PHYS345 Numerical Project Proposal

Goal: To numerically solve some partial differential equations, or to investigate other numerically defined models.

For the partial differential equations a scalar field is defined over 1D, 2D, or 3D, and the evolution of that scalar field is calculated as a function of time based on a partial differential equation, with some initial conditions. The most interesting cases will be where there is a stochastic term, so that analytic solutions don't exist.

Heat equation

The general form of the heat equation including a convective term is

$$\frac{\partial T}{\partial t} = D\nabla^2 T + \nabla \cdot (\vec{v}T) + R(\vec{r}, t)$$

- $T(\vec{r}, t)$ represents the temperature at a given location in space and time, where D is the thermal diffusivity, and $\vec{v}$ is the velocity field the heat energy is moving with. The second term on the right-hand side represents *advection*.
- R represents any sources or sinks of heat energy.
- The case $\vec{v} = 0$, $R = 0$ is the usual heat equation.
- A radiative term could also be added.
- I may or may not have got the units/coefficients right in the above equation…
- See below for notes about solving such an equation.

Surface smoothing equation

Thin film growth with smoothing: The scalar field represents the surface height of a growing thin film $h(\vec{r}, t)$, where $\vec{r} = (x, y)$ represents the location on the substrate. The surface evolution can be modelled with the following equation.

$$\frac{\partial h}{\partial t} = \nu\nabla^2 h + \frac{\lambda}{2}(\nabla h)^2 + \gamma\nabla^4 h + \eta(\vec{r}, t).$$

This can model other similar situations, such as sediment falling to the bottom of a lake, snow falling on rough ground, the growth of the edge of a bacteria colony, etc.

- This is a stochastic version of the het equation with some extra terms.
- ν, λ, and γ are constants. Their sign and magnitude will determine whether the surface tends to smooth as it grows or to roughen.
- $\eta(\vec{r}, t)$ is a "white noise" term. It would represent the random arrival on the surface of depositing atoms. Generally $\langle \eta(\vec{r}, t) \rangle = 0$, although for film growth the average growth rate will not be zero, so $\langle \eta(\vec{r}, t) \rangle > 0$. The noise is completely uncorrelated, so $\langle \eta(\vec{r}, t)\eta(\vec{r}', t') \rangle = 2D\delta^d(\vec{r} - \vec{r}')\delta(t - t')$ (D is a constant, d the dimensionality).
- The case $\gamma = 0$ is called the Kardar-Parisi-Zhang equation (https://en.wikipedia.org/wiki/Kardar%E2%80%93Parisi%E2%80%93Zhang_equation). The non-linear term is an approximation that represent a situation where a surface grows outwards (or inwards) in a direction perpendicular to the surface rather than from the surface normal. This can lead to cusps, which occur for example in "sun cups"

(https://www.nationalgeographic.com/travel/article/your-shot-photo-of-the-month-icelandic-ice-cave) – the physics of these may or may not be described by the KPZ equation, but it certainly can predict the cusp features.

- The second and fourth order linear terms represent smoothing of the surface by diffusion of material along the surface. The second order term is obtained by assuming there is a net downhill current density $\vec{j} = -\nu\vec{\nabla}h$, so the continuity equation gives $\frac{\partial h}{\partial t} = -\vec{\nabla}\cdot\vec{j} = \nu\nabla^2 h$.
- The surface may be initially flat, or it may have initial features.
- Once a numerical solution is obtained over a sufficiently large surface area use the fast-Fourier-transform (in 2D) and plot the power spectral density. An equilibrium is reached between the smoothing terms and the roughening induced by the white noise. The power spectral density should be a power law after a long enough time, with the exponent dependent on which terms of the equation are included. The equilibrium situation (i.e., $PSD \sim q^{-\tau}$, with q the spatial frequency) will be reached most quickly for the high spatial frequency components.
- *Example method of solution in 1D when* $\lambda = \gamma = 0$: Define the initial height $h(x_i, t = 0)$ at each point on the surface, where x_i ranges from $x_0 = x_{min}$ to $x_N = x_{max}$ in uniform steps Δx. Now incrementally change the height at each point according to
$$\Delta h(x_i) = \left[\nu\left(\frac{d^2h}{dx^2}\right)_{x=x_i} + \eta\right]\Delta t\eta,$$
where η is a random number representing the noise term. The derivative is defined by the finite difference method:
$$\left(\frac{d^2h}{dx^2}\right)_{x=x_i} = \frac{h(x_{i-1}) - 2h(x_i) + h(x_{i+1})}{\Delta x^2}.$$
The best way to calculate this is to create two new arrays where all the h values are shifted one step in the positive or negative x direction, thereby defining $h(x_{i-1})$ and $h(x_{i+1})$. This is much more efficient than creating a loop to step through calculating Δh at each point x_i. Repeatedly update the surface height for as long as desired.
- For stability Δx and Δt must be suitably small.

Hunter-prey (Lotka-Volterra) equations, or closed film growth ones, fully vectorised

Coupled differential rate equations, e.g.,

$$\frac{dR}{dt} = \alpha R - \beta FR,$$

$$\frac{dF}{dt} = -\gamma F + \nu(\beta FR) = -\gamma F + \nu' FR.$$

Here $F(t)$ is the population of foxes, $R(t)$ the population of rabbits, α is the breeding rate for rabbits (baby rabbits per adult rabbit per unit time), and βFR is the rate at which foxes eat rabbits (β is some measure of how easy rabbits are to catch, and the rate of predation depends on how many rabbits there are to catch and how many foxes there are to catch them). The foxes have a death rate determined by the constant γ, but their breeding rate depends not just on how many foxes there are to breed but also on how much they get to eat, so it depends on R as well as F. The constant ν represents the breeding efficiency for foxes. The coefficient $\nu' \equiv \nu\beta$.

Other situations might require more coupled equations, e.g., modelling adatoms sticking and desorbing from step edges on a surface, where you need to know the adatoms density, the step-edge density, and some other parameters which I forget even though I wrote a paper about this…

It's important here to write efficient, vectorised code (I hope I remember how…).

Newton II

Write Newton’s law

$$\vec{F}_i(\vec{r}_i, \vec{v}_i, t) = m_i \vec{a}_i,$$

where $i = 1..N$ labels the particles, and for each particle the force depends on the position of that particle and its velocity, and it will depend on the position of every other particle if the particles interact. It may also depend explicitly on time, e.g., if you include an external driving force when modelling a mass bouncing on a spring. Example problems:

- Mass falling under gravity with drag (easy): $\vec{F}_i(\vec{r}_i, \vec{v}_i, t) = -mg + bv^2$
- A collision in 1D between two cars each with a spring on the front (i.e., an elastic collision). *I did this myself after noting that when a very heavy mass moving at v_i has an elastic collision with an initially stationary much lighter mass the light mass will move with final velocity twice the velocity of the heavy mass (the heavy mass is unaffected). I wondered how the light mass accelerates to $2v_i$ when it must break contact with the heavy mass as soon as it has a velocity infinitesimally larger than the heavy mass. The force between the cars should vanish as soon as they lose contact, and hence no more acceleration. It turns out the springs extend and contract such that they don’t lose contact with each other until the light mass reaches speed $2v_i$.*
- Mass orbiting a planet in 2D or 3D (using Newtonian gravity). Beware: the numerical solution will fail if you define the planet as a point object and let the orbiting mass get too close – the force will diverge.
- Mass orbiting two planets in 2D or 3D
- Several masses orbiting a planet in 2D or 3D

The solution again involves iteratively updating the position and velocity of the particle(s) using (in 1D):

$$x_i(t + \Delta t) = x_i(t) + v_i(t)\Delta t,$$

$$v_i(t + \Delta t) = v_i(t) + a_i(t)\Delta t,$$

$$a_i(t) = \frac{F_i(x_i, v_i, t)}{m_i}.$$

The genius of Newton is that having the force depend only on x_i and v_i (and t) closes the above set of equations – we always know where the particle is *now* (i.e., at time t) and how fast it is going, and that’s enough to calculate the force and hence to update the position and velocity.

Each numerical solution for the above examples will have regions of stability and instability. Test the stability criteria, both based on theory and trial and error (the nonlinear one might be the most difficult to test this for). ChatGPT won’t be too helpful at this testing, which is good, since it might be very good at writing the required code for the solutions.

Flock of birds/mosh pit/cellular automata

Simulate the motion of a group of “particles” (e.g., birds, bodies in a mosh pit…) which obey a set of simple rules, e.g.:

- The birds don’t collide
- The birds fly towards the centre of mass of their n nearest neighbours (e.g., $n = 5$)
- The birds try to match their velocity to the average of their n nearest neighbours

Treat the birds as point particles placed somewhere on a grid, and iteratively update their positions based on the rules. Figuring out which are the nearest neighbours for a given bird is tricky.

Quantum Espresso

Instal and run Quantum Espresso to calculate the electronic energy levels for some interesting molecules (e.g., H_2, methane, something more exotic), or even solids like silicon. I managed to install it and get it to run and used it to calculate the band structure of GaAs (with a lot of help from ChatGPT).

Magnetic dipoles

Simulate some interacting dipoles to find their lowest energy configuration.