Lab assignment #9: Partial Differential Equations, Pt. II

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)
TA & Marker: Ahmed Rayyan (a.rayyan@mail.utoronto.ca)

Due Friday, 19 November 2021, 5 pm

It appears that there is no need for a second room. Should it be incorrect and the room overflows, N.G. will open up the room he had for the previous labs (grab the link on the old question documents, e.g., Lab #3).

Room 1 (also for office hour) Ahmed Rayyan and Nicolas Grisouard,
`https://gather.town/mIAKeWKElOnF4uL3/PHY407-AR`

General Advice

- **Work with a partner!**
- Read this document and do its suggested readings to help with the pre-lectures and labs.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Carefully check what you are supposed to hand in for grading in the section "Lab Instructions".
- Whether or not you are asked to hand in pseudocode, you **need** to strategize and pseudocode **before** you start coding. Writing code should be your last step, not your first step.
- Test your code as you go, **not** when it is finished. The easiest way to test code is with `print('')` statements. Print out values that you set or calculate to make sure they are what you think they are.
- Practice modularity. It is the concept of breaking up your code into pieces that as independent as possible from each other. That way, if anything goes wrong, you can test each piece independently.
- One way to practice modularity is to define external functions for repetitive tasks. An external function is a piece of code that looks like this:

```
def MyFunc(argument):
"""A header that  explains the function
INPUT:
argument [float] is the angle in rad
OUTPUT:
res [float] is twice the argument"""
res = 2.*argument
return res
```

Place them in a separate file called e.g. MyFunctions.py, and call and use them in your answer files with:

```
import MyFunctions as fl2  # make sure file is in same folder
ZehValyou = 4.
ZehDubble = fl2.MyFunc(ZehValyou)
```

Physics Background

Waves on a string The one-dimensional wave equation that we solved in Lab08 is, using Newman's notation,

$$\frac{\partial^2\phi}{\partial t^2} = v^2\frac{\partial^2\phi}{\partial x^2}. \tag{1}$$

To fully specify the problem we require the boundary conditions that the string is fixed on each end, i.e.,

$$\forall t > 0, \quad \phi\big|_{x=0} = \phi\big|_{x=L} = 0, \tag{2}$$

and initial conditions on the displacement and velocity, i.e.,

$$\forall x \in [0, L], \quad \phi_0(x) = \phi\big|_{t=0} \quad \text{and} \quad \psi_0(x) = \psi\big|_{t=0} = \left.\frac{\partial\phi}{\partial t}\right|_{t=0}. \tag{3}$$

Time-dependent Schrödinger equation Solving the time-*independent* Schrödinger 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* Schrödinger equation

$$i\hbar\frac{\partial\psi}{\partial t} = \mathbf{H}\psi. \tag{4}$$

In Q1, we will focus on the one-dimensional equation for a particle of mass m in a potential $V(x)$, in which case eqn. (4) reduces to

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar}{2m}\frac{\partial^2\psi}{\partial x^2} + V\psi, \tag{5}$$

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

$$\int_{-\infty}^{\infty} \psi^*\psi\,\mathrm{d}x = 1. \tag{6}$$

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

$$\langle X\rangle(t) = \int_{-\infty}^{\infty} \psi^*(x,t)x\psi(x,t)\,\mathrm{d}x \quad \text{and} \quad E(t) = \int_{-\infty}^{\infty} \psi^*(x,t)\mathbf{H}\psi(x,t)\,\mathrm{d}x. \tag{7}$$

As for V, we will be considering three different potentials in this lab. Within a potential of width L, i.e., between $-L/2$ and $+L/2$, the three cases will be

$$\text{Square well:} \quad V(x) = 0, \tag{8a}$$

$$\text{Harmonic oscillator:} \quad V(x) = \frac{1}{2}m\omega^2 x^2, \tag{8b}$$

$$\text{Double well:} \quad V(x) = V_0\left(\frac{x^2}{x_1^2} - 1\right)^2. \tag{8c}$$

Outside of $(-L/2, L/2)$, the potential V will be infinite.

In this lab, you will be able to check results related to the Ehrenfest theorem. I will however focus on one consequence of this theorem, which has a certain regime of validity, and that you should be able to confirm in your results[1], namely,

> "[f]or general systems, if the wave function is highly concentrated around a point x_0, then $V'(\langle x\rangle)$ [*where V' is the gradient of the potential*] and $\langle V'(x)\rangle$ will be almost the same, since both will be approximately equal to $V'(x_0)$. In that case, the expected position and expected momentum will approximately follow the classical trajectories, at least for as long as the wave function remains localized in position."

Burger's equation We will carry out a second exercise with Burgers' equation, which we studied in Lab08. For reference we include Burger's equation in conservative form here:

$$\frac{\partial u}{\partial t} + \epsilon\frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = 0. \tag{9}$$

Computational Background

Spectral method Section 9.3.4 of the Newman text discusses solving the wave-on-string problem using spectral methods. The solution method is to decompose the waveform into a Fourier series of a finite number of sine waves, each of which satisfies the boundary conditions. An example of the discretized Fourier series is shown in Section 9.3.4 and the coefficients in the series can be obtained using the fast discrete sine transform (DST) that is discussed briefly in Section 7.4.3 and is made available in the package `dcst.py` provided in the textbook's online resources, which implements the fast cosine and sine transforms. (We will use slightly different notation from the book in Q2 below.)

The spectral method writes the solution in terms of Fourier sine series and uses an algorithm to calculate the series for a finite number of terms. We need to derive expressions for derivatives in spectral space, as discussed in class, and then find the solution doing an inverse sine transform.

If you understand the principles of this question there is actually very little coding involved. You will see, in fact, that the code you have runs almost instantly, in contrast to the FTCS exercise where you have to integrate a long time to get a snapshot of the system at the desired time. The spectral method is great as long as the conditions are right to apply it!

[1] lifted from `https://en.wikipedia.org/wiki/Ehrenfest_theorem`

Animations in Matplotlib return WARNING: creating animations is as illuminating as it is time-consuming. I (N.G.) believe that your experience will be much more enriching if you use them, but we are not requiring you to hand in any.

Last week, I proposed a very simple way to animate Matplotlib pictures. Now that we have it down, let's try a somewhat more elaborate method, based on the `matplotlib.animation.FuncAnimation` function, with the addition of saving the result in `.mp4` format.

See this method in the script `animation_demo.py`, itself based off of one of the basic examples you will find at https://matplotlib.org/stable/api/animation_api.html. In that script, I animate $\sin(x-t)$. It works on my computer (a Mac; I am fairly confident it works on Linux, but not sure about Windows). If it does not work on yours, please let me know, and I will try to amend this section and the corresponding script.

Spatial discretisation of the time-dependent Schrödinger equation *Adapted from Newman's intro to Exercise 4.8, pp. 439–441.*

As we saw in class, to solve for PDEs, we first discretise in space, followed by a discretisation in time. So, first, we write that $x = x_p = p\delta - L/2$, with $p \in \{1, \dots, P-1\}$, P the number of cells in the x direction, and δ the step size (in other words, $L = \delta P$). Note that because the potentials have 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. (5) as

$$\mathrm{i}\hbar\frac{\partial \Psi}{\partial t} = \mathbf{H}_D \Psi, \tag{10}$$

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

$$\Psi(t) = \begin{pmatrix} \psi_1(t) \\ \vdots \\ \psi_p(t) \\ \vdots \\ \psi_{P-1}(t) \end{pmatrix} \quad \text{with} \quad \psi_p(t) = \psi(p\delta - L/2, t), \tag{11}$$

and where $\mathbf{H}_D$ is the discretized Hamiltonian,

$$\mathbf{H}_D = \begin{pmatrix} B_1 & A & 0 & \cdots & & & & \\ A & B_2 & A & 0 & \cdots & & & \\ 0 & \ddots & \ddots & \ddots & & & & \\ \vdots & 0 & A & B_{p-1} & A & 0 & \cdots & \\ & \cdots & 0 & A & B_p & A & 0 & \cdots \\ & & & & & \ddots & \ddots & \end{pmatrix}, \tag{12}$$

with $A = -\hbar^2/(2m\delta^2)$ and $B_p = V(p\delta - L/2) - 2A$.

Crank-Nicolson time stepper for the time-dependent Schrödinger equation *Also adapted from Newman's intro to Exercise 4.8, pp. 439–441.*

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

$$\mathrm{i}\hbar\Psi^{n+1} = \mathrm{i}\hbar\Psi^n + \tau\mathbf{H}_D\Psi^n \quad \Rightarrow \quad \Psi^{n+1} = \left(\mathbf{I}_{P-1} - \mathrm{i}\frac{\tau}{\hbar}\mathbf{H}_D\right)\Psi^n, \quad \text{(explicit)} \quad \text{and} \tag{13a}$$

$$\mathrm{i}\hbar\Psi^{n+1} - \tau\mathbf{H}_D\Psi^{n+1} = \mathrm{i}\hbar\Psi^n \quad \Rightarrow \quad \left(\mathbf{I}_{P-1} + \mathrm{i}\frac{\tau}{\hbar}\mathbf{H}_D\right)\Psi^{n+1} = \Psi^n \quad \text{(implicit)}, \tag{13b}$$

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

$$\left(\mathbf{I}_{P-1} + \mathrm{i}\frac{\tau}{2\hbar}\mathbf{H}_D\right)\Psi^{n+1} = \left(\mathbf{I}_{P-1} - \mathrm{i}\frac{\tau}{2\hbar}\mathbf{H}_D\right)\Psi^n. \tag{14}$$

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

$$\mathbf{L}\Psi^{n+1} = v, \tag{15}$$

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

Computing diagnostics We are now reaching a level of complexity where it is impossible to know what the "true" (e.g., analytical) solution should be, and we will have to resort to indirect measures of accuracy. In particular, on the first reflexes of a numerical physicist is 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

$$D = \int_{-\infty}^{\infty} \psi^* \mathbf{O} \psi \, \mathrm{d}x, \tag{16}$$

with D some diagnostic quantity and $\mathbf{O}$ some operator (though because our potential wells will have 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 inherited from the discretisation schemes, 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.

Saving output for reuse Now that we are starting to write longer programs to handle more complex problems, it becomes important to save your data and output from intermediate steps to reuse at another time. For example, the time-integrations in this lab can take several tens of minutes. But what if you computed the field correctly, and then realized you made a mistake in plotting a figure? You may want to create an if statement such as

```
I_want_to_compute everything = False
if I_want_to_compute_everything:
    # ...
    # The time-integration procedure
    # ....
    # concludes with command to save all the results you will need for plotting
    # in a separate file
else:
    # a command to load the files you need in the separate file created above
```

A simple way to do this is through `numpy.save` and `numpy.savez` commands. For example, if you want to save arrays `x`, `y`, and `phi` to a file, you can do the following.

```
import numpy as np
...
# x, y, and phi have already been defined

np.savez('output_file', x=x, y=y, phi=phi)
```

The output statement `np.savez` takes a filename `output_file`, creates a file with the name `output_file.npz`, and saves data to that file. The other keyword/value pair arguments `x=x`, etc., represent variables in the file that are custom-defined by you, the user. In this example, variables `x`, `y`, and `phi` will be saved with their corresponding names in the output file `output_file.npz`, in a binary format.

Once your program wrote the output file, you can open it in another program, using `np.load`, and read values in, as in the following example:

```
npzfile = np.load('output_file.npz')
x = npzfile['x']
y = npzfile['y']
phi = npzfile['phi']
```

In this example, `np.load` returns a Python dictionary[2] with the name `npzfile` whose keys (`'x'`, `'y'`, `'z'`) correspond to the variable names and whose values are the numpy arrays read from the `.npz` file.

Notation reminder for solving Burger's equation As for solving the Burger's equation in Lab08, the space and time directions are discretized into N and T steps of sizes Δx and Δt respectively:

$$x_i = i\Delta x, \quad i = 0, ..., N-1 \tag{17}$$

$$t_j = j\Delta t, \quad j = 0, ..., T-1 \tag{18}$$

and we write $u_i^j = u(x_i, t_j)$.

Questions

In some of these questions (especially Q2), these simulations take a while, minutes to tens of minutes! (though your computers might be faster than N.G.'s antique) When testing, reduce the number of grid cells or time steps, and only use the full integration durations and/or number of grid cells to produce the report figures.

[2] https://realpython.com/python-dicts/, but you don't need to know what it is for this lab.

1. **[15%] Solving the wave equation with the spectral method** Consider the wave problem Lab08 Q2, consisting of equations (1)–(3) and the initial conditions provided by Newman in Exercise 9.5(a), p. 431, which you used last week. Suppose you can expand the initial conditions in terms of a Fourier sine series, namely,

$$\phi_0(x) = \sum_{k=1}^{\infty} \tilde{\phi}_{0,k} \sin\left(\frac{k\pi x}{L}\right) \quad \text{and} \quad \psi_0(x) = \sum_{k=1}^{\infty} \tilde{\psi}_{0,k} \sin\left(\frac{k\pi x}{L}\right). \tag{19}$$

Note that all the terms in this series vanish at $x = 0$ and $x = L$, automatically satisfying the boundary conditions (2) for ϕ, and similarly for $\psi = \partial_t \phi$.

(a) Show by substitution that the general solution to (1)–(3) in terms of the Fourier series is

$$\phi(x,t) = \sum_{k=1}^{\infty} \sin\left(\frac{k\pi x}{L}\right)\left[\tilde{\phi}_{0,k}\cos(\omega_k t) + \frac{\tilde{\psi}_{0,k}}{\omega_k}\sin(\omega_k t)\right] \tag{20}$$

where you will need to derive the expression for the frequency ω_k, which depends on k.

HAND IN A CONCISE DERIVATION, EITHER HAND-WRITTEN OR IN TEX.

(b) Using a series solution like this but truncating at a finite N, use the `dst` and `idst` functions in `dcst.py` to calculate and plot the solution $\phi(x,t)$ for the same problem as in Q2 of Lab08, at $t = 2$, 4, 6, 12 and 100 ms. You can use a very large number of Fourier coefficients in this calculation since you only have to do one Fourier transform.

HAND IN CODE AND PLOTS FOR EACH TIME STEP, t.

(c) Compare your solution to the FTCS solution using the code from Q2 of Lab08. The latter can be your code or the code posted with the Lab08 solutions.

HAND IN A SEVERAL LINES OF TEXT EXPLAINING THE DIFFERENT APPROACHES, AND THE RELATIVE ADVANTAGES AND DISADVANTAGES OF EACH.

2. **[60%] Solving the time-dependent Schrödinger equation with the Crank-Nicolson scheme:** This question, and the accompanying Physical and Computational backgrounds, are somewhat based on Exercise 9.8 (p. 439) of Newman's textbook, but not only. You may use it as support if you don't understand how to proceed.

For this question, we will use the following values:

- typical values for an electron, i.e., $L = 10^{-8}$ m, $m = 9.109 \times 10^{-31}$ kg or the value of m given by `scipy.constants`.
- Initial condition

$$\psi(x, t=0) = \psi_0 \exp\left(-\frac{(x-x_0)^2}{4\sigma^2} + \mathrm{i}\kappa x\right), \tag{21}$$

with $\sigma = L/25$ and $\kappa = 500/L$. Note that the value of sigma 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 $x = \pm L/2$. The value of x_0 will depend on the potential, and you will need to compute the value of ψ_0, either analytically or numerically, in order for the initial wave function to be normalized.

- Interval $[-L/2, L/2]$ split into $P = 1024$ segments.
- Time split into segments of duration $\tau = 10^{-18}$ s.

(a) In a first time, code the Crank-Nicolson for the wave equation in the time-dependent Schrödinger equation in the case of an infinite square well (eqn. 8a). To do so,

  - define $x_0 = L/5$,
  - integrate for $N = 3000$ time steps, i.e., for a duration $T = N\tau$.

  As a reminder, steps in programming this scheme should include the definition of the potential $V(x)$, the definition of the discretized Hamiltonian $\mathbf{H}_D$, the definition of one or more function(s) to compute the requested diagnostics (energy, normalization and expected position), and the loop to advance the solution in time.

  In order to ensure that your code is not doing anything funny, check that energy is constant, and that the wave function remains normalized throughout.

  HAND IN PLOTS OF ENERGY AND NORMALIZATION OF THE WAVE FUNCTION WITH TIME, AND COMMENTS ON IMPLEMENTATION AND PHYSICAL RESULTS.

(b) Describe the behaviour of the wave function. Alternatively, if you think that the wave function is too messy, you can comment on the probability density of the position, $\psi^*\psi$. How about the "trajectory" $\langle X\rangle(t)$: how does it compare with the out-of-context quote about the Ehrenfest theorem I used in the Physics background?

  HAND IN PLOTS OF THE REAL PART OF ψ, OR ALTERNATIVELY, OF $\psi^*\psi$, AT $t = 0$, $T/4$, $T/2$, AND T. ALTERNATIVELY, YOU MAY SUBMIT VIDEO FILES OF YOUR ANIMATIONS. IN ANY CASE, CLEARLY EXPLAIN WHAT YOU ARE SUBMITTING. FINALLY, HAND IN YOUR COMMENTS.

(c) Now do the same as part (b), but for the harmonic potential (eqn. 8b), with $\omega = 3\times 10^{15}$ rad s^{-1}, $x_0 = L/5$ again, but integrate over $N = 4000$ steps.

  HAND IN PLOTS OF THE REAL PART OF ψ, OR ALTERNATIVELY, OF $\psi^*\psi$, AT $t = 0$, $T/4$, $T/2$, AND T. ALTERNATIVELY, YOU MAY SUBMIT VIDEO FILES OF YOUR ANIMATIONS. IN ANY CASE, CLEARLY EXPLAIN WHAT YOU ARE SUBMITTING. FINALLY, HAND IN YOUR COMMENTS.

(d) Now do the same as parts (b) and (c), but for the double-well (eqn. 8c), with $V_0 = 6\times 10^{-17}$ J, $x_0 = L/3$, $x_1 = L/4$ and $N = 6000$ steps.

  SAME INSTRUCTIONS AS FOR THE PREVIOUS TWO QUESTIONS. ADDITIONALLY, HAND IN YOUR CODE FOR AT LEAST ONE CASE.

3. **[25%] Solving Burger's equation using Lax-Wendroff (25% of the lab):** In Lab08 we solved Burger's equation using leapfrog in time and centred differencing in space. We will now derive a more accurate algorithm to solve Burgers' equation. Specifically, we will use the *Lax-Wendroff* method, which uses a second order approximation for the time derivative. This derivation follows that given in the textbook *Computational Physics* by Landau, Paez and Bordeianu.

(a) To begin with, use a Taylor expansion to estimate the time derivative as

$$u(x,t+\Delta t) = u(x,t) + \frac{\partial u}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 u}{\partial t^2}(\Delta t)^2. \tag{22}$$

Then, using Equation (9), show that one can re-express the second order time derivative as

$$\left.\frac{\partial^2 u}{\partial t^2}\right|_{x,t} = \epsilon^2 \frac{\partial}{\partial x}\left[u\frac{\partial}{\partial x}\left(\frac{u^2}{2}\right)\right]. \tag{23}$$

Now, substitute Equation (9) and (23) into Equation (22) in order to show that

$$u(x,t+\Delta t) = u(x,t) - \Delta t\epsilon\frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) + \frac{(\Delta t)^2}{2}\epsilon^2\frac{\partial}{\partial x}\left[u\frac{\partial}{\partial x}\left(\frac{u^2}{2}\right)\right]. \tag{24}$$

Next, approximate the x-derivatives using central difference approximations. Specifically, for the second term on the right hand side of Equation 24, use

$$\begin{aligned}\frac{\partial u^2}{\partial x} &= \frac{u^2(x+\Delta x,t) - u^2(x-\Delta x,t)}{2\Delta x}\\ &= \frac{\left(u_{i+1}^j\right)^2 - \left(u_{i-1}^j\right)^2}{2\Delta x}\end{aligned} \tag{25}$$

For the third term on the right hand side of Equation (24), first estimate the outer derivative using a spacing of $\Delta x/2$, namely,

$$\frac{\partial}{\partial x}\left(u\frac{\partial u^2}{\partial x}\right) = \frac{1}{\Delta x}\left[u(x+\Delta x/2)\frac{\partial u^2(x+\Delta x/2,t)}{\partial x} - u(x-\Delta x/2)\frac{\partial u^2(x-\Delta x/2,t)}{\partial x}\right]. \tag{26}$$

Now, estimate the values of u at the intermediate grid points by taking the average of the adjacent grid points, as in,

$$\begin{aligned}u\left(x\pm\frac{\Delta x}{2},t\right) &= \frac{u(x,t)+u(x\pm\Delta x,t)}{2}\\ &= \frac{u_i^j + u_{i\pm1}^j}{2}.\end{aligned} \tag{27}$$

And finally, estimate the interior x-derivative using another central difference approximation with spacing $\Delta x/2$:

$$\frac{\partial u^2(x\pm\Delta x/2,t)}{\partial x} = \frac{u^2(x\pm\Delta x,t) - u^2(x,t)}{\pm\Delta x} \tag{28}$$

$$= \frac{\left(u_{i\pm1}^j\right)^2 - \left(u_i^j\right)^2}{\pm\Delta x}. \tag{29}$$

Now, substituting Equations (25) through (29) into Equation (24), find an expression for $u_i^{j+1} = u(x,t+\Delta t)$ in terms of u_i^j, u_{i+1}^j, u_{i-1}^j and $\beta = \epsilon\Delta t/\Delta x$.

HAND IN A CONCISE DERIVATION, EITHER HAND-WRITTEN OR IN TEX.

(b) Implement a code to solve Burgers' equation using the Lax-Wendroff method as derived above. Use the same parameters as in Q3 of Lab08 and save figures at the same time steps, $t = 0, 0.5, 1, 1.5$. Compare your solutions for the leapfrog and Lax-Wendroff methods. What changes?

HAND IN CODE AND PLOTS FOR EACH TIME STEP, t.