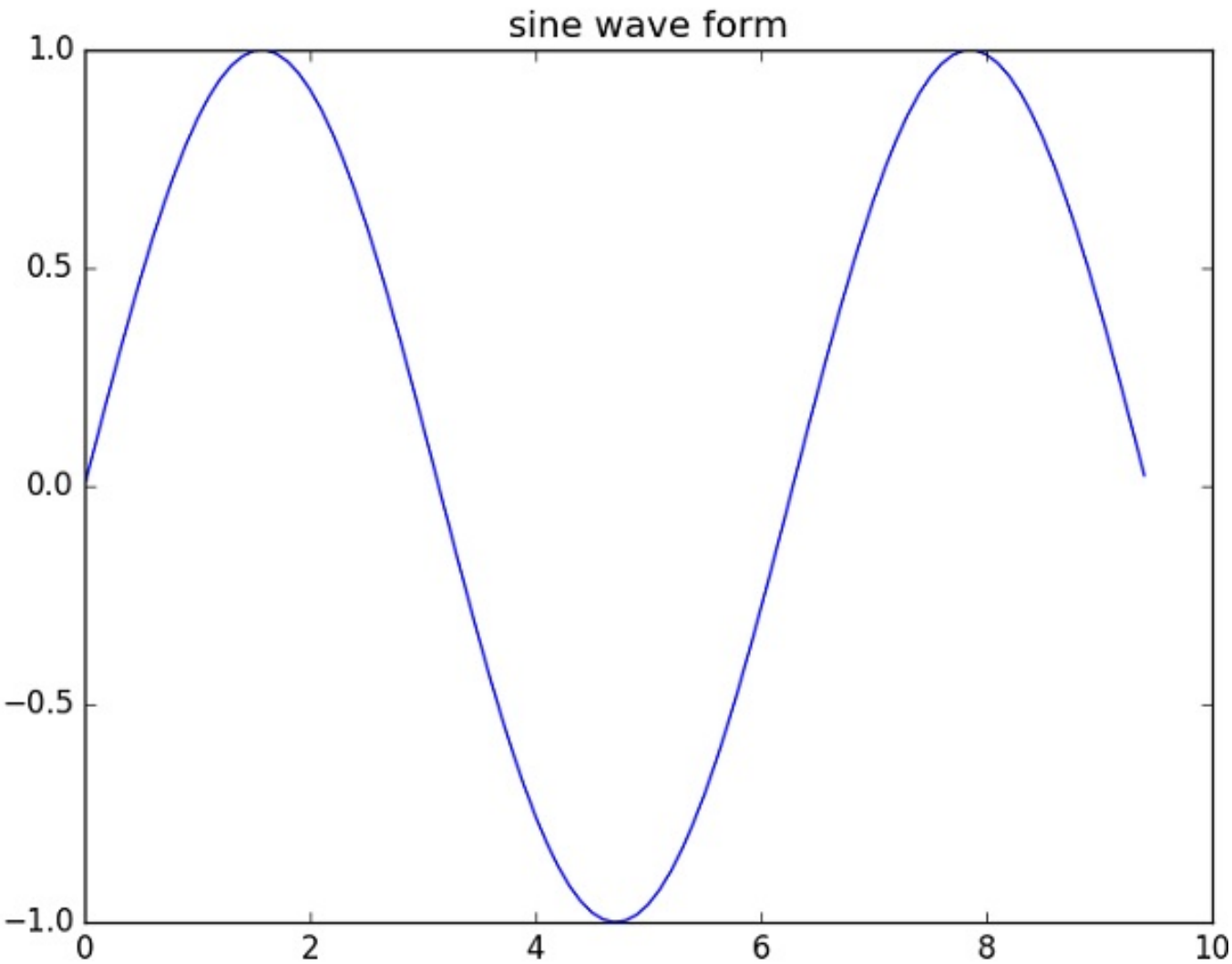


Plotting

Pyplot function allows to apply some changes to a figure: e.g., creates a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

```
# Compute the x and y coordinates for points on
# a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave form")

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```

Plotting

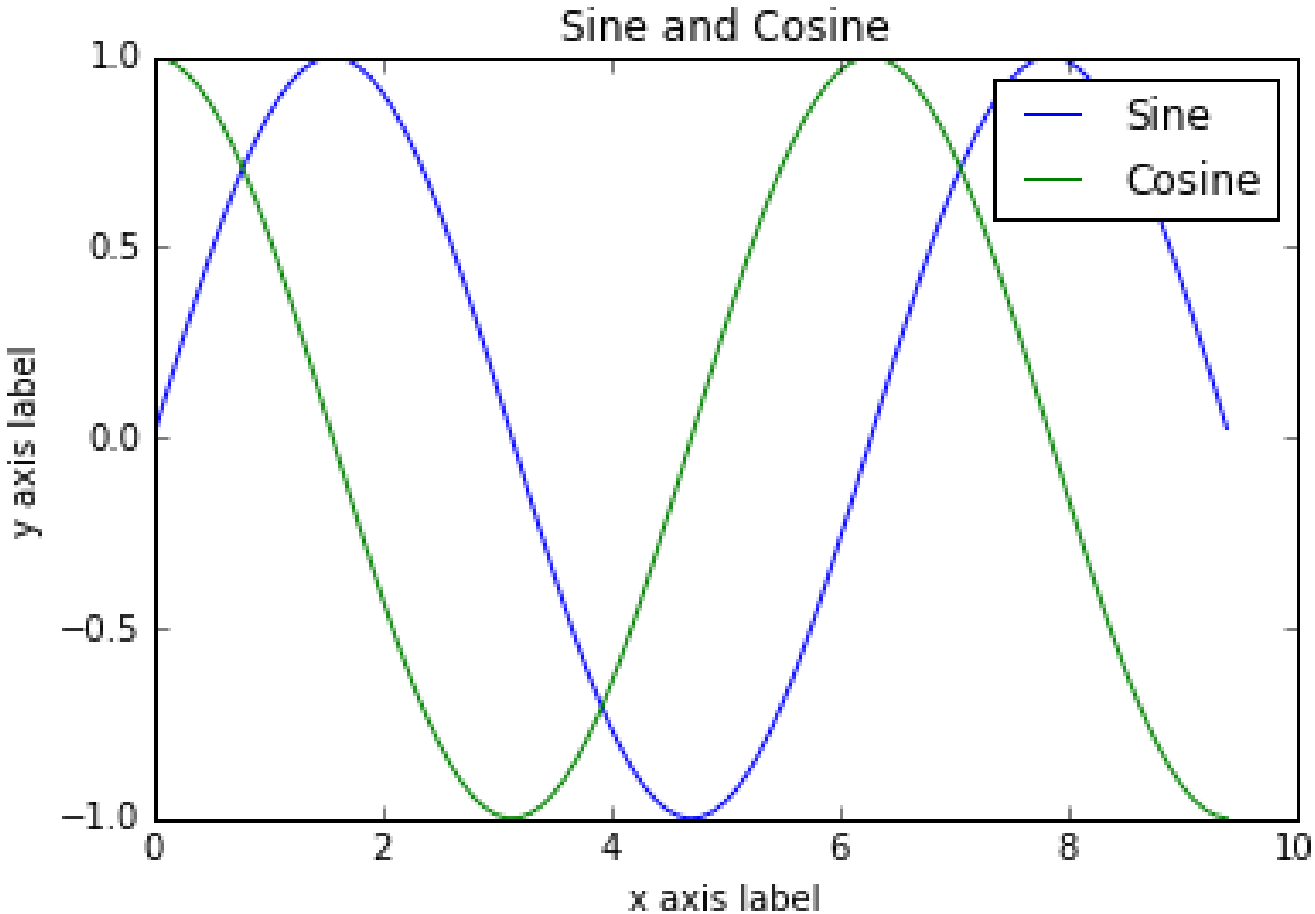


Plotting

We can easily plot multiple lines at once, and add a title, legend, axis labels:

```
# Compute the x and y coordinates for points on
# sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin), plt.plot(x, y_cos)
plt.xlabel('x axis label'), plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```

Plotting

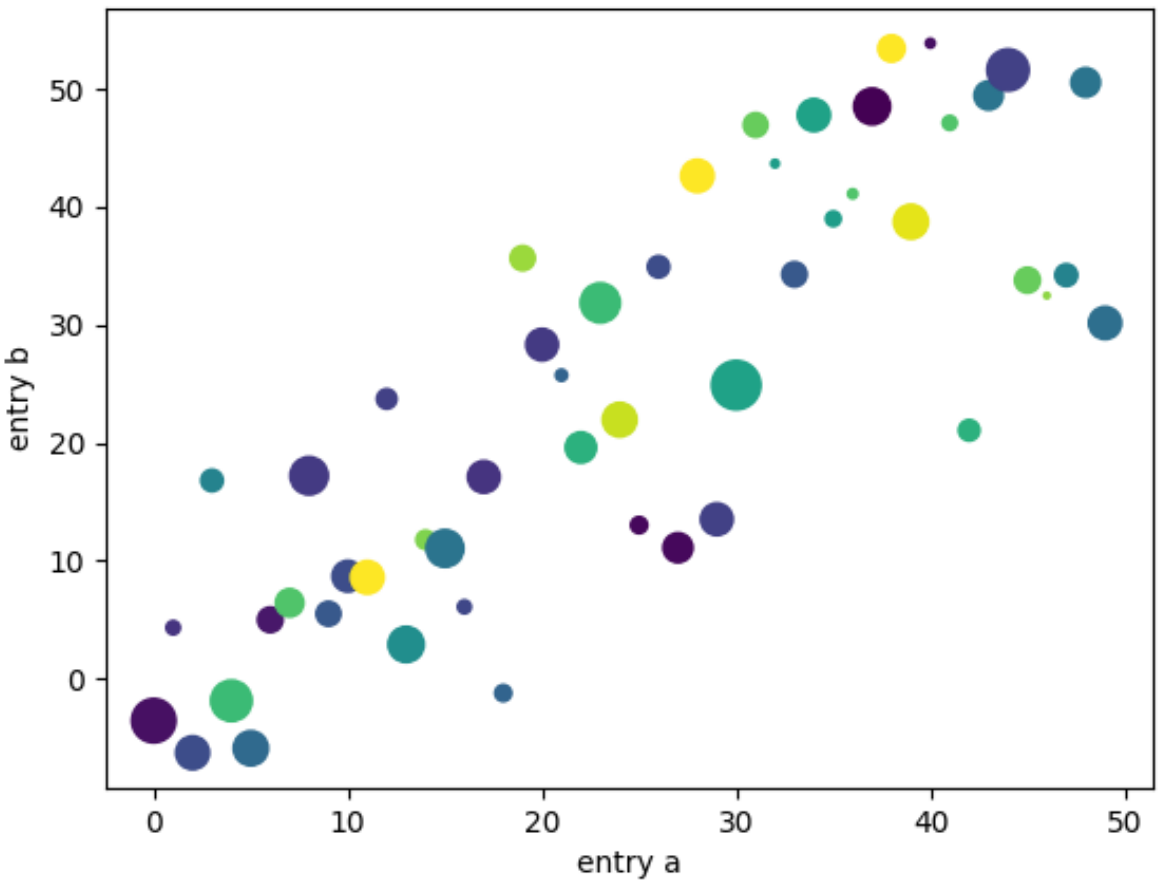

Sine and Cosine
Sine
Cosine
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
y axis label
x axis label

Plotting

Matplotlib may generate plots with the strings corresponding to object with the data keyword argument.

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```

Plotting

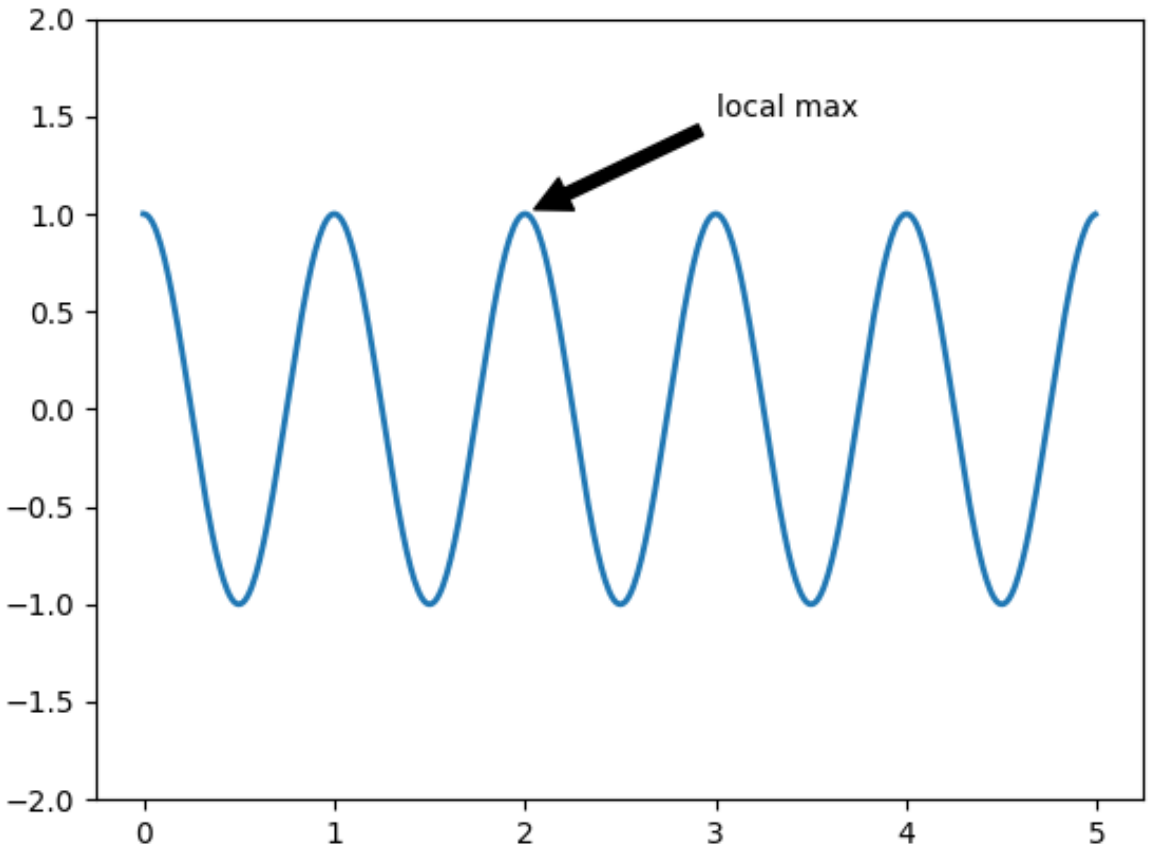


Plotting

Annotating text.

```
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black',
                             shrink=0.05))

plt.ylim(-2, 2)
plt.show()
```

Plotting



Subplots

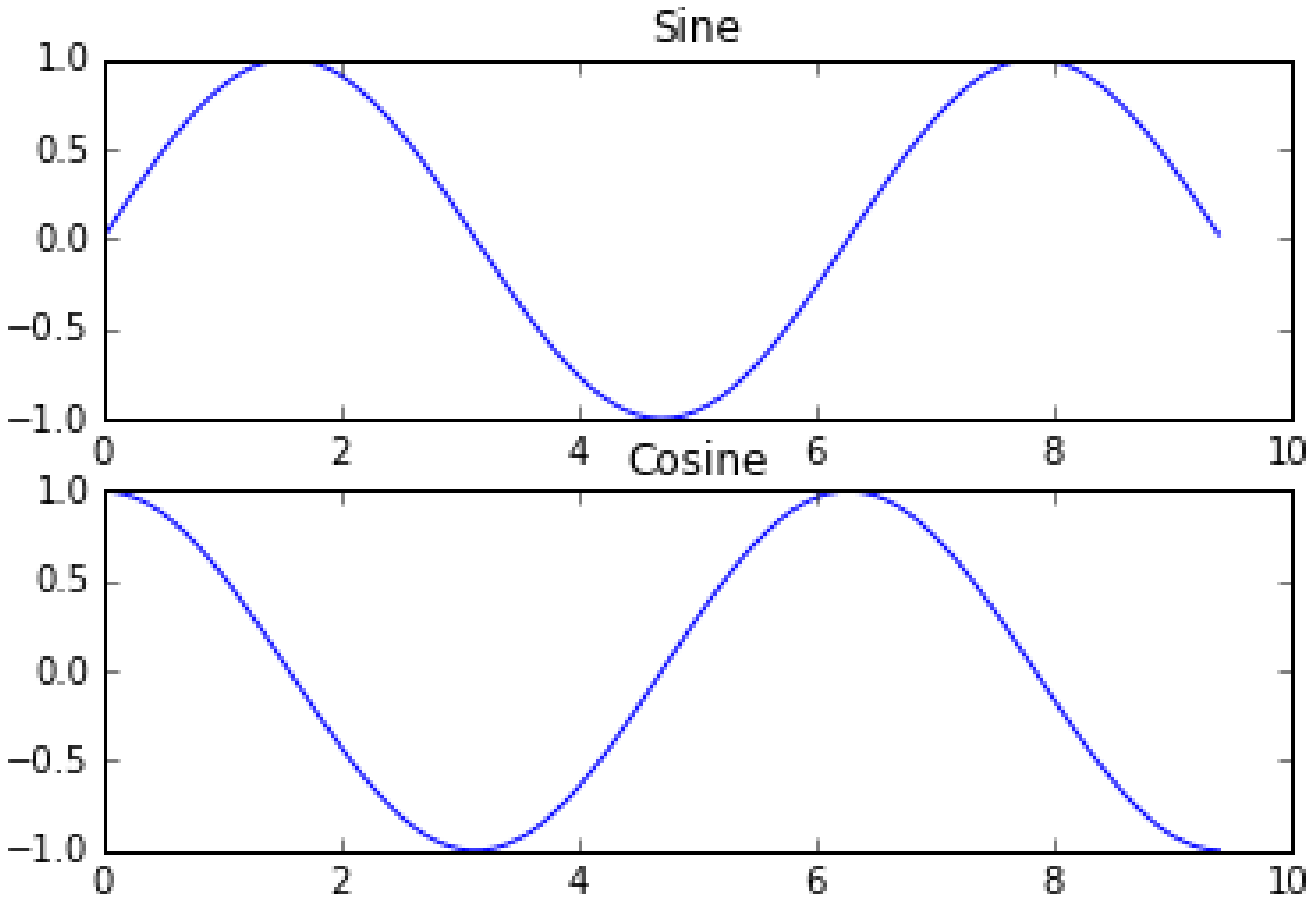
You can plot different things in the same figure using the subplot function.

```
# Compute the x and y coordinates
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
# Set up a subplot grid: height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')
```

Subplots

```
# Set the second subplot as active
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

Plotting

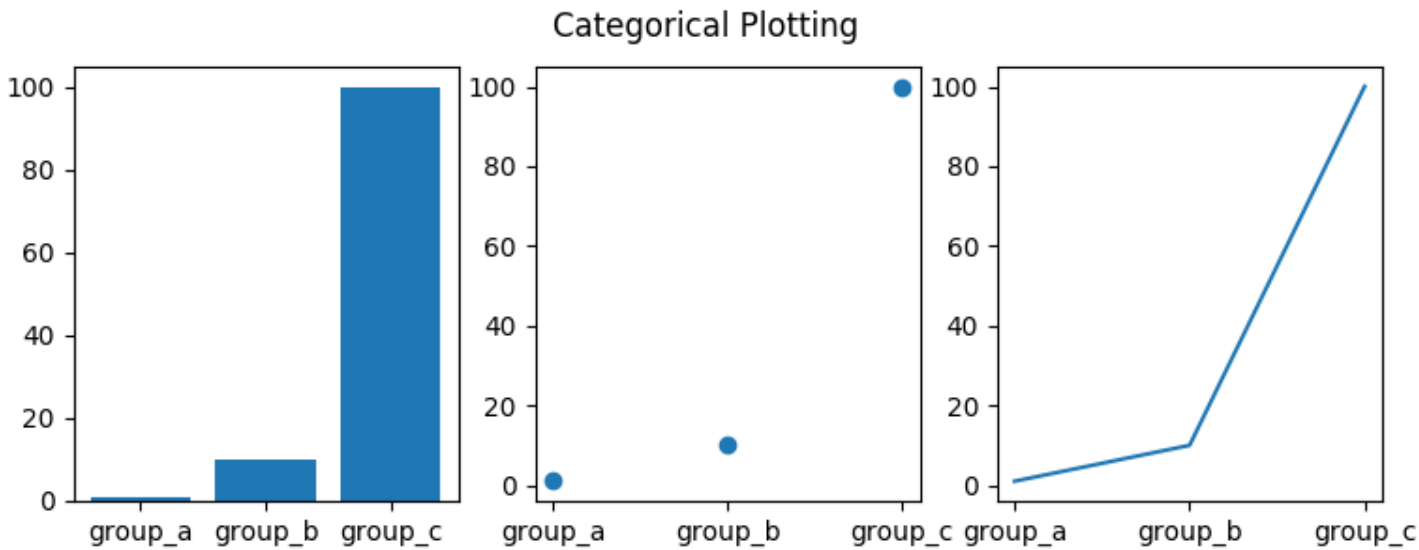


Subplots

It is also possible to create a plot using categorical variables.

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(1, figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```

Plotting



Images

Pillow module supports many image formats, including: BMP, JPEG, TIFF, GIF, PNG, JPEG 2000, EPS, ICNS, ICO, IM, MSP, PCX, PPM, WebP and more. To load an image from a file, we use:

```
from PIL import Image
image = Image.open('dog.jpg')
```

Saving image to file:

```
image.save('new_dog.png')
# extension defines image format or
image.save('new_dog.png', 'PNG')
```

Displaying the image:

```
image.show()  % open default Windows image browser
```

Images

Pillow module supports many image formats, including: BMP, JPEG, TIFF, GIF, PNG, JPEG 2000, EPS, ICNS, ICO, IM, MSP, PCX, PPM, WebP and more. To load an image from a file, we use:

```
from PIL import Image
image = Image.open('dog.jpg')
```

Saving image to file:

```
image.save('new_dog.png')
# extension defines image format or
image.save('new_dog.png', 'PNG')
```

Displaying the image:

```
image.show()  % open default Windows image browser
```

Images

Images

Reading image attributes:

```
# The file format of the source file.
print(image.format) # Prints: JPEG

# The pixel format used by the image.
print(image.mode) # Prints: RGB

# Image size, in pixels and 2-tuple format
# (width, height).
print(image.size) # Prints: (2520, 1445)

# Colour palette table.
print(image.palette) # Prints: None
```

Images

To resize an image, we use resize() method.

```
image = Image.open('dog.jpg')
new_image = image.resize((400, 400))
new_image.save('dog_400.jpg')

print(image.size) # Print: (2520, 1445)
print(new_image.size) # Print: (400, 400)
```

- The function doesn't modify the used image, it instead returns another image with the new dimensions.
- The function can change image's Aspect Ratio, so you might end up with an image that either looks stretched or squished.

Images

Images

Resizing images with unchanged aspect ratios with use of the thumbnail() function:

```
image.thumbnail((3000, 200))
image.save('dog_thumb.jpg')
print(image.size) # Print: (348, 200)
```

Difference between the resize() and thumbnail() functions:

- resize() can increase image size,
- thumbnail() cannot increase image size.

Images

Images

Cropping a rectangular region inside the image:

```
image = Image.open('dog.jpg')
box = (300, 300, 1000, 800)
cropped_image = image.crop(box)
cropped_image.save('dog_crop.jpg')
```

Properties:

- The method takes a box tuple that defines the position and size of cropped region and returns an Image object representing the cropped image.
- The coordinates for the box are (left, upper, right, lower). The Python Imaging Library uses a coordinate system that starts with (0, 0) in the upper left corner.

Images

Images

Pillow enables you to paste an image onto another one:

```
image = Image.open('dog.jpg')
logo = Image.open('logo_PW.jpg')
image_copy = image.copy()
position = ((image_copy.width - logo.width),
            (image_copy.height - logo.height))
image_copy.paste(logo, position)
image_copy.save('dog_PW.jpg')
```

We want to paste the logo image onto the copied image in the bottom right corner.

Images


POLITECHNIKA WARSZAWSKA
WARSAW UNIVERSITY OF TECHNOLOGY

Images

You can rotate images with Pillow using the rotate() method.

```
image = Image.open('dog.jpg')
image.rotate(90).save('dog_r90.jpg')
image.rotate(45, expand=True).save('dog_r45e.jpg')
```

By default, the rotated image keeps the dimensions of the original image without expand parameter. This means that for angles other than multiples of 180, the image will be cut and/or padded to fit the original dimensions.

Images

Images

Images

You can also flip images to get their mirror version:

```
image = Image.open('dog.jpg')
image_flip = image.transpose(Image.FLIP_LEFT_RIGHT)
image_flip.save('dog_flip.jpg')
```

Images

Images

The Pillow library enables you to convert images between different color space.

```
image = Image.open('dog.jpg')

# supports conversions mode: \RGB", CMYK, \L", \1".
greyscale_image = image.convert('L')
greyscale_image.save('dog_gray.jpg')
```

Images

Images

Access and change of the data stored in the pixels of an image.

```
image = Image.open('dog.jpg')
pixelMap = image.load() #create the pixel map

# Get Pixel value
pixel = pixelMap[0,0] #get the first pixel's value
# Set Pixel value
pixelMap[i, j] = (255, 255, 255)

# Set all pixeles value grayscale image
image = image.convert('L').point(lambda i: 255 - i)
```

The pixel data can then be retrieved by indexing the pixel map as an array.