Python 02

Elias Wachmann

2024

Content

Numpy

Numpy

Numpy is a python module that provides functions to work with arrays.

- Arrays are a data structure that can store multiple values of the same type.
- Arrays are indexed starting from 0.
- Arrays can be multidimensional.

Numpy examples

Quickstart Guide to Numpy

```
1 import numpy as np
2 # Create a 2D array of size 3x4
3 a = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
      [9, 10, 11, 12]])
4 print(a.shape)  # (3, 4)
5 print(a.ndim)  # 2
6 print(a.size)  # 12
7 print(a.dtype)  # int64
```

Numpy examples – Basics

The shape attribute of an array returns the dimensions of the array. In our example the array has 3 rows and 4 columns.

```
1 a = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
    [9, 10, 11, 12]])
2 print(a.shape)  # (3, 4)
```

ndim specifies the number of dimensions of the array.

```
1 a = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
    [9, 10, 11, 12]])
2 print(a.ndim)  # 2
```

Numpy examples – Basics

size returns the total number of elements in the array.

```
1 a = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
      [9, 10, 11, 12]])
2 print(a.size)  # 12
```

Numpy examples – Zero & One arrays

zeros and ones can be used to create arrays filled with zeros or ones.

```
1 my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2 new_array = np.array(my_list)
3 # 3 rows, 4 columns
4 all_zeros = np.zeros((3, 4))
5 # 3 rows, 1 column
6 all_ones = np.ones((3, 1))
```

Arange & Linspace

arange and linspace can be used to create arrays with evenly spaced values.

```
1 ints = np.arange(5, 42) # 5, 6, 7, ..., 41
2 linspace = np.linspace(0,5,11) # 0, 0.5, 1,
     1.5, ..., 5
3 fixed_width = np.arange(0, 10, 0.5) # 0,
     0.5, 1, 1.5, ..., 9.5
```

Arange: given step size
Linspace: given number of elements.

Numpy examples – Reshaping

reshape can be used to change the shape of an array.

```
square_matrix = np.array([[1, 2], [3, 4]])
column = square_matrix.reshape(4, 1)
row = square_matrix.reshape(1, 4)
print(column)
# [[1]
#  [2]
#  [3]
#  [4]]
print(row)  # [[1, 2, 3, 4]]
```

Numpy examples – Ravel

ravel can be used to flatten an array.

```
square_matrix = np.array([[1, 2], [3, 4]])
print(square_matrix)
# [[1 2]
# [3 4]]
ravelled = square_matrix.ravel()
print(ravelled)  # [1 2 3 4]
```

flatten is another function that can be used to flatten an array.

Generate random numbers

True randomness is not easy

- Computers are deterministic machines.
- Pseudo-random number generators (PRNG) are algorithms that can generate numbers that appear random.
- PRNGs are initialized with a seed value.
- The same seed value will result in the same sequence of random numbers.

Generate a random number

rand can be used to generate a random number between 0 and 1.
uniform can be used to generate a random number between a given range.

```
1 rand_ = np.random.random()
2 from numpy import random as rng
3 rand_ = rng.random() # 0 <= rand_ < 1
4 rand_ = rng.uniform(5, 42) # 5 <= rand_ < 42
```

Matplotlib

Matplotlib - Basics

Matplotlib is a plotting library for the Python programming language. It provides various functions to create 2D, 3D plots and animations.
The `pyplot` module provides a MATLAB-like interface to the underlying plotting library.

Import the module with:

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

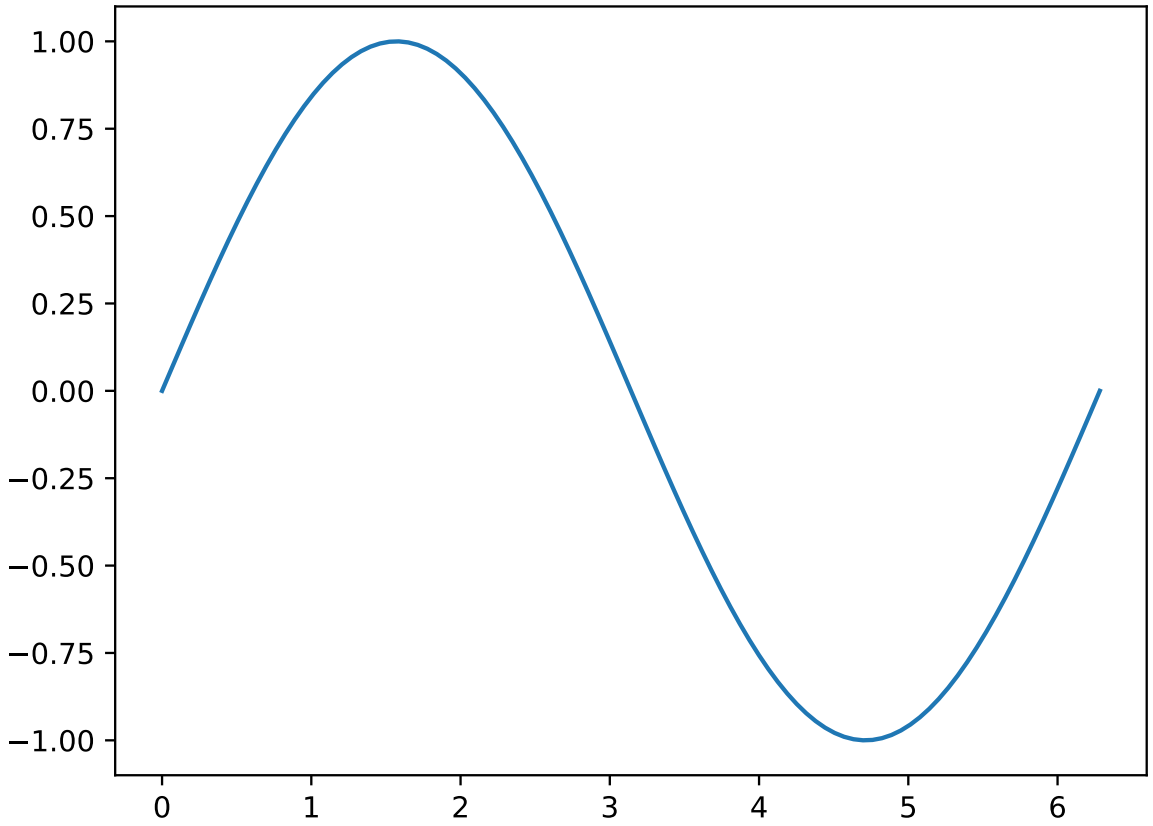
In the following examples, we will use the `pyplot` module (with alias `plt`) to create plots.

Matplotlib - Basic plotting

Let's create a basic plot of a sine function:

```
1 # Create a list of 100 evenly-spaced numbers
      over the range 0 to 100
2 x = np.linspace(0, 2 * np.pi, 100)
3 y = np.sin(x)  # Calculate sine for all x
4 plt.plot(x, y)  # Plot the sine for all x
5 plt.show()  # Display the plot
```

Matplotlib - Basic plotting output



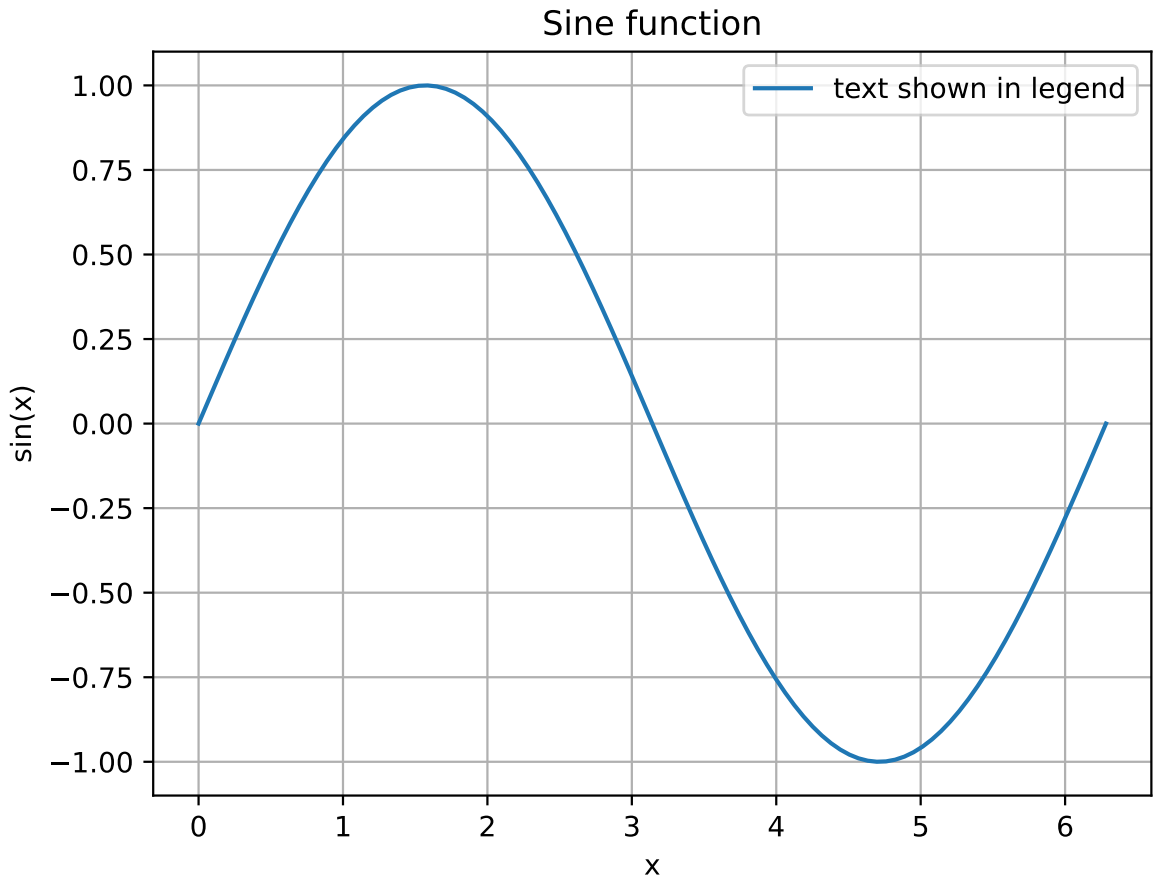
Matplotlib - Labels

We can add labels to the axes a legend and a title to the plot:

```
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)  # Calculate sine for all x
plt.plot(x, y, label="text shown in legend")
plt.xlabel("x")  # Label the x-axis
plt.ylabel("sin(x)")  # Label the y-axis
plt.title("Sine function")  # Add a title
plt.legend()  # Add a legend
plt.grid()  # Add a grid
plt.show()  # Display the plot
```

Documentation: `title()`, `xlabel()`, `ylabel()`, `legend()`

Matplotlib - Labels

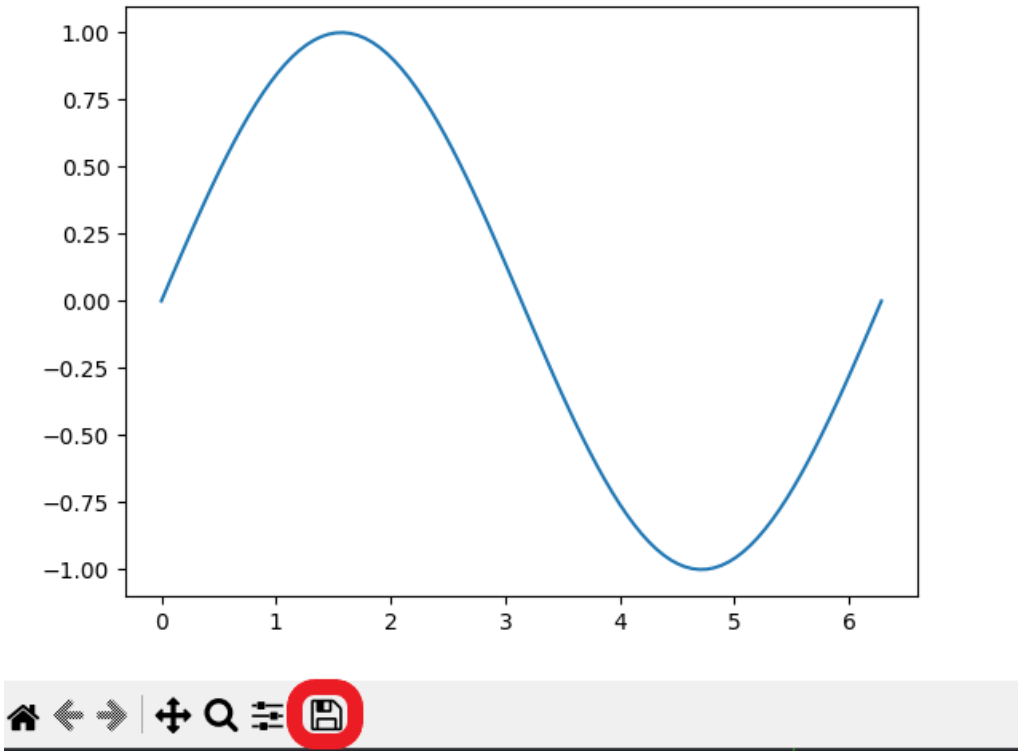


Matplotlib - Save figures

To save plotted figures, use the `savefig()` function:

```
1 plt.savefig("path/to/save/plot/to.pdf")
```

Or use the floppy disk symbols in the pop-up windows



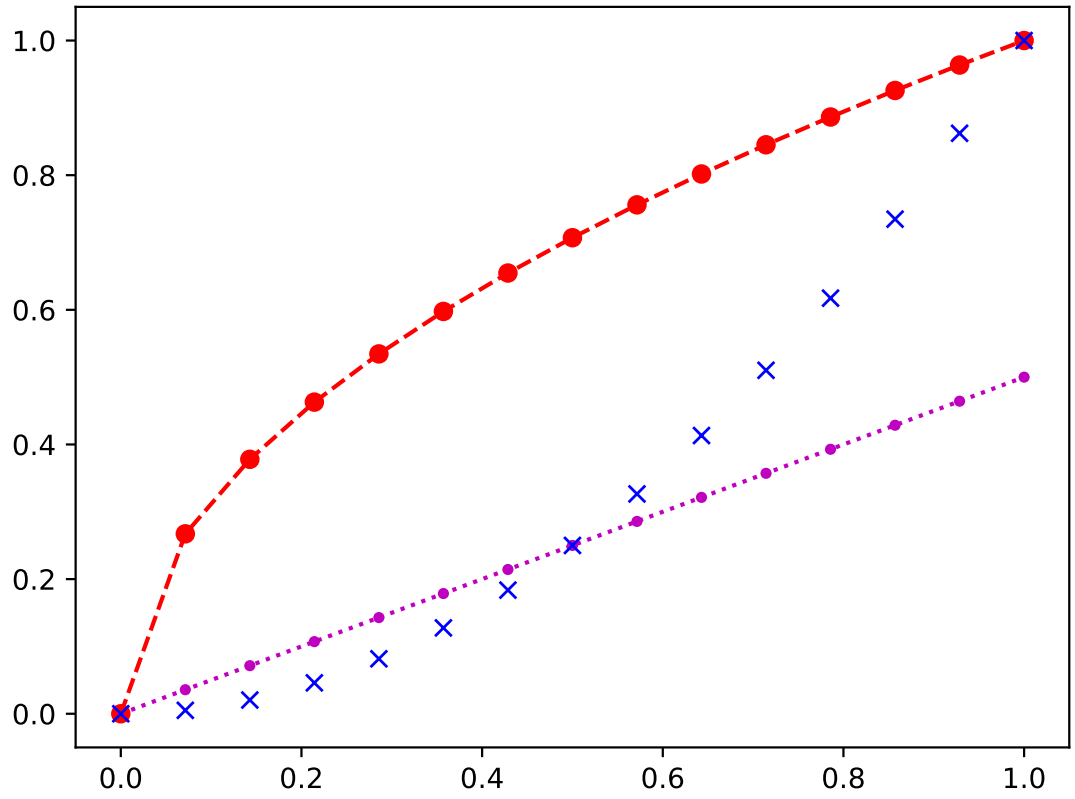
Matplotlib - Plotoptions

`plot()` function has many options to customize the plot. Format strings `fmt = '[marker][line][color]'` change the appearance of the lines:

```
1 x = np.linspace(0, 1, 15)
2 y1 = 0.5 * x
3 y2 = x**0.5
4 y3 = x**2
5 plt.plot(x, y1, ".:m")
6 plt.plot(x, y2, "o--r")
7 plt.plot(x, y3, "xb")
```

Matplotlib - Plotoptions

Using the plot options from the last slide we get:



Matplotlib - Markers/Linestyles/Colors

Some markers: `.` (point), `o` (circle), `+` (plus), `*` (star), `s` (square), `p` (pentagon), `h` (hexagon 1), `H` (hexagon 2)

Line styles: `-` (solid line), `--` (dashed line), `-.` (dash-dot line), `:` (dotted line)

Colors: `b` (blue), `g` (green), `r` (red), `c` (cyan), `m` (magenta), `y` (yellow), `k` (black), `w` (white)

File I/O

Reading and writing files

loadtxt can be used to read data from a text file.

```
text = np.loadtxt('example_text.txt')
print(text) # prints the contents of the
    file
newtext = "Hello world!"
np.savetxt('example_text.txt', newtext)
text = np.loadtxt('example_text.txt')
print(text) # "Hello world!"
```

Reading csv files – input

simple_example.csv U

slides24 > 02 > examples >

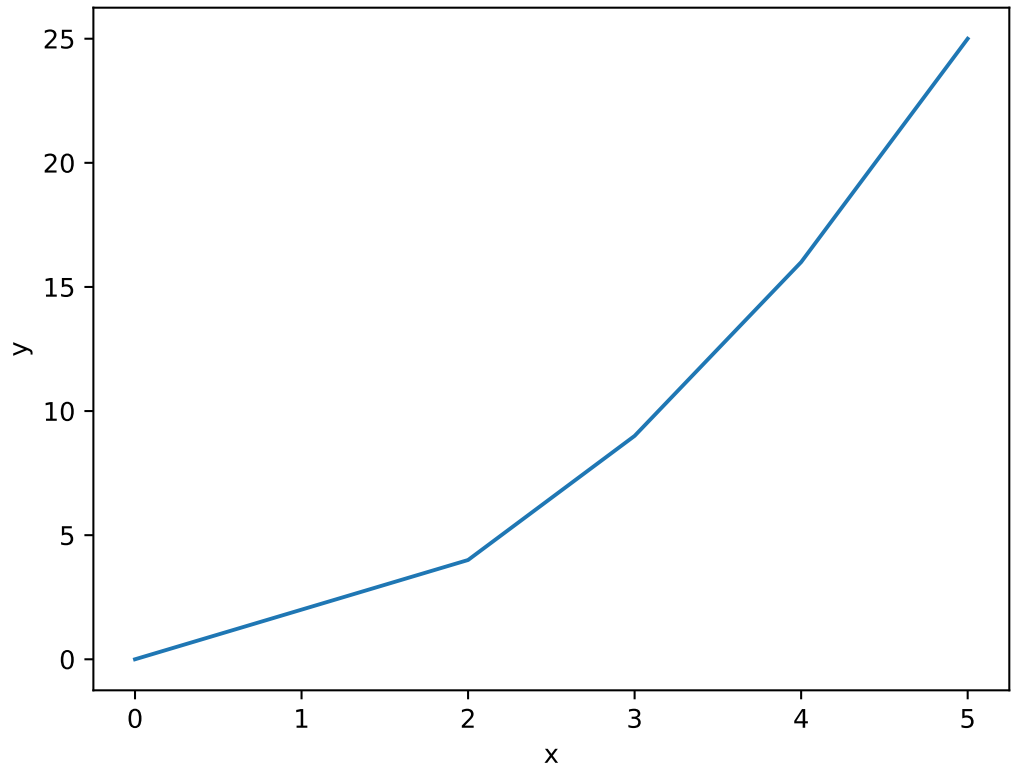
```
1   x;y
2   0;0
3   1;2
4   2;4
5   3;9
6   4;16
7   5;25
```

Reading csv files

genfromtxt can be used to read data from a csv file.

```
1 data = np.genfromtxt('slides24/02/examples/
    simple_example.csv', delimiter=';')
2 labels = np.genfromtxt('slides24/02/examples
    /simple_example.csv', delimiter=';',
    max_rows=1, dtype=str)
3 print(labels)
4 print(data)
5 plt.plot(data[:,0], data[:,1])
6 plt.xlabel(labels[0])
7 plt.ylabel(labels[1])
8 plt.show()
```

Reading csv files – output



Numpy (extra)

Ravel vs.Flatten

ravel returns a view of the original array. flatten returns a **copy** of the original array.

```
square_matrix = np.array([[1, 2], [3, 4]])
flattened = square_matrix.flatten()
ravelled = square_matrix.ravel()
square_matrix[0, 0] = 5
print(flattened)  # [1 2 3 4]
print(ravelled)   # [5 2 3 4]
```

Numpy examples – all & any

all and any can be used to check if all or any elements of an array comply with a condition.

```
1 x = np.array([1, 2, 3])
2 print(np.all(x != 0))   # True
3 print(np.any(x == 2))   # True
```