Міністерство освіти і науки України

Львівський національний університет імені Івана Франка

факультет електроніки та комп’ютерних технологій

кафедра системного проектування

**ЗВІТ**

про виконання лабораторної роботи №3

“Python & NumPy. Аналіз особливостей бібліотеки NumPy ”

виконала студентка групи ФеС-32

Богоніс Є.А.

перевірив старший викладач

Рибак А.В.

Львів-2019

**Мета:** проаналізувати особливості бібліотеки Numpy на основі реалізованих методів в порівнянні Python vs Python + NumPy.

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

NumPy - це бібліотека мови Python, що підтримує роботу з великими багатовимірними масивами і матрицями, разом з великою бібліотекою високорівневих (і дуже швидких) математичних функцій для операцій з цими масивами.

**Завдання:**

1. Дослідити базові методи бібліотеки NymPy.

2. Реалізувати наступні методи без використання бібліотеки NumPy: множення двох матриць (N x M), множення матриці (N x M) на вектор (N), множення вектора (N) на матрицю (N x M) та множення двох векторів (N).

Примітка:

- N є в межах [100 - 1000];

- M є в межах [100 - 1000];

- Матриці та вектори ініціалізуємо випадковими даними в межах [0; 1];

3. Реалізувати методи, які описані в пункті 2, з використанням бібліотеки NumPy.

4. Написати Unit tests для методів.

5. Провести аналіз особливостей бібліотеки NumPy на основі реалізованих методів в порівнянні Python vs Python + NumPy. Характеристика аналізу - час виконання методу.

**Виконання завдання**

1. Дослідили базові методи Numpy, які висвітлили у 16 прикладах, у файлі під назвою *numpymethods.py* ( при виконанні завдання було використане таке програмне середовище, як Colaboratory ).

2. Реалізували потрібні нам методи без використання бібліотеки NumPy (файл *matrixwithoutnumpy.py*)

Лістинг програми:

from itertools import starmap  
from operator import mul  
import random  
# starmap(function,iterable)- застосовує ф-цію до кожного елемента послідовності  
# mul - множення  
  
class My\_Matrix:  
   
 def create\_random\_matrix(self, rows, columns):  
 random.seed(3)  
 matr = [[random.random()  
 for i in range(columns)]  
 for j in range(rows)]  
 return matr  
  
 def create\_vector(self, element):  
 random.seed(1)  
 return [random.random() for i in range(element)]  
  
 def multy\_two\_matrix(self, a, b):  
 c = [[0 for row in range(len(b[0]))] for col in range(len(a))]  
 for i in range(len(a)):  
 for j in range(len(b[0])):  
 for k in range(len(a[0])):  
 c[i][j] += a[i][k] \* b[k][j]  
 return c  
  
 def multy\_vector\_on\_vector(self, vector1, vector2):  
 return [vector1[i] \* vector2[i] for i in range(len(vector1))]  
  
 def multy\_vector\_matrix(self, vector, matrix):  
 return [sum(starmap(mul, zip(vector, col))) for col in zip(\*matrix)]  
   
if \_\_name\_\_ == '\_\_main\_\_':  
 A = My\_Matrix()  
  
 m1 = A.create\_random\_matrix(20,5)  
 m2 = A.create\_random\_matrix(5,20)  
 print("Matrix A:", m1)  
 print("Matrix B:", m2)  
  
 v1 = A.create\_vector(20)  
 v2 = A.create\_vector(20)  
 print("Vector V1:", v1)  
 print("Vector V2:", v2)  
  
 c1 = A.multy\_two\_matrix(m1,m2) #множення матриць  
 print("Multy two matrix:",c1)  
  
 k1 = A.multy\_vector\_on\_vector(v1,v2) #множення векторів  
 print("Multy vector on vector:",k1)  
  
 f1 = A.multy\_vector\_matrix(v1,m1) #множення вектора на матрицю  
 print("Multy vector on matrix:",f1)

Результат програми:

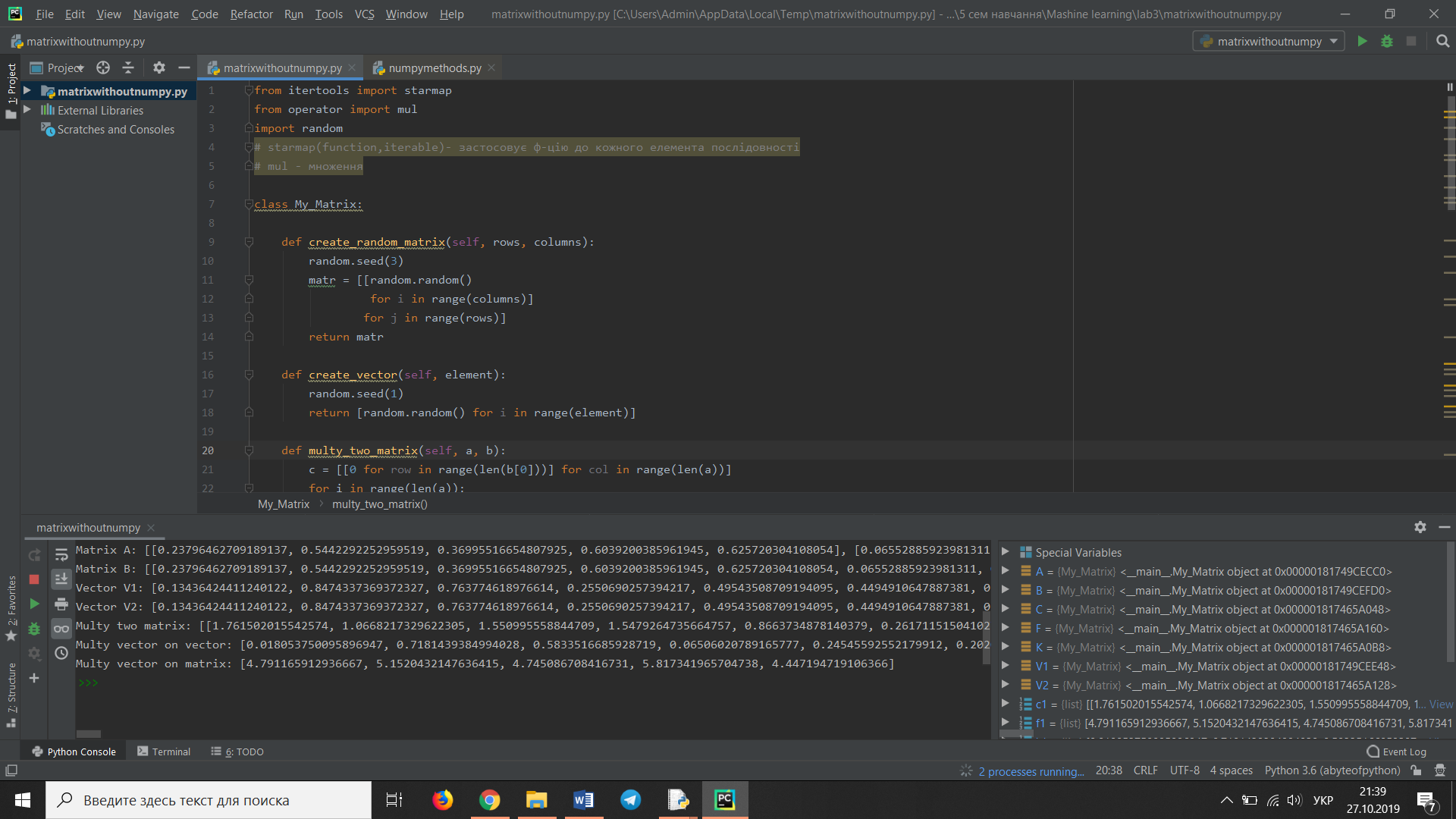


Рис.1 Реалізація методів без використання Numpy

3. Ті ж самі методи були реалізовані, але вже з використанням бібліотеки NumPy (файл *numpymatrix.py)*.

Лістинг програми:

import numpy as np  
import random  
  
#множення двох матриць  
A = np.random.rand(20,5) #N=200  
B = np.random.rand(5,20) #N=200  
C = np.random.rand(5)  
C1 = np.random.rand(5)  
K = np.dot(A,B) #множення матриць  
L = np.dot(A,C) #множення матриці на вектор  
L1 = np.dot(C,B) #множення вектора на матрицю  
P = np.dot(C,C1) #множення вектора на вектор  
print("Matrix A:", A)  
print("Matrix B:", B)  
print("Vector C:", C)  
print("Vector C1:", C1)  
print("Множення двох матриць:", K)  
print("Множення матриці на вектор:", L)  
print("Множення вектора на матрицю:", L1)  
print("Множення вектора на вектор:", P)  
  
import time  
  
  
start\_time = time.clock()  
print ("program execution time: ", time.clock() - start\_time, "seconds")

Результат програми:

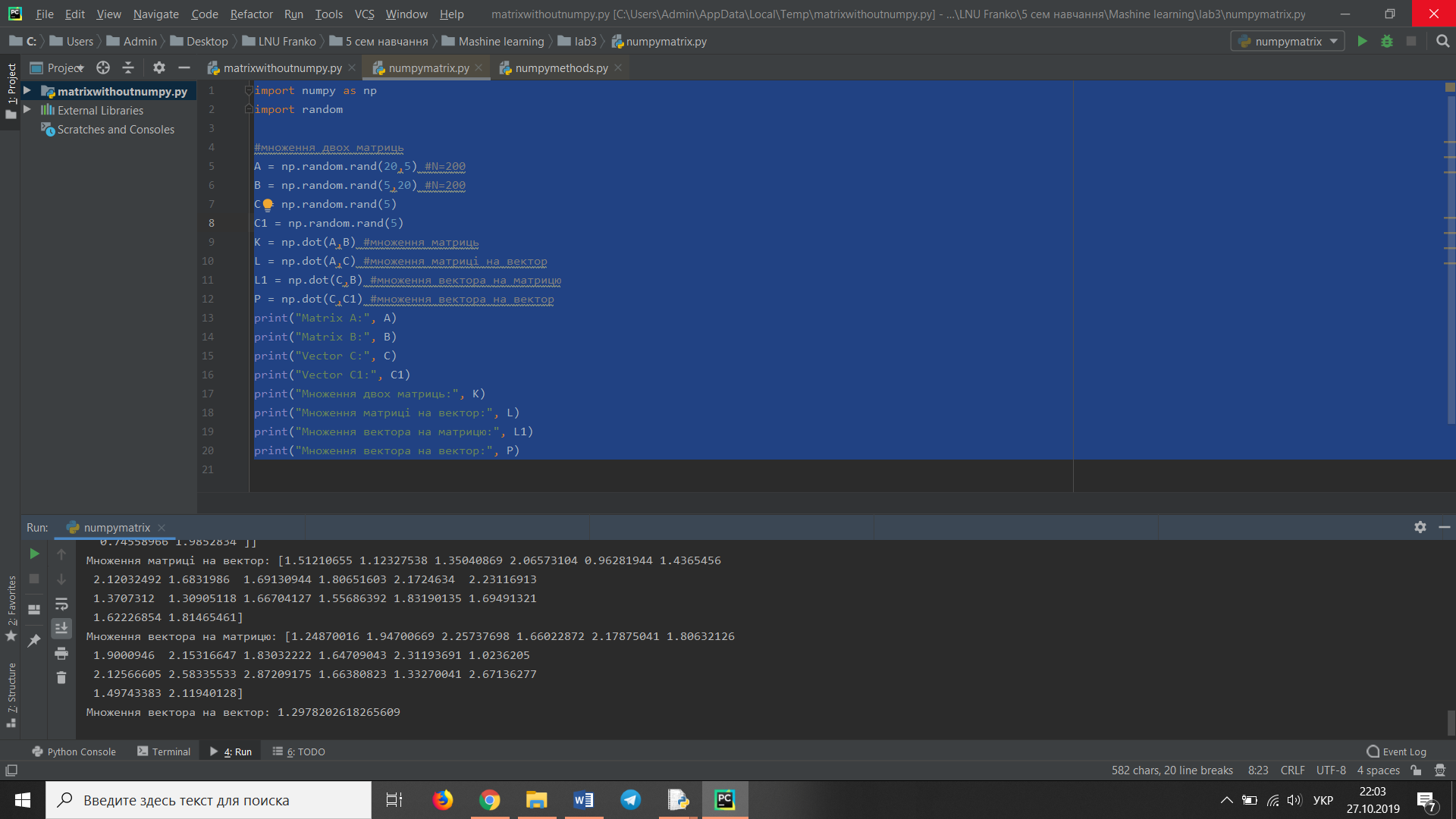


Рис.2 Реалізація методів з використанням Numpy

4. Написати Unit tests для методів.

Unit tests зробили для методів, які були написані без використання бібліотеки NumPy , для того, щоб перевірити їх на коректність, вони записані у файлі unittest*.py* у вигляді функцій :

import numpy as np  
from itertools import starmap  
from operator import mul  
  
  
def generate\_data(n):  
 np.random.seed(0)  
 a = np.random.rand(n)  
 b = np.random.rand(n)  
 return a, b  
  
  
def generate\_matrix(n, m):  
 np.random.seed(0)  
 a = np.random.rand(n, m)  
 b = np.random.rand(n, m)  
 return a, b  
  
  
def multy\_2\_vectors(v1, v2):  
 if (not isinstance(v1, np.ndarray)) or (not isinstance(v2, np.ndarray)):  
 return None  
 if len(v1) != len(v2):  
 return None  
 v3 = 0.0  
 for i in range(len(v1)):  
 v3 += v1[i] \* v2[i]  
 return v3  
  
  
def multy\_2\_matrix(m1, m2):  
 if (not isinstance(m1, np.ndarray)) or (not isinstance(m2, np.ndarray)):  
 return None  
 if len(m1[0]) != len(m2):  
 return None  
 c = [[0 for row in range(len(m2[0]))] for col in range(len(m1))]  
 for i in range(len(m1)):  
 for j in range(len(m2[0])):  
 for k in range(len(m1[0])):  
 c[i][j] += m1[i][k] \* m2[k][j]  
 return c  
  
  
def multy\_vect\_matr(v, m):  
 if (not isinstance(v, np.ndarray)) or (not isinstance(m, np.ndarray)):  
 return None  
 if np.all(len(v) != len(m)):  
 return None  
 return [sum(starmap(mul, zip(v, col))) for col in zip(\*m)]  
  
  
def multy\_2\_vectors\_numpy(v1, v2):  
 if (not isinstance(v1, np.ndarray)) or (not isinstance(v2, np.ndarray)):  
 return None  
 if len(v1) != len(v2):  
 return None  
 return np.dot(v1, v2)  
  
  
def multy\_2\_matrix\_numpy(m1, m2):  
 if (not isinstance(m1, np.ndarray)) or (not isinstance(m2, np.ndarray)):  
 return None  
 if len(m1[0]) != len(m2):  
 return None  
 return np.dot(m1, m2)  
  
  
def multy\_vect\_matr\_numpy(v, m):  
 if (not isinstance(v, np.ndarray)) or (not isinstance(m, np.ndarray)):  
 return None  
 if len(v) != len(m):  
 return None  
 return np.dot(v, m)

Сам unit test був виконаний у файлі *numpyunit.py* :

import unittest  
import unittestmy as my  
import numpy as np  
  
  
class MyTest(unittest.TestCase):  
  
 def test\_multy\_2\_vectors\_2(self):  
 print("id: " + self.id())  
 self.assertEqual(my.multy\_2\_vectors(np.array([3, 1, 3]),  
 np.array([2, 4])), None)  
  
 def test\_multy\_2\_vectors\_3(self):  
 print("id: " + self.id())  
 self.assertEqual(my.multy\_2\_vectors([3, 1, 3], [2, 2, 4]), None)  
  
 def test\_multy\_2\_matrix\_1(self):  
 print("id: " + self.id())  
 self.assertEqual(my.multy\_2\_matrix(np.array([[2,3,4],[3,4,1]]),  
 np.array([[2,1],[3,0],[4,2]])),  
 [[29,10],[22,5]])  
  
 def test\_multy\_2\_matrix\_2(self):  
 print("id: " + self.id())  
 self.assertEqual(my.multy\_2\_matrix(np.array([[2,3], [0,3]]),  
 np.array([[0,9,3], [4,1,5], [5,1,4]])), None)  
  
 def test\_multy\_vect\_matr\_1(self):  
 print("id: " + self.id())  
 self.assertEqual(my.multy\_vect\_matr(np.array([2,3,4,5]),  
 np.array([[2,0,1], [3,0,1,], [3,1,2], [4,6,0,1]])),  
 ([45, 34, 13]))  
  
 def test\_multy\_vector\_matr\_2(self):  
 print("id: " + self.id())  
 self.assertEqual(my.multy\_vect\_matr(np.array([2, 0]),  
 np.array([2, 1, 3])), None)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

Результат роботи програми(перевірки методів) :

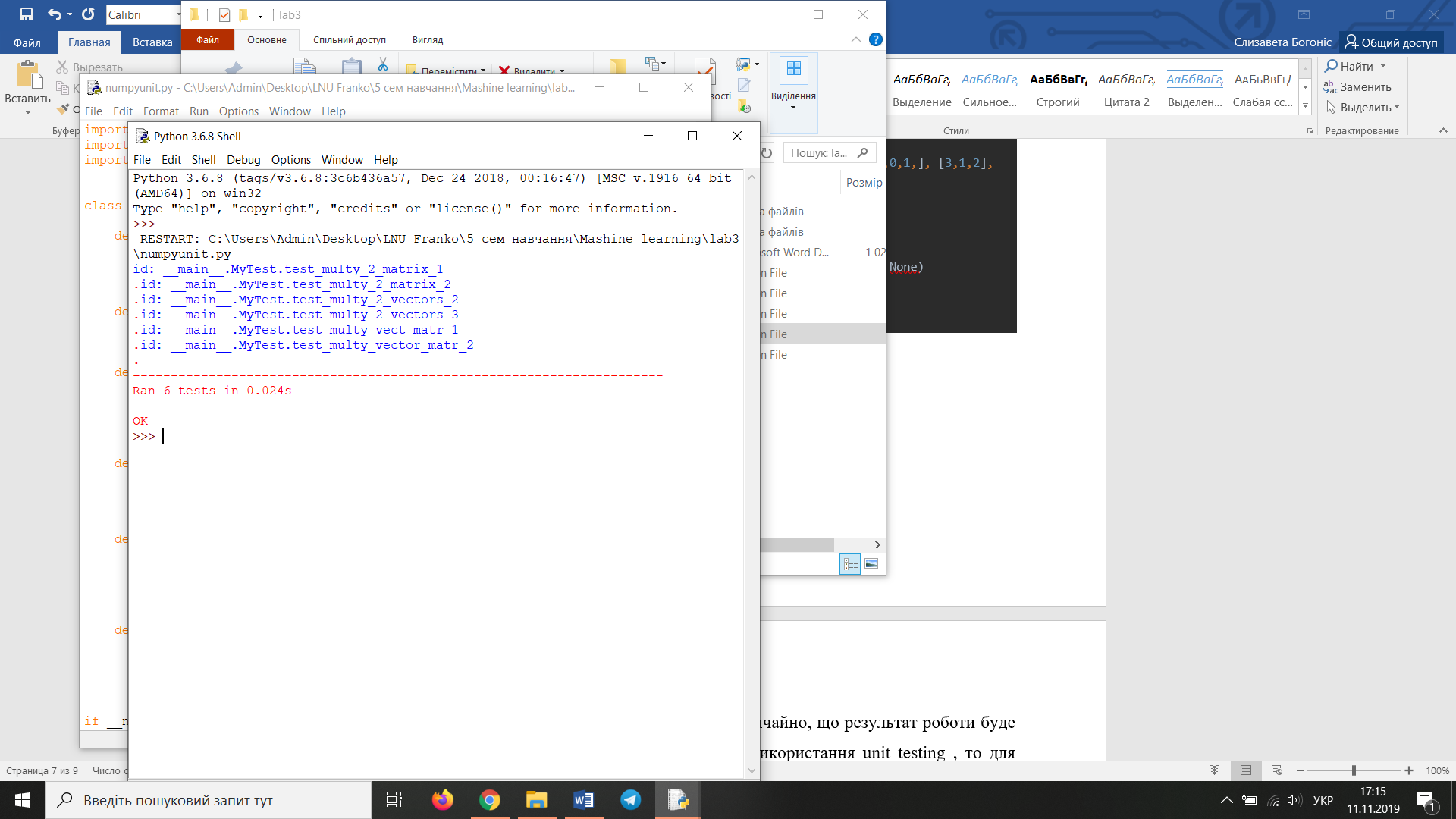


Рис.3 Unit tests для методів

5. Провести аналіз особливостей бібліотеки NumPy на основі реалізованих методів в порівнянні Python vs Python + NumPy. Характеристика аналізу - час виконання методу.

Python + NumPy є набагато практичнішим у використанні, оскільки кількість написаного коду та кількість роботи у кілька разів менша, при викорисатнні модуля numpy .

Проте, використання Python без numpy допомагає зрозуміти математичні аспекти масивів та покращити сам синтаксис мови Python.

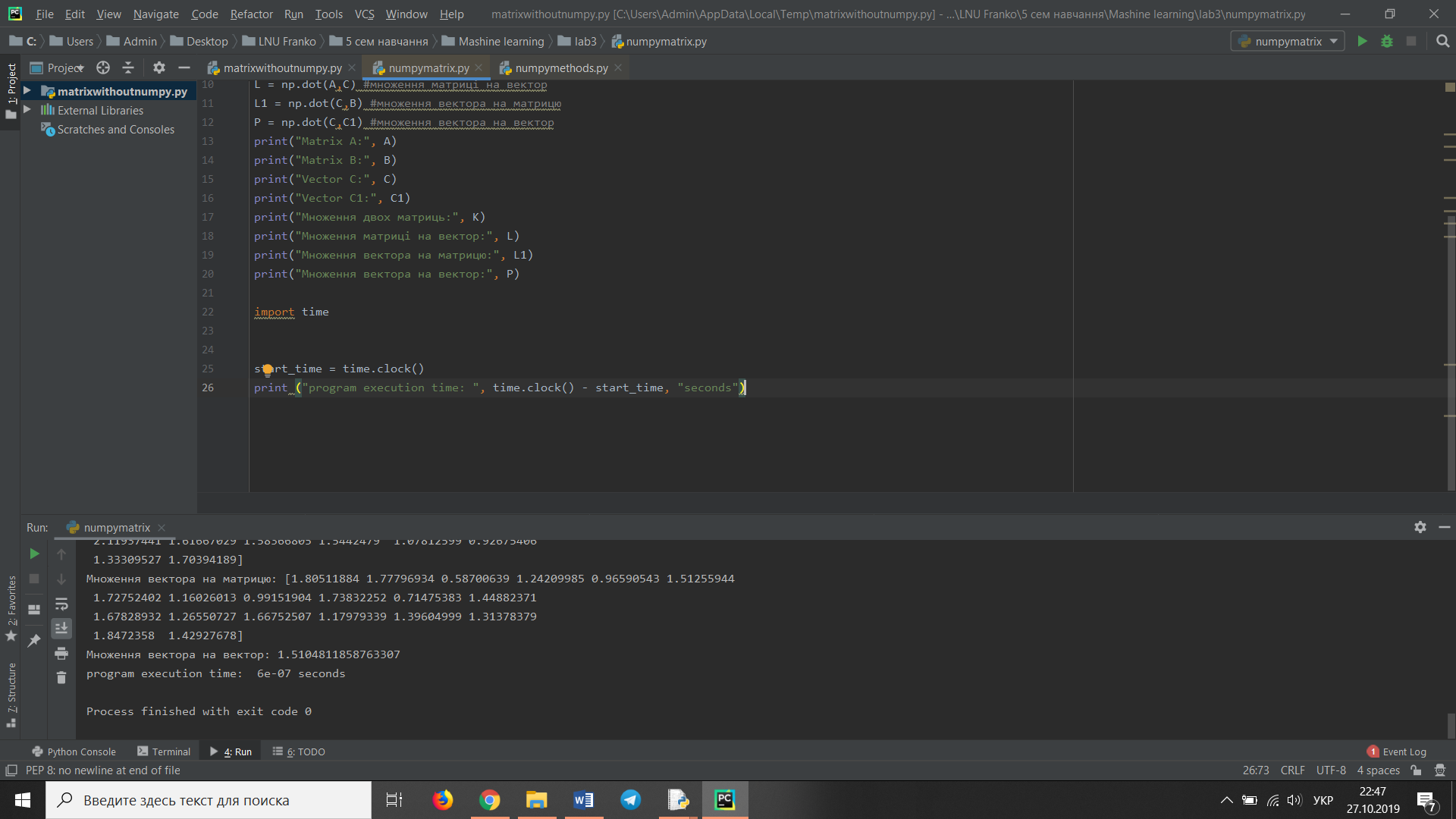
Для NumPy також існують подібні «пакети», наприклад, бібліотека SciPy надає більше MATLAB-подібної функціональності, бібліотека Matplotlib дозволяє створювати графіки в стилі MATLAB.

Аналіз часу був проведений, за допомогою модуля *time* та наступного коду:

import time  
  
  
start\_time = time.clock()  
print ("program execution time: ", time.clock() - start\_time, "seconds")

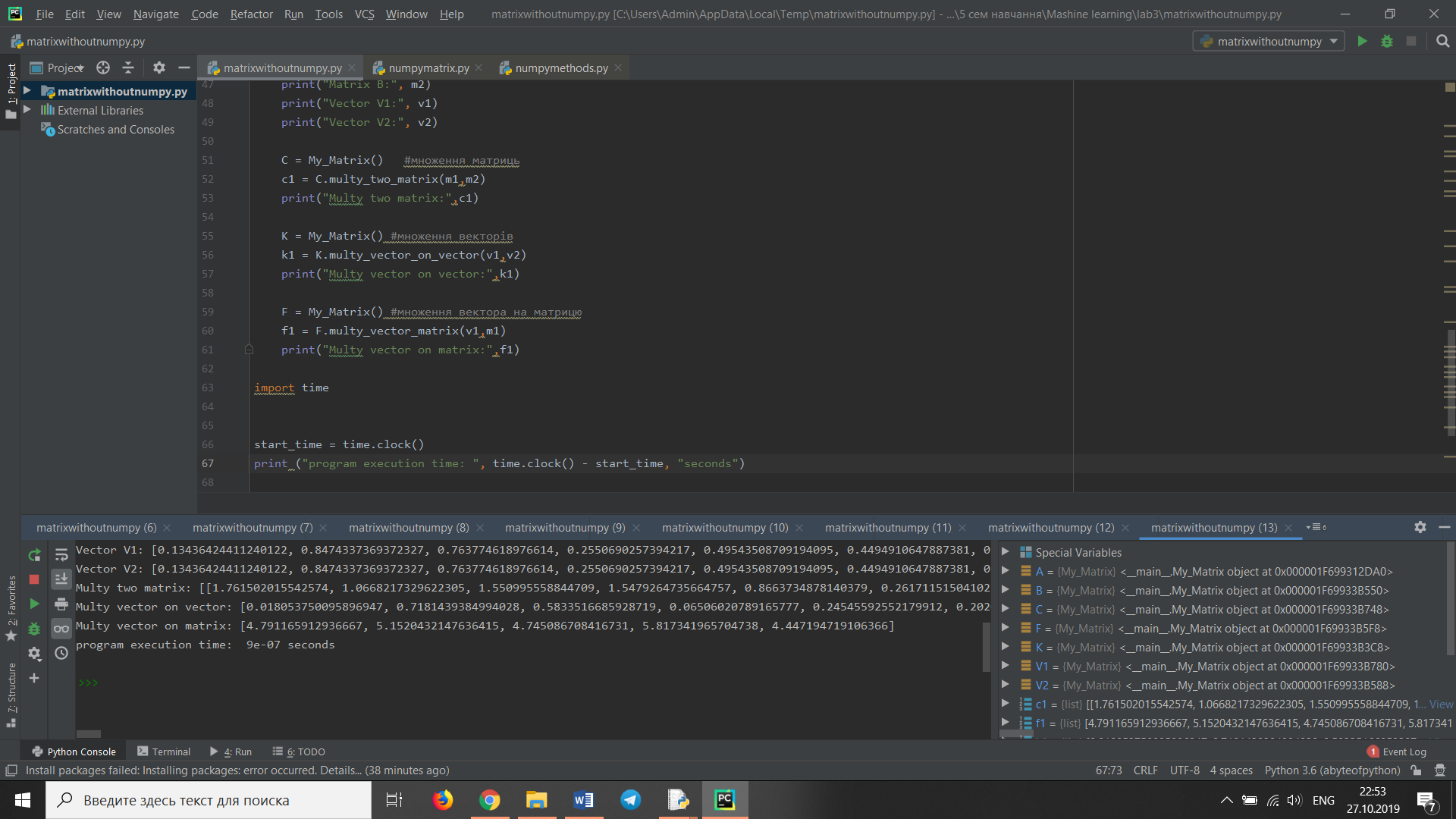
Добавили його у кінці програми , яка була написана з модулем numpy , та без цього модуля.

Файл *numpymatrix.py* , показав такі дані:



6е-07 = 0,0000006 сек.

Файл *matrixwithoutnumpy.py* , показав такі дані:



9е-07 = 0,0000009 сек.

Оскільки 0,0000009 > 0,0000006 , то це ще один плюс модуля numpy, завдяки ньому , можна зекономити час майже у двічі.

**Висновок:** виконавши дану лабораторну роботу, ми проаналізували особливості бібліотеки Numpy на основі реалізованих методів в порівнянні Python vs Python + NumPy.