PHY407-Lab09

due November 22 2022

Physics Background

Time-dependent Schrodinger equation Solving the time-*independent* Schrodinger equation ($\mathbf{H}\psi = E\psi$, with $\mathbf{H}$ the Hamiltonian of the system) gives the eigenstates of quantum system. However, a quantum system can evolve over time, in which case we turn to the time-*dependent* Schrodinger equation

$$i\hbar\frac{\partial\psi}{\partial t} = \mathbf{H}\psi. \tag{1}$$

In this lab, we will focus on the one-dimensional equation for a particle of mass m in a potential $V(x)$, in which case eqn. (1) reduces to

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar}{2m}\frac{\partial^2\psi}{\partial x^2} + V\psi, \tag{2}$$

We can solve the equation above if we know the potential $V(x)$ and the initial wave function $\psi(x, t = 0)$. The probability to find the particle at a time t and location x is $\psi^*\psi$, where the asterisk denotes complex conjugation, which has to satisfy

$$\int_{-\infty}^{\infty}(\psi^*\psi)dx = 1. \tag{3}$$

Note that if the condition above is satisfied initially, a good numerical integration should maintain normalization. Once we know ψ, the "position" and energy of the particle at a given instant t are respectively given by

$$\langle X\rangle(t) = \int_{-\infty}^{\infty}\psi^*(x,t)\; x\; \psi(x,t)\; dx \tag{4}$$

$$E(t) = \int_{-\infty}^{\infty}\psi^*(x,t)\;\mathbf{H}\;\psi(x,t)\; dx$$

As for V, we will be considering the square well of width L, i.e. $V(x) = 0$ between $-L/2$ and $+L/2$ and infinite elsewhere.

Electric and magnetic field in 2D resonant cavity We can write the electric and magnetic fields within the cavity we will investigate in this lab as:

$$\vec{E} = \begin{pmatrix} 0 \\ 0 \\ E_z(x,y,t) \end{pmatrix} \tag{5}$$

$$\vec{B} = \begin{pmatrix} B_x(x,y,t) \\ B_y(x,y,t) \\ 0 \end{pmatrix}.$$

It follows from Maxwell's equations that

$$\frac{\partial H_x}{\partial t} + c\frac{\partial E_z}{\partial y} = 0 \tag{6a}$$

$$\frac{\partial H_y}{\partial t} + c\frac{\partial E_z}{\partial x} = 0 \tag{6b}$$

$$\frac{\partial E_z}{\partial t} + c\frac{\partial H_y}{\partial x} + c\frac{\partial H_x}{\partial y} = J_z \tag{6c}$$

where c is the speed of light, $H_x = cB_x$, $H_y = -cB_y$, $J_z = -\mu_0 c^2 j_z$ is the electric current density, and μ_0 is the vacuum permeability. The above system of equations takes the form of three coupled advection equations with a source term. The boundary conditions are that the tangential electric field and the normal magnetic field must be zero at the conducting walls, explicitly stated as

- $E_z = 0$ at all the walls (which are located at $x = 0, L_x$ and $y = 0, L_y$),
- $H_x = \partial_x H_y = 0$ at $x = 0, L_x$, and
- $H_y = \partial_y H_x$ at $y = 0, L_y$.

Finally, the normalized current pattern associated with the (m, n) mode takes the form

$$J_z(x, y, t) = J_0 \sin\left(\frac{m\pi x}{L_x}\right) \sin\left(\frac{n\pi y}{L_y}\right) \sin(\omega t). \tag{7}$$

Algorithmic Background

Spatial discretisation of the time-dependent Schrodinger equation As we saw in class, to solve PDEs, we can first discretise in space followed by a discretisation in time. First, we write that $x = x_p = pa - L/2$, with $p \in \{1, \dots, P-1\}$, P the number of cells in the x direction, and a the step size (in other words, $L = aP$). Note that because the potential has infinite walls, $\psi = 0$ at both ends of the domain, which is why we start the count at $p = 1$ and end it at $p = P - 1$. We can then approximate eqn. (2) as

$$i\hbar\frac{\partial \Psi}{\partial t} = \mathbf{H}_D \Psi, \tag{8}$$

where $\Psi(t)$ is an array containing all discretized values of ψ, i.e.,

$$\Psi(t) = \begin{pmatrix} \psi_1(t) \\ \vdots \\ \psi_p(t) \\ \vdots \\ \psi_{P-1}(t) \end{pmatrix} \tag{9}$$

with $\psi_p(t) = \psi(pa - L/2, t)$, and where $\mathbf{H}_D$ is the discretized Hamiltonian,

$$\mathbf{H}_D = \begin{pmatrix} B_1 & A & 0 & \cdots & & & & \\ A & B_2 & A & 0 & \cdots & & & \\ 0 & \ddots & \ddots & \ddots & & & & \\ \vdots & 0 & A & B_{p-1} & A & 0 & \cdots & \\ & \cdots & 0 & A & B_p & A & 0 & \cdots \\ & & & & & \ddots & \ddots & \end{pmatrix}, \tag{10}$$

with $A = -\hbar^2/(2ma^2)$ and $B_p = V(pa - L/2) - 2A$.

Crank-Nicolson time stepper for the time-dependent Schrodinger equation Picking up where we left off, we can discretize Ψ in time, with $\Psi^n = \Psi(n\tau)$, where $n \in \{1 \dots N-1\}$, N the number of time steps, and τ is the time step (in other words, the final time of the integration is τN). Recall that the Crank-Nicolson scheme averages results from an explicit and an implicit Euler time step, that is, it works with

$$i\hbar\Psi^{n+1} = i\hbar\Psi^n + \tau\mathbf{H}_D\Psi^n \Rightarrow \Psi^{n+1} = \left(\mathbf{I}_{P-1} - i\frac{\tau}{\hbar}\mathbf{H}_D\right)\Psi^n, \quad \text{(explicit)} \tag{11a}$$

$$i\hbar\Psi^{n+1} - \tau\mathbf{H}_D\Psi^{n+1} = i\hbar\Psi^n \Rightarrow \left(\mathbf{I}_{P-1} + i\frac{\tau}{\hbar}\mathbf{H}_D\right)\Psi^{n+1} = \Psi^n \quad \text{(implicit)}, \tag{11b}$$

where $\mathbf{I}_n$ denotes the rank-n identity matrix, to form

$$\left(\mathbf{I}_{P-1} + i\frac{\tau}{2\hbar}\mathbf{H}_D\right)\Psi^{n+1} = \left(\mathbf{I}_{P-1} - i\frac{\tau}{2\hbar}\mathbf{H}_D\right)\Psi^n. \tag{12}$$

Therefore, at each time step, we need to solve a linear system of equations

$$\mathbf{L}\Psi^{n+1} = v, \tag{13}$$

with $\mathbf{L} = \left(\mathbf{I}_{P-1} + i\frac{\tau}{2\hbar}\mathbf{H}_D\right)$, $v = \mathbf{R}\Psi^n$, and $\mathbf{R} = \left(\mathbf{I}_{P-1} - i\frac{\tau}{2\hbar}\mathbf{H}_D\right)$

Spectral solutions to Maxwell Equations To solve the Maxwell Equations, we will take a different approach in this lab than we saw in class. Our first step will be to define a Crank-Nicolson temporal procedure. Then we will design the handling of the spatial derivatives with Fourier modes: we will decompose each quantity we are solving for as series of sine or cosine functions, each of which satisfies the boundary conditions for said quantity.

Let us discretize time as $t_n = n\tau$, with τ the duration of a time step and $n \in \{1, \dots N-1\}$, and space as $(x_p, y_q) = (pa_x, qa_y)$, with a_x, a_y the sizes of a grid cell in the x- and y-directions, respectively, and $(p, q) \in \{1, \dots P-1\}^2$. The Crank-Nicolson temporal differentiation of eqns. (6) yields

$$\frac{(H_x)^{n+1}_{p,q} - (H_x)^n_{p,q}}{\tau} + \frac{c}{2}\left(\frac{\partial E_z}{\partial y}\right)^{n+1}_{p,q} + \frac{c}{2}\left(\frac{\partial E_z}{\partial y}\right)^{n}_{p,q} = 0, \tag{14a}$$

$$\frac{(H_y)^{n+1}_{p,q} - (H_y)^n_{p,q}}{\tau} + \frac{c}{2}\left(\frac{\partial E_z}{\partial x}\right)^{n+1}_{p,q} + \frac{c}{2}\left(\frac{\partial E_z}{\partial x}\right)^{n}_{p,q} = 0, \tag{14b}$$

$$\frac{(E_z)^{n+1}_{p,q} - (E_z)^n_{p,q}}{\tau} + \frac{c}{2}\left(\frac{\partial H_y}{\partial x}\right)^{n+1}_{p,q} + \frac{c}{2}\left(\frac{\partial H_y}{\partial x}\right)^{n}_{p,q} + \frac{c}{2}\left(\frac{\partial H_x}{\partial y}\right)^{n+1}_{p,q} + \frac{c}{2}\left(\frac{\partial H_x}{\partial y}\right)^{n}_{p,q} = 0, \tag{14c}$$

where $(H_x)^n_{p,q} = H_x(x_p, y_q, t_n)$, etc.

In space, we adopt a Fourier approach, and to satisfy the boundary conditions, we compute the discrete sine transforms (DST) and discrete cosine transforms (DCT) or the quantities, as relevant to satisfy the boundary conditions, namely,

$$(E_z)^n_{p,q} = \sum_{q'=0}^{P}\sum_{p'=0}^{P} \hat{E}^n_{p',q'} \sin\left(\frac{pp'\pi}{P}\right)\sin\left(\frac{qq'\pi}{P}\right), \tag{15a}$$

$$(H_x)^n_{p,q} = \sum_{q'=0}^{P}\sum_{p'=0}^{P} \hat{X}^n_{p',q'} \sin\left(\frac{pp'\pi}{P}\right)\cos\left(\frac{qq'\pi}{P}\right), \tag{15b}$$

$$(H_y)^n_{p,q} = \sum_{q'=0}^{P}\sum_{p'=0}^{P} \hat{Y}^n_{p',q'} \cos\left(\frac{pp'\pi}{P}\right)\sin\left(\frac{qq'\pi}{P}\right), \tag{15c}$$

$$(J_z)^n_{p,q} = \sum_{q'=0}^{P}\sum_{p'=0}^{P} \hat{J}^n_{p',q'} \sin\left(\frac{pp'\pi}{P}\right)\sin\left(\frac{qq'\pi}{P}\right). \tag{15d}$$

Plugging these equations in the Crank-Nicolson scheme (eqns. 14) and projecting on the basis functions sin and cos as relevant, we ultimately end up with the relations

$$\hat{E}^{n+1}_{p,q} = \frac{(1 - p^2 D_x^2 - q^2 D_y^2)\hat{E}^n_{p,q} + 2qD_y\hat{X}^n_{p,q} + 2pD_x\hat{Y}^n_{p,q} + \tau\hat{J}^n_{p,q}}{1 + p^2D_x^2 + q^2D_y^2} \tag{16a}$$

$$\hat{X}^{n+1}_{p,q} = \hat{X}^n_{p,q} - qD_y(\hat{E}^{n+1}_{p,q} + \hat{E}^n_{p,q}) \tag{16b}$$

$$\hat{Y}^{n+1}_{p,q} = \hat{Y}^n_{p,q} - pD_x(\hat{E}^{n+1}_{p,q} + \hat{E}^n_{p,q}) \tag{16c}$$

where $D_x = \pi c\tau/(2L_x)$, $D_y = \pi c\tau/(2L_y)$, and $L_x, L_y = Pa_x, Pa_y$, respectively.

Computational Background

Some common matrix manipulations in Python The matrices in eqns. (12) have properties that we can use to our advantage: they are banded. There are two ways to go about it: to use the NumPy matrix manipulation functions, or the more efficient (but more complicated) SciPy sparse matrix manipulation. This section will only discuss the former, and will omit the prefix `numpy.` in front of all functions (just assume they were already imported them from NumPy).

- The function `eye` primary purpose is to build an identity matrix of a certain size, though it also gives the option to choose which sub-diagonal to fill.
- The function `diag` is a more elaborate version of `eye`, in the sense that it allows you to fill the diagonal of your choice with the array of your choice, provided you know what you are doing with the dimensions (very easy to mess up!).
- The `dot` and `matmul` functions are similar and can sometimes used interchangeably. A rule of thumb is that the former should be used to take the dot product of two vectors (you can also take a look at `vdot` to take the dot product of a vector and its complex conjugate, but again, easy to mess it up), while the former should be used to multiply a matrix and a vector, or two matrices.
- To take the complex conjugate of an array A, element-wise, use `conj(A)`. (OK, this is not strictly a matrix manipulation function, but it's useful nonetheless)
- **Optional** Since the matrices in eqns. (12) are tridiagonal, Newman discusses a method to solve corresponding linear systems in § 6.1.6, with the corresponding routine `banded.py` downloadable on the online material of the textbook. If you are feeling adventurous you may use this less straightforward, but more efficient, method.

Below is a code snippet that would compute the matrix product $x = \mathbf{M}v$, with

$$\mathbf{M} = \begin{pmatrix} 2 & +i & 0 \\ -i & 3 & +i \\ 0 & -i & 4 \end{pmatrix} v = \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix}. \tag{17}$$

It is not the most efficient to go about in terms of number and length of lines, but it illustrates most functions described above.

```
from numpy import eye, diag, matmul, conj, arange, ones
vec_diag = arange(2, 5)
D = diag(vec_diag, k=0)  # k=0 means diagonal
Sup = 1j*eye(3, k=1)  # 3x3 matrix, with ones on 1st super-diagonal
Sub = conj(1j*eye(3, k=-1))  # 3x3 matrix, with ones on 1st sub-diagonal
M = D + Sub + Sup; print('\n', M)
v = 5*ones(3)
x = matmul(M, v); print('\n', x)
```

Computing diagnostics We are now reaching a level of complexity where it is often impossible to know what the "true" solution should be, and we will have to resort to indirect measures of accuracy. In particular, one of the first reflexes of a numerical physicist should be to check conservation laws. In our case, we can check that normalization and total energy are conserved (you can also throw in momentum if you wish).

Computing diagnostics of your solutions means computing quantities of the form

$$D = \int_{-\infty}^{\infty} \psi^* \, \mathbf{O} \, \psi \, dx, \tag{18}$$

with D some diagnostic quantity and $\mathbf{O}$ some operator (though because our potential well in this lab has infinitely tall walls, the integration bounds will be finite).

You can use the old methods for computing integrals that we have seen earlier in the term. When it comes to other operators however, some caution needs to be taken. Indeed, diagnosing quantities from a simulation is different than computing the integral of an analytical function, because the former has an internal consistency while the latter has a "true" value that exists regardless of how we compute it.

For example, to compute the wave function, you need to use the discretized Hamiltonian $\mathbf{H}_D$, which is essentially a differential operator. And to compute the energy, you will need to use the same discretized Hamiltonian, warts and all, for consistency's sake.

2D discrete Fourier transforms In Q2, we'll be taking the discrete cosine transform (DCT) and discrete sine transform (DST) for two dimensions, as seen in eqns. (15). The code in `dcst_for_q2.py` and `dcst.py` provides functions for computing the 1D transforms and inverse transforms. In order to accomplish the goal of going from 1D to 2D transforms, you may start with the following templates (the first function is the forward transform, the second is the backward or inverse transform.) Note that the direction order is reversed for the inverse transform.

```python
import dcst
import numpy as np

def dXXt2(f):
    """ Takes DXT along x, then DXT along y (X = C/S)
    IN: f, the input 2D numpy array
    OUT: b, the 2D transformed array """

    M = f.shape[0]  # Number of rows
    N = f.shape[1]  # Number of columns
    a = np.zeros((M, N))  # Intermediate array
    b = np.zeros((M, N))  # Final array

    # Take transform along x
    for j in range(N):
        # DXT f[:, j] and set as a[:, j]
    # Take transform along y
    for i in range(M):
        # DXT a[i, :] and set as b[i, :]
    return b

def idXXt2(b):
    """ Takes iDXT along y, then iDXT along x (X = C/S)
    IN: b, the input 2D numpy array
    OUT: f, the 2D inverse-transformed array """

    M = f.shape[0]  # Number of rows
    N = f.shape[1]  # Number of columns
```

```
a = np.zeros((M, N))  # Intermediate array
f = np.zeros((M, N))  # Final array

# Take inverse transform along y
for i in range(M):
    # iDXT b[i,:] and set as a[i,:]
# Take inverse transform along x
for j in range(N):
    # iDXT a[:,j] and set as f[:,j]
return f
```

Testing code for time-consuming simulations. In this lab, simulations take a while, minutes to tens of minutes! Therefore, while testing your code, reduce the number of grid cells or time steps. Only use the full integration durations and/or number of grid cells when you're producing the final figures / answers for the report.

Questions

1. Time-Dependent Schrodinger Equation [40%]

Use the following:

- Typical values for the properties of an electron: $L = 10^{-8}$ m, $m = 9.109 \times 10^{-31}$ kg.
- Initial condition $\psi(x, t = 0) = \psi_0 \exp\left(-\frac{(x-x_0)^2}{4\sigma^2} + i\kappa x\right)$, with $\sigma = L/25$ and $\kappa = 500/L$ and $x_0 = L/5$. Note that the value of σ ensures that $|\psi(x, t = 0)|/\psi_0$ is less than machine precision away from x_0, and less than the smallest number representable by Python at $x = \pm L/2$. You will need to compute the value of ψ_0, either analytically or numerically, in order for the initial wave function to be normalized.
- Spatial interval $[-L/2, L/2]$ split into $P = 1024$ segments.
- Time split into segments of duration $\tau = 10^{-18}$ s. Integrate for $N = 3000$ time steps, i.e., for a duration $T = N\tau$.

(a) Code the Crank-Nicolson method for the wave equation in the time-dependent Schrodinger equation in the case of an infinite square well.

As a reminder, steps in programming this scheme should include:

- definition of the potential $V(x)$
- definition of the discretized Hamiltonian $\mathbf{H}_D$
- loop to advance the solution in time

Submit your code.

(b) In order to validate your code, check that energy is constant, and that the wave function remains normalized throughout.

Submit appropriate plots.

2. Resonant EM cavity [60%]

Figure 1 shows a 2D resonant cavity consisting of a hollow, rectangular, perfectly conducting channel of dimensions $L_x \times L_y$. Suppose that the walls of the channel are aligned along the x- and y-axes. We shall excite this cavity in an artificial manner by imposing a z-directed alternating current pattern of angular frequency ω, which has the same spatial structure as the mode in which we are interested. We will calculate the electric and magnetic field patterns excited within the cavity by such a current pattern.

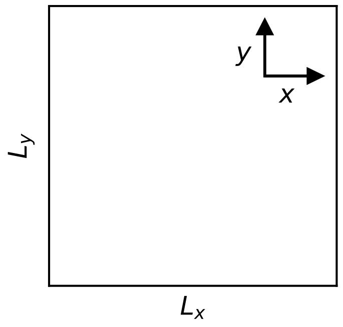

Figure 1: A $L_x \times L_y$ 2D resonant cavity with conducting walls.

(a) Starting from the sample code given in the computational background and the `dcst_for_q2.py` and `dcst.py` provided, write functions to compute the 2D Fourier transforms (and their inverses) for eqns. (15). Pay attention to what directions the transforms are taken, particularly for H_x and H_y. A way to check your functions is to test that $f = \mathcal{F}_{2D}^{-1}(\mathcal{F}_{2D}(f))$ is true for some 2D array f.

Submit your code, and printout of your test.

(b) Compute the electric and magnetic fields as follows:

- Implement the discretizations for t, x and y described in the Physics Background. Then, use eqn. (7) to generate the current pattern $J_z(x, y, t)$.
- Take the Fourier transforms of H_x, H_y, J_z, and E_z (use your code from the previous part).
- Evolve the Fourier coefficients using eqns. (16), which follow the Crank-Nicolson method.
- Reconstruct H_x, H_y, and E_z via a 2D inverse Fourier transform (use your code from the previous part).
- Loop over time.

Set the final time $T = N\tau = 20$, $\tau = 0.01$, $L_x = L_y = J_0 = m = n = c = 1$, $P = 32$. Use the driving frequency $\omega = 3.75$.

Plot the traces $H_x(x = 0.5, y = 0.0)$, $H_y(x = 0.0, y = 0.5)$, and $E_z(x = 0.5, y = 0.5)$ as a function of time.

Submit your code and plots.

(c) Explain the pattern of the traces (i.e. what is happening in the cavity).

Submit brief written explanation.