Kurs rozszerzony języka Python

Wykład 8.

Marcin Młotkowski

26 listopada 2024

Plan wykładu

Plan wykładu

Wstęp

Analiza, przetwarzanie i wizualizacja danych

Pakiety

matplotlib	wizualizowanie danych
NumPy	pakiet do pracy z danymi numerycznymi
SciPy	obliczenia symboliczne
Pandas	przetwarzanie danych
SciPy	obliczenia symboliczne

Narzędzia

IPython

Jupyter Notebook

Plan wykładu

matplotlib

Biblioteka w Pythonie do rysowania wykresów. Bardzo różnych.

Podstawy

Co to jest wykres (np. wykres funkcji) na płaszczyźnie: to zbiór punktów (x_i, y_i).

W `matplotlib` punkty podajemy za pomocą dwóch wektorów:

- $[x_1, x_2, \ldots, x_n]$
- $[y_1, y_2, \ldots, y_n]$

Podstawy

Co to jest wykres (np. wykres funkcji) na płaszczyźnie: to zbiór punktów (x_i, y_i).

W `matplotlib` punkty podajemy za pomocą dwóch wektorów:

- $[x_1, x_2, \ldots, x_n]$
- $[y_1, y_2, \ldots, y_n]$

Na przykład

```
import random

xs = [ random.random() for i in range(10) ]
ys = [ random.random() for i in range(10) ]
```

matplotlib

```
import random

xs = [ random.random() for i in range(10) ]
ys = [ random.random() for i in range(10) ]

import matplotlib.pyplot as plt

plt.plot(xs, ys)
# albo: plt.scatter(xs, ys)

plt.show()
```

Wykresy funkcji: funkcje harmoniczne

```
import matplotlib.pyplot as plt
import math

def frange(fr, to, st):
    while fr < to:
        yield fr
        fr += st

xs = [ wart for wart in frange(0.1, 10*math.pi, 0.1) ]
ys = [ math.sin(t)*(t*t - 30*t) for t in xs ]

plt.plot(xs, ys)

plt.show()
```

Wyliczanie n-tego wyrazu ciągu Fibonacciego

Popularne implementacje:

- implementacja rekurencyjna $O(2^n)$;
- implementacja iteracyjna $O(n)$;
- implementacja potęgowanie macierzy $\Theta(\log n)$.

Wyliczanie n-tego wyrazu ciągu Fibonacciego

Popularne implementacje:

- implementacja rekurencyjna $O(2^n)$;
- implementacja iteracyjna $O(n)$;
- implementacja potęgowanie macierzy $\Theta(\log n)$.

Na różnych wykresach

Porównanie implementacji: ciąg Fibonacciego

```
import timeit

def fib1(n):
    ....

def fib2(n):
    ....

def fib3(n):
    ....

xs = [ n for n in range(1, 30, 5) ]

ys1 = [ timeit.timeit(lambda : fib1(i), number=100) for i in xs ]
ys2 = [ timeit.timeit(lambda : fib2(i), number=100) for i in xs ]
ys3 = [ timeit.timeit(lambda : fib3(i), number=100) for i in xs ]
```

Na jednym wykresie

```
plt.plot(xs, ys1, marker='x')
plt.plot(xs, ys2, marker='o')
plt.plot(xs, ys3, marker='*')

plt.legend(["rekurencja","iteracja", "macierzowy"])

plt.show()
```

Na odrębnych wykresach

```
fig, (ax1, ax2, ax3) = plt.subplots(3,1)
# fig to cały obrazek, ax1, ax2, ax3 to wykresy na obrazku

ax1.plot(xs, ys1, marker='x')
ax1.set_title('rekurencyjna')
ax1.set_ylabel("sec")

ax2.plot(xs, ys2, marker='o')
ax2.set_title('iteracyjna')

ax3.plot(xs, ys3, marker='*')
ax3.set_title('macierzowa')

plt.show()
```

Poprzedni wykład

```
params = {'q': 'Wrocław', 'mode': 'json', 'units': 'metric'
url = "http://api.openweathermap.org/data/2.5/forecast"

res = requests.get(url, params=params)

with open("prognoza.json", 'w') as fh:
    dane = res.json()
    fh.write(json.dumps(dane))
```

Przetwarzanie json'a

```
time = []
temp = []
cisn = []
wiatr = []

for progn in dane['list']:
    time.append(progn['dt_txt'])
    temp.append(progn['main']['temp'])
    cisn.append(progn['main']['pressure'])
    wiatr.append(progn['wind']['speed'])

time = [ cz[5:16] for cz in time ]
```

Dwa wykresy na jednej osi

```
plt.plot(time, temp, color='blue')
plt.legend(['temperatura'])

plt.twinx()

plt.plot(time, cisn, color='red')
plt.legend(['ciśnienie'])

plt.show()
```

Uzupełnienie wykresu

Dokładamy wykres wiatru:

```
plt.bar(time, wiatr)

plt.show()
```

Inny przykład: funkcja parametryczna

Krzywe Lissajoux: złożenie dwóch prostopadłych ruchów harmonicznych

$$x(t) = sin(a * t + \pi/2)$$
$$y(t) = sin(b * t)$$

gdzie $t \in [-\pi, \pi]$. Gdy $\frac{a}{b}$ jest wymierna, to krzywa jest zamknięta.

Implementacja

```
import matplotlib.pyplot as plt
import math

a, b = 9, 8

ts = list(frange(-math.pi, math.pi, 0.01))

xs = [ math.sin(a * t + math.pi/2) for t in ts ]
ys = [ math.sin(b*t) for t in ts ]

plt.plot(xs, ys)
plt.show()
```

Jak animować wykresy

- wykres początkowy dla pewnych danych (wektory `xdata` i `ydata`) początkowych;
- aktualizacja: zmodyfikować `xdata` i `ydata`, narysować;
- wykorzystać obiekt klasy `matplotlib.animation.FuncAnim`

Początek

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import math

a, b = 9, 8

fig = plt.figure()
ax = plt.axes(xlim=(-2, 2), ylim=(-2, 2))

xdata, ydata = [], []
line, = ax.plot([], [])

def init():
    line.set_data([], [])
    return line,
```

Aktualizacja wykresu

```
def animate(i):
    t = 0.01*i
    x = math.sin(a * t + math.pi / 2.0)
    y = math.sin(b*t)
    xdata.append(x)
    ydata.append(y)
    line.set_data(xdata, ydata)
    return line,
```

I na koniec:

```
ani = animation.FuncAnimation(fig, animate,
        init_func=init, frames=500,
        interval=50, blit=True)
plt.show()
```

Trochę inna animacja

```
def animate(i):
    t = 0.01*i
    x = math.sin(a * t + np.pi / 2.0)
    y = math.sin(b*t)
    xdata.append(x)
    xdata = xdata[-50:]
    ydata.append(y)
    ydata = ydata[-50:]
    line.set_data(xdata, ydata)
    return line,
```

Inne możliwosci i rozszerzenia `matplotlib`

- różne rodzaje wykresów;
- wykresy trójwymiarowe;
- mapy;

Plan wykładu

Zadanie

Chcemy zobaczyć, jak wygląda dźwięk z mikrofonu komputera.

Odczyt z mikrofonu systemowego

```
from pvrecorder import PvRecorder

recorder = PvRecorder(device_index=-1, frame_length=512)

while True:
    frame = recorder.read()
    print(f"{frame}")

recorder.stop()
recorder.delete()
```

frame to 512-elementowa lista `int`-ów.

Podłączenie bufora do animacji

Potrzebujemy funkcji

```
def animate(i):
    ...
    ydata = <kolejny bufor z danymi>
    ...
```

Generator buforów z odczytu

```
def voice():
    recorder = PvRecorder(device_index=-1, frame_length=512
    while True:
        frame = recorder.read()
        yield frame
```

Odczyt danych z generatora

```
bufor = iter(voice())

def animate(i):
    ...
    ydata = next(bufor)
    ...
```

Odczyt danych z generatora, cd

```
def animate():
    bufor = iter(voice())

    def anim(i):
        ydata = next(bufor)
        line.set_data(xdata, ydata)
        return line,

    return anim

ani = animation.FuncAnimation(fig, animate(),
        init_func=init, frames=500,
        interval=50, blit=True)
```