1 Jupyter Notebook

Jupiter Notebook (dawniej iPython notebook) = środowisko do obliczeń matematycznych, korzysta z:

- interpretera Pythona (w odmianie iPython)
- biblioteki numeryczne (SciPy, NumPy)
- bibliotekę graficzną (matplotlib).

Komendy używane w Jupyter są (w większości) komendami Pythona, ale umiejętość progamowania nie jest konieczna, choć bardzo pomocna. Jupyter i Python są Open Source = free. Jupyter jest obsługiwany przez przeglądarkę internetową. Dokument Jupytera składa się z wierszy, które mają różny typ:

- tekst (formatowany za pomocą składni markdown, dodatkowo z wyrażeniami matematycznymi a la LaTeX)
- kod wykonywalny (komendy i polecenia, które służą do prowadzenia obliczeń = kod w Pythonie)
- grafikę (wysokiej jakości wykresy 2D 3D animacje itd)

Dostęp do Jupyter Notebook:

- na własnym komputerze Instalacja Jupytera (w pakiecie Anaconda)
- poprzez stronę `https://tmpnb.org/` —— polecane
- poprzez darmowe konto w serwisie Wakari
- poprzez stronę Try Jupyter

Przykładowe notatniki (notebooki):

- `http://nb.bianp.net/sort/views/`
- A-gallery-of-interesting-IPython-Notebooks
- Notatniki do “papierowych” książek

Więcej o Pythonie/SciPy/NumPy:

- `https://github.com/jrjohansson/scientific-python-lectures`
- `http://www.scipy-lectures.org/index.html`
- `http://kitchingroup.cheme.cmu.edu/pycse/`
- http://www.southampton.ac.uk/~fangohr/training/python/pdfs/Python-for-Computational-Science-and-Engineering.pdf)
- `http://www.python-course.eu/numpy.php`

2 Python - zmienne

Najprostszą formę mają zmienne tzw. skalarne:

```
In [1]: # tekst po haszu jest komentarzem pomijanym przez Jupyter/Python
        # dobrym zwyczajem jest komentowanie swojego kodu
        # (robienie notatek, wstawianie uwag itp)

        # zmienne liczbowe (całkowite, zmiennoprzecinkowe)

        a = 1
        b = 1.23
        c = 2e-1 # 0.2

        print("Zmienne numeryczne")
        print(a) # wyświetlanie zawartości zmiennych
        print(b)
        print(c)

Zmienne numeryczne
1
1.23
0.2
```

Działania arytmetycznie i nawiasy (tylko okrągłe):

```
In [2]: d = (a + 1.3*a)/a - 1
        print(d)
        print( (c+2)/b )
        d = b**3 # potęgowanie
        print(d)

1.2999999999999998
1.788617886178862
1.8608669999999998
```

Trzeba pamiętać o jawnym podawaniu znaku mnożenia:

```
In [3]: d = 4*a
        print(d)

4
```

Zmienne mogą też zawierać napisy (łańcuchy znaków, stringi):

```
In [4]: a = "Napis"
        b = "Ala ma "
        c = "kota"
        print("Zmienne mogą zawierać też napisy")
        print(a)
        print(b+c)  # łączenie napisów

Zmienne mogą zawierać też napisy
Napis
Ala ma kota
```

3 Python - matematyka

Dostęp do funkcji matematycznych jest po zaimportowaniu “numpy”:

```
In [5]: import numpy as np
```

i teraz wiele funkcji zaczyna się od przedrostka “np.”:

```
In [6]: print( np.log(100) )  # logarytm naturalny
        print( np.log10(100) ) # logarytm o podstawie dziesiętnej
        print( np.sin(2) )
        print( np.sqrt(2) ) # pierwiastek

4.60517018599
2.0
0.909297426826
1.41421356237
```

Dostęp do stałych matematycznych:

```
In [7]: print( np.e )
        print( np.pi )

2.718281828459045
3.141592653589793
```

Więcej o funkcjach matematycznych w Pythonie:
https://docs.scipy.org/doc/numpy/reference/routines.html

4 Python - obliczenia cz. 1

Tworzenie wektorów (tablic jednowymiarowych):

```
In [8]: # przykładowe dane użytkownika:
        a = np.array([1, 2, 3, 4])
        b = np.array([3, 3, 3, 3])
        print(a)
        print( b )

[1 2 3 4]
[3 3 3 3]
```

Tworzenie wektorów wypełnionych seriami wartości - funkcje linspace:

```
In [9]: #ciągi: od start do end podzielony na N części (licząc start i end)
        start = 0
        end = 10
        N = 5
        c = np.linspace(start, end, N)
        print(c)

[  0.    2.5   5.    7.5  10. ]
```

i arange:

```
In [10]: #ciągi: od start do end-N krok N
         start = 0
         end = 20
         N = 5
         c = np.arange(start, end, N)
         print(c)

[ 0  5 10 15]
```

Działania na wektorach (element po elemencie):

```
In [11]: # wektory a i b z komórki [8]
         aa = a+2
         bb = b**2
         print(aa)
         print(bb)

[3 4 5 6]
[9 9 9 9]
```

i z wykorzystaniem funkcji matematycznych:

```
In [12]: print("Teraz test log10 ...")
         aaa = np.log10( a )   # <--- log10(1) log10(2) log10(3) log10(4)
         print(aaa)

Teraz test log10 ...
[ 0.          0.30103     0.47712125  0.60205999]
```

4.1 Przykład 1

Oto wyniki eksperymentu polegającego na pomiarze stężenia po czasie: czas (min): 10, 30, 50, 60; stężenie (mol/L): 0.2, 0.3, 0.6, 0.9 Przygotować powyższe dane do postaci przydatnej do obliczeń w Jupyter/Pythonie. Utworzyć również wektor zawierający czas wyrażony w godzinach.

```
In [13]: # Rozwiązanie:

         czas_min = np.array( [10, 30, 50, 60] )      # wektor - czas w min
         stezenie = np.array( [0.2, 0.3, 0.6, 0.9] ) # wektor - stezenia

         # utworzenie wektora z czasem wyrażonym w godzinach
         czas_godziny = czas_min/60.0

         #wyniki:
         print(czas_min)
         print(stezenie)
         print(czas_godziny)

[10 30 50 60]
[ 0.2  0.3  0.6  0.9]
[ 0.16666667  0.5         0.83333333  1.        ]
```

4.2 Przykład 2

Dla t równego od 0 do 3600 (co 360), obliczyć funkcję f(t) = $0.25 \exp -0.001t$

```
In [14]: # Rozwiązanie:

         t = np.linspace(0, 3600, 10)     # wektor t
         f = 0.25*np.exp(-0.001*t)        # f(t)

         #wydruk wyników:

         print(t)
         print(f)

[    0.   400.   800.  1200.  1600.  2000.  2400.  2800.  3200.  3600.]
[ 0.25        0.16758001  0.11233224  0.07529855  0.05047413  0.03383382
  0.02267949  0.01520252  0.01019055  0.00683093]
```

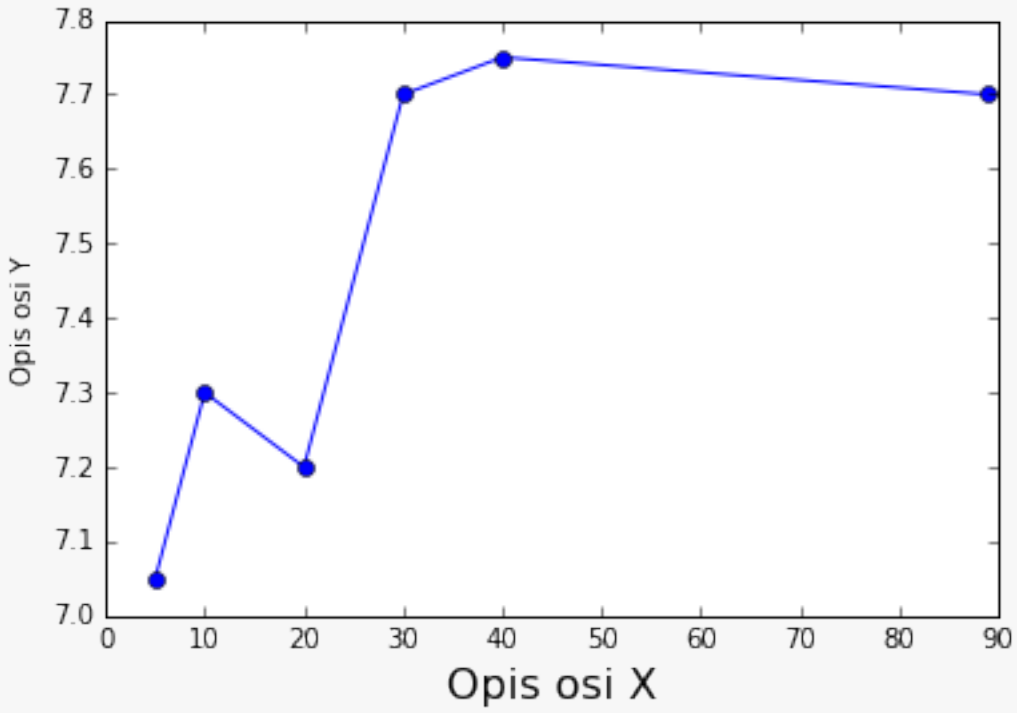
5 Wykresy

Do rysowania wykresów służy moduł “matplotlib.pyplot”.
Dokumentacja: http://matplotlib.org/users/beginner.html

5.1 Przykład 3

Narysować wykres funkcji pH = f(czas), dla danych: czas: 5, 10, 20, 30, 40, 89 pH: 7.05, 7.3, 7.2, 7.7, 7.75, 7.7

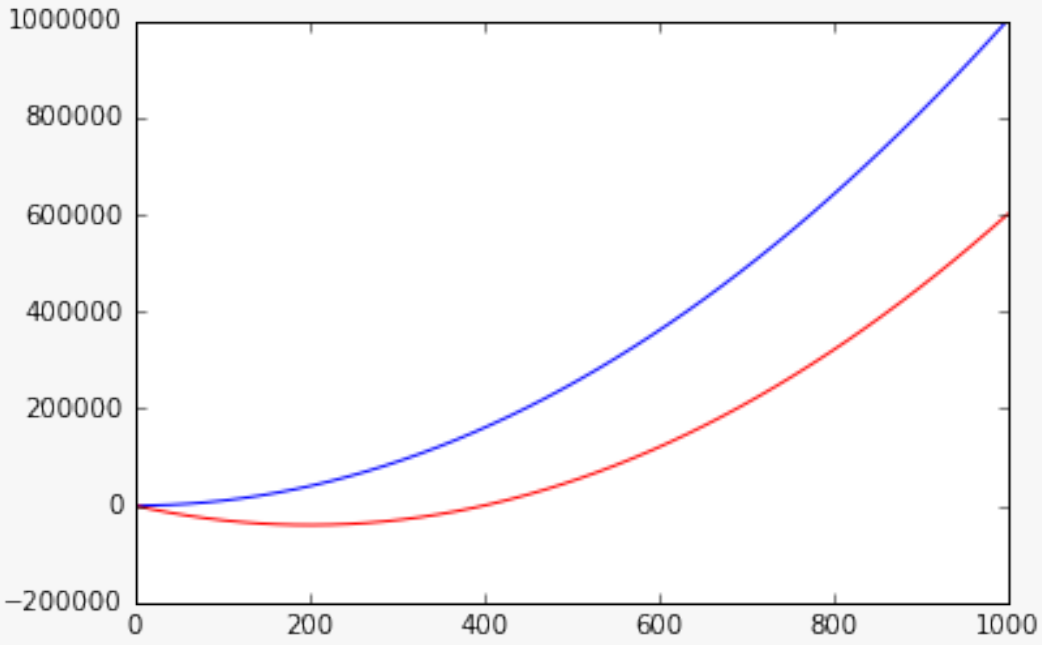
In [15]:
```
# import potrzebnych modułów:
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

# dane do wykresu
czas = np.array([ 5, 10, 20, 30, 40, 89 ])
pH = np.array([ 7.05, 7.3, 7.2, 7.7, 7.75, 7.7 ])
# narysowanie wykresu
plt.plot(czas, pH,"bo-")
# kolory: b, c, y, m, g, y, k
#symbole: o . * D - -- o- *-

# podpisy (nie są konieczne do działania Jupyter/Pythona)
plt.xlabel("Opis osi X", fontsize=16)  # dodatkowo, zmiana wielkości czcionki
plt.ylabel("Opis osi Y")

plt.show() # wyświetlenie dzieła
```



5.2 Przykład 4

Narysować wykresy funkcji $f_1(x) = x^2$ i $f_2(x) = x^2 - 400x$, dla x = 0 do 10^3. Na wykresie umieścić obie serie, w postaci kolorowych linii ciągłych

```
In [16]: # import potrzebnych modułów:
         %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np

         # dane do wykresu
         x = np.linspace(0, 10**3, 50)  # 50 to ilość punktów na wykresie
         #serie
         f1 = x**2
         f2 = x**2 - 400*x
         # narysowanie wykresu
         plt.plot(x, f1,"b-")
         plt.plot(x, f2,"r-")
         plt.show() # wyświetlenie rysunku
```



Do zapisywania wykresów w formie graficznej (bardzo dobrej jakości) służy polecenie:

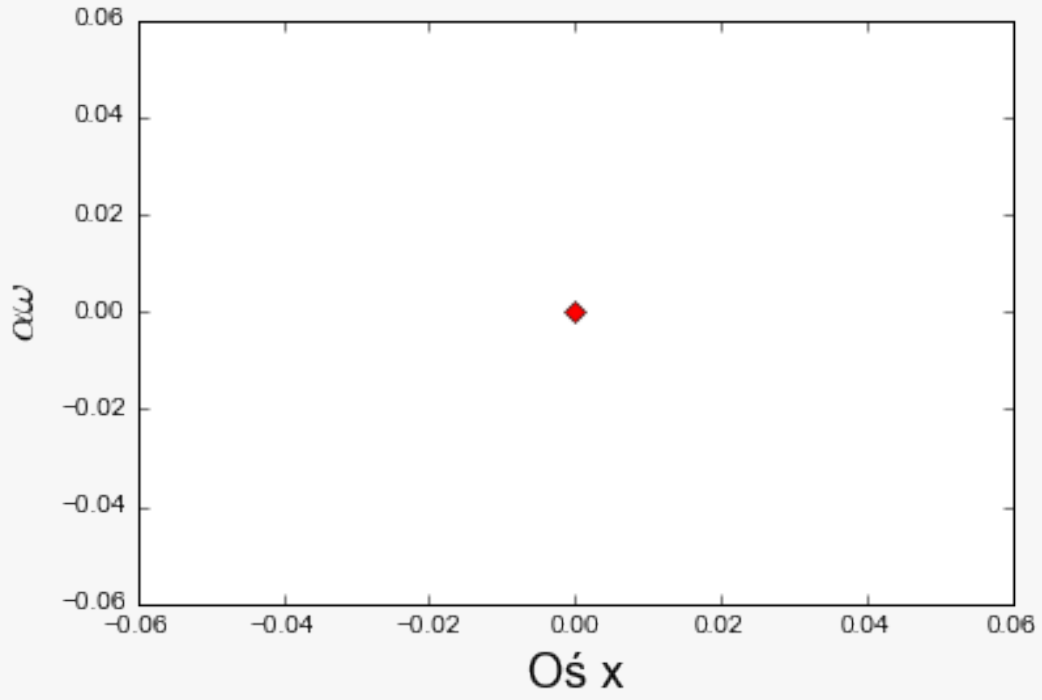
```
In [17]: plt.savefig("nazwa_pliku.png", dpi=300)

<matplotlib.figure.Figure at 0x544c5c0>
```

przy czym dpi - oznacza rozdzielczość grafiki (100 - ekran, 300 - wydruk). Inne dostępne formaty graficzne to: pdf i svg (wystarczy zmienić rozszerzenie “png” w nazwie pliku).

5.3 Przykład 5

Jak uzyskać na wykresach polskie i np. greckie symbole?

In [18]:
```
# import potrzebnych modułów:
%matplotlib inline

import matplotlib
matplotlib.rc("font", family="Arial")
# lub matplotlib.rc("font", family="DejaVu LGC Sans")
import matplotlib.pyplot as plt
import numpy as np

# znak "u" przed napisem !!!
plt.xlabel( u"Oś x", fontsize=18)

# znak "r" przed napisem, w dolarach symbole !!!
plt.ylabel( r"$\alpha \omega$", fontsize=18)

plt.plot(0,0,"rD")
plt.show()
```

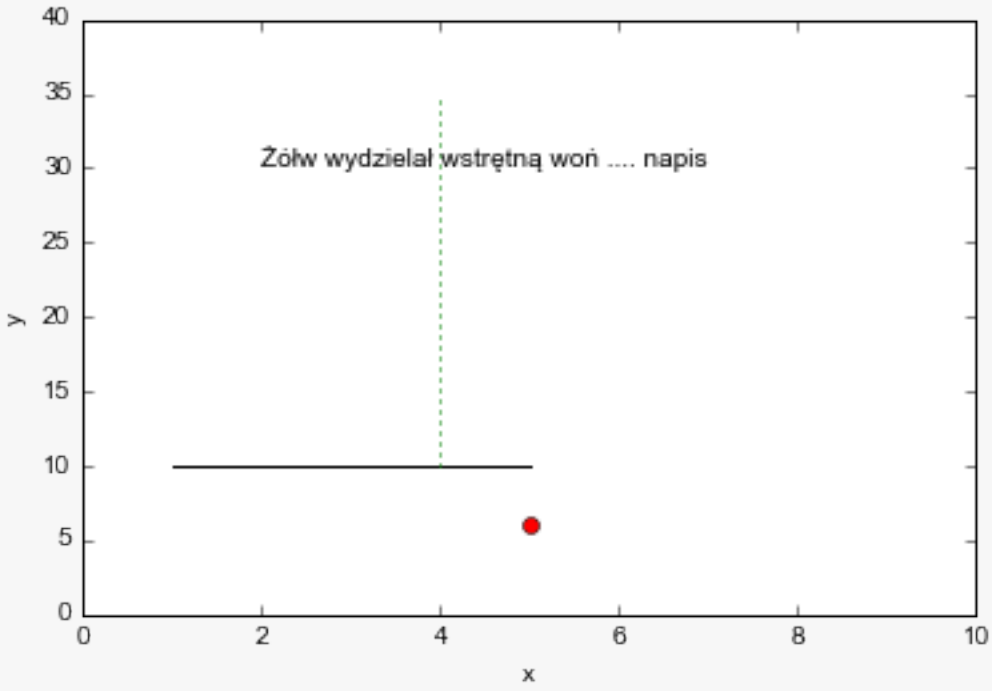


Więcej informacji o symbolach w dokumentacji:
http://matplotlib.org/users/mathtext.html
oraz tutaj:
http://www.latex-kurs.x25.pl/paper/wyrazenia_matematyczne

5.4 Przykład 6

Zakresy na osiach: “xlim” i “ylim”; rysowanie pojedynczych punktów i odcinków oraz dodawanie do wykresu adnotacji (tekstu).

In [19]:
```
%matplotlib inline
import matplotlib
matplotlib.rc("font", family="Arial") # polskie fonty
import matplotlib.pyplot as plt
import matplotlib

# zakresy: od ... do ...
plt.xlim(0,10)
plt.ylim(0,40)

#znacznik w punkcie (5,6)
plt.plot(5,6, "ro")
#odcinek od punktu 1,10 do 5,10
plt.plot( (1,5), (10,10) ,"k-")
#odcinek od punktu 4,10 do 4,35
plt.plot( (4,4), (10,35) ,"g:")

# napisy na wykresie - współrzędne wykresu, początek 2,30
plt.text(2,30, u"Żółw wydzielał wstrętną woń .... napis")

plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



6 Python - macierze

Tworzenie macierzy, np.: 4 wiersze i 2 kolumny:

```
In [20]: m = np.array([[10,20],[30,40],[50,60],[70,80]])
         print( m )

[[10 20]
 [30 40]
 [50 60]
 [70 80]]
```

Adresowanie macierzy / pobieranie danych / wskazywanie elementów:

```
In [21]: print("pierwszy wiersz macierzy m")
         print( m[0] )

         print("drugi wiersz macierzy m")
         print( m[0] )

         print("ostatni wiersz macierzy m")
         print( m[-1] )

pierwszy wiersz macierzy m
[10 20]
drugi wiersz macierzy m
[10 20]
ostatni wiersz macierzy m
[70 80]
```

Pobieranie/wskazywanie całych kolumn:

```
In [22]: print("pierwsza kolumna:")
         print( m[:,0] )

         print("druga kolumna:")
         print( m[:,1] )

         # dwukropek wybiera całą kolumną lub cały wiersz

pierwsza kolumna:
[10 30 50 70]
druga kolumna:
[20 40 60 80]
```

i na koniec pobieranie pojedynczych elementów, np. z wiersza 2 i kolumny 1:

```
In [23]: # wiersz 2 = indeks 1
         # kolumna 1 = indeks 0
         # m[indeks wiersza][indeks kolumny]

         print( m[1][0] )

30
```

6.1 Przykład 7

Utworzyć macierz z danych w pliku CSV. Narysować wykres-chromatogram czas-sygnał, wiedząc, że w pliku kolumna 1 to czas, kolumna 2 zawiera sygnał. Plik: chromatogram.txt.

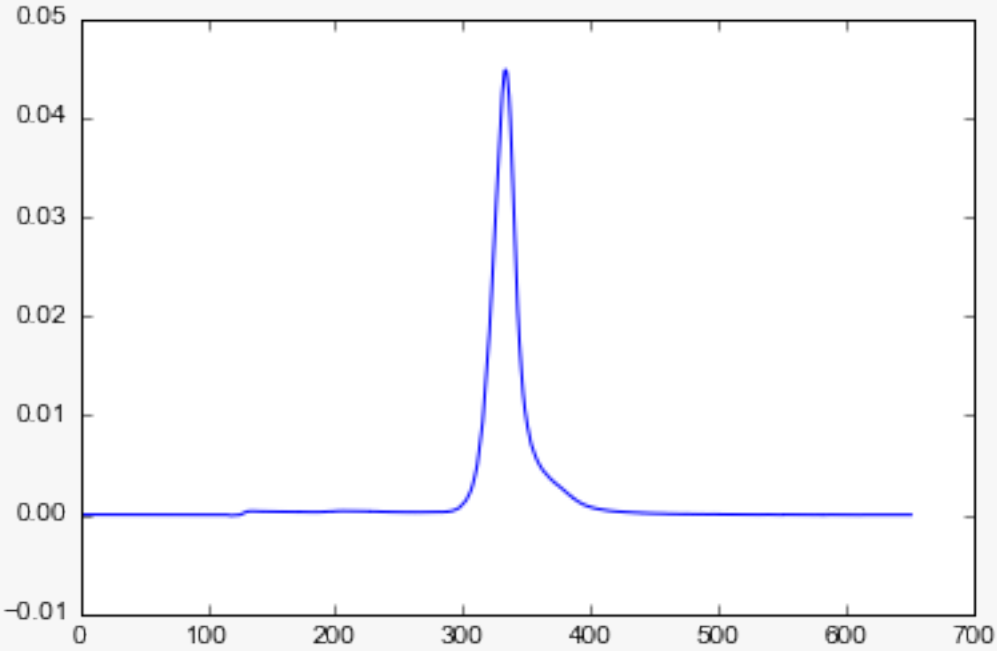
```
In [24]: %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np

         # załadowanie danych do zmiennej "dane" z pliku "chromatogram.txt"
         dane = np.genfromtxt("chromatogram.txt")
         # dane jest macierzą

         print(dane)
```

```
[[  3.00000000e-02   1.40000000e-05]
 [  1.10000000e-01   2.00000000e-06]
 [  1.90000000e-01   6.00000000e-06]
 ...,
 [  6.51590000e+02  -1.50000000e-05]
 [  6.51670000e+02  -1.90000000e-05]
 [  6.51750000e+02  -1.50000000e-05]]
```

teraz wykres:

```
In [25]: x = dane[:,0]
         y = dane[:,1]
         plt.plot(x, y, "b-")
```

```
Out[25]: [<matplotlib.lines.Line2D at 0x77c39b0>]
```



7 Python - równania/układy równań

Do rozwiązywania równań jednaj zmiennej i układów równań służy funkcja “fsolve”. Opis działania: https://www.tau.ac.il/~kineret/amit/scipy_tutorial/

7.1 Przykład 8 - równanie

Rozwiązać równanie: $2.5 = \sqrt{x}$ inaczej mówiąc, znaleźć miejsce (lub miejsca) zerowe funkcji: $2.5 - \sqrt{x} = 0$.

```
In [26]: from scipy.optimize import fsolve # potrzebna funkcja fsolve
         import numpy as np
         guess = 8 # wartość szacunkowa, startowa - do poszukiwać rozwiązania
         # definicja problemu
         def f(x):
             return 2.5 - np.sqrt(x) # wyrazy po lewej stronie

         x0, = fsolve(f, guess)
         print("Wynik")
         print(x0)

Wynik
6.25
```

7.2 Przykład 9 - układ równań

Obliczyć pH kwasu octowego (Ka=1.86e-5) o stężeniu 0.1 mol/L.

```
In [27]: import numpy as np
         from scipy.optimize import fsolve
         def model(zmienne):
             Ka = 1.86e-5
             Kw = 1e-14
             c_ch3cooh = 0.1
             h, oh, ch3coo, ch3cooh = zmienne
             eq1 = h*oh-Kw
             eq2 = (h*ch3coo)/ch3cooh - Ka
             eq3 = h - oh - ch3coo
             eq4 = ch3coo + ch3cooh - c_ch3cooh
             return [eq1, eq2, eq3, eq4]

         #kolejność: h, oh, ch3coo, ch3cooh
         guess = [1e-5, 1e-5, 1e-5, 1e-5]
         h, oh, ch3coo, ch3cooh = fsolve(model, guess)
         print( "pH:" )
         print( -np.log10(h) )
         print( "Sprawdzenie:" )
         print( h*oh )               # powinno być ~Kw
         print( ch3coo + ch3cooh )   # powinno być ~c_ch3cooh

pH:
2.86820499728
Sprawdzenie:
9.83217026803e-15
0.1
```