Politechnika Wrocławska

Programowanie obiektowe

W11FTE-SI0080G (INP001129WL)

rok akademicki 2023/24

semestr letni

Wykład 8

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220

unite! University Network for Innovation, Technology and Engineering

HR EXCELLENCE IN RESEARCH

Evaluated by IEP INSTITUTIONAL EVALUATION PROGRAMME www.iep-qaa.org

Plan

Biblioteka scipy

- funkcje specjalne
- interpolacja
- aproksymacja
- rozwiązywanie zagadnienia początkowego

https://docs.scipy.org/doc/scipy/tutorial/index.html

Politechnika Wrocławska

Biblioteka scipy

Narzędzia numeryczne dotyczące m. in.:

- funkcji specjalnych (scipy.special)
- całkowania numerycznego (scipy.integrate)
- problemów optymalizacyjnych (scipy.optimize)
- interpolacji (scipy.interpolate)
- transformaty Fouriera (scipy.fft)
- zagadnień algebry liniowej (scipy.linalg)

https://docs.scipy.org/doc/scipy/tutorial/index.html

Politechnika Wrocławska

Funkcje specjalne

Przykład pokazuje:

- importowanie modułu special biblioteki scipy
- obliczanie wartości funkcji Bessela
- wykorzystanie polecenia meshgrid do wygenerowania dwuwymiarowej siatki punktów

scipy_01.py

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje użycie funkcji jv
z modułu special z biblioteki scipy
do obliczania wartości funkcji Bessela.
https://docs.scipy.org/doc/scipy/tutorial/special.html
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy import special

# siatka argumentów
x = np.linspace(0,10)

# tablica rzędów funkcji Bessela
nu = np.array([0, 1, 2, 3])

# wykorzystanie funkcji meshgrid
# do wygenerowania dwuwymiarowych tablic:
# rzędów (nus) oraz argumentów (xs)
nus, xs = np.meshgrid(nu, x)
```

Funkcje specjalne

Przykład pokazuje:

- importowanie modułu special biblioteki scipy
- obliczanie wartości funkcji Bessela
- wykorzystanie polecenia meshgrid do wygenerowania dwuwymiarowej siatki punktów

scipy_01.py

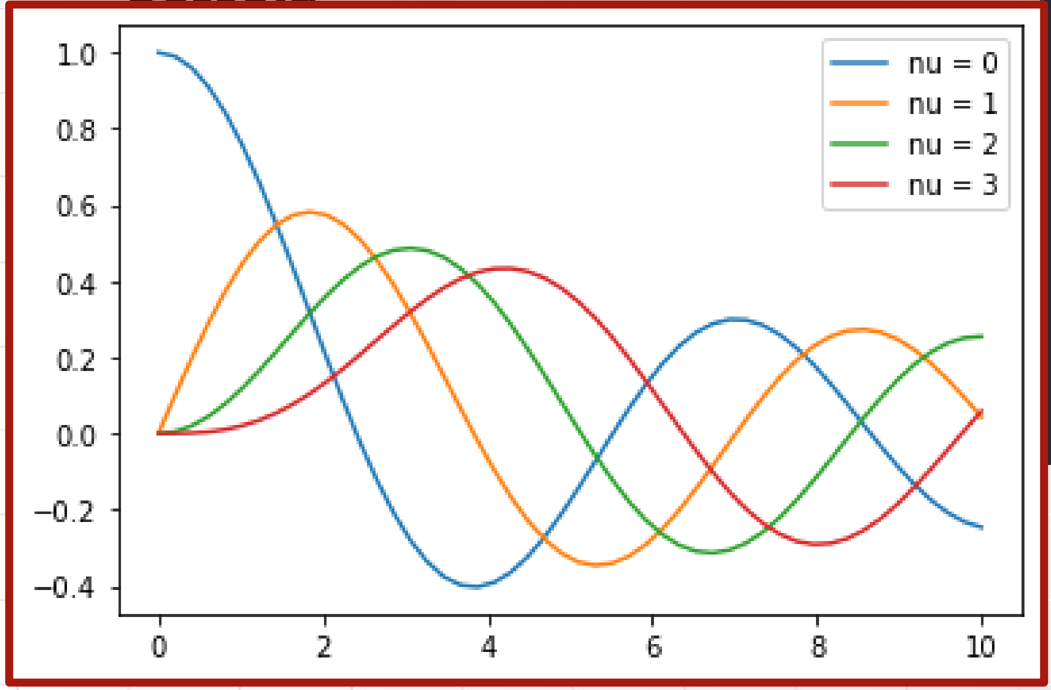
```python
# tablica rzędów funkcji Bessela
nu = np.array([0, 1, 2, 3])

# wykorzystanie funkcji meshgrid
# do wygenerowania dwuwymiarowych tablic:
# rzędów (nus) oraz argumentów (xs)
nus, xs = np.meshgrid(nu, x)

# obliczenie funkcji Bessela
bessel = special.jv(nus, xs)

# stworzenie wykresów
lines = plt.plot(xs, bessel)

# utworzenie legendy
plt.legend(lines,("nu = " + format(n) for n in nu))
#plt.legend(lines,(r"$\nu$ = " + f"{n}" for n in nu))
#plt.legend(lines,(f"$J_{n}$" for n in nu))
```

Politechnika Wrocławska

Funkcje specjalne

Przykład pokazuje:

- importowanie modułu special biblioteki scipy
- obliczanie wartości funkcji Bessela



scipy_01.py

```python
# tablica rzędów funkcji Bessela
nu = np.array([0, 1, 2, 3])

# wykorzystanie funkcji meshgrid
# do wygenerowania dwuwymiarowych tablic:
# rzędów (nus) oraz argumentów (xs)
nus, xs = np.meshgrid(nu, x)

# obliczenie funkcji Bessela
bessel = special.jv(nus, xs)

# stworzenie wykresów
lines = plt.plot(xs, bessel)

# utworzenie legendy
plt.legend(lines,("nu = " + format(n) for n in nu))
#plt.legend(lines,(r"$\nu$ = " + f"{n}" for n in nu))
#plt.legend(lines,(f"$J_{n}$" for n in nu))
```

Politechnika Wrocławska

Funkcje specjalne

Przykład pokazuje:

- obliczanie funkcji Bessela
- wykorzystanie funkcji special.jn_zeros()

scipy_02.py

```
# -*- coding: utf-8 -*-
"""
Przykład pokazuje użycie funkcji j0 oraz jn_zeros
z modułu special z biblioteki scipy
do obliczania wartości funkcji Bessela.
https://docs.scipy.org/doc/scipy/tutorial/special.html
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy import special

# skalowanie funkcji Bessela
def scaled_bessel(x, k):
    a = special.jn_zeros(0, k)[-1]
    return special.j0(a*x)
```

Politechnika Wrocławska

Funkcje specjalne

Przykład pokazuje:

- obliczanie funkcji Bessela
- wykorzystanie funkcji special.jn_zeros()

scipy_02.py

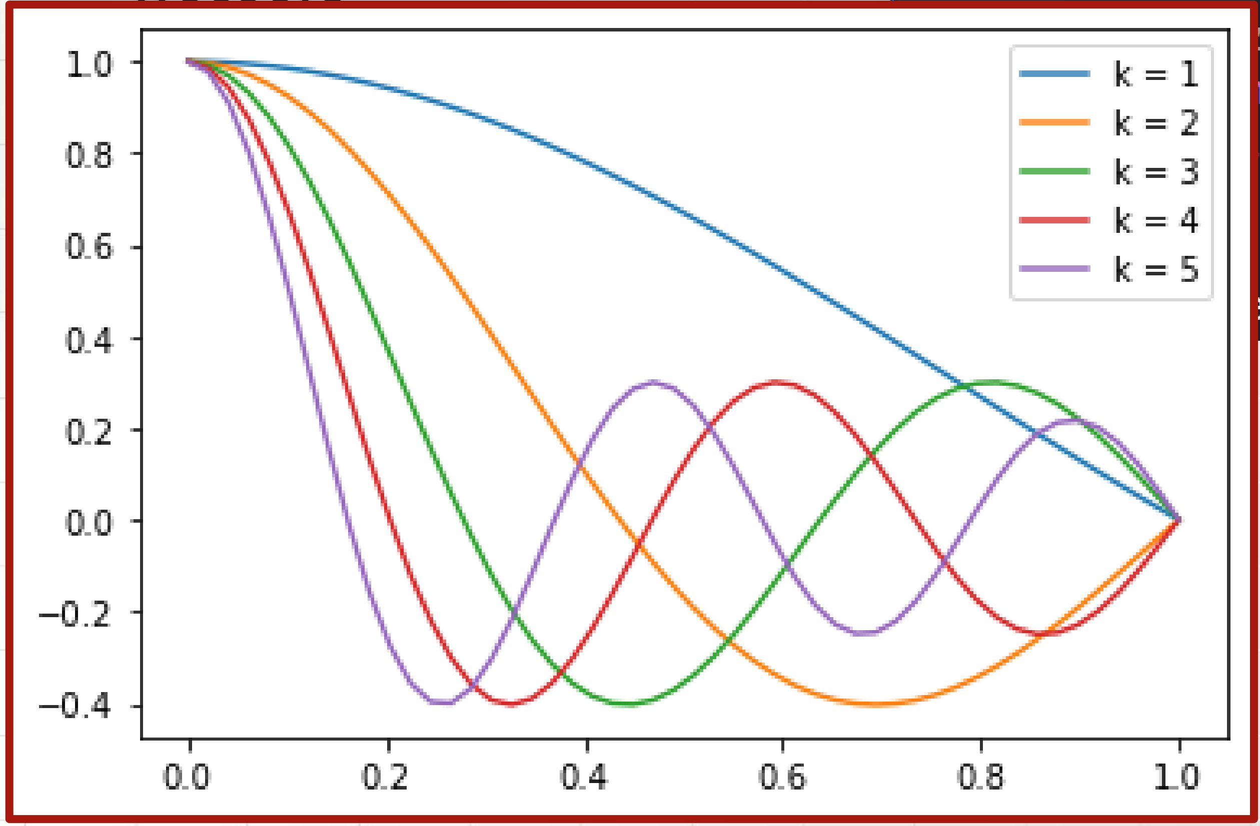
```python
# utworzenie siatki punktów
# na odcinku jednostkowym
x = np.linspace(0,1)

# utworzenie wykresów skalowanych funkcji Bessela
for k in range(1, 6):
    plt.plot(x, scaled_bessel(x, k),
                 label="k = " + format(k,"d"))

plt.legend()
```

Funkcje specjalne

Przykład pokazuje:

- obliczanie funkcji Bessela

```
# utworzenie siatki punktów
# na odcinku jednostkowym
x = np.linspace(0,1)

zenie wykresów skalowanych funkcji Bessela
 range(1, 6):
 plot(x, scaled_bessel(x, k),
         label="k = " + format(k,"d"))

end()
```



Politechnika Wrocławska

Interpolacja

Przykład pokazuje:

- interpolację danych funkcją interp1d (scipy.interpolate)
- funkcja interp1d zwraca funkcję, którą można wywołać, aby obliczyć wartości interpolowane dla dowolnych argumentów

scipy_03.py

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazujący interpolację danych.
Znane wartości funkcji sinus w węzłach
są interpolowane z funkcji interp1d
z modułu interpolate biblioteki scipy.
Funkcja interp1d pozwala wyznaczyć
funkcję interpolującą.
https://docs.scipy.org/doc/scipy/reference/generated/sc
"""


import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

# dane węzłów interpolacji
x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
y = np.sin(x)
```

Interpolacja

Przykład pokazuje:

- interpolację danych funkcją interp1d (scipy.interpolate)
- funkcja interp1d zwraca funkcję, którą można wywołać, aby obliczyć wartości interpolowane dla dowolnych argumentów

scipy_03.py

```python
# dane węzłów interpolacji
x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
y = np.sin(x)

# wyznaczenie funkcji interpolujących
# domyślnie liniowej funkcji sklejanej
f_lin = interp1d(x, y)
# funkcji sklejanej kwadratowej
f_qua = interp1d(x, y, kind="quadratic")
# funkcji sklejanej sześciennej
f_cub = interp1d(x, y, kind="cubic")

# siatka punktów, dla których będą obliczane
# wartości funkcji interpolujących
x_int = np.linspace(0, 2, 100)*np.pi

# utworzenie wykresów
fig, ax = plt.subplots()
ax.plot(x, y, 'o', label="węzły interpolacji")
ax.plot(x_int, f_lin(x_int), label="interpolacja liniowa")
ax.plot(x_int, f_qua(x_int), label="interpolacja kwadratowa")
ax.plot(x_int, f_cub(x_int), label="interpolacja sześcienna")
ax.legend()
```

Politechnika Wrocławska

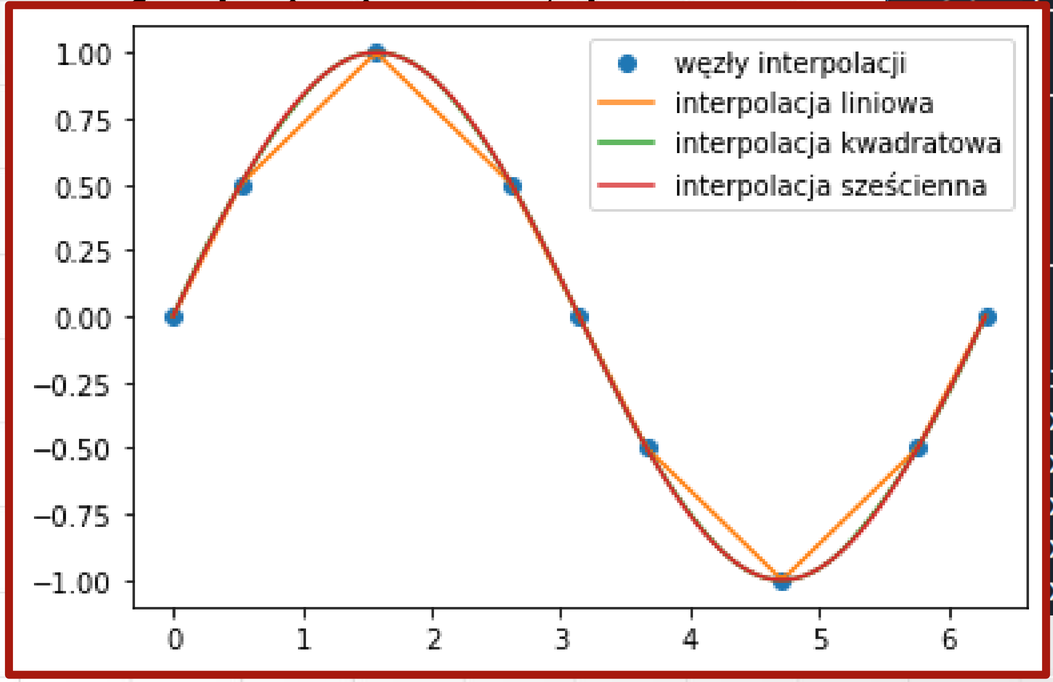
Interpolacja

Przykład pokazuje:

- interpolację danych funkcją interp1d (scipy.interpolate)

scipy_03.py

```
# dane węzłów interpolacji
x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
y = np.sin(x)

# wyznaczenie funkcji interpolujących
# domyślnie liniowej funkcji sklejanej
f_lin = interp1d(x, y)
# funkcji sklejanej kwadratowej
f_qua = interp1d(x, y, kind="quadratic")
 funkcji sklejanej sześciennej
_cub = interp1d(x, y, kind="cubic")

 siatka punktów, dla których będą obliczane
 wartości funkcji interpolujących
_int = np.linspace(0, 2, 100)*np.pi

 utworzenie wykresów
ig, ax = plt.subplots()
x.plot(x, y, 'o', label="węzły interpolacji")
x.plot(x_int, f_lin(x_int), label="interpolacja liniowa")
x.plot(x_int, f_qua(x_int), label="interpolacja kwadratowa")
x.plot(x_int, f_cub(x_int), label="interpolacja sześcienna")
x.legend()
```



Politechnika Wrocławska

Aproksymacja

Przykład pokazuje:

- aproksymację danych - curve_fit (scipy.optimize)

- zależność liniową wykorzystano do wygenerowania danych
- następnie do danych dodano szum
- funkcją curve_fit znaleziono parametry dopasowania funkcji liniowej do danych z szumem

scipy_04.py

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazujący aproksymację danych
funkcją liniową.
Dopasowanie wyznaczane z użyciem
funkcji curve_fit z modułu optimize
biblioteki scipy.
https://docs.scipy.org/doc/scipy/reference/genera
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# definicja funkcji liniowej
# x - argument
# a - współczynnik kierunkowy
# b - wyraz wolny
#def fun(x, a, b):
#    return a*x + b
# równoważna definicja z wykorzystaniem
# funkcji anonimowej
fun = lambda x, a, b : a*x + b
```

Politechnika Wrocławska

Aproksymacja

Przykład pokazuje:

- aproksymację danych - curve_fit (scipy.optimize)

- zależność liniową wykorzystano do wygenerowania danych
- następnie do danych dodano szum
- funkcją curve_fit znaleziono parametry dopasowania funkcji liniowej do danych z szumem

scipy_04.py

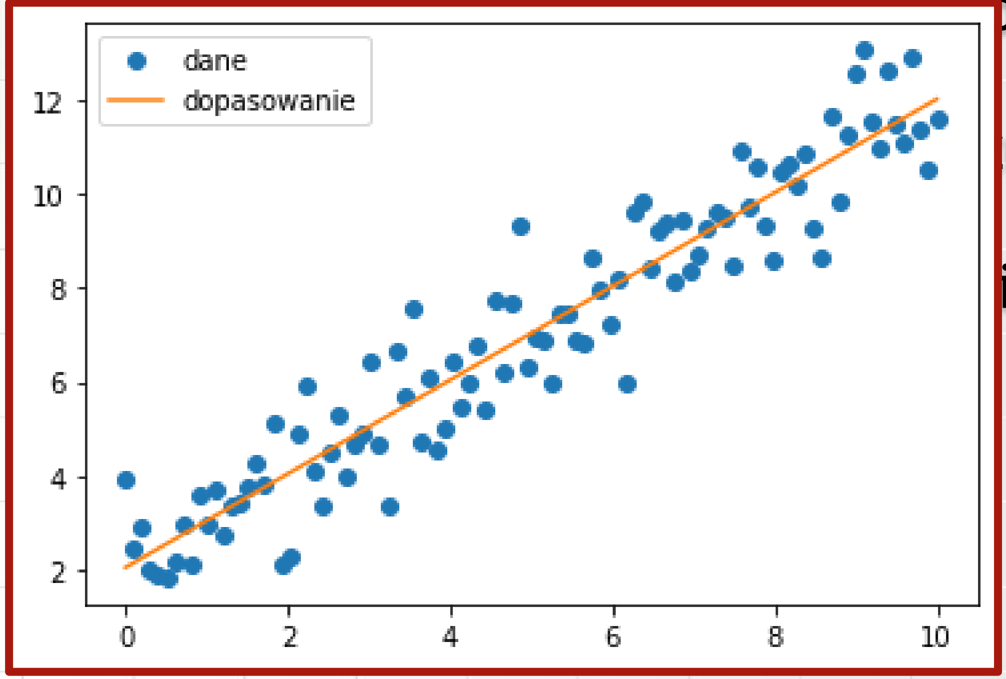
```python
# utworzenie danych do aproksymacji
# dane obliczone z wykorzystaniem
# funkcji liniowej z dodanym szumem
x = np.linspace(0, 10, 100)
y = fun(x, 1, 2) # y = 1*x + 2
yn = y + np.random.normal(size=len(x))

# dopasowanie do danych (x, yn)
# funkcja curve_fit znajduje parametry
# funkcji fun, które najlepiej dopasowywują ją
# do danych wejściowych
p_opt, _ = curve_fit(fun, x, yn)
print(p_opt)

# utworzenie wykresów
fig, ax = plt.subplots()
# danych wejściowych
ax.plot(x, yn, 'o', label="dane")
# danych dopasowanych
ax.plot(x, fun(x, *p_opt), label="dopasowanie")
ax.legend()
```

Politechnika Wrocławska

Aproksymacja

Przykład pokazuje:

- aproksymację danych - curve_fit (scipy.optimize)



scipy_04.py

```python
# utworzenie danych do aproksymacji
# dane obliczone z wykorzystaniem
# funkcji liniowej z dodanym szumem
x = np.linspace(0, 10, 100)
y = fun(x, 1, 2) # y = 1*x + 2
yn = y + np.random.normal(size=len(x))

# dopasowanie do danych (x, yn)
# funkcja curve_fit znajduje parametry
# funkcji fun, które najlepiej dopasowywują ją
# do danych wejściowych
p_opt, _ = curve_fit(fun, x, yn)
print(p_opt)

# utworzenie wykresów
fig, ax = plt.subplots()
# danych wejściowych
ax.plot(x, yn, 'o', label="dane")
# danych dopasowanych
ax.plot(x, fun(x, *p_opt), label="dopasowanie")
ax.legend()
```

Politechnika Wrocławska

Zagadnienie początkowe

Przykład pokazuje:

- numeryczne rozwiązanie zagadnienia początkowego z wykorzystaniem funkcji solve_ivp (scipy.integrate)

scipy_05.py

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazujący rozwiązywanie zagadnienia
początkowego z użyciem funkcji odeint
z modułu integrate biblioteki scipy.
https://docs.scipy.org/doc/scipy/reference/generated/s
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# prawa strona równania różniczkowego
# dg/dt = f(g, t)
# f(g, t) = -g/tau
tau = 1.
f = lambda g, t : -g/tau
```

Politechnika Wrocławska

Zagadnienie początkowe

Przykład pokazuje:

- numeryczne rozwiązanie zagadnienia początkowego z wykorzystaniem funkcji solve_ivp (scipy.integrate)

scipy_05.py

```python
# prawa strona równania różniczkowego
# dg/dt = f(g, t)
# f(t, g) = -g/tau
tau = 1.
f = lambda t, g : -g/tau

# zakres zmiennej niezależnej
t = np.array([0, 5])
# wektor zmiennej niezależnej
t_eval = np.linspace(*t)
# obiekt sol zawieta obliczone rozwiązanie
# dla podanego równania różniczkowego (f)
# z zadanym warunkiem początkowym (g(0)=100)
# dla podanych wartoci zmiennej niezależnej t_eval
sol = solve_ivp(f, t, [100], t_eval = t_eval)

# wykres obliczonego rozwiązania
plt.plot(sol.t,sol.y[0])
```

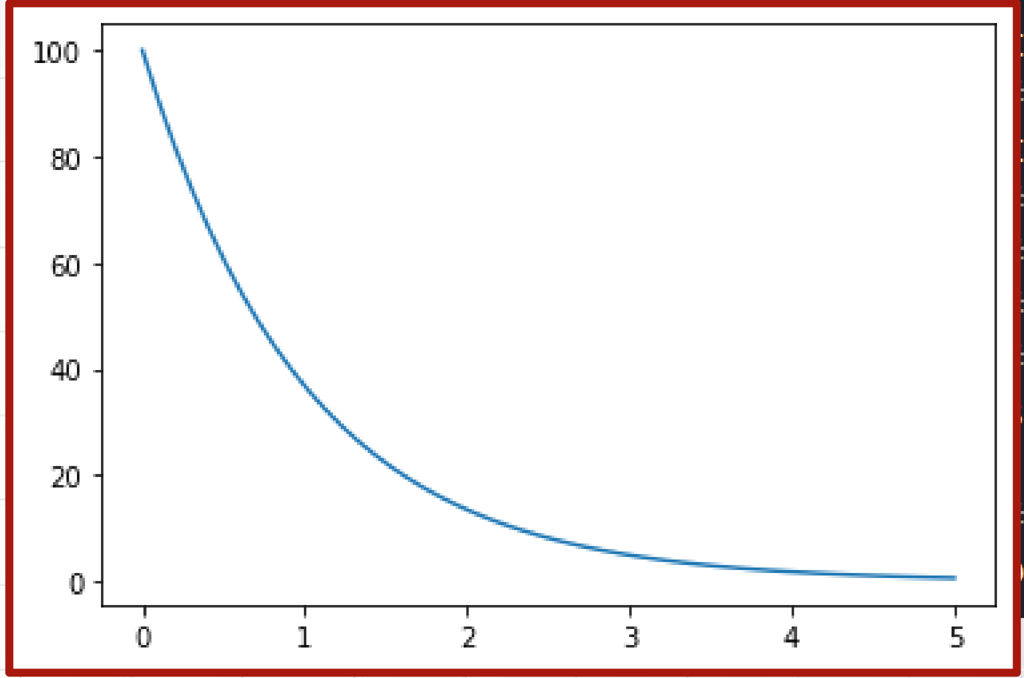
Zagadnienie początkowe

Przykład pokazuje:

- numeryczne rozwiązanie zagadnienia początkowego z wykorzystaniem funkcji solve_ivp (scipy.integrate)

scipy_05.py

```
# prawa strona równania różniczkowego
# dg/dt = f(g, t)
# f(t, g) = -g/tau
tau = 1.
f = lambda t, g : -g/tau

# zakres zmiennej niezależnej
 = np.array([0, 5])
 wektor zmiennej niezależnej
_eval = np.linspace(*t)
 obiekt sol zawieta obliczone rozwiązanie
 dla podanego równania różniczkowego (f)
 z zadanym warunkiem początkowym (g(0)=100)
 dla podanych wartoci zmiennej niezależnej t_eval
ol = solve_ivp(f, t, [100], t_eval = t_eval)

 wykres obliczonego rozwiązania
lt.plot(sol.t,sol.y[0])
```



Politechnika Wrocławska

Podsumowanie

Wybrane funkcje biblioteki scipy

- interpolacja
- aproksymacja
- rozwiązywanie zagadnienia początkowego