(25.10.16) Computational Physics B

class 1

- initial value problem (IVP) (most important)
  - solve for time evolution of a giant system.
  - starting from some initial values
  - we focus on first order ODEs, because higher order ODEs can always be transformed into first order ODEs.
  - explicit method vs implicit method
    - explicit method: the value at the next time step is determined by the value at the current time step.
    - implicit method: the value at the next time step is determined by the value at the next time step itself, and we need to guess a value and by iteration to get the final value.
- boundary value problem (BVP)
  - usually solving 2nd order ODEs rather than 1st order ODEs.
    - Why 2nd order ODEs? (below are generated by ChatGPT)
      - 1st order ODEs usually have only one boundary condition, while 2nd order ODEs have two boundary conditions, which is suitable for BVP.
      - for example, the Poisson equation $\nabla^2\phi = 4\pi G\rho$ needs two boundary conditions to solve.
  - solve a static system with specified boundary conditions, not time evolution.
- eigenvalue problem (EVP)
  - usually for linear differential operator equations (e.g. $\hat{L}\psi = \lambda\psi$)
  - seek both the function and the eigenvalue to satisfy the equation and boundary conditions simultaneously.
  - example: linear stability analysis, mode instability analysis
  - physical meaning of eigenvalue λ : energy, frequency, growth rate, etc.
- The basic structure of numerical simulations (IVP)
  - initial values, boundary conditions -- these two are the input of the simulation.
  - solvers -- numerically solve the ODEs or PDEs.
  - time series outputs
  - data analysis and visualization

class 2

- how to construct and realize boundary conditions
- implemented in ghost cells, and will dynamically change with simulation time to suit for the specific boundary condition requirements.
- block: $n_x \times n_x$, with n_g ghost cells in each side.
  - the boundary conditions are implemented in the ghost cells, which is outside the blocks we care about.
  - the simulation region covers all the blocks and ghost cells, but in data analysis we only care about the physical quantities in the blocks.
- boundary condition types
  - periodic boundary conditions
    - the value in the left **ghost cells** is copied from the right side of the **block**, and vice versa.
    - this is suitable for infinitely repeating systems, so any a small area can represent the whole system, so we need periodic boundary conditions to repeat.
  - outflow/zero-gradient boundary conditions
    - $$A_{i(0\leq i\leq n_g-1)} = A_i(n_g)$$
    - the value in the ghost cells is copied from the nearest cell in the block.
    - for velocity, this allows matter to flow in and out of the simulation region freely.
  - reflect boundary conditions
    - for scalar and tangential vectoe quantities
      - $$A_{i(0\leq i\leq n_g-1)} = A_i(n_g)$$
    - for normal vector quantities
      - $$A_{i(0\leq i\leq n_g-1)} = -A_i(n_g)$$
    - this means the flux of the blocks is zero, so the matter will be reflected back to the domain.
  - inflow/fixed-valur boundary conditions
    - $$A_{i(0\leq i\leq n_g-1)} = A_{fixed}$$
    - the value in the ghost cells is fixed to a constant value.
    - this means the simulation region is constantly ejected by a constant value(e.g. winds).
  - hydrostatic boundary conditions

        - $$\frac{dP}{dx} = -\rho g$$

          needs to be satisfied
        - in the ghost cells, the pressure in the center of the ghost cell is determined by the density and gravity value of the block boundary rather than the center of both cell.
- pgen -- problem generator -- xx.cpp
    - write the initial condition of the simulation.
    - ProblemGenerator -- place to write the initial condition.
    - here I can write the code to suit for my specific probelm, so I don't need to use his algorithm.
    - every time modify the cpp file, I need to recompile the code.