Problem Set 3

- *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 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. TIME-INDEPENDENT SCHRÖDINGER EQUATION IN 2D (15 MARKS)

In this problem, we will study the 2D time-independent Schrödinger equation,

$$\left[-\frac{1}{2}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)+V(x,y)\right]\psi(x,y)=E\,\psi(x,y), \tag{0}$$

where (x, y) are 2D Cartesian coordinates, $V(x, y)$ is the potential, and $\psi(x, y)$ is an energy eigenfunction with energy E. We assume $\hbar = m = 1$. For simplicity, take a rectangular domain of size $L_x \times L_y$, centered at the origin, with Dirichlet boundary conditions:

$$\psi(x=\pm L_x/2, y)=\psi(x, y=\pm L_y/2)=0. \tag{1}$$

(a) (6 marks) Implement a finite-difference eigensolver for Eqs. (0)–(1):

`def schrodinger_solver_2d(Lx, Ly, Nx, Ny, Vfun, neigs, E0, args=()):`	
Inputs	
`Lx`, `Ly`	Length of the domain in each direction.
`Nx`, `Ny`	Number of discretization points in each direction (integers).
`Vfun`	Function specifying the potential. It is called as `Vfun(x,y,...)`, where x and y are equal-shaped arrays specifying x and y coordinates, and additional inputs (if any) specified by `args`. Its return value should be an array with the same shape/size as x and y, giving $V(x, y)$ at the specified points.
`neigs`	Number of energy eigenstates to find.
`E0`	Target energy around which to find energy eigenvalues. The function should solve for the `neigs` states with energies closest to `E0`.
`args`	Optional parameter, specifying a tuple of additional inputs for `Vfun`.

Return values	
`E`	1D array of energy eigenvalues.
`psi`	3D array such that `psi[n,i,j]` $\equiv \psi_n(x_i, y_j)$. In other words, `psi[n]` stores the n-th eigenfunction, in the form of a 2D array corresponding to the specified x and y coordinate points.
`x`	1D array of x coordinates.
`y`	1D array of y coordinates.

Notes:

- It is up to the function to construct the `x` and `y` coordinates based on the inputs. The points corresponding to the boundary (where the wavefunction vanishes) should be *excluded*. To convert each eigenfunction into a 2D array for storing into `psi`, you may use `meshgrid`, but pay attention to its `indexing` argument.
- Use sparse arrays to construct the Hamiltonian, and use `eigsh` to find the eigenvalues.
- The returned wavefunctions should be approximately normalized to unity, according to

$$\int_{-L_x/2}^{L_x/2} dx \int_{-L_y/2}^{L_y/2} dy \, |\psi(x,y)|^2 = 1. \tag{2}$$

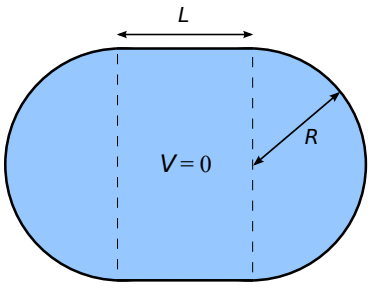
(b) (5 marks) Consider a 2D "particle in a box" problem, with a box of size $L_x \times L_y$, and $V(x,y) = 0$ inside the box. For this problem, write a function `schrodinger_2D_error_demo()` that demonstrates the validity of the solver from (a) in the following ways:

(i) Pseudocolor plots for some representative eigenfunctions produced by the solver.

(ii) For a few representative eigenfunctions, test the numerically-calculated energy eigenvalue by comparing it to the exact energy, which you should be able to derive analytically. Show how the error scales with the spatial discretization (e.g., you can discretize space by $N = N_x = N_y$, and see how the error varies with N). Characterize the observed scaling relation (e.g., if it is a power law, estimate the power.)

Be sure to label all figures clearly, and document your choice of parameters, and other relevant details, in code comments.

(c) (4 points) Next, we investigate a potential well known as a "stadium billiard". Suppose $V = 0$ inside a stadium-shaped region, as shown on the right, consisting of two semicircles of radius R joined by a rectangle of length L. Outside the stadium, V is infinite (or some very large value).



Write a function `stadium_demo()`, to plot some representative eigenstates for such a stadium. For instance, consider $R = 10$, $L \in \{1, 10\}$, and energies $E \sim 20$ (or choose other interesting values). Discuss your findings in code comments.

1. TIGHT-BINDING MODEL WITH HEXAGONAL LATTICE (15 MARKS)

We previously encountered 1D tight-binding models in Problem Set 1; in this problem we will deal with a tight-binding model of greater spatial complexity.

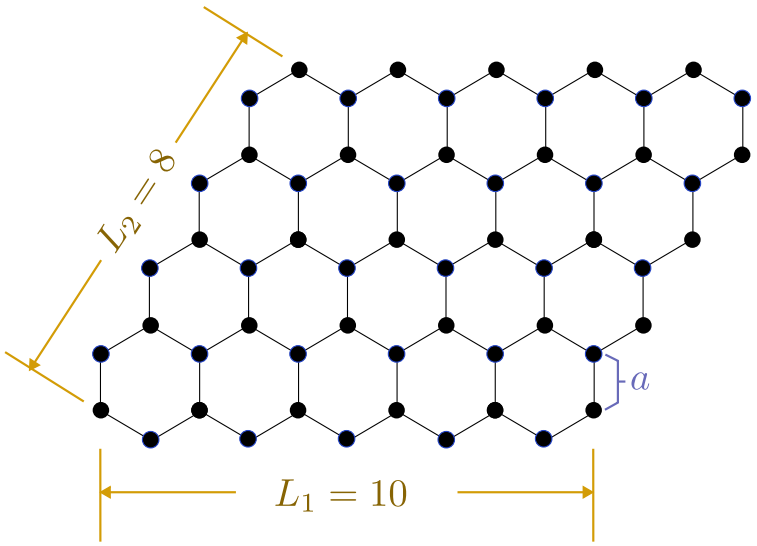
In general, a tight-binding model can contain N sites, with site j located at position

$$\mathbf{r}_j = (x_j, y_j, \dots), \qquad j = 0, 1, \dots, N-1. \tag{3}$$

These positions are not necessarily laid out in a nice Cartesian grid. Moreover, each quantum state is described by a complex vector $[\psi_0, \dots, \psi_{N-1}]^T$, where ψ_j is the quantum amplitude on site j. These state vectors are normalized by $\sum_j |\psi_j|^2 = 1$. An energy eigenstate $|\psi_n\rangle$ is a solution to the time-independent Schrödinger equation

$$H|\psi_n\rangle = E_n|\psi_n\rangle \quad \leftrightarrow \quad \sum_j H_{ij}\psi_j^{(n)} = E_n\psi_j^{(n)}. \tag{4}$$

We will consider a tight-binding model formed by a honeycomb lattice in 2D, with a parallelogram-like boundary:



The spacing between nearest neighbors is a, and the paralellogram has L_1 and L_2 spacings along each side. We moreover assume the Hamiltonian has the form

$$H_{ij} = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are not nearest neighbors,} \\ t\exp\left[i\mathbf{A}\cdot(\mathbf{r}_i - \mathbf{r}_j)\right] & \text{otherwise.} \end{cases} \tag{5}$$

Here, $t \in \mathbb{R}$ is some constant, while $\mathbf{A}(\mathbf{r})$ is some vector potential. Note that $H_{ii} = 0$. For each pair of nearest neighbors (i, j), the vector potential in Eq. (5) is to be evaluated at the midpoint between $\mathbf{r}_i$ and $\mathbf{r}_j$. You can check that this Hamiltonian satisfies the Hermiticity condition $H_{ij} = H_{ji}^*$.

(a) (10 marks) Implement a function to generate the lattice and its corresponding Hamiltonian.

`def honeycomb_model(a, L1, L2, t, Afun, args=()):`	
Inputs	
`a`	Nearest-neighbor distance.
`L1, L2`	Number of spacings on each side of the sample, as indicated in the above figure (integers).
`t`	The hopping parameter t in the Hamiltonian.
`Afun`	Function specifying the vector potential. It is called as `Afun(x,y,...)` where x and y are 2D position coordinates and additional inputs (if any) specified by `args`. Its return value is an array `[Ax, Ay]` giving the vector potential at that point.
`args`	Optional parameter, specifying a tuple of additional inputs for `Afun`.

Return values	
`sites`	2D array of site positions where `sites[i,:]` $\leftrightarrow \mathbf{r}_i$.
`H`	The Hamiltonian H, as a sparse array.

(b) (5 marks) Write a function `honeycomb_demo()` to demonstrate the features of the tight-binding model from (a) for reasonably large lattice sizes (several hundreds or thousands of sites).

For instance, you may take the vector potential

$$\mathbf{A}(x, y) = \frac{B_0}{2} \begin{pmatrix} -y \\ x \end{pmatrix}, \tag{6}$$

which corresponds to a magnetic field that is uniform everywhere in the 2D plane. You can also focus on eigenstates with energies closest to $E = 0$.

Look at both the eigenstates and the energy eigenvalues. It is up to you to decide what plots to generate. Label all figures clearly, and document your choices and your findings in code comments.