Politechnika Wrocławska

Programowanie obiektowe

INP001129WL

rok akademicki 2022/23

semestr zimowy

Wykład 6

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220

Politechnika Wrocławska

Plan wykładu

- Biblioteka scipy
  - funkcje specjalne
  - interpolacja
  - aproksymacja
  - rozwiązywanie zagadnienia początkowego
- Dziedziczenie
  - polimorfizm

Politechnika Wrocławska

Biblioteka scipy

- Biblioteka dostarcza narzędzi przydatnych do rozwiązywania wielu zagadnień numerycznych dotyczących 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

Biblioteka scipy

- Przykład pokazuje:
  - importowanie modułu special biblioteki scipy
  - obliczanie wartości funkcji Bessela funkcją special.jv()
  - dodatkowo wykorzystane jest polecenie meshgrid biblioteki numpy do wygenerowania dwuwymiarowej siatki punktów

Politechnika Wrocławska

Biblioteka scipy

scipy_01.py

```
# -*- 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)
```

Politechnika Wrocławska

Biblioteka scipy

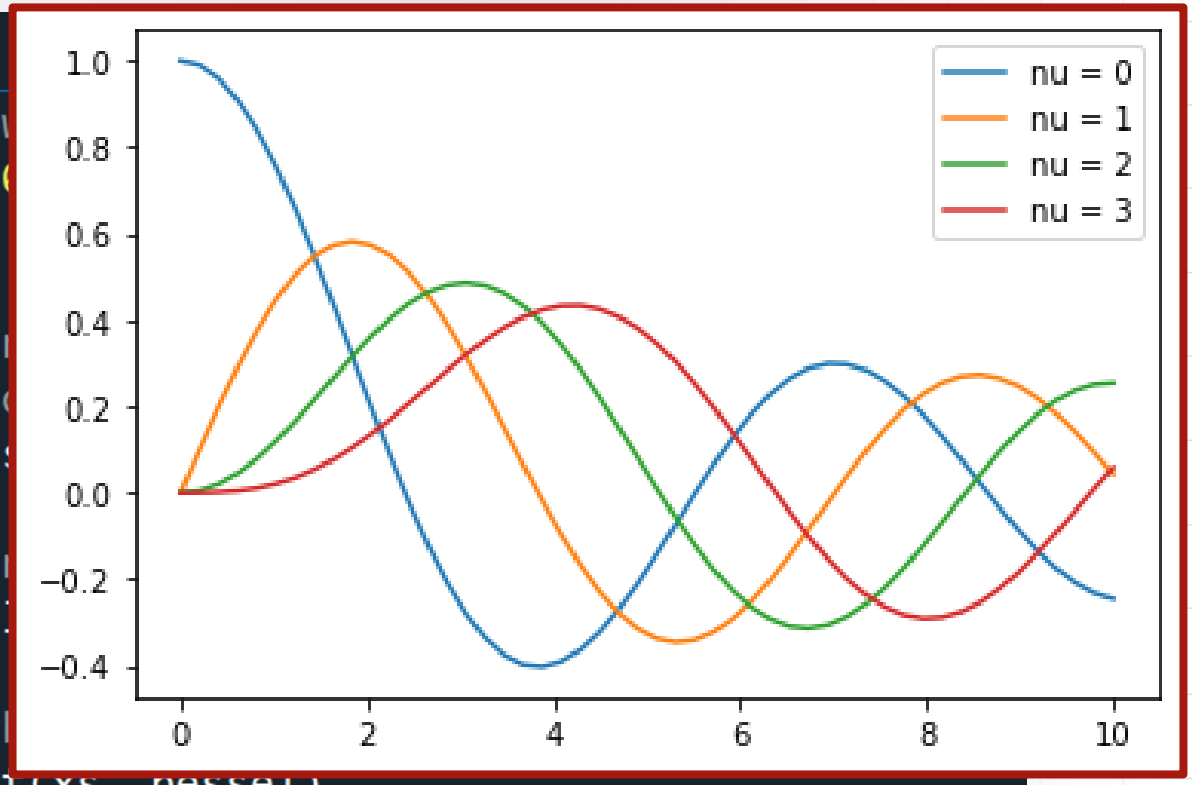
scipy_01.py

```
16  # tablica rzędów funkcji Bessela
17  nu = np.array([0, 1, 2, 3])
18
19  # wykorzystanie funkcji meshgrid
20  # do wygenerowania dwuwymiarowych tablic:
21  # rzędów (nus) oraz argumentów (xs)
22  nus, xs = np.meshgrid(nu, x)
23
24  # obliczenie funkcji Bessela
25  bessel = special.jv(nus, xs)
26
27  # stworzenie wykresów
28  lines = plt.plot(xs, bessel)
29
30  # utworzenie legendy
31  plt.legend(lines,("nu = " + format(n) for n in nu))
32  #plt.legend(lines,(r"$\nu$ = " + f"{n}" for n in nu))
33  #plt.legend(lines,(f"$J_{n}$" for n in nu))
```

Biblioteka scipy

scipy_01.py

```
16  # tablica rzędó
17  nu = np.array([
18
19  # wykorzystanie
20  # do wygenerowa
21  # rzędów (nus)
22  nus, xs = np.me
23
24  # obliczenie fu
25  bessel = specia
26
27  # stworzenie wy
28  lines = plt.plot(xs, bessel)
29
30  # utworzenie legendy
31  plt.legend(lines,("nu = " + format(n) for n in nu))
32  #plt.legend(lines,(r"$\nu$ = " + f"{n}" for n in nu))
33  #plt.legend(lines,(f"$J_{n}$" for n in nu))
```



Politechnika Wrocławska

Biblioteka scipy

- Przykład pokazuje:
  - obliczanie wartości funkcji Bessela funkcją special.j0()
  - wykorzystanie funkcji special.jn_zeros() generującej sekwencję miejsc zerowych funkcji Bessela

Politechnika Wrocławska

Biblioteka scipy

scipy_02.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje użycie funkcji j0 oraz jn_zeros
4  z modułu special z biblioteki scipy
5  do obliczania wartości funkcji Bessela.
6  https://docs.scipy.org/doc/scipy/tutorial/special.html
7  """
8
9  import numpy as np
10 import matplotlib.pyplot as plt
11 from scipy import special
12
13 # skalowanie funkcji Bessela
14 def scaled_bessel(x, k):
15     a = special.jn_zeros(0, k)[-1]
16     return special.j0(a*x)
```

Politechnika Wrocławska

Biblioteka scipy

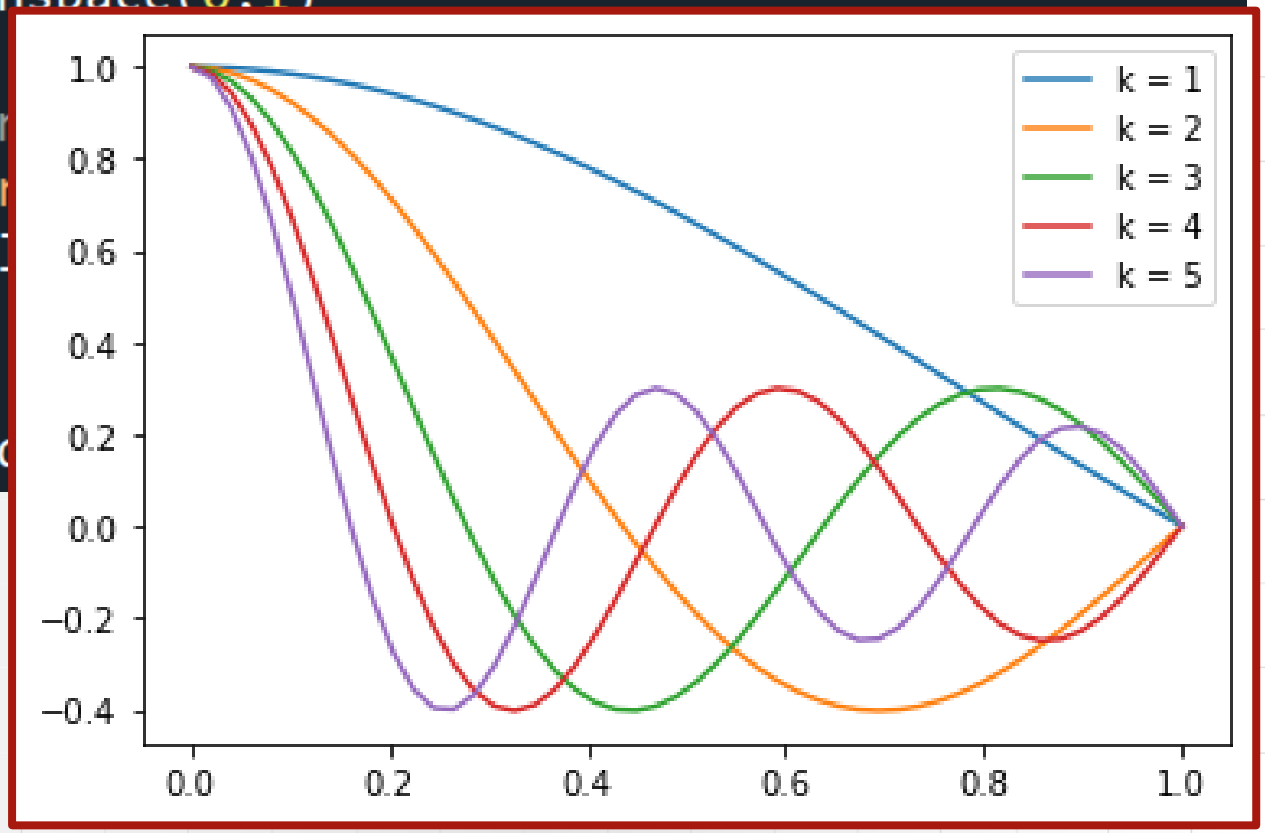
scipy_02.py

```
18  # utworzenie siatki punktów
19  # na odcinku jednostkowym
20  x = np.linspace(0,1)
21
22  # utworzenie wykresów skalowanych funkcji Bessela
23  for k in range(1, 6):
24      plt.plot(x, scaled_bessel(x, k),
25               label="k = " + format(k,"d"))
26
27  plt.legend()
```

Biblioteka scipy

scipy_02.py

```
18  # utworzenie siatki punktów
19  # na odcinku jednostkowym
20  x = np.linspace(0,1)
21
22  # utworzer
23  for k in
24      plt.pl
25
26
27  plt.legend
```



Politechnika Wrocławska

Biblioteka scipy

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

Politechnika Wrocławska

Biblioteka scipy

scipy_03.py

```
# -*- 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)
```

Politechnika Wrocławska

Biblioteka scipy

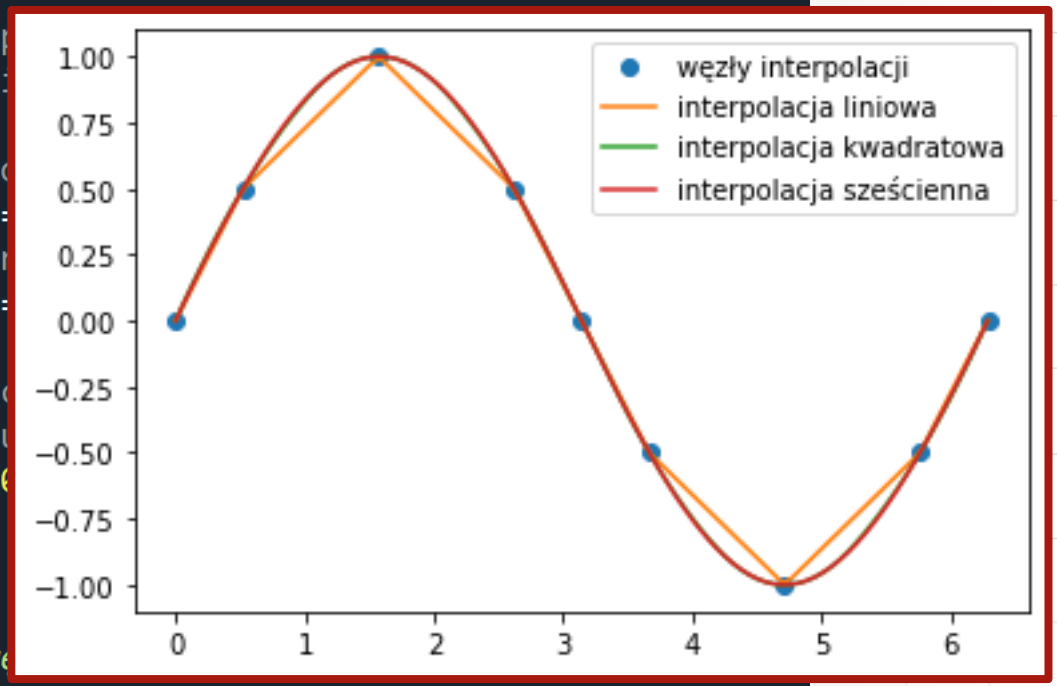
scipy_03.py

```
16  # dane węzłów interpolacji
17  x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
18  y = np.sin(x)
19
20  # wyznaczenie funkcji interpolujących
21  # domyślnie liniowej funkcji sklejanej
22  f_lin = interp1d(x, y)
23  # funkcji sklejanej kwadratowej
24  f_qua = interp1d(x, y, kind="quadratic")
25  # funkcji sklejanej sześciennej
26  f_cub = interp1d(x, y, kind="cubic")
27
28  # siatka punktów, dla których będą obliczane
29  # wartości funkcji interpolujących
30  x_int = np.linspace(0, 2, 100)*np.pi
31
32  # utworzenie wykresów
33  fig, ax = plt.subplots()
34  ax.plot(x, y, 'o', label="węzły interpolacji")
35  ax.plot(x_int, f_lin(x_int), label="interpolacja liniowa")
36  ax.plot(x_int, f_qua(x_int), label="interpolacja kwadratowa")
37  ax.plot(x_int, f_cub(x_int), label="interpolacja sześcienna")
38  ax.legend()
```

Biblioteka scipy

scipy_03.py

```
16  # dane węzłów interpolacji
17  x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
18  y = np.sin(x)
19
20  # wyznaczenie funkcji interp
21  # domyślnie liniowej funkcji
22  f_lin = interp1d(x, y)
23  # funkcji sklejanej kwadrato
24  f_qua = interp1d(x, y, kind=
25  # funkcji sklejanej sześcien
26  f_cub = interp1d(x, y, kind=
27
28  # siatka punktów, dla którym
29  # wartości funkcji interpolu
30  x_int = np.linspace(0, 2, 10
31
32  # utworzenie wykresów
33  fig, ax = plt.subplots()
34  ax.plot(x, y, 'o', label="we
35  ax.plot(x_int, f_lin(x_int), label="interpolacja liniowa")
36  ax.plot(x_int, f_qua(x_int), label="interpolacja kwadratowa")
37  ax.plot(x_int, f_cub(x_int), label="interpolacja sześcienna")
38  ax.legend()
```



Politechnika Wrocławska

Biblioteka scipy

- Przykład pokazuje:
  - aproksymację danych z wykorzystaniem funkcji curve_fit (scipy.optimize)
  - zależność liniową (zapisaną jako funkcję anonimową) wykorzystano do wygenerowania danych
  - następnie do danych dodano szum
  - funkcją curve_fit znaleziono parametry dopasowania funkcji liniowej do danych z szumem

Politechnika Wrocławska

Biblioteka scipy

scipy_04.py

```
# -*- 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

Biblioteka scipy

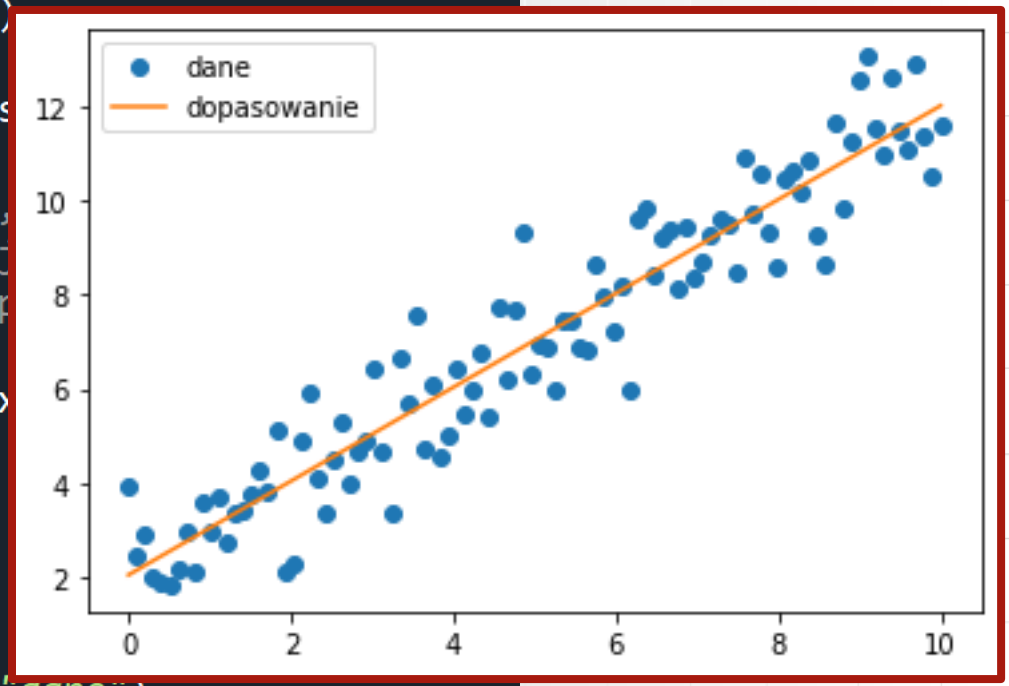
scipy_04.py

```
25  # utworzenie danych do aproksymacji
26  # dane obliczone z wykorzystaniem
27  # funkcji liniowej z dodanym szumem
28  x = np.linspace(0, 10, 100)
29  y = fun(x, 1, 2) # y = 1*x + 2
30  yn = y + np.random.normal(size=len(x))
31
32  # dopasowanie do danych (x, yn)
33  # funkcja curve_fit znajduje parametry
34  # funkcji fun, które najlepiej dopasowywują ją
35  # do danych wejściowych
36  p_opt, _ = curve_fit(fun, x, yn)
37  print(p_opt)
38
39  # utworzenie wykresów
40  fig, ax = plt.subplots()
41  # danych wejściowych
42  ax.plot(x, yn, 'o', label="dane")
43  # danych dopasowanych
44  ax.plot(x, fun(x, *p_opt), label="dopasowanie")
45  ax.legend()
```

Biblioteka scipy

scipy_04.py

```
25  # utworzenie danych do aproksymacji
26  # dane obliczone z wykorzystaniem
27  # funkcji liniowej z dodanym szumem
28  x = np.linspace(0, 10, 100)
29  y = fun(x, 1, 2) # y = 1*x
30  yn = y + np.random.normal(s
31
32  # dopasowanie do danych (x,
33  # funkcja curve_fit znajdu
34  # funkcji fun, które najlep
35  # do danych wejściowych
36  p_opt, _ = curve_fit(fun, x
37  print(p_opt)
38
39  # utworzenie wykresów
40  fig, ax = plt.subplots()
41  # danych wejściowych
42  ax.plot(x, yn, 'o', label="dane")
43  # danych dopasowanych
44  ax.plot(x, fun(x, *p_opt), label="dopasowanie")
45  ax.legend()
```



Politechnika Wrocławska

Biblioteka scipy

- Przykład pokazuje:
  - numeryczne rozwiązanie zagadnienia początkowego z wykorzystaniem funkcji odeint (scipy.integrate)

Politechnika Wrocławska

Biblioteka scipy

scipy_05.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący rozwiązywanie zagadnienia
4  początkowego z użyciem funkcji odeint
5  z modułu integrate biblioteki scipy.
6  https://docs.scipy.org/doc/scipy/reference/generated/
7  """
8
9  import numpy as np
10 import matplotlib.pyplot as plt
11 from scipy.integrate import odeint
12
13 # prawa strona równania różniczkowego
14 # dg/dt = f(g, t)
15 # f(g, t) = -g/tau
16 tau = 1.
17 f = lambda g, t : -g/tau
```

Politechnika Wrocławska

Biblioteka scipy

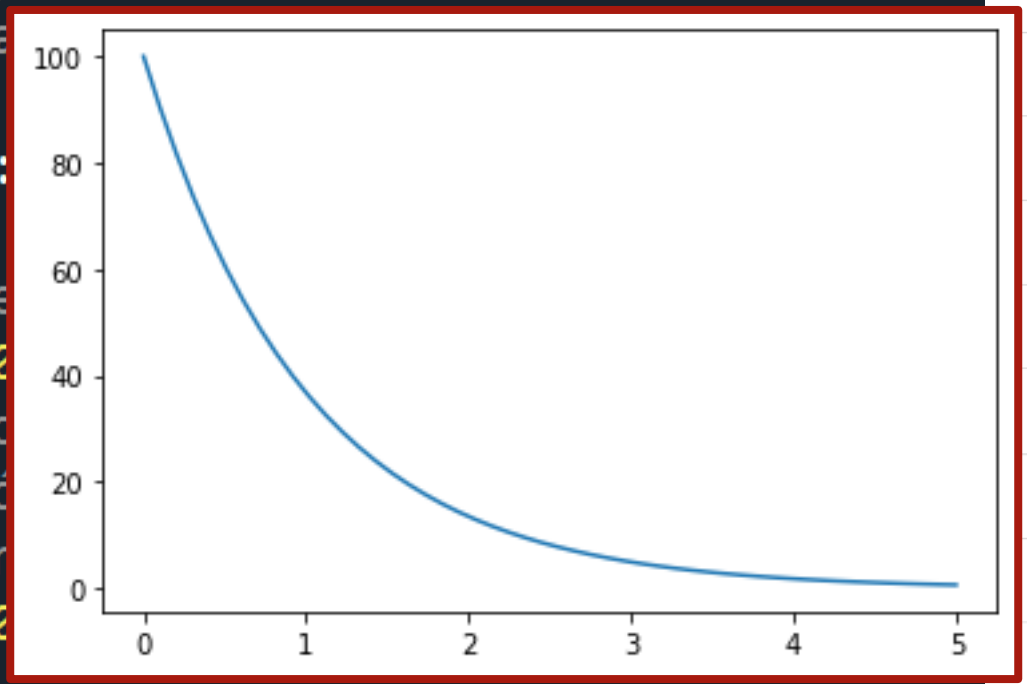
scipy_05.py

```
13  # prawa strona równania różniczkowego
14  # dg/dt = f(g, t)
15  # f(g, t) = -g/tau
16  tau = 1.
17  f = lambda g, t : -g/tau
18
19  # tablica zmiennej niezależnej
20  t = np.linspace(0, 5)
21  # obliczone wartoci zmiennej zależnej
22  # dla podanego równania różniczkowego (f)
23  # z zadanym warunkiem początkowym (g(0)=100)
24  gs = odeint(f,100,t)
25
26  # wykres obliczonego rozwiązania
27  plt.plot(t,gs)
```

Biblioteka scipy

scipy_05.py

```
13  # prawa strona równania różniczkowego
14  # dg/dt = f(g, t)
15  # f(g, t) = -g/ta
16  tau = 1.
17  f = lambda g, t :
18
19  # tablica zmienne
20  t = np.linspace(0
21  # obliczone warto
22  # dla podanego ró
23  # z zadanym warur
24  gs = odeint(f,100
25
26  # wykres obliczonego rozwiązania
27  plt.plot(t,gs)
```



Politechnika Wrocławska

Dziedziczenie

- Mechanizm dziedziczenia pozwala uprościć proces tworzenia różnych klas, jeśli można dla nich wyróżnić pewne wspólne cechy
- Klasa pochodna (nazywana podklasą) jest tworzona na podstawie klasy bazowej (nadklasy)

Politechnika Wrocławska

Dziedziczenie

- Prosty przykład pokazuje utworzenie dwóch klas pochodnych reprezentujących ptaki oraz ssaki dziedziczących po klasie bazowej reprezentującej zwierzęta

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Prosty przykład ilustrujący tworzenie hierarchii klas.
4  """
5
6  # definicja klasy bazowej reprezentującej zwierzę
7  class Animal:
8      # klasa implementuje konstruktor
9      # jedynym atrybutem obiektu jest m (masa)
10     def __init__(self, m):
11         self.m    = m
12
13     # klasa implemenutuje metodę moves(),
14     # która wyświetla ogólną informację o zwierzętach
15     def moves(self):
16         print("Zwierzęta się przemieszczają.")
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```
18  # definicja klasy pochodnej
19  class Bird(Animal):
20      # klasa pochodna ma wszystkie cechy klasy bazowej,
21      # a dodatkowo implementuje metodę flights(),
22      # która wyświetla informację o ptakach
23      def flights(self):
24          print("Ptaki latają.")
25
26  # definicja klasy pochodnej
27  class Mammal(Animal):
28      # klasa pochodna implementuje metodę runs(),
29      # która wyświetla informację o ssakach
30      def runs(self):
31          print("Ssaki biegają.")
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```
33  # utworzenie obiektu klasy Animal
34  a = Animal(1.)
35  print(a.m)
36  a.moves()
37
38  # klasa Bird dziedziczy konstruktor po klasie Animal
39  b = Bird(0.2)
40  print(b.m)
41  # obiekt tej klasy ma dodatkową metodę flights()
42  b.flights()
43
44  # klasa Mammal dziedziczy konstruktor po klasie Animal
45  m = Mammal(1000)
46  print(m.m)
47  # obiekt tej klasy ma dodatkową metodę runs()
48  m.runs()
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```
33  # utworzenie obiektu klasy Animal
34  a = Animal(1.)
35  print(a.m)
36  a.moves()
37
38  # klasa Bird dziedz
39  b = Bird(0.2)
40  print(b.m)
41  # obiekt tej klasy
42  b.flights()
43
44  # klasa Mammal dzie
45  m = Mammal(1000)
46  print(m.m)
47  # obiekt tej klasy ma dodatkową metodę runs()
48  m.runs()
```

```
1.0
Zwierzęta się przemieszczają.
0.2
Ptaki latają.
1000
Ssaki biegają.
```

Politechnika Wrocławska

Dziedziczenie

- Przykład pokazuje tworzenie hierarchii klas
- Klasy pochodne przeciążają metodę klasy bazowej

- Pozwala to uzyskać polimorfizm metody moves(), której działanie zależy od klasy obiektu dla którego będzie wywołana

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py dziedziczenie02.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Prosty przykład ilustrujący tworzenie hierarchii klas.
4  Klasy pochodne przeciążają metodę klasy bazowej.
5  """
6
7  # definicja klasy bazowej reprezentującej zwierzę
8  class Animal:
9      def __init__(self, m):
10         self.m     = m
11
12     # klasa implemenutuje metodę moves(),
13     # która będzie przeciążana przez klasy pochodne
14     def moves(self):
15         print("Zwierzęta się przemieszczają.")
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py dziedziczenie02.py

```
17  class Bird(Animal):
18      # klasa Bird przeciąża metodę moves()
19      # pochodzącą z klasy bazowej
20      def moves(self):
21          print("Ptaki latają.")
22
23  class Mammal(Animal):
24      # klasa Mammal również przeciąża metodę moves()
25      # pochodzącą z klasy bazowej
26      def moves(self):
27          print("Ssaki biegają.")
28
29  # klasa Reptiles nie przeciąża metody moves()
30  class Reptiles(Animal):
31      pass
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py dziedziczenie02.py

```
33  a = Animal(1.)
34  b = Bird(0.2)
35  m = Mammal(1000)
36  r = Reptiles(300)
37  # obiekty różnych klas zebranej na wspólnej liście
38  animals = [a, b, m, r]
39
40  # wywołanie metody moves() uruchomi różne wersje tej metody
41  # w zależności od klasy obiektu
42  for z in animals:
43      print(z.m)
44      z.moves()
```

Dziedziczenie

dziedziczenie01.py | dziedziczenie02.py

```
33  a = Animal(1.)
34  b = Bird(0.2)
35  m = Mammal(1000)
36  r = Reptiles(300)
37  # obiekty różnych klas zebranej na wspólnej liście
38  animals = [a, b, m, r]
39
40  # wywołanie metody moves() uruchomi różne wersje tej metody
41  # w zależności od klasy obiektu
```

wywołanie dla klasy Animal

wywołanie dla klasy Bird

wywołanie dla klasy Mammal

wywołanie dla klasy Reptile (implementacja w klasie Animal)

```
1.0
Zwierzęta się przemieszczają.
0.2
Ptaki latają.
1000
Ssaki biegają.
300
Zwierzęta się przemieszczają.
```

Politechnika Wrocławska

Podsumowanie

- Kilka wybranych funkcji biblioteki scipy
  - interpolacja
  - aproksymacja
  - rozwiązywanie zagadnienia początkowego
- Tworzenie klas z wykorzystaniem dziedziczenia
- Polimorfizm klas