Metody Numeryczne

Instrukcja do laboratorium 1

Andrii Shekhovtsov

7 marca 2023

1 Zasady ogólne

Do wykonania i przesłania jest 7 zadań. Ocena za laboratorium zależy od liczby dobrze zrobionych zadań, skala ocen jest pokazana w Tabeli 1.

Tabela 1: Skala ocen.

Liczba zrobionych zadań domowych	Ocena
2	3.0
3	3.5
4-5	4.0
6	4.5
7	5.0

Kod wszystkich wykonanych programów proszę wkleić do jednego pliku, podpisując poszczególne zadania za pomocą komentarzy. Zadanie proszę wklejać po kolei (1, 2, 3...). Plik z zadaniami proszę przesłać na platformie Moodle. Plik musi mieć nazwę `numeralbumu_lab1.py`. **Plik musi być plikiem tekstowym z rozszerzeniem `.py`**. Wysłanie pracy w nieodpowiedniej formie może skutkować wystawieniem oceny niedostatecznej za te laboratorium.

UWAGA: Termin oddania zadania jest ustawiony w systemie moodle. W przypadku nie oddania zadania w terminie, uzyskana ocena będzie zmniejszana o 0,5 za każdy zaczęty tydzień opóźnienia. Zadania oddawane później niż miesiąc po terminie ustawionym na moodle są oddawane i rozliczane w trybie indywidualnym na zajęciach lub po umówieniu się z prowadzącym.

UWAGA: W przypadku wysłania zadania w formie niezgodnej z opisem w instrukcji prowadzący zastrzega prawo do wystawienia oceny negatywnej za taką pracę. Przykład: wysłanie `.zip` lub `.pdf` tam, gdzie był wymagany plik tekstowy z rozszerzeniem `.py`.

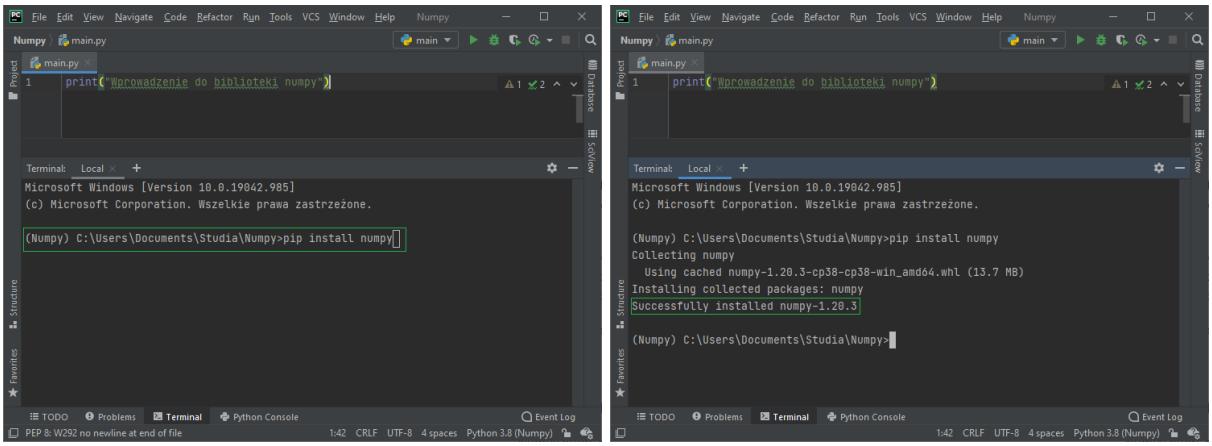
Listing 1: **Przykład dobrze sformatowanego pliku z zadaniami.**

```python
# Zadanie 1
a = 4
b = 2
print(a + b)

# Zadanie 2
# Brak

# Zadanie 3
a = 3
b = 2
print(a ** b)

# ...
```

2 Materiał pomocniczy

2.1 Instalacja bibliotek

W celu zainstalowania biblioteki Numpy w środowisku Pycharm należy wejść w zakładkę terminal znajdującą się na dole środowiska. Następnie przy pomocy komendy `pip install numpy` należy zainstalować bibliotekę Numpy. Poprawna instalacja tej biblioteki została przedstawiona przy pomocy zrzutów ekranu przedstawionych przy pomocy figury 1.



Rysunek 1: Przykład poprawnej instalacji biblioteki NumPy w środowisku Pycharm

W ten sam sposób instalujemy bibliotekę matplotlib tj. komendą `pip install matplotlib`.

2.2 Obiekt `ndarray`

Obiekt `ndarray` z biblioteki Numpy odpowiada macierzy matematycznej. Można go stworzyć przy pomocy funkcji `np.array()`, gdzie jako argument podajemy wartości naszej tabeli w strukturze typu lista lub krotka. Oprócz tego funkcja ta posiada takie przydatne argumenty jak *dtype* oraz *ndim* odpowiadające kolejno za typ tabel oraz wymiarowość tabeli. Przy pomocy listingu 2.2 przedstawiono przykład stworzenia obiektu `ndarray` oraz wyświetlenia go.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> import numpy as np
>>> # Stworzenie obiektu ndarray
>>> a = np.array([1, 2, 3])
>>> # Wyświetlenie obiektu ndarray
>>> print(a)
[1 2 3]
>>> # Wyświetlenie wymiaru obiektu ndarray
>>> print(a.shape)
(3,)
```

Obiekt `ndarray` posiada również różne wbudowane metody, które ułatwiają działania na nim np.:

- *argmax([axis, out])* - zwrócenie indeksów wartości maksymalnych wzdłuż danej osi
- *argmin([axis, out])* - zwrócenie indeksów wartości minimalnych wzdłuż danej osi
- *copy()* - zwrócenie kopii tablicy
- *flatten()* - zwrócenie wartości tablicy w jednym wymiarze
- *min([axis, out, keepdims, initial, where])* - zwrócenie minimum dla danej osi

- *mean([axis, out, keepdims, initial, where])* - zwrócenie średniej dla danej osi
- *max([axis, out, keepdims, initial, where])* - zwrócenie maksimum dla danej osi
- *nonzero()* - zwrócenie indeksów elementów niezerowych
- *reshape(shape[, order])* - zwrócenie tablicy dla nowego rozmiaru
- *tolist()* - zwrócenie tablicy w formie listy z n-zagnieżdżonymi listami

Reszta metod obiektu `ndarray` znajduje się w dokumentacji pod linkiem: https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html.

Poniżej zostało przedstawione użycie kilku metod klasy `ndarray`.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> import numpy as np
>>> # Stworzenie dwuwymiarowej tabeli
>>> a = np.array([[0, 1], [2, 3]])
>>> # Wyświetlenie tabeli a
>>> print(a)
[[0 1]
 [2 3]]
>>> # Wyświetlenie indeksów największych wartości tabeli dla kolumn
>>> print(a.argmax(axis=0))
[1 1]
>>> # Wyświetlenie średnich wartości tabeli dla kolumn
>>> print(a.mean(axis=0))
[1. 2.]
>>> # Wyświetlenie indeksów niezerowych elementów tabeli
>>> print(a.nonzero())
(array([0, 1, 1], dtype=int64), array([1, 0, 1], dtype=int64))
>>> # Wyświetlenie tabeli sformatowanej do jednego wymiaru
>>> print(a.flatten())
[0 1 2 3]
```

2.3 Przykładowe macierze

Istnieją również różne funkcje pomagające tworzyć tabele *ndarray*. Biblioteka *NumPy* umożliwia tworzenie przykładowo macierzy zerowych, jedynkowych, pustych czy losowych. Macierze te przyjmują jako argument wejściowy rozmiar. Ponadto można również zdefiniować inne parametry np. typ tabeli. Przy pomocy poniższego przykładu zaprezentowano tworzenie wyżej wymienionych macierzy.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> import numpy as np
>>> # Stworzenie tabeli wypełnionej zerami o wymiarze 2x4
>>> arr_zeros = np.zeros((2, 4))
>>> print(arr_zeros)
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
>>> # Stworzenie tabeli wypełnionej jedynkami o wymiarze 3x2
>>> arr_ones = np.ones((3, 2))
>>> print(arr_ones)
[[1. 1.]
 [1. 1.]
 [1. 1.]]
>>> # Stworzenie tabeli pustej o wymiarze 3x3
>>> arr_empty = np.empty((3,3))
```

```
>>> print(arr_empty)
[[0.00000000e+000 0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000 3.26083326e-321]
 [1.05700345e-307 2.03414517e-310 1.70182262e-123]]
>>> # Stworzenie tabeli losowej z zakresu (0, 1] o wymiarze 3x3
>>> arr_rand = np.random.random((3,3))
>>> print(arr_rand)
[[0.68315047 0.15083053 0.29382465]
 [0.69909852 0.52137945 0.06072993]
 [0.58139614 0.50988131 0.21379071]]
```

2.4 Podstawowe operacje

Na tabelach pochodzących z biblioteki *NumPy* można wykonywać również różne operacje arytmetyczne takie jak dodawanie, odejmowanie, mnożenie czy dzielenie. Przy pomocy przykładu 2.4 zaprezentowano kilka operacji arytmetycznych na utworzonych przedtem tablicach.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> import numpy as np
>>> # Utworzenie dwóch macierzy
>>> a = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
>>> b = np.array([[3, 2, 1], [3, 2, 1], [3, 2, 1]])
>>> # Podniesienie do potęgi 2 macierzy a
>>> print(a ** 2)
[[1 4 9]
 [1 4 9]
 [1 4 9]]
>>> # Suma macierzy a oraz b
>>> print(a + b)
[[4 4 4]
 [4 4 4]
 [4 4 4]]
>>> # Różnica macierzy a oraz b
>>> print(a - b)
[[-2  0  2]
 [-2  0  2]
 [-2  0  2]]
>>> # Iloczyn elementarny macierzy a oraz b
>>> print(a * b)
[[3 4 3]
 [3 4 3]
 [3 4 3]]
>>> # Iloczyn macierzowy macierzy a oraz b
>>> print(a @ b)
[[18 12  6]
 [18 12  6]
 [18 12  6]]
>>> # Dzielenie z resztą macierzy a oraz b
>>> print(a / b)
[[0.33333333 1.         3.        ]
 [0.33333333 1.         3.        ]
 [0.33333333 1.         3.        ]]
>>> Dzielenie bez reszty macierzy a oraz b
>>> print(a // b)
[[0 1 3]
```

```
 [0 1 3]
 [0 1 3]]
```

Oprócz operacji arytmetycznych takich jak dodawanie możemy również porównywać między sobą obiekty typu *ndarray* przy pomocy różnych operatorów. Przy pomocy poniższego przykładu zaprezentowano kilka operacji porównań na utworzonych przedtem tablicach.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> # Utworzenie dwóch macierzy
>>> a = np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
>>> b = np.array([[3, 2, 1], [3, 2, 1], [3, 2, 1]])
>>> # Wyświetlenie elementów macierzy a i b które macierzy które są równe
>>> print(a == b)
[[False  True False]
 [False  True False]
 [False  True False]]
>>> # Wyświetlenie elementów macierzy a  które są większe od elementów macierzy b
>>> print(a > b)
[[False False  True]
 [False False  True]
 [False False  True]]
>>> print(a <= b)
>>> # Wyświetlenie elementów macierzy a  które są mniejsze lub równe od elementom macierzy b
[[ True  True False]
 [ True  True False]
 [ True  True False]]
```

2.5 Poruszanie się po obiektach typu `ndarray`

W przypadku obiektów typu `ndarray` aby odwołać się do danego elementu tabeli należy posługiwać się indeksami. Odwoływanie się poprzez indeksy jest skonstruowane w taki sam sposób jak w przypadku struktur typu lista lub typu krotka.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> import numpy as np
>>> # Utworzenie jednowymiarowej tabeli
>>> a = np.array([1, 2, 3])
>>> # Odwołanie się do drugiego elementu tabeli
>>> print(a[1])
2
>>> # Utworzenie dwuwymiarowej tabeli
>>> b = np.array([[1, 2], [3, 4]])
>>> # Odwołanie się do elementu znajdującego się w drugim wierszu i pierwszej kolumnie
>>> print(b[1, 0])
3
```

Struktura *ndarray* umożliwia również wykrojenia (slicing). Wykonuje się je w ten sam sposób, co w przypadku struktur podobnych do listy.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> import numpy as np
>>> # Utworzenie jednowymiarowej tabeli
>>> a = np.array([1, 2, 3])
>>> # Odwołanie się do wszystkich elementów tablicy a występujących po elemencie pierwszym
>>> print(a[1:])
```

```
[2 3]
>>> # Utworzenie dwuwymiarowej tabeli
>>> b = np.array([[1, 2, 3], [3, 4, 3], [3, 4, 3]])
>>> # Odwołanie się do wszystkich elementów tablicy a występujących po elemencie pierwszym
    kolumny oraz elemencie pierwszym wiersza
>>> print(b[1:, 1:])
[[4 3]
 [4 3]]
```

Możemy również iterować się po elementach macierzy *ndarray*. Poniżej zaprezentowano przykład iterowania się po elementach macierzy b. W pierwszej pętli przechodzimy przez kolejne jednowymiarowe tablice w macierzy b o nazwie vec. W drugiej pętli przechodzimy po elementach jednowymiarowej macierzy vec które są nazwane element.

Przykład:

```
>>> # Zaimportowanie biblioteki NumPy
>>> import numpy as np
>>> # Utworzenie dwuwymiarowej tabeli
>>> b = np.array([[1, 2, 3], [3, 4, 3]])
>>> Iterowanie się po elementach macierzy b
>>> for vec in b:
...     for element in vec:
...         print(element)
...
1
2
3
3
4
3
```

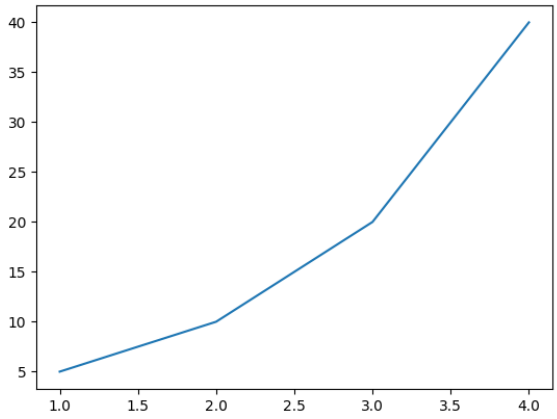
2.6 Biblioteka matplotlib

Biblioteka *matplotlib* jest biblioteką służącą do wizualizacji danych. Tworzy ona wykresy na oknach, z których w każdym oknie może być wyświetlony dowolny wykres (np. w jednym oknie wykres 2D, w drugim oknie wykres 3D). Jednym z prostszych sposób utworzenia figury jest użycie funkcji pochodzącej z modułu pyplot.subplots().

Przykład:

```
>>> # Zaimportowanie z biblioteki matplotlib modułu pyplot
>>> import matplotlib.pyplot as plt
>>> # Utworzenie listy x oraz y
>>> x = [1, 2, 3, 4]
>>> y = [5, 10, 20, 40]
>>> # Utworzenie figury oraz okna
>>> fig, ax = plt.subplots()
>>> # Wykreślenie w oknie funkcji na podstawie listy wartości x oraz y
>>> ax.plot(x, y)
[<matplotlib.lines.Line2D object at 0x000001EDFF92E160>]
>>> plt.show()
```

Za pomocą figury 2 przedstawiono utworzony wykres przy pomocy powyższego kodu.



Rysunek 2: Wykres funkcji na podstawie listy wartości x oraz y dla przykładu pierwszego biblioteki matplotlib

Możliwe jest również wykonanie wykresu z domyślnie zdefiniowanym oknem. Na początku definiujemy sobie listy współrzędnych x oraz y. Następnie przy pomocy funkcji `plot()` wykreślamy prostą przechodzącą przez te punkty. W celu wyświetlenia wykresu używamy funkcji `show()`.

Przykład:

```
>>> # Zaimportowanie z biblioteki matplotlib modułu pyplot
>>> import matplotlib.pyplot as plt
>>> # Utworzenie listy x oraz y
>>> x = [1, 2, 3, 4]
>>> y = [5, 10, 20, 40]
>>> # Wykreślenie w oknie funkcji na podstawie listy x oraz y
>>> plt.plot(x, y)
[<matplotlib.lines.Line2D object at 0x000001EDFF92E160>]
>>> plt.show()
```

2.7 Formatowanie okna

Okno wykresu pochodzące z biblioteki matplotlib możemy formatować w dowolny sposób. Przykładowo możemy ustalać przedział w którym okno ma się znajdować na osi x oraz osi y lub dodawać do niego siatkę. Kilka przydatnych funkcji formatujących okno:

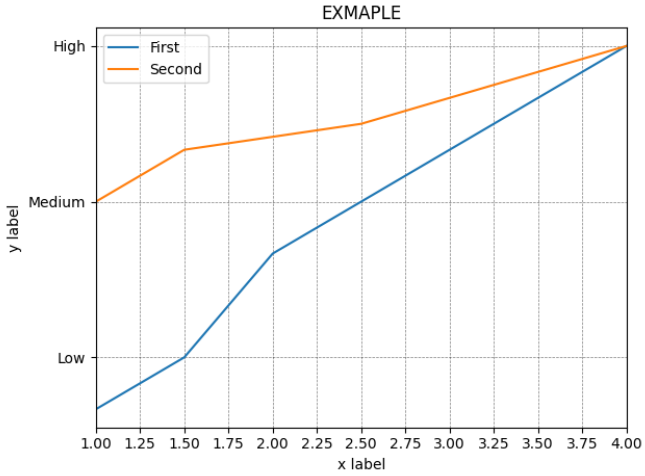
- *ax.set_title(podpis)* - ustawienie podpisu wykresu (domyślnie znajduje się on na górze wykresu)
- *ax.set_ylabel(podpis)* - ustawienie podpisu osi y
- *ax.set_xlabel(podpis)* - ustawienie podpisu osi x
- *ax.grid()* - ustawienie siatki
- *ax.set_ylim(zakres)* - ustawienie zakresu okna osi y
- *ax.set_xlim(zakres)* - ustawienie zakresu okna osi x
- *ax.legend()* - wyświetlenie legendy
- *ax.set_xticks(ticks)* - ustalenie znaczników osi x
- *ax.set_yticks(ticks)* - ustalenie znaczników osi y
- *ax.set_xticklabels(nazwy)* - ustalenie naz znaczników osi x
- *ax.set_yticklabels(nazwy)* - ustalenie naz znaczników osi y

Poniżej przedstawiono przykład użycia kilku metod formatujących okno. Na początku zdefiniowano trzy listy odpowiadające za współrzędne prostych [x, y] oraz [x, y2]. Następnie zdefiniowano znaczniki osi y i ustalono dla nich nazwy. Dla osi x został zdefiniowany przedział okna oraz znaczniki. Później została zdefiniowana figura z oknem dla której wykreślono dwie proste, ustawiono siatkę, zdefiniowano parametry osi x oraz y, wyświetlono legendę i dodano podpis wykresu. Ostatecznie dostosowano wykres, aby jego wszystkie komonenty mieściły się w oknie przy pomocy funkcji `tight_layout()` i wyświetlono okno z uzyskanym wykresem.

Przykład:

```
>>> # Zaimportowanie potrzebnych bibliotek
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> # Zdefiniowanie współrzędnych prostych
>>> x = [1, 1.5, 2, 2.5, 3, 3.5, 4]
>>> y = [1, 2, 4, 5, 6, 7, 8]
>>> y2 = [5, 6, 6.25, 6.5, 7, 7.5, 8]
>>> # Ustalenie parametrów dla osi y
>>> yticks = [2, 5, 8]
>>> yticks_names = ['Low', 'Medium', 'High']
>>> # Ustalenie paraemtrów dla osi x
>>> xlim = [1, 4]
>>> xticks = np.arange(1, 4.25, 0.25)
>>> # Stworzenie figury oraz okna
>>> fig, ax = plt.subplots()
>>> # Narysowanie prostych [x, y] oraz [x, y2] z nazwami First oraz Second
>>> ax.plot(x, y, label='First')
>>> ax.plot(x, y2, label='Second')
>>> # Zdefiniowanie czarnej siatki o stylu -- z grubością linii 0.5 oraz przezroczystością
    0.5
>>> ax.grid(color='black', linestyle='--', linewidth=0.5, alpha=0.5)
>>> # Ustalenie parametrów osi y
>>> ax.set_yticks(yticks)
>>> ax.set_yticklabels(yticks_names)
>>> ax.set_ylabel('y label')
>>> # Ustalenie parametrów osi x
>>> ax.set_xlim(xlim)
>>> ax.set_xticks(xticks)
>>> ax.set_xlabel('x label')
>>> # Wyświetlenie legendy
>>> ax.legend()
>>> # Wyświetelenie podpisu
>>> ax.set_title('EXMAPLE')
>>> # Dopasowanie okien do figury
>>> plt.tight_layout()
>>> plt.show()
```

Poniżej znajduje się wykres uzyskany przy pomocy powyższej implementacji. Lista funkcji formatujących okno wraz z ich parametrami wejściowymi znajduję się w dokumentacji pod linkiem https://matplotlib.org/stable/api/axes_api.html#axis-limits.



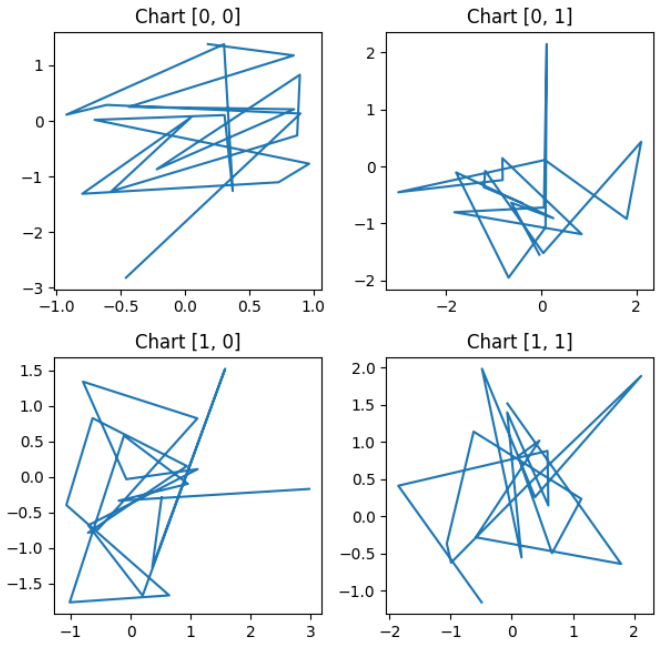
Rysunek 3: Wykres ze sformatowanym oknem dla przykładu drugiego biblioteki matplotlib

2.8 Definiowanie kilku okien w figurze

Zdefiniowana figura może również posiadać kilka okien. W poniższym przykładzie zaprezentowano jak stworzyć kilka okien przy pomocy funkcji `subplots()`. Na początku zdefiniowano figurę z 4 oknami, która posiada 2 okna w wierszu oraz 2 okna w kolumnie o rozmiarze (6, 6). Następnie iterując się po wierszach i kolumnach okien zostały wykreślane proste na poszczególnych oknach ax[i, j] oraz zostały przypisywane do nich podpisy. Po wykreśleniu prostych nastąpiło dopasowanie okien do figury oraz zapisanie figury do pliku *figura.png* przy pomocy funkcji `savefig()`. Na sam koniec wyświetlono uzyskaną figurę.

Przykład:

```
>>> # Zaimportowanie potrzebnych bibliotek
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> # Utworzenie figury która posiada 2 okna w wierszu oraz 2 okna w kolumnie o rozmiarze
    (6, 6)
>>> fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(6, 6))
>>> # Odczytanie liczby okien
>>> n, m = ax.shape
>>> # Iterowanie się po wierszach i kolumnach okien oraz tworzenie wykresów prostych
    utworzonych losowo na danym oknie
>>> for i in range(n):
...     for j in range(m):
...         ax[i, j].plot(np.random.randn(20), np.random.randn(20))
...         ax[i, j].set_title(f'Chart [{i}, {j}]')
...
>>> # Dopasowanie okien do figury
>>> plt.tight_layout()
>>> # Zapisanie figury do pliku figura.png
>>> plt.savefig('figura.png')
>>> plt.show()
```



Rysunek 4: Wykres z czterema oknami dla przykładu trzeciego biblioteki matplotlib

2.9 Przykładowe wykresy

Oprócz wykresów funkcji 2D można również tworzyć wykresy słupkowe, konturowe czy wykresy 3D. Definiowanie tego typów wykresów zostało wyraźnie opisane w dokumentacji znajdującej się na stronie `https://matplotlib.org/stable/gallery/index.html`.

3 Zadania do wykonania

1. Stwórz wektor (`np.array`) składający się z pięciu dowolnych wartości liczbowych. Wyświetl utworzony wektor. Następnie, zmień liczbę na drugiej pozycji (nie pod indeksem 2, tylko na drugiej pozycji) na -1, a liczbę znajdującą się na pozycji czwartej powiększ dwa razy.

2. Stwórz macierz dwuwymiarową o wymiarach 3 na 4 (3 wiersze, 4 kolumny) składającą się z losowych wartości za pomocą polecenia `np.random.randint`. Przedział wartości może być dowolny. Następnie, wyświetl wygenerowaną macierz.

   Wykonaj następujące polecenia:

   - Wyświetl drugi wiersz macierzy (tutaj i dalej to są pozycje, nie indeksy. Chyba że jest powiedziane *pod indeksem*);
   - Wyświetl drugą kolumnę macierzy;
   - Pomnóż cały pierwszy wiersz razy dwa.
   - Wyświetl kwadratową macierz 2x2 składają się z pierwszych dwóch elementów pierwszego wiersza i pierwszych dwóch elementów drugiego wiersza.

3. Napisz funkcję simple_plot(a, b), która wyświetli wykres, dwóch podanych wektorów a oraz b. Wektor a będzie określony przerywaną linią czerwoną, natomiast wektor b będzie określony ciągłą linią niebieską. Legenda do wykresu pojawi się w prawym górnym rogu. Oś x powinna być podpisana x, oś y powinna być podpisana y. Dodatkowo należy dodać podpis wykresu oraz siatkę. Siatka powinna być zdefiniowana zieloną linią przerywaną z przezroczystością 0.5 i grubością linii 1.15.

   Pomoc: Chodzi o to żeby utworzyć dwa wektory wartości (np.array lub lista z dowolnymi liczbami) i wrzucić je do funkcji `plt.plot()` (każdy osobno do innej).

4. Napisz funkcje func_plot(vmin, vmax, step), która wykona wykres funkcji $f(x) = x^2 - x * 4 + 8$. Argument vmin oznacza wartość początkową wektora x, vmax oznacza wartość końcową wektora x, step oznacza krok w wektorze x.

   Pomoc: Link do dokumentacji.

5. Napisz funkcję multi_plot(sizes, labels), która w jednym oknie wyświetli dwa wykresy: wykres kołowy dla podanych rozmiarów w wektorze *sizes* oraz podpisów *labels*, oraz wykres słupkowy dla podanych rozmiarów w wektorze *sizes* oraz podpisów *labels*. Jako przykładowe dane można wziąć na przykład liczbę mieszkańców 4-5 polskich miast oraz nazwy tych miast.

   Pomoc:

   - Wykres słupkowy (bar).
   - Wykres kołowy (pie).
   - Kilka wykresów w jednym oknie.

6. Napisz funkcję scatter_plot(), która wyświetli wykres punktowy dwóch zestawów danych (po 100 punktów każdy): współrzędne x i y punktów w pierwszym zestawie ma być wygenerowane za pomocą funkcji `np.random.rand()`, a współrzędne drugiego zestawu danych mają być wygenerowane za pomocą funkcji `np.ranom.randn()`. Pierwszy zestaw punktów ma być wizualizowany w kolorze niebieskim, a drugi w kolorze zielonym. Punkty drugiego zestawu danych muszą być w postaci gwiazdek (`marker`). Na wykresie ma być pokazana legenda i siatka.

   Pomoc:

   - Podstawowy przykład.
   - Dokumentacja funkcji `scatter()`.
   - Przykład legendy.

7. Napisz funkcję make_3D(x, y), która wyświetli wykres powierzchni 3D funkcji funkcji $f(x, y) = \sqrt{x^2 + y^2}$. Oś x powinna być podpisana x, oś y powinna być podpisana y, oś z powinna być podpisana jako z.

   Pomoc: Przykład takiego wykresu.