

Nástroje na analýzu dát a numerické výpočty

## Prečo potrebujeme knižnice na numerické výpočty?

- ▶ rýchlosť
- ▶ numerické algoritmy často sú netriviálne
- ▶ pohodlie používania - Python nie je jazyk navrhnutý pre vedcov

# Naivné násobenie matíc

Matice je možné reprezentovať zoznamami:

```
a = [[1, 2], [3, 4]]
```

```
b = [[0, 1], [1, 0]]
```

## Naivné násobenie matíc

**A**:  $m \times n$

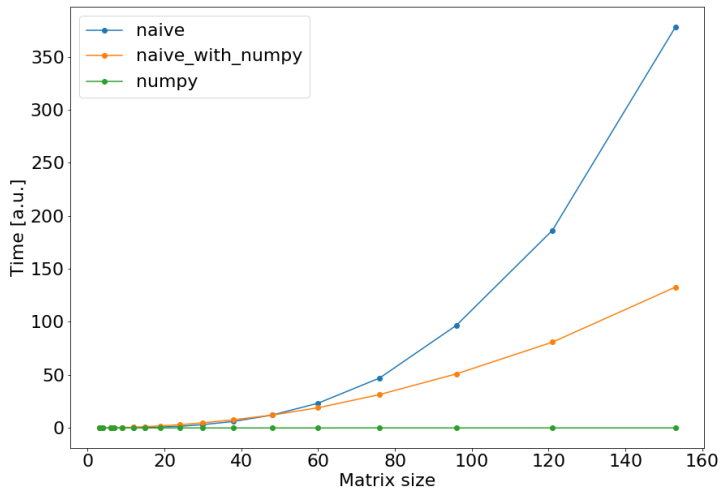
**B**:  $n \times p$

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj} \quad (1)$$

```
def naive_matmul(a, b):  
    res = []  
    for i in range(len(a)):  
        new_row = []  
        res.append(new_row)  
        for j in range(len(b[0])):  
            val = 0  
            for k in range(len(a[0])):  
                val += a[i][k] * b[k][j]  
            new_row.append(val)  
    return res
```

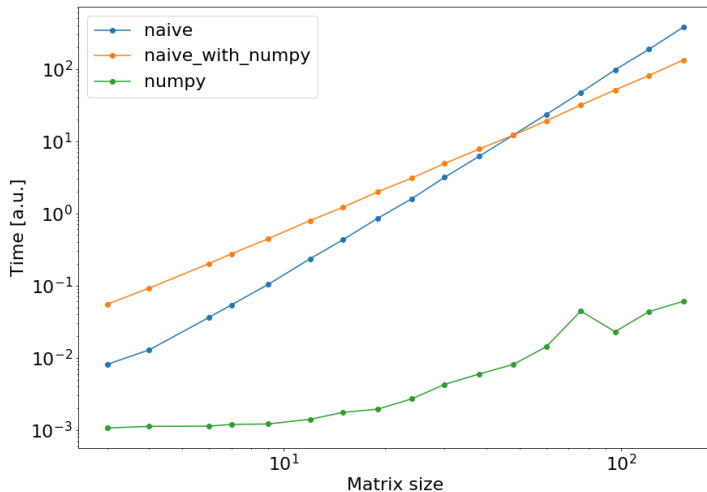
```
c = naive_matmul(a, b)
```

# Naivné násobenie matíc



Obr. 1: Lineárna škála

# Naivné násobenie matíc



Obr. 2: Logaritmická škála

## Maticové násobenie v numpy

```
import numpy as np

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

c = a @ b
# alebo
c = np.matmul(a, b)
```

Maticové násobenie nie je to isté čo normálne násobenie:

```
c = a * b      # nasobi po prvkoch
```

# Knižnica numpy

## Stránka numpy

Knižnica na základné numerické výpočty, to znamená:

- ▶ pohodlné používanie vektorov a matíc
- ▶ lineárna algebra – násobenie matíc, výpočet vlastných hodnot, atď
- ▶ štatistika – priemer, odchýlka, atď
- ▶ FFT (Fast Fourier Transform)
- ▶ Veľa iných vecí – čítajte dokumentáciu (nie moje slidy)!



# NumPy Array

- ▶ základná dátová štruktúra v NumPy
- ▶ prvky sú homogénne = rovnakého typu
- ▶ rýchlejšie, úspornejšie ako zoznamy
- ▶ pripravené na prácu s číslami

```
import numpy as np

python_list = [1.0, 2.0, 3.0, 4.0]
numpy_array = np.array(python_list)

nuly = np.zeros(2)
jednicky = np.ones(2)

arr_arange = np.arange(2, 9, 2)
arr_linspace = np.linspace(0, 10, num=5)
```

## Operácie s array

```
print(3 * numpy_array)
print(numpy_array / 3)
print(numpy_array**3)
print(numpy_array * numpy_array)

# Toto je TypeError:
print(python_list / 3)
```

## Indexovanie

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

```
a[0] # -> array([1, 2])
```

```
a[0][1] # -> 1
```

```
a[0][1] = 32
```

```
a[0,1] # -> 1
```

```
a[:,1] # -> array([2, 4])
```

```
a[:,1] = 32
```

```
a[:,1] = [2, 4]
```

```
a[a > 2] += 10
```

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

```
b[[1,2]]
```

## Iterovanie

```
a = np.array([[1, 2], [3, 4]])  
for i in a:  
    print(i)
```

```
for i in range(a.shape[0]):  
    for j in range(a.shape[1]):  
        print(a[i][j])
```

# Rozmery

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

## Matematické funkcie

```
x = np.array([0.1, 0.5, 2.0])

print("sin",      np.sin(x))
print("sqrt",    np.sqrt(x))
print("exp",      np.exp(x))
print("rad2deg", np.rad2deg(x))
print("sum",      np.sum(x))
print("cumsum",  np.cumsum(x))
print("cumprod", np.cumprod(x))
print("diff",    np.diff(x))
```

a veľa iných.

## Nové “čísla”

```
np.pi    # 3.141...  
np.e     # 2.718...
```

```
np.inf  
np.log(0) == -np.inf  
  
0.1**np.inf == 0  
1.1**np.inf == np.inf
```

NaN nie je číslo:

```
np.log(-1) == np.nan  
  
np.nan + 1      # -> np.nan  
np.nan > 1      # -> False  
np.nan < 1      # -> False  
np.nan == np.nan  # -> False
```

## Iné užitočné veci:

Knižnica numpy obsahuje hlavne funkcie (metódy moc nie):

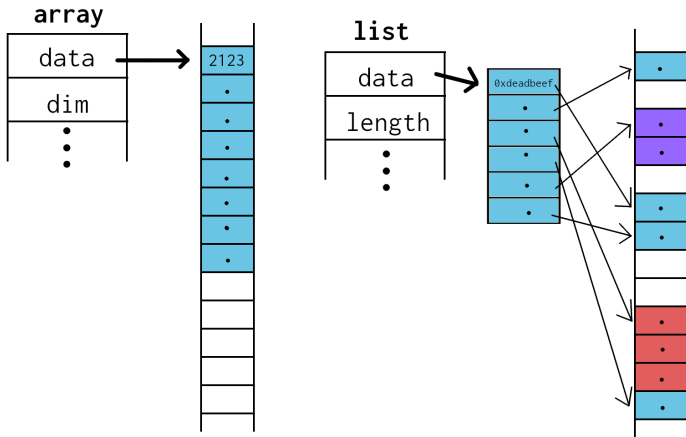
- ▶ vytváranie array-ov: `numpy.zeros`, `numpy.ones`, `numpy.linspace`, **iné**
- ▶ manipulácia: `numpy.concatenate`, `numpy.reshape`, **iné**
- ▶ skalárny a vektorový súčin: `numpy.dot`, `numpy.cross`
- ▶ štatistika: `numpy.mean`, `numpy.std`, **iné**
- ▶ tranpozícia matice: `numpy.transpose` alebo jednoducho `a.T` (a je nejaký `numpy.array`).

Existuje príliš veľa funkcií v numpy, aby ste si všetko zapamätali – použite internetový vyhľadávač, napr.:

*“numpy how to solve linear system”*



# numpy.array vs list



## dtype a shape

Z predchádzajúceho obrázku plynú dve implikácie.

- ▶ Celý array musí byť jedného datového typu:

```
a = np.array([1, 2])
print(a.dtype) # int64
a = np.array([1, 2.0]) # vsimnite si desatinnej bodky
print(a.dtype) # float64
```

- ▶ Pamäť uvažujeme lineárnu -> matica je v skutočnosti vektor + tvar:

```
a = np.array([[1, 2], [3, 4]])
print("2x2", a)
a.shape = (4, 1)
print("4x1", a)
a.shape = (4,)
print("vektor", a)
a.shape = (2, 2, 1)
print("3D: 2x2x1", a)
```

## Ukladanie a načítanie

```
np.savetxt('array_example.txt', a)  
b=np.loadtxt('array_example.txt')
```

```
np.save('array_npy_example', a)  
b = np.load('array_npy_example.npy')
```

# matplotlib

Pravedepodobne najpokročilejšia knižnica na vizualizáciu dát je **matplotlib**.

- ▶ všetky možné 2D grafy
- ▶ jednoduché 3D grafy
- ▶ animácie
- ▶ interaktívne tlačítka

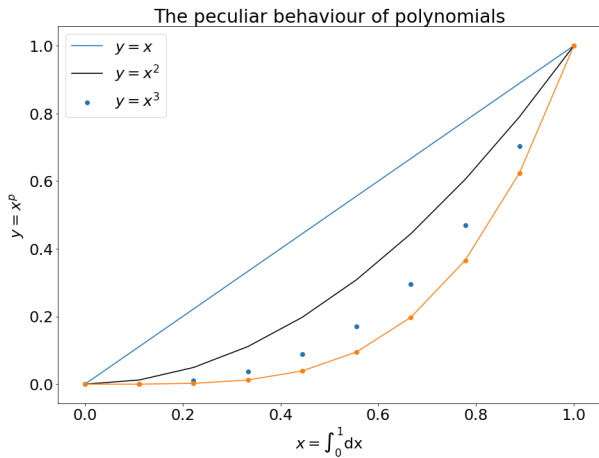
Pozrite sa na **príklady**.

## matplotlib

```
import matplotlib.pyplot as plt
import numpy as np  # ale plot funguje aj na zoznam
x = np.linspace(0, 1, 10)
y, y2, y3, y4 = x, x**2, x**3, x**4

plt.plot(x, y, label=r'$y = x$') # popis s TeXom
plt.plot(x, y2, color='k', label=r'$y = x^2$') # farba
plt.scatter(x, y3, label=r'$y = x^3$') # body
plt.plot(x, y4, 'o-') # styl

plt.legend()
plt.xlabel(r'$x = \int_0^1 \mathrm{d} x$')
plt.ylabel(r'$y = x^p$')
plt.title('The peculiar behaviour of polynomials')
plt.savefig('polynomials.png') # uloz do suboru
plt.show() # otvor okno z obrazkom
```



Obr. 3: Výsledok

# scipy

Zložitejšie numerické algoritmy sú v `scipy`. Pre ich **správne** použitie by ste mali pozorne čítať dokumentáciu, alebo zapísať si predmet na numerické metódy.

- ▶ špeciálne funkcie
- ▶ riešenie obyč. dif. rovníc
- ▶ interpolácia
- ▶ fitovanie
- ▶ integrácia
- ▶ niektoré veci sa dopĺňujú / prekrývajú s `numpy` (štatistika, lin. alg., FFT)

## sympy

Ak poznáte [wolframalpha.com](https://www.wolframalpha.com), tak sympy je podobný nástroj.

Príklad zo [stránky sympy](#) – analytické riešenie obyč. dif. rovnice:

```
from sympy import Function, dsolve
from sympy import Eq, Derivative
from sympy import sin, cos, symbols
from sympy.abc import x

y = Function('y')(x)

dy_dx = Derivative(y, x)
d2y_dx2 = Derivative(dy_dx, x)
eq = d2y_dx2 + 9*y

print(dsolve(eq, y))
# -> Eq(y(x), C1*sin(3*x) + C2*cos(3*x))
```