Problem Set 0

- *Submission format*—Submit your solution via NTULearn, with **one Python source file per problem (`.py` not `.ipynb`)**. Each file should begin with the lines

  ```
  import numpy as np
  import scipy
  import matplotlib.pyplot as plt
  ```

  You may also import other Scipy modules or standard Python modules, but do not use non-standard modules (e.g., stuff from PyPI).

- *Coding assistants*—AI coding assistants may be used if you want, though all assignments can be completed without them. If you use an AI assistant, you must carefully scrutinize, test, and fix up the code it generates. But do not perform verbatim copying of other people's work, including other students taking this course; that is treated as plagiarism.

- *Code quality*—For full marks, the submitted programs must meet these criteria:

  1. Keep all other top-level code, aside from import statements and function/class definitions, to a minimum. Most of your code should live inside functions. Your program should work if we `import` it as a module and call its functions.
  2. If the assignment asks you to write a function, follow the specification exactly. Do not modify the order or definitions of the inputs/outputs to suit yourself. You are free to define additional helper functions, classes, or data structures.
  3. Use comments to document important code blocks. In particular, if you define helper functions or class methods, document their inputs/outputs clearly.
  4. Avoid cryptic function or variable names like `abc`.
  5. Nontrivial numerical constants should be grouped appropriately (e.g., the start of a function), not "hard-coded" deep inside code blocks.
  6. Functions should run appropriate "sanity checks" (e.g., using `assert`) on their inputs.
  7. Every generated plot should be properly formatted, with appropriate axis ranges, a proper figure title and axis labels, a legend if there are multiple curves per graph, etc.

0. INTERFERENCE PATTERNS (20 MARKS)

In this problem, we will compute and visualize waves produced by point sources. Consider a point source located at $\vec{r}_0$ in 3D space, emitting a scalar monochromatic wave described by a time-independent complex wavefunction

$$\psi(\vec{r}) = \frac{a \exp(ik|\vec{r} - \vec{r}_0|)}{|\vec{r} - \vec{r}_0|}, \tag{0}$$

where $\psi(\vec{r})$ gives a complex number at each point $\vec{r}$, $a \in \mathbb{C}$ is a complex amplitude, and k is the wavenumber (which depends on the frequency). Note that $|\vec{v}|$ denotes the magnitude of vector $\vec{v}$.

For N point sources with complex amplitudes $a_1, \ldots, a_N$ and positions $\vec{r}_1, \ldots, \vec{r}_N$, all with the same frequency (and hence the same k), the total wavefunction is

$$\psi(\vec{r}) = \sum_{n=0}^{N-1} a_n \frac{\exp(ik|\vec{r} - \vec{r}_n|)}{|\vec{r} - \vec{r}_n|}. \tag{1}$$

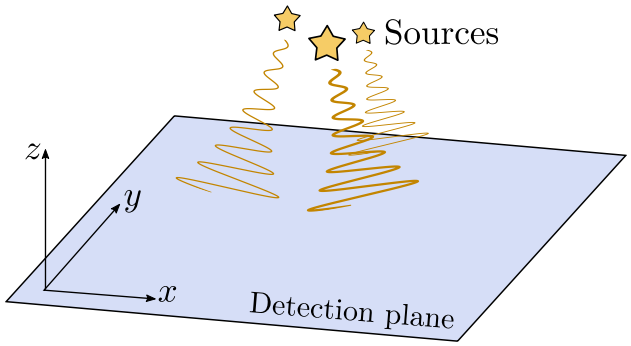
(a) (5 marks) Write a function to compute the wavefunction based on Eq. (1):

`def wavefunction(r, src_a, src_r, k):`	
Inputs	
`r`	2D array giving the positions at which to calculate $\psi(\vec{r})$, such that `r[m,j]` is the j-th coordinate of the m-th position.
`src_a`	A 1D complex array of wave amplitudes, such that `src_a[n]` $= a_n$. It can also be a single number, which is treated as the amplitude for a single source.
`src_r`	A 2D array of source positions, using the same format as `r`: i.e., `src_r[n,j]` is the j-th component of $\vec{r}_n$. It can also be a 1D array, which is treated as the position for a single source.
`k`	The wavenumber k (a real number).
Return Value	
`psi`	A complex 1D array, such that entry n is the value of ψ at the n-th point specified in `r`.

Note that all other parameters required for the calculation can be inferred from the function inputs (e.g., N can be inferred from the length of `src_a` and/or the shape of `src_r`). For full credit, the function should (i) do some sanity checks on its inputs, and (ii) be reasonably efficient, e.g. avoiding unnecessary nested for-loops.

Hint—Many Numpy array functions default to real arrays, so you may need to use the `dtype` argument if you need to make a complex array.

(b) (5 marks) Suppose the wavefunction produced by a set of sources is measured along some 2D "detection plane" (assumed to be the x-y plane):



We can visualize the interference pattern using a "pseudocolor plot" (a.k.a. a "heat map"), showing the detection plane with a color at each (x, y) point corresponding to the local intensity

$$I(x, y) = |\psi(x, y, z)|^2 . \tag{2}$$

Pseudocolor plots can be made using the `pcolormesh` function from Matplotlib.

Write the following function:

`def interference_plot(xspan, yspan, z, src_a, src_r, k):`	
Inputs	
`xspan`	A tuple of three numbers `(xmin, xmax, M)`, specifying the x coordinates for the detector: i.e., `M` numbers between `xmin` and `xmax` (inclusive).
`yspan`	A tuple of three numbers `(ymin, ymax, N)`, specifying the y coordinates for the detector: i.e., `N` numbers between `ymin` and `ymax` (inclusive).
`z`	The value of z for the detection plane (a real number).
`src_a`, `src_r`, `k`	Specifications for the sources, with the same meanings as in the `wavefunction` function from part (a).

This function should generate and show the pseudocolor plot of $I(x, y)$ in the detection plane.

Hint—to generate the x-y coordinate arrays, you may use `linspace`, `meshgrid`, and/or `reshape`.

(c) (3 marks) Write a function, `interference_demo()`, which takes no inputs and produces a plot demonstrating the use of `interference_plot` for some interesting configuration(s) of source(s). You are free to choose the parameters for this demo. In code comments, explain your choices and/or your interpretation of the results.

(d) (7 marks) We can translate the time-independent complex wavefunction $\psi(\vec{r})$ to a "physical" wavefunction, which is time-dependent and real-valued, as follows:

$$\Psi(\vec{r}, t) = \mathrm{Re}\left[\psi(\vec{r})e^{-i\omega t}\right]. \tag{3}$$

Here, ω is the angular frequency, which is related to the wavenumber k by $\omega = ck$, where c is the wave speed (which we can normalize to 1).

Write a function to generate an animated visualization of $\Psi(\vec{r}, t)$ in a detection plane:

`def wavefunction_animate(xspan, yspan, z, src_a, src_r, k):`	
Inputs	
`xspan`, `yspan`, `z`, `src_a`, `src_r`, `k`	These all have the same definitions as in `interference_plot`. The wave speed is assumed to be normalized to $v = 1$.

Each frame of the animation should be a `pcolormesh` plot similar to part (b), but showing the physical wavefunction Ψ rather than the intensity I. The animation should cycle, making use of the wave period $T = 2\pi/\omega$.

1. THE GAMBLER'S RUIN (10 MARKS)

A gambler starts with $\$x$ and repeatedly plays a game. In each round of the game, the probability of winning is p, with $p = 0.5$ corresponding to a "fair" game. The gambler gains \$1 for each win and losing \$1 for each loss. It turns out that if $p \leq 0.5$ (*including* the fair case), the gambler is guaranteed to be ruined (i.e., reaching $x = 0$) provided the game runs long enough. Here, we will set up simulations of the Gamber's Ruin scenario and extract its statistical properties.

(a) (5 marks) Write a function to run and plot several simulations of the Gambler's Ruin:

`def gamblers_ruin_trajectory_plot(p, x_init, nsim, max_rounds):`	
Inputs	
`p`	The win probability p (a real number).
`x_init`	The gambler's starting capital (a real number).
`nsim`	The number of independent simulations to perform.
`max_rounds`	The maximum number of rounds to simulate.

The function should plot all simulated "capital trajectories", as graphs of x versus t (the number of elapsed rounds), in a single figure. Note that each trajectory should stop if the gambler is ruined before `max_rounds` (in which case the graph should simply stop).

Hint 1—Matplotlib's `plot` function is pretty slow, so try to avoid calling it inside nested loops, e.g. by accumulating data into arrays and plotting in one shot.

Hint 2—Drawing N random numbers in one shot can be much faster than drawing N individual numbers. Experiment with this, and try to find a reasonably efficient way to do the random walks.

(b) (5 marks) Write a function, `gamblers_ruin_stats()`, which takes no inputs and runs simulations to generate the following plots:

- A histogram (plotted with `hist`) of T, the number of rounds before ruin, for $p = 0.5$.
- A plot of $\langle T \rangle$ (the mean number of rounds to ruin) versus p, for one or more choices of initial capital x_0. The plot must include error bars to indicate the uncertainty.
- A plot of $\langle T \rangle$ versus initial capital x_0, for one or more choices of p. As with the previous plot, error bars should be included.

Your code may assume that $p \leq 0.5$. You should include comments documenting the following:

- How the uncertainty in the $\langle T \rangle$ plots is defined and determined.
- What values you chose for the other model parameters.
- The maximum number of simulated rounds in used, if any. (This should be sufficiently large that it does not affect the statistical results, since the Gambler's Ruin game is supposed to run indefinitely.)

(Optional task; 0 marks)—Add theoretical predictions to some or all of the above plots. Label and document these predictions clearly.