# Exercise 2
# Numpy, Pydoc, Vector Operations

Computer Modelling 2017-18

Due: 5pm Thursday, Week 7, Semester 1

## 1 Aims

In this exercise, you will get to know a ubiquitous Python package, `NumPy`, which adds array functionality to Python. You will use this to implement your own methods to manipulate 3D vectors, test those methods to verify well-known vector calculus identities, and make use of the `pydoc` tool to create your own HTML documentation. This (as all others) is a group exercise, and submissions from individuals will not be accepted.

## 2 Tasks

### 2.1 Vector manipulation with lists

In a new python module, `vector.py`, collate functions that operate on 3D vectors that are represented as Python lists $[v_x, v_y, v_z]$. Write methods that return

- The squared magnitude and magnitude of a vector, respectively.
- The vector obtained by multiplying or dividing a vector by a scalar.
- The vector sum of two vectors.
- The vector difference of two vectors.
- The cross product of two vectors.
- The dot product of two vectors.

Use sufficient comments for all functions that `pydoc` will create a satisfactory documentation; this includes descriptions of each method, what their arguments are, and what they return.

## 2.2 Tester Program

Write a Python program `vector_tester.py` that imports `vector.py` and:

- randomly creates three vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}_3$, and stores those as Python lists;
- prints to the screen the three vectors, their magnitudes, as well as the vector sum $\boldsymbol{v}_1 + \boldsymbol{v}_2$, the dot product $\boldsymbol{v}_1 \cdot \boldsymbol{v}_2$, and the cross product $\boldsymbol{v}_1 \times \boldsymbol{v}_2$;
- tests the following vector identities by independently calculating their left and right hand sides:

$$\boldsymbol{v}_1 \times \boldsymbol{v}_2 = -\boldsymbol{v}_2 \times \boldsymbol{v}_1 \quad (1)$$

$$\boldsymbol{v}_1 \times (\boldsymbol{v}_2 + \boldsymbol{v}_3) = (\boldsymbol{v}_1 \times \boldsymbol{v}_2) + (\boldsymbol{v}_1 \times \boldsymbol{v}_3) \quad (2)$$

$$\boldsymbol{v}_1 \times (\boldsymbol{v}_2 \times \boldsymbol{v}_3) = (\boldsymbol{v}_1 \cdot \boldsymbol{v}_3)\boldsymbol{v}_2 - (\boldsymbol{v}_1 \cdot \boldsymbol{v}_2)\boldsymbol{v}_3 \quad (3)$$

## 2.3 Vector manipulation with `NumPy`

Rewrite the tester program above as `numpy_tester.py` such that vectors are instead represented as (1,3) `NumPy` arrays. You should not use the `vector.py` module for this program. The program should randomly create three vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}_3$, and store them as `NumPy` arrays. Then, the same tasks should be fulfilled as in the previous subsection, but using only intrinsic `NumPy` functions.

# 3 Detailed Instructions

You may want to base your code on the example provided for complex numbers, `complex.py` and `complex_tester.py`. The instructions below are a suggestion of how to gradually build your code based on those examples. You do not have to follow this suggested route, but beware that TA's and lecturers can best help you with issues and debugging if you do not stray too far from it.

## 3.1 Starting Point: Complex Numbers Example

You should refresh your knowledge with the sections of the course notes that discuss functions and methods before attempting this checkpoint.

Download the two Python files `complex.py`, and `complex_tester.py` from LEARN. Examine the files and make sure you understand what is going on:

- `complex.py` defines functions to manipulate complex variables. These variables are stored in Python lists.
- `complex_tester.py` contains an example test code that imports and uses the functions defined in `complex.py`, and applies them to two randomly created complex numbers.

Run the example code by typing

```
[user@cplab001 ~]$ python complex_tester.py
```

You should be able to explain why the program produces the output it does. Produce the HTML documentation for the Python source code files by running

```
[user@cplab001 ~]$ pydoc -w complex
```

Open the resulting file `complex.html` in a web browser and verify that it contains properly formatted documentation for all functions defined in `complex.py`, derived from the triply-quoted Python comments `"""` ... `"""`. You can also look up this documentation on the command line, by typing

```
[user@cplab001 ~]$ pydoc complex
```

## 3.2 Vector Operations

You should base `vector.py` on `complex.py`. Some functions will need little change, others need to take into account that each vector has three components, and yet others (such as the cross product) will have to be written completely now, as they don't have an equivalent amongst complex numbers.

Remember to add sufficient comments such that `pydoc` can create a decent documentation from this file.

## 3.3 Vector Testing

In similar vein to `complex_tester.py`, you should write a program that can test the functions of `vector.py`. For verification of vector identities, think about how you break up the respective mathematical statements to produce a paper algorithm that computes the two sides of the identities and compares the results to check if they are equivalent.

Add sufficient comments at the beginning of the tester program to explain its purpose and usage. You do not need to create HTML documentation for your tester program, though you might find it a useful exercise to do so.

## 3.4 Numpy Testing

You should write a program `numpy_tester.py` that performs the same tasks as the previous implementation, but uses `NumPy` arrays instead of Python lists to store 3D vectors. `NumPy` is an external Python module that provides efficient functionality to work with multidimensional array data structures. It can be used straightforward as

```
import numpy as np
vector = np.array([1, 0, -2.5])
matrix = np.array([1, 0], [0, -1])
```

Example `NumPy` functions that you might find useful include

- `np.linalg.norm(a1)`: returns the norm of an array `a1`
- `np.inner(v1, v2)`: returns the inner (dot) product of vectors `v1`, `v2`
- `np.cross(v1,v2)`: returns the cross product of vectors `v1`, `v2`

Note that `NumPy` arrays of the same dimension can be manipulated element-wise using the standard arithmetic operators. Hence,

```
vector1 = np.array([-1, 3, 0])
vector2 = np.array([2, 1, 5])
vsum= vector1 + vector2
```

will result in `vsum = [1, 4, 5]`. Make sure to exploit this so-called operator overloading when writing your tester program.

# 4 Submission

Package the subdirectory that contains `vector.py` the HTML documentation created by `pydoc`, and your tester programs. For instance, if your subdirectory is called `exercise-2`, you can use the `tar` and `gzip` commands by running:

```
[user@cplab001 ~]$ tar cvf ex2.tar exercise-2
[user@cplab001 ~]$ gzip ex2.tar
```

One of your group members should submit the compressed package (e.g., `ex2.tar.gz`) through the course LEARN page, by **5pm on Thursday, week 7 of semester 1**. Make sure the members of your group are stated in the Python files or during the submission to LEARN. Submissions from individuals will not be accepted.

# 5 Marking Scheme

This assignment counts for 10% of your total course mark.

1. Module `vector.py` [7]
   - Has required functionality [5]
   - HTML documentation is present, and comments are concise, sufficient [2]
2. Tester program `vector_tester.py` [5]
3. Tester program `numpy_tester.py` [5]
4. Code layouts, comments, naming conventions, etc. [3]

Total: 20 marks.