

Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny
Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia
Rok akademicki 2024/2025

Wykład nr 7 (27.05.2025)

dr inż. Jarosław Forenc

Plan wykładu nr 7

■ Macierze w Pythonie

- implementacja w postaci listy list
- odwołania do elementów, wybrane operacje

■ Biblioteka NumPy

- instalacja, tworzenie tablic
- odwołania do elementów, operacje i funkcje arytmetyczne

■ Biblioteka Matplotlib

- instalacja, wykresy liniowe
- opisywanie wykresów, sposób rysowania linii
- wykresy punktowe, słupkowe, histogramy, kołowe

■ Biblioteka SciPy

- instalacja, moduły

Python - macierze

- do zaimplementowania macierzy można użyć **list zagnieżdżonych** lub metod z modułu **NumPy**
- korzystanie z biblioteki **NumPy** zalecane jest w przypadku bardziej zaawansowanych operacji algebraicznych na macierzach
- w przypadku **zagnieżdżania list**, każdy wiersz macierzy jest listą, a cała macierz jest listą list

```
macierz = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]  
  
print("Macierz:")  
for wiersz in macierz:  
    print(wiersz)
```

```
Macierz:  
[1, 2, 3]  
[4, 5, 6]  
[7, 8, 9]
```

Python - macierze (odwołania do elementów)

- dostęp do elementu znajdującego się na pozycji (wiersz, kolumna), wymaga zastosowania indeksowania dwukrotnego `[wiersz][kolumna]`

```
macierz = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]  
element = macierz[1][2]  
print("Element:", element)
```

Element: 6

- można również użyć pętli zagnieżdżonych, aby przejść przez wszystkie elementy macierzy

```
for wiersz in macierz:  
    for element in wiersz:  
        print(element)
```

1
2
...

Python - macierze (wybrane operacje)

```
import random

# Generowanie i wyświetlanie macierzy o rozmiarze 3x4
macierz = []
for _ in range(3):
    wiersz = [random.randint(0, 99) for _ in range(4)]
    macierz.append(wiersz)

print("Macierz:")
for wiersz in macierz:
    print(wiersz)
```

```
Macierz:
[86, 5, 15, 74]
[37, 19, 78, 27]
[47, 22, 8, 92]
```

Python - macierze (wybrane operacje)

```
# Obliczanie sumy elementów macierzy
suma = 0
for wiersz in macierz:
    suma += sum(wiersz)
print("Suma elementów macierzy:", suma)

# Obliczanie średniej arytmetycznej elementów macierzy
ilosc_elementow = 0
suma = 0
for wiersz in macierz:
    ilosc_elementow += len(wiersz)
    suma += sum(wiersz)
srednia = suma / ilosc_elementow
print("Średnia arytmetyczna elementów macierzy:", srednia)
```

```
Suma elementów macierzy: 510
Średnia arytmetyczna elementów macierzy: 42.5
```

Python - macierze (wybrane operacje)

```
# Znajdowanie wartości min i max w macierzy
min_wartosc = float('inf')
max_wartosc = float('-inf')
for wiersz in macierz:
    min_wartosc = min(min_wartosc, min(wiersz))
    max_wartosc = max(max_wartosc, max(wiersz))
print("Wartość najmniejsza w macierzy:", min_wartosc)
print("Wartość największa w macierzy:", max_wartosc)
```

```
Wartość najmniejsza w macierzy: 5
Wartość największa w macierzy: 82
```

- ❑ `min(wiersz)` - zwraca najmniejszą wartość w danym wierszu
- ❑ `min(min_wartosc, min(wiersz))` - wybiera mniejszą wartość z dotychczasowego minimum i minimum z bieżącego wiersza

Python - macierze (wybrane operacje)

```
# Ilość liczb parzystych i nieparzystych w macierzy
```

```
parzyste = 0
```

```
nieparzyste = 0
```

```
for wiersz in macierz:
```

```
    for liczba in wiersz:
```

```
        if liczba % 2 == 0:
```

```
            parzyste += 1
```

```
        else:
```

```
            nieparzyste += 1
```

```
print("Liczby parzyste:", parzyste)
```

```
print("Liczby nieparzyste:", nieparzyste)
```

Macierz:

[86, 5, 15, 74]

[37, 19, 78, 27]

[47, 22, 8, 92]

Liczby parzyste: 6

Liczby nieparzyste: 6

NumPy - podstawowe informacje



- **NumPy** (Numerical Python) - wydajna biblioteka numeryczna umożliwiająca tworzenie obiektów reprezentujących wielowymiarowe tablice oraz dostarczająca funkcji do wykonywania operacji na tych tablicach
- strona internetowa: <https://numpy.org/> (EN)
- **NumPy** umożliwia wykonywanie obliczeń tylko na CPU (do obliczeń na GPU konieczne jest zastosowanie innych bibliotek, np. CuPy)
- API **NumPy** jest stosowane w innych pakietach: Matplotlib, SciPy, Pandas
- cechy charakterystyczne tablic w **NumPy**:
 - tablice są stałej wielkości - nie można zmieniać ich rozmiaru po utworzeniu
 - tablice przechowują elementy tego samego typu, np. liczby całkowite, liczby zmiennoprzecinkowe
- tablice **NumPy** są szybsze niż listy w Pythonie i zużywają mniej pamięci do przechowywania danych

NumPy - instalacja

- w edytorze Visual Studio Code wybieramy w menu głównym:
Terminal → Nowy terminal
- wpisanie: **pip install numpy** spowoduje ściągnięcie i instalację **NumPy**

```
PS C:\Users\jaros> pip install numpy
Collecting numpy
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
    -----
    61.0/61.0 kB 466.1 kB/s eta 0:00:00
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
    -----
    15.5/15.5 MB 29.7 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.26.4
```

- aktualizacja biblioteki wymaga wpisania: **pip install --upgrade numpy**

NumPy - tworzenie tablic, funkcja `array()`

- zastosowanie biblioteki **NumPy** w programie wymaga jej importowania

```
import numpy as np
```

- **np** - skrócona nazwa biblioteki, zalecane jest stosowanie tej nazwy
- tworzenie tablicy z zastosowaniem funkcji **`array()`** i listy jednowymiarowej

```
arr = np.array([1, 2, 3, 4, 5])  
print(arr)
```

```
[1 2 3 4 5]
```

```
arr = np.array([2.5127, -4, 3.6])  
print(arr)
```

```
[ 2.5127 -4.          3.6      ]
```

NumPy - tworzenie tablic, rank, shape, axes

- tworzenie tablicy z zastosowaniem funkcji `array()` i listy wielowymiarowej

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
print(arr)
```

```
[[1 2 3 4]  
 [5 6 7 8]]
```

- pojęcia stosowane w odniesieniu do tablic (rozmiary tablic):
 - rząd tablicy (`rank`) - liczba wymiarów tablicy, czyli liczba `osi` (axes)
 - do sprawdzenia rzędu służy `ndim`

```
print(arr.ndim)
```

```
2
```

NumPy - tworzenie tablic, rank, shape, axes

- tworzenie tablicy z zastosowaniem funkcji `array()` i listy wielowymiarowej

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
print(arr)
```

```
[[1 2 3 4]  
 [5 6 7 8]]
```

- pojęcia stosowane w odniesieniu do tablic (rozmiary tablic):
 - **kształt** tablicy (**shape**) - rozmiar tablicy w każdym wymiarze, określa ile elementów jest wzdłuż każdej osi
 - reprezentowany jest przez krotkę liczb całkowitych

```
print(arr.shape)
```

```
(2, 4)
```

NumPy - tworzenie tablic, rank, shape, axes

- tworzenie tablicy z zastosowaniem funkcji `array()` i listy wielowymiarowej

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])  
print(arr)
```

```
[[1 2 3 4]  
 [5 6 7 8]]
```

- pojęcia stosowane w odniesieniu do tablic (rozmiary tablic):
 - **osie** tablicy (**axes**) - kierunki w wielowymiarowej tablicy, wzdłuż których wykonywane są operacje
 - **axis=0**: wzdłuż wierszy (czyli kolumny)
 - **axis=1**: wzdłuż kolumn (czyli wiersze)

NumPy - rank, shape, axes

```
import numpy as np

arr1D = np.array([1,2,3,4,5])
arr2D = np.array([[1,2,3],[4,5,6]])
print(arr1D)
print(f"arr1D: rank = {arr1D.ndim}, shape = {arr1D.shape}")
print(arr2D)
print(f"arr2D: rank = {arr2D.ndim}, shape = {arr2D.shape}")
print(f"suma kolumn (axis=0) = {arr2D.sum(axis=0)}")
print(f"suma wierszy (axis=1) = {arr2D.sum(axis=1)}")
```

```
[1 2 3 4 5]
arr1D: rank = 1, shape = (5,)
[[1 2 3]
 [4 5 6]]
arr2D: rank = 2, shape = (2, 3)
suma kolumn (axis=0) = [5 7 9]
suma wierszy (axis=1) = [ 6 15]
```

NumPy - tworzenie tablic o stałej wartości

- `np.zeros()` - tworzy tablicę, w której wszystkie elementy wynoszą 0 (zero)

```
zeros(shape, dtype=float, order='C')
```

- `shape` - kształt nowej tablicy, jako liczba lub krotka liczb
- `dtype` - opcjonalny typ danych elementów tablicy (domyślnie `float`)
- `order` - opcjonalny sposób rozmieszczenia elementów w pamięci
(`C` - wierszami, `F` - kolumnami, jak w Fortranie)

```
arr = np.zeros((3,6))  
print(arr)
```

```
[[0.  0.  0.  0.  0.  0.]  
 [0.  0.  0.  0.  0.  0.]  
 [0.  0.  0.  0.  0.  0.]
```

NumPy - tworzenie tablic o stałej wartości

- `np.ones()` - tworzy tablicę, w której wszystkie elementy wynoszą **1** (jeden)

```
arr = np.ones((2,4))  
print(arr)
```

```
[[1.  1.  1.  1.]  
 [1.  1.  1.  1.]
```

- `np.empty()` - tworzy tablicę bez inicjalizowania jej elementów (mogą to być wartości przypadkowe)

```
arr = np.empty((2,4))  
print(arr)
```

```
[[0.00e+000  0.00e+000  0.00e+000  0.00e+000]  
 [0.00e+000  1.78e-321  0.00e+000  0.00e+000]]
```

NumPy - tworzenie tablic o stałej wartości

- `np.arange()` - generuje tablicę z wartościami w określonym zakresie, podobnie jak funkcja `range(start, stop, step)`

```
print(np.arange(5))  
print(np.arange(5,9))  
print(np.arange(1,10,2))
```

```
[0 1 2 3 4]  
[5 6 7 8]  
[1 3 5 7 9]
```

- `np.linspace(start, stop, num=50)` - generuje tablicę z wartościami równomiernie rozmieszczonymi w określonym przedziale

```
print(np.linspace(1, 10, num=6))
```

```
[ 1.    2.8  4.6  6.4  8.2 10. ]
```

NumPy - tworzenie tablic z wartościami losowymi

- do generowania liczb losowych stosowany jest moduł `np.random`
- `np.random.rand()` lub `np.random.random()` - generuje losowe liczby z rozkładu jednostajnego z przedziału `[0, 1)`
- `np.random.randint()` - generuje losowe liczby całkowite z podanego przedziału, włączając dolną wartość, ale wyłączając górną
- `np.random.seed()` - ustala wartość początkową dla generatora, umożliwiając otrzymanie powtarzalnych wyników

```
print(np.random.random(3))  
print(np.random.rand(2,3))  
print(np.random.randint(2,9,10))
```

```
[0.12618219 0.01468085 0.40756073]  
[[0.53353022 0.68664792 0.69271609]  
 [0.50939703 0.78763989 0.76998518]]  
[8 7 4 2 4 2 8 2 6 5]
```

NumPy - łączenie tablic

- `np.concatenate()` - służy do łączenia dwóch lub więcej tablic wzdłuż określonej osi

```
arr1 = np.array([1,2,3,4,5])
arr2 = np.array([6,7,8,9,0])
arr = np.concatenate((arr1,arr2))
print(arr)
```

```
[1 2 3 4 5 6 7 8 9 0]
```

```
arr1 = np.array([[1,2],[3,4]])
arr2 = np.array([[5,6]])
arr = np.concatenate((arr1,arr2),axis=0)
print(arr)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

NumPy - odwołania do elementów tablic

- odwołania do elementów tablic jednowymiarowych wyglądają tak samo jak odwołania do elementów list, indeksy rozpoczynają się od 0

```
arr = np.array([1,2,3,4,5])  
print(arr[0])  
print(arr[3])  
print(arr[-1])
```

1
4
5

- w przypadku tablic dwuwymiarowych używa się dwóch indeksów (numer wiersza i numer kolumny)

```
arr = np.array([[1,2,3,4],[5,6,7,8]])  
print(arr)  
print(arr[0,0])  
print(arr[0,2])  
print(arr[1,3])
```

```
[[1 2 3 4]  
 [5 6 7 8]]  
1  
3  
8
```

NumPy - odwołania do elementów tablic

- można stosować **wycinki** (slices) w indeksach

nazwa [indeks_początkowy : indeks_końcowy : krok]

```
arr = np.array([1,2,3,4,5])  
print(arr[0:2])  
print(arr[1:])  
print(arr[-2:])
```

```
[1 2]  
[2 3 4 5]  
[4 5]
```

```
arr = np.array([[1,2,3,4],[5,6,7,8]])  
print(arr)  
print(arr[:,0:3])  
print(arr[0,2:])
```

```
[[1 2 3 4]  
 [5 6 7 8]]  
[[1 2 3]  
 [5 6 7]]  
[3 4]
```

NumPy - operacje arytmetyczne

- do wykonania podstawowych operacji arytmetycznych można zastosować operatory: $+$, $-$, $*$, $/$

```
a = np.array([1,2])  
b = np.array([3,4])  
print(a+b)  
print(a-b)  
print(a*b)  
print(a/b)
```

```
[4 6]  
[-2 -2]  
[3 8]  
[0.33333333 0.5      ]
```

- jeśli jednym z argumentów jest skalar, to operacja wykonywana jest na każdym elemencie (wykonywany jest tzw. broadcast)

```
arr = np.array([[1,2],[3,4]])  
print(arr * 0.5)
```

```
[[0.5 1. ]  
 [1.5 2. ]]
```

NumPy - funkcje matematyczne

- wynikiem poniższych funkcji jest skalar:
 - `np.sum()` - suma elementów tablicy
 - `np.mean()` - średnia arytmetyczna elementów tablicy
 - `np.median()` - mediana elementów tablicy
 - `np.min()` - wartość najmniejszego elementu tablicy
 - `np.max()` - wartość największego elementów tablicy

```
arr = np.array([[1,2,3],[4,5,6]])  
print(f"sum = {np.sum(arr)}")  
print(f"mean = {np.mean(arr)}")  
print(f"median = {np.median(arr)}")  
print(f"min = {np.min(arr)}")  
print(f"max = {np.max(arr)}")
```

```
sum = 21  
mean = 3.5  
median = 3.5  
min = 1  
max = 6
```

NumPy - funkcje matematyczne

- stosując parametr **axis** można określić wzdłuż których kierunków wykonywane są operacje

```
arr = np.array([[1,2,3],[4,5,6]])
print(arr)
print(f"sumy wierszy = {np.sum(arr,axis=1)}")
print(f"sumy kolumn = {np.sum(arr,axis=0)}")
print(f"średnie wierszy = {np.mean(arr,axis=1)}")
print(f"średnie kolumn = {np.mean(arr,axis=0)}")
```

```
[[1 2 3]
 [4 5 6]]
sumy wierszy = [ 6 15]
sumy kolumn = [5 7 9]
średnie wierszy = [2. 5.]
średnie kolumn = [2.5 3.5 4.5]
```

NumPy - inne funkcje

- `np.sort(arr)` - sortowanie elementów tablicy w kolejności rosnącej
- `np.transpose(arr)` - transponowanie elementów tablicy
- `np.dot(arr1, arr2)` - oblicza iloczyn macierzy
- `np.linalg.det(arr)` - oblicza wyznacznik macierzy
- `np.linalg.inv(arr)` - oblicza odwrotność macierzy
- `np.linalg.solve(A, B)` - rozwiązuje układ równań $Ax = B$
- `np.linalg.norm(arr)` - oblicza normę wektora lub macierzy

NumPy - przykład

- rozwiązanie układu równań

```
import numpy as np

A = np.array([
    [1, 1, 1, 1],
    [1, 2, 2, 2],
    [1, 2, 3, 3],
    [1, 2, 3, 4],
])

b = np.array([6, 11, 14, 16])

x = np.linalg.solve(A, b)
print(x)
```

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 6 \\ x_1 + 2x_2 + 2x_3 + 2x_4 = 11 \\ x_1 + 2x_2 + 3x_3 + 3x_4 = 14 \\ x_1 + 2x_2 + 3x_3 + 4x_4 = 16 \end{cases}$$

[1. 2. 1. 2.]

Matplotlib - instalacja



- ❑ **Matplotlib** - biblioteka umożliwiająca wizualizację danych z wykorzystaniem różnego rodzaju wykresów (słupkowe, kołowe, histogramy, mapy)
- ❑ strona internetowa: <https://matplotlib.org/> (EN)
- ❑ szczególnie wydajne jest połączenie możliwości biblioteki z innymi bibliotekami naukowymi, takimi jak **NumPy**, **Pandas** czy **SciPy**
- ❑ w celu instalacji biblioteki w edytorze Visual Studio Code wybieramy w menu głównym: **Terminal** → **Nowy terminal** i wpisujemy:
pip install matplotlib
- ❑ do uaktualnienia wpisujemy: **pip install --upgrade matplotlib**
- ❑ zastosowanie biblioteki **NumPy** w programie wymaga jej importowania, zazwyczaj importuje się główny moduł tej biblioteki, czyli **pyplot**

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

Matplotlib - wykres liniowy

- wykresy tworzone są na podstawie tzw. figur (**figure**)
- figura to kontener zawierający jedną lub więcej osi (**axes**) - wykres 2D zawiera osie X i Y, wykres 3D zawiera osie X, Y i Z
- wykresy można opisywać dodając etykiety osi, legendy, tytuły, itp.
- przykład: wykres funkcji $y = \cos(x)$

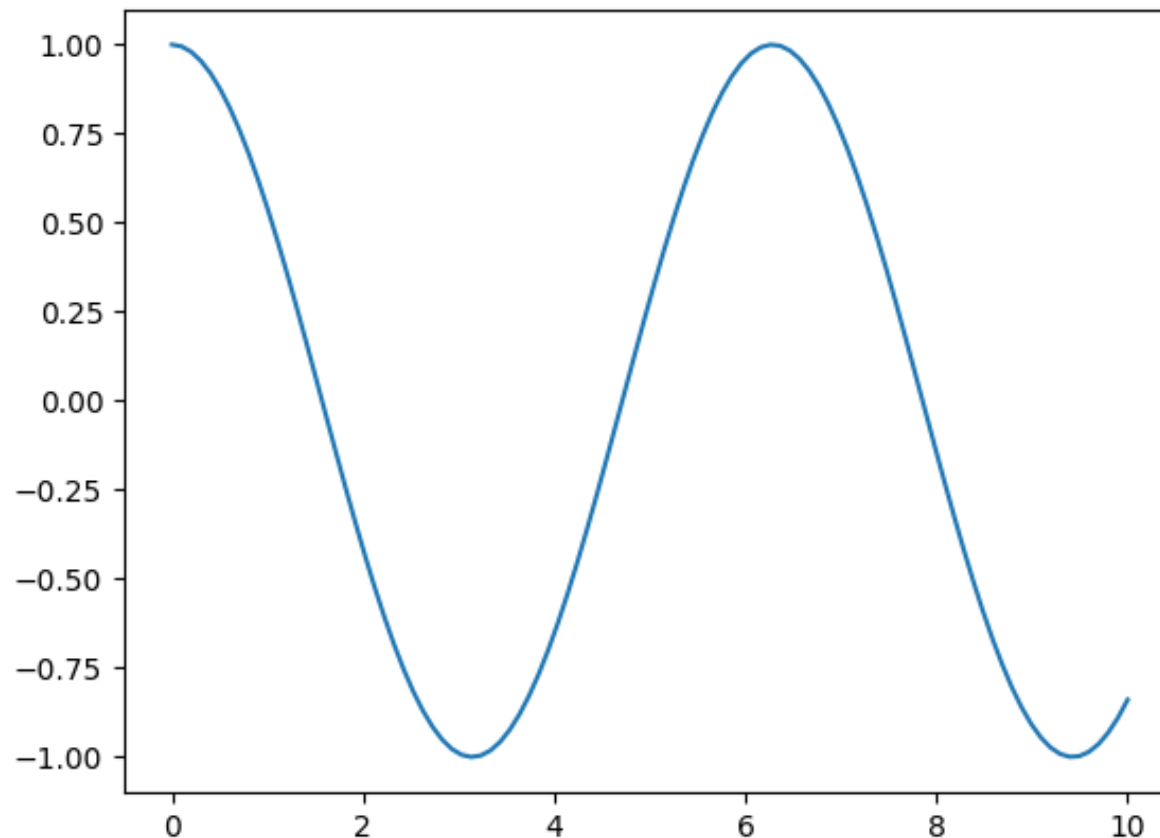
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y = np.cos(x)

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

Matplotlib - wykres liniowy

- wykres funkcji: $y = \cos(x)$



Matplotlib - opisanie wykresu

- `plt.title(tekst)` - dodaje tytuł wykresu
- `plt.xlabel(tekst)` - dodaje etykietę osi X
- `plt.ylabel(tekst)` - dodaje etykietę osi Y
- `plt.legend()` - wyświetla legendę
- tekst legendy podajemy w funkcji `plot()` jako `label`

```
plt.plot(x, y, label="cos(x)")  
...  
plt.legend()
```

- `plt.grid(wartość_logiczna)` - włącza/wyłącza wyświetlanie siatki

Matplotlib - wykres liniowy

- wykresy funkcji $y = \cos(x)$ z dodatkowymi elementami

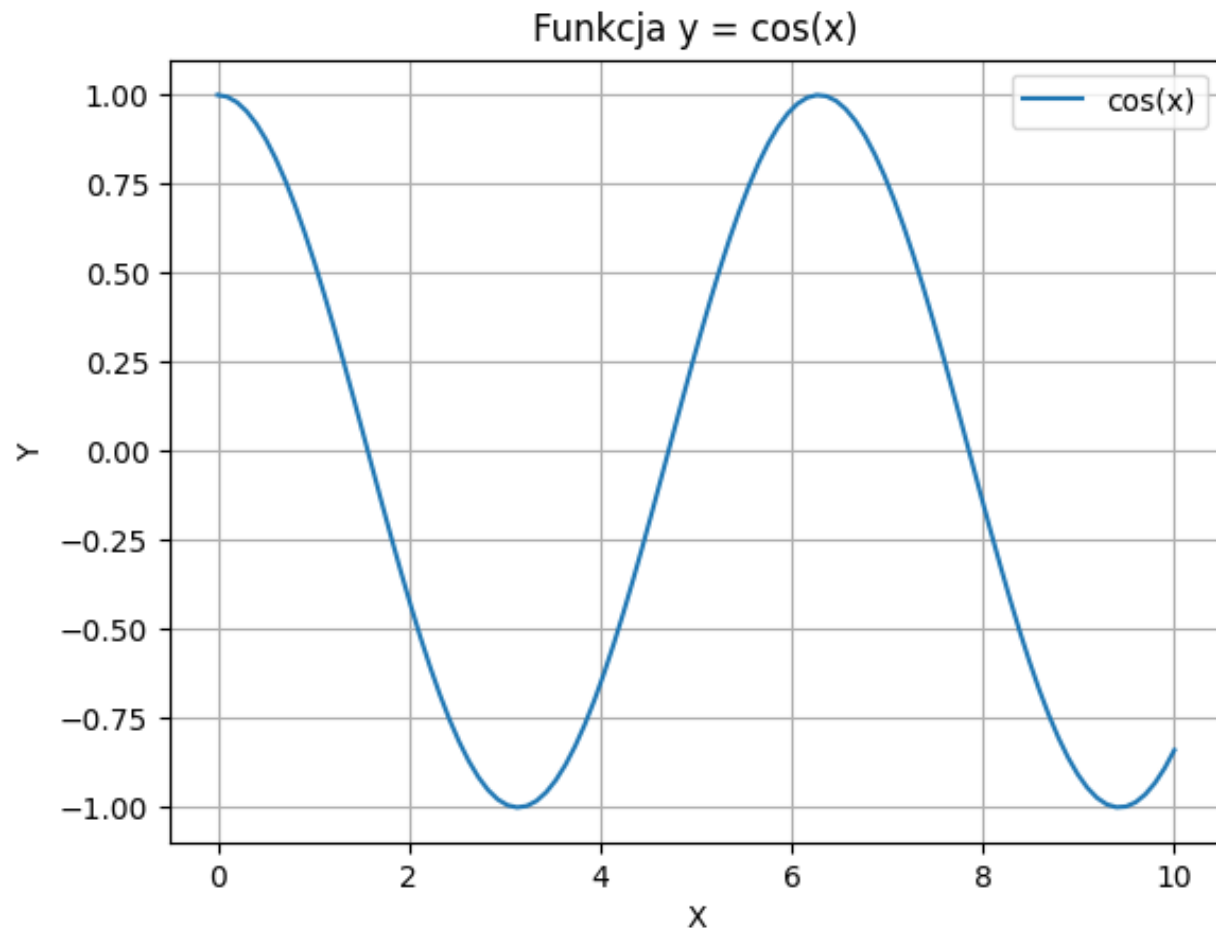
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y = np.cos(x)

plt.plot(x, y, label="cos(x)")
plt.title("Funkcja y = cos(x)")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.legend()
plt.show()
```

Matplotlib - wykres liniowy

- wykresy funkcji $y = \cos(x)$ z dodatkowymi elementami



Matplotlib - zmiana sposobu rysowania linii

- sposób rysowania linii ustala się w funkcji `plt.plot()`:
 - stosując parametry `color`, `linestyle`, `marker`

```
plt.plot(x, y, color="green", linestyle="--", marker="o")
```

- stosując format stringu: `[marker][linestyle][color]`

```
plt.plot(x, y, "o--g")
```

- kolory rysowania linii:

- | | | |
|---------------|---------------|--------------|
| ■ "b" - blue | "g" - green | "r" - red |
| ■ "c" - cyan | "m" - magenta | "y" - yellow |
| ■ "k" - black | "w" - white | |

Matplotlib - zmiana sposobu rysowania linii

□ sposób rysowania linii:

- "-" solid line style
- "--" dashed line style
- "-." dash-dot line style
- ":" dotted line style

□ znaczniki:

- "." - point marker
- "," - pixel marker
- "o" - circle marker
- "v" - triangle_down marker
- "^" - triangle_up marker
- "<" - triangle_left marker
- ">" - triangle_right marker
- "1" - tri_down marker
- "2" - tri_up marker
- "3" - tri_left marker
- "4" - tri_right marker
- "8" - octagon marker
- "s" - square marker
- "p" - pentagon marker
- "P" - plus (filled) marker
- "*" - star marker
- "h" - hexagon1 marker
- "H" - hexagon2 marker
- "+" - plus marker
- "x" - x marker

Matplotlib - dwie serie danych na jednym wykresie

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

# Zakres X
X = np.arange(-np.pi, np.pi, 0.01)

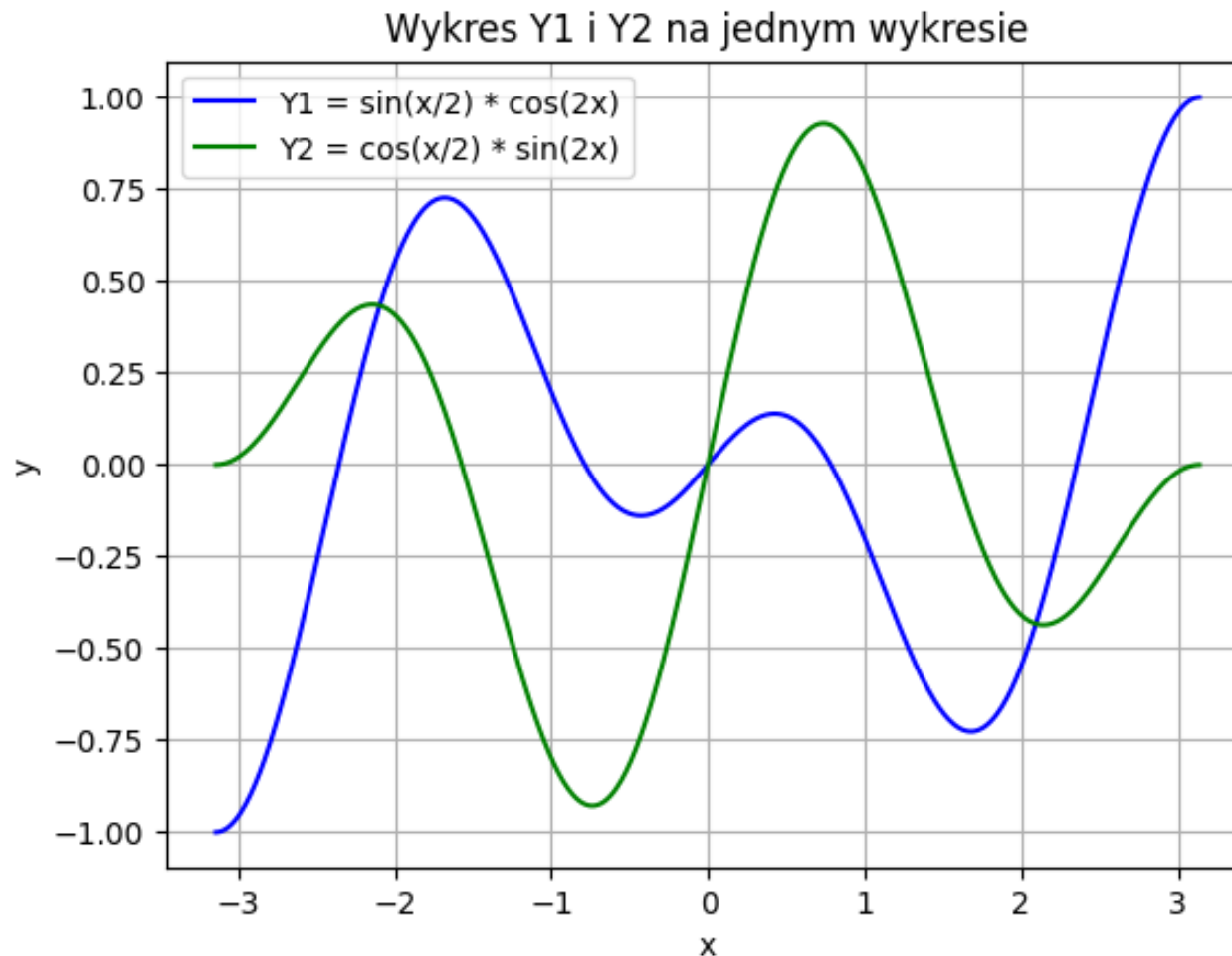
# Obliczanie funkcji
Y1 = np.sin(X / 2) * np.cos(2 * X)
Y2 = np.cos(X / 2) * np.sin(2 * X)

# Jeden wykres z dwoma seriami danych
plt.plot(X, Y1, color='blue', label="Y1 = sin(x/2) * cos(2x)")
plt.plot(X, Y2, color='green', label="Y2 = cos(x/2) * sin(2x)")

plt.title("Wykres Y1 i Y2 na jednym wykresie")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()
plt.show()
```

Matplotlib - dwie serie danych na jednym wykresie

- wykresy funkcji: $y_1 = \sin(x/2) \cdot \cos(2x)$ i $y_2 = \cos(x/2) \cdot \sin(2x)$



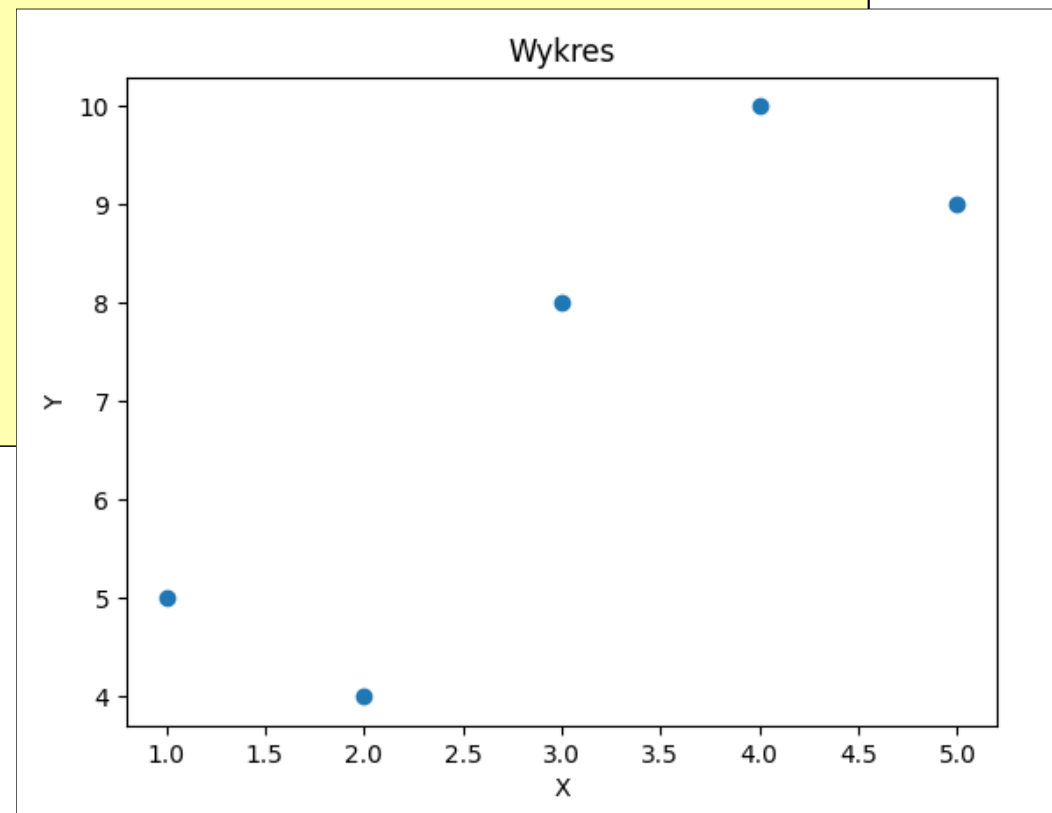
Matplotlib - wykres punktowy (Scatter Plot)

- przedstawia relację między dwoma zestawami danych

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

```
x = [1, 2, 3, 4, 5]  
y = [5, 4, 8, 10, 9]
```

```
plt.scatter(x, y)  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.title("Wykres")  
plt.show()
```



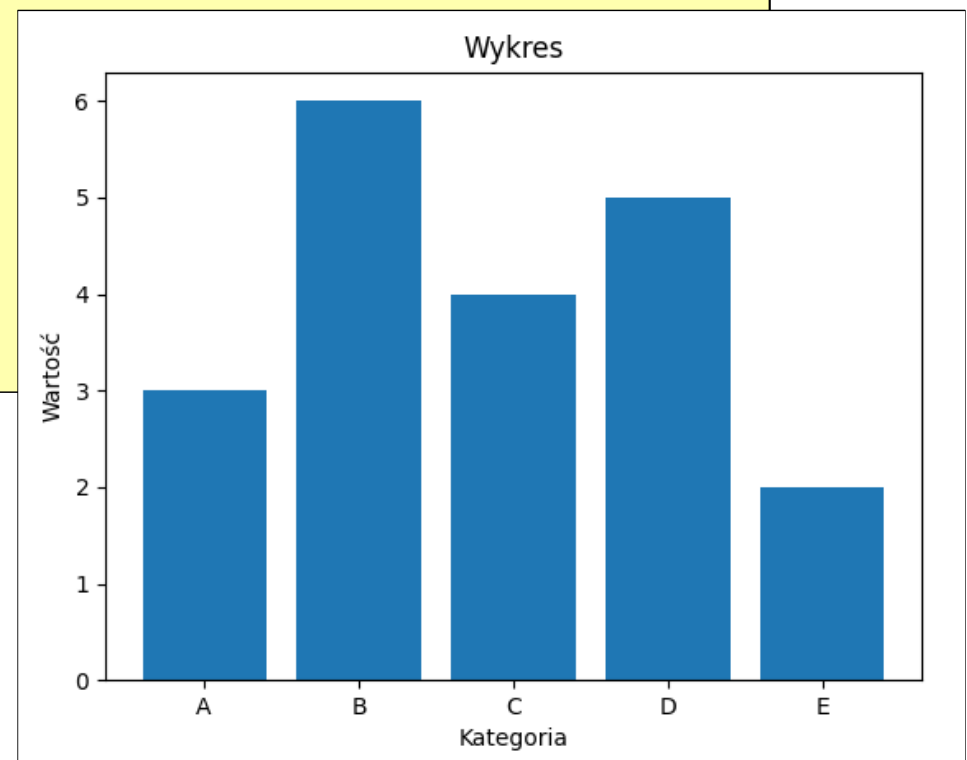
Matplotlib - wykres słupkowy (Bar Plot)

- przedstawia porównanie różnych kategorii

```
import matplotlib.pyplot as plt

kategoria = ['A', 'B', 'C', 'D', 'E']
wartosc = [3, 6, 4, 5, 2]

plt.bar(kategoria, wartosc)
plt.xlabel('Kategoria')
plt.ylabel('Wartość')
plt.title('Wykres')
plt.show()
```



Matplotlib - histogram

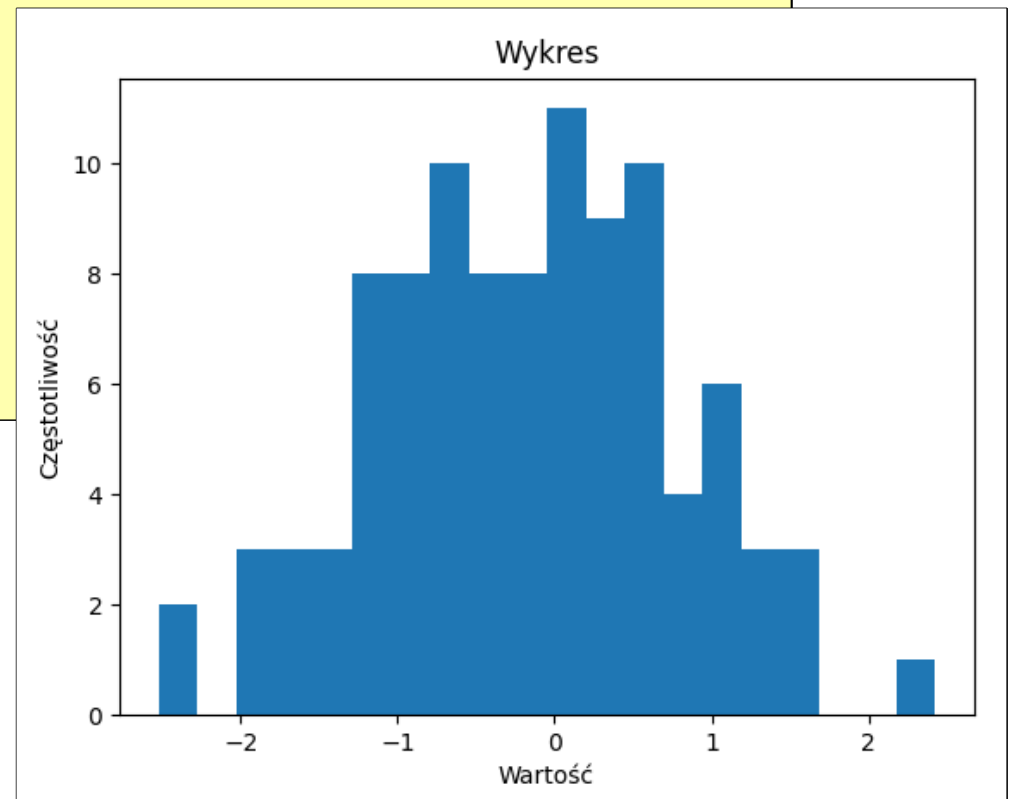
- przedstawia rozkład zestawu danych

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

data = np.random.randn(100)

plt.hist(data, bins=30)
plt.xlabel('Wartość')
plt.ylabel('Częstotliwość')
plt.title('Wykres')
plt.show()
```

- **bins** - liczba przedziałów w histogramie



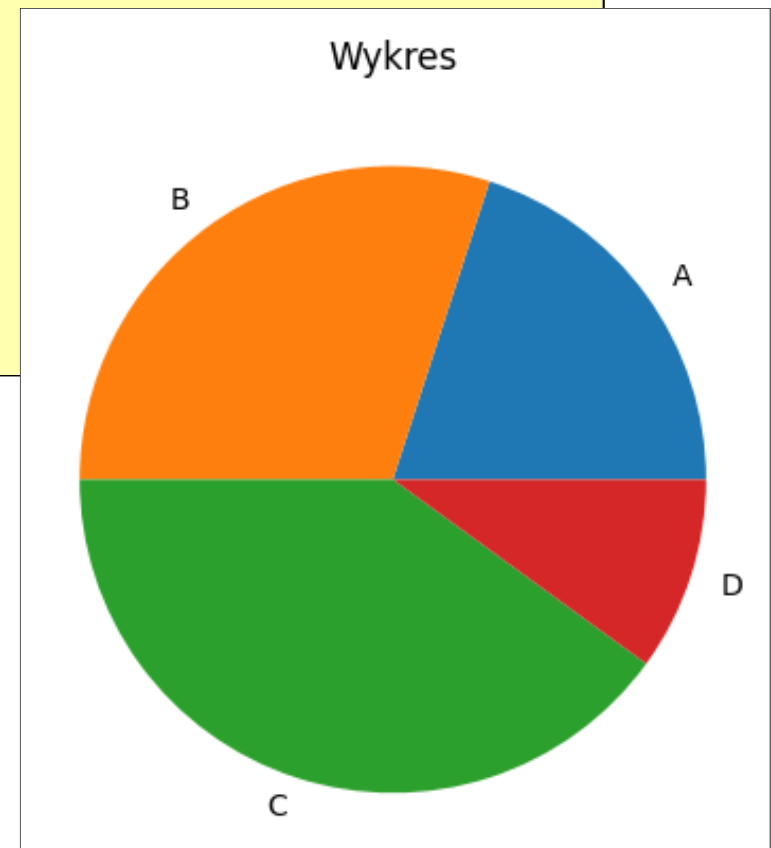
Matplotlib - wykres kołowy (Pie Chart)

- przedstawia proporcje różnych części względem całości zestawu danych

```
import matplotlib.pyplot as plt

wartosci = [20, 30, 40, 10]
etykiety = ['A', 'B', 'C', 'D']

plt.pie(wartosci, labels=etykiety)
plt.title("Wykres")
plt.show()
```



SciPy - instalacja



- ❑ **SciPy** (Scientific Python) - biblioteka zawierająca wiele naukowych i numerycznych algorytmów i funkcji, które rozszerzają **NumPy**
- ❑ strona internetowa: <https://scipy.org/> (EN)
- ❑ w celu instalacji biblioteki w edytorze Visual Studio Code wybieramy w menu głównym: **Terminal** → **Nowy terminal** i wpisujemy:
pip install scipy

```
PS C:\Users\jaros> pip install scipy
Collecting scipy
  Downloading scipy-1.13.1-cp312-cp312-win_amd64.whl.metadata (60 kB)
    _____ 60.6/60.6 kB
645.4 kB/s eta 0:00:00
Requirement already satisfied: numpy<2.3,>=1.22.4 in
c:\users\jaros\appdata\local\programs\python\python312\lib\site-packages
(from scipy) (1.26.4)
Downloading scipy-1.13.1-cp312-cp312-win_amd64.whl (45.9 MB)
  _____ 45.9/45.9 MB
29.7 MB/s eta 0:00:00
Installing collected packages: scipy
Successfully installed scipy-1.13.1
```

SciPy - moduły

- ❑ **constants** - zawiera nieprzybliżone stałe fizyczne i matematyczne
- ❑ **datasets** - zawiera różnego rodzaju zbiory danych, które można wykorzystać przykładowo do testowania algorytmów uczenia maszynowego
- ❑ **linalg** - zawiera funkcje do algebry liniowej, jak np. operacje na macierzach, dekompozycja macierzy, rozwiązywanie układów równań czy znalezienie macierzy odwrotnej
- ❑ **spatial** - zawiera funkcje działające na danych przestrzennych takich jak geometria wyrażona we współrzędnych kartezjańskich
- ❑ **stats** - zawiera bardzo wiele funkcjonalności takich jak rozkłady prawdopodobieństwa, funkcje takie jak miara rozkładu i testy statystyczne
- ❑ **integrate** - zawiera funkcje służące do całkowania i rozwiązywania zwyczajnych równań różniczkowych
- ❑ **interpolate** - zawiera różne algorytmy interpolacji danych

SciPy - moduły

- ❑ **io** - zawiera proste funkcje pozwalające na odczytywanie i zapisywanie danych w różnych formatach
- ❑ **ndimage** - zawiera funkcje służące do obróbki zdjęć jak nakładanie filtrów, skalowanie zdjęcia czy konwolucja
- ❑ **odr** - zawiera funkcje odpowiedzialne za dopasowywanie modeli matematycznych do danych
- ❑ **optimize** - zawiera funkcje, które są iterowane tak długo aż rozwiązanie się uzbieżni poniżej zadanego kryterium
- ❑ **fft** - zawiera warianty transformaty Fouriera
- ❑ **signal** - zawiera funkcje pozwalające na obróbkę sygnału np. plików audio.
- ❑ **sparse** - zawiera funkcjonalności do pracy z macierzami rzadkimi

SciPy - przykład

- rozwiązanie układu równań

```
import numpy as np
from scipy.linalg import solve

A = np.array([
    [1, 1, 1, 1],
    [1, 2, 2, 2],
    [1, 2, 3, 3],
    [1, 2, 3, 4],
])
b = np.array([6, 11, 14, 16])

x = solve(A, b)
b_new = np.dot(A, x)

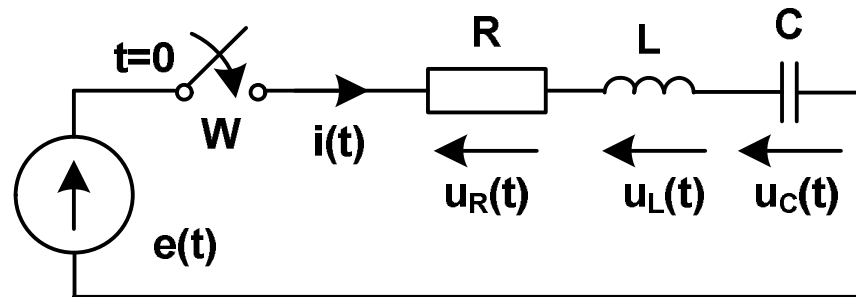
print("x:", x)
print("b_new:", b_new)
```

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 6 \\ x_1 + 2x_2 + 2x_3 + 2x_4 = 11 \\ x_1 + 2x_2 + 3x_3 + 3x_4 = 14 \\ x_1 + 2x_2 + 3x_3 + 4x_4 = 16 \end{cases}$$

```
x: [1.  2.  1.  2.]
b_new: [ 6. 11. 14. 16.]
```

SciPy - stan nieustalony w układzie RLC

- w szeregowym obwodzie RLC w chwili $t = 0$ zamykamy wyłącznik W



- dynamikę obwodu opisuje układ dwóch równań pierwszego rzędu:

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -\frac{1}{LC}x_1(t) - \frac{R}{L}x_2(t) + \frac{1}{LC}e(t) \end{cases}$$

$$x_1(t) = u_C(t)$$

- gdzie:

$$x_2(t) = \frac{du_C(t)}{dt}$$

SciPy - stan nieustalony w układzie RLC

- wartości elementów obwodu:

$$L = 1 \text{ mH}, \quad C = 1 \mu\text{F}, \quad e(t) = E = 5 \text{ V}$$

- zerowe warunki początkowe:

$$i(0) = i_L(0) = 0, \quad u_C(0) = 0 \quad \longrightarrow \quad x_1(0) = 0, \quad x_2(0) = 0$$

- rezystancja krytyczna:

$$R_k = 2 \sqrt{\frac{L}{C}}$$

- przypadek aperiodyczny:

$$R > R_k \quad \longrightarrow \quad R = 100 \, \Omega$$

- przypadek aperiodyczny krytyczny:

$$R = R_k \quad \longrightarrow \quad R = 63.25 \, \Omega$$

- przypadek oscylacyjny:

$$R < R_k \quad \longrightarrow \quad R = 10 \, \Omega$$

SciPy - stan nieustalony w układzie RLC

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# Parametry układu
L = 1.0e-3 # Henry
C = 1.0e-6 # Farad
R = 100    # Ohm
E = 5      # V

# Układ równań ODE
def rlc_system(t, x):
    x1, x2 = x
    dx1dt = x2
    dx2dt = (-1 / (L * C)) * x1 - (R / L) * x2 + E / (L * C)
    return [dx1dt, dx2dt]
```

SciPy - stan nieustalony w układzie RLC

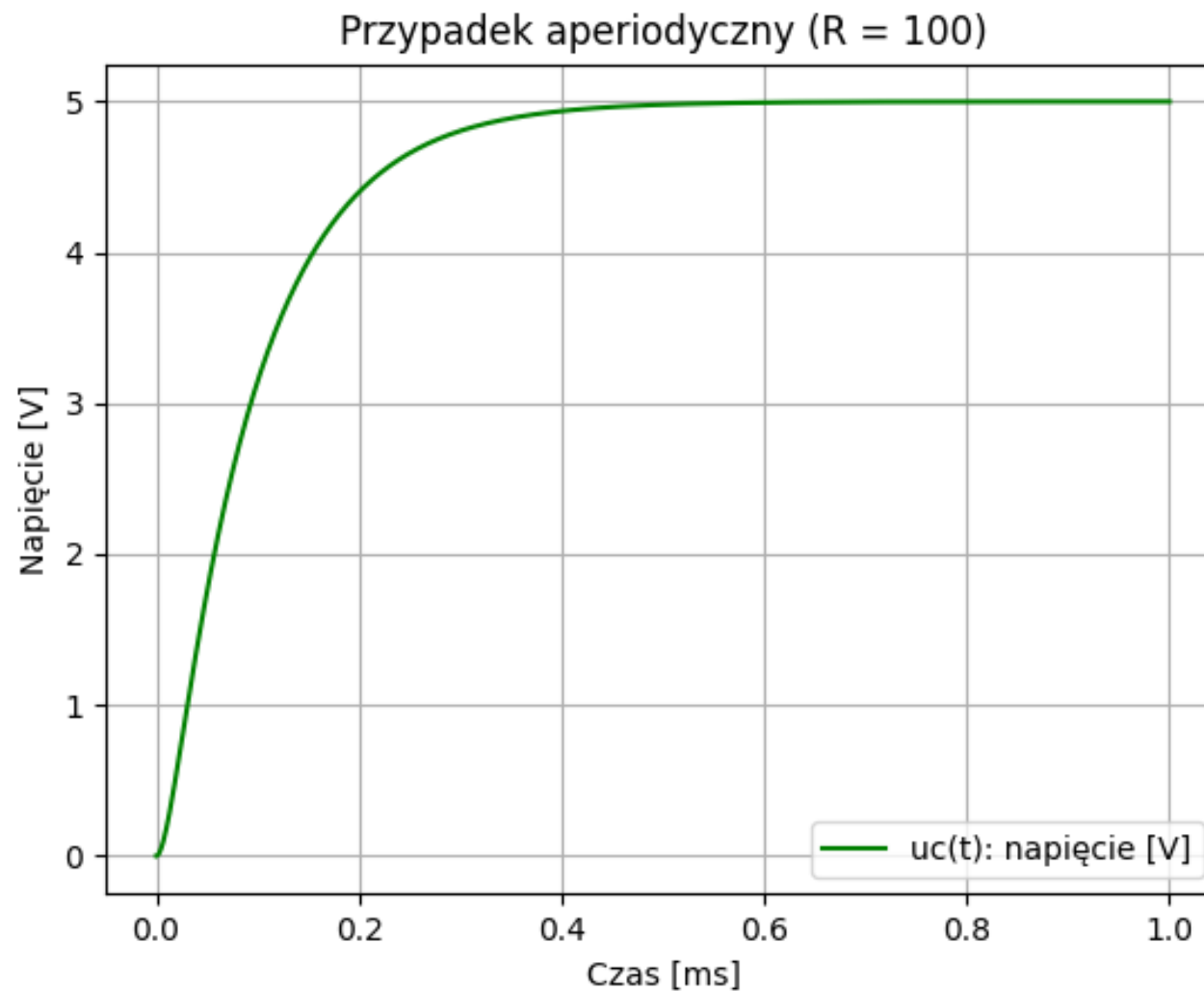
```
# Czas symulacji
t_span = (0, 0.001) # od 0 do 1 ms
t_eval = np.linspace(0, 0.001, 1000)

# Warunki początkowe:
x0 = [0, 0]

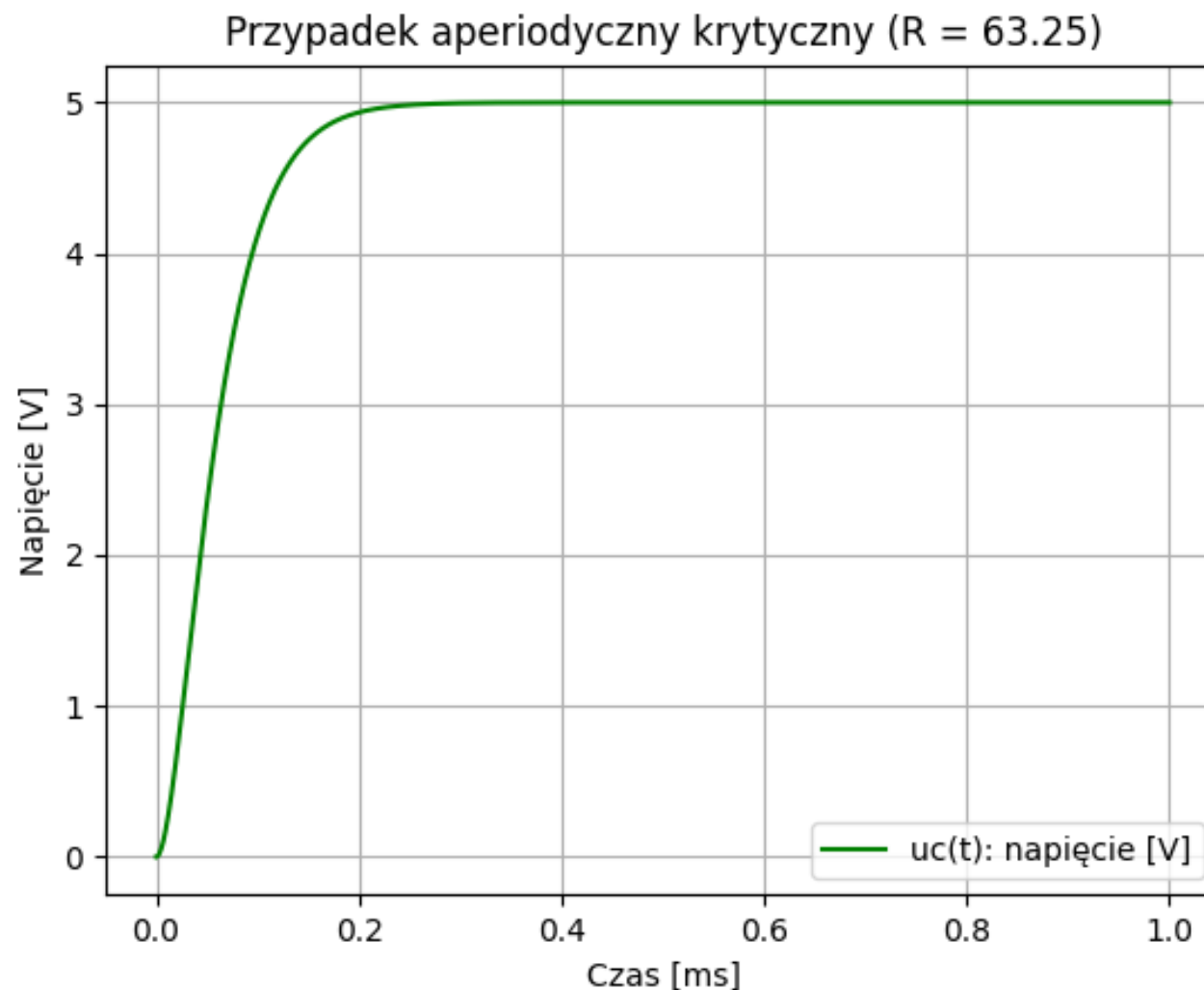
# Rozwiązanie ODE
sol = solve_ivp(rlc_system, t_span, x0, t_eval=t_eval)

# Wykresy
plt.plot(sol.t*1000, sol.y[0], label='uc(t) [V]', color='green')
plt.xlabel('Czas [ms]')
plt.ylabel('Napięcie [V]')
plt.title('Stan nieustalony w układzie RLC')
plt.grid(True)
plt.legend()
plt.show()
```

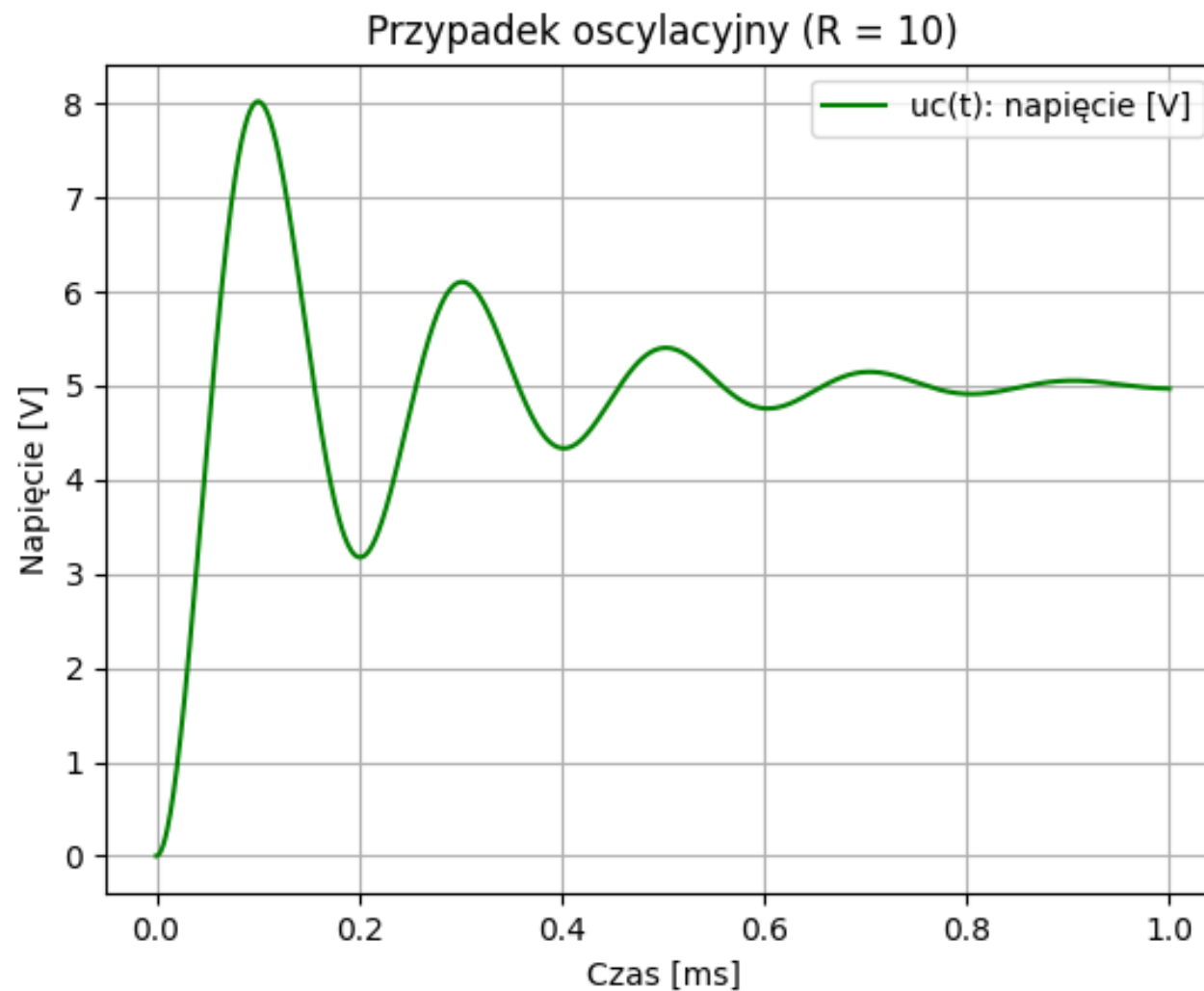
SciPy - stan nieustalony w układzie RLC



SciPy - stan nieustalony w układzie RLC



SciPy - stan nieustalony w układzie RLC



Koniec wykładu nr 7

Dziękuję za uwagę!