Nástroje na analýzu dat a numerických výpočtů

NumPy
SciPy
Matplotlib

Proč potřebujeme knihovny na numerické výpočty

- Rychlejší a kompaktnější než listy
- NumPy používá méně paměti na uložení dat (optimalizace)
- Numerické algoritmy jsou často netriviální
- Pohodlné používání

```
import numpy as np
```

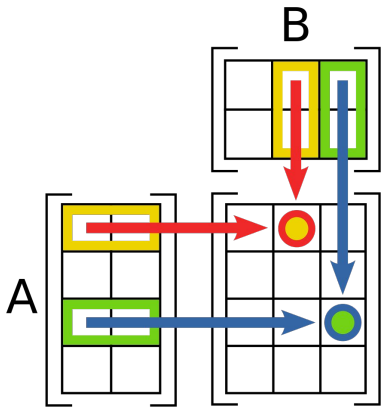
NumPy – násobení matic

Matice můžeme reprezentovat seznamy:

```
a = [[1, 2], [3, 4]]
b = [[0, 1], [1, 0]]
```

NumPy – násobení matic

$$(\boldsymbol{A} \cdot \boldsymbol{B})_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}.$$




```python
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)
```

NumPy – Array

- Základní struktura dat v NumPy je **numpy.array()**
- Prvky jsou homogenní = stejného typu (list mohl mít různé typy prvků)
- Rychlejší a úspornější než seznamy
- Připravené na práci s čísly

```python
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)
```

NumPy – násobení matic

Pomocí NumPy arrays můžeme definovat matice jako:

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[0, 1], [1, 0]])
c = a @ b
# nebo
c = np.matmul(a, b)
```

Maticové násobení není to stejné jako normální násobení

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

NumPy – operace 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)
```

NumPy – indexování

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

print(a)
print(b)

print(a[0]) # prvni element
print(a[-1]) # posledni element

print(b[0,1]) #element 1 v druhem sloupci
print(b[1,:]) #elementy v druhe rade

print(b[:,2])
a = np.arange(12)
print(a)
```

NumPy – rozměry

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

NumPy – iterování

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

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

NumPy – matematické funkce

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

print("sin", np.sin(x))
print("cos", np.cos(x))
print("sqrt", np.sqrt(x))
print("exp", np.exp(x))
print("rad2deg", np.rad2deg(x))
print("sum", np.sum(x))
print("abs", np.abs(x))
```

NumPy – nové „čísla“

```
np.pi
np.e

np.inf
np.log(0) == -np.inf

0.1**np.inf == 0
1.1**np.inf == np.inf
```

NaN není číslo:

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

NumPy – dtype a shape

- Celý array je stejného datového typu:

```
a = np.array([1, 2])
print(a.dtype)

a = np.array([1, 2.0])
print(a.dtype)
```

- Uvažujeme linární pamět -> matice je ve skutečnosti 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)
```

NumPy – ukládání a načítání

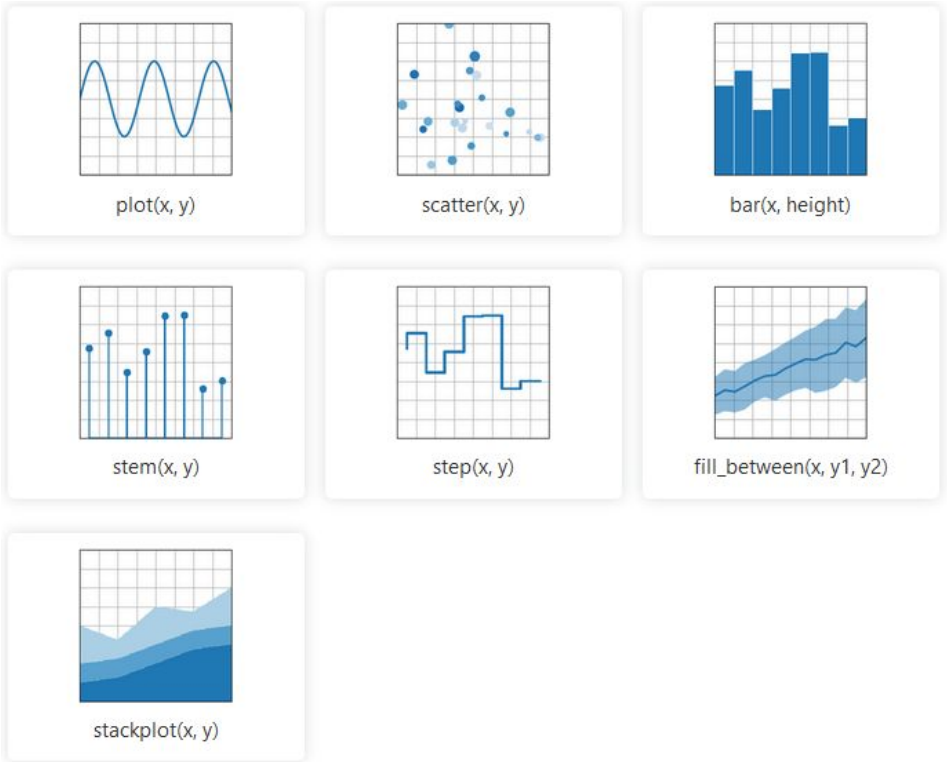
```
a = np.array([[1, 2], [3, 4]])

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

print(b)
```

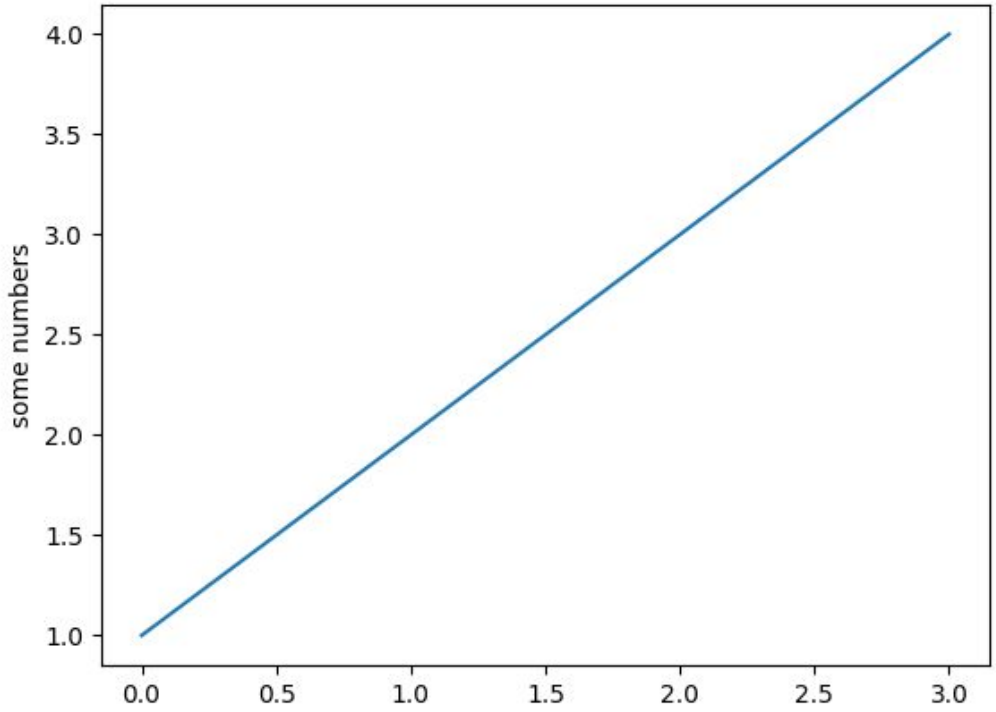
Matplotlib

- Pravděpodobně nejjednodušší knihovna pro vizualizaci dat. matplotlib
    - Zobrazení různých 2D a 3D grafů



Matplotlib – jednoduchý graf

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

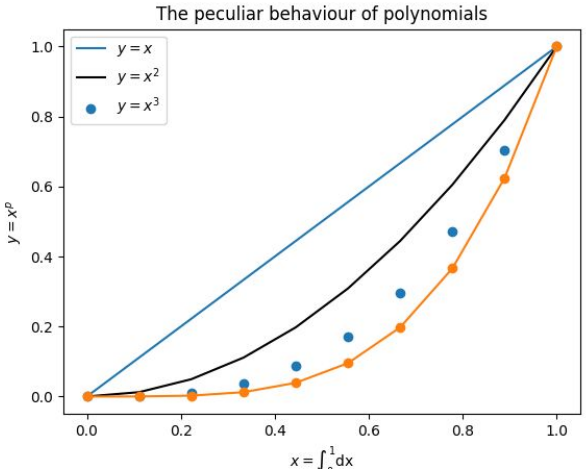


Matplotlib – detajlnější graf

```python
import matplotlib.pyplot as plt
import numpy as np # ale plot funguje i na seznam

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 TeXem
plt.plot(x, y2, color='k',label=r'$y = x^2$') #
barva
plt.scatter(x, y3, label=r'$y = x^3$') # body
plt.plot(x, y4, 'o-') # styl
plt.legend()
plt.xlabel(r'$x = \int_0^1 \rm{d} x$')
plt.ylabel(r'$y = x^p$')
plt.title('The peculiar behaviour of polynomials')
plt.savefig('polynomials.png') # uloží do souboru
plt.show() # otevře okno s obrázkem
```



SciPy

Složitější numerické algoritmy jsou v SciPy. Pro jejich správné použití by jste měli pozorně číst dokumentaci!

SciPy nám může pomoci s:

1. Speciálními funkcemi
2. Řešením obyčejných diferenciálních rovnic
3. Interpolací
4. Fitováním
5. Integrováním a derivecame
6. A mnoho dalšího

SciPy – ukázka integrace

Složitější numerické algoritmy jsou v SciPy. Pro jejich správné použití byste měli pozorně číst dokumentaci!

```python
import numpy as np
from scipy.integrate import quad

def f(x):
    return x**2

iv, err = quad(f,0,1)

print("Zintegrovana hodnota: ", iv)
print("Error:", err)
```

Cvičení 1.

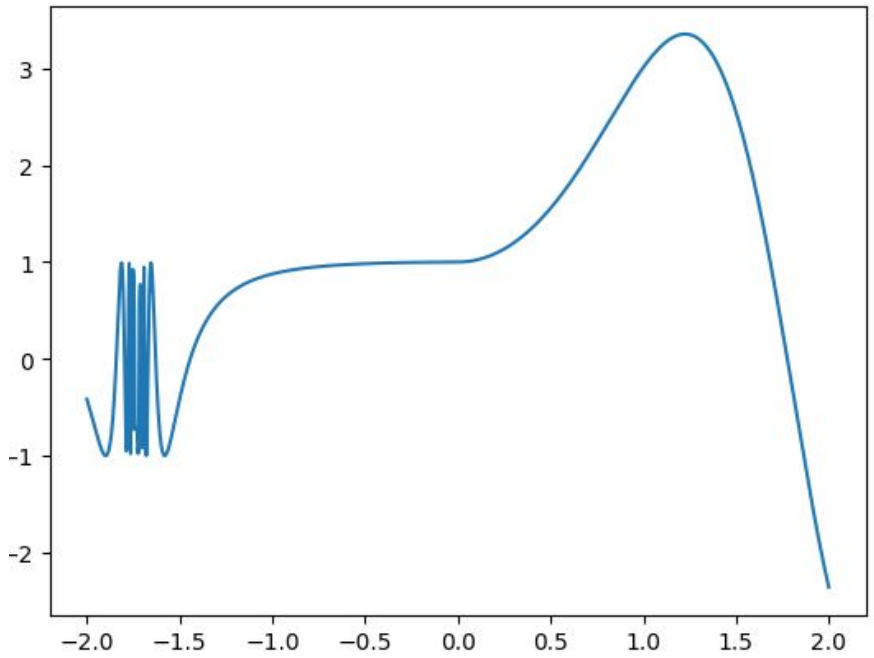
1. Vytvořte matici s názvem „**a1**“ s následujícími hodnotami „**[1, 2, 3, 4, 5]**“.
2. Použijte **numpy.sum()** funkci pro zjištění sumy matice **a1**
3. Použijte **numpy.mean()** funkci pro zjištění střední hodnoty matice **a1**
4. Vytvořte matici „**a2**“, která bude obsahovat druhé mocniny matice **a1**
5. Vytvořte matici pomoci **numpy.arange()** s názvem „**a3**“, která bude obsahovat pouze liché čísla v rozsahu 1 až 10.
6. Použijte **numpy.linspace()** a vytvořte matici „**a4**“ která bude obsahovat 100 hodnot funkce cosinus mezi 0 a π .

```
suma:  15
střední hodnota:  3.0
a2:  [ 1  4  9 16 25]
a3:  [1 3 5 7 9]
a4:  [ 1.          0.99798668  0.99195481  0.9819287   0.9679487   0.95007112
  0.92836793  0.90292654  0.87384938  0.84125353  0.80527026  0.76604444
  0.72373404  0.67850941  0.63055267  0.58005691  0.52722547  0.47227107
  0.41541501  0.35688622  0.29692038  0.23575894  0.17364818  0.1108382
  0.04758192 -0.01586596 -0.07924996 -0.14231484 -0.20480667 -0.26647381
 -0.32706796 -0.38634513 -0.44406661 -0.5        -0.55392006 -0.60560969
 -0.65486073 -0.70147489 -0.74526445 -0.78605309 -0.82367658 -0.85798341
 -0.88883545 -0.91610846 -0.93969262 -0.95949297 -0.97542979 -0.98743889
 -0.99547192 -0.99949654 -0.99949654 -0.99547192 -0.98743889 -0.97542979
 -0.95949297 -0.93969262 -0.91610846 -0.88883545 -0.85798341 -0.82367658
 -0.78605309 -0.74526445 -0.70147489 -0.65486073 -0.60560969 -0.55392006
 -0.5        -0.44406661 -0.38634513 -0.32706796 -0.26647381 -0.20480667
 -0.14231484 -0.07924996 -0.01586596  0.04758192  0.1108382   0.17364818
  0.23575894  0.29692038  0.35688622  0.41541501  0.47227107  0.52722547
  0.58005691  0.63055267  0.67850941  0.72373404  0.76604444  0.80527026
  0.84125353  0.87384938  0.90292654  0.92836793  0.95007112  0.9679487
  0.9819287   0.99195481  0.99798668  1.        ]
```

Cvičení 2.

1. Napište program, který vykreslí funkci:

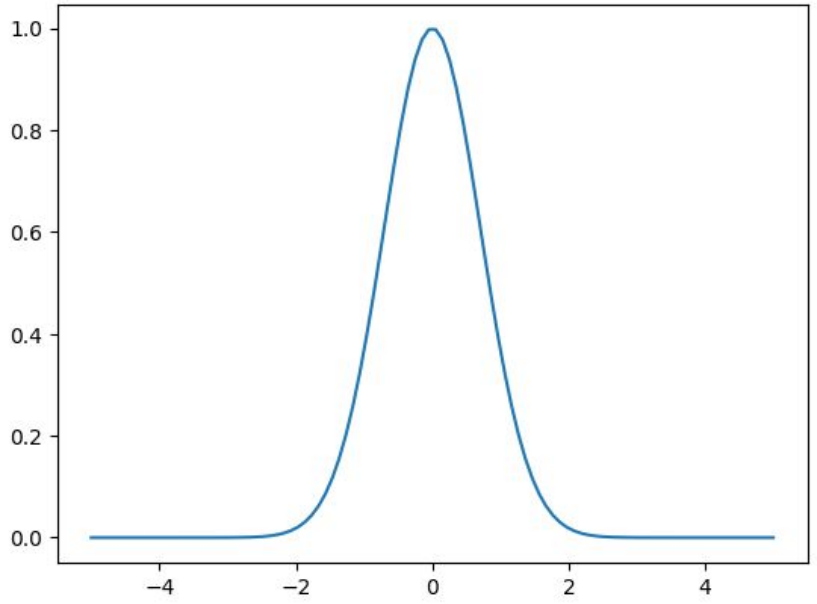
$$y(x) = \pi \sin(x^2) + e^{-x^2} \quad \textbf{if} \quad x > 0 \quad \textbf{else} \quad \cos\left(\frac{x}{|x^2 - 3|}\right),$$



Cvičení 3.

1. Vytvořte matici „**x**“, která bude mít rovnoměrné vzdálených 100 bodů mezi hodnotami -5 a 5.
2. Vytvořte funkci pro gaussovo rozdělení. # $f(x) = \exp(-x^2)$
3. Zobrazte gaussovu křivku do grafu.
4. Použijte funkci quad z knihovny **scipy.integrate** a spočítejte integrál gaussovy křivky mezi hodnotami -2 a 2

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
```




```
Integral value: 1.764162781524843
Percentile: 88.20813907624215
```

Cvičení 4.

1. Vytvořte matici „**x**“, která bude mít rovnoměrné vzdálených 10 bodů mezi hodnotami 0 a π/2.
2. Vytvořte matici „**y**“, která bude obsahovat hodnoty pro funkci cosinus v intervalu matice „**x**“
3. Zobrazte body z matice „x“ a „y“ do grafu.

4. Napište funkci pro lineární regresi. # a*x + b
5. Použijte funkci curve_fit z knihovny **scipy.optimize** pro získání smernice a posunu
6. Zobrazte lineární regresi do grafu.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

```
směrnice:  -0.7320566717685115
posun:  1.1499119310196753
```