Zadanie 4 Pracownia Specjalistyczna - Eksploracja danych

Krzysztof Funkowski

In [1]:
```python
from math import *
import random
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Zadanie 4

- Konstrukcja klasyfikatora liniowego w oparciu o kryterium Fishera
- Wybór progu na podstawie dwu zbiorów danych

Funkcja służąca do obliczania kowariancji dla dwóch wektorów

In [2]:
```python
def cov(x, y, x_avg, y_avg):
    elements_count = len(x) # ilość elementów w tablicy
    elements_sum = 0

    for x,y in zip(x,y):
        elements_sum = elements_sum + ((x - x_avg)*(y - y_avg)) # sumowanie (x - średnia_arytmetyczna_x) * (y - średnia_arytmetyczna_y)

    return elements_sum / elements_count # suma dzielona przez liczbę elementów
```

Funkcja obliczająca macierz kowariancji dla zbioru danych

In [3]:
```python
def cov_for_dataset(cov_dataset):
    avgs = []
    matrix_shape = cov_dataset.shape[1] # wymiary_macierzy czyli liczba_kolumn_bioru_danych x liczba_kolumn_bioru_danych
    for i in range(matrix_shape):
        avgs.append(cov_dataset[:,i].mean()) # obliczanie średnich dla każdego parametru

    matrix = []
    for i in range(matrix_shape):
        matrix_row = []
        for j in range(matrix_shape):
            #matrix_shape*i + j
            res = cov(cov_dataset[:,i],cov_dataset[:,j],avgs[i], avgs[j]) # wyznaczanie kowariancji dla każdej pary parametrów
            matrix_row.append(res) # budowanie wiersza macierzy
        matrix.append(matrix_row) # dodawanie wiersza macierzy do tablicy bazowej

    return np.matrix(matrix) # przekształcanie typu tablicy na macierz
```

Funkcja do obliczania długości wektora własnego

In [4]:
```python
def eigenvector_len(v1):
    dims = v1.shape[0] # ilość elementów wektora
    length = 0

    for i in range(dims):
        val = v1[i].ravel().tolist()[0][0]
        length = length + pow(val,2) # sumowanie elementów podniesionych do kwadratu


    length = sqrt(length) # pierwiastkowanie wyniku sumy elementów wektora podniesionych do kwadratów

    return length # zwracanie długości
```

Funkcja normalizująca, zwracająca wartość własną i wektor własny

In [5]:
```python
def eigenvector_normalization(v1):
    eigenvalue = eigenvector_len(v1) # wylicznaie wartości własnej
    eigenvector = v1 / eigenvalue # wyliczanie wektora własnego

    return eigenvalue, eigenvector # zwrócenie wartości własnej i wektora własnego
```

Pomocnicza funkcja zmieniająca tablicę wektorów własnych w odpowiednią formę macierzy

In [6]:
```python
def rearrange_matrix(eigenvectors):
    res = []

    for i in range(len(eigenvectors)):
        row = eigenvectors[i].ravel().tolist()[0]
        res.append(row)

    return np.matrix(res).T
```

Funkcja odpowiadająca za obliczanie wartości własnych i wektorów własnych

In [7]:
```python
def eigenvalues_eigenvectors(matrix, num_iterations):
    n = matrix.shape[0] # jeden z wymiarów macierzy, ponieważ jest kwadratowa to wystarczy tylko jeden
    eigenvalues = [] # lista na wartości własne
    eigenvectors = [] # lista na wektory własne

    for i in range(n):
        # losowe dane wektora na początku
        eigenvector = np.random.rand(n,1)

        # algorytm iteracyjny wykonywany num_iterations razy
        for _ in range(num_iterations):
            eigenvector = np.dot(matrix, eigenvector) # mnożenie macierzy kowariancji przez wektor
            eigenvalue, eigenvector = eigenvector_normalization(eigenvector) # normalizacja wektora

        # usuwanie z macierzy znalezionej wartości własnej i wektora własnego
        # marix - wartość własna * iloczyn zewnętrzny wektora własnego
        matrix = matrix - eigenvalue * np.multiply(eigenvector,eigenvector.T).T

        eigenvalues.append(eigenvalue) # dodanie znalezionej wartości własnej do listy
        eigenvectors.append(eigenvector) # dodanie znalezionego wektora własnego do listy



    return eigenvalues, rearrange_matrix(eigenvectors) # zwrócenie wartości własnych i wektorów własnych
```

Rysowanie wykresu dla serii

In [8]:
```python
def draw_plot(datasets):
    colors = ['red','blue','green']

    print(len(datasets))

    # dla każdego zbioru danych z danej kategorii
    for i in range(len(datasets)):
        # przejście po wszystkich punktach
        for j in range(datasets[i][1].shape[0]):
            if j==0:
                # zaznaczanie punktu dla pierwszego (SepalLengthCm) i drugiego (SepalWidthCm) parametru
                plt.scatter(datasets[i][1][j,0],datasets[i][1][j,1], color=colors[i], label=datasets[i][0])
            else:
                # zaznaczanie punktu dla pierwszego (SepalLengthCm) i drugiego (SepalWidthCm) parametru
                plt.scatter(datasets[i][1][j,0],datasets[i][1][j,1], color=colors[i])

    plt.grid()
    plt.legend()
    plt.show()
```

ZADANIE

In [9]:
```python
df = pd.read_csv('Iris.csv', sep=',', index_col=0)
dataset = df.to_numpy()
df
```

Out[9]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
146	6.7	3.0	5.2	2.3	Iris-virginica
147	6.3	2.5	5.0	1.9	Iris-virginica
148	6.5	3.0	5.2	2.0	Iris-virginica
149	6.2	3.4	5.4	2.3	Iris-virginica
150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [10]:
```python
species = np.array(list(set(dataset[:,4])))
species
```

Out[10]:
```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa'], dtype='<U15')
```

In [11]:
```python
species1 = df[df['Species'].str.contains('Iris-virginica')].to_numpy() # wybieranie danych tylko dla badanego gatunku
species1 = species1[:, [0, 1, 2, 3]]
```

Obliczanie wektorów i wartości własnych dla klasy Iris-virginica

In [12]:
```python
num_iterations = 10000 # ustalenie liczby iteracji
species1_cov_matrix = cov_for_dataset(species1) # macierz kowariancji dla głównego w tym momencie gatunku
# wartości własne i wektory własne dla badanego gatunku
species1_eigenvalues, species1_eigenvectors = eigenvalues_eigenvectors(species1_cov_matrix,num_iterations)
print(f'Iris-virginica: \n macierz cov:\n{species1_cov_matrix} \n\n wartości własne: {species1_eigenvalues}\n wektory własne:\n{species1_eigenvectors}')
```

```
Iris-virginica:
 macierz cov:
[[0.396256 0.091888 0.297224 0.048112]
 [0.091888 0.101924 0.069952 0.046676]
 [0.297224 0.069952 0.298496 0.047848]
 [0.048112 0.046676 0.047848 0.073924]]

 wartości własne: [0.681349741460896, 0.1044202014237568, 0.0512495192248223, 0.03358053789052518]
 wektory własne:
[[ 0.74101679 -0.16525895 -0.53445017  0.37141165]
 [ 0.20328772  0.74864279 -0.3253749  -0.54068405]
 [ 0.62789179 -0.16942776  0.65152357 -0.39059336]
 [ 0.12377451  0.61928804  0.4289653   0.64587225]]
```

Projekcja klas na wektory własne klasy Iris-virginica

In [13]:
```python
current_datasets = [] # lista w której będą zbiory do rysowania
s1 = np.asarray(species1 * species1_eigenvectors[:,:2])
current_datasets.append(('Iris-virginica',s1))
```

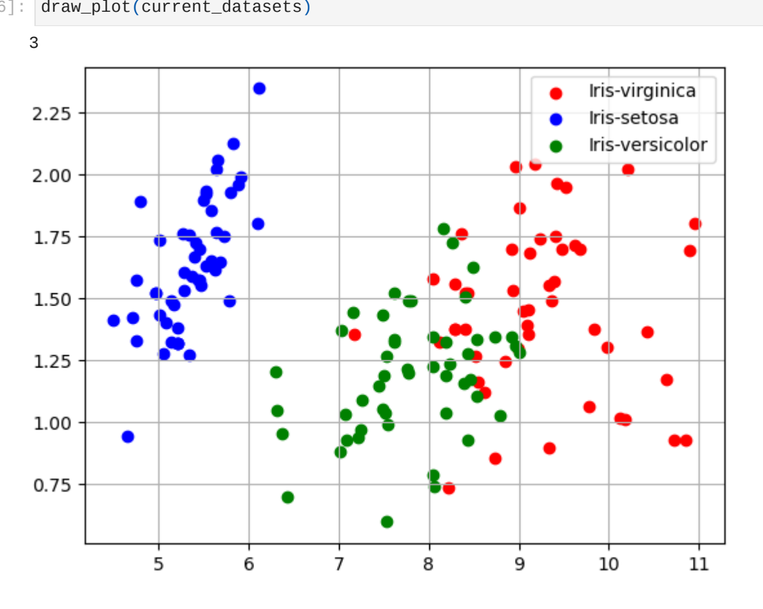
In [14]:
```python
species2 = df[df['Species'].str.contains('Iris-setosa')].to_numpy() # wybieranie danych tylko dla badanego gatunku
species2 = species2[:, [0, 1, 2, 3]]
s2 = np.asarray(species2 * species1_eigenvectors[:,:2])
current_datasets.append(('Iris-setosa',s2))
```

In [15]:
```python
species3 = df[df['Species'].str.contains('Iris-versicolor')].to_numpy() # wybieranie danych tylko dla badanego gatunku
species3 = species3[:, [0, 1, 2, 3]]
s3 = np.asarray(species3 * species1_eigenvectors[:,:2])
current_datasets.append(('Iris-versicolor',s3))
```

Trzy klasy po projekcji na wektory własne klasy Iris-virginica

In [16]:
```python
draw_plot(current_datasets)
```

3



Zmiana typu serii na macierzowy z biblioteki numpy

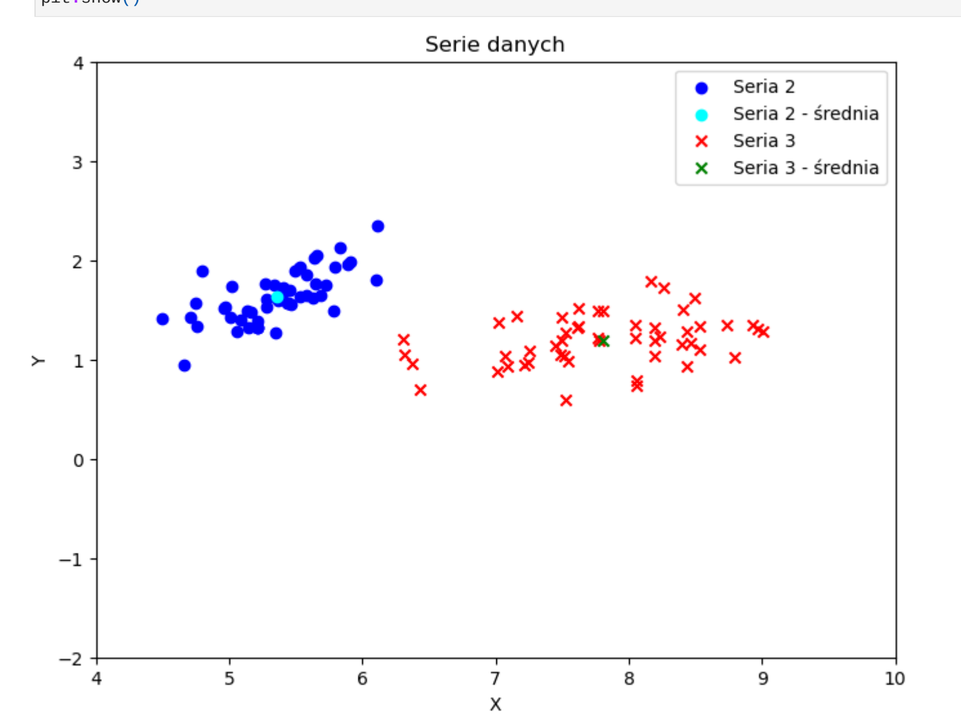
In [17]:
```python
s2_m = np.matrix(s2)
s3_m = np.matrix(s3)
```

In [18]:
```python
s2_m_mean = np.mean(s2_m, axis=0)
s3_m_mean = np.mean(s3_m, axis=0)
print(f'Wartości średnie w macierzach Setosa: {s2_m_mean}, Versicolor: {s3_m_mean}')
```

```
Wartości średnie w macierzach Setosa: [[5.353802013598783 1.634638797372408]], Versicolor: [[7.80072663740321 1.192177078876823]]
```

Wykres serii 2 (Iris-setosa) i serii 3 (Iris-versicolor) oraz ich wartości średnich

In [19]:
```python
plt.figure(figsize=(8, 6))

plt.scatter(s2[:,[0]], s2[:,[1]], label='Seria 2', color='blue', marker='o')
plt.scatter(s2_m_mean[0, 0], s2_m_mean[0, 1], label='Seria 2 - średnia', color='cyan', marker='o')
plt.scatter(s3[:,[0]], s3[:,[1]], label='Seria 3', color='red', marker='x')
plt.scatter(s3_m_mean[0, 0], s3_m_mean[0, 1], label='Seria 3 - średnia', color='green', marker='x')

plt.xlabel('X')
plt.ylabel('Y')

plt.xlim(4, 10)
plt.ylim(-2, 4)

plt.legend()
plt.title('Serie danych')
plt.show()
```



Obliczenie Sw

In [20]:
```python
Sw = cov_for_dataset(s2)+cov_for_dataset(s3) # dodanie macierzy kowariancji s2 i s3 do siebie
Sw = Sw.astype(np.float64)
Sw
```

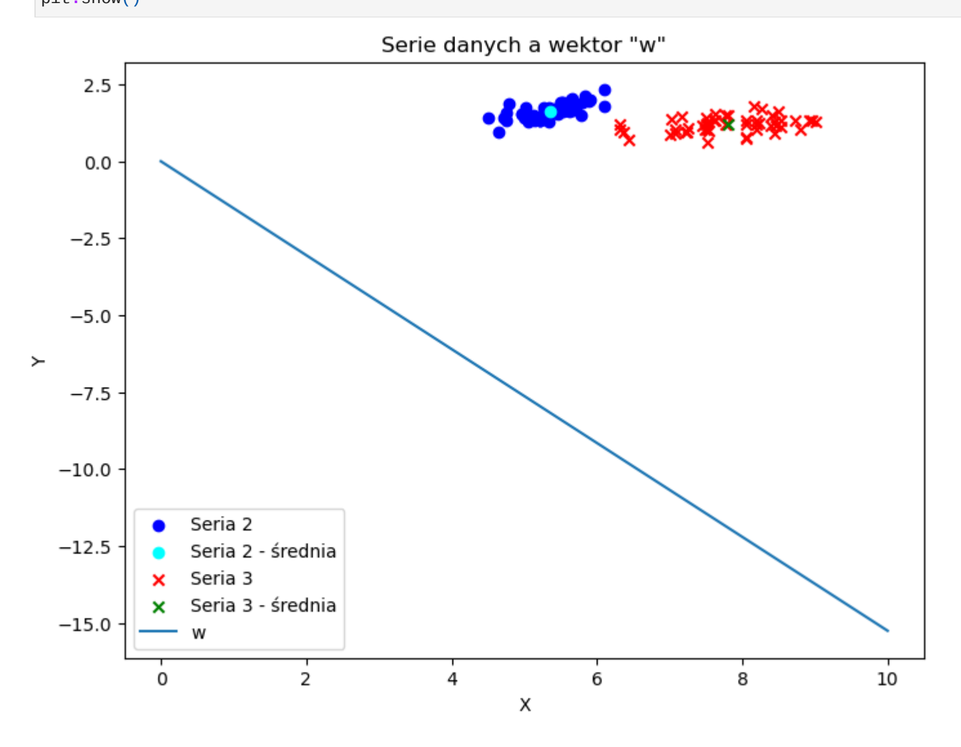
Out[20]:
```
matrix([[0.6042078 , 0.1255536 ],
        [0.1255536 , 0.13117688]])
```

Obliczamy wektor "w"

In [21]:
```python
w = np.linalg.inv(Sw) * (s3_m_mean.T - s2_m_mean.T)
w = w / sqrt(np.dot(w.T, w)[0, 0]) # [0, 0] ponieważ wyciągamy ten element z obiektu np.Matrix żeby przyjęło go sqrt
w
```

Out[21]:
```
matrix([[0.5481254282141249],
        [-0.8363961471366796]], dtype=object)
```

In [22]:
```python
plt.figure(figsize=(8, 6))

plt.scatter(s2[:,[0]], s2[:,[1]], label='Seria 2', color='blue', marker='o')
plt.scatter(s2_m_mean[0, 0], s2_m_mean[0, 1], label='Seria 2 - średnia', color='cyan', marker='o')
plt.scatter(s3[:,[0]], s3[:,[1]], label='Seria 3', color='red', marker='x')
plt.scatter(s3_m_mean[0, 0], s3_m_mean[0, 1], label='Seria 3 - średnia', color='green', marker='x')

x = np.linspace(0, 10, 100)
plt.plot(x, x * w[1,0] / w[0,0], label="w")

plt.xlabel('X')
plt.ylabel('Y')

#plt.xlim(4, 10)
#plt.ylim(-2, 4)

plt.legend()
plt.title('Serie danych a wektor "w"')
plt.show()
```



Projekcja serii na wektor "w"

In [23]:
```python
s2_m_proj_w = s2_m * np.multiply(w,w.T).T # przemnożenie serii 2 przez iloczyn zewnętrzny
s2_m_proj_w
```

Out[23]:
```
matrix([[0.8578293219828673, -1.308979848180032],
        [0.9392199034571735, -1.4331754524236113],
        [0.7964811390148981, -1.2153673623746835],
        [0.8481193257165591, -1.2941631601597146],
        [0.7997758846206464, -1.2203948841581957],
        [0.8650427264973992, -1.319986934210232],
        [0.7121731087929613, -1.08671996157801],
        [0.8828356189666946, -1.3471374839598091],
        [0.8182362505307106, -1.248563945703502],
        [0.9623104311243441, -1.4684097754126002],
        [0.9175519122168303, -1.4001118077599037],
        [0.8497884785883009, -1.2967101557177485],
        [0.9340529486725023, -1.4252910871818654],
        [0.7049597042784299, -1.0757128755478116],
        [0.8723725363134627, -1.3311716455442315],
        [0.7602278806355383, -1.1600477510797995],
        [0.7585151676260623, -1.157434286189211],
        [0.8331567617194242, -1.2713314682981196],
        [1.0074477542192906, -1.5372857683612957],
        [0.7751468844949383, -1.1828129736088389],
        [1.055457458594819, -1.6105447884649506],
        [0.7786882465456024, -1.1882168126251056],
        [0.5738902655568536, -0.8757112556425437],
        [0.9201351549742554, -1.4040536323464199],
        [0.9296841477418035, -1.4186246417335144],
        [1.0223231979409564, -1.55998452155778],
        [0.8601223881576432, -1.31247888620124],
        [0.9143007267488159, -1.3951507553089462],
        [0.9158827593450882, -1.3975648122018691],
        [0.8763768081684008, -1.3372818483904492],
        [0.9344302455306213, -1.4258668124122855],
        [0.9528485586322653, -1.4539717046906162],
        [0.7696897531276177, -1.1744858218017846],
        [0.7796899259765753, -1.1897452963874107],
        [0.9623104311243441, -1.4684097754126002],
        [0.8593677944414055, -1.3113274357404012],
        [0.9505554924574889, -1.450472666669408],
        [0.9623104311243441, -1.4684097754126002],
        [0.7633904384987694, -1.1648735648000674],
        [0.9126751340148087, -1.392670229083467],
        [0.7766853569534763, -1.1851605611692073],
        [0.9660548494821886, -1.4741234623290653],
        [0.7069625938705559, -1.07876912700371],
        [0.7825633453166508, -1.194129907539237],
        [0.8570018831028327, -1.3077172417479477],
        [0.8847078281456164, -1.3499943274180413],
        [0.8264513344762157, -1.2610995154960063],
        [0.793273513684618, -1.2104727792562802],
        [0.8877123971687161, -1.3545790626362457],
        [0.8844176515629669, -1.3495515408527323]], dtype=object)
```

In [24]:
```python
s3_m_proj_w = s3_m * np.multiply(w,w.T).T # przemnożenie serii 2 przez iloczyn zewnętrzny
s3_m_proj_w
```

Out[24]:
```
matrix([[2.0922035123665705, -3.192537449814546],
        [1.835230082378775, -2.8004162751801758],
        [2.119169138804789, -3.233684867717665],
        [1.7367854197104253, -2.650197488884146],
        [2.0045571764011507, -3.0587960579018043],
        [1.7885542868252902, -2.729192694671112],
        [1.8057678641887804, -2.7554592552869384],
        [1.417128859873123, -2.1624268048572386],
        [2.055527889662043, -3.136573343891107],
        [1.483106735328379, -2.2631038360375584],
        [1.6130878436133333, -2.461444749584122],
        [1.6625646826129161, -2.5369425013424793],
        [1.9882145980554302, -3.033858573058618],
        [1.9082896438758654, -2.911899400396163],
        [1.4908138420025612, -2.274864272601978],
        [1.9510032203828325, -2.977076947325985],
        [1.652941806622077, -2.522258751987272],
        [1.814097846106502, -2.7681701503176632],
        [2.057690275423616, -3.13987297391465],
        [1.7329103209393768, -2.644284393974283],
        [1.6919104955013795, -2.581721895931916],
        [1.774752898428575, -2.7081328651394663],
        [2.1094155824007457, -3.2188017103647915],
        [1.9858486867168579, -3.0302483790581656],
        [1.9159531904123126, -2.9235933676280266],
        [1.9493776276488255, -2.9745964211005065],
        [2.172012061244604, -3.3143189971652007],
        [2.064990800213616, -3.1510129986099593],
        [1.8005137891286398, -2.747441951380082],
        [1.6526809150654895, -2.5218606520600653],
        [1.7046528384875352, -2.601165705743549],
        [1.7026935090331439, -2.5981759236202056],
        [1.7114889461439484, -2.611597066543329],
        [1.9920604118004162, -3.039726981326059],
        [1.5932627765258491, -2.4311932617399563],
        [1.6347716172946631, -2.494532477007276],
        [2.0062263292728924, -3.061343053459839],
        [2.0820290989666743, -3.1770120978986975],
        [1.595759368277625, -2.4350028637300745],
        [1.6803575750822115, -2.5640930510920565],
        [1.7833437719028848, -2.7212418600968107],
        [1.8534438318439241, -2.828209019492728],
        [1.7663347581758895, -2.695287447446763],
        [1.4751822972353437, -2.2510117688790747],
        [1.7070330249377803, -2.604797682429867],
        [1.6769033333070165, -2.5588221507409],
        [1.6804446953576808, -2.5642259897571678],
        [1.8562741603160846, -2.832527877380711],
        [1.3440257382382983, -2.0508772103086965],
        [1.6820267279539536, -2.566640046650091]], dtype=object)
```

Wyliczenie średnich dla serii 2 i 3 po projekcji na wektor w

In [25]:
```python
s2_m_mean_proj_w = s2_m_mean * w * w.T
s2_m_mean_proj_w
```

Out[25]:
```
matrix([[0.8591040770387063, -1.3109250237956218]], dtype=object)
```

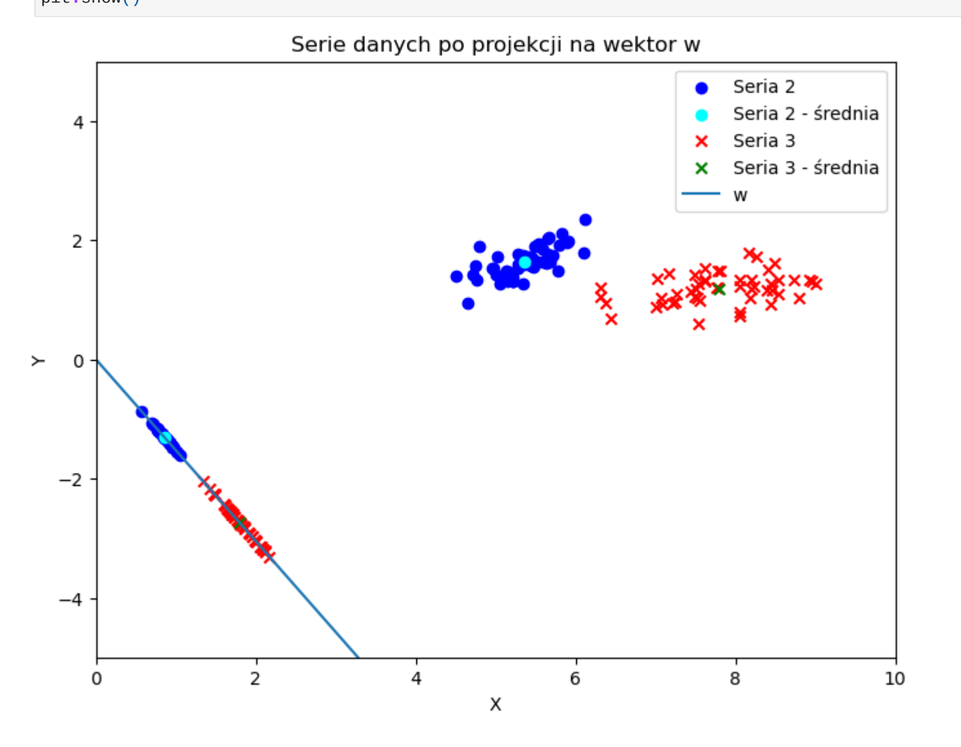
In [26]:
```python
s3_m_mean_proj_w = s3_m_mean * w * w.T
s3_m_mean_proj_w
```

Out[26]:
```
matrix([[1.797108318041774, -2.7422454712504876]], dtype=object)
```

In [27]:
```python
s2_proj_w = np.asarray(s2_m_proj_w[:,[0,1]])
s3_proj_w = np.asarray(s3_m_proj_w[:,[0,1]])
```

Pokazanie separacji klas po projekcji na wektor w

In [28]:
```python
plt.figure(figsize=(8, 6))

plt.scatter(s2_proj_w[:,[0]], s2_proj_w[:,[1]], label='Seria 2', color='blue', marker='o')
plt.scatter(s2_m_mean_proj_w[0, 0], s2_m_mean_proj_w[0, 1], label='Seria 2 - średnia', color='cyan', marker='o')
plt.scatter(s3_proj_w[:,[0]], s3_proj_w[:,[1]], label='Seria 3', color='red', marker='x')
plt.scatter(s3_m_mean_proj_w[0, 0], s3_m_mean_proj_w[0, 1], label='Seria 3 - średnia', color='green', marker='x')
plt.scatter(s2[:,[0]], s2[:,[1]], color='blue', marker='o')
plt.scatter(s2_m_mean[0, 0], s2_m_mean[0, 1], color='cyan', marker='o')
plt.scatter(s3[:,[0]], s3[:,[1]], color='red', marker='x')
plt.scatter(s3_m_mean[0, 0], s3_m_mean[0, 1], color='green', marker='x')

x = np.linspace(0, 10, 100)
plt.plot(x, x * w[1,0] / w[0,0], label="w")

plt.xlabel('X')
plt.ylabel('Y')

plt.xlim(0, 10)
plt.ylim(-5, 5)

plt.legend()
plt.title('Serie danych po projekcji na wektor w')
plt.show()
```



Łączenie zbiorów w jedną kolekcję danych. Posłuży do wyboru kandydatów na theta

In [29]:
```python
s2_t = np.dot(s2_m_proj_w, w)

s3_t = np.dot(s3_m_proj_w, w)

s2_t = np.array(s2_t).flat
s2_t = list(s2_t)

s3_t = np.array(s3_t).flat
s3_t = list(s3_t)
```

In [30]:
```python
s2_tt = [[element, 'Iris-setosa'] for element in s2_t]
s3_tt = [[element, 'Iris-versicolor'] for element in s3_t]
s23_t = sorted(s2_tt + s3_tt, key=lambda x: x[0])
s23_t
```

Out[30]:
```
[[1.0470053677799158, 'Iris-setosa'],
 [1.2861284443148255, 'Iris-setosa'],
 [1.2897825159725695, 'Iris-setosa'],
 [1.2992885791000948, 'Iris-setosa'],
 [1.3838350285944934, 'Iris-setosa'],
 [1.3869597021113855, 'Iris-setosa'],
 [1.3927294724968524, 'Iris-setosa'],
 [1.4042219417467692, 'Iris-setosa'],
 [1.4141779319023449, 'Iris-setosa'],
 [1.4169847209680348, 'Iris-setosa'],
 [1.4206387926257789, 'Iris-setosa'],
 [1.422466256522567, 'Iris-setosa'],
 [1.4277085226028647, 'Iris-setosa'],
 [1.4472481531630859, 'Iris-setosa'],
 [1.453100144632869, 'Iris-setosa'],
 [1.459111078328252, 'Iris-setosa'],
 [1.4927901688426453, 'Iris-setosa'],
 [1.5077777675247037, 'Iris-setosa'],
 [1.520011148605118, 'Iris-setosa'],
 [1.547308849508879, 'Iris-setosa'],
 [1.5503540519129708, 'Iris-setosa'],
 [1.563514186698241, 'Iris-setosa'],
 [1.5650237661438262, 'Iris-setosa'],
 [1.5678305552095169, 'Iris-setosa'],
 [1.5692072359424216, 'Iris-setosa'],
 [1.578183900929097, 'Iris-setosa'],
 [1.591556405539848, 'Iris-setosa'],
 [1.5988617988838216, 'Iris-setosa'],
 [1.6106452529361863, 'Iris-setosa'],
 [1.6135315131146766, 'Iris-setosa'],
 [1.6140609112555273, 'Iris-setosa'],
 [1.6195424468100608, 'Iris-setosa'],
 [1.6650844624896195, 'Iris-setosa'],
 [1.6680501937809113, 'Iris-setosa'],
 [1.6709364539594014, 'Iris-setosa'],
 [1.6739816563634942, 'Iris-setosa'],
 [1.6786945243029396, 'Iris-setosa'],
 [1.6961157061639238, 'Iris-setosa'],
 [1.7040861463329429, 'Iris-setosa'],
 [1.704774486699396, 'Iris-setosa'],
 [1.7135127383476685, 'Iris-setosa'],
 [1.7341933862739078, 'Iris-setosa'],
 [1.7383768560725037, 'Iris-setosa'],
 [1.755639095707886, 'Iris-setosa'],
 [1.755639095707886, 'Iris-setosa'],
 [1.755639095707886, 'Iris-setosa'],
 [1.762470412346569, 'Iris-setosa'],
 [1.837987625390245, 'Iris-setosa'],
 [1.8651263840684038, 'Iris-setosa'],
 [1.9255765273172207, 'Iris-setosa'],
 [2.452040480255288, 'Iris-versicolor'],
 [2.5854098112002255, 'Iris-versicolor'],
 [2.6913224990157993, 'Iris-versicolor'],
 [2.7057798434211744, 'Iris-versicolor'],
 [2.7198406883983783, 'Iris-versicolor'],
 [2.906748518704813, 'Iris-versicolor'],
 [2.91130330055449, 'Iris-versicolor'],
 [2.9429173699695266, 'Iris-versicolor'],
 [2.982477245438138, 'Iris-versicolor'],
 [3.0151509672707077, 'Iris-versicolor'],
 [3.015626937811679, 'Iris-versicolor'],
 [3.033182912221026, 'Iris-versicolor'],
 [3.0593423457302826, 'Iris-versicolor'],
 [3.065644264228115, 'Iris-versicolor'],
 [3.0658032064537175, 'Iris-versicolor'],
 [3.068689466632208, 'Iris-versicolor'],
 [3.0867214115825243, 'Iris-versicolor'],
 [3.106393940855427, 'Iris-versicolor'],
 [3.10996854140037, 'Iris-versicolor'],
 [3.1143109534245665, 'Iris-versicolor'],
 [3.1224403358191877, 'Iris-versicolor'],
 [3.161521490775313, 'Iris-versicolor'],
 [3.1685912207523987, 'Iris-versicolor'],
 [3.2225010321649807, 'Iris-versicolor'],
 [3.237859086762289, 'Iris-versicolor'],
 [3.2535322758392855, 'Iris-versicolor'],
 [3.2630383389668114, 'Iris-versicolor'],
 [3.2848572542875534, 'Iris-versicolor'],
 [3.29444278852788 1, 'Iris-versicolor'],
 [3.309640007063905, 'Iris-versicolor'],
 [3.3481936577148605, 'Iris-versicolor'],
 [3.3814228212012036, 'Iris-versicolor'],
 [3.3865864723045327, 'Iris-versicolor'],
 [3.481483517546997, 'Iris-versicolor'],
 [3.495464891411399, 'Iris-versicolor'],
 [3.5564444328010674, 'Iris-versicolor'],
 [3.559410164092358, 'Iris-versicolor'],
 [3.622982230886554, 'Iris-versicolor'],
 [3.6272985993978284, 'Iris-versicolor'],
 [3.6343149017750345, 'Iris-versicolor'],
 [3.6571139984005114, 'Iris-versicolor'],
 [3.6601592008046038, 'Iris-versicolor'],
 [3.750104964769217, 'Iris-versicolor'],
 [3.7540500212294114, 'Iris-versicolor'],
 [3.7673690982402825, 'Iris-versicolor'],
 [3.7984537695144676, 'Iris-versicolor'],
 [3.8170159687414693, 'Iris-versicolor'],
 [3.848417668331023, 'Iris-versicolor'],
 [3.866212056078772, 'Iris-versicolor'],
 [3.9626186807668207, 'Iris-versicolor']]
```

Obliczanie theta

Wybieranie punktów kandydatów

In [31]:
```python
point_candidates = [] # lista na punkty będące kandydatami na punkt wyznaczenia podziału klas

# wybieranie pierwszego punktu który mógłby być kandydatem
prev_point = s23_t[0]

# przechodzenie po wszystkich punktach
for actual_point in s23_t[1:]:
    if actual_point[1] != prev_point[1]: # jeżeli klasa aktualnie badana różni się od klasy poprzedniej to obliczamy kandydata na punkt
        between_point = ((actual_point[0] + prev_point[0]) / 2) # średnia arytmetyczna z aktualnego i poprzednio badanego
        point_candidates.append(between_point) # dodawanie do listy kandydatów

    prev_point = actual_point

point_candidates
```

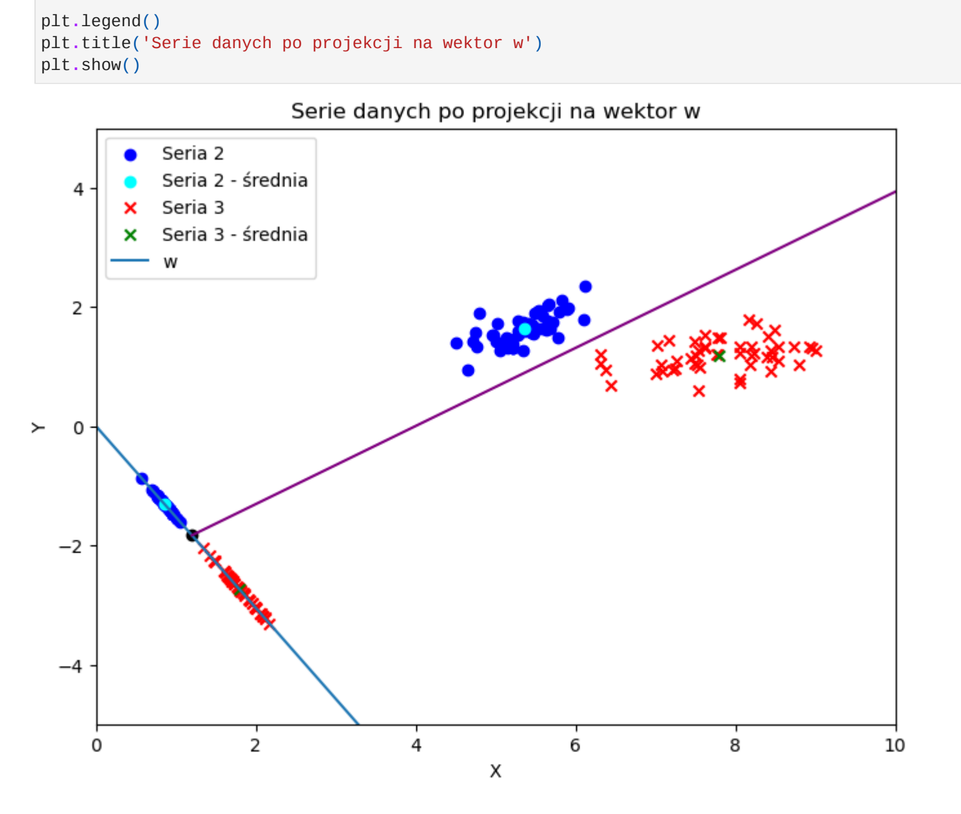
Out[31]:
```
[2.1888085037862544]
```

Z kandydatów wybierany jest theta

In [39]:
```python
best_score = 0 # zmienna na najlepszą ocenę
best_candidate = None # zmienna na kandydata z najlepszą oceną

# przejście po wszystkich punkach kandydatach
for point_candidate in point_candidates:
    # wybór punktów klasy Iris-setosa posiadające mniejsze wartości od kandydata na względem wektora "w"
    s2_good_classified = [element for element in s23_t if element[0] < point_candidate and element[1] == 'Iris-setosa']
    # zliczenie punktór klasy Iris-setosa posiadające większe wartości od kandydata
    s2_score = len(s2_good_classified)

    # wybór punktów klasy Iris-versicolor posiadające mniejsze wartości od kandydata na względem wektora "w"
    s3_good_classified = [element for element in s23_t if element[0] > point_candidate and element[1] == 'Iris-versicolor']
    # zliczenie punktór klasy Iris-versicolor posiadające większe wartości od kandydata
    s3_score = len(s2_good_classified)

    current_score = s2_score + s3_score # sumowanie ilości dobrze sklasyfikowanych zbiorów

    # jeżeli nowa ocena jest lepsza od obecnie uznawanej za najlepszą
    if current_score > best_score:
        # to nowa ocena jest ustawiana jako najlepsza
        best_score = current_score
        # oraz jako najlepszy kandydat jest ustawiany nowy punkt
        best_candidate = point_candidate

theta = best_candidate * w # rzutowanie najlepszego kandydata na wektor "w"
best_score, best_candidate
```

Out[39]:
```
(100, 2.1888085037862544)
```

In [40]:
```python
plt.figure(figsize=(8, 6))

plt.scatter(s2_proj_w[:,[0]], s2_proj_w[:,[1]], label='Seria 2', color='blue', marker='o')
plt.scatter(s2_m_mean_proj_w[0, 0], s2_m_mean_proj_w[0, 1], label='Seria 2 - średnia', color='cyan', marker='o')
plt.scatter(s3_proj_w[:,[0]], s3_proj_w[:,[1]], label='Seria 3', color='red', marker='x')
plt.scatter(s3_m_mean_proj_w[0, 0], s3_m_mean_proj_w[0, 1], label='Seria 3 - średnia', color='green', marker='x')
plt.scatter(s2[:,[0]], s2[:,[1]], color='blue', marker='o')
plt.scatter(s2_m_mean[0, 0], s2_m_mean[0, 1], color='cyan', marker='o')
plt.scatter(s3[:,[0]], s3[:,[1]], color='red', marker='x')
plt.scatter(s3_m_mean[0, 0], s3_m_mean[0, 1], color='green', marker='x')

x = np.linspace(0, 10, 100)
plt.plot(x, x * w[1,0] / w[0,0], label="w")

plt.scatter(theta[0][0,0], theta[1][0,0], color='black')
x_div = x + theta[0][0,0]
y_div = -x / (w[1,0] / w[0,0]) + theta[1][0,0]
plt.plot(x_div,y_div, color='purple')

plt.xlabel('X')
plt.ylabel('Y')

plt.xlim(0, 10)
plt.ylim(-5, 5)

plt.legend()
plt.title('Serie danych po projekcji na wektor w')
plt.show()
```



Wirosaki

Uldki nie pamietal wymyslyl podnet pomyslu kleeme wnerii a uroczyszli kriwuli. Na szmiewili widal wspomnieniuli wykwiczli nie.