

Programowanie obliczeń komputerowych

W11IKW-SI0106W, W11IKW-SI0106L
rok akademicki 2024/25
semestr zimowy

Wykład 6

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 221

Plan

Biblioteka scipy

- funkcje specjalne
- interpolacja
- aproksymacja
- rozwiązywanie zagadnienia początkowego
- rozwiązywanie zagadnienia brzegowego

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)

Funkcje specjalne

Przykład pokazuje:

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

```
scipy_01.py X
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)
```

Funkcje specjalne

Przykład pokazuje:

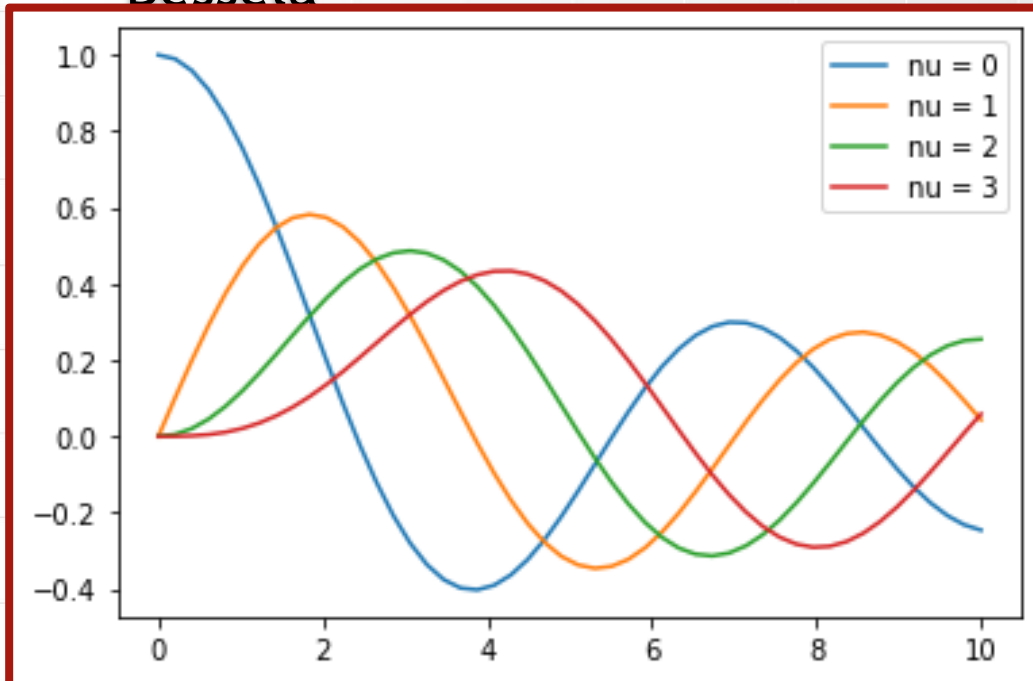
- importowanie modułu special biblioteki scipy
- obliczanie wartości funkcji Bessela
- wykorzystanie polecenia meshgrid do wygenerowania dwuwymiarowej siatki punktów

```
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))
```

Funkcje specjalne

Przykład pokazuje:

- importowanie modułu special biblioteki scipy
- obliczanie wartości funkcji Bessela



```
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))
```

Funkcje specjalne

Przykład pokazuje:

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

```
scipy_02.py X
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)
```

Funkcje specjalne

Przykład pokazuje:

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

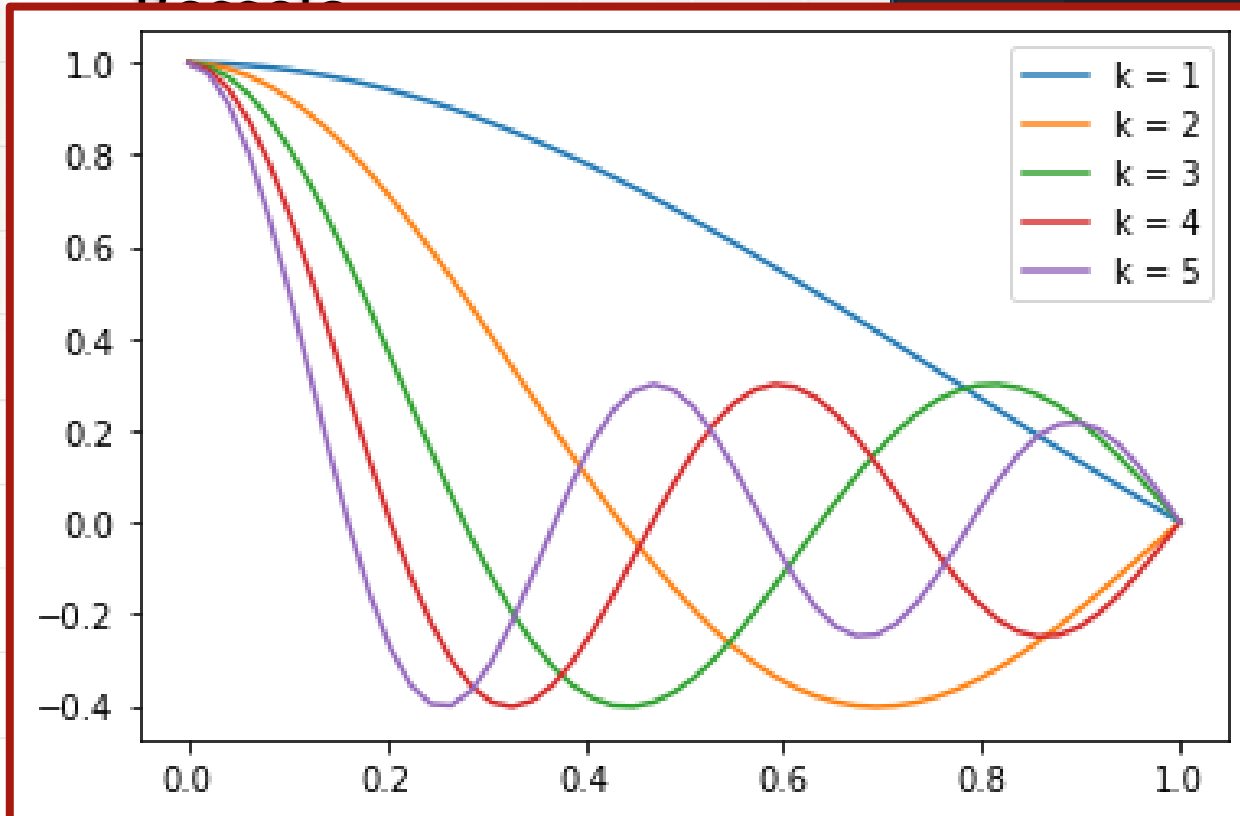
```
scipy_02.py X
18 # utworzenie siatki punktów
19 # na odcinku jednostkowym
20 x = np.linspace(0,1)
21
22 # utworzenie wykresów skalowanych funkcji Bessela
23 for k in range(1, 6):
24     plt.plot(x, scaled_bessel(x, k),
25             label="k = " + format(k, "d"))
26
27 plt.legend()
```


Funkcje specjalne

Przykład pokazuje:

- obliczanie funkcji

```
scipy_02.py X
18 # utworzenie siatki punktów
19 # na odcinku jednostkowym
20 x = np.linspace(0,1)
21
...
for k in range(1, 6):
    plot(x, scaled_bessel(x, k),
         label="k = " + format(k, "d"))
end()
```



Interpolacja

Przykład pokazuje:

- interpolację danych funkcją `interp`
- funkcja `interp` zwraca tablicę wartości interpolowanych dla podanej tablicy argumentów na podstawie dostarczonych danych
- klasa `CubicSpline` (`scipy.interpolate`) pozwala stworzyć obiekt reprezentujący interpolujący wielomian sklepany

```
scipy_03a.py X scipy_04.py X scipy_05.py X scipy_06.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący interpolację danych.
4  Znane wartości funkcji sinus w węzłach
5  są interpolowane funkcją interp z biblioteki scipy
6  oraz klasą CubicSpline z modułu scipy.interpolate
7  https://docs.scipy.org/doc/scipy/tutorial/interpolate
8  """
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from scipy.interpolate import CubicSpline
13
14 # dane węzłów interpolacji
15 x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
16 y = np.sin(x)
```

Interpolacja

Przykład pokazuje:

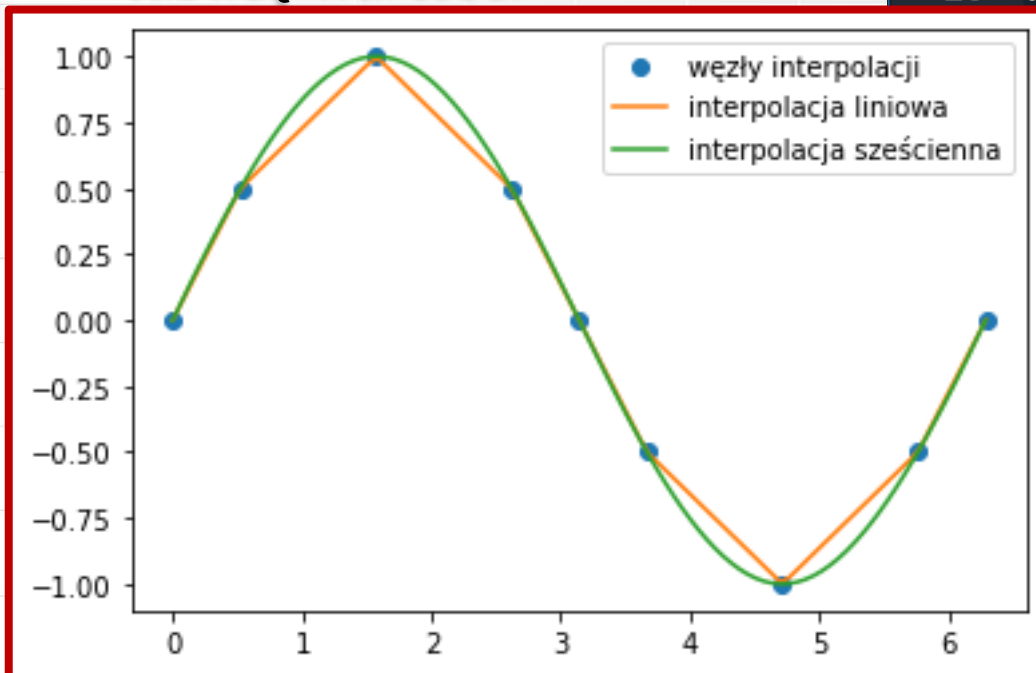
- interpolację danych funkcją `interp`
- funkcja `interp` zwraca tablicę wartości interpolowanych dla podanej tablicy argumentów na podstawie dostarczonych danych
- klasa `CubicSpline` (`scipy.interpolate`) pozwala stworzyć obiekt reprezentujący interpolujący wielomian sklepany

```
scipy_03a.py X scipy_04.py X scipy_05.py X scipy_06.py X
13
14 # dane węzłów interpolacji
15 x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
16 y = np.sin(x)
17
18 # siatka punktów, dla których będą obliczane
19 # wartości funkcji interpolujących
20 x_int = np.linspace(0, 2, 100)*np.pi
21
22 # wyznaczenie funkcji interpolujących
23 # domyślnie liniowej funkcji sklepanej
24 y_int = np.interp(x_int, x, y)
25 # funkcji sklepanej sześciennej
26 spl = CubicSpline(x, y)
27
28
29 # utworzenie wykresów
30 fig, ax = plt.subplots()
31 ax.plot(x, y, 'o', label="węzły interpolacji")
32 ax.plot(x_int, y_int, label="interpolacja liniowa")
33 ax.plot(x_int, spl(x_int), label="interpolacja sześcienna")
34 ax.legend()
35
```

Interpolacja

Przykład pokazuje:

- interpolację danych funkcją `interp`
- funkcja `interp` zwraca tablicę wartości



```
scipy_03a.py X scipy_04.py X scipy_05.py X scipy_06.py X
13
14 # dane węzłów interpolacji
15 x = np.array([0, 1, 3, 5, 6, 7, 9, 11, 12])/6*np.pi
16 y = np.sin(x)
17
18 # siatka punktów, dla których będą obliczane
19 # wartości funkcji interpolujących
20 x_int = np.linspace(0, 2, 100)*np.pi

wyznaczenie funkcji interpolujących
domyślnie liniowej funkcji sklejanej
_int = np.interp(x_int, x, y)
funkcji sklejanej sześciennej
pl = CubicSpline(x, y)

utworzenie wykresów
fig, ax = plt.subplots()
x.plot(x, y, 'o', label="węzły interpolacji")
x.plot(x_int, y_int, label="interpolacja liniowa")
x.plot(x_int, spl(x_int), label="interpolacja sześcienna")
x.legend()
```



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 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
```

Aproksymacja

Przykład pokazuje:

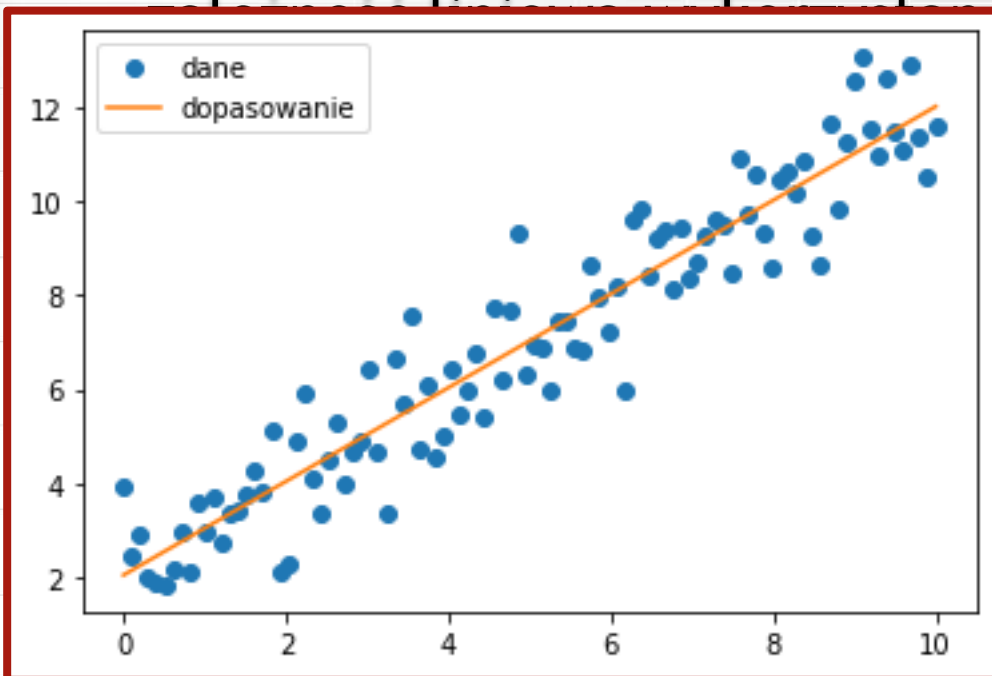
- aproksymację danych - `curve_fit` (`scipy.optimize`)
- zależność liniową wykorzystano do wygenerowania danych
- następnie do danych dodano szum
- funkcją `curve_fit` znaleziono parametry dopasowania funkcji liniowej do danych z szumem

```
scipy_04.py 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()
```

Aproksymacja

Przykład pokazuje:

- aproksymację danych - `curve_fit` (`scipy.optimize`)



```
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()
```

Zagadnienie początkowe

Przykład pokazuje:

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

```
scipy_05.py X scipy_06.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący rozwiązywanie zagadnienia
4  początkowego z użyciem funkcji solve_ivp
5  z modułu integrate biblioteki scipy.
6  https://docs.scipy.org/doc/scipy/reference/gener
7  """
8
9  import numpy as np
10 import matplotlib.pyplot as plt
11 from scipy.integrate import solve_ivp
12
13 # prawa strona równania różniczkowego
14 # dg/dt = f(g, t)
15 # f(t, g) = -g/tau
16 tau = 1
17 f = lambda t, g : -g/tau
```


Zagadnienie początkowe

Przykład pokazuje:

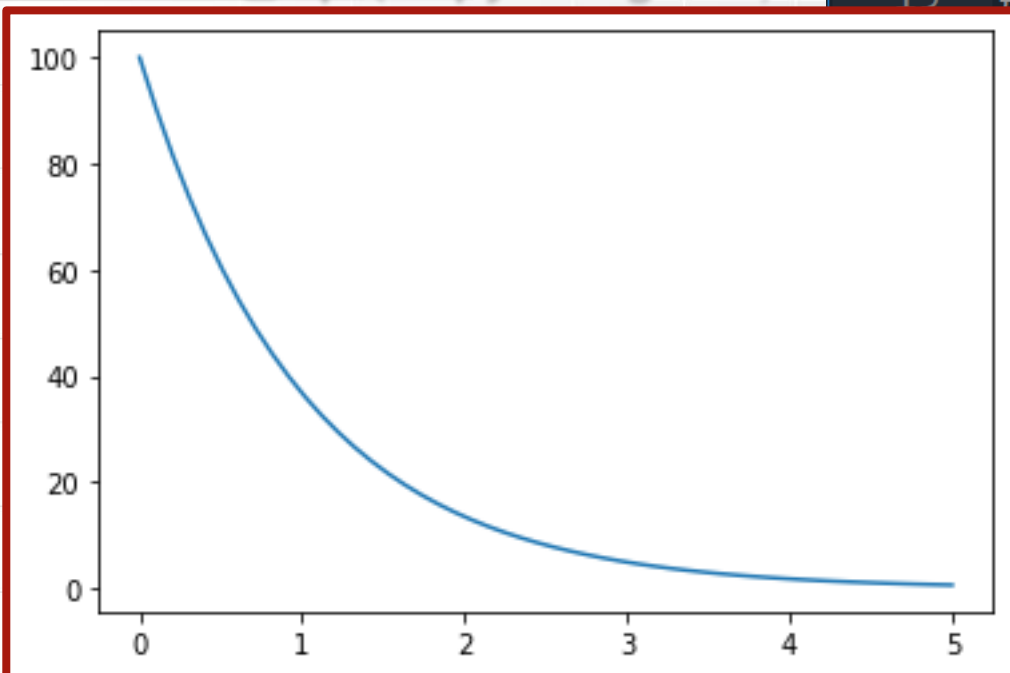
- numeryczne rozwiązanie zagadnienia początkowego z wykorzystaniem funkcji `solve_ivp` (scipy.integrate)

```
scipy_05.py X
13 # prawa strona równania różniczkowego
14 # dg/dt = f(g, t)
15 # f(t, g) = -g/tau
16 tau = 1.
17 f = lambda t, g : -g/tau
18
19 # zakres zmiennej niezależnej
20 t = np.array([0, 5])
21 # wektor zmiennej niezależnej
22 t_eval = np.linspace(*t)
23 # obiekt sol zawieta obliczone rozwiązanie
24 # dla podanego równania różniczkowego (f)
25 # z zadany warunkiem początkowym (g(0)=100)
26 # dla podanych wartości zmiennej niezależnej t_eval
27 sol = solve_ivp(f, t, [100], t_eval = t_eval)
28
29 # wykres obliczonego rozwiązania
30 plt.plot(sol.t, sol.y[0])
31
```

Zagadnienie początkowe

Przykład pokazuje:

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



```
scipy_05.py X
13 # prawa strona równania różniczkowego
14 # dg/dt = f(g, t)
15 # f(t, g) = -g/tau
16 tau = 1.
17 f = lambda t, g : -g/tau
18
19 # zakres zmiennej niezależnej
    = np.array([0, 5])
    wektor zmiennej niezależnej
    t_eval = np.linspace(*t)
    obiekt sol zawieta obliczone rozwiązanie
    dla podanego równania różniczkowego (f)
    z zadany 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 brzegowe

Przykład pokazuje:

- numeryczne rozwiązanie zagadnienia brzegowego z wykorzystaniem funkcji `solve_bvp` (`scipy.integrate`)

```
scipy_06.py* X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazujący rozwiązywanie zagadnienia
4  początkowego z użyciem funkcji solve_bvp
5  z modułu integrate biblioteki scipy.
6  https://docs.scipy.org/doc/scipy/reference/generat
7  """
8
9  import numpy as np
10 from scipy.integrate import solve_bvp
11 import matplotlib.pyplot as plt
12
13 # Definicja równania różniczkowego
14 def ode(x, y, p):
15     """
16     Układ równań różniczkowych przekształcony
17     na postać równań pierwszego rzędu.
18     y[0] = u, y[1] = du/dx
19     """
20     k = p[0] # Parametr (k)
21     return np.vstack((y[1], -k**2 * y[0]))
22
```

Zagadnienie brzegowe

Przykład pokazuje:

- numeryczne rozwiązanie zagadnienia brzegowego z wykorzystaniem funkcji `solve_bvp` (`scipy.integrate`)

```
scipy_06.py X
13 # Definicja równania różniczkowego
14 def ode(x, y, p):
15     """
16     Układ równań różniczkowych przekształcony
17     na postać równań pierwszego rzędu.
18     y[0] = u, y[1] = du/dx
19     """
20     k = p[0] # Parametr (k)
21     return np.vstack((y[1], -k**2 * y[0]))
22
23 # Definicja warunków brzegowych
24 def bc(ya, yb, p):
25     """
26     Warunki brzegowe:
27     u(0) = 0
28     u(1) = 0
29     """
30     return np.array([ya[0], yb[0], ya[1] - p[0]])
31
```

Zagadnienie brzegowe

Przykład pokazuje:

- numeryczne rozwiązanie zagadnienia brzegowego z wykorzystaniem funkcji `solve_bvp` (`scipy.integrate`)

```
scipy_06.py X
31
32 # Siatka początkowa (przedział x od 0 do 1)
33 x = np.linspace(0, 1, 100)
34
35 # Zgadywanie początkowe dla funkcji u(x) i jej pochodnej du/dx
36 y_guess = np.vstack((x, np.ones_like(x))) # y[0] = x, y[1] = du/dx
37
38 # Początkowe zgadywanie dla wartości k
39 k_guess = 9.0
40
41 # Rozwiązanie za pomocą solve_bvp
42 sol = solve_bvp(ode, bc, x, y_guess, p=[k_guess])
43
```

Zagadnienie brzegowe

Przykład pokazuje:

- numeryczne rozwiązanie zagadnienia brzegowego z wykorzystaniem funkcji `solve_bvp` (`scipy.integrate`)

```
41 # Rozwiązanie za pomocą solve_bvp
42 sol = solve_bvp(ode, bc, x, y_guess, p=[k_guess])
43
44 # Sprawdzenie sukcesu
45 if sol.success:
46     print(f"Rozwiązanie zakończone sukcesem. Znaleziona k: {sol.p[0]}")
47 else:
48     print("Rozwiązanie nie powiodło się.")
49
50 # Wizualizacja rozwiązania
51 plt.plot(sol.x, sol.y[0], label=f"u(x), k = {sol.p[0]:.4f}")
52 plt.title("Rozwiązanie równania różniczkowego")
53 plt.xlabel("x")
54 plt.ylabel("u(x)")
55 plt.axhline(0, color="black", linewidth=0.8, linestyle="--")
56 plt.legend()
57 plt.grid(True)
58 plt.show()
```

Podsumowanie

Wybrane funkcje biblioteki scipy

- interpolacja
- aproksymacja
- rozwiązywanie zagadnienia początkowego
- rozwiązywanie zagadnienia brzegowego