Workshop 11: Working with Vectors and Matrices

Beth Wingate

January 15, 2021

The purpose of this workshop is to introduce you to working with vectors and matrices in Python such as the dot product, matrix-vector multiplication, and matrix-matrix multiplication. By the end of this workshop you will have experience programming your own functions to perform matrix operations as well as be able to perform matrix operations using Python's modules such as `numpy.dot`. As a part of this experience you will develop your own unit tests to check that your understanding of Python's modules matches your own mathematical understanding.

Programming Vector and Matrix operations

In this section we look at the construction of matrix and vector operations. To begin let's define a few common matrix operations.

We denote a vector with n components as $\mathbf{v} = [v_1, v_2, ..., v_n]$ where $v_i \in \mathbb{R}$ (this means each v_i is a real number!). Likewise, we define an $n \times m$ Matrix to be,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

so that a_{ij} is the ijth element of the matrix A.

The dot product

In mathematics last term you learned to compute the dot product of two vectors of length n by multiplying their elements together:

$$\mathbf{u} \cdot \mathbf{w} = \sum_{1}^{n} u_i \, w_i \tag{1}$$

An example of this is given to the right [1].

[1] If $\mathbf{u} = [1, 2, 3]$ and $\mathbf{w} = [1, 1/2, 1/3]$ then $\mathbf{u} \cdot \mathbf{w} = 3$.

The program for the example on the right looks like:

```
v = [1, 2, 3]       # Initialize v
w = [1, 1/2, 1/3]   # Initialize w
n = len(v)          # Rather than typing in n=3, we can compute the length
vdotw = 0.          # Intialize vdotw
for i in range(n): # This loop begins at 0 and ends at N-1
    vdotw += v[i] * w[i]
```

Program this into Python now[2]. Test it for another pair of vectors that you make up yourself.

[2] ⟵ This exercise is Question 1 of the Workshop.

Matrix-matrix multiply

In a similar way, the definition of multiplying a matrix **B** with dimensions $q \times n$ with matrix entries (b_{ij}) with matrix **A** which has dimensions $n \times m$ whose matrix entries are denoted (a_{ij}), then the product matrix $\mathbf{C} = \mathbf{BA}$ is a matrix of dimension $q \times m$ with entries defined by (c_{ij}) defined by the formula,

$$c_{ij} = b_{i1}a_{1j} + ... + b_{in}a_{nm} = \sum_{p=1}^{n} b_{ip}a_{pj}, \tag{2}$$

for $1 \leq i \leq q$, $1 \leq j \leq m$.

To convert the above formula into Python code you will need to use multi-dimensional arrays which we covered last term. To develop an intuition for how to use multi-dimensional arrays in Python, consider the code at right. The integer numbers in the left most column of the output are the row numbers – the number next to that is the column number which, in this loop, is referred to as the "fastest running index". Notice where i and j appear in the loop structure of the program. Notice that the inner loop, or 'fastest running index' is the inner loop of the program. Notice that to access the data for matrix element A_{ij} you use the Python syntax $A[i, j]$ with square brackets. In the workshop questions, below, you will be writing your own program to multiply two matrices together. You may be asked about matrix multiplication in Python on your May exam.

```
import numpy as np
m = 3
n = 2
A = np.random.rand(m, n)
for i in range(m):
    for j in range(n):
        print(i, j, A[i, j])
gives:
0 0 0.511616029658
0 1 0.104673988753
1 0 0.230826605125
1 1 0.562958145447
2 0 0.88264931162
2 1 0.939669591566
```

Using Python modules for vector and matrix operations

NUMPY CONTAINS FUNCTIONS THAT PERFORM MATRIX OPERATIONS. In this exercise we will explore the function `numpy.dot(A, B)`, where A and B are numpy arrays. If they are 1-dimension arrays `numpy.dot(A, B)` will return the dot product of two vectors as in Eq (1). If they are 2D arrays it will return the equivalent of matrix multiplication as in Eq (2). An example of how to use this function is:

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
C = np.dot(A, A)
print(C)
```

Try programming this example yourself now[3]. Compare the output of this program to the operation `D =A*A`. What is the difference between Python's computation of `C` versus `D`. You will need to know the difference between these two types of operations for your May exam.

[3] ⟵ This exercise is Question 2 of the Workshop.

ALWAYS TEST YOUR UNDERSTANDING OF MODULES USING UNIT TESTS. One of the advantages of participating in Python programming community is having access to the wealth of already-programmed and tested modules and functions, but care has to be taken that you understand the differences between Python's default processes and those of module functions, as well as how you might program something on your own. To make sure you are using module functions correctly use unit tests. You will explore this idea in the questions below.

WHAT IS A UNIT TEST? A unit test refers to a small 'unit' of code that is used to test larger blocks of code. For example, in Question 3. of this workshop you will write a program to compute the dot product of two vector. A good unit test would be to write a separate function that tests a well-known dot product, like two vectors composed only of ones. Consider the following code:

```python
import numpy as np

def KnownDotProduct(N):
    """ This unit test will test a dot product"""
    A = np.ones(N)
    ADotA = N
    return np.float(ADotA)

def MyDotUnitTest(KnownDotProductOfOnes, N):
    """ This unit test compares Numpy’s dot product to a known dot product
        It uses a user-defined known function of dot product of ones
        to compare to Numpy """

    ArrayOfOnes = np.ones(N)
    NumpyDotProduct = np.dot(ArrayOfOnes,ArrayOfOnes)

    if np.abs(KnownDotProductOfOnes(N) - NumpyDotProduct) > 1.e-8:
        print("There is an error!")
        print("Numpy Dot Product gives ",NumpyDotProduct)
        print("However, ny Unit Test gives ",KnownDotProductOfOnes(N))
    else:
        print("Numpy passes the unit test!")
        print("Numpy Dot Product gives ",NumpyDotProduct)
        print("My Unit Test gives ",KnownDotProductOfOnes(N))


N = 4

#
# Now try out the unit test!
#

MyDotUnitTest(KnownDotProduct, N)
```

In summary, Unit Tests are specific, small blocks of code that will return a known (correct) answer so that you can test a more general block of code, such as the code you will write in Question 3. that will be able to compute the dot product of *any* two vectors.

Working with complex numbers

THE COMPLEX NUMBER $j = \sqrt{-1}$ is represented by the letter j in Python. To represent a complex number in Python you use $z =$

$z_r + z_i j$, where z_r is the real part of z and z_i is the imaginary part of z. See an example at right[4]. Complex arithmetic also works for `numpy`'s matrix and vector classes.

[4] if $z_1 = 1 + 1j$ and $z_2 = 1 - 1j$, then $z_1 \times z_1 = 2j$ and $z_1 \times z_2 = 2$.

If you have a complex variable, `z`, in Python, to access its real part you would use `z.real` and `z.imag`. The following is an example of defining a complex number in Python. Notice that the complex number j can be accessed by the Python symbol `1j`.

```
import numpy as np
v1 = 4.+2*1j             # Initialize v1
v2 = complex(1,1)        # Initialize v2
v3 = np.ones(3,dtype='complex128')  # Define a complex numpy array
v3[0] = 1.+3.j                      # Redefine v3[0]
v3[1] = 2.+62j                      # Redefine v3[1]
print(v1, v1.real)                  # print v1 and its real part
print(v2, v2.imag)                  # print v2 and its imaginary part
print(v3)                           # print the array v3
print("j*j = ", 1j*1j)              # test that 1j*1j = -1
print("complex conjugate example ",v3[0]*v3[0].conj())
```

Questions

1. Write a program to compute the dot product as described in the workshop (look in the margin for Question 1).

2. Write a program to compare Numpy's function `np.dot(A,A)` to the direct computation of A*A. What is the difference? This problem can be found in the workshop text above (look in the margin for Question 2).

3. Write a Python function to compute the dot product of two vectors[5].

   - Your function should allow for two input vectors: **u** & **w**.
   - Your function should return of single number, the result of taking the dot product of the two vectors. You should use the formula in Equation (1) and should use a loop.
   - Your function should use conditionals to test whether the length of vector **u** is the same length as the vector **w**.
   - Your program should use a unit test (a separate function) to make sure your function works. **Hint:** Consider using vectors whose elements are all zero except for one.

   Use your function to compute the following dot products:

   (a) u = [1., 1.], w = [2., 2.]

   (b) u = [2., 4., 6., 8.], w = [1.2, 3.4, 6.8, 4.2]

   (c) u = [1., 1., 1., 1., 1., 1.], w = [1.5, 2.3, 4.4, 5.6, 1.2, .01]

   (d) u = [1., 1.], w = [2., 2., 2.]

[5] **Answers for Q3:**
(a) $\mathbf{u} \cdot \mathbf{w} = 4.0$
(b) $\mathbf{u} \cdot \mathbf{w} = 90.4$
(c) $\mathbf{u} \cdot \mathbf{w} = 15.01$
(d) undefined

4. Edit the program of Question 3 to compute the same dot products, but rather than use your own function, use the Python function `numpy.dot(a, b)`. Make sure it works like you think it should by comparing the difference in values between the function you wrote and `numpy.dot(u, w)` and printing out the error.

5. Write a Python function that multiplies two matrices together using the formula of Equation (2). [6]

   - Your function should have two inputs, a matrix **A** and matrix **B**.
   - Your function should return a new matrix $\mathbf{C} = \mathbf{BA}$.
   - Your function should use nested loops to compute the multiplication of two matrices.
   - You should debug and test your function using a unit test. **Hint:** Consider using the identity matrix.

   Using your function multiply the following matrices:

   (a)

$$\mathbf{A} = \begin{bmatrix} 1. & 2. \\ 2. & 4. \end{bmatrix} \mathbf{B} = \begin{bmatrix} 16. & 52. \\ 1. & 7. \end{bmatrix}$$

   (b)

$$\mathbf{A} = \begin{bmatrix} 6. & 5. & 4. \\ 2. & 4. & 13. \end{bmatrix} \mathbf{B} = \begin{bmatrix} 16. & 52. \\ 1. & 1. \\ 1. & 0. \end{bmatrix}$$

6. Modify your program to use `numpy.dot` to compute the matrix multiplications above.

7. Recalling the polar form of complex numbers, write a Python function that will accept any complex number and return its polar form. Use your function to compute the polar form of each complex number, below, in terms of the vector's length and the angle made with the positive real axis. Recall that given the rectangular form of a complex number $z = z_r + z_i j$, the length of the vector is $r^2 = z_r^2 + z_i^2$ and the angle θ made with the vector and the real axis is θ, which is defined by the trigonometic rules,

$$\cos\theta = \frac{z_r}{r} \qquad \sin\theta = \frac{z_i}{r}. \tag{3}$$

   Make sure to check which quadrant the number is in. If $-\pi < \theta < \pi$, make sure the angle, in radians, has the correct sign. You will need to use conditionals[7]

   (a) z = 1 + 2j

   (b) z = 1 - 2j

   (c) z = -45 + 66j

   (d) z = -60 - 60j

[6] **Answers for Q4:**
(a)

$$\mathbf{BA} = \begin{bmatrix} 120. & 240. \\ 15. & 30. \end{bmatrix}$$

(b)

$$\mathbf{BA} = \begin{bmatrix} 200. & 288. & 740. \\ 8. & 9. & 17. \\ 6. & 5. & 4. \end{bmatrix}$$

[7] **Answers for Q7 in the format** $r, theta$**:**
(a) 2.23606797775 1.10714871779
(b) 2.23606797775 -1.10714871779
(c) 79.8811617342 2.16921522027
(d) 84.8528137424 -2.35619449019

8. Write a Python program to compute the dot product of the following complex vectors (Hint: consider using the program you used in Q (3).:[8]

   (a) v = [.2 +.2j, 4 + .53j], w= [1.+0j, 1.+0j]

   (b) v = [1+1j, 1-1j, 2+4j, .4 + .5j], w = [1.+0j, 1.+0j, 1.+0j, 1.+.1j]

   (c) v = [1j, 2j, 3j, 4j], w = [2j, 3j, 4j, 5j]

[8] **Answers for Q8:**
(a) 4.2+0.73j
(b) 4.35+4.54j
(c) -40+0j