

## 4 — PHY 432: Classroom Activity (30 points total)

For a refresher on using **NumPy** work through the [NumPy tutorial](#).<sup>1</sup> Do the examples while you read it.

“BONUS: ” problems are tested/autograded but you do not need to solve the problem to get the maximum number of points.

### 4.1 NumPy arrays (12 points)

For the following, add your code to the file `problem1.py`. Given the three arrays

```
import numpy as np

sx = np.array([[0, 1], [1, 0]])
sy = np.array([[0, -1j], [1j, 0]])
sz = np.array([[1, 0], [0, -1]])
```

- (a) What is the result of `result1a = sx * sy * sz`? Explain what NumPy array multiplication does to the arrays. (Note: your code should assign the result to the variable `result1a` in `problem1.py`.) [2 points]
- (b) Use `np.dot()` to multiply the three arrays (like  $\sigma_x \cdot \sigma_y \cdot \sigma_z$ , where the dot “.” indicates a matrix product). Add your code to `problem1.py` and assign your result to variable `result1b`. What happened? [2 points]
- (c) Compute the “commutator”  $[\sigma_x, \sigma_y] := \sigma_x \cdot \sigma_y - \sigma_y \cdot \sigma_x$  and show that it equals  $2i\sigma_z$ .<sup>2</sup> Add your code to `problem1.py`, assign the result to variable `result1c`. [3 points]

---

<sup>1</sup>Some stuff such as the `ix_()` function is fairly esoteric for beginners but almost everything else is what you should be familiar with for your daily work with arrays.

<sup>2</sup>These are the Pauli matrices that describe the three components of the spin operator for a spin 1/2 particle,  $\hat{\mathbf{S}} = \frac{\hbar}{2}\boldsymbol{\sigma}$ . The fact that any two components of the spin operator do *not* commute is a fundamental aspect of quantum mechanics.

(d) BONUS: Given a “state vector”

$$\mathbf{v} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -i \end{pmatrix}$$

calculate the “expectation value”  $\mathbf{v}^\dagger \cdot \sigma_y \cdot \mathbf{v}$  (i.e., the multiplication of the hermitian conjugate of the vector,  $\mathbf{v}^\dagger$  with the matrix  $\sigma_y$  and the vector  $\mathbf{v}$  itself) using NumPy.<sup>3</sup> Add your code to `problem1.py` and assign your result to variable `result1d`.<sup>4</sup> [bonus +2\*]

## 4.2 Coordinate manipulation with NumPy (15 points)

We can represent the cartesian coordinates  $\mathbf{r}_i = (x_i, y_i, z_i)$  for four particles as a numpy array `positions`:

```
import numpy as np
positions = np.array(\
    [[0.0, 0.0, 0.0], [1.34234, 1.34234, 0.0], \
     [1.34234, 0.0, 1.34234], [0.0, 1.34234, 1.34234]])
t = np.array([1.34234, -1.34234, -1.34234])
```

and `t` will be a translation vector. *For the following use NumPy.* Add your code to file `problem2.py` and assign results to variables as indicated in the problems.

- (a) What is the *shape* of the array `positions` and what is its *dimension*?  
Assign results to variables `results2a_shape` and `results2a_dimensions`.  
[2 points]

---

<sup>3</sup>The *hermitian conjugate*  $\mathbf{v}^\dagger = (v_1^*, v_2^*)$  is `v.conjugate().T` where `v.T` is shorthand for `v.transpose()`. It turns out that you don’t need the transposition when you use `np.dot()` but I include it here for conceptual clarity. (Including `transpose()` comes at a minor performance penalty — check with `%timeit` if you are curious.)

<sup>4</sup>Note for anyone having done PHY 315 (Quantum Mechanics II) that here you are calculating the quantum mechanical expectation value of the  $y$ -component of a spin  $\frac{1}{2}$  particle in an eigenstate of the operator of the  $y$ -component of the spin ( $\sigma_y \mathbf{v} = -\mathbf{v}$ ) and because  $\mathbf{v}$  is normalized, you should get the eigenvalue as the expectation value.

- (b) What is the *shape* of the array `t` and what is its *dimension*? Assign results to variables `results2b_shape` and `results2b_dimensions`. [2 points]
- (c) How do you access the coordinates of the second particle in `positions`? Assign the result to variable `result2c`. [1 points]
- (d) For the second particle:
- (i) How do you access its *y*-coordinate? Assign the result to variable `result2d`. [1 points]
  - (ii) What type of object is this output, what is its *shape* and its *dimension*?<sup>5</sup>
- (e) How do you access the *y*-coordinates of *all* particles? Assign to variable `result2e`. [1 points]
- What is the *shape* (`result2e_shape`) and number of (`result2e_dimensions`) of the resulting array? [2 points] dimensions
- (f) Write Python code to translate all particles by a vector  $\mathbf{t} = (1.34234, -1.34234, -1.34234)$ ,  
`t = np.array([1.34234, -1.34234, -1.34234])`  
 Add your code to `problem2.py` and assign the translated coordinates to variable `result2f`. [3 points]
- (g) Make your solution of (f) a function `translate(coordinates, t)`, which translates all coordinates in the argument `coordinates` (an `np.array` of shape `(N, 3)`) by the translation vector in `t`. The function should return the translated coordinates as a numpy array.  
 Add the function to `problem2.py`. [3 points]
- Try out your function when applied to (1) the input `positions` and `t` from above and (2) for `positions2 = np.array([[1.5, -1.5, 3], [-1.5, -1.5, -3]])` and `t2 = np.array([-1.5, 1.5, 3])`.<sup>6</sup>

<sup>5</sup>Do not assign results to any variables but answer the question for yourself.

<sup>6</sup>Do not assign results to variables, just try out your `translate()` function on input where you can easily compute the *expected result*.

### 4.3 NumPy function plotting with matplotlib (8 points)

We want to plot the function <sup>7</sup>

$$\text{sinc}(x) := \frac{\sin(\pi x)}{\pi x} \quad (1)$$

over the range  $-6 \leq x < 6$ .

- (a) Use the NumPy `numpy.linspace()` function to generate an array `X` with 601 values from -6 to 6, including the upper boundary. [1 points]
- (b) Evaluate `sinc(x)` (use the NumPy `sinc()` function) for all  $x$  values in `X` <sup>8</sup> and assign it to a variable `Y`. [4 points]
- (c) Plot your function with matplotlib:
  - plot `X` vs `Y`
  - label the  $x$ -axis with “x” and the  $y$ -axis with “y = sinc(x)”
  - write the plot to a PNG file `sinc.png`.[3 points]

Submit your code as file `problem3.py` together with the plot in file `sinc.png`.

---

<sup>7</sup>This is the definition used in `numpy.sinc` function.

<sup>8</sup>You do *not* need any loops!