Problem Set 0

In this assignment, you will learn the basics of Scientific Python and Matplotlib.

- *Preliminaries*—Before doing this assignment, please go through the tutorial at ⟨`http://spike.spms.ntu.edu.sg/wiki/Scipy_tutorial`⟩.
- *Submission format*—Submit your solution via NTULearn, with one Python source file per problem (i.e., 2 files for this assignment). Each file should begin with the lines

```
from scipy import *
import matplotlib.pyplot as plt
```

  You may also import other standard Python libraries and Scipy libraries. But do *not* use the `matrix` module; all matrices should be handled as 2D arrays. Do not import the `numpy` module separately, since `scipy` is a superset of `numpy` (but note that the online documentation often refers to Numpy instead of Scipy).
- *Documentation*—Whenever the assignment mentions a Scipy function you might need, consult the Scipy/Numpy online documentation to learn what the function does and how to use it. (Use Google to search for "`scipy [name of function]`".)
- *Marks*—Some problems are labeled like this: (5* marks). For undergraduates, these problems are optional and can be attempted for a bonus of 1/5 the indicated marks. For graduate students, the problems are compulsory and receive full marks.
- *Grading*—For full marks, code must follow good programming style. Key code blocks must be clearly commented. Avoid cryptic variable and function names (like `abc`). The program structure should be modular; avoid unnecessary code duplication, and group numerical constant definitions neatly together.

  The output of your programs should be clear. Each generated plot should have x and y axis labels, and if multiple curves are shown, they should be well-labeled.

0. 2D INTERFERENCE PATTERNS

In this problem, you will write a program to visualize interference patterns formed by plane waves in 2D space. A single plane wave can be described by a "wavefunction" of the form

$$\psi(\vec{r}) = A \exp\left[i\vec{k}\cdot\vec{r}\right], \tag{0}$$

where $\vec{r} \equiv [x, y]$ is the position vector, $\vec{k} \equiv [k_x, k_y]$ is the wave-vector, and $A \in \mathbb{C}$ is a complex amplitude. A superposition of N plane waves is described by

$$\psi(\vec{r}) = \sum_{n=0}^{N-1} A_n \exp\left[i\vec{k}_n\cdot\vec{r}\right]. \tag{1}$$

(a) (3 marks) Write a function to compute the superposition of a set of plane waves with specified A_n and $\vec{k}_n$ parameters, at specified positions $\vec{r}$:

```
def wave_superposition(x, y, amplitudes, kvectors):
```

Inputs	
`x, y`	Arrays specifying x and y coordinates of the positions where we want to compute $\psi(\vec{r})$. These two arrays must have equal shape/size, but the exact shape/size is arbitrary (i.e., the caller can decide).
`amplitudes`	A 1D array of wave amplitudes. Each element of this array is a complex number, A_n, specifying the (complex) amplitude of the n-th wave.
`kvectors`	An array of wave-vectors, such that the value of `kvectors[n]` is an array of the form $[k_{x,n}, k_{y,n}]$, specifying the wave-vector of the n-th wave. Note: `kvectors` and `amplitudes` must have equal lengths.

Output	
`psi`	An array of complex numbers, specifying the wavefunction $\psi(\vec{r})$ at the positions specified by `x` and `y`. The shape/size of this array must be the same as the shape/size of `x` and `y`.

Note that there is no separate input for N, the number of waves in the superposition. N is specified implicitly, via the length of the `amplitudes` and `kvectors` arrays. The function should work properly whether `x` and `y` are 1D arrays, 2D arrays, or any other size.

Be sure to test the function for correctness. For instance, try a single plane wave with $A_0 = 1$, and $\vec{k}_0 = [1, -1]$, to be evaluated at two points, $\vec{r}_0 = [1, 2]$ and $\vec{r}_1 = [3, 4]$:

```
x, y = array([1.0, 3.0]), array([2.0, 4.0])

A = array([1.0])
k = array([[1., -1., 1.]])
psi = wave_superposition(x, y, A, k)
print(psi)
```

The above code should print `[ 0.54030231-0.84147098j 0.54030231-0.84147098j]`, which you can verify to be correct. You should also try more complicated tests.

(b) (3 marks) Write a function to plot the interference pattern for a superposition of waves:

```
def interference_xy_plot(xspan, yspan, amplitudes, kvectors):
```

Inputs	
`xspan`	A tuple of three numbers `(xmin, xmax, M)`; this specifies x-coordinates consisting of `M` numbers between `xmin` and `xmax` (inclusive).
`yspan`	A tuple of three numbers `(ymin, ymax, N)`; this specifies y-coordinates consisting of `N` numbers between `ymin` and `ymax` (inclusive).
`kvectors`	An array of wave-vectors, such that the value of `kvectors[n]` is an array of the form $[k_{x,n}, k_{y,n}]$, specifying the wave-vector of the n-th wave. Note: `kvectors` and `amplitudes` must have equal lengths.

This function has no return value, but it should generate two plots:

(i) An intensity plot showing $|\psi(\vec{r})|^2$ versus x and y.

(ii) A phase plot showing $\arg[\psi(\vec{r})]$ versus x and y.

These plots can be made using the `plt.pcolormesh` function (`pcolor` stands for "pseudo-color"). The x-y coordinates for `plt.pcolormesh` can be calculated by using `linspace` to generate 1D arrays of x and y coordinates, and then using `meshgrid` to generate two 2D arrays giving the x and y coordinates for the x-y grid. Be sure to label the plots clearly, and to make use of the `wave_superposition` function you wrote in (a)—don't duplicate code!

Note: the argument of a complex number can be extracted using the `angle` function.

(c) (2 marks) Write a function `interference_demo(k0=1.0)`, which takes no inputs and calls `wave_superposition` for the following set of three plane waves:

$$A_1 = A_2 = A_3 = 1, \quad \text{and} \quad \begin{cases} \vec{k}_1 &= [k_0,\ 0] \\ \vec{k}_2 &= [-k_0\cos(\pi/3), \quad k_0\sin(\pi/3)] \\ \vec{k}_3 &= [-k_0\cos(\pi/3),\ -k_0\sin(\pi/3)] \end{cases}$$

where k_0 is the input to the `interference_demo` function. Choose appropriate x and y coordinates so that the key features of the interference pattern are easily seen. In code comments, discuss the results.

(d) (*4 marks) Modify `interference_demo()`, from part(c), to overlay marker points on the intensity and phase plots, indicating the exact positions of the interference pattern's minimum-intensity points (nodes) and maximum-intensity points (antinodes). These node and antinode positions can be worked out analytically. In code comments, discuss the results.

1. RANDOM WALKS

A *random walk* is a mathematical model for a random path through space. Random walk models have many important applications for statistical mechanics and other areas of science.

Let $\vec{r}$ denote position coordinates in a continuous d-dimensional space. A "particle" begins at the origin, $\vec{r}_0 = 0$, and takes a discrete sequence of steps. Let $\vec{r}_n$ denote the position after the n-th step. An N-step "walk" is described by $\{\vec{r}_0, \vec{r}_1, \vec{r}_2, \dots, \vec{r}_N\}$. Each displacement

$$\vec{\ell}_n \equiv \vec{r}_n - \vec{r}_{n-1}$$

is an independent random variable chosen from some probability distribution.

(a) (2 marks) Write a function to plot a 2D random walk:

`def random_walk_plot(stepfun, nsteps):`	
Inputs	
`stepfun`	A function to be used for performing each step of the random walk. Called with no inputs, this function should return a real array $\vec{\ell}$, specifying a displacement vector.
`nsteps`	The number of steps, N, in the random walk.

This function has no return value. It should plot the path through x-y plane taken by the random walk of length N.

Note that the input `stepfun` is supposed to be a *function*. The caller is responsible for defining an appropriate function to generate displacement vectors, and supplying that function as an input to `random_walk_plot`. For example, the following code defines a step function where every step moves to the right by distance 1.0, and passes that function to `random_walk_plot`:

```
def step_right():
    return array([1.0, 0.0]) # Step by +1.0 in the x direction

random_walk_plot(step_right, 1000)
```

(b) (3 marks) We are now interested in *ensembles* of independent random walks, each consisting of N steps. From $\vec{r}_N$, we can compute the mean displacement and the mean squared deviation, which are defined as follows:

$$\begin{aligned} \langle \vec{r}_N \rangle &= \frac{1}{M} \sum_{M \text{ walks}} \vec{r}_N \\ \langle \Delta r_N^2 \rangle &= \left(\frac{1}{M} \sum_{M \text{ walks}} |\vec{r}_N|^2 \right) - |\langle \vec{r}_N \rangle|^2 . \end{aligned} \tag{2}$$

Write a function to determine the statistical properties of an ensemble of random walks:

`def random_walk_stats(stepfun, nsteps, nwalks):`	
Inputs	
`stepfun`	A function to be used for performing each step of the random walk. Called with no inputs, this function should return a real 1D array $\vec{\ell}$, specifying a displacement vector.
`nsteps`	The number of steps, M, in each random walk.
`nwalks`	The number of independent random walks to generate.
Return values	
`rmean`	The mean displacement, $\langle \vec{r}_N \rangle$. This is a real 1D array, of the same size as the displacement vectors returned by `stepfun`.
`msd`	The mean squared deviation $\langle \Delta r_N^2 \rangle$.

(c) (4 marks) Consider a 2D random walk where each displacement has the form

$$\vec{\ell}_n = \begin{pmatrix} \Delta x_n \\ \Delta y_n \end{pmatrix}, \quad \text{where} \quad \begin{cases} \Delta x_n & \sim \mathcal{N}(\mu_x, \sigma_x), \\ \Delta y_n & \sim \mathcal{N}(\mu_y, \sigma_y). \end{cases} \tag{3}$$

Here, $\sim$ denotes "drawn from", and $\mathcal{N}(\mu, \sigma)$ denotes the Gaussian distribution with mean μ and standard deviation σ. In Scipy, you can draw from a normal distribution using `random.standard_normal` or `scipy.stats.norm`.

Write a function to generate and plot the statistics of this random walk:

```
def random_walk_gaussian_demo(mux=0.1, muy=0.0, sigx=1.0, sigy=1.0):
```

Inputs	
`mux`, `muy`	The mean displacements in each direction, μ_x and μ_y.
`sigx`, `sigy`	The standard deviations for the displacements, σ_x and σ_y.

This function has no return value, but it should produce two plots:

(i) The absolute value of the mean displacement, $|\langle \vec{r}_N \rangle|$, versus N.

(ii) The mean squared deviation, $\langle \Delta r_N^2 \rangle$, versus N.

You should choose appropriate values for the walk length N and sample size M.

(d) (3 marks) Extend the `random_walk_gaussian_demo` function from part (c) as follows. First, use the numerical data to find the *best-fit* trend line for the mean squared deviation:

$$\langle \Delta r_N^2 \rangle \approx D N^\alpha \quad \Rightarrow \quad \log(\langle \Delta r_N^2 \rangle) \approx \alpha \log(N) + \log(D), \tag{4}$$

where D and α are constants to be determined. You can perform the linear fit of $\log(\langle \Delta r_N^2 \rangle)$ versus $\log(N)$ using the `polyfit` function. Then plot the resulting best-fit curves in the mean squared deviation plot generated by `random_walk_gaussian_demo`, overlaid on the numerical data. In the figure title or legend, print the fitted values of α and D.

(e) (*2 marks) Extend the `random_walk_gaussian_demo` function from parts (c) and (d), to also include the theoretical predictions for the plots. (If you don't know the theoretical predictions, look them up in a statistical mechanics textbook.)

The theoretical curves should be computed from the inputs `mux`, `muy`, `sigx`, and `sigy`. Label all curves clearly.

(f) (*4 marks) Lévy flights are a special class of random walks where the steps are drawn from "fat-tailed" probability distributions. One such probability distribution is the Cauchy distribution, which has the probability density function

$$p(x) = \frac{1}{\pi(1+x^2)}. \tag{5}$$

Consider a 2D Lévy flight where the displacement vectors are

$$\vec{\ell}_n = \begin{pmatrix} \Delta r \cos\theta \\ \Delta r \sin\theta \end{pmatrix}, \quad \begin{cases} \Delta r & \sim \text{Cauchy distribution} \\ \theta & \sim \text{Uniform}(-\pi, \pi). \end{cases} \tag{6}$$

Write a function `levy_flight_demo(N)` to investigate the properties of Lévy flights of length N. This function should plot the following:

(i) The x-y trajectory of a sample Lévy flight of length N.

(ii) The histogram of $|\vec{r}_N|^2$. You can plot histograms using the `plt.hist` function. You should specify the histogram bins explicitly, by supplying an array as a `bins` input to `plt.hist`. The size of the statistical sample is up to you.

Discuss in code comments: how do the Lévy flight trajectories differ from the Gaussian random walk trajectories studied previously? What happens if you try to calculate the mean squared deviation for the Lévy flight, and why?