Faculty of Geodesy
and Cartography
WARSAW UNIVERSITY OF TECHNOLOGY

Informatyka Geodezyjna - wykłady/ćwiczenia, rok akad. 2021-2022

Wyk. 08: Python – prezentacja graficzna danych numerycznych (Matplotlib, Cartopy)

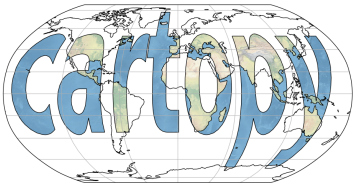
Kinga Wężka
kinga.wezka@pw.edu.pl
Katedra Geodezji i Astronomii Geodezyjnej

Warsaw University
of Technology

Plan

1. Matplotlib – wprowadzenie
2. Matplotlib – interfejsy obsługi – Object-Oriented API vs PyPlot
3. Matplotlib – architektura warstw: Backend, Artist, Scripting
   - Matplotlib warstwa zaplecza – backend layer
   - Matplotlib wartwa wygladu – artist Layer – elementy rysunku
   - Matplotlib wartwa skryptowa – scripting layer - PyPlot API
   - Podsumowanie – PyPlot API vs Object-Oriented API
   - Podsumowanie – architektura warstw
4. Matplotlib – hierarchia obiektów
5. Matplotlib – modułu i funkcje
6. Mtplotlib – arkusze stylów i rcParams
7. Matplotlib Toolkits – narzędzia rozszerzające funkcjonalność
8. Kartografia w matplotlib
   - Cartopy – tworzenie map – analizy geoprzestrzenne
   - GeoPandas – analiza danych geoprzestrzennych
9. Inne biblioteki do edycji grafiki – OpenCV i PIL – cyfrowe przetwarzanie obrazów

- Matplotlib - graficzna prezentacja wyników (wykresy);
- Cartophy (obecnie Basemap) - tworzenie map, przetwarzanie danych geoprzestrzennych;



- Zasadniczo **matplotlib** jest zorientowany obiektowo.
- Moduł **matplotlib** tworzy zaawansowane, wysokiej jakości wykresy.
- Jest dostarczany z dodatkowymi narzędziami (add-on toolkits): `mplot3d`, `axes_grid1`, `axisartist`.
- Wykorzystywany jako podstawa działania wielu innych bibliotek graficznych.
- Polecana literatura (McGreggor, 2015, Poladi, 2018)

Użycie biblioteki wymaga jej instalacji i importu:

In[1]:

```
1 import matplotlib.pyplot as plt
```

Oficjalna strona internetowa Matplotlib

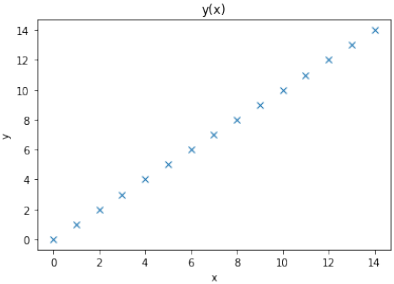
https://matplotlib.org/

- **Object-Oriented API** – Każdy wykres jest podzielony na pewne obiekty, a hierarchia obiektów jest jasna. Wykres jest podzielony na dwa obiekty: `Figure` i `Axes`. Instancja `axes.Axes` jest wykorzystywana w celu renderowania wizualizacji na instancji `figure.Figure`.
- **Pyplot API** – to zbiór funkcji w stylu poleceń, które sprawiają, że Matplotlib działa jak MATLAB. Jest przeznaczony głównie do prostych wykresów w konsolach.
- **Pylab API** łączy funkcję PyPlot (do kreślenia) z funkcjami Numpy w pojedynczą **przestrzeń nazw** (ang. *namespace*). Istnieje z przyczyn historycznych, ale zdecydowanie nie zaleca się jego używania. Zanieczyszcza przestrzenie nazw funkcjami, które zakrywają (maskują) wbudowane funkcje Pythona co powoduje trudności w śledzeniu błędów (Python Zen: jawne jest lepsze niż niejawne).

API (ang. *application programming interface*) – interfejs programowania aplikacji – ściśle określony zestaw reguł i ich opisów, w jaki programy komputerowe komunikują się między sobą. Zadaniem API jest dostarczenie odpowiednich specyfikacji podprogramów, struktur danych, klas obiektów i wymaganych protokołów komunikacyjnych.

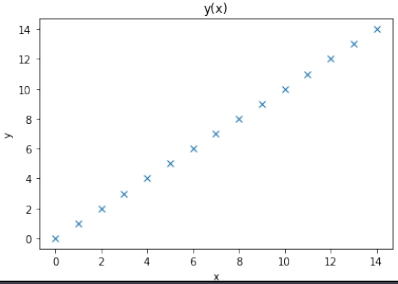
Pylab API

```
1 from matplotlib.pylab import *
2 plot(range(0,15), 'x')
3 title('y(x)')
4 xlabel('x')
5 ylabel('y')
6 show()
```



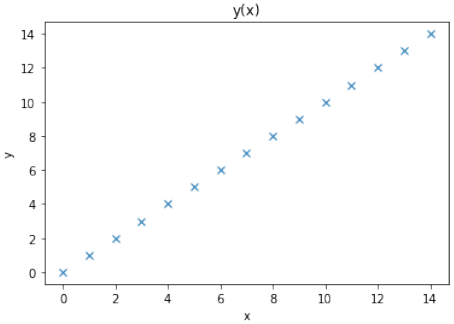
Pyplot API

```
1 import matplotlib.pyplot as plt
2 plt.plot(range(0,15), 'x')
3 plt.title('y(x)')
4 plt.xlabel('x')
5 plt.ylabel('y')
6 plt.show()
```



OPP API

```
1 import matplotlib.pyplot as plt
2 fig = plt.figure(figsize = (5, 3))
3 ax  = fig.add_subplot(111)
4 ax.plot(range(0,15), 'x')
5 ax.set_title('y(x)')
6 ax.set_xlabel('x')
7 ax.set_ylabel('y')
8 plt.show()
```



Zalecane jest używanie interfejsu zorientowanego obiektowo.

Interfejs OPP daje większą kontrolę nad wykresem.

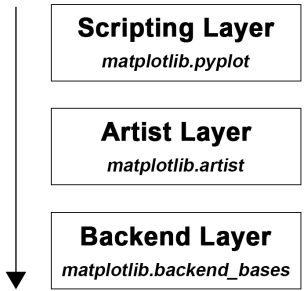
https://matplotlib.org/3.3.3/api/index.html

Większość galerii przykładów wykorzystuje podejście obiektowe:

https://matplotlib.org/gallery/index.html#examples-index

Kompleksowa architektura matplotlib odpowiada za operacje tworzenia, renderowania i aktualizowania obiektu **Figure** (arkusz wykresów).

- **Scripting Layer** – najlżejszy interfejs skryptowy spośród trzech warstw, zaprojektowany tak, aby Matplotlib działał jak skrypt MATLAB. Jest używane przez developera do tworzenia wykresów.
- **Artist Layer** – tworzenie i edycja wyglądu wykresów, umożliwia pełną kontrolę i edycję wszystkich elementów. To warstwa logiczna która jest odpowiedzialna za wygląd (aspekt wizualny) każdego elementu widocznego na rysunku. Np. **Figure** składa się z wielu obiektów indywidualnie modyfikowanych w taki sposób, aby miała przewidywalny wpływ na wszystkie elementy **Figure**.
- **Backend Layer** – obsługuje najważniejsze prace poprzez komunikację z zestawami narzędzi do rysowania na komputerze. To najbardziej złożona warstwa, pełni rolę implementacji elementów graficznych na niskim poziomie (np. renderowanie).



- Arkusze wykresów matplotlib mogą być prezentowane w różnych formatach wyjściowych np. szybkie rysowanie w notatnikach Jupyter; osadzanie wykresów w graficznych interfejsach użytkownika – GUI – (pyqt, wxpython lub pygtk); używanie w skryptach wsadowych do generowania obrazów postscriptowych; lub uruchamianie na serwerach aplikacji internetowych, aby dynamicznie wyświetlać wykresy.
- Aby obsługiwać wszystkie przypadki, matplotlib można kierować się na różne wyjścia, a każda z tych możliwości jest nazywana zapleczem **backend**;
- W tym przypadku **frontend** to kod widoczny dla użytkownika, tj. kod kreślący, podczas gdy **backend** wykonuje całą ciężką pracę za kulisami, aby stworzyć wykres.

Matplotlib Backend

Matplotlib Backend (pl. zaplecze) jest warstwą abstrakcyjną zajmującą się procesem faktycznego renderowania **Figure** (arkusz wykresów). Takie wykresy pojawiają się w aplikacjach komputerowych osadzonych w widżetach, wybranych interfejsach graficznych GUI lub na stronach internetowych; inne wykresy to obrazy w publikacjach. Istnieją dwa rodzaje backendów: interaktywne i nieinteraktywne.

- Warstwa **backend** ma trzy wbudowane główne abstrakcyjne klasy interfejsu (zwykły użytkownik prawie ma do czynienia z tą warstwą):
  - **FigureCanvas** – obszar roboczy na którym renderowany jest wykres . To obiekt, który zawiera koncepcję obszaru na którym można narysować (np. „Papier"). `matplotlib.backend_bases.FigureCanvasBase`
  - **Renderer** – abstrakcyjna klasa bazowa do obsługi operacji rysowania/renderowania (np. „Pędzel"). Odpowiedzialna za rysowanie na FigureCanvas. `matplotlib.backend_bases.RendererBase`
  - **Event** – to obiekt, który obsługuje dane wejściowe użytkownika, takie jak zdarzenia klawiatury i myszy. `matplotlib.backend_bases.Event`
- Rodzaje **backendu**:
  - **backendy w formie nieedytowalnej** (ang. *hardcopy backends*) zwane również jako **nie-interaktywne backendy** (ang. *non-interactive backends*) do tworzenia plików obrazów: AGG (.png), PDF (.pdf), SVG (.svg), PS (.ps, .eps), PGF (.pgf, .pdf), Cairo (.png, .pdf).
    - Mogą być podzielone na grafikę: wektorową i niewektorową (raster)
  - **backendy z interfejsem użytkownika** (ang. *user interface backends*) zwane również jako **interaktywne backendy** (ang. *interactive backends*) – należą do nich: Qt5Agg, GTK3Agg, ipympl (jupyter), wxAgg, TkAgg, macosx, WebAgg – Agg – Anti-Grain Geometry – silnik renderujący grafikę 2D.

- **Zdefiniowanie w pliku konfiguracyjnym `matplotlibrc` (Qt5agg, gtk3agg etc.):**

```
backend : qt5agg # uzycie Qt z renderowaniem anti-grain (agg)
```

Lokalizacja pliku `matplotlibrc`:

```
>>> import matplotlib
>>> matplotlib.matplotlib_fname()
'/home/foo/.config/matplotlib/matplotlibrc'
```

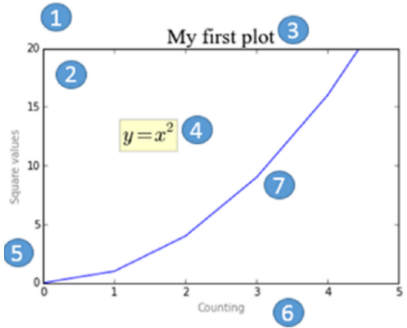
- **Zdefiniowanie za pomocą funkcji `use()`, jeśli kod zależy od określonego backendu:**

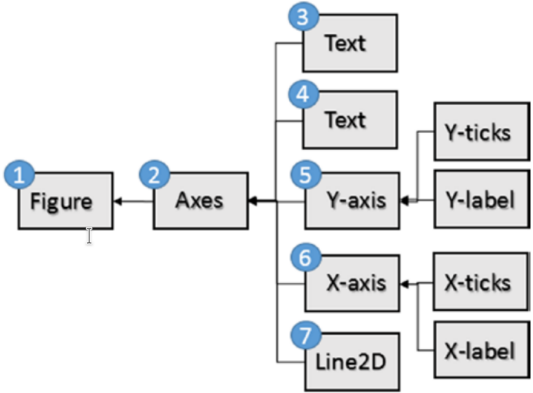
```
import matplotlib
matplotlib.use('Qt5agg')
```

Funkcję `use()`, należy użyć przed `import matplotlib.pyplot`, wywołanie `use()` po nie przyniesie żadnego efektu. Należy unikać jawnego wywoływania metody `use()`.

Jeśli ustawimy kilka backendów równocześnie, nastąpi konflikt. Zastosowana zostanie ostatnia wymieniona metoda, np. wywołanie `use()` zastąpi ustawienie w `matplotlibrc`

- Warstwa Artist składa się z jednego głównego obiektu, **Artist**, który używa modułu renderującego do rysowania na **FigureCanvas**.
- Daje większe możliwości od **warstwy skryptowej** i jest wydajniejsza w zastosowaniu do zaawansowanych wykresów. Warstwa ta nazywana jest **plotowaniem obiektowym**.
- To co widzimy na obiekcie **Figure**, jest obiektem klasy **Artist** (tytuł, linie, etykiety).





Scripting Layer - to API dla użytkownika aby uprościć typowe zadania, a matplotlib robi to również w interfejsie `matplotlib.pyplot`

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.random.randn(10000)
5
6 plt.hist(x, 100) # metoda Axes do generowania histogramu
7 plt.title(r'Normal distribution with $\mu=0, \sigma=1$')
8 plt.savefig('matplotlib_histogram.png')
9 plt.show()
```

- Wywołanie `plt.plot()` to po prostu wygodny sposób na uzyskanie aktualnych osi bieżącej Figury, a następnie wywołanie jej metody plot() . To właśnie oznacza twierdzenie, że interfejs *stateful* zawsze „domyślnie śledzi" fabułę, do której chce się odwoływać.
- **pyplot API** udostępnia wiele funkcji, które tak naprawdę są tylko opakowaniami wokół **Object-Oriented API**. Na przykład, dla `plt.title()`, istnieją odpowiadające metody podejścia zorientowanego obiektowo (OO) tj. ustawiające (ang. *setter*) i pobierające (ang. *getter*) `ax.set_title()`, `ax.get_title()`
- Wywołanie `plt.title()` zostaje przetłumaczone na jedną linię: `gca().set_title(s, *args, **kwargs)`. Oto co to robi:
  - `gca()` pobiera bieżącą oś i zwraca ją.
  - `set_title()` jest metodą ustawiającą, która ustawia tytuł dla tego obiektu `Axes`.

więcej: https://realpython.com/python-matplotlib-guide/

Scripting Layer
(pyplot)

Artist Layer

Primitive Layer
(Line, Rectangle, Circle, Text)

Composite Layer
(Axis, Ticks, Axes & Figures)

Backend Layer

Figure Canvas Layer
(Encompasses area in which figures are drawn)

Renderer Layer
(Knows how to draw on figure canvas)

Event Layer
(Handle user inputs such as keyboard, strokes and mouse clicks)

- `Figure` to cały obszar kreślenia. **Figure** kontroluje wszystkie podrzędne **Axes**, oraz kilka „specjalnych” narzędzi artist (tytuły, legendy itp.) oraz kanwę (canvas).
- **Figure** może zawierać dowolną liczbę **Axes** – minimum jedną.
- Utworzenie obiektu **Figure**:

```
1 import matplotlib.pyplot as plt
2 # pusta figure bez Axes
3 fig = plt.figure()
4 # figure z jedną Axes
5 fig, ax = plt.subplots()
6 # figure z 4 Axes w gridzie 2x2
7 fig, axs = plt.subplots(2, 2)
8 plt.show()
```

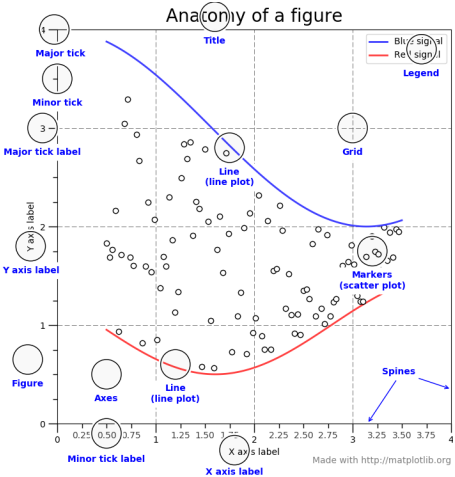


Figure: Na podstawie: https://matplotlib.org/3.3.3/tutorials/introductory/usage.html

- `Axes` - to jest nasz wykres – obszar obrazu z danymi. Dana figura może zawierać wiele wykresów (**Axes**), ale dany obiekt **Axes** może znajdować się tylko na jednej **Figure**.
- **Axes** zawiera dwa (lub trzy w przypadku 3D) obiekty **Axis** (uwaga na różnicę między **Axes** i **Axis**), które zajmują się limitami danych (limity danych można również kontrolować za pomocą metod `axes.Axes.set_xlim()` i `axes.Axes.set_ylim()`).
- Każda **Axes** ma tytuł (`set_title()`), etykiety x i y (`set_xlabel()`, `set_ylabel()`).
- Klasa **Axes** i jej funkcje są podstawowym punktem wyjścia do pracy z interfejsem obiektowym.

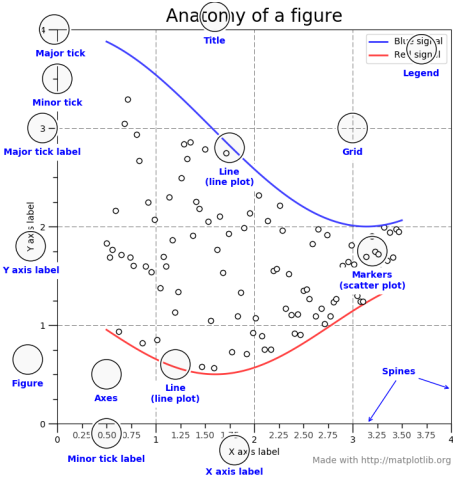


Figure: Na podstawie: https://matplotlib.org/3.3.3/tutorials/introductory/usage.html

- `Axis` – są to obiekty podobne do linii liczbowych.
- Zajmują się ustalaniem granic wykresu i generowaniem znaczników na osi (**ticks**) i opisów znaczników (**ticklabels** – opis tekstowy).
- Lokalizacja znaczników (**ticks**) jest określana przez obiekt Locator,
- Opisy znaczników (**tickslabels**) są formatowane przez narzędzie Formatter
- Połączenie funnkcji **Locator** i **Formatter** zapewnia bardzo dokładną kontrolę nad znaczników osi oraz ich opisami.

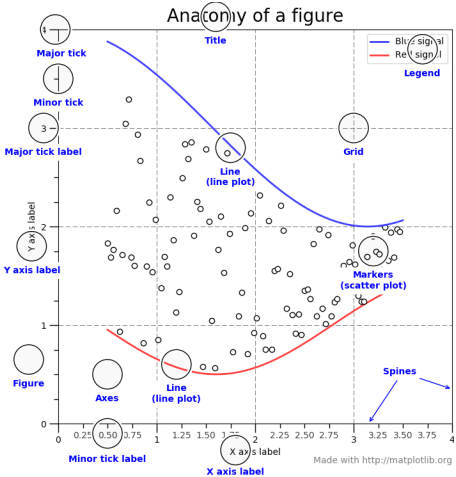


Figure: Na podstawie:
https://matplotlib.org/3.3.3/tutorials/introductory/usage.html

- `Artist` – to zasadniczo wszystko, co widać na tej **Figure** (`Figure`, `Axes`, `Axis`).
- `Artist` obejmuje obiekty tekstowe (`Text`), obiekty linii (`Line2D`), (`Pathch`) etc.
- Kiedy figure jest renderowana, wszytskie obiekty `Artist` są rysowane na canvie.
- Większość obiektów `Artist` jest przywiązana do `Axes`; taki `Artist` nie może być współużytkowany przez wiele `Axes` ani przenoszony między nimi.

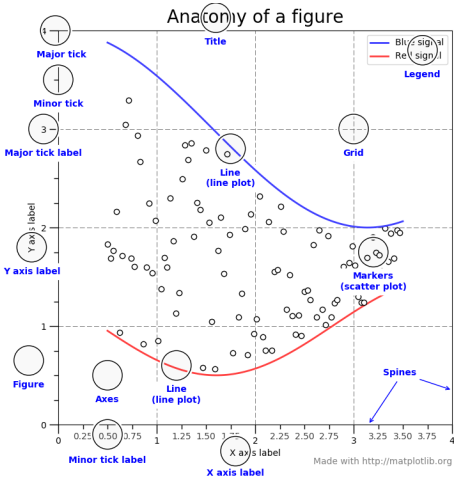


Figure: Na podstawie:
https://matplotlib.org/3.3.3/tutorials/introductory/usage.html

- `plt.plot([1, 2, 3])` taki zapis ukrywa fakt, że wykres jest tak naprawdę hierarchią zagnieżdżonych obiektów Pythona. **Hierarchia** oznacza tutaj, że pod każdym polem znajduje się rozgałęziona struktura obiektów matplotlib.
- Obiekt `Figure` to najbardziej zewnętrzny kontener dla grafiki Matplotlib, który może zawierać wiele obiektów `Axes`. Jednym ze źródeł zamieszania jest nazwa: **Axes** faktycznie oznacza indywidualny wykres (nie jakby się można spodziewać liczbę mnogą „osi").
- Obiekt `Figure` można interpretować jako pudełkowy pojemnik, w którym znajduje się jedna lub więcej `Axes` (rzeczywiste wykresy). Poniżej `Axes` w hierarchii znajdują się mniejsze obiekty, takie jak `tick marks, lines, legends, text boxes`. Prawie każdy element wykresu jest edytowalnym obiektem Pythona, aż do `tick, label`

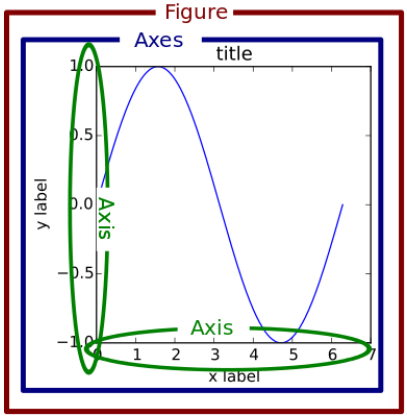
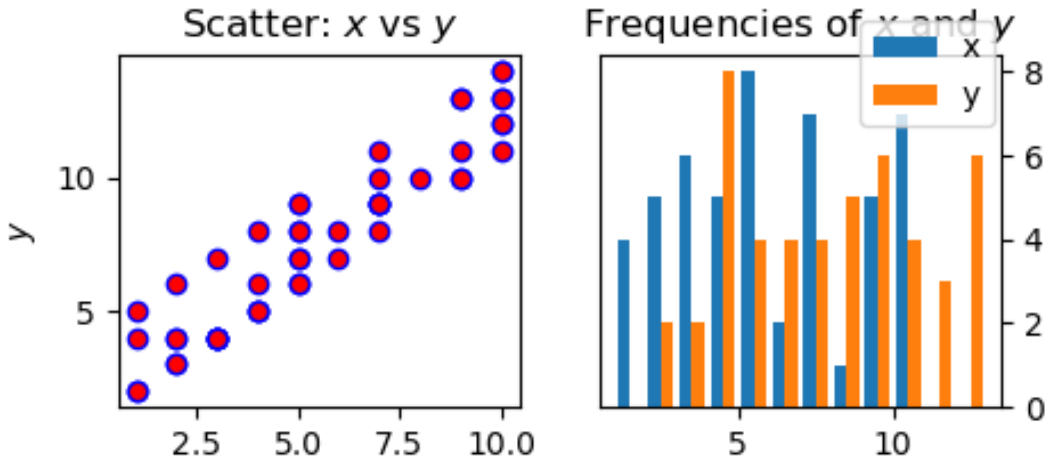


Figure: Na podstawie: `realpython.com/python-matplotlib-guide`

```
1  import matplotlib.pyplot as plt
2  x = np.random.randint(low=1, high=11, size=50)
3  y = x + np.random.randint(1, 5, size=x.size)
4  data = np.column_stack((x, y))
5  # utworzenie obiektu Figure oraz dwóch obiektów Axes
6  fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(4, 5))
7  # 1szy wykres - Axes 1
8  ax1.scatter(x=x, y=y, marker='o', c='r', edgecolor='b')
9  ax1.set_title('Scatter: $x$ vs $y$')
10 ax1.set_ylabel('$y$') # Axis
11 ax1.set_xlabel('$x$') # Axis
12 # 2gi wykres - Axes 2
13 ax2.hist(data, bins=np.arange(data.min(), data.max()), label=('x', 'y'))
14 ax2.legend(loc=(0.65, 0.8))
15 ax2.set_title('Frequencies of $x$ and $y$')
16 ax2.yaxis.tick_right() # Axis
```


Scatter: x vs y
y
10
5
2.5
5.0
7.5
10.0
Frequencies of x and y
x
y
8
6
4
2
0
5
10

- Lista podstawowych modułów matplotlib:
  https://matplotlib.org/3.1.1/py-modindex.html
- Do podstawowych modułów należą np.

```
matplotlib.artist
matplotlib.axes
matplotlib.figure
matplotlib.pyplot
matplotlib.rcsetup
```

- Wybrane funkcje dostępne dla `matplotlib.pyplot` -
  https://matplotlib.org/api/pyplot_summary.html

```
bar     # Utworzenie wykresu słupkowego.
plot    # wykres y vs x (wartości połączone linią)
ioff    # wyłączenie trybu interaktywnego
ion     # wyłączenie trybu interaktywnego
scatter # y vs x ze zmianą wyglądu markera punktu (wykres punktów)
```

- Arkusze stylów i rcParams – pozwalają na globalną konfigurację wyglądu wykresów.
- Pakiet stylów pozwala obsługę łatwych do przełączania „stylów" z tymi samymi parametrami, co plik **matplotlibrc** (który jest odczytywany podczas uruchamiania w celu globalnego skonfigurowania Matplotlib).
- Istnieje wiele predefiniowanych stylów udostępnianych przez Matplotlib. Na przykład, istnieje predefiniowany styl o nazwie „ggplot", który emuluje estetykę ggplot (popularnego pakietu kreślenia dla R). Aby użyć tego stylu, należy dodać:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 plt.style.use('ggplot')
```

- Lista dostepnych stylów:

```
1 print(plt.style.available)
```

- Można tworzyć własne arkusze stylów i używać ich, wywołując `style.use()` ze ścieżką lub adresem URL do arkusza stylów.
- Na przykład utworzyć `../dane/wykresy.mplstyle` z następującymi elementami:

```
axes.titlesize : 24
axes.labelsize : 20
lines.linewidth : 3
lines.color : 'r'
lines.markersize : 10
xtick.labelsize : 16
ytick.labelsize : 16
```

- Następnie wywołac go w kodzie Pythona:

```
import matplotlib.pyplot as plt
plt.style.use('./dane/wykresy.mplstyle')
```

- Można również definiować style dynamicznie za pomocą `rcParams`

```
1 import matplotlib.pyplot as plt
2 import matplotlib as mpl
```

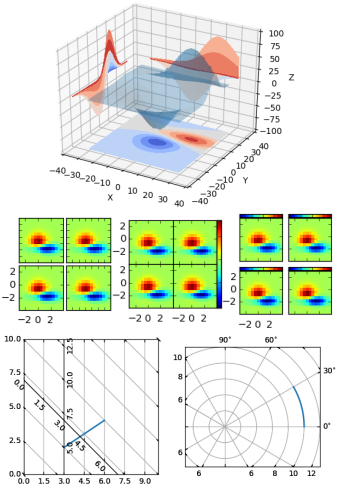
- Wtedy odpowiednie parametry definujemy za pomocą `rcParams`, na początku kodu generującego dany wykres:

```
1 import matplotlib.pyplot as plt
2 import matplotlib as mpl
3 mpl.rcParams['lines.linewidth'] = 2
4 mpl.rcParams['lines.linestyle'] = '--'
```

- Alternatywnie grupując parametry za pomoca `rc`,

```
1 mpl.rc('lines', linewidth=2, color='r')
```

Do podstawowe narzędzi rozszerzających funkcjonalność matplotlib należą:



- `mplot3d` - zawiera klasę `Axes3D` do tworzenia wykresów 3D.

```
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.mplot3d import Axes3D
3 fig = plt.figure()
4 ax = fig.add_subplot(111, projection='3d')
```

- `axes_grid1` - zbiór klas `ImageGrid`, `AxesDivider`, `ParasiteAxes` pomocniczych ułatwiających wyświetlanie obrazów.
- `axisartist` - zawiera niestandardową klasę `AxisArtist`, która może definiować krzywoliniowe osie współrzędnych.

https://matplotlib.org/tutorials/index.html#toolkits

Cartopy (następca Basemap)

- Biblioteka przeznaczona do przetwarzania danych geoprzestrzennych w celu tworzenia map i analiz danych geoprzestrzennych.
- **Cartopy** działa w oparciu o PROJ.4 C i GEOS. Wykorzystanie **PROJ.4 C** pozwala na transformację współrzędnych do jednego z 25 dostępnych odwzorowań (ang. projection)
- Cartpy ma funkcjonalność podobną do Generic Mapping Tools (GMT), GrADS, IDL,
- Kluczowymi cechami **Cartopy** są jej zorientowane obiektowo definicje odwzorowań oraz zdolność do transformacji punktów, linii, wektorów i obrazów między odwzorowaniami.
- Pozwala na prace z danymi **Open Street Map (OSM) i Web Map Service (WMS)**
- Pomoc i galeria przykładów:https://scitools.org.uk/cartopy/docs/latest/
- Użycie biblioteki wymaga jej instalacji (w konsoli conda) i importu do programu: https://anaconda.org/anaconda/cartopy

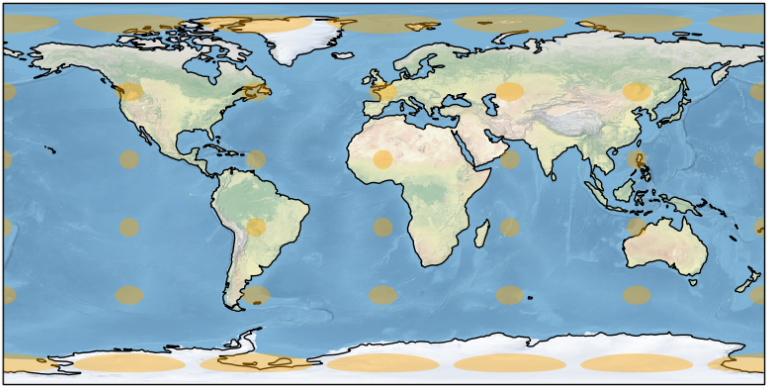
In[2]:

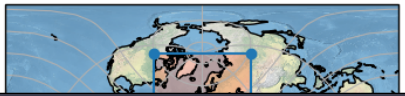
```
conda install -c anaconda cartopy
```

In[3]:

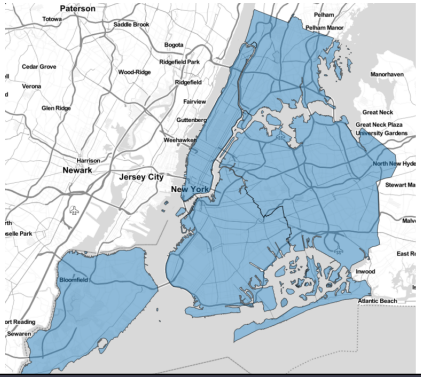
```
import matplotlib.pyplot as plt
import cartopy
ax = plt.axes(projection=cartopy.crs.Mercator())
ax.add_feature(cartopy.feature.LAND)
ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.COASTLINE)
ax.add_feature(cartopy.feature.BORDERS, linestyle='-', alpha=.5)
ax.add_feature(cartopy.feature.LAKES, alpha=0.95)
ax.add_feature(cartopy.feature.RIVERS)
ax.set_extent([-150, 60, -25, 60])
```

https://scitools.org.uk/cartopy/docs/v0.16/gallery





- **GeoPandas** rozszerza typy danych używane przez **Pandas**, aby umożliwić operacje przestrzenne na typach geometrycznych. Operacje geometryczne są wykonywane przez **Shapely**. **GeoPandas** zależą również od **Fiona** w zakresie dostępu do plików oraz od kart i **matplotlib** do wykresów. `http://geopandas.org/`



- **PIL** (ang. *Python Imaging Library*) – podstawowa biblioteka do czytania obrazów.
- **Pillow** to przyjazne rozgałęzienie PIL (ang. *fork PIL*) do przetwarzania obrazów. Dodaje obsługę grafiki np. otwieranie, modyfikowanie, zapisywanie plików graficznych. Do podstawowych funkcji PIL należą: obrót zdjęcia, proste rysowanie, linie, koła, łuki – za pomocą współrzędnych, nakładanie filtrów – rozmycie, wyostrzania itp., obsługa wielu fontów, wklejanie, kopiowanie plików graficznych
- `matplotlib.image.mpimg` https://matplotlib.org/tutorials/introductory/images.html#sphx-glr-tutorials-introductory-images-py
- Przetwarzanie obrazu za pomocą SciPy i NumPy pozwala na odczytanie i zapisanie danych do obrazu oraz wyświetlaniem obrazu. Biblioteki te udostępniają wiele opcji manipulacji obrazem takich jak zaawansowane metody filtrowania https://data-flair.training/blogs/image-processing-with-scipy-and-numpy/

```
1    import matplotlib.pyplot as plt
2    import matplotlib.image.mpimg
```

Fork – sytuacja, w której rozwój projektu nie prowadzi już jedną drogą, ale rozwidla się na dwie lub więcej gałęzi.

- **OpenCV-Python** – rozbudowana biblioteka oparta na OpenCy (napisane w c/c++) służąca do cyfrowego przetwarzanie obrazów, kalibracji kamer, fotogrametrii cyfrowej, uczenia maszynowego, wykrywania obiektów.
- Instalacja dla ekosystemu Anaconda: https://anaconda.org/conda-forge/opencv
- Instrukcje dla OpenCV-Python: opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html https://circuitdigest.com/tutorial/getting-started-with-opencv-image-processing

- **Seaborn** – biblioteka oparta na **matplotlib** (`https://seaborn.pydata.org/`)
- **Bokeh** – Interaktywna wizualizacja dla serwisów webowych (również mapy)
- **Plotly** – Interaktywna wizualizacja dla serwisów webowych (commercial - free for educational purposes)
- **Shapely** – Manipulacja i analiza płaskich obiektów geometrycznych (interfejs do GEOS).
- **Fiona** – Odczytywanie i zapisywanie danych przestrzennych, alternatywa dla geopanda (interfejs do GDAL).
- **Pyproj** – Wykonuje transformacje kartograficzne i obliczenia geodezyjne (PROJ.4).
- **Pysal** – Biblioteka funkcji analizy przestrzennej napisanych w języku Python.
- **Geopy** – Biblioteka geokodowania: współrzędne do adresu <-> adres do współrzędnych.
- **GeoViews** – Interaktywne mapy.
- **Networkx** – Analiza tras nawigacyjnych w Pythonie (np. Algorytmy Dijkstra i A *, grafy),
- **Scipy.spatial** – Algorytmy przestrzenne i struktury danych.
- **Rtree** – Indeksowanie przestrzenne do szybkiego wyszukiwania przestrzennego.
- **Rasterio** – Analizy geoprzestrzenne na podstawie rastrów.
- **RSGISLib** – Biblioteka oprogramowania do teledetekcji i GIS dla Pythona.

Pomoc

- Przykłady – Jupyter Notebook:
  OneDrive –
  `tutoriale_pomoce/lib_matplotlib/help_matplotlib_przyk\T1\lady.ipynb`
- Zestawienie najczęściej uzywanych metod:
  OneDrive – `tutoriale_pomoce/lib_matplotlib/matplotlib_cheatsheet[*].pdf`

OneDrive – `https://wutwaw-my.sharepoint.com/:f:/g/personal/kinga_wezka_pw_edu_pl/EsQqBLWPPMdKqIxCaQSn8xQBYrdo87VD8OFOzBD_v_GBXw?e=AqU8dw`

D. M. McGreggor. *Mastering matplotlib: A practical guide that takes you beyond the basics of matplotlib and gives solutions to plot complex data*. Packt Publishing, 2015.

S. R. Poladi. *Matplotlib 3.0 Cookbook [1 ed.]*. Packt Publishing, 2018.

Dziękuje za uwagę

Kinga Węzka kinga.wezka@pw.edu.pl