

# Random Exercise - symptoms vector

## Approach

My approach to this problem will be to attempt this independently and then watch the video and see how my approach/solution is different from the lecturers, annotating my code etc.

Problem Statement: Symptoms vector. A 20-vector "s" records whether each of 20 different symptoms is present in a medical patient, with "s\_i" = 1 meaning the patient has the symptom and "s\_i" = 0 meaning she does not. Expressing the following using vector notation.

- a) The total number of symptoms the patient has.
- b) The patient exhibits five out of the first ten symptoms

## My Attempt

### Part A

In [56]:

```
'''
The total number of symptoms the patient has is equivalent to the sum of each element of
each element of the vector is a boolean value where 0 = false (false meaning the patient
and each true = 1

total number of symptoms in patient symptom vector s = s_1 + s_2 + ... + s_20
written

sum of s_i where i = 1 to i = 20 and s is a vector of size 20

in python sum(s) could be a useful shorthand or maybe its s.sum(), I'm not sure what met
on lists to support this and will need to investigate

the list data structure is a good proxy for a 1D vector, and I will use that here. To si
'''
#I thought the library was rand, it's actually random when I looked at the documentation
#import rand
from random import random

def total_number_symptoms(s):
    print(f'total number of patient symptoms (sum of s_i for i=1...20): {sum(s)}')
    return sum(s)

def generate_patient_symptom_vector():
    #I'll need some form of randomness, so will probably use the python native rand libr
    #I'll also need to populate each cell, range generates a range of integer values fro
    #of the vector
    s = []
    for i in range(20):
        #append is the function needed to add elements to the end of a list. random gene
        #number between 0 and 1. round will round to the nearest number so x > 0.5 -> 1
        s.append(round(random()))
    print(f'symptom vector "s": {s}')
    return s

#run and print output where generated vector is input to total_number_symptoms function
total_number_symptoms(generate_patient_symptom_vector())
```

symptom vector "s": [0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1]

```
total number of patient symptoms (sum of s_i for i=1...20): 10
Out[56]: 10
```

```
In [16]: #testing that sum(list) is short hand for sum each element in the list
sum([1,2,3])
```

```
Out[16]: 6
```

```
In [17]: #testing that random.random() will generate a numer between 0 and 1 as described in pyth
random()
```

```
Out[17]: 0.33797746000041395
```

## Part B

```
In [31]: '''
The wording on Part B is somewhat ambiguous, it's not clear to me if it's asking that we
exhibits 5 of 10 first symptoms, or to create a vector where the patient exhibits five o

If I'm going to express in vector notation. Then a subvector of s, we will call v, that
and 0 < i <= 10, and a patient that exhibits 5 of 10 first symptoms is one where:

(sum of v_i where i = 1 and n = 10) >= 5

'''
def five_symptom_patient(patient):
    print(f"subvector v: {patient[:10]}")
    if sum(patient[:10]) >= 5:
        print("Patient has 5 of first 10 symptoms")
        return True
    else:
        print("Patient does not have 5 of first 10 symptoms")
        return False
patient = generate_patient_symptom_vector()
total_symptoms = total_number_symptoms(patient)
five_symptom_patient(patient)

symptom vector "s": [0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1]
total number of patient symptoms (sum of s_i for i=1...20): 10
vector v: [0, 0, 0, 1, 0, 0, 0, 1, 1, 1]
Patient does not have 5 of first 10 symptoms
False
Out[31]:
```

## Reflection after Reviewing instructor solution

The response to the question should have been expressed in vector notation, and while I did express it in series summation notation, and was able to implement useful functions around that, it was not aligned with the initial expectations of the question.

A series summation can also be expressed as the inner product of an n-vector and a 1-vector of the same size, which is how the lecturer described it in the solution video. The correct answer was  $s^T \mathbf{1}$  where the inner product of a vector and a 1's vector of the same size will yield a single value equal to the sum of each element of the vector multiplied by 1.

Similarly, for the second part of the question, after reviewing the lecture, I think that I did not explicitly express the solution in vector notation. The correct answer is  $s_{1:10} = 5$ . Also, I created a new vector  $v$  to represent the subvector of  $s_{1:10}$ . I also misunderstood the statement and used the  $\geq$  operation

indicating that any value over 5 for the total number of symptoms in  $s_{1:10}$  was an acceptable answer. In actuality, only a strict equality of ' $= 5$ ' is acceptable.

## 1.8 Problem

### My Approach

I am going to attempt this independently and then refer to the solution afterwards to see where I was wrong

### Problem Statement

- Company sells  $n$  different products or items.
- vector  $p$  gives the profit in dollars per unit for each of the  $n$  items. (negative entries indicate loss leaders)
- vector  $s$  gives total sales of each item sold over some period

Express the total profit in terms of  $p$  and  $s$  using vector notation

### My Attempt

The contribution to total profit of a single item can be represented as: total profit of item = (profit per unit) \* (# of units sold within the period)

or  $x_i = p_i * s_i$  where  $x$  represents total profit,  $i$  is the index of the element in the vector, and vectors  $p$  and  $s$  are the same size.

The inner product is defined as the sum of the products of the corresponding entries.

Thus, by the definition of inner product, the total profit  $x$  can be described as the inner product of the vector of profits per unit and  $s$  is the vector of sales  $x = p^T s$ .

$$\sum_{i=1}^n p_i * s_i == p^T s$$

In [ ]:

## 1.16 Problem

Inner product of nonnegative vectors. All entries must be nonnegative. a) Explain why the inner product of two nonnegative vectors is nonnegative? b) Suppose the inner product of two  $n$  vectors is zero. What can you say about them? Your answer should be in terms of their respective sparsity patterns. i.e., which entries are zero and nonzero.

### My attempt

a) Proof by contradiction:

Hypothesis: the inner product of two nonnegative vectors is negative.

$$n_1^T n_2 < 0 \text{ where } \forall_i (n_{1i} \in n_1 | n_{1i} \geq 0) \text{ and } \forall_j (n_{2j} \in n_2 | n_{2j} \geq 0)$$

**Proof:**

$$\text{given: } \forall_i (n_{1i} \in n_1 | n_{1i} \geq 0) \text{ and } \forall_j (n_{2j} \in n_2 | n_{2j} \geq 0)$$

by definition of inner product:

$$n_1^T n_2 = n_{11}n_{21} + n_{12}n_{22} + \dots + n_{1q}n_{2r}$$

because  $n_{1i}$  and  $n_{2i}$  are both positive numbers, then by the property of real numbers, the sum and product of two positive numbers is also a positive number.

**Contradiction:**

The sum of the series  $\sum_{i=1}^n n_{1i}n_{2j}$  will always be a positive number given every element in the each vector is a positive number, therefore the inner product of two nonnegative vectors will always be nonnegative.

QED

b) If the inner product of two  $nn$  vectors is zero then the product of each corresponding entry must equal 0. This means that either:

- 1.) both vectors are 0 vectors
- 2.) one vector is a 0 vector
- 3.) one or both elements in each of the corresponding entries is equal to 0

$$\forall_{(x,y)} (x = 0 \vee y = 0)$$

where  $x$  and  $y$  are corresponding entries in equal size vectors.

## CODE BLOCKS

### Block 1

Notes: Anything in square brackets is automatically interpreted by python as a `list` which is a data structure similar to an array. This is appropriate for representing a vector which is an ordered finite list of numbers [page 3]. We could also represent a vector as a tuple, of the same size using parentheses, however, lists and tuples have different methods and properties.

```
In [1]: x = [-1.1, 0.0, 3.6, -7.2]
        len(x)
```

```
Out[1]: 4
```

Additional example:  $v$  is a vector of boolean values represented as 0's for False and 1's for True.

$$v = (0, 1, 0, 1)$$

```
In [16]: v = [0, 1, 0, 1]
# or
v = (0, 1, 0, 1)
```

## Block 2

Notes: Numpy is not a native library and must be imported. We can pass a list as a parameter to the array method on the numpy class imported from the numpy module.

Numpy array's support linear algebra operations like vector addition and scalar multiplication [page 15]

```
In [22]: import numpy as np
x = np.array([-1.1, 0.0, 3.6, -7.2])
len(x)
```

```
Out[22]: 4
```

Additional Example: If I multiply a python list by a scalar, 4, it just concatenates the list to itself 4 times. However, if I multiply a numpy array by a scalar, it correctly performs the linear algebra operation of scaling each element by 4.

```
In [28]: y = [-1.1, 0.0, 3.6, -7.2]
print (f'wrong: {y * 4}')
print (f'right: {4 * x}')
```

```
wrong: [-1.1, 0.0, 3.6, -7.2, -1.1, 0.0, 3.6, -7.2, -1.1, 0.0, 3.6, -7.2, -1.1, 0.0, 3.6, -7.2]
right: [ -4.4    0.    14.4 -28.8]
```

## BLOCK 3

Notes: You can index into an np array with the  $x[i]$  shorthand, where  $i$  is equal to the index of an element in an  $n$ -vector array, and the expression returns the element in the list at  $i$ . However, unlike mathematical notation in the book, which is 1-indexed, the indexing here will be 0-indexed [page 5]

```
In [30]: x = np.array([-1.1, 0.0, 3.6, -7.2])
x[2]
```

```
Out[30]: 3.6
```

Additional Example:

the  $n$ -vector above is of size  $n=4$ . Therefore in mathematical notation  $x_4 = -7.2$  however, as we can see below, the np array is 0 indexed and  $x[4]$  returns an index error

```
In [32]: x[4]
```

```
-----
IndexError                                Traceback (most recent call last)
Cell In[32], line 1
----> 1 x[4]

IndexError: index 4 is out of bounds for axis 0 with size 4
```

# BLOCK 4

Notes:

Array indexing on the left-hand side of an assignment statement to change the value of the corresponding element is useful python shorthand, however it is not a valid mathematical expression. After the below code is run, there is effectively a new vector that has been generated  $x_2$ , although the python program just assigns the vector to the original variable name  $x$

```
In [4]: x[2] = 20.0  
print(x)
```

```
[-1.1  0.  20. -7.2]
```

Additional example: We can assign any valid vector index a new value

```
In [35]: x[3] = 1  
print(x)
```

```
[-1.1  0.   3.6  1. ]
```

## Block 5

Notes: Negative indexing is not a valid operation in mathematics, but is useful shorthand for python.

The book says the following about integer indexes: "If we denote an  $n$ -vector using the symbol  $a$ , the  $i$ th element of the vector  $a$  is denoted  $a_i$ , where the subscript  $i$  is an integer index that runs *from 1 to  $n$* , the size of the vector." [page 3]

```
In [37]: x[-1]
```

```
Out[37]: 1.0
```

Additional Example: Negative indexing can also return second from last element in the vector, third from last, etc.

```
In [39]: x[-2]
```

```
Out[39]: 3.6
```

## Block 6

Notes: If we are going to change a vector, it's better convention to copy the vector into a new variable. Whenever we update an array directly the original data is mutated or destroyed. Additionally if we don't copy and assign a new variable to the value of an existing variable, a change to either variable would mutate both, potentially introducing unintended bugs.

```
In [42]: x = np.array([-1.1, 0.0, 3.6, -7.2])  
y = x.copy()  
x[2] = 20.0  
print(y)
```

```
[-1.1  0.   3.6 -7.2]
```

Additional Example: If we mutate  $y$ , it will not have any impact on the value of  $x$ , because  $y$  is an

independent array from x

```
In [45]: print(f'x {x}')  
y[3] = 1  
print(f'y {y}')
```

```
x [-1.1  0.  20. -7.2]  
y [-1.1  0.   3.6  1. ]
```

## BLOCK 7

Notes: "Two vectors a and b are equal, which we denote  $a = b$ , if they have the same size, and each of the corresponding entries is the same. If a and b are n-vectors, then  $a = b$  means  $a_1 = b_1, \dots, a_n = b_n$ ." [page 3]

We can use the python `==` equality operator that will return a boolean vector x of size n, where  $x_i$  is true if vector  $a_i = b_i$ , and false otherwise

```
In [48]: x = np.array([-1.1, 0.0, 3.6, -7.2])  
y = x.copy()  
x == y
```

```
Out[48]: array([ True,  True,  True,  True])
```

Additional example: If we recreate the example above, but create an entirely new vector instead of copying, with the same values for each corresponding entry, then we get the same output as expected

```
In [58]: y = np.array([-1.1, 0.0, 3.6, -7.2])  
y == x
```

```
Out[58]: array([ True,  True,  True,  True])
```

## Block 8

Notes: "If  $a_1 \dots a_m$  are n-vectors, and  $\beta_1 \dots \beta_m$  are scalars, the n-vector  $\beta_1 a_1 + \dots + \beta_m a_m$  is called a linear combination of the vectors  $a_1 \dots a_n$ . The scalars  $\beta_1 \dots \beta_m$  are called the coefficients of the linear combination." [page 17]

We can see that the numpy module allows for us to express linear combinations in syntax that is similar to the formal mathematical notation.

```
In [59]: import numpy as np  
a = np.array([1,2])  
b = np.array([3,4])  
alpha = -0.5  
beta = 1.5  
c = alpha*a + beta*b  
print(c)
```

```
[4. 5.]
```

Additional Example: If we didn't use numpy, but just lists, we can see that we would not be able to use similar syntax to express the linear combination. We get a type error that a list cannot be multiplied by a scalar.

```
In [60]: a = [1, 2]
b = [3, 4]
alpha = -0.5
beta = 1.5
print (alpha * a + beta * b)
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[60], line 5
      3 alpha = -0.5
      4 beta = 1.5
----> 5 print (alpha * a + beta * b)

TypeError: can't multiply sequence by non-int of type 'float'
```

## Block 9

Notes:

Another difference between python and mathematical notation of vectors is that a scalar is just a vector of size 1. [page 3] So where  $x = 2.4$  and  $x$  is a scalar and  $y = [2.4]$  is a vector of size 1.  $x$  and  $y$  are equivalent. However, as we can see below, when implemented in python the two are not equivalent, as one is of type list and the other is of type float/int

```
In [62]: x = 2.4
y = [2.4]
x == y
```

Out[62]: False

Additional Example: We can extract the value of the list of size 1 and show that the two are equivalent.  $y[0]$  returns an individual element which we is of the same type as  $x$  was originally intialized as.

```
In [63]: y[0] == x
```

Out[63]: True

## Block 10

Notes: To find the inner product of two vectors, we must use a method on the np module, and pass as arguments to the `inner` function two numpy arrays.

The inner product is defined as:  $a^T b = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$

```
In [64]: x = np.array([-1, 2, 2])
y = np.array([1, 0, -3])
print(np.inner(x, y))
```

-7

Additional example: To implement without the `inner` method of np module I could define as follows

```
In [69]: def inner_product(x, y):
# we assume that x and y are of the same size
# for this example
# result is the accumulator variable
    result = 0
```



```
    for i in range(len(x)):
        #we index into corresponding elements in both lists
        # we add their product to the result accumulator
        result += x[i] * y[i]

    return result
x = [-1, 2, 2]
y = [1, 0, -3]
inner_product(x, y)
```

Out[69]: -7

In [ ]: