Wizualizacja danych
semestr letni 2024

Dr Anna Muranova
UWM w Olsztynie

Wykład 10

Biblioteka SciPy

SciPy w Pythonie to biblioteka typu open source służąca do rozwiązywania problemów matematycznych, naukowych, inżynieryjnych i technicznych. Umożliwia użytkownikom manipulowanie danymi i wizualizację danych przy użyciu szerokiej gamy poleceń języka Python wysokiego poziomu. SciPy jest zbudowany na rozszerzeniu Pythona NumPy.
Nie ma dużo sensu w importowaniu

```
import scipy
```

ponieważ tam będzie to samo, co w NumPy. Trzeba importować (moduly) podbiblioteki.
`https://scipy.org/`
`https://docs.scipy.org/doc/scipy/index.html#`

SciPy constants

```
from scipy import constants

print(constants.pi)
```

Lista wszystkich stałych:

```
from scipy import constants

print(dir(constants))
```

https://docs.scipy.org/doc/scipy/reference/constants.html

Przedrostki metryczne (SI)

Zwraca określoną jednostkę w metrach (np. centi zwraca 0,01)

```
from scipy import constants

print(constants.yotta)    #1e+24
print(constants.zetta)    #1e+21
print(constants.exa)      #1e+18
print(constants.peta)     #1000000000000000.0
print(constants.tera)     #1000000000000.0
print(constants.giga)     #1000000000.0

print(constants.centi*169)
```

https://www.w3schools.com/python/scipy/scipy_constants.php

Przedrostki binarne (SI)

Zwraca określoną jednostkę w bajtach (np. kibi zwraca 1024)

```
from scipy import constants

print(constants.kibi)    #1024
print(constants.mebi)    #1048576
print(constants.gibi)    #1073741824
print(constants.tebi)    #1099511627776
print(constants.pebi)    #1125899906842624
print(constants.exbi)    #1152921504606846976
print(constants.zebi)    #1180591620717411303424
print(constants.yobi)    #1208925819614629174706176
```

https://www.w3schools.com/python/scipy/scipy_constants.php

Waga (SI)

Zwraca określoną jednostkę w kilogramach (np. gram zwraca 0.001)

```
from scipy import constants

from scipy import constants

print(constants.gram)          #0.001
print(constants.metric_ton)    #1000.0
print(constants.grain)         #6.479891e-05
print(constants.lb)            #0.4535923699999997
print(constants.pound)         #0.4535923699999997
print(constants.oz)            #0.02834952312499998
print(constants.ounce)         #0.02834952312499998
print(constants.stone)         #6.350293179999995
print(constants.long_ton)      #1016.0469088
```

https://www.w3schools.com/python/scipy/scipy_constants.php

Jeszcze:

`https://www.w3schools.com/python/scipy/scipy_constants.php`

- **Kąty**: zwraca określoną jednostkę w radianach
- **Czas**: zwraca określoną jednostkę w sekundach
- **Długość**: zwraca określoną jednostkę w metrach
- **Ciśnienie**: zwraca określoną jednostkę w paskalach
- **Obszar**: zwraca określoną jednostkę w metrach kwadratowych
- **Objętość**: zwraca określoną jednostkę w metrach kubicznych
- **Prędkość**: zwraca określoną jednostkę w metrach na sekundą
- **Temperatura**: zwraca określoną jednostkę w Kelvinach
- **Energia**: zwraca określoną jednostkę w Dżulach
- **Moc**: zwraca określoną jednostkę w wattach
- **Siła**: zwraca określoną jednostkę w Newtonach

Optymizacja: pierwiastki równania

Znaleźć pierwiastek równania $x + \cos(x) = 0$

```
from scipy.optimize import root
from math import cos

def eqn(x):
  return x + cos(x)

myroot = root(eqn, 0)

print(myroot)
print(myroot.x)
```

Optymizacja: pierwiastki równania

Porównaj wyniki:

```
from scipy.optimize import root

def eqn(x):
  return x**2-2

myroot = root(eqn, 0)
#myroot = root(eqn, 1)

print(myroot)
print(myroot.x)
```

Optymizacja: minimum, maximum

Aby zminimalizować funkcję, możemy użyć funkcji `scipy.optimize.minimize()`.
Funkcja `minimize()` przyjmuje następujące argumenty:

- `fun` – funkcja reprezentująca równanie.
- `x0` – wstępne przypuszczenie dotyczące pierwiastka
- `method` – nazwa metody, która ma zostać użyta. Możliwe wartości: 'CG', 'BFGS', 'Newton-CG', 'L-BFGS-B', 'TNC', 'COBYLA', 'SLSQP'.
- `callback` – funkcja wywoływana po każdej iteracji optymalizacji.
- `options` – słownik definiujący dodatkowe parametry:
  {
  "disp": boolean – wydrukuj szczegółowy opis
  "gtol": number – tolerancja błędu
  }

Optymizacja: minimum, maximum

Aby zminimalizować funkcję, możemy użyć funkcji `scipy.optimize.minimize()`.
Funkcja `minimize()` przyjmuje następujące argumenty:

```
from scipy.optimize import minimize

def eqn(x):
  return x**2 + x + 2

mymin = minimize(eqn, 0, method='BFGS')

print(mymin.x)
print(mymin)
```

Optymizacja: minimum, maximum

Aby zminimalizować funkcję, możemy użyć funkcji `scipy.optimize.minimize()`.
Funkcja `minimize()` przyjmuje następujące argumenty:

```
import scipy.optimize as optimize

def f(params):
    # print(params)  # <-- you'll see that params is a NumPy array
    a, b, c = params # <-- for readability you may wish to assign
names to the component variables
    return a**2 + b**2 + c**2

initial_guess = [1, 1, 1]
result = optimize.minimize(f, initial_guess)
if result.success:
    fitted_params = result.x
    print(fitted_params)
else:
    raise ValueError(result.message)
```

Macierz rzadka

Macierz rzadka – macierz, w której większość elementów ma wartość zero.
https://www.w3schools.com/python/scipy/scipy_sparse_data.php

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])

print(csr_matrix(arr))#tworzenie rzadkiej macierze
print(mat)
print(mat.toarray())
```

Istnieją przede wszystkim dwa typy rzadkich macierzy, których używamy:
CSC – skompresowana rzadka kolumna. Do wydajnej arytmetyki i szybkiego dzielenia kolumn.
CSR – skompresowany rzadki wiersz. Do szybkiego krojenia wiersze, szybsze produkty wektorów macierzowych

Metody dla rzadkich macierze (angl. sparse matrix, sparse data)

```
import numpy as np
from scipy.sparse import csr_matrix

arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])

print(csr_matrix(arr))#tworzenie rzadkiej macierze
print(csr_matrix(arr).data)
print(csr_matrix(arr).count_nonzero())
mat.eliminate_zeros()
print(mat)
print(mat.toarray())
newarr = csr_matrix(arr).tocsc()
print(newarr)
print(newarr.toarray())
```

`eleminate_zeros()` – zmienia podstawową reprezentację rzadkiej macierzy bez wpływu na jej logiczną zawartość. Usuwa jawnie zapisane zera z tablicy danych stanowiącej podstawę macierzy rzadkiej. Służy do zmniejszenia zużycia miejsca i przygotowania rzadkiej macierzy dla algorytmów, które zakładają, że nie będzie jawnie przechowywanych zer.

Grafy

Podbiblioteka: `scipy.sparse.csgraph`
Przykład:

```
from scipy.sparse.csgraph import connected_components
from scipy.sparse import csr_matrix

arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])#macierz sasiedstwa

newarr = csr_matrix(arr)

print(connected_components(newarr))
```

https://www.w3schools.com/python/scipy/scipy_graphs.php

Dane przestrzenne

Dane przestrzenne (ang. spatial data) – dane, powiązane z obiektami w przestrzeni, takimi jak: punkty, linie i wielokąty.
Przykład: otoczka wypukła, powłoka wypukła, uwypuklenie podzbioru przestrzeni liniowej – najmniejszy (w sensie inkluzji) zbiór wypukły zawierający ten podzbiór. Otoczkę wypukłą A oznacza się zwykle jako $\operatorname{conv} A$.

```
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt

points = np.array([[2, 4], [3, 4],  [3, 0],  [2, 2], [4, 1],
[1, 2],  [5, 0], [3, 1], [1, 2], [0, 2]])
hull = ConvexHull(points)
hull_points = hull.simplices

plt.scatter(points[:,0], points[:,1])
for simplex in hull_points:
  plt.plot(points[simplex,0], points[simplex,1], 'k-')
plt.show()
```

https://www.w3schools.com/python/scipy/scipy_spatial_data.php

Odległości

```
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import cityblock

p1 = (1, 0)
p2 = (10, 2)

res = euclidean(p1, p2)#odleglosc Eukledesowa
print(res)

res = cityblock(p1, p2)#odleglosc Manhattan
print(res)
```

Interpolacja

Interpolacja to metoda generowania punktów pomiędzy podanymi punktami. Na przykład: dla punktów 1 i 2 możemy interpolować i znaleźć punkty 1,33 i 1,66.
Interpolacja ma wiele zastosowań: w uczeniu maszynowym często mamy do czynienia z brakującymi danymi w zbiorze danych i wtedy stosuje się interpolację w celu zastąpienia tych wartości.
Ta metoda wypełniania wartości nazywa się imputacją.
Oprócz imputacji często stosuje się interpolację, gdy musimy wygładzić dyskretne punkty w zbiorze danych.

Interpolcja 1D

Funkcja`interp1d()` służy do interpolacji rozkładu z 1 zmienną.
Pobiera punkty x i y i zwraca wywoływalną funkcję, którą można wywołać z nowym x, i zwraca odpowiadające mu y.

```
from scipy.interpolate import interp1d
import numpy as np

x = np.arange(10)
y = 2*x + 1

interp_func = interp1d(x, y)

print(interp_func)
newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

Uwaga! Nowe wartości x powinny mieścić się w tym samym zakresie co stare x, co oznacza, że nie możemy wywołać funkcji `interp_func()` z wartościami większymi niż 10 lub mniejszymi niż 0.

Interpolacja funkcjami sklejanymi (spline)

W interpolacji 1D punkty dopasowywane są do pojedynczej krzywej, natomiast w interpolacji funkcjami sklejanymi punkty dopasowywane są do funkcji zdefiniowanej przedziałowo za pomocą wielomianów zwanych splajnami.
Funkcja zdefiniowana przedziałowo jest funkcją zbudowaną z kawałków różnych funkcji w różnych przedziałach
Funkcja `UnivariateSpline()` pobiera X i y i tworzy wywoływalną funkcję, którą można wywołać z nowym x.

Interpolacja funkcjami sklejanymi (spline)

```
from scipy.interpolate import UnivariateSpline
import numpy as np

x = np.arange(10)
y = x**2 + np.sin(x) + 1

interp_func = UnivariateSpline(x, y)

newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

Interpolacja funkcjami sklejanymi (spline) vs 1D

```
from scipy.interpolate import UnivariateSpline
from scipy.interpolate import interp1d
import numpy as np

x = np.arange(10)
y = x**2 + np.sin(x) + 1

interp_func = UnivariateSpline(x, y)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)

interp_func = interp1d(x, y)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)

xnew = np.arange(2.1, 3, 0.1)
print(xnew**2 + np.sin(xnew) + 1)
```

Interpolacja funkcjami sklejanymi (spline) vs 1D

```
from scipy.interpolate import UnivariateSpline
from scipy.interpolate import interp1d
import numpy as np

x = np.arange(10)
y = x**2 + np.sin(x) + 1

interp_func = UnivariateSpline(x, y)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)

interp_func = interp1d(x, y)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)

xnew = np.arange(2.1, 3, 0.1)
print(xnew**2 + np.sin(xnew) + 1)
```

Interpolacja funkcjami radialnymi

Radialna funkcja bazowa (ang. radial basis function – RBF) – funkcja rzeczywista, której wartość zależy zwykle wyłącznie od odległości od określonego punktu ($\rho(x - x_0) = \rho(\|x - x_0\|)$). Funkcje te są używane w funkcjach aproksymacji, dla ciągów prognoz czasowych i sterowania. Często są używane w sztucznych sieciach neuronowych.

```
from scipy.interpolate import Rbf
import numpy as np

x = np.arange(10)
y = x**2 + np.sin(x) + 1

interp_func = Rbf(x, y)

newarr = interp_func(np.arange(2.1, 3, 0.1))

print(newarr)
```

Porównanie

```
from scipy.interpolate import UnivariateSpline
from scipy.interpolate import interp1d
from scipy.interpolate import Rbf
import numpy as np

x = np.arange(10)
y = x**2 + np.sin(x) + 1

interp_func = UnivariateSpline(x, y)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)
interp_func = interp1d(x, y)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)
interp_func = Rbf(x, y)
newarr = interp_func(np.arange(2.1, 3, 0.1))
print(newarr)
xnew = np.arange(2.1, 3, 0.1)
print(xnew**2 + np.sin(xnew) + 1)
```

Statystyka

https://www.w3schools.com/python/scipy/scipy_statistical_significance_tests.php
Przykład:

```
import numpy as np
from scipy.stats import describe

v = np.random.normal(size=100)
res = describe(v)

print(res)
print(res.mean)
```