Assignment 2
Due by 11:59 pm on 30 October 2025

Covered contents:
Lectures 4-7

Submission method:
All codes must be submitted as separate `.py` files with names specified in the questions. For parts (a) and (b) in Question 4, you should also submit a PDF file showing your work.

A note on using generative tools:
You are welcome to use generative tools to assist you in completing assignments. However, you are not allowed to simply copy and paste; you should understand the outputs and submit works that are completed based on your own understanding. Failure to do so may result in penalties.

All coding questions must be done in Python.
If your have any questions, please email phycsan@ust.hk.

Question 1 [20 points]

Using the `threading` module in Python, implement a parallel histogram algorithm. Your parallel histogram function should accept 5 arguments:

1. `data`: an array of numbers from which the histogram is built
2. `low`: the lower range of the bins
3. `high`: the upper range of the bins
4. `n_bins`: number of bins used
5. `n_threads`: number of threads to use

The bins (except the last) should be half-open on the right: that is, they are of the form $[a, b)$. Your histogram function should return the count of elements in each bin.

For example, `histogram(data = [0.1, 0.2, 0.4, 0.2, 0.4, 0.9, 0.2, 0.1], low=0., high=1., n_bins=10, n_threads=<whatever>)` should return `[0, 2, 3, 0, 2, 0, 0, 0, 0, 1]`.

Here's one idea on how to do it:

- Maintain a global array `results` to store final output
- Each thread works on a chunk of the data by incrementing the counts in `results`

You are free to use alternative implementations as well if you think the one above is not going to be efficient ;)

You should also include a serial implementation for measuring the speed-up on randomly generated data.

A skeleton code titled '`histogram_threading.py`' is included along with this problem. Complete the missing sections there.

Question 2 [20 points]

In this problem, your goal is to implement a similar parallel histogram as in Question 1 by using `mpi4py`. Since by nature different processes do no share address spaces, we cannot use a global `results` array. Instead, all communication must be done explicitly. Here's one possible implementation:

- A master process (say process 0) scatters data to all processes
- Each process computes their own local histogram
- The master process gathers and combines all local histograms into the final historam

A skeleton code titled '`histogram_mpi.py`' is included along with this problem. Complete the missing sections there.

Question 3 [20 points]

We are given an $n \times d$ `numpy` array of Cartesian coordinates of n points in d-dimension. Our goal is to compute the $n \times n$ distance matrix D defined by

$$D_{ij} = \text{Euclidean distance between point } i \text{ and point } j. \tag{1}$$

(a) Compute the distance matrix by using nested `for` loops.

(b) You might notice that the nested loops in (a) are very slow for large n and d. Compute D by using only `numpy` functions. Do not use any for loops.

A skeleton code titled `distance.py` is included along with this assignment. Complete and missing sections.

Question 4 [10 points]

Laplace's equation reads $\nabla^2\phi = 0$. It is an important equation governing a lot of systems in physics. For example, imagine having an enclosed space, and the temperature distribution at the boundary is specified. In this case, the temperature distribution in the interior is given by the solution to Laplace's equation with the specified boundary condition.

For this question, let's assume that we are working in a two-dimensional (flat) space, which is a square of size 1 by 1 (in some unit). In this space, Laplace's equation becomes

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0. \tag{2}$$

One method to numerically solve (2) is to replace the space with its discretized version: Let's discretize by turning the square into an $(n+1) \times (n+1)$ grid of points, which we label by $\phi_{i,j}$. See fig. 1 below.

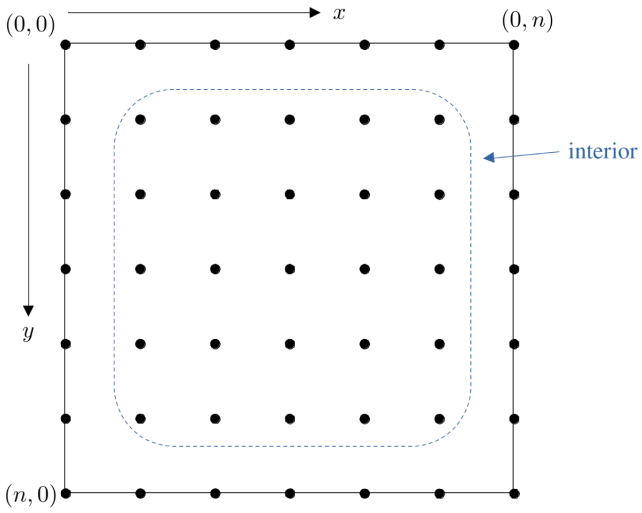

Figure 1: Discretization by a square grid of points.

The spacing of the grid is given by $h = 1/n$.

(a) Under this discretization scheme, show that the second derivative at the interior is approximately given by

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{(i,j)} \approx \frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{h^2}. \tag{3}$$

Derive a similar expression for the y-derivative.

(b) Using your results in (a), show that the Laplace equation can be approximately replaced by the following set of equations (one each for each pair of (i, j)):

$$\phi_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1}, \phi_{i,j-1}), \tag{4}$$

where $1 \leq i < n$ and $1 \leq j < n$.

One can devise an iterative scheme to numerically solve eq. (4):

```
phi_old = initialize()
max_diff = np.inf
iter = 0

while (iter <= iter_max) and (max_diff >= tol):
    phi_new = initialize()

    for i in range(1, n):
        for j in range(1, n):
            phi_new[i, j] = (phi_old[i+1, j] + phi_old[i-1, j] + \
                             phi_old[i, j+1] + phi_old[i, j-1])/4.0

    max_diff = np.max(np.abs(phi_new - phi_old))
    phi_old = phi_new
    iter += 1
```

In the algorithm above, `iter_max` is the maximum number of iterations allowed, `tol` is the error tolerance below which we define the algorithm to have converged. (We haven't proved the algorithm will actually converge to the correct answer, but let's just assume this is the case.)

(c) Write the `initialize` function which accepts five inputs:

- `phi_top`: value of ϕ at the top boundary
- `phi_bottom`: value of ϕ at the bottom boundary
- `phi_left`: value of ϕ at the left boundary
- `phi_right`: value of ϕ at the right boundary
- `n`: size of the grid minus 1

This function should return an $(n+1)\times(n+1)$ two-dimensional `numpy` array which satisfies the boundary conditions and has value zero for all interior points.

A skeleton code titled '`laplace_solver.py`' is included along with this problem. Complete the missing sections there.

To be continued in Assignment 3.....