

Programowanie obiektowe

INP001045L

rok akademicki 2022/23

semestr zimowy

Laboratorium 10

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220



Plan wykładu

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



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)



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



Biblioteka scipy

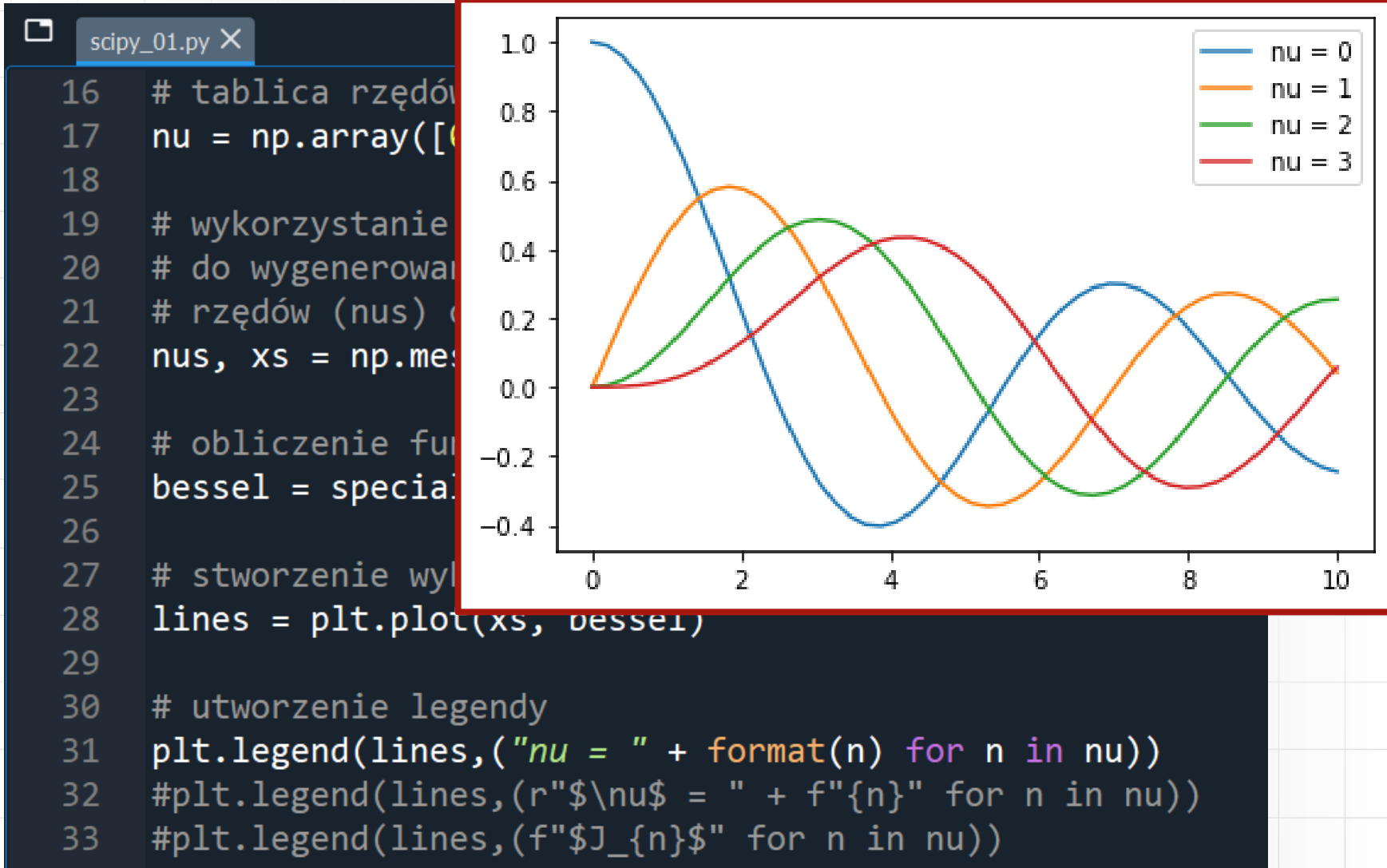
```
scipy_01.py ×  
1 # -*- coding: utf-8 -*-  
2 """  
3 Przykład pokazuje użycie funkcji jv  
4 z modułu special z biblioteki scipy  
5 do obliczania wartości funkcji Bessela.  
6 https://docs.scipy.org/doc/scipy/tutorial/special.html  
7 """  
8  
9 import numpy as np  
10 import matplotlib.pyplot as plt  
11 from scipy import special  
12  
13 # siatka argumentów  
14 x = np.linspace(0,10)  
15  
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)
```



Biblioteka scipy

```
scipy_01.py X
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





Biblioteka scipy

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



Biblioteka scipy



scipy_02.py X

```
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje użycie funkcji  $j_0$  oraz  $j_n\_zeros$ 
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)
```



Biblioteka scipy

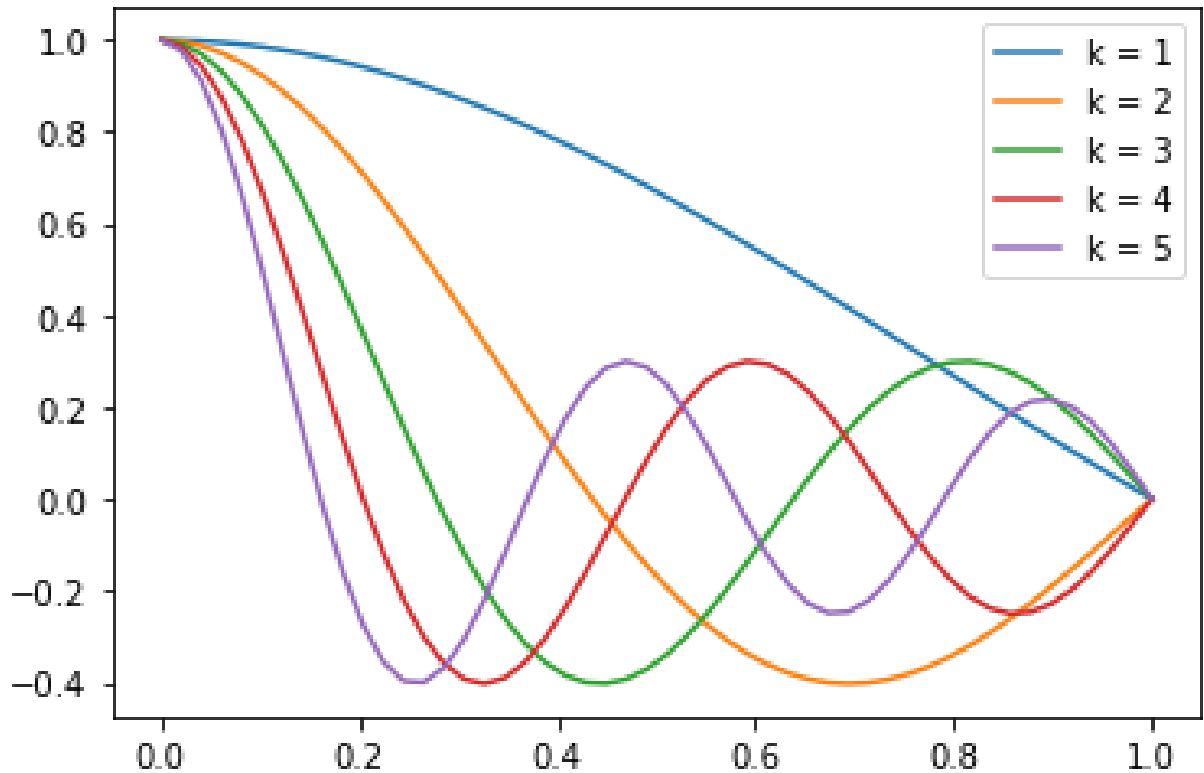


scipy_02.py X

```
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 X  
18 # utworzenie siatki punktów  
19 # na odcinku jednostkowym  
20 x = np.linspace(0,1)  
21  
22 # utworzenie  
23 for k in range(1,6):  
24     plt.plot(x, np.cos(k*np.pi*x))  
25  
26  
27 plt.legend()
```





Biblioteka scipy

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



Biblioteka scipy

```
scipy_03.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący interpolację danych.
4  Znane wartości funkcji sinus w węzłach
5  są interpolowane z funkcji interp1d
6  z modułu interpolate biblioteki scipy.
7  Funkcja interp1d pozwala wyznaczyć
8  funkcję interpolującą.
9  https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html
10 """
11
12 import numpy as np
13 import matplotlib.pyplot as plt
14 from scipy.interpolate import interp1d
15
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)
```



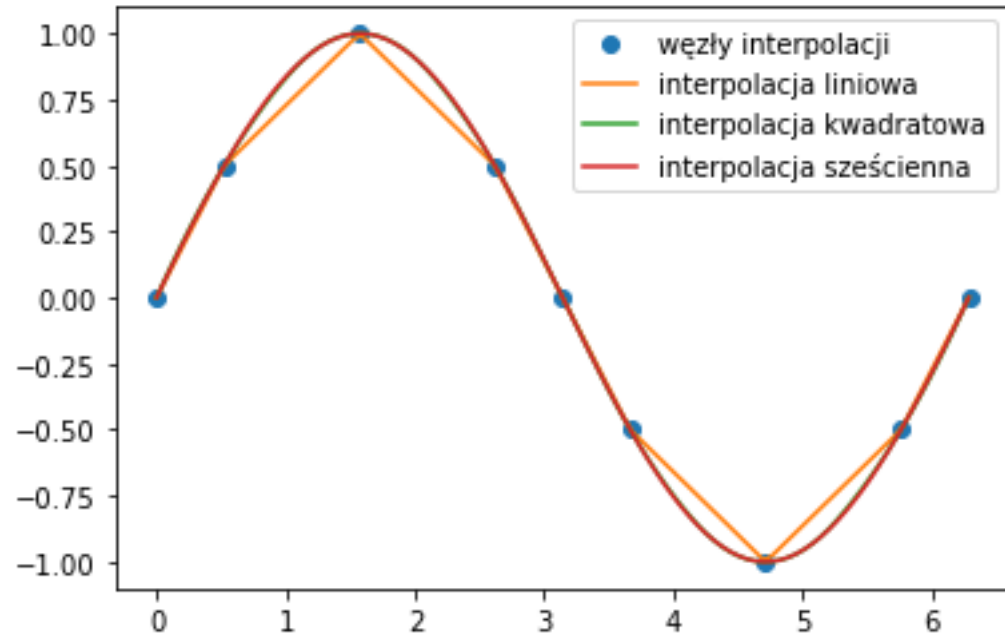
Biblioteka scipy

```
scipy_03.py X
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 sklejaney
22 f_lin = interp1d(x, y)
23 # funkcji sklejaney kwadratowej
24 f_qua = interp1d(x, y, kind="quadratic")
25 # funkcji sklejaney 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 X

```
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 sklejaney kwadratowej
24 f_qua = interp1d(x, y, kind='quadratic')
25 # funkcji sklejaney sześciennych
26 f_cub = interp1d(x, y, kind='cubic')
27
28 # siatka punktów, dla których
29 # wartości funkcji interpolacji
30 x_int = np.linspace(0, 2, 10)
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

- 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



Biblioteka scipy

```
scipy_04.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący aproksymację danych
4  funkcją liniową.
5  Dopasowanie wyznaczone z użyciem
6  funkcji curve_fit z modułu optimize
7  biblioteki scipy.
8  https://docs.scipy.org/doc/scipy/reference/genera
9  """
10
11  import numpy as np
12  import matplotlib.pyplot as plt
13  from scipy.optimize import curve_fit
14
15  # definicja funkcji liniowej
16  # x - argument
17  # a - współczynnik kierunkowy
18  # b - wyraz wolny
19  #def fun(x, a, b):
20  #     return a*x + b
21  # równoważna definicja z wykorzystaniem
22  # funkcji anonimowej
23  fun = lambda x, a, b : a*x + b
```

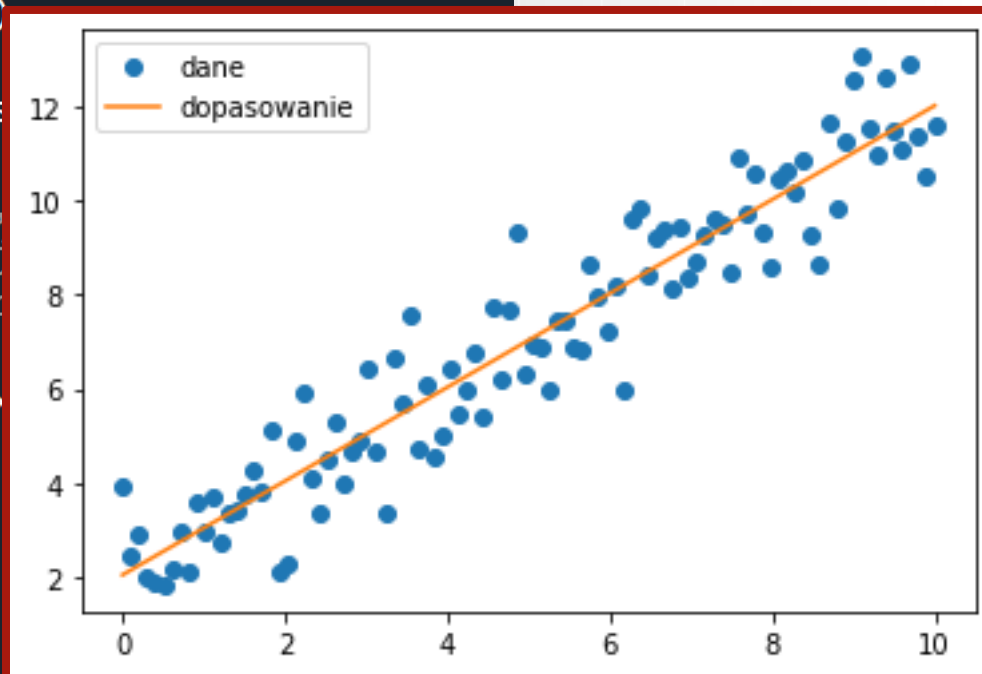


Biblioteka scipy

```
scipy_04.py X
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 dopasowują 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 X
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 najle
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()
```





Biblioteka scipy

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



Biblioteka scipy



scipy_05.py X

```
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
```



Biblioteka scipy



scipy_05.py X

```
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 wartości zmiennej zależnej
22 # dla podanego równania różniczkowego (f)
23 # z zadany warunkiem początkowym ( $g(0)=100$ )
24 gs = odeint(f, 100, t)
25
26 # wykres obliczonego rozwiązania
27 plt.plot(t, gs)
```

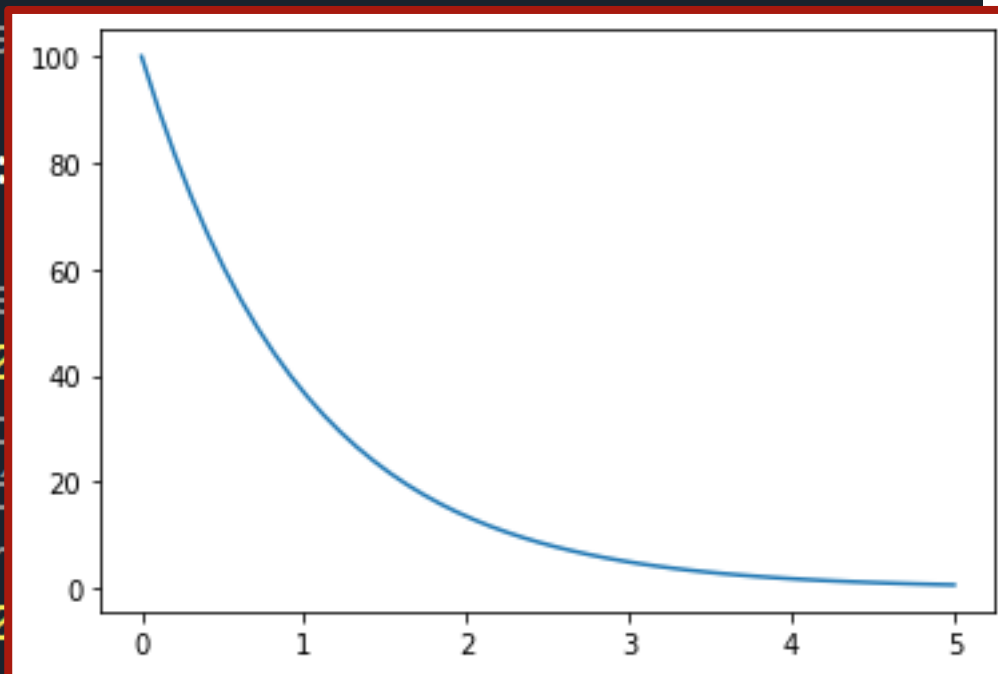


Biblioteka scipy



scipy_05.py X

```
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 :
18
19 # tablica zmienne
20 t = np.linspace(0, 5, 100)
21 # obliczone wartosc
22 # dla podanego równania
23 # z zadany warunk
24 gs = odeint(f, 100, t)
25
26 # wykres obliczonego rozwiązania
27 plt.plot(t, gs)
```





Podsumowanie

- Kilka wybranych funkcji biblioteki scipy
 - interpolacja
 - aproksymacja
 - rozwiązywanie zagadnienia początkowego