11. Numerické výpočty, vizualizace

BAB37ZPR – Základy programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přehled témat

- Část 1 – NumPy
  - Tvorba a vlastnosti polí
  - Speciální funkce pro tvorbu polí
  - Skládání a rozkládání polí
  - Základní operace
  - Broadcasting
- Část 2 – Vizualizace

Část I

NumPy

I. NumPy

Datové struktury – jednodimenzionální pole

- Základní a zdaleka nejdůležitější datovou strukturou v NumPy jsou vícerozměrná pole, objekty typu `ndarray`

```
>>> import numpy as np
>>> # vstup jako seznam
>>> np.array([1, 3, 5, 7, 9])
array([1, 3, 5, 7, 9])

>>> # vstup jako n-tice
>>> np.array((1, 3, 5, 7, 9))
array([1, 3, 5, 7, 9])

>>> # vstup jako výsledek jiné operace
>>> np.array(range(5))
array([0, 1, 2, 3, 4])
```

Datové struktury – vícedimenzionální pole

- Dvourozměrná pole se zadávají ještě celkem dobře a relativně přehledně

```
>>> np.array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3],
       [4, 5, 6]])
```

- Ve vyšších dimenzich se přehlednost vytrácí

```
>>> np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

Převod `ndarray` na `list`

- pole `ndarray` vypadá jako seznam, ale sezna to není (i když má řadu podobných vlastností)
- ke konverzi na seznam slouží funkce `tolist()`

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> x.tolist()
[[1, 2, 3], [4, 5, 6]]
```

Vlastnosti polí

shape	tvar pole
ndim	počet dimenzí pole
size	celkový počet prvků pole

```
>>> x = np.array([1, 2, 3])
>>> x.shape
(3,)
>>> x.ndim, x.size
(1, 3)

>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> x.shape
(2, 3)
>>> x.ndim, x.size
(2, 6)
```

Interní velikost prvku

- Informaci o velikosti jednoho prvku pole v bajtech obsahuje atribut `itemsize`
- Ve výchozím nastavení se prvky generují typicky v přesnosti 64 bitů

```
>>> np.array([1, 2, 3], np.complex)
array([1.+0.j, 2.+0.j, 3.+0.j])
>>> np.array([1, 2, 3], np.complex).itemsize
16
>>> np.array([1, 2, 3], dtype=np.float)
array([1., 2., 3.])
>>> np.array([1, 2, 3], dtype=np.float).itemsize
8
>>> np.array([1, 2, 3], int)
array([1, 2, 3])
>>> np.array([1, 2, 3], int).itemsize
4
```

Iterace po skupinách

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]]) # 1D
>>> [i for i in x]
[array([1, 2, 3]), array([4, 5, 6])]
>>> xs = np.arange(8).reshape(2, 2, 2)   # 2D
>>> xs
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
>>> for i,x in enumerate(xs):
...     print(i, x)
...
0 [[0 1]
 [2 3]]
1 [[4 5]
```

Rozbalování do n-tic

```
>>> x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> for a,b,c in x:
...     print(a, b, c, sep='|', end='\\\n')
...
1|2|3\
4|5|6\
7|8|9\
>>> xs = np.arange(8).reshape(2, 2, 2)
>>> for a,b in xs:
...     print(a, b, sep='|')
...
[0 1]|[2 3]
[4 5]|[6 7]
```

Iterace přes prvky

```
>>> # a) plochý pohled s kopií do nového seznamu
>>> x.flat
<numpy.flatiter object at 0x000001D6D0BD3FE0>
>>> [i for i in x.flat]
[1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> # b1) kopie prvků vždy
>>> y1 = x.flatten()
>>> y1
array([1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> # b2) kopie prvků pouze je-li nutno
>>> y2 = x.ravel()
>>> y2
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Další atributy polí

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> # pohled na transponované pole
>>> x.T
array([[1, 4],
       [2, 5],
       [3, 6]])

>>> # typ prvků pole
>>> x.dtype
dtype('int32')

>>> # převod pole na binární reprezentaci
>>> x = np.array([1, 2, 3])
>>> x.tobytes()
b'\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00'
```

I. NumPy

Tvorba a vlastnosti polí

Speciální funkce pro tvorbu polí

Skládání a rozkládání polí

Základní operace

Broadcasting

Pole s nulovými prvky

```
>>> # vektor
>>> np.zeros(3)
array([0., 0., 0.])

>>> # pole/matice
>>> np.zeros([2, 3])     # rozměry seznamem
array([[0., 0., 0.],
       [0., 0., 0.]])

>>> np.zeros((2, 3))     # rozměry n-ticí
array([[0., 0., 0.],
       [0., 0., 0.]])

>>> # více rozměrů a jiný typ
>>> np.zeros([2, 3, 2], dtype=int)
array([[[0, 0],
```

Pole s jednotkovými nebo stejnými prvky

```
>>> np.ones([2, 3])
array([[1., 1., 1.],
       [1., 1., 1.]])

>>> np.full([2, 3], 7)
array([[7, 7, 7],
       [7, 7, 7]])

>>> np.full([2, 3], np.inf)
array([[inf, inf, inf],
       [inf, inf, inf]])
```

Diagonální matice

```
>>> np.identity(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> np.eye(3, k=1)
array([[0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 0.]])
>>> np.eye(3, k=-1)
array([[0., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

Vektor zadaný rozsahem 1/2

```
>>> np.arange(3)
array([0, 1, 2])

>>> np.arange(3, 9)
array([3, 4, 5, 6, 7, 8])

>>> np.arange(3, 9, 2)
array([3, 5, 7])

>>> np.arange(3.0, 9, 2)
array([3., 5., 7.])
```

Vektor zadaný rozsahem 2/2

```
>>> np.linspace(2, 4)
array([2.        , 2.04081633, 2.08163265, 2.12244898, 2.16326531,
       2.20408163, 2.24489796, 2.28571429, 2.32653061, 2.36734694,
       2.40816327, 2.44897959, 2.48979592, 2.53061224, 2.57142857,
       2.6122449 , 2.65306122, 2.69387755, 2.73469388, 2.7755102 ,
       2.81632653, 2.85714286, 2.89795918, 2.93877551, 2.97959184,
       3.02040816, 3.06122449, 3.10204082, 3.14285714, 3.18367347,
       3.2244898 , 3.26530612, 3.30612245, 3.34693878, 3.3877551 ,
       3.42857143, 3.46938776, 3.51020408, 3.55102041, 3.59183673,
       3.63265306, 3.67346939, 3.71428571, 3.75510204, 3.79591837,
       3.83673469, 3.87755102, 3.91836735, 3.95918367, 4.        ])
>>> np.linspace(2, 4, num=5)
array([2. , 2.5, 3. , 3.5, 4. ])
>>> np.logspace(1, 2, num=5, base=2)
array([2.        , 2.37841423, 2.82842712, 3.36358566, 4.        ])
```

Změna tvaru pole

```
>>> # původní vektor
>>> a = np.arange(8)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> # jiný pohled na něj
>>> b = a.reshape(2, 4)
>>> b
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

- Měnit tvar se samozřejmě dá libovolným směrem, pokud to jde rozumně provést.
- Plus je možno jednu z dimenzí nechat dopočítat automaticky z nového rozměru a počtu prvků – stačí ji označit kódem `-1`.

I. NumPy

Skládání podél vybrané osy

```
>>> a = np.array([[1,2], [3,4]])     # a.shape == (2, 2)
>>> b = np.array([[5,6]])            # b.shape == (1, 2)

>>> np.concatenate((a, b))
array([[1, 2],
       [3, 4],
       [5, 6]])

>>> np.concatenate((a, b.T), axis=1)
array([[1, 2, 5],
       [3, 4, 6]])
```

Skládání polí stejného tvaru

```
>>> a = np.array([1,2,3])
>>> b = np.array([4,5,6])
>>> c = np.array([7,8,9])

>>> np.stack((a, b, c))
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> np.stack((a, b, c), axis=-1)
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

Pole z dílčích matic

```
>>> np.block([
... [np.eye(2) * 2,   np.zeros((2, 3))],
... [np.ones((3, 2)), np.eye(3) * 3   ]
... ])
array([[2., 0., 0., 0., 0.],
       [0., 2., 0., 0., 0.],
       [1., 1., 3., 0., 0.],
       [1., 1., 0., 3., 0.],
       [1., 1., 0., 0., 3.]])
```

Rozdělování polí 1/2

```
>>> # A) jednoduché dělení
>>> x = np.arange(9)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8])

>>> np.split(x, 3)
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]

>>> # složitější dělení (neexistující index vrátí [])
>>> np.split(x, [3, 5, 10])
[array([0, 1, 2]), array([3, 4]), array([5, 6, 7, 8]), array([], dtype=int32
```

Rozdělování polí 2/2

```
>>> # B) složitější tvar pole a dělení
>>> x = np.arange(9).reshape(3, 3)
>>> x
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.split(x, [2])    # výchozí osa
[array([[0, 1, 2],
       [3, 4, 5]]), array([[6, 7, 8]])]
>>> np.split(x, [2], axis=1)    # jiná osa
[array([[0, 1],
       [3, 4],
       [6, 7]]), array([[2],
       [5],
       [8]])]
```

I. NumPy

Binání a unární operace

```
>>> # základní pole
>>> a = np.array([[1, 2, 3], [4, 5, 6]])

>>> 3 * a     # násobení pole skalárem
array([[ 3,  6,  9],
       [12, 15, 18]])

>>> a + a     # sčítání polí
array([[ 2,  4,  6],
       [ 8, 10, 12]])

>>> a * a     # násobení polí - pozor, po složkách
array([[ 1,  4,  9],
       [16, 25, 36]])
```

Příklady se skaláry

```
>>> a = np.arange(6).reshape(2, 3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])

>>> a + 1
array([[1, 2, 3],
       [4, 5, 6]])

>>> 2**a
array([[ 1,  2,  4],
       [ 8, 16, 32]], dtype=int32)

>>> 2**(a + 1)
array([[ 2,  4,  8],
       [16, 32, 64]], dtype=int32)
```

Složitější typy

```
>>> a = np.arange(6).reshape(2, 3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])

>>> # Přičtení seznamu zprava:
>>> a + [1, 1, 1]
array([[1, 2, 3],
       [4, 5, 6]])

>>> # Násobení n-ticí zleva:
>>> (2,) * a
array([[ 0,  2,  4],
       [ 6,  8, 10]])
```

Logické operace po prvcích

```
>>> a = np.array([1, 1, 0, 0], dtype=bool)
>>> b = np.array([1, 0, 1, 0], dtype=bool)

>>> # operátor
>>> a == b
array([ True, False, False,  True])

>>> # logická funkce
>>> np.logical_or(a, b)
array([ True,  True,  True, False])
```

Logické operace po polích

```
>>> a = np.array([1, 2, 0, 4])
>>> b = np.array([1, 2, 3, 4])

>>> # porovnání polí
>>> np.array_equal(a, b)
False

>>> # redukce pole podle pravdivostní hodnoty prvků
>>> np.all(a)
False
>>> np.any(a)
True
```

Matematické funkce

```
>>> x = np.array([0, np.pi / 2, np.pi])
>>> x
array([0.        , 1.57079633, 3.14159265])

>>> np.sin(x)
array([0.0000000e+00, 1.0000000e+00, 1.2246468e-16])

>>> np.log(x)
array([      -inf, 0.45158271, 1.14472989])

>>> # lineární algebra
>>> np.linalg.inv([[1, 1], [1, 0]])
array([[ 0.,  1.],
       [ 1., -1.]])
```

I. NumPy

Princip broadcastingu

- Knihovní metody dokážou za jistých (přesně definovaných!) podmínek zpracovat požadavky na operace mezi polemi, jejichž rozměry si z hlediska dané operace jinak nemusí vůbec vyhovovat.
- Trochu zjednodušeně se dá říct, že pokud je možné jedno (nebo i obě!) pole *protáhnout* z jedničky na vyhovující rozměr, bude tak (fiktivně) provedeno jeho nakopírováním do chybějících rozměrů a výsledek bude spočítán klasicky složku po složce, jak by se stalo v případě, kdy by obě měla vyhovující rozměr hned na začátku.
- Spoustu operací dokáže broadcasting velmi elegantně zapsat a díky vnitřní optimalizaci i rychle vyřešit. Pro velmi velké vstupy však už nemusí být rychlejší a vůbec méně náročnější než třebas méně přehledná smyčka nad menšími výseky dat.

```
>>> np.array([1, 2, 3]) * 2
array([2, 4, 6])
>>> np.array([1, 2, 3]) * np.array([2, 2, 2])
array([2, 4, 6])
```

Příklad

```
>>> a = np.array([[0,0,0], [10,10,10], [20,20,20], [30,30,30]])
>>> a
array([[ 0,  0,  0],
       [10, 10, 10],
       [20, 20, 20],
       [30, 30, 30]])
>>> b = np.array([0,1,2])
>>> b
array([0, 1, 2])
>>> a+b
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

Část II

Vizualizace

Možnosti vizualizace v Pythonu

Matplotlib

- Nejběžnější a nejpoužívanější balíček
- Jednoduchý na používání, práce v něm podobná jako v Matlabu
- Podpora animací, může být problém s rychlostí

pyqtgraph

- Optimalizovaný na rychlost a integraci s PyQt
- API velmi podobné PyQt
- Vhodné i na velmi rychlé animace

Integrovaná vizualizace v seaborn, pandas, sympy, ...

- Většinou založené na Matplotlib
- Založené hlavně na pohodlnost při práci s konkrétními knihovnami

11.x Graf funkce sinus

- NumPy není nutné, ale použijeme ho

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

- NumPy nepracuje se symbolickou matematikou
  - V grafu jsou vykreslené body na souřadnicích `(x,z)`, kde `x` je hodnota nezávislé a `y` závislé proměnné
  - V našem případě `y=sin(x)`
- Příprava dat pro vykreslení
  - Příprava dělení požadovaného intervalu hodnot nezávislé proměnné
  - Výpočet funkčních hodnot v bodech tohoto dělení

```
>>> x = np.linspace(0, 3 * np.pi, 500)
>>> y = np.sin(x)
```