

Ordinary Differential Equations

One dimensional integrators
Schrödinger equation
Root solvers

Summary

- ◆ Integration of ordinary differential equations
 - ◆ Initial value problem
- ◆ The 1-D stationary Schrödinger equation
 - ◆ Eigenvalue problem
 - ◆ Shooting method
- ◆ Root solvers
 - ◆ One dimension
 - ◆ Higher dimensions

The coffee cooling problem

- ◆ Maybe you came too late today because your 3pm coffee was still too hot
- ◆ Here we discuss how you can be in time
- ◆ The differential equation for cooling is

$$\frac{dT}{dt} = -\gamma(T - T_{room})$$

- ◆ Questions
 - ◆ How hot will the coffee be after 5 minutes?
 - ◆ How long will it take the coffee to cool to a drinkable temperature
 - ◆ Will it cool faster if you add the milk immediately or if you wait a while?

Solution by integration

- ◆ How hot will the coffee be after 5 minutes?
- ◆ This question can be answered by straight-forward analytical integration:

$$\frac{dT}{dt} = -\gamma(T - T_{room})$$

$$T = T_{room} + (T_0 - T_{room}) \exp(-\gamma t)$$

- ◆ However, doing this requires
 - ◆ an integration
 - ◆ a solution of a nonlinear equation
- ◆ For a general numeric solution this is awkward

Solution by finite difference equation

- ◆ We have already learned how to integrate such differential equations by replacing the differential equation by a difference equation

- ◆ The **Euler algorithm**

$$\frac{dT}{dt} = -\gamma(T - T_{room})$$

$$T(t + \Delta t) \approx T(t) + \Delta t \frac{dT}{dt} + O(\Delta t^2) = T(t) - \gamma \Delta t (T(t) - T_{room}) + O(\Delta t^2)$$

- ◆ Iterating this equation with a small time step Δt can be used to numerically solve the ODE
- ◆ How can it be improved?
 - ◆ Smaller time steps
 - ◆ Higher order methods

Predictor-corrector methods

- ◆ The Euler method for $\frac{dy}{dt} = f(y, t)$

- ◆ can be improved using a higher order scheme

$$y(t + \Delta t) \approx y(t) + \frac{\Delta t}{2} [f(y(t), t) + f(y(t + \Delta t), t + \Delta t)]$$

- ◆ But this is an implicit equation!

- ◆ We use a low order **predictor** step

$$\tilde{y}(t + \Delta t) = y(t) + f(y(t), t) \Delta t$$

- ◆ and then a higher order **corrector**

$$y(t + \Delta t) = y(t) + \frac{\Delta t}{2} [f(y(t), t) + f(\tilde{y}(t + \Delta t), t + \Delta t)]$$

- ◆ More iterations are also possible

2nd order Runge Kutta

- ◆ The Runge Kutta algorithm is a systematic improvement

$$y(t + \Delta t) = y(t) + \Delta t \cdot f(y(t + \Delta t/2), t + \Delta t/2) + O(\Delta t^3)$$

- ◆ The unknown $y(t + \Delta t/2)$ is estimated by Euler:

$$y(t + \Delta t/2) \approx y(t) + \frac{\Delta t}{2} \cdot f(y(t), t)$$

- ◆ Thus the 2nd order Runge Kutta algorithm is

$$k_1 = \Delta t \cdot f(y(t), t)$$

$$k_2 = \Delta t \cdot f(y(t) + k_1/2, t + \Delta t/2)$$

$$y(t + \Delta t) \approx y(t) + k_2$$

4th order Runge Kutta

- ◆ Usually at least a 4th order scheme is used

$$k_1 = \Delta t \cdot f(y(t), t)$$

$$k_2 = \Delta t \cdot f(y(t) + k_1/2, t + \Delta t/2)$$

$$k_3 = \Delta t \cdot f(y(t) + k_2/2, t + \Delta t/2)$$

$$k_4 = \Delta t \cdot f(y(t) + k_3, t + \Delta t)$$

$$y(t + \Delta t) \approx y(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

- ◆ The Runge Kutta schemes are not unique, which can be used to fine-tune the algorithm for the specific problem at hand

Initial and boundary value problems

- ◆ Initial value problems are easy to solve
 - ◆ Just start with the initial condition
 - ◆ Integrate the differential equation
- ◆ Boundary value problems are harder
 - ◆ $u' = f(u)$
 - ◆ with $u(0) = 0$ and $u(1) = 1$
 - ◆ Reformulate as system of two coupled equations

$$y_1' = y_2$$

$$y_2' = f(y_1)$$

- ◆ with $y_1(0) = 0$ and $y_1(1) = 1$
- ◆ Initial conditions are known for y_1 but not for y_2

Solution of the boundary value problem

- ◆ This boundary value problem can be solved by “**Shooting**”, a combination of integrator and root solver
 - ◆ Start with $y_1(0) = 0$ and $y_2(0) = \alpha$, an arbitrary number
 - ◆ Integrate (shoot) to the interval. This gives some value for $y_1(1; \alpha)$, depending on α .
 - ◆ Our aim is to find an α so that $y_1(1; \alpha) = 1$. We thus want to solve $y_1(1; \alpha) - 1 = 0$
 - ◆ This can be done by a numerical root solver

The 1D Schrödinger equation

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

- ◆ In the time independent case the ansatz

$$\psi(x, t) = \psi(x) \exp(-iEt)$$

- ◆ simplifies the Schrödinger equation to an ODE

$$E\psi = -\frac{\hbar^2}{2m} \frac{\partial^2 \psi}{\partial x^2} + V(x)\psi$$

- ◆ The time independent Schrödinger equation in one dimension can be solved with the methods discussed beforehand

Simplify the problem

- ◆ Integrate out all dimensions that can be solved exactly
 - ◆ For a two-particle scattering process this gives just a 1D problem
 - ◆ For a particle in a spherical potential this gives just a 1D problem
 - ◆ For plane wave propagation onto a barrier only one dimension is non-trivial
- ◆ Start integrating from the outside to the center
 - ◆ Asymptotic solution for large distances usually known
 - ◆ can be used to integrate from infinity to some small distance and to start numerically from there
 - ◆ can be used to fix the width of steps Δx so that the integrator is exact for large distances
 - ◆ Large errors are usually due to strong forces in the center. We want to keep these parts of the calculation at the end to avoid propagation of errors

Numerov algorithm

- ◆ is better than Runge Kutta for ODEs of the form

$$\psi''(x) + k(x)\psi(x) = 0$$

- ◆ this fits the Schrödinger equation with

$$k(x) = 2m(V(x) - E) / \hbar^2$$

- ◆ The Numerov algorithm is

$$\left(1 + \frac{\Delta x^2}{12} k(x + \Delta x)\right) \psi(x + \Delta x) = 2 \left(1 - \frac{5\Delta x^2}{12} k(x)\right) \psi(x) - \left(1 + \frac{\Delta x^2}{12} k(x - \Delta x)\right) \psi(x - \Delta x) + O(\Delta x^6)$$

- ◆ The derivation will be shown on the blackboard.

Initial conditions for Numerov

- ◆ The Numerov algorithm needs the wave function at two neighboring points to start
- ◆ For even potentials $V(x) = V(-x)$ we can make use of symmetries
 - ◆ For even parity wave functions
 - ◆ choose $\psi(\Delta x/2) = \psi(-\Delta x/2)$
 - ◆ For odd parity wave functions
 - ◆ choose $\psi(0) = 0, \psi(\Delta x) \neq 0$
- ◆ For general potentials
 - ◆ Use a high order method to estimate $\psi(\Delta x)$ from $\psi(0)$ and $\psi'(0)$

Eigenvalue problems

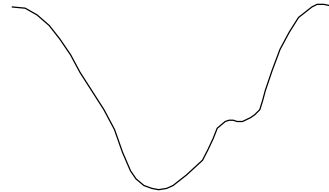
- ◆ If $E < V(\pm\infty)$ only bound states for certain energies E exist
They have to behave as $\psi(x) \rightarrow 0$ for $x \rightarrow \pm\infty$

- ◆ Again we use shooting:

- ◆ Select a guess E for the energy
- ◆ Choose a point a where $V(a) = E$
- ◆ Integrate ψ_l from $-\infty$ to a
- ◆ Integrate ψ_r from $+\infty$ to a
- ◆ Match the functions ψ_l and ψ_r . As we can multiply each wave function by an arbitrary constant the only nontrivial condition is

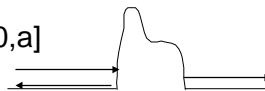
$$\frac{d \log \psi_l(a; E)}{dx} \equiv \frac{\psi_l'(a; E)}{\psi_l(a; E)} = \frac{\psi_r'(a; E)}{\psi_r(a; E)} = \frac{d \log \psi_r(a; E)}{dx}$$

- ◆ Use a root finder to solve for that equation to obtain E and ψ



Scattering problems

- ◆ Assume a potential $V(x) \neq 0$ only in $[0, a]$



- ◆ This time we can choose any energy $E = \hbar^2 k^2 / 2m$

- ◆ We have an incoming and reflected wave for $x < 0$:

$$\psi(x) = Ae^{ikx} + Be^{-ikx}$$

- ◆ And a transmitted wave for $x > a$

$$\psi(x) = Ce^{ikx}$$

- ◆ We set $C=1$ and integrate the transmitted wave back through the potential to the other side.

- ◆ Matching to a plane wave gives A and B
- ◆ From A and B we can then transmissivity and reflectivity

Root solvers and Minimization

- ◆ Any root solver to solve

$$\vec{F}(\vec{x}) = 0$$

- ◆ can be used to minimize $g(x)$ by solving for

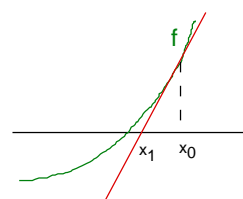
$$\nabla g(\vec{x}) = 0$$

- ◆ We will now discuss
 - ◆ One-dimensional root solvers
 - ◆ Higher dimensional root solvers
 - ◆ Higher dimensional optimization algorithms
- ◆ Be warned:
 - ◆ Do not code yourself
 - ◆ Use existing libraries

Newton and secant methods

- ◆ Should be well known
 - ◆ We start from a guess x_0
 - ◆ Linearize $f(x)$ around x_0
 - ◆ Find the root of the linearized equation

$$0 = f(x_0) + (x - x_0) f'(x_0)$$



- ◆ Take this root as new guess:

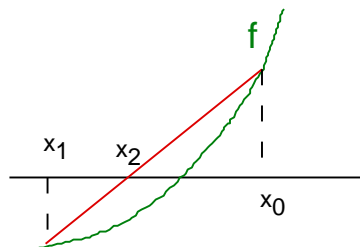
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- ◆ If we do not know $f'(x)$ we can replace it by a numerical estimate from a finite difference of the last two guesses This **secant method** needs two start values

$$x_{n+1} = x_n - (x_n - x_{n-1}) \frac{f(x_n)}{f(x_n) - f(x_{n-1})}$$

Regula falsi

- ◆ Uses two starting points that bracket the root
 - ◆ $f(x_0) > 0$
 - ◆ $f(x_1) < 0$
- ◆ Determine the point x_2 , where the line connecting the two points crosses the x-axis
- ◆ Repeat the procedure for
 - ◆ x_0 and x_2 if $f(x_2) < 0$
 - ◆ x_2 and x_1 if $f(x_2) > 0$
- ◆ In contrast to Newton, this will **always** find a root!



Higher dimensional Newton

- ◆ The Newton method can be generalized to higher dimensions:

$$\vec{f}(\vec{x}) = 0$$

- ◆ Define the matrix of derivatives

$$A_{ij}(\vec{x}) = \frac{\partial f_i(\vec{x})}{\partial x_j}$$

- ◆ And use a higher dimensional Newton estimate:

$$\vec{x} \rightarrow \vec{x} - A^{-1} \vec{f}(\vec{x})$$

- ◆ Again a numerical derivative can be used instead

$$A_{ij}(\vec{x}) = \frac{f_i(\vec{x} + h_j \vec{e}_j) - f_i(\vec{x})}{h_j} \quad \text{with} \quad h_j \approx x_j \sqrt{\epsilon}$$

Steepest descent

- ◆ Is a very simple minimization algorithm
 - ◆ Determine the gradient of the function at the guess x_0
 - ◆ Minimize the 1-D problem in the direction of the gradient

$$g(\bar{x}_0 + \lambda \nabla g(\bar{x}_0))$$

- ◆ Repeat the procedure until a global minimum is found
- ◆ It is simple but not very efficient
- ◆ The standard problem is a narrow but steep valley
 - ◆ It always minimizes in the direction perpendicular to the valley
 - ◆ It hardly minimizes along the direction of the valley

Conjugate gradient

- ◆ is a method that usually performs better
- ◆ The problem is approximated locally by a *parabola*
- ◆ The minimum of this parabola is found and used as a new starting point
- ◆ This can solve a steep and narrow parabolic valley problem in **one** step!