1 Metoda potęgowa

Napisz funkcję obliczającą metodą potęgową dominującą wartość własną (największą co do modułu) i odpowiadający jej wektor własny dla danej macierzy rzeczywistej symetrycznej. Sprawdź poprawność działania programu porównując własną implementację z wynikami funkcji bibliotecznej. Przedstaw na wykresie zależność czasu obliczeń od rozmiaru macierzy (rozmiary macierzy 100x100, 500x500, ...).

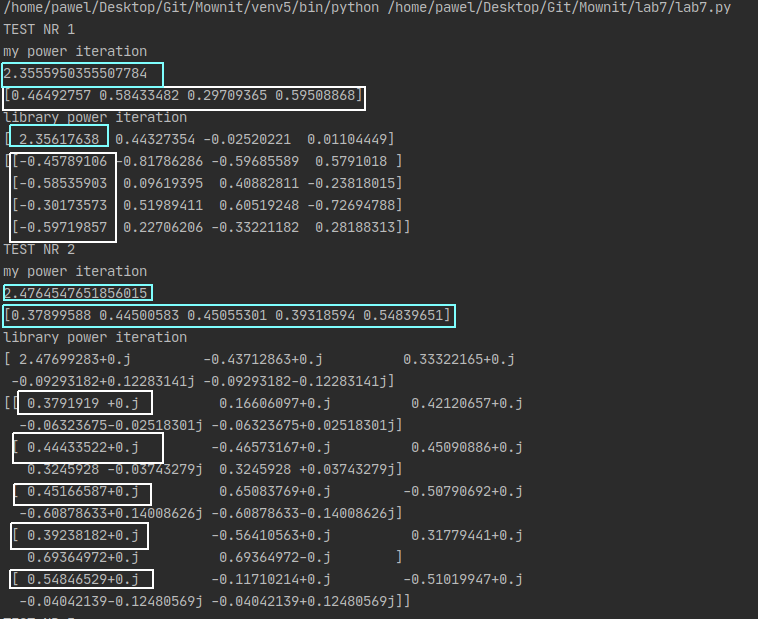
Metoda potęgowa

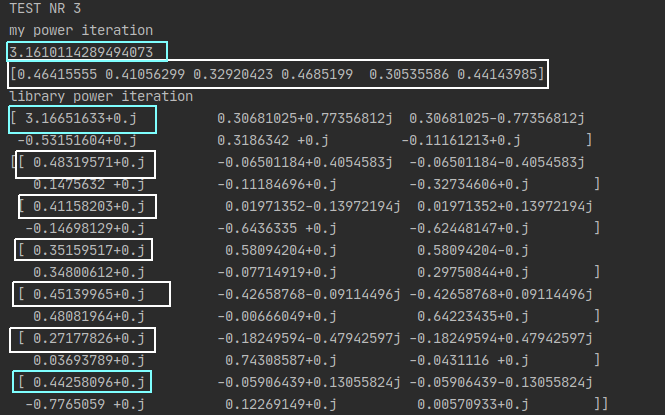
ef eigenvalue(A, v):  
 Av = A.dot(v)  
 return v.dot(Av)  
  
def power\_iteration(A, max\_iteration, eps):  
 n, d = A.shape  
  
 v = np.ones(d) / np.sqrt(d)  
 ev = eigenvalue(A, v)  
  
 for i in range(max\_iteration):  
 Av = A.dot(v)  
 v\_new = Av / np.linalg.norm(Av)  
  
 ev\_new = eigenvalue(A, v\_new)  
 if np.abs(ev - ev\_new) < eps:  
 break  
  
 v = v\_new  
 ev = ev\_new

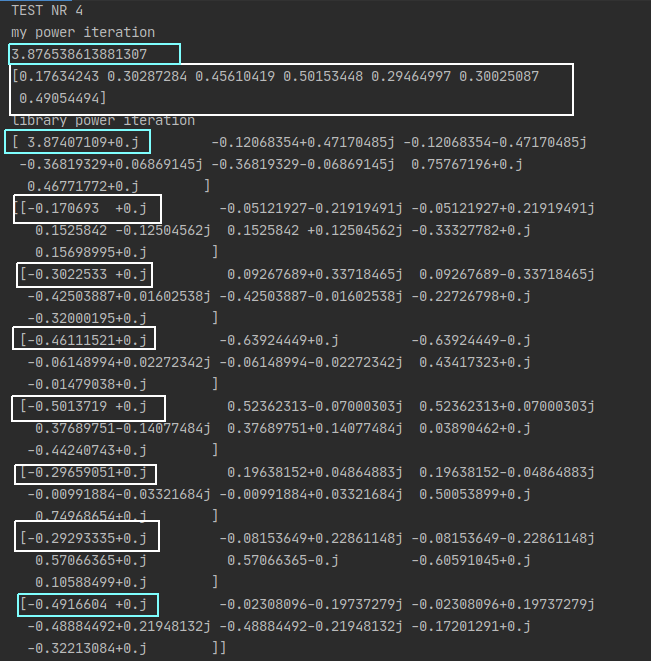
Test poprawności metody potęgowej

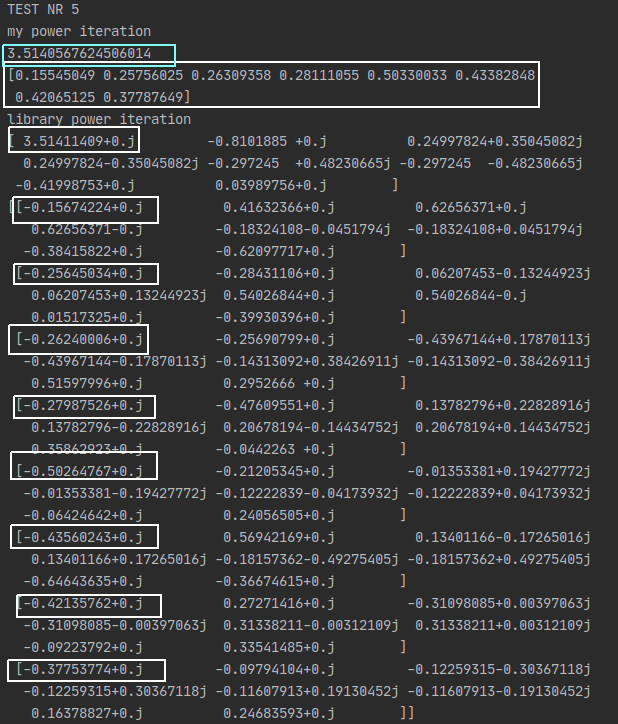
def check\_power\_iteration():  
 np.set\_printoptions(suppress=True)  
 j = 0  
 for i in range(4, 10):  
 j += 1  
 print("TEST NR {0}".format(j))  
 arr = generate\_matrix(i)  
 copy\_arr = np.copy(arr)  
 e\_val, e\_vec = power\_iteration(arr, 10000, 0.01)  
 print("my power iteration")  
 print(e\_val)  
 print(e\_vec)  
 e\_val, e\_vec = np.linalg.eig(copy\_arr)  
 print("library power iteration")  
 print(e\_val)  
 print(e\_vec)

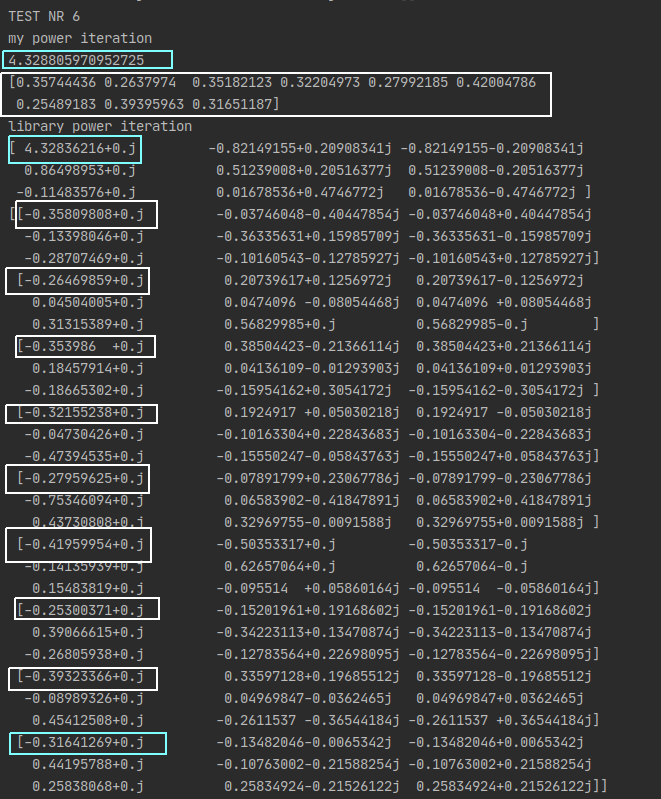
Wyniki testu







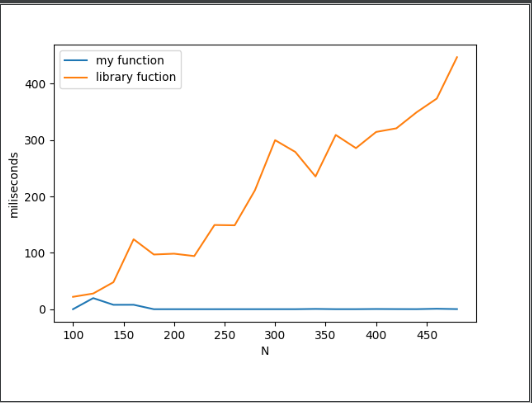


****

**Wioski:**Zaimplementowana funkcja daje wyniki zgodne z funkcją biblioteczną

Funkcja mieżąca czas

def test\_power\_iteration():  
 my\_func = []  
 lib\_func = []  
 sizes = []  
 for size in range (100,500,20):  
 sizes.append(size)  
 arr = generate\_matrix(size)  
 copy\_arr = np.copy(arr)  
  
 start\_time = time.time()  
 np.linalg.eig(copy\_arr)  
 diff\_time = time.time() - start\_time  
 lib\_func.append(diff\_time \* 1000)  
  
 start\_time = time.time()  
 power\_iteration(arr,1000000,0.1)  
 diff\_time = time.time() - start\_time  
 my\_func.append(diff\_time\*1000)  
  
  
  
  
 fig = plt.figure()  
 ax1 = fig.add\_axes((0.1, 0.2, 0.8, 0.7))  
 ax1.set\_xlabel('N')  
 ax1.set\_ylabel('miliseconds')  
 plt.plot(sizes, my\_func, label="my function")  
 plt.plot(sizes, lib\_func, label="library fuction")  
 plt.legend(loc="upper left")  
  
 plt.show()



**Wnioski:**

-funkcja oparta na metodzie potęgowej jest szybsza od biliotecznej, zapewne to wynika z tego, że wyliczamy tylko jedną wartość własną i tylko dla niej wektor własny

- na wykresie można zauważyć wzrost czasu potrzebnego na wykonanie funkcji opartej ma teodzi potęgowej przy wielkości macierzy około 120x120.

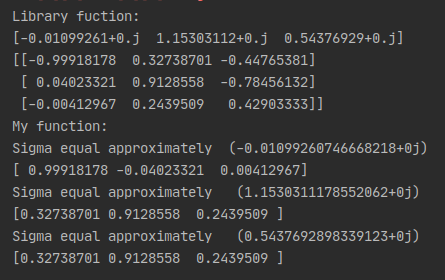
2.Odwrotna metoda potęgowa

def inverse\_power\_method(A, sgn, eps, max\_iterations):  
 size, d = A.shape  
 x0 = np.array([1.5 for i in range(size)])  
 x0 = np.array(x0/np.linalg.norm(x0, ord=np.inf))  
  
 for i in range(size):  
 A[i][i] = A[i][i] - sgn  
  
 flag = False  
 LU = scipy.linalg.lu\_factor(A)  
  
 for i in range(max\_iterations):  
 x1 = scipy.linalg.lu\_solve(LU, x0)  
 x2 = x1/np.linalg.norm(x1, ord=np.inf)  
  
 if np.linalg.norm(x2 - x0) < eps:  
 flag = True  
  
 x0 = x2  
 if flag:  
 break  
  
 return x1/np.linalg.norm(x1)

Test metody:

def check\_inverse\_power\_method():  
 A = generate\_matrix(3)  
  
 e\_val, e\_vec = scipy.linalg.eig(A)  
 print("Library fuction:")  
 print(e\_val)  
 print(e\_vec)  
 print("My function:")  
 print("Sigma equal approximately ", e\_val[0])  
 print(inverse\_power\_method(A, e\_val[0] + 0.1, 0.01, 10000))  
 print("Sigma equal approximately ", e\_val[1])  
 print(inverse\_power\_method(A, e\_val[1] + 0.2, 0.01, 10000))  
 print("Sigma equal approximately ", e\_val[2])  
 print(inverse\_power\_method(A, e\_val[2] + 0.5, 0.01, 10000))

Wyniki testu



**Wnioski**-funkcja działa poprawnie