Wprowadzenie do obliczeń w Pythonie

Krzysztof Trajkowski

2017-03-07

Contents

1 Wprowadzenie

1.1 Wersje Pythona

Na początek warto podkreślić, że równolegle są dostępne dwie wersje Pythona tzn. 2.x (aktualna stabilna wersja to 2.7.12) oraz 3.x (aktualne stabilna wersja to 3.5.2).

Aby skorzystać z wersji Pythona 2.x w konsoli systemowej (dystrybucje linuksa) należy wpisać komendę `python` lub `python2` (ewentualnie `python3` dla wersji 3.x) i zatwierdzić enterem. Python jest domyślnie instalowany w systemach linuksowych wraz z dystrybucją. Dotyczy to także Mac OS X.

Więcej szczegółowych informacji można znaleźć po adresem:

- http://python.edu.pl/byteofpython/2.x/03.html

W tym poradniku będziemy korzystać z wersji 3.5.2. Po drobnych modyfikacjach kod zamieszczony w tym dokumencie powinien działać w Pythonie 2.7.12. Różnice pomiędzy wersjami nie są zbyt duże. Jedną z nich przedstawimy na przykładzie operatora arytmetycznego dzielenia.

```
# Python 3:
print (4/7)
```

```
## 0.5714285714285714
```

Aby uzyskać poprawny wynik dzielenia w Pythonie 2.x należy przedstawić przynajmniej jedną liczbę w formie dziesiętnej:

```
# Python 2:
print 4/7.
```

```
## 0.571428571429
```

lub na początku kodu dodać poniższą instrukcję:

```
# Python 2:
from __future__ import division
print 4/7
```

```
## 0.571428571429
```

Więcej informacji na ten temat można znaleźć pod adresami:

- http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html#python-2-2
- https://wiki.python.org/moin/Python2orPython3

1.2 Środowiska programistyczne

Pisanie komend bezpośrednio w konsoli programu nie jest optymalnym rozwiązaniem – szczególnie gdy mamy dużą ilość kodu. W takich sytuacjach warto pisać kod w pliku tekstowym i za pomocą funkcji `exec` uruchomić skrypt w Pythonie. Poniżej przykład dla skryptu o nazwie FUN.py:

```
>>> exec(open('/FUN.py').read())
```

lub z poziomu konsoli systemowej:

```
python3 '/FUN.py'
```

Innym bardzo wygodnym rozwiązaniem jest skorzystanie z edytora Spyder – dla każdej wersji Pythona jest oddzielna wersja Spaydera.

- https://github.com/spyder-ide/spyder.

Warto też wspomnieć o środowisku Jupyter dzięki któremu nie musimy ograniczać się tylko do języka Python. Poniżej instrukcja jego instalacji.

```
pip3 install jupyter
```

Należy pamiętać aby wcześniej zainstalować instalator pakietów `pip3`. Poniżej przykład jak to zrobić w systemie Linux - dystrybucja Ubuntu dla Pythona 3.5.2:

```
sudo apt-get install python3-pip
```

Po wpisaniu w konsoli systemowej:

```
ipython3 notebook
```

zostanie uruchomiony notatnik Jupyter.

Więcej informacji można znaleźć pod adresem:

- http://perseba.github.io/blog/jupyter-wprowadzenie.html

1.3 Pakiety

Funkcjonalność języka Python możemy rozszerzać o dodatkowe biblioteki – http://www.scipy.org/. Poniżej zostaną wymienione niektóre z nich:

- *SymPy* – obliczenia symboliczne,
- *SciPy* – obliczenia numeryczne, rozszerzenie możliwości pakietu *NumPy*,
- *Pandas* – odczyt/zapis i przekształcanie zbiorów danych,
- *matplotlib* – generowanie wykresów.

Aby zainstalować dowolny pakiet np. *SymPy* wystarczy wpisać poniższą komendę:

```
pip3 install sympy
```

Import wybranego pakietu można dokonać na kilka sposobów. Poniżej przykład jak załadować dwie biblioteki *SymPy* oraz *SciPy* aby obliczyć: $\sqrt{5}$. Zaznaczmy, że w podstawowej wersji Pythona nie ma wbudowanej funkcji do obliczeń pierwiastków kwadratowych.

```
from sympy import *
from scipy import *

print (sqrt(5))
```

```
## 2.2360679775
```

Gwiazdka oznacza, że importujemy wszystkie funkcje. Oczywiście jest możliwość aby wybrać tylko te które nas interesują. Jednak nazwy funkcji z różnych pakietów mogą się dublować. Na przykład funkcja *sqrt* jest w pakiecie *SymPy* oraz *SciPy*. W takich przypadkach będziemy mogli korzystać tylko z funkcji z ostatnio załadowanego pakietu tzn. *SciPy*. Aby zapobiec takiej sytuacji lepiej użyć innego sposobu.

```
import sympy
import scipy

print (sympy.sqrt(5))
print (scipy.sqrt(5))
```

```
## sqrt(5)
## 2.2360679775
```

Dzięki takiemu zabiegowi musimy zawsze deklarować z jakiej biblioteki będzie użyta funkcja. Inaczej mówiąc poprzedzamy nazwę funkcji (np. `sqrt`) nazwą pakietu z jakiego pochodzi. Aby zapobiec wpisywania za każdym razem całej nazwy pakietu można mu przypisać własną nazwę. Poniżej sytuacja w której dla funkcji

`sqrt` z pakietu *SymPy* nie odwołujemy się do nazwy biblioteki. Natomiast w przypadku korzystania z funkcji z pakietu *SciPy* już tak ale wykorzystujemy do tego celu własną nazwę.

```
from sympy import *
import scipy as sc

print (sqrt(5))
print (sc.sqrt(5))
```

```
## sqrt(5)
## 2.2360679775
```

Cechą charakterystyczą pakietu *SciPy* jest to, że zawiera on wszystkie funkcje z biblioteki *NumPy*. Dodatkowo posiada tzw. subpakiety które należy oddzielnie importować. Poniżej przykład załadowania subbiblioteki *scipy.special* zawierającej funkcje specjalne. Poniżej obliczona wartość funkcji gamma dla argumentu: 2.17.

```
import scipy.special as spec

print (spec.gamma(2.17))
```

```
## 1.08423854442
```

Aby sprawdzić jaka jest zainstalowana wersja pakietu np. *matplotlib* należy użyć poniższej komendy:

```
import pkg_resources

print (pkg_resources.get_distribution("matplotlib").version)
```

```
## 2.0.0
```

Aby zaktualizować pakiet np. *matplotlib* należy użyć poniższej komendy:

```
pip3 install --upgrade matplotlib
```

Wiele dodatkowych informacji na temat Pythona można znaleźć pod adresem:

- http://python101.readthedocs.io/pl/latest/index.html#

2 Zbiory danych

2.1 Ramka danych

Poniżej przykładowa ramka danych i jej zapis do pliku tekstowego *FUNDACJA.csv*:

```
import pandas as pd

f = pd.DataFrame({'X' :[1.2, 5.6, 5.8, 3.1], 'Y' :[1, 6, 8, 7]})
f.to_csv('FUNDACJA.csv')
```

Alternatywą do powyższego zapisu może być zapis:

```
import pandas as pd

f = pd.DataFrame()
f['X'] = [1.2, 5.6, 5.8, 3.1]
f['Y'] = [1, 6, 8, 7]
f.to_csv('FUNDACJA.csv')
```

Za pomocą pakietu `pandas` możemy zapisywać (wczytywać) dane także w innych formatach. Służą do tego funkcje: `read_excel` (arkusz kalkulacyjny excel), `read_sas` (pakiet SAS) i `read_stata` (pakiet STATA).

2.2 Przekształcanie danych

Podczas pracy z danymi istotną rolę odgrywają funkcje które umożliwiają czyszczenie i przekształcanie danych. Poniżej przykłady kilku podstawowych operacji na danych z pakietem `pandas`.

- jak otworzyć zbiór danych z pliku .csv ?

```
import pandas as pd

df = pd.read_csv("PKN.csv")[['Data','Zmiana']]
print(df.head(3))
```

```
##          Data    Zmiana
## 0  2010:10:29 -0.005951
## 1  2010:11:02  0.014706
## 2  2010:11:03  0.013047
```

- jak zamienić wszystkie znaki `:` na `-` ?

```
import pandas as pd

df = pd.read_csv("PKN.csv")[['Data','Zmiana']]
df['Data'] = df['Data'].str.replace(':','-')
print(df.head(3))
df.to_csv('PKN_v01.csv',index=False)
```

```
##          Data    Zmiana
## 0  2010-10-29 -0.005951
## 1  2010-11-02  0.014706
## 2  2010-11-03  0.013047
```

- jak dodać na końcu liczb znak `%` ?

```
import pandas as pd
```

```
df = pd.read_csv("PKN_v01.csv")
df['Procent'] = round(df['Zmiana']*100,4).astype(str) + ' %'
print(df.head())
```

```
##          Data    Zmiana    Procent
## 0  2010-10-29 -0.005951  -0.5951 %
## 1  2010-11-02  0.014706   1.4706 %
## 2  2010-11-03  0.013047   1.3047 %
## 3  2010-11-04  0.060104   6.0104 %
## 4  2010-11-05  0.031256   3.1256 %
```

- jak rozdzielić kolumnę `Data` na trzy inne ?

```
import pandas as pd

df = pd.read_csv("PKN_v01.csv")
df = df.join(df['Data'].str.split('-', 2, expand=True).\
     rename(columns={0:'Rok', 1:'Miesiac', 2:'Dzien'}))
print(df.head())
df.to_csv('PKN_v02.csv',index=False)
```

```
##          Data    Zmiana   Rok Miesiac Dzien
## 0  2010-10-29 -0.005951  2010      10    29
## 1  2010-11-02  0.014706  2010      11    02
## 2  2010-11-03  0.013047  2010      11    03
## 3  2010-11-04  0.060104  2010      11    04
## 4  2010-11-05  0.031256  2010      11    05
```

- jak scalić dwie kolumny `Rok` i `Miesiąc` w jedną ?

```
import pandas as pd

df = pd.read_csv("PKN_v02.csv")
df['Data'] = df['Rok'].astype(str)  + '-' + df['Miesiac'].astype(str)
print(df.head())
```

```
##       Data    Zmiana   Rok  Miesiac  Dzien
## 0  2010-10 -0.005951  2010       10     29
## 1  2010-11  0.014706  2010       11      2
## 2  2010-11  0.013047  2010       11      3
## 3  2010-11  0.060104  2010       11      4
## 4  2010-11  0.031256  2010       11      5
```

- jak przekształcać daty ?

```
import pandas as pd
import datetime
import time

df = pd.read_csv("PKN_v02.csv")
df['Data'] = pd.to_datetime(df['Data'])
df['Miesiac'] = df['Data'].apply(lambda x: x.strftime('%B'))
df['Dzien'] = df['Data'].apply(lambda x: x.strftime('%A'))
print(df.head(3))
df.iloc[:,1:5].to_csv('PKN_v03.csv',index=False)
```

```
##         Data    Zmiana   Rok   Miesiac      Dzien
## 0 2010-10-29 -0.005951  2010   October     Friday
```

```
## 1 2010-11-02  0.014706  2010  November    Tuesday
## 2 2010-11-03  0.013047  2010  November  Wednesday
```

- jak grupować dane ?

```
import pandas as pd

df = pd.read_csv("PKN_v03.csv")
df = df.groupby(['Rok','Miesiac','Dzien'],as_index=False)['Zmiana'].agg({'zm' : 'mean'})
print(df.head())
```

```
##     Rok   Miesiac      Dzien        zm
## 0  2010  December     Friday -0.007143
## 1  2010  December     Monday -0.003315
## 2  2010  December   Thursday  0.003981
## 3  2010  December    Tuesday  0.006987
## 4  2010  December  Wednesday  0.004976
```

Do agregowania danych zamiast pakietu `pandas` możemy używać także biblioteki `dplython` która jest wzorowana na pakiecie `dplyr` dla języka R.

```
from dplython import *
import pandas as pd

df = DplyFrame(pandas.read_csv('PKN_v03.csv'))
DF = df >> group_by(X.Rok,X.Miesiac,X.Dzien) >> summarize(zm = X.Zmiana.mean())
print(DF.head())
```

```
##     Rok   Miesiac      Dzien        zm
## 0  2010  December     Friday -0.007143
## 1  2010  December     Monday -0.003315
## 2  2010  December   Thursday  0.003981
## 3  2010  December    Tuesday  0.006987
## 4  2010  December  Wednesday  0.004976
```

- jak utworzyć szereg czasowy ?

```
import pandas as pd
from datetime import datetime

df = pd.read_csv("PKN_v02.csv")[['Data','Zmiana']]
df['Data'] = pd.to_datetime(df['Data'])
df.index = df['Data']
del df['Data']
print(df.head(3))
```

```
##              Zmiana
## Data
## 2010-10-29 -0.005951
## 2010-11-02  0.014706
## 2010-11-03  0.013047
```

Dzięki pakietowi `pandas_datareader.data` mamy dostęp do wielu ciekawych zbiorów danych dostępnych w internecie. Możemy pobierać dane z Eurostatu, Bank Światowego, OECD, serwisów internetowych np. Oanda, Yahoo!Finanse i inne.

Więcej informacji można znaleźć pod adresem:

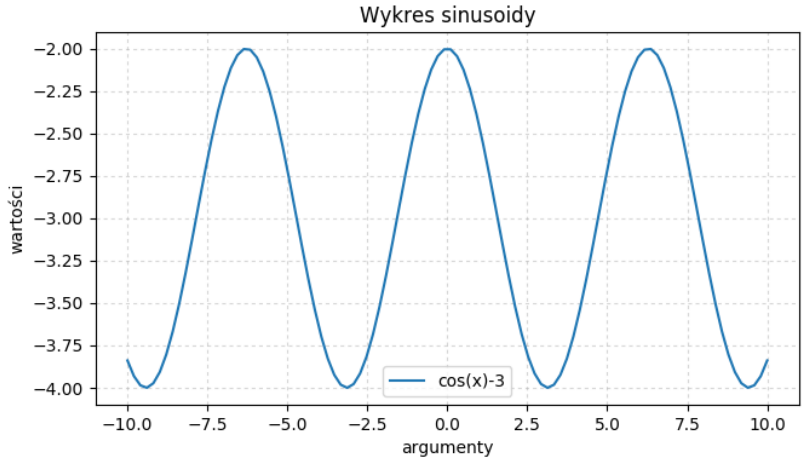
- https://pandas-datareader.readthedocs.io/en/latest/whatsnew.html

3 Grafika

3.1 Wykresy klasyczne

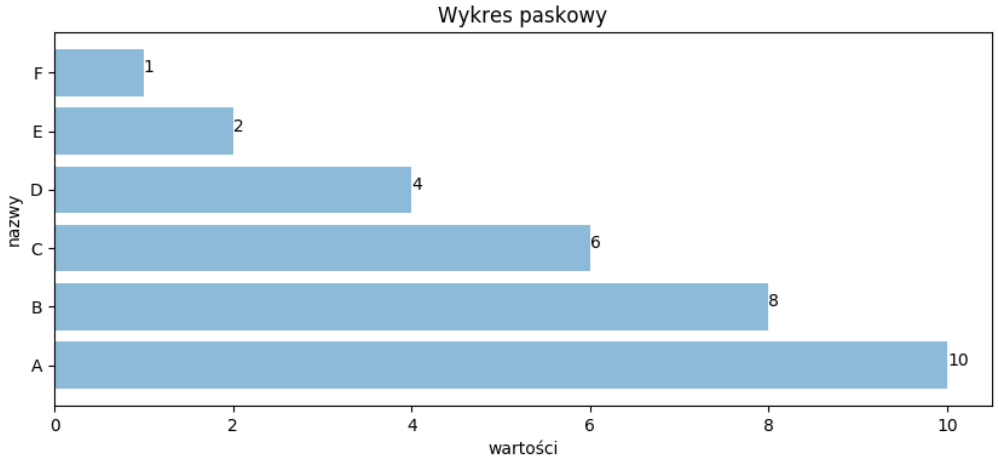
- Jak narysować wykres liniowy?

```
from scipy import *
import matplotlib.pyplot as plt

xdata = linspace(-10,10, 100)
x = linspace(-10,10, 10)
plt.figure(figsize=(7.5,4))
plt.grid(True, alpha=0.5, linestyle=':')
plt.title('Wykres sinusoidy')
plt.plot(xdata, cos(xdata)-3, label='cos(x)-3')
plt.xlabel('argumenty')
plt.ylabel('wartości')
plt.legend()
plt.savefig('matplotlib01.png')
```



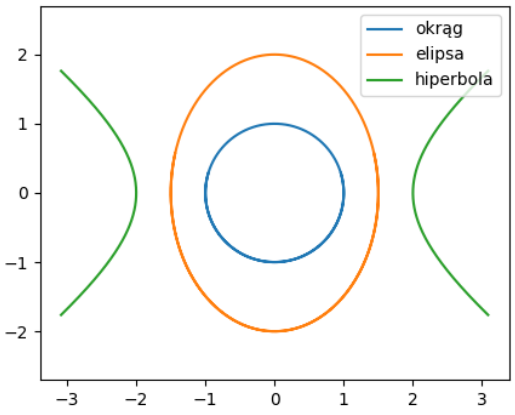
- Jak narysować wykres paskowy?

```
from scipy import *
import matplotlib.pyplot as plt

objects = ('A', 'B', 'C', 'D', 'E', 'F')
y_pos = arange(len(objects))
performance = [10,8,6,4,2,1]

plt.figure(figsize=(10,4))
plt.barh(y_pos, performance, align='center', alpha=0.5)
plt.yticks(y_pos, objects)
for a,b in zip(performance, y_pos):
    plt.text(a,b, str(a))
plt.xlabel('wartości')
plt.ylabel('nazwy')
plt.title('Wykres paskowy')
plt.savefig('paski01.png')
```



- Jak narysować wykres krzywej?

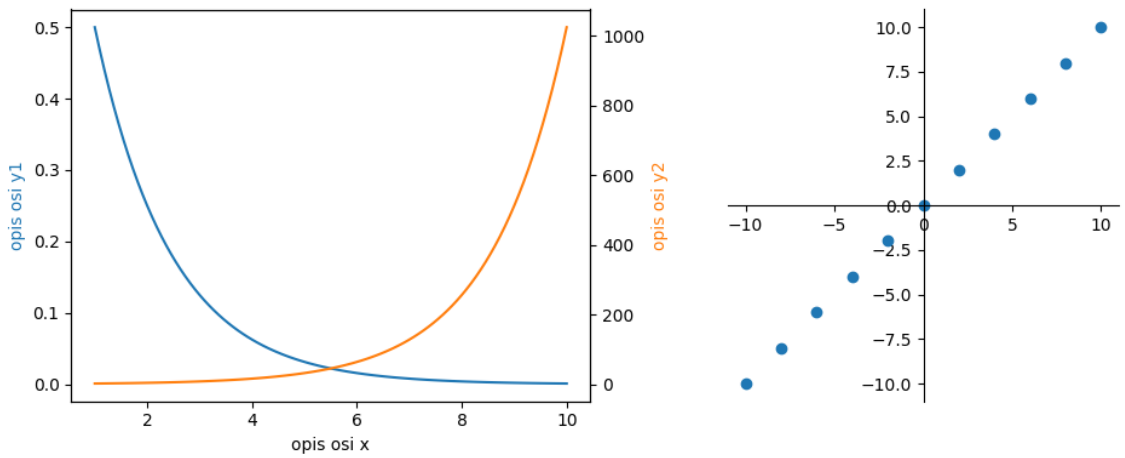
```
from scipy import *
import matplotlib.pyplot as plt

plt.figure(figsize=(5,4))
t = linspace(-5,5,1000)
plt.plot(sin(t), cos(t), label='okrąg')
plt.plot(1.5*sin(t), 2*cos(t), label='elipsa')
t = linspace(-1,1,1000)
plt.plot(2*cosh(t), 1.5*sinh(t), color='C2',label='hiperbola')
plt.plot(-2*cosh(t), -1.5*sinh(t), color='C2')
plt.axes().set_aspect('equal', 'datalim')
plt.legend()
plt.savefig('krzywe01.png')
```



- Jak rozmieszczać osie?

```
from scipy import *
import matplotlib.pyplot as plt

x = linspace(1,10, 100)
```

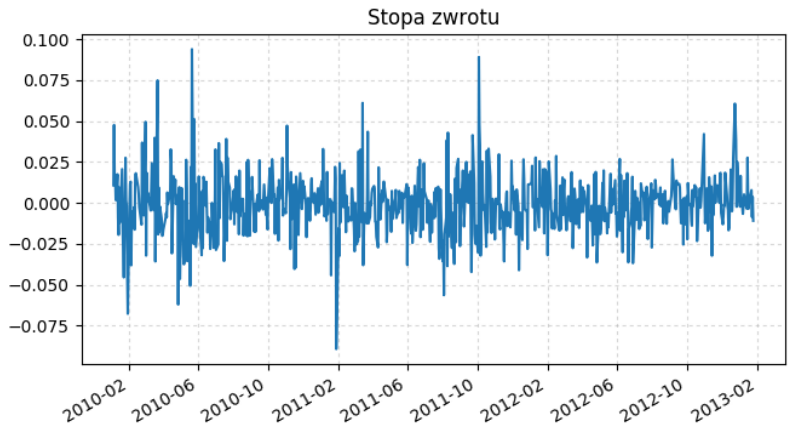
```
y = 2**(-x)
z = 2**x

fig = plt.figure(figsize=(10,4))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = ax1.twinx()
ax1.plot(x,y,color='C0',label='f(x)')
ax2.plot(x,z,color='C1',label='g(x)')
ax1.set_ylabel('opis osi y1', color='C0')
ax2.set_ylabel('opis osi y2', color='C1')
ax1.set_xlabel('opis osi x')
ax = plt.subplot(1, 2, 2, aspect='equal')
ax = plt.gca()
ax.plot(linspace(-10,10, 11), linspace(-10,10, 11),'o')
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.tight_layout()
plt.savefig('osie01.png')
```



- Jak narysować szereg czasowy?

```
from scipy import *
import matplotlib.pyplot as plt
from matplotlib import dates
import pandas_datareader.data as web
import pandas as pd
from datetime import datetime
import scipy.stats as stats

start = datetime(2010, 1, 1)
end = datetime(2013, 1, 27)
dft = web.DataReader("F", 'yahoo', start, end)
dft['return'] = log(dft['Close']/dft['Open'])

plt.figure(figsize=(7.5,4))
plt.grid(True, alpha=0.5, linestyle=':')
```

```
plt.title('Stopa zwrotu')
plt.plot(dft['return'])
plt.gcf().autofmt_xdate()
plt.savefig('matplotlib02.png')
```

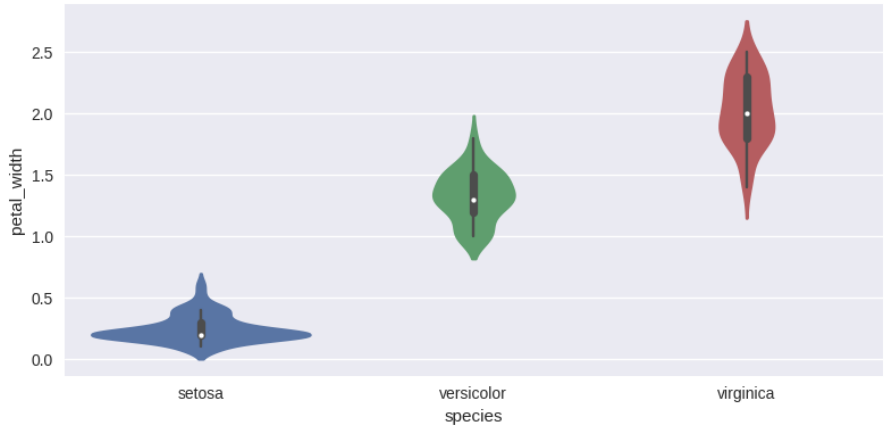


3.2 Wykresy niestandardowe

- Jak narysować wykres wiolinowy?

```
import warnings
warnings.filterwarnings("ignore")

import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset("iris")
print(iris.head(12))

plt.figure(figsize=(8,4))
sns.violinplot(x = 'species', y = 'petal_width', data = iris)
plt.tight_layout()
plt.savefig('seaborn01.png')
```

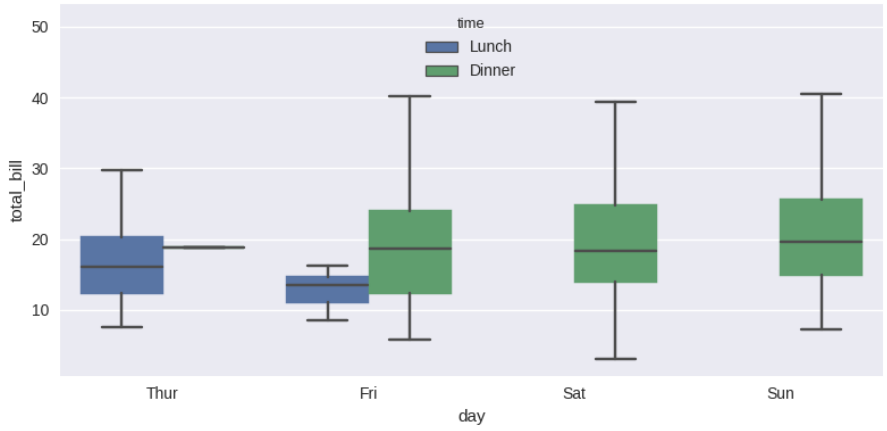
```
##     sepal_length  sepal_width  petal_length  petal_width species
## 0            5.1          3.5           1.4          0.2  setosa
## 1            4.9          3.0           1.4          0.2  setosa
## 2            4.7          3.2           1.3          0.2  setosa
## 3            4.6          3.1           1.5          0.2  setosa
## 4            5.0          3.6           1.4          0.2  setosa
## 5            5.4          3.9           1.7          0.4  setosa
## 6            4.6          3.4           1.4          0.3  setosa
## 7            5.0          3.4           1.5          0.2  setosa
## 8            4.4          2.9           1.4          0.2  setosa
## 9            4.9          3.1           1.5          0.1  setosa
## 10           5.4          3.7           1.5          0.2  setosa
## 11           4.8          3.4           1.6          0.2  setosa
```



- Jak narysować boxplot z podziałem na grupy?

```
import warnings
warnings.filterwarnings("ignore")

import seaborn as sns
import matplotlib.pyplot as plt

tips = sns.load_dataset("tips")
print(tips.head(4))

plt.figure(figsize=(8,4))
sns.boxplot(x="day", y="total_bill", hue="time", data=tips)
plt.tight_layout()
plt.savefig('seaborn02.png')
```

```
##    total_bill   tip     sex smoker  day    time  size
## 0       16.99  1.01  Female     No  Sun  Dinner     2
## 1       10.34  1.66    Male     No  Sun  Dinner     3
## 2       21.01  3.50    Male     No  Sun  Dinner     3
## 3       23.68  3.31    Male     No  Sun  Dinner     2
```



4 Elementy matematyki

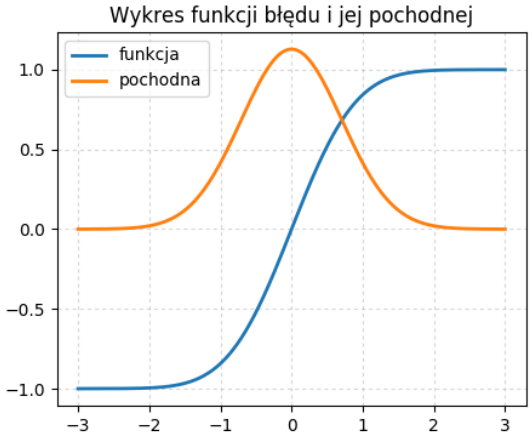
4.1 Pochodna

Wyznaczenie postaci analitycznej pochodnej funkcji nie jest zawsze możliwe. Za przykład niech posłuży funkcja błędu o postaci:

$$f(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\, dt$$

W takiej sytuacji można wykorzystać metody numeryczne aby naszkicować wykres pochodnej funkcji. Poniżej przykład jak to zrobić.

```
from scipy import *
import matplotlib.pyplot as plt
import scipy.special as spec
from scipy.misc import derivative

f = lambda x: spec.erf(x)
d = lambda x: derivative(f, x, dx=1e-6)

plt.figure(figsize=(5,4))
plt.grid(True, alpha=0.5, linestyle=':')
plt.title('Wykres funkcji błędu i jej pochodnej')
X = linspace(-3, 3, 500)
plt.plot(X,f(X), linewidth=2,label='funkcja')
plt.plot(X,d(X), linewidth=2,label='pochodna')
plt.legend()
plt.savefig('pochodna01.png')
```



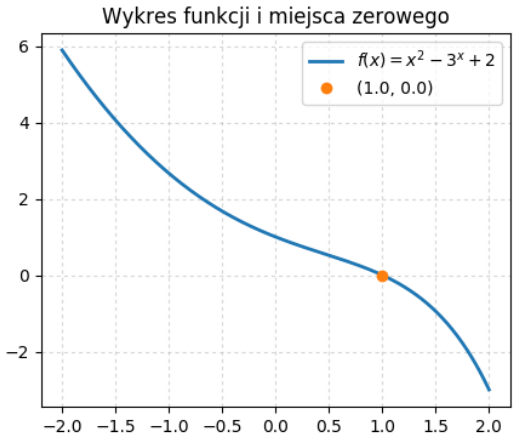
4.2 Miejsce zerowe

Miejsca zerowe pewnych funkcji nie można wyznaczyć za pomocą obliczeń symbolicznych. W takich sytuacjach dobrze sprawdzają się metody numeryczne: `brentq`, `brenth`, `ridder`, `bisect`, `newton`. Poniżej przykład z wykorzystaniem funkcji `newton`.

$$f(x) = x^2 - 3^x + 2$$

```
from scipy import *
from scipy.optimize import *
import matplotlib.pyplot as plt

f = lambda x: x**2-3**x+2
sol = newton(f, 1)
print (sol)

plt.figure(figsize=(5,4))
plt.grid(True, alpha=0.5, linestyle=':')
plt.title('Wykres funkcji i miejsca zerowego')
X = linspace(-2,2, 500)
plt.plot(X,f(X), linewidth=2,label=r'$f(x)=x^2-3^x+2$')
plt.plot(sol,0,'o',label='(%.1f, %.1f)' % (sol,f(sol)))
plt.legend()
plt.savefig('mz01.png')
```

```
## 1.0
```

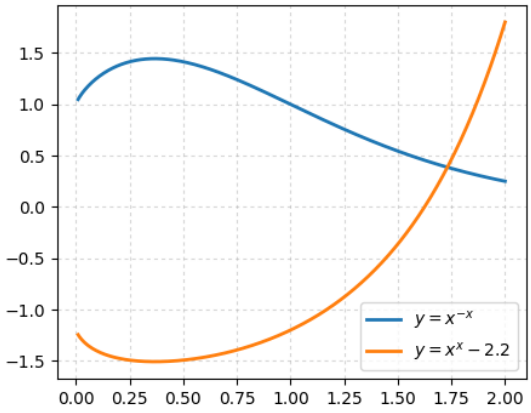


4.3 Układ równań

Do rozwiązywania nieliniowych układów równań służą dwie funkcje: `fsolve` oraz `root` z pakietu `scipy.optimize` – często są też wykorzystywane do obliczania miejsc zerowych funkcji. Poniżej przykład rozwiązania układu równań o postaci:

$$\begin{cases} y = x^{-x} \\ y = x^x - 2.2 \end{cases}$$

```
from scipy import *
from scipy.optimize import *

def eq(X):
    x, y = X
    eq1 = x**(-x)-y
    eq2 = x**x-2.2-y
    return [eq1, eq2]
```

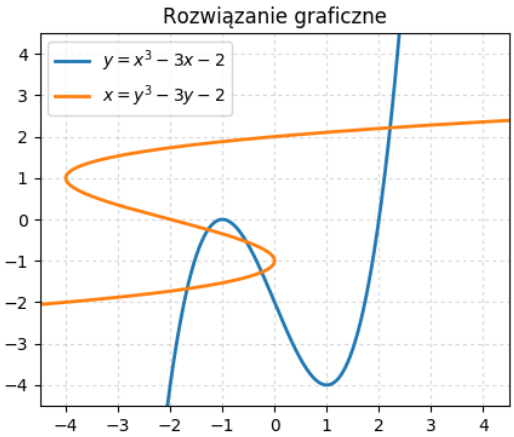



```
sol = fsolve(eq,[1.75,0.5]);
print (sol)
```

```
## [ 1.73135413  0.38660687]
```

Często się zdarza, że układ równań ma kilka rozwiązań. W takich sytuacjach wygodnie jest wykorzystać pętle `for` ponieważ funkcje `fsolve` i `root` mogą wykonać obliczenia tylko dla jednego punktu startowego. Za przykład posłuży nam układu równań o postaci:

$$\begin{cases} y = x^3 - 3x - 2 \\ x = y^3 - 3y - 2 \end{cases}$$



Na podstawie wykresu możemy założyć, że układ równań ma pięć rozwiązań. Zatem będziemy szukać tych współrzędnych na podstawie pięciu punktów startowych:

$$(2, 2),\ (-1.5, -1.5),\ (-1, -0.5),\ (-0.5, -0.5),\ (-0.5, -1)$$

```
from scipy import *
from scipy.optimize import *
from pandas import *
```

```
def eq(X):
    x, y = X
    eq1 = (x**3-3*x-2)-y
    eq2 = (y**3-3*y-2)-x
    return [eq1, eq2]

df = DataFrame({'x' :[2,-1.5,-1,-0.5,-0.5], 'y' :[2,-1.5,-0.5,-0.5,-1]})
sol = [ fsolve(eq, df.reindex([i]) ) for i in range(5) ]
df = DataFrame(sol, columns=['x','y'])
print (df)
```

```
##           x         y
## 0  2.214320  2.214320
## 1 -1.675131 -1.675131
## 2 -1.274551 -0.246829
## 3 -0.539189 -0.539189
## 4 -0.246829 -1.274551
```

W przypadku gdy nie wiemy jakie punkty startowe wybrać możemy zastosować symulację monte carlo z wykorzystaniem np. rozkładu jednostajnego. Tym razem obliczenia wykonamy za pomocą funkcji `root` dzięki której oprócz rozwiązania otrzymujemy kilka dodatkowych informacji np. o tym czy algorytm zakończył działanie pomyślnie: `success=True` lub nie: `success=False`. Wykorzystamy tę informacje do filtrowania uzyskanych rozwiązań.

```
from scipy import *
from scipy.optimize import *
from pandas import *

def eq(X):
    x, y = X
    eq1 = (x**3-3*x-2)-y
    eq2 = (y**3-3*y-2)-x
    return [eq1, eq2]

B = 100
sol = [ root(eq, [random.uniform(-4,4,2)]) for i in range(B) ]
s = [ sol[i].success for i in range(B) ]
df = DataFrame({'s' :s})
S = [ sol[i].x for i in df[df['s'] == True].index.values ]
DF = DataFrame(S, columns=['x','y'])
DF = DF.round(3)
DF = DF.drop_duplicates()
print (DF)
```

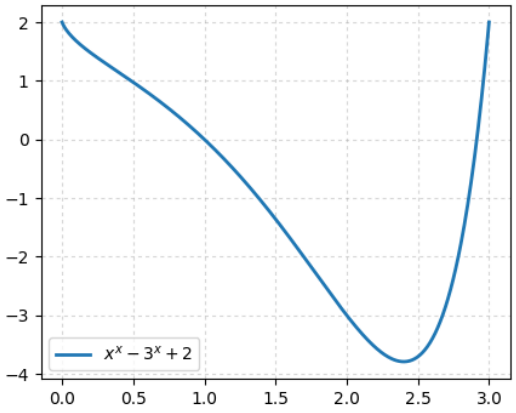
```
##         x      y
## 0  -1.675 -1.675
## 3  -1.275 -0.247
## 5  -0.247 -1.275
## 8   2.214  2.214
## 16 -0.539 -0.539
```

4.4 Ekstremum

Aby wyznaczyć minimum jednowymiarowej funkcji o postaci:

$$f(x) = x^x - 3^x + 2$$

można posłużyć się takimi funkcjami jak: `brent`, `fminbound`, `golden`. Poniżej przykład szukania mimimum funkcji $f(x)$ metodą `brent`.




```
from scipy import *
from scipy.optimize import *

f = lambda x: x**x-3**x+2

res = fminbound(f, 0,3)
fun = f(res)
print ('Dla argumentu x:', ("%.7f" % res), 'funkcja osiąga minimum:', ("%.7f" % fun))
```

```
## Dla argumentu x: 2.4007346 funkcja osiąga minimum: -3.7912525
```

4.5 Całka oznaczona

$$\int_{0.25}^{1} x^x$$

```
from scipy import *
from scipy.integrate import *

f = lambda x: x**x

sol = quad(f, 0.25,1)
print (sol)
```

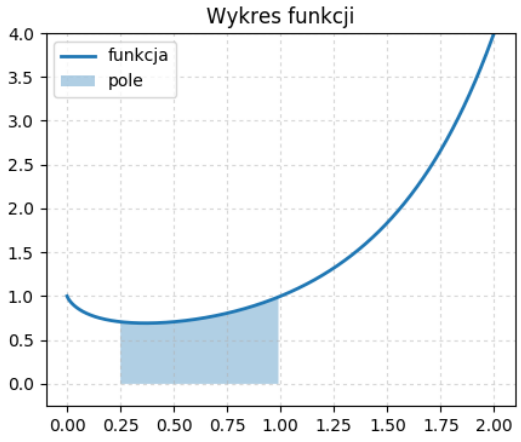
```
## (0.5851010692820607, 6.495926788497427e-15)
```

W pewnych sytuacjach do obliczania całek oznaczonych wygodnie jest zastosować metodę monte carlo. Do tego celu wykorzystamy generator liczb losowych z rozkładu jednostajnego o parametrach $min = 2$ oraz $max = 5$. Liczba 3 oznacza szerokość przedziału całkowania.

```
from scipy import *

f = lambda x: x**x

sol = mean([f(random.uniform(0.25,1)) for x in range(10000)])*0.75
print (sol)
```

```
## 0.585041655194
```

```
from scipy import *
import matplotlib.pylab as plt

f = lambda x: x**x

a, b = 0.25, 1

x = linspace(0,2, 10000)
y = f(x)

plt.figure(figsize=(5,4))
plt.grid(True, alpha=0.5, linestyle=':')
plt.ylim(-0.25,4)
plt.title('Wykres funkcji')
plt.plot(x, y, linewidth = 2, label='funkcja')
plt.fill_between(x=arange(a,b,0.01), y1=f(arange(a,b,0.01)), alpha=0.35, label='pole')
plt.legend()
plt.savefig('calka01.png')
```



5 Optymalizacja

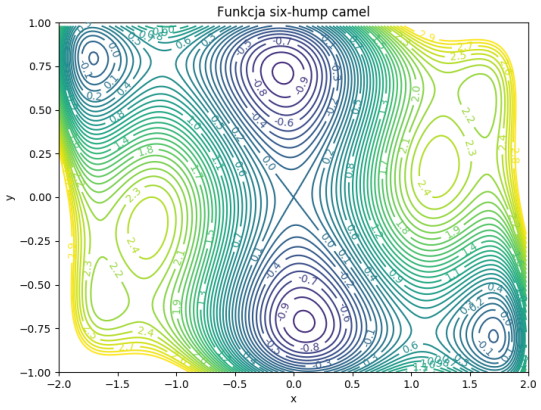
5.1 Funkcja six-hump camel

W przypadku gdy funkcja ma kilka wartości ekstremalnych (i chcemy je wszystkie znaleźć) to dobrym rozwiązaniem jest wykorzystanie symulacji monte carlo. Dla każdej interacji parametry startowe będą losowane z rozkładu jednostajnego z uwzględnieniem ograniczeń pudełkowych. Poniżej przykład z wykorzystaniem funkcji six-hump camel:

$$F(\mathbf{x}) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1 x_2 + (-4 + 4x_2^2)x_2^2$$

$$\begin{bmatrix} -2 \\ -1 \end{bmatrix} \leq \mathbf{x} \leq \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

```
from scipy import *
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
delta = 0.02
x, y = meshgrid(arange(-2, 2, delta), arange(-1, 1, delta))
z = (4-2.1*x**2+x**4/3)*x**2+x*y+(-4+4*y**2)*y**2
levels = arange(-1.6,3.0,0.1)
CS = plt.contour(x, y, z, levels=levels)
plt.clabel(CS, fontsize=10, inline=1, fmt='%1.1f')
plt.title('Funkcja six-hump camel')
plt.xlim(-2, 2)
plt.ylim(-1, 1)
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('camel01.png')
```




```
from scipy import *
from scipy.optimize import *
from pandas import *

def F(x):
    return (4-2.1*x[0]**2+x[0]**4/3)*x[0]**2+x[0]*x[1]+(-4+4*x[1]**2)*x[1]**2
```

```
bnds = ((-2,2), (-1,1))
B = 300
res = [minimize(F, [random.randint(-1,1,1),random.randint(-2,2,1)],\
       method='L-BFGS-B', bounds=bnds)\
      for i in range(B) ]

Xv = [res[i]['x'] for i in range(B)]
Fv = [res[i]['fun'] for i in range(B)]
df = DataFrame(Xv, columns=['x1','x2'])
df = df.assign(Fv= Fv)
df = df.sort_values(by=['Fv'], ascending=[True])
df = df.round(2)
df = df.drop_duplicates()
print (df.head())
```

```
##        x1    x2    Fv
## 191  0.09 -0.71 -1.03
## 144 -0.09  0.71 -1.03
## 269  0.00  0.00  0.00
```

5.2 Funkcja Eggholder

W pakiecie `scipy.optimize` jest zaimplementowanych kilka stosunkowo nowych technik do optymalizacji funkcji nieliniowych. Wzorowane są na zjawiskach fizycznych lub są zaczerpnięte z obszaru nauk biologicznych. Przykładem może być metoda ewolucji różnicowej dla której źródłem inspiracji były procesy ewolucji występujące w przyrodzie – funkcja `differential_evolution`. Inną ciekawą techniką jest metoda symulowanego wyżarzania która powstała na podstawie obserwacji zjawiska wyżarzania w metalurgii. Sprawdza się ona dobrze w przypadku optymalizacji tzw. funkcji chropowatych (duża deformacja powierzchni) gdzie konwencjonalne metody gradientowe często mają problem ze znalezieniem minimum globalnego. Dobrą alternatywą dla tej metody jest technika "basin-hopping" – funkcja `basinhopping` która domyślnie wykorzystuje algorytm minimalizacji lokalnej `BFGS`. Poniżej przykład optymalizacji funkcji Eggholder – szukanie jej minimum:

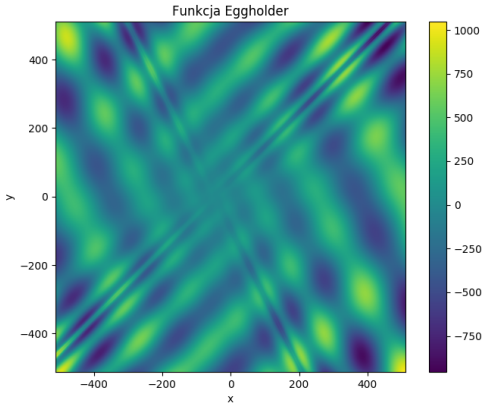
$$F(\mathbf{x}) = -(x_2+47)\sin\left(\sqrt{\left|x_2+\frac{x_1}{2}+47\right|}\right) - x_1\sin\left(\sqrt{|x_1-(x_2+47)|}\right)$$
$$-512 \leq \mathbf{x} \leq 512$$

```
from scipy import *
from scipy.optimize import *

def F(x):
    return -(x[1]+47)*sin(sqrt(abs(x[1]+x[0]/2+47)))-x[0]*sin(sqrt(abs(x[0]-(x[1]+47))))

l_bfgs_b = {"method":"L-BFGS-B","bounds":[(-512,512),(-512,512)]}
res = basinhopping(F, (1,1), minimizer_kwargs=l_bfgs_b, niter=5000)
print (res.x)
print (res.fun)
```

```
## [ 512.          404.23180507]
## -959.640662721
```

```
from scipy import *
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(8,6))
x, y = mgrid[-512:512, -512:512]
z = -(y+47)*sin(sqrt(abs(y+x/2+47)))-x*sin(sqrt(abs(x-(y+47))))
plt.title('Funkcja Eggholder')
plt.xlabel('x')
plt.ylabel('y')
plt.imshow(z, origin='lower', interpolation='bicubic',\
           extent=(x.min(), x.max(), y.min(), y.max()))
plt.colorbar()
plt.savefig('Eggholder01.png')
```



5.3 Hock and Schittkowski problem 71

Jeśli chcemy uzyskać jak najbardziej dokładne rozwiązanie (algorytmy oparte o symulacje wyszukują przybliżone rozwiązanie) to otrzymane parametry optymalne możemy wykorzystać jako wartości wyjściowe w innej metodzie optymalizacyjnej np. `SLSQP` (Sequential Least SQuares Programming) lub `COBYLA` (Constrained Optimization BY Linear Approximation). W przypadku sekwencyjnego programowania kwadratowego możemy stosować dwa rodzaje ograniczeń: `cons` (równościowe i nierównościowe) oraz `bounds` (nierównościowe typu pudełkowego). Dla algorytmu `COBYLA` jest przewidziana tylko opcja `cons` obsługująca wyłącznie ograniczenia nierównościowe.Poniżej przykład [Hock and Schittkowski problem 71] który zostanie rozwiązany za pomocą sekwencyjnego programowania kwadratowego.

$$F(\mathbf{x}) = x_1 x_4 (x_1 + x_2 + x_3) + x_3$$
$$1 \quad \leq \mathbf{x} \leq \quad 5$$

ograniczenie równościowe i nierównościowe:

$$\begin{aligned} x_1^2 + x_2^2 + x_3^2 + x_4^2 &= 40 \\ x_1 x_2 x_3 x_4 &\leq 25 \end{aligned}$$

```
from scipy import *
from pandas import *
from scipy.optimize import *

def G(x):
    return x[0]*x[3]*(x[0]+x[1]+x[2])+x[2]
```

```
bnds = ((1,5), (1,5), (1,5), (1,5))
cons = ({'type': 'eq', 'fun': lambda x:  x[0]**2+x[1]**2+x[2]**2+x[3]**2-40},
        {'type': 'ineq', 'fun': lambda x: -x[0]*x[1]*x[2]*x[3]+25})
res = minimize(G,[1,1,1,1],method='SLSQP', bounds=bnds, constraints=cons)
print (res)
```

```
##      fun: 13.211102550928462
##      jac: array([ 10.60555124,   1.        ,   2.        ,   9.60555124,   0.        ])
##  message: 'Optimization terminated successfully.'
##     nfev: 43
##      nit: 7
##     njev: 7
##   status: 0
##  success: True
##        x: array([ 1.        ,  5.        ,  3.60555128,  1.        ])
```

5.4 Funkcja kwadratowa

Optymalizacja funkcji kwadratowej z ograniczeniami liniowymi to szczególny przypadek programowania nieliniowego. Warto zaznaczyć, że rozwiązanie tak sformułowanego problemu można znaleźć wykonując obliczenia na macierzach bez korzystania z algorytmów interacyjnych. Poniżej przykład optymalizacji funkcji kwadratowej (szukanie jej minimum/maksimum) z uwzględnieniem ograniczeń liniowych z wykorzystaniem pakietu `cvxopt`.

Funkcja kwadratowa:

$$F(\mathbf{x}) = 3x_1^2 + 4x_2^2 + 2x_1x_2 - 2x_1 + 3x_2$$

ograniczenia liniowe:

$$\begin{aligned} x_1 + 2x_2 &= 3 \\ 2x_1 + x_2 &\leq 4 \\ x_i &\geq 0 \quad i = 1, 2 \end{aligned}$$

```
from cvxopt import *

P = matrix([[6.0, 2.0],[2.0, 8.0]])
q = matrix([-2.0, 3.0])
G = matrix([[-1.0, 0.0, 2.0],
            [0.0, -1.0, 1.0]])
h = matrix([0.0, 0.0, 4.0])
A = matrix([[1.0],
            [2.0]])
b = matrix([3.0])
sol = solvers.qp(P,q,G,h,A,b, options = {'show_progress' : False})
print (sol['x'])
print (sol['primal objective'])
```

```
## [ 1.08e+00]
## [ 9.58e-01]
##
## 9.979166666666668
```

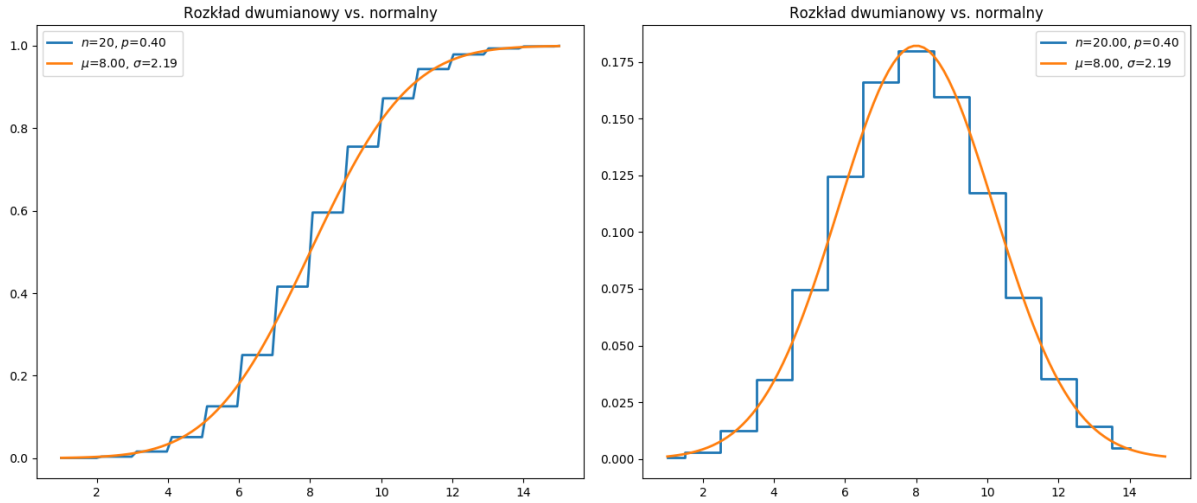
6 Elementy statystyki

6.1 Rozkład normalny

Rozkład normalny to jeden z ważniejszych rozkładów prawdopodobieństwa ponieważ za jego pomocą możemy przybliżać rozkłady zmiennej losowej ciągłej oraz skokowej. Przykładowo rozkład dwumianowy $B(n,p)$ zbiega do rozkładu normalnego dla $np \geq 5$ oraz $n(1-p) \geq 5$.

$$B(n,p) \approx N\left(np,\ \sqrt{np(1-p)}\right)$$

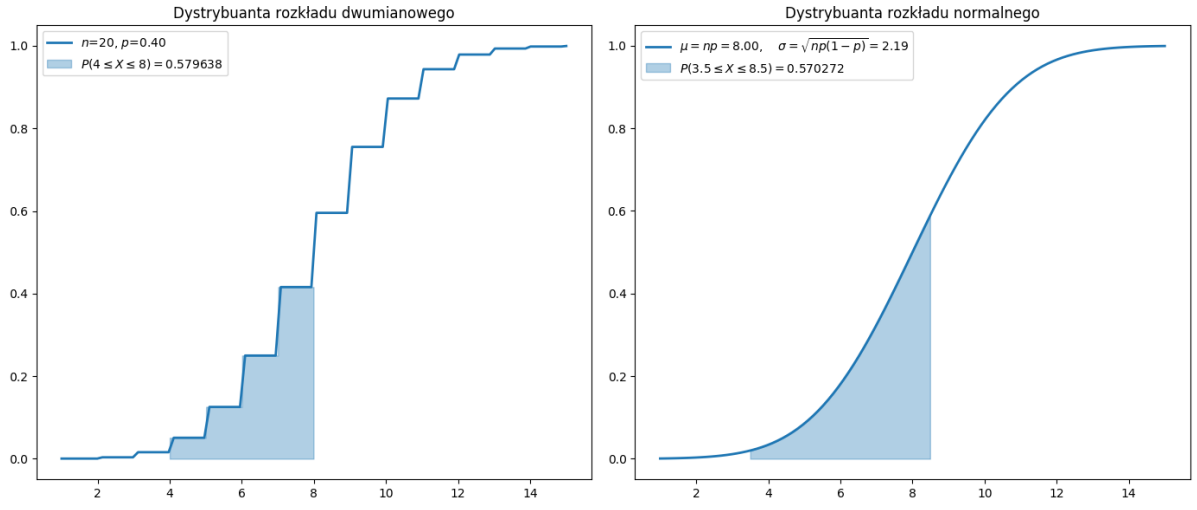
```
from scipy import *
import scipy.stats as stats
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(14,6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)

X = linspace(1,15, 100)
ax1.plot(X, stats.binom.cdf(X, n= 20, p= 0.4), '-', color='C0',
linewidth=2,label='$n$=%.0f, $p$=%.2f' % (20,0.4))
ax1.plot(X, stats.norm.cdf(X, loc= 20*0.4, scale= sqrt(20*0.4*0.6)), '-', color='C1',
linewidth=2,label='$\mu$=%.2f, $\sigma$=%.2f' % (20*0.4, sqrt(20*0.4*0.6)))
ax1.set_title("Rozkład dwumianowy vs. normalny")
ax1.legend()
X = range(1,15)
ax2.plot(X, stats.binom.pmf(X, n= 20, p= 0.4), color='C0',linestyle='steps-mid',
linewidth=2,label='$n$=%.2f, $p$=%.2f' % (20,0.4))
X = linspace(1,15, 100)
ax2.plot(X, stats.norm.pdf(X, loc= 20*0.4, scale= sqrt(20*0.4*0.6)), '-', color='C1',
linewidth=2,label='$\mu$=%.2f, $\sigma$=%.2f' % (20*0.4, sqrt(20*0.4*0.6)))
ax2.set_title("Rozkład dwumianowy vs. normalny")
ax2.legend()
fig.tight_layout()
fig.savefig('binom01.png')
```



Warto zaznaczyć, że w przypadku przybliżania rozkładu dyskretnego (np. dwumianowy) rozkładem normalnym stosujemy poprawkę na ciągłość.

$$P(a-0,5 \leq X \leq b+0,5) = \int_{a-0,5}^{b+0,5} N(np, \sqrt{np(1-p)})$$

```
from scipy import *
import scipy.stats as stats
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(14,6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)

n = 20
p = 0.4

X = linspace(1,15, 100)
ax1.plot(X, stats.binom.cdf(X, n, p), '-', color='C0',
linewidth=2, label='$n$=%.0f, $p$=%.2f' % (n,p))
ax1.fill_between(x= arange(4,8,0.01), label= '$P(4\leq X\leq 8) = 0.579638$',
                 y1= stats.binom.cdf(arange(4,8,0.01), n, p),
                 facecolor='C0', color='C0', alpha=0.35)
ax1.set_title("Dystrybuanta rozkładu dwumianowego")
ax1.legend()

ax2.plot(X, stats.norm.cdf(X, loc= n*p, scale= sqrt(n*p*(1-p))), '-', color='C0',
linewidth= 2, label= '$\mu=np=$%.2f, $\quad\sigma=\sqrt{np(1-p)}=$%.2f' % (n*p, sqrt(n*p*(1-p))))
ax2.fill_between(x= arange(4-0.5,8+0.5,0.01),  label= '$P(3.5\leq X\leq 8.5) = 0.570272$',
                 y1= stats.norm.cdf(arange(4-0.5,8+0.5,0.01),n*p, sqrt(n*p*(1-p))),
                 facecolor='C0', color='C0', alpha=0.35)
ax2.set_title("Dystrybuanta rozkładu normalnego")
ax2.legend()
fig.tight_layout()
fig.savefig('pois01.png')
```



```
from scipy import *
import scipy.stats as stats

n= 20; p= 0.4
bin = stats.binom.cdf(8, n= n, p= p)- stats.binom.cdf(3, n= n, p= p)
nor = stats.norm.cdf(8+0.5, loc= n*p, scale= sqrt(n*p*(1-p)))- \
      stats.norm.cdf(4-0.5, loc= n*p, scale= sqrt(n*p*(1-p)))
print('P(4 <= x <= 8): binomial p-value = %.6f, normal p-value = %.6f' % (bin,nor))
```

```
## P(4 <= x <= 8): binomial p-value = 0.579638, normal p-value = 0.570272
```

6.2 Rozkład chi-kwadrat

Centralny rozkład chi-kwadrat jest szczególnym przypadkiem niecentralnego rozkładu chi-kwadrat tzn.

$$\chi^2_{df} = \chi^2_{df,\ ncp=0}$$

gdzie:

- dla χ^2_{df} mamy: $\bar{x} = df$ oraz $s^2 = 2df$,
- dla $\chi^2_{df,ncp}$ mamy: $\bar{x} = df + ncp$ oraz $s^2 = 2df + 4ncp$.

Rozkład chi-kwadrat to także szczególny przypadek rozkładu gamma:

$$\chi^2_{df} = \Gamma(shape = df/2,\ scale = 2) \quad \text{lub} \quad \Gamma(shape = df/2,\ rate = 1/2)$$

gdzie:

- dla $\Gamma(shape,\ scale)$ mamy: $\bar{x} = shape \cdot scale$ oraz $s^2 = shape \cdot scale^2$,
- dla $\Gamma(shape,\ rate)$ mamy: $\bar{x} = shape/rate$ oraz $s^2 = shape/rate^2$.

Poniżej przykład oszacowania parametrów rozkładu chi-kwadrat oraz gamma dla zmiennej `c2`.

```
from scipy import *
import scipy.stats as stats

c2 = stats.chi2.rvs(df= 7, size= 1000, random_state= 2305)
fitC2 = stats.chi2.fit(c2)
logLik = sum(stats.chi2.logpdf(c2, df=fitC2[0], loc=fitC2[1], scale=fitC2[2]))
print('Rozkład chi-kwadrat:')
print('szukane parametry: df= %.6f, loc= %.6f, scale= %.6f' % (fitC2[0], fitC2[1], fitC2[2]))
print('logarytm wairygodności: %.6f' % logLik, '\n')
fitG = stats.gamma.fit(c2)
logLik = sum(stats.gamma.logpdf(c2, a=fitG[0], loc=fitG[1], scale=fitG[2]))
print('Rozkład gamma:')
print('szukane parametry: shape= %.6f, loc= %.6f, scale= %.6f' % (fitG[0], fitG[1], fitG[2]))
print('logarytm wairygodności: %.6f' % logLik)
```

```
## Rozkład chi-kwadrat:
## szukane parametry: df= 6.867740, loc= 0.040242, scale= 0.991986
## logarytm wairygodności: -2616.535734
##
## Rozkład gamma:
## szukane parametry: shape= 3.433912, loc= 0.040211, scale= 1.983955
## logarytm wairygodności: -2616.535734
```

Poniżej przykład obliczenia prawdopodobieństwa $P(X \leq 6)$ dla rozkładu $\chi^2_{df=12}$ oraz $\Gamma(shape = 6,\ scale = 2)$:

```
from scipy import *
import scipy.stats as stats

pC2 = stats.chi2.cdf(6, df= 12)
pG = stats.gamma.cdf(6, a= 12/2, scale= 2)
print('Dla rozkładu chi-kwadrat: df= 12:')
print(' P(x<=6) = %.6f' % pC2)
print('Dla rozkładu gamma: shape= 6, loc= 0, scale= 2:')
print(' P(x<=6) = %.6f' % pG)
```

```
## Dla rozkładu chi-kwadrat: df= 12:
##  P(x<=6) = 0.083918
## Dla rozkładu gamma: shape= 6, loc= 0, scale= 2:
##  P(x<=6) = 0.083918
```

6.3 Test proporcji

Dla dużych prób i proporcji z próby leżącej pomiędzy $0,2$ a $0,8$ stosujemy asymptotyczny test Score lub Walda.

$$Z_{\text{Score}} = \frac{\bar{p} - p_0}{\sqrt{p_0(1-p_0)}}\sqrt{n} \quad \text{lub} \quad Z_{\text{Wald}} = \frac{\bar{p} - p_0}{\sqrt{\bar{p}(1-\bar{p})}}\sqrt{n}$$

Dla danych:

$$n = 100,\ m = 40 \quad \text{czyli} \quad \bar{p} = 40/100 = 0,4$$

przeprowadzimy weryfikscję hipotezy zerowej:

$$H_0:\ p = 0,5 \quad \text{vs} \quad H_1:\ p \neq 0,5$$

za pomocą dwóch wyżej wymienionych testów asymptotycznych.

```
from statsmodels.stats.proportion import *

sol = proportions_ztest(40, 100, value= 0.5)
print('Test Walda: z = %.4f, p-value = %.6f' % (sol[0],sol[1]))
sol = proportions_ztest(40, 100, value= 0.5, prop_var= 0.5)
print('Test Score: z = %.4f, p-value = %.6f' % (sol[0],sol[1]))
```

```
## Test Walda: z = -2.0412, p-value = 0.041227
## Test Score: z = -2.0000, p-value = 0.045500
```

Warto w tym miejscu podkreślić, że opcja uwzględniająca korektę na ciągłość nie została zaimplementowana do funkcji `proportions_ztest` oraz `proportions_chisquare` z pakietu `statsmodels`. Poniżej przykład jak zdefiniować dwustronny test Score oraz Walda z poprawką na ciągłość.

```
from scipy import *
import scipy.stats as stats

def proportions_ztest_correct(m, n, p0, test="Score"):
    p = m/n
    z_cS = (abs(p-p0)-0.5*(1/n))/sqrt((p0*(1-p0))/n)
    p_cS = 2*(1-stats.norm.cdf(z_cS, loc= 0, scale= 1))
    z_cW = (abs(p-p0)-0.5*(1/n))/sqrt((p*(1-p))/n)
    p_cW = 2*(1-stats.norm.cdf(z_cW, loc= 0, scale= 1))
    if test=="Score": return z_cS,p_cS
```

```
    elif test=="Wald": return z_cW,p_cW

sol = proportions_ztest_correct(40, 100, 0.5, test="Wald")
print('Test Walda z korektą: z = %.4f, p-value = %.6f' % (sol[0], sol[1]))
sol = proportions_ztest_correct(40, 100, 0.5, test="Score")
print('Test Score z korektą: z = %.4f, p-value = %.6f' % (sol[0], sol[1]))
```

```
## Test Walda z korektą: z = 1.9392, p-value = 0.052479
## Test Score z korektą: z = 1.9000, p-value = 0.057433
```

Przyjmuje się, że gdy rozmiar próbki jest mały a proporcja z próbki jest mniejsza od $0,2$ lub większa od $0,8$ to stosujemy dokładny test dwumianowy. Dla testu dwustronnego p-wartość obliczamy według wzoru:

$$\text{p-value} = 2 \cdot \min\{\, P(X \geq k, n, p_0),\ P(X \leq k, n, p_0)\,\}$$

```
from statsmodels.stats.proportion import *

print('Dokładny test dwumianowy, p-value = %.6f' % binom_test(40, 100, prop= 0.5) )
```

```
## Dokładny test dwumianowy, p-value = 0.056888
```

Warto zwrócić uwagę na fakt, że p-wartość z dokładnego testu dwumianowego jest bardzo zbliżona do p-wartości asymptotycznego testu Score z poprawką na ciągłość.

W literaturze można spotkać różne sposoby wyznaczania mocy testu dla proporcji np. za pomocą dystrybuanty rozkładu dwumianowego:

$$P(X \geq h_1, n, \bar{p}) + P(X \leq h_2, n, \bar{p})$$

- dla $\bar{p} \leq p_0$

$$h_1 = \left\lceil np_0 - q_{\alpha/2}\sqrt{np_0(1-p_0)} \right\rceil \quad \text{oraz} \quad h_2 = \left\lfloor np_0 + q_{\alpha/2}\sqrt{np_0(1-p_0)} \right\rfloor$$

- dla $\bar{p} \geq p_0$

$$h_1 = \left\lfloor np_0 - q_{\alpha/2}\sqrt{np_0(1-p_0)} \right\rfloor \quad \text{oraz} \quad h_2 = \left\lceil np_0 + q_{\alpha/2}\sqrt{np_0(1-p_0)} \right\rceil$$

lub za pomocą dystrybuanty rozkładu normalnego $N(0,1)$:

$$\text{moc} = P(X \leq h\sqrt{n} - q_{1-\alpha/2}) + P(X \leq -h\sqrt{n} - q_{1-\alpha/2})$$

dla $h = \frac{\bar{p}-p_0}{\sqrt{\bar{p}(1-\bar{p})}}$ będziemy mieli:

$$\text{moc} = P(X \leq Z_{\text{Wald}} - q_{1-\alpha/2}) + P(X \leq -Z_{\text{Wald}} - q_{1-\alpha/2})$$

```
from statsmodels.stats.proportion import *
from statsmodels.stats.power import *
from scipy import *

h = (0.4-0.5)/sqrt(0.4*0.6)
pow = NormalIndPower().solve_power(h, nobs1=100, alpha=0.05, ratio=0)
print('Moc testu Walda = %.4f'%pow)
```

```
## Moc testu Walda = 0.5324
```

Innym sposobem wyznaczania mocy testu jest wykorzystanie symulacji.

```
from statsmodels.stats.proportion import *
from scipy import *

B = 10000
resZ = [ proportions_ztest(random.binomial(n=1,p=0.4, size= 100).sum(), 100, value= 0.5) \
       for i in range(B) ]
pvalZ = [ resZ[i][1] for i in range(B) ]
mocZ = mean([ pvalZ[i] <= 0.05 for i in range(B) ])
print ('Symulacyjna moc testu Walda:', ("%.6f" % mocZ))
```

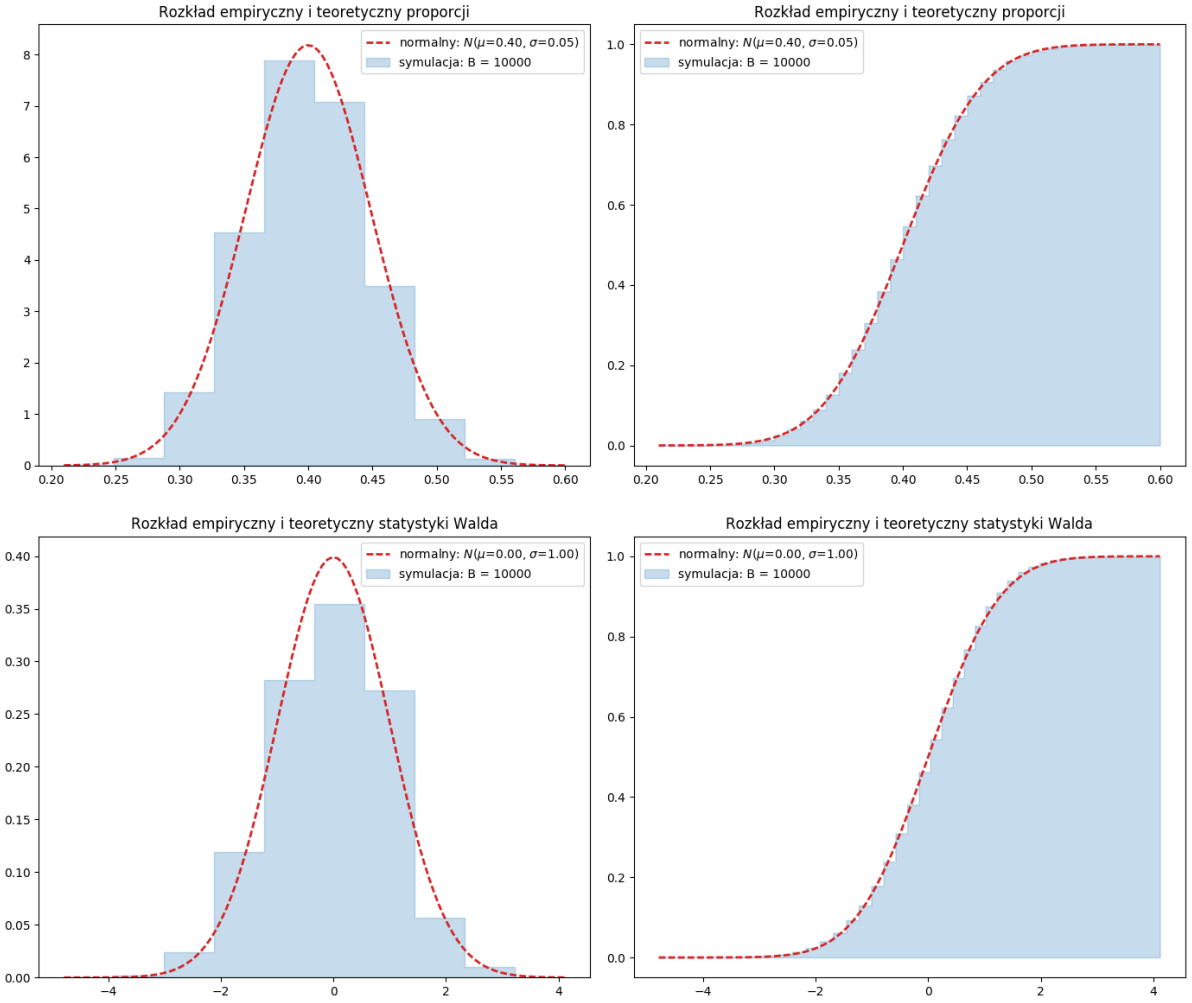
```
## Symulacyjna moc testu Walda: 0.547000
```

W sposób symulacyjny można także wyznaczyć rozkład proporcji oraz statystykę testu.

```
from scipy import *
import scipy.stats as stats
import matplotlib.pyplot as plt
from statsmodels.stats.proportion import *
from statsmodels.distributions.empirical_distribution import ECDF

B = 10000
prop = [ random.binomial(n=100,p=0.4)/100 for i in range(B) ]
test = [ proportions_ztest(random.binomial(n=1,p=0.4, size= 100).sum(), 100, value= 0.5) \
       for i in range(B) ]
test = [ test[i][0] for i in range(B) ]
test = (test-mean(test))/std(test)

fig = plt.figure(figsize=(14,12))
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
ax4 = fig.add_subplot(2,2,4)

X = linspace(min(prop), max(prop), 100)
N, bins, patches = ax1.hist(prop ,normed=1, facecolor='C0',edgecolor='C0',
                   histtype='stepfilled', alpha=0.25, label='symulacja: B = %.0f' % (B))
ax1.plot(X, stats.norm.pdf(X, loc=mean(prop), scale=std(prop)), '--', color='C3',
linewidth=2,label='normalny: $N(\mu$=%.2f, $\sigma$=%.2f)' % (mean(prop),std(prop)))
ax1.set_title("\nRozkład empiryczny i teoretyczny proporcji")
ax1.legend()

ax2.fill_between(ECDF(prop).x, 0, ECDF(prop).y, color='C0', alpha=0.25,
                 label='symulacja: B = %.0f' % (B))
ax2.plot(X, stats.norm.cdf(X, loc= mean(prop), scale= std(prop)), '--', color='C3',
linewidth=2,label='normalny: $N(\mu$=%.2f, $\sigma$=%.2f)' % (mean(prop), std(prop)))
ax2.set_title("\nRozkład empiryczny i teoretyczny proporcji")
ax2.legend()

X = linspace(min(test), max(test), 100)
N, bins, patches = ax3.hist(test ,normed=1, facecolor='C0',edgecolor='C0',
                   histtype='stepfilled', alpha=0.25, label='symulacja: B = %.0f' % (B))
ax3.plot(X, stats.norm.pdf(X, loc=mean(test), scale=std(test)), '--', color='C3',
linewidth=2,label='normalny: $N(\mu$=%.2f, $\sigma$=%.2f)' % (mean(test),std(test)))
ax3.set_title("\nRozkład empiryczny i teoretyczny statystyki Walda")
ax3.legend()
```

```
ax4.fill_between(ECDF(test).x, 0, ECDF(test).y, color='C0', alpha=0.25,
                 label='symulacja: B = %.0f' % (B))
ax4.plot(X, stats.norm.cdf(X, loc= mean(test), scale= std(test)), '--', color='C3',
linewidth=2,label='normalny: $N(\mu$=%.2f, $\sigma$=%.2f)' % (mean(test), std(test)))
ax4.set_title("\nRozkład empiryczny i teoretyczny statystyki Walda")
ax4.legend()

plt.tight_layout()
plt.savefig('prop01.png')

print('95% przedział ufności proporcji:')
print(percentile(prop,(2.5,97.5)))
print('Empiryczny rozkład statystyki testu Walda (bootstrapowa p-watrość dla B = %.0f):' % B)
print('%.4f' % (ECDF(test)(-2.0412)+(1-ECDF(test)(2.0412))) )
```

```
## 95% przedział ufności proporcji:
## [ 0.31  0.5 ]
## Empiryczny rozkład statystyki testu Walda (bootstrapowa p-watrość dla B = 10000):
## 0.0404
```



7 Aproksymacja

7.1 Model Michaelis-Menten

Jedną z bardziej popularnych metod dopasowywania krzywych do zbioru danych jest nieliniowa metoda najmniejszych kwadratów np. z wykorzystaniem algorytmu Levenberga-Marquardta. Ta metoda jest dostępna w pakiecie `scipy.optimize` – funkcje `leastsq`, `least_squares` oraz `curve_fit`. W przypadku gdy chcemy dodać ograniczenia na szukane parametry musimy wybrać jedną z dwóch opcji: `trf` – trust region reflective algorithm (opcja domyślna) lub `dogbox` – dogleg algorithm. Te dwa algorytmy są przewidziane tylko dla funkcji `least_squares` oraz `curve_fit`. W funkcji `leastsq` jest zaimplementowany tylko algorytm Levenberga-Marquardta.

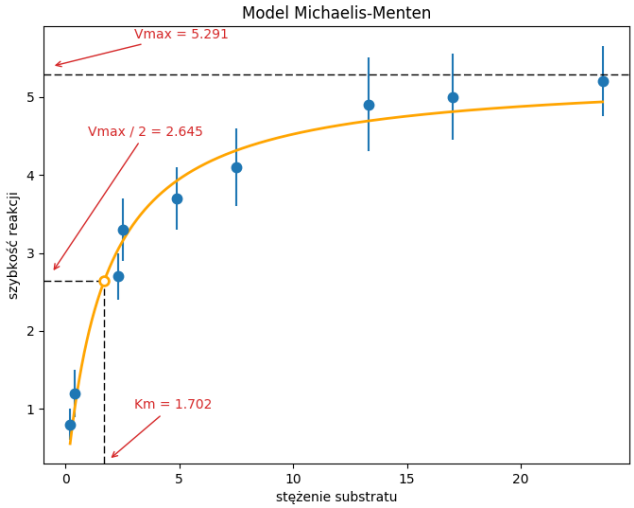
Postać analityczna modelu:

$$V = \frac{S \cdot V_{max}}{S + K_m}$$

```
from scipy import *
import matplotlib.pyplot as plt
from scipy.optimize import *

S = array([0.2, 0.4, 2.3, 2.5, 4.9, 7.5, 13.3, 17.0, 23.6])
V = array([0.8, 1.2, 2.7, 3.3, 3.7, 4.1, 4.9, 5.0, 5.2])
w = array([0.2, 0.3, 0.3, 0.4, 0.4, 0.5, 0.6, 0.55, 0.45])

def micmen(S,Vmax,Km):
    return (S*Vmax)/(S+Km)

sol, pcov = curve_fit(micmen, S, V, sigma=w, absolute_sigma=False, p0=(5,2))
print ("Vmax =", sol[0], "+/-", pcov[0,0]**0.5, \
" ", round((pcov[0,0]**0.5/sol[0])*100,3),"%")
print ("  Km =", sol[1], "+/-", pcov[1,1]**0.5, \
" ", round((pcov[1,1]**0.5/sol[1])*100,3),"%" '\n')
df = len(S) - len(sol)
chi_squared = sum(((micmen(S, *sol) - V) / w)**2)
reduced_chi_squared = chi_squared / df
print ('degrees of freedom: ', df)
print ('chi-square value: ', ("%.3f" % chi_squared))
print ('reduced chi-square value: ', ("%.3f" % reduced_chi_squared))

x = arange(0.2,23.6,0.01)
plt.figure(figsize=(8,6))
plt.title('Model Michaelis-Menten')
plt.errorbar(S, V, yerr = w, fmt = '.', ecolor='#1f77b4', color='#1f77b4', markersize=15)
plt.plot(x, micmen(x,*sol), linewidth=2, color="orange")
plt.xlabel(u'stężenie substratu')
plt.ylabel(u'szybkość reakcji')
plt.axvline(x=sol[1],ymax=0.4,color='k',linestyle='--',linewidth=1)
plt.axhline(y=sol[0]/2,xmax=0.1,color='k',linestyle='--',linewidth=1)
plt.axhline(y=sol[0],color='k',linestyle='--',linewidth=1)
plt.plot(sol[1],sol[0]/2,'o',color='orange',markersize=8)
plt.plot(sol[1],sol[0]/2,'.',color='white',markersize=8)
plt.annotate('Vmax / 2 = %.3f' % (sol[0]/2),xy=(0-0.6,sol[0]/2+0.1),xytext=(1,4.5),\
 arrowprops=dict(arrowstyle='->',color='#d62728'),color='#d62728')
plt.annotate('Km = %.3f' % (sol[1]),xy=(sol[1]+0.2,0.25+0.1),xytext=(3,1),\
```

```
 arrowprops=dict(arrowstyle='->',color='#d62728'),color='#d62728')
plt.annotate('Vmax = %.3f' % (sol[0]),xy=(0-0.6,sol[0]+0.1),xytext=(3,5.75),\
 arrowprops=dict(arrowstyle='->',color='#d62728'),color='#d62728')
plt.savefig("micmen01.png")
```

```
## Vmax = 5.29092809253 +/- 0.286778084255   5.42 %
##   Km = 1.70184477354 +/- 0.312836236328   18.382 %
##
## degrees of freedom:  7
## chi-square value:  4.424
## reduced chi-square value:  0.632
```



7.2 Model odporny

W pakiecie `kapteyn` jest dostępna funkcja `kmpfit` dzięki której szukane parametry modelu możemy wyznaczyć na podstawie wybranego kreterium optymalizacji:

- minimalizacja sumy kwadratów reszt:

$$\sum_{i=1}^{n} \left(\frac{y_i - f}{\sigma_i} \right)^2$$

- minimalizacja sumy wartości bezwzględnych reszt:

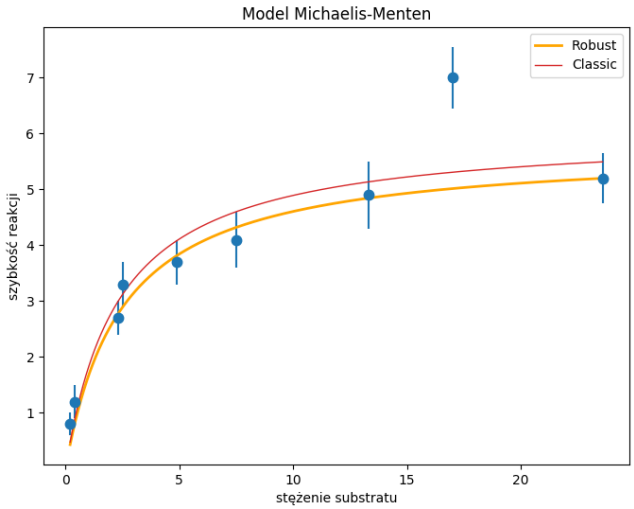
$$\sum_{i=1}^{n} \frac{|y_i - f|}{\sigma_i}$$

```
from scipy import *
import matplotlib.pyplot as plt
from kapteyn import kmpfit

def model(p, data):
   Vmax, Km = p
```

```
    y = (x*Vmax)/(x+Km)
    return y

def fun(p, data):
    x, y = data
    Vmax, Km = p
    model = (x*Vmax)/(x+Km)
    return (y-model)/w

def rob(p, data):
    x, y = data
    Vmax, Km = p
    model = (x*Vmax)/(x+Km)
    return sqrt(abs(y-model)/w)

S = array([0.2, 0.4, 2.3, 2.5, 4.9, 7.5, 13.3, 17.0, 23.6])
V = array([0.8, 1.2, 2.7, 3.3, 3.7, 4.1, 4.9, 7.0, 5.2])
w = array([0.2, 0.3, 0.3, 0.4, 0.4, 0.5, 0.6, 0.55, 0.45])

frob = kmpfit.Fitter(rob, params0=[5,2], data=(S,V))
frob.fit()

ffun = kmpfit.Fitter(fun, params0 = [5,2], data=(S,V))
ffun.fit()

print ("\n==== Robust estimation model ====")
print ("Params: ", frob.params)
print ("Stderr: ", frob.stderr)
print ("Chi^2 min: ", frob.chi2_min)

print ("\n==== Classic estimation model ====")
print ("Params: ", ffun.params)
print ("Stderr: ", ffun.stderr)
print ("Chi^2 min: ", ffun.chi2_min)

x = arange(0.2,23.6,0.01)
plt.figure(figsize=(8,6))
plt.title('Model Michaelis-Menten')
plt.errorbar(S, V, yerr = w, fmt = '.', ecolor='#1f77b4', color='#1f77b4', markersize=15)
plt.plot(x, model(frob.params,S), linewidth=2, color="orange",label="Robust")
plt.plot(x, model(ffun.params,S), linewidth=1, color="C3",label="Classic")
plt.legend()
plt.xlabel(u'stężenie substratu')
plt.ylabel(u'szybkość reakcji')
plt.savefig("robust01.png")
```

```
##
## ==== Robust estimation model ====
## Params:  [5.7443334316605954, 2.470436343321075]
## Stderr:  [ 0.15208103  0.23023822]
## Chi^2 min:  8.879481604048097
##
## ==== Classic estimation model ====
## Params:  [6.039707051014882, 2.3418962277362101]
```

```
## Stderr:  [ 0.64647947  0.77766233]
## Chi^2 min:  16.910065512023017
```

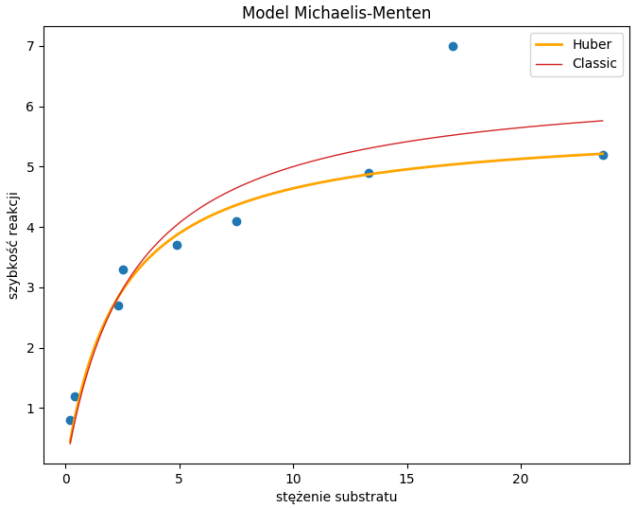


Warto zwrócić uwagę, że za pomocą funkcji `least_squares` (pakiet `scipy.optimize`) również można wyznaczyć parametry modelu na podstawie solidnych metod np. funkcji straty Hubera.

```
from scipy import *
import matplotlib.pyplot as plt
from scipy.optimize import least_squares

def model(p, data):
   Vmax, Km = p
   y = (x*Vmax)/(x+Km)
   return y

def fun(x, t, y):
    return ((t*x[0])/(t+x[1]))-y

x0 = array([ 2.,  1.])

S = array([0.2, 0.4, 2.3, 2.5, 4.9, 7.5, 13.3, 17.0, 23.6])
V = array([0.8, 1.2, 2.7, 3.3, 3.7, 4.1, 4.9, 7.0, 5.2])

res_robust = least_squares(fun, x0, loss='soft_l1', f_scale=0.1, args=(S, V))
res_class = least_squares(fun, x0, args=(S, V))
print('Classic: Vmax = %.4f, Km = %.4f' % (res_class.x[0],res_class.x[1]))
print('Huber:   Vmax = %.4f, Km = %.4f' % (res_robust.x[0],res_robust.x[1]))

x = arange(0.2,23.6,0.01)
plt.figure(figsize=(8,6))
plt.title('Model Michaelis-Menten')
plt.plot(S, V, 'o')
plt.plot(x, model(res_robust.x,S), linewidth=2, color="orange",label="Huber")
plt.plot(x, model(res_class.x,S), linewidth=1, color="C3",label="Classic")
plt.legend()
plt.xlabel(u'stężenie substratu')
```

```
plt.ylabel(u'szybkość reakcji')
plt.savefig("robust02.png")
```

```
## Classic: Vmax = 6.4873, Km = 2.9714
## Huber:   Vmax = 5.7355, Km = 2.3583
```



7.3 Model Voigt

Pakiet `lmfit` zawiera pewne ułatwienia w dopasowywaniu krzywych do danych. Jednym z nich jest szereg wbudowanych funkcji np. model Voigta.

$$V = \frac{\mathrm{Re}\left[\exp(-z^2)\mathrm{erfc}(-iz)\right]\ A}{\sigma\sqrt{2\pi}}$$

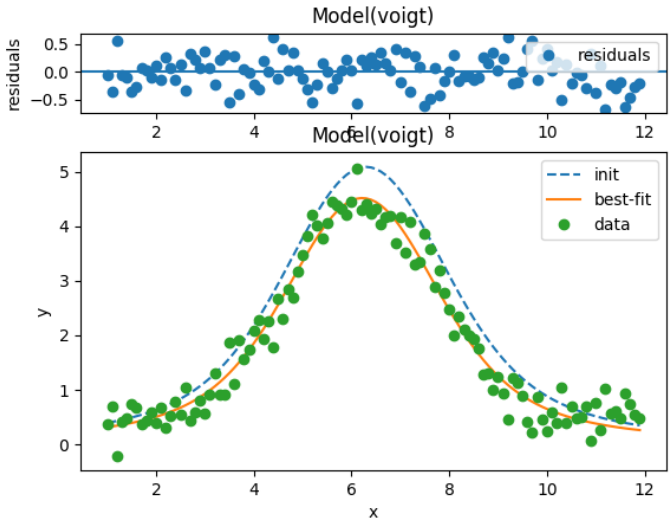
gdzie: $z = \frac{x-\mu+i\gamma}{\sigma\sqrt{2}}$

Dzięki temu nie musimy podawać postaci analitycznej funkcji. Dodatkowo możemy zdecydować o automatycznym doborze parametrów startowych. Warto też wspomnieć o funkcji `minimize` która w porównaniu do funkcji `fit` oferuje większą liczby algorytmów np. Sequential Linear Squares Programming.

```
import matplotlib.pyplot as plt
from scipy import *
from lmfit import *
from lmfit.models import *

x = arange(1,12,0.1)
random.seed(2307)
y = 0.4+4*exp(-(x-6.2)**2/(2*1.5**2))+ random.normal(0,0.25,size=len(x))+0.1
mod = VoigtModel()
pars = mod.guess(y, x=x)
out = mod.fit(y, pars, x=x)
print(out.fit_report())

plt.figure(figsize=(8,6))
```

```
out.plot()
plt.savefig('voigt01.png')
```

```
## [[Model]]
##     Model(voigt)
## [[Fit Statistics]]
##     # function evals   = 19
##     # data points      = 110
##     # variables        = 3
##     chi-square         = 9.120
##     reduced chi-square = 0.085
##     Akaike info crit   = -267.902
##     Bayesian info crit = -259.801
## [[Variables]]
##     amplitude:   23.0854679 +/- 0.367235 (1.59%) (init= 27.71622)
##     center:      6.22026928 +/- 0.029921 (0.48%) (init= 6.297143)
##     sigma:       1.06724765 +/- 0.022152 (2.08%) (init= 1.1375)
##     gamma:       1.06724765 +/- 0.022152 (2.08%)  == 'sigma'
##     fwhm:        3.84348964 +/- 0.079777 (2.08%)  == '3.6013100*sigma'
##     height:      8.62945883 +/- 0.135545 (1.57%)  == '0.3989423*amplitude/max(1.e-15, sigma)'
## [[Correlations]] (unreported correlations are <  0.100)
##     C(amplitude, sigma)          =  0.662
```



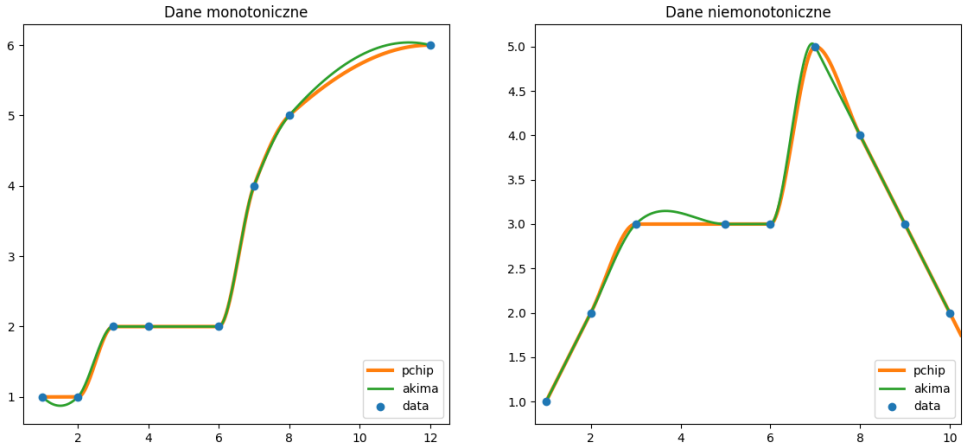
Więcej informacji na temat tego pakietu można znaleźć po adresem:

- https://lmfit.github.io/lmfit-py/

8 Interpolacja a krzywa Beziera

W pakiecie `scipy.interpolate` jest zaimplementowanych kilka ciekawych funkcji do interpolacji danych. Poniżej kilka przykladowych metod interpolacji danych niemonotonicznych oraz monotonicznych czyli takich w których punkty zachowują określony porządek.

```
import matplotlib.pyplot as plt
from scipy import *
from scipy.interpolate import *

xp = [1,2,3,4,6,7,8,12]
yp = [1,1,2,2,2,4,5,6]
Xp = [1,2,3,5,6,7,8,9,10]
Yp = [1,2,3,3,3,5,4,3,2]

xv = arange(1,12,0.01)

fig = plt.figure(figsize=(14,6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
ax1.plot(xv,PchipInterpolator(xp, yp)(xv),\
         linewidth=3,color="C1",label="pchip")
ax1.plot(xv,Akima1DInterpolator(xp, yp)(xv),\
         linewidth=2,color="C2",label="akima")
ax1.plot(xp,yp,'o',color="C0",label="data")
ax1.set_title("Dane monotoniczne")
ax1.legend(loc='lower right')
ax2.plot(xv,PchipInterpolator(Xp, Yp)(xv),\
         linewidth=3,color="C1",label="pchip")
ax2.plot(xv,Akima1DInterpolator(Xp, Yp)(xv),\
         linewidth=2,color="C2",label="akima")
ax2.plot(Xp,Yp,'o',color="C0",label="data")
ax2.set_xlim(0.75,10.25)
ax2.set_ylim(0.75,5.25)
ax2.set_title("Dane niemonotoniczne")
ax2.legend(loc='lower right')
fig.savefig('inter01.png')
```



Warto w tym miejscu przedstawić również krzywą Beziera. Ta metoda interpoluje tylko punkt początkowy i końcowy a pozostałe (punkty pośrednie) aproksymuje. Poniżej przykład dla danych nieposortowanych oraz posortowanych.

```
from scipy import *
from pandas import *
import matplotlib.pyplot as plt
from scipy.stats import *

t = arange(0,1,0.01)
d = DataFrame({'x' :[0,0,2,8,4], 'y' :[0,4,5,4,0]})
px = [sum(binom.pmf(arange(0,5,1),4,i)*list(d['x']))
     for i in t]
py = [sum(binom.pmf(arange(0,5,1),4,i)*list(d['y']))
     for i in t]

D = d.sort_values(by=['x'], ascending=[True])
Px = [sum(binom.pmf(arange(0,5,1),4,i)*list(D['x']))
     for i in t]
Py = [sum(binom.pmf(arange(0,5,1),4,i)*list(D['y']))
     for i in t]

fig = plt.figure(figsize=(14,5))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
ax1.plot(d['x'],d['y'],'--',label="")
ax1.plot(d['x'],d['y'],'o',color="C0",label="data")
ax1.plot(px,py,linewidth=2,color="orange",label="Bezier curve")
ax1.set_title("Dane nieposortowane")
ax1.legend(loc='lower right')
ax2.plot(D['x'],D['y'],'--',label="")
ax2.plot(D['x'],D['y'],'o',color='C0',label="data")
ax2.plot(Px,Py,linewidth=2,color="orange",label="Bezier curve")
ax2.set_xlim(-0.25,8.25)
ax2.set_ylim(-0.25,5.25)
ax2.set_title("Dane posortowane")
ax2.legend(loc='lower right')
fig.savefig('Bezier01.png')
```