# Exercise 1 – Marking Sheet. Marker: AM

## Student: S1969574

## Marks

|  |  |  |
| --- | --- | --- |
| **Module vector.py [6]** | Vector operations on Python lists [5]   * Magnitude and Magnitude-squared methods * Scalar multiplication and division methods * Vector addition, subtraction, dot product, cross product | 5 |
| HTML docs present; comments concise & sufficient [1] | 1 |
| **Tester module [2]** | * Random setup of three vectors * Print the vectors, magnitudes, their sum, dot product, and cross product * Checks of vector identities * Vector equality test | 1 |
| **Numpy Tester [2]** | * Random creation of three (1,3) Numpy arrays * Printouts as above * Checks of vector identities | 2 |
| **PBC & MIC [4]** | * PBC: nearest neighbour cells? Arbitrary cell? * MIC: nearest neighbour? Arbitrary cell? | 2 |
| **Style [6]** | Code layout, comments, naming conventions clear and helpful | 3 |
| **Total [20]** |  | 14 |

## Feedback:

**Vector:**All functions here implemented correctly, nice work here. HTML documentation is nice and is appropriate.

**Vector tester:**Your code to test your vector functions is clear and well written.   
Unfortunately you’ve got a slight error in the way that you check for equality between two vectors. This is because if your first if-statement if False (i.e. abs(v1[0] – v2[0]) > thresh) then your code doesn’t return anything when it should return False. Also, your third if-statement should be testing the third coordinate, not second. It’s also better practice to return raw Boolean variables (i.e. True or False), rather than as strings.

**Numpy tester:**Generally, good use of Numpy functions here. An easier way to add two arrays together is to simply use the addition operator “+” instead of calling “np.add” every time (e.g. “vec\_1 + vec\_2” is the same as “np.add(vec\_1, vec\_2)” and is clearer). Also note that your way of comparing if two vectors are equal is incorrect, as in the previous part. Now that we’re using Numpy arrays, we can use the Numpy function “np.allclose” which tests if all elements in the provided arrays are within a tolerance of eachother.Note that there’s an easier way to create an array of random values with a specific shape, as instead of calling random.random() in a list, we can instead call np.random.rand(3) with length 3 here.

**PBC & MIC:**Your code for computing the PBC is generally correct. It’s a bit convoluted to set the values of “x\_image” as you create an empty array of values initially “x\_image = np.array([])” but then overwrite it later using   
“x\_image = np.mod(x,l)” – hence you do not need the initial initialisation. Good use of “np.mod” here, as this allows for arbitrary sized coordinate arrays and box sizes.   
Only issue here is that you explicitly cast the cube length to an integer, whereas it can be a float (i.e. l=0.5 is perfectly fine) and so providing it with a float causes your code to fail. This error is repeated in your MIC code.

**Style:**Generally good code with clear variable names. In “vector.py” you’ve got an inconsistent number of new lines after each function, it’s generally a good idea to have two blank lines after each function.  
You also have an extra indentation layer when checking for equality between two vectors for some reason.It is generally bad practice to round numbers when computing quantities (as done in your “cross\_product” function) as we usually want to keep as much precision as possible in our numerical methods. Try to avoid rounding at all cost!