Optimization, or finding minima of functions

Given a function $f(\vec{r})$, find $\vec{r}$ that minimizes it. Local vs. global optimization.

At minima, $\nabla f(\vec{r}) = 0.$ A nonlinear equation that can be solved. But there are reasons to consider the optimization problem separately.

Some of the methods parallel those for root-finding.

1D methods:

Golden section search – analog of bisection.

Successive parabolic interpolation – analog of secant and false direction.

Less reliable. Should be combined with golden section. Brent method.

Newton's method – application of Newton-Raphson.

Higher-dimensional methods:

Methods that do not require derivatives. Nelder-Mead.

Steepest descent and its shortcomings. Conjugate gradient.

Newton method can be used in multidimensions in principle, but will require a Hessian. For roots, we had

$$\vec{x}_{n+1} = \vec{x}_n - \boldsymbol{J}^{-1}\vec{f} \qquad \boldsymbol{J} = \left(\frac{\partial f_i}{\partial x_j}\right)$$

$$f_i \to \frac{\partial f}{\partial x_i}$$

$$\vec{x}_{n+1} = \vec{x}_n - \boldsymbol{H}^{-1}\nabla f \qquad \boldsymbol{J} = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right) = \boldsymbol{H}$$

Can calculate $\boldsymbol{H}^{-1}\nabla f$ by solving the linear system iteratively. Do not need to converge. E.g., conjugate gradient iterations.

Quasi-Newton methods estimate the Hessian instead of calculating it accurately, updating the estimate at every step.

Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

Solve $\boldsymbol{H}_k \vec{s}_k = -\nabla f(\vec{x}_k)$

$\vec{x}_{k+1} = \vec{x}_k + \vec{s}_k$ (or an exact line search, $\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{s}_k$ and then $\vec{s}_k \leftarrow \alpha_k \vec{s}_k$)

$$\vec{y}_k = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$$

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + (\vec{y}_k \otimes \vec{y}_k)/(\vec{y}_k \cdot \vec{s}_k) - (\boldsymbol{H}_k \cdot \vec{s}_k) \otimes (\boldsymbol{H}_k \cdot \vec{s}_k)/(\vec{s}_k \cdot \boldsymbol{H}_k \cdot \vec{s}_k)$$

Satisfies $\boldsymbol{H}_{k+1}(\vec{x}_{k+1} - \vec{x}_k) = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$.

Constrained optimization

Minimize $f(\vec{r})$ subject to additional constraints. Equality constraints $\vec{g}(\vec{r})=\vec{0}$

Minimize Lagrangian function $L(\vec{r},\vec{\lambda})=f(\vec{r})+\vec{\lambda}\cdot\vec{g}(\vec{r})$.

$\nabla L_{\vec{r}}(\vec{r})=\nabla f(\vec{r})+\lambda_i \nabla g_i(\vec{r})=0.$ ∇f is in the space spanned by ∇g_i.

$\nabla L_{\vec{\lambda}}(\vec{r})=\vec{g}(\vec{r})=\vec{0}$ There are special methods.

Inequality constraints. $h(\vec{r})\geq 0.$ At any given iteration, can check if the constraint is being violated – if so, switch to treating it as an equality constr.

Linear programming

Linear function + linear inequality constraints: $\max\ \vec{c}\cdot\vec{r},\ \boldsymbol{A}\vec{r}\leq\vec{b},\ \vec{r}\geq\vec{0}$

The inequality describes a polytope. The solution is one of the vertices.

Combinatorial optimization. Simplex method. Even though the number of vertices is exponential, there are polynomial methods. Not always the case, e.g., the famous traveling salesman problem.

Summary

1. Errors.

   Round-off errors and those associated with approximations of numerical algorithms (truncation, discretization, algorithmic error)

2. Numerical differentiation.

3. Interpolation.

   Polynomial. Runge phenomenon, Chebyshev polynomials. Splines

4. Numerical integration.

   Newton-Cotes, Clenshaw-Curtis, Gauss

5. Discrete Fourier transforms, FFT algorithm

   Trigonometric interpolation.

6. Solving nonlinear equations (root-finding)

Summary

7. ODEs, initial value problems.

   Euler, Runge-Kutta, stability, explicit and implicit methods, stiff problems, symplectic algorithms, molecular dynamics.

8. Nonlinear dynamics.

   Iterated maps, bifurcations, chaos, attractors, fractals, etc.

9. ODEs, boundary value problems.

   Eigenvalue problems. Shooting, finite differences, finite elements.

10. Linear algebra.

    Solving linear systems, eigenvalues, SVD, direct and iterative methods

11. PDEs.

    Elliptic, parabolic, hyperbolic, method of lines, finite difference, finite volume, finite element, boundary element

12. Optimization

These are methods for solving **mathematical** problems. Obviously, relevant for physics (we have seen that in homework problems), but there is not much physics in the methods themselves and, of course, they can be applied outside physics.

However, there are also methods designed specifically to solve physics problems using physically motivated approximations and assumptions. Often, such methods are necessary when the straightforward formulation of the mathematical problem is computationally too hard.

Brief overview of major applications of computational methods in physics and associated methods. Organized hierarchically, from the deepest, most fundamental, microscopic level to the most “coarse-grained”, macroscopic level.

On the most fundamental level, we have elementary particles described by quantum field theories (QFT), such as quantum electrodynamics (QED) and quantum chromodynamics (QCD).

One way to formulate and solve quantum problems is through the path-integral formulation. In quantum mechanics, the probability to go from point x_i to point x_f is given by an integral over all possible trajectories between these points: $\int [Dx(t)] e^{-iS/\hbar}$ Action $S = \int_{t_i}^{t_f} dt\, L(x, \dot{x}, t)$

In QFT, we deal with fields instead of particles. Suppose our field is $f(\vec{r}, t)$. So an analog of path integrals would be $\int [Df(\vec{r}, t)] e^{-iS/\hbar}$

$$S = \int dt\, d\vec{r}\, L(f, \dot{f}, \nabla f) \qquad \langle F \rangle = \frac{\int Df\, F[f] e^{-iS/\hbar}}{\int Df\, e^{-iS/\hbar}}$$

To do these infinite-dimensional integrals, need to discretize them on a lattice.

By switching to imaginary time, $e^{-iS/\hbar} \to e^{-S/\hbar}$ Similar to partition function

Monte Carlo methods. **Lattice QCD** community is at the forefront of developing new MC methods.

On the next level, we have ordinary quantum mechanics. Describe atoms, molecules, solids. Solving the Schrodinger equation directly using the methods we have studied in this course is only possible for very small systems (up to 10 particles at most). Need to make a number of approximations.

Born-Oppenheimer approximation

Heavy nuclei, light electrons.

$$H=\sum_{i=1}^{N}\frac{p_i^2}{2m}+\sum_{n=1}^{K}\frac{P_n^2}{2M_n}+\frac{1}{2}\sum_{i\neq j}\frac{e^2}{|\vec{r}_i-\vec{r}_j|}-\sum_{i,n}\frac{Z_n e^2}{|\vec{r}_i-\vec{R}_n|}+\frac{1}{2}\sum_{n\neq n'}\frac{Z_n Z_{n'} e^2}{|\vec{R}_n-\vec{R}_{n'}|}$$

Represent $\psi(\{\vec{r}_i\},\{\vec{R}_n\})=\psi_1(\{\vec{R}_n\})\psi_2(\{\vec{r}_i\},\{\vec{R}_n\})$

Neglect action of $P_n^2/(2M_n)$ on ψ_2. Then in the equation for ψ_2 there are no R derivatives, R enter as parameters. Corresponds to solving the electron problem for fixed nuclei. Then the equation for ψ_2 corresponds to nuclei moving in the field created by electrons. This second problem can often be considered classically.

Hartree-Fock approximation

$$\left(-\Sigma_i \frac{\hbar^2}{2m}\nabla_i^2+\frac{1}{2}\Sigma_{i\neq j}\frac{e^2}{|\vec{r}_i-\vec{r}_j|}-\Sigma_{i,n}\frac{Z_n e^2}{|\vec{r}_i-\vec{R}_n|}\right)\psi(\vec{r}_1,\vec{r}_2,..,\vec{r}_N)=E\,\psi(\vec{r}_1,\vec{r}_2,..,\vec{r}_N)$$

If the electrons did not interact, the wave function would be given by (forgetting for a moment about Pauli principle):

$$\psi(\vec{r}_1,\vec{r}_2,..,\vec{r}_N)=\psi_1(\vec{r}_1)\psi_2(\vec{r}_2)...\psi_N(\vec{r}_N)$$

After antisymmetrization,

$$\psi(\vec{r}_1,\vec{r}_2,..,\vec{r}_N)=\frac{1}{\sqrt{N!}}\begin{vmatrix}\psi_1(\vec{r}_1) & \psi_2(\vec{r}_1) & ... & \psi_N(\vec{r}_1)\\ \psi_1(\vec{r}_2) & \psi_2(\vec{r}_2) & ... & \psi_N(\vec{r}_2)\\ ... & ... & ... & ...\\ \psi_1(\vec{r}_N) & \psi_2(\vec{r}_N) & ... & \psi_N(\vec{r}_N)\end{vmatrix}$$

Variational formulation: the ground state wave function minimizes

$$\int \psi^{\dagger} H\,\psi\,dV \quad \text{fixing} \int \psi^{\dagger}\psi\,dV=1$$

$$\left[-\frac{\hbar^2}{2m}-\Sigma_n\frac{Z_n}{|\vec{r}-\vec{R}_n|}\right]\psi_k+\Sigma_{l=1}^{N}\int dV'|\psi_l(\vec{r}\,')|^2\frac{1}{|\vec{r}-\vec{r}\,'|}\psi_k(\vec{r}\,')$$

$$-\Sigma_{l=1}^{N}\int dV'\psi_l^*(\vec{r}\,')\frac{1}{|\vec{r}-\vec{r}\,'|}\psi_k(\vec{r}\,')\psi_l(\vec{r})=\epsilon_k\psi_k$$

A nonlinear integral equation, but at least it's only 3D.

$$\left[-\frac{\hbar^2}{2m}-\Sigma_n\frac{Z_n}{|\vec{r}-\vec{R}_n|}\right]\psi_k+\Sigma_{l=1}^{N}\int dV'|\psi_l(\vec{r}\,')|^2\frac{1}{|\vec{r}-\vec{r}\,'|}\psi_k(\vec{r}\,')$$

$$-\Sigma_{l=1}^{N}\int dV'\psi_l^*(\vec{r}\,')\frac{1}{|\vec{r}-\vec{r}\,'|}\psi_k(\vec{r}\,')\psi_l(\vec{r})=\epsilon_k\psi_k$$

To solve numerically, it is common to expand the wave functions in a basis:

$$\psi_k(\vec{r})=\Sigma_{p=1}^{M}C_{pk}\chi_p(\vec{r})$$

Eigenvalue problem (nonlinear, because ***F*** depends on ***C***):

$$\boldsymbol{F}\vec{C}_k=\epsilon_k\boldsymbol{S}\vec{C}_k$$

Solved iteratively.

Want a small number of functions to describe the wave function accurately.

But also preferable to be able to calculate integrals analytically. Tradeoff. STO (like H atom) vs. GTO functions. Special notation.

Going beyond Hartree-Fock: post-HF methods. Møller-Plesset perturbation theory (MP2, MP4), coupled-cluster methods.

Another approach, more popular in physics (vs. chemistry): density-functional theory (DFT).

Hohenberg-Kohn theorem: for a system of electrons in an external potential $V(\vec{r})$, there exists a unique density functional F[n] independent of $V(\vec{r})$ such that $E[n]=F[n]+\int n(\vec{r})U(\vec{r})dV$ is minimized by the density corresponding to the ground state. Unfortunately, the exact F[n] is unknown.

Kohn-Sham equation

$$\left[-\frac{\hbar^2}{2m}\nabla^2+V_{\text{eff}}(\vec{r})\right]\psi_k(\vec{r})=\epsilon_k\psi_k(\vec{r})$$

$$V_{\text{eff}}(\vec{r})=V(\vec{r})+\int n(\vec{r}\,')\frac{1}{|\vec{r}-\vec{r}\,'|}dV'+\frac{\delta E_{\text{xc}}[n]}{\delta n} \qquad n(\vec{r})=\Sigma_{k=1}^{N}|\psi_k(\vec{r})|^2$$

The simplest approximation for E_{xc}: local density approximation (LDA). Take density gradients into account: GGA.

Strictly speaking, only valid for the ground state. TDDFT.

One additional approximation: only valence electrons are considered explicitly. It is assumed that the core electron orbitals remain the same as in the isolated atom, so they can just be represented as an effective potential that is calculated once for a given atom type.

For a deep singular potential like Coulomb, the wave function would oscillate rapidly near the centre. If, e.g., the plane wave basis is used, need too many plane waves. Replace by a more shallow and flat **pseudopotential** such that at large distances the behaviour of the wave function is the same, but there are no oscillations near the centre.

Some heavy atoms require relativistic calculations.

Calculating the ground-state energy for electrons gives the potential for nuclei. Not necessary to pre-calculate – the potential and the forces can be obtained on the fly. **Car-Parrinello method**.

Car-Parrinello is one way to do molecular dynamics (MD). But it can also be done with pre-determined potentials, either calculated quantum-mechanically, obtained indirectly from experiments, or just guessed.

Particularly important for simulating soft matter, including biomolecules. Different levels of approximation and coarse-graining. Does not have to be atomistic: a single atom can correspond to a group of atoms (e.g., an amino acid residue in a protein).

Solvent effects. Hydrophilic/hydrophobic effect, hydrodynamic effects. Very costly to simulate explicitly. Can be represented implicitly, by modifying the interactions. Or various simplified representations: dissipative particle dynamics (DPD); multi-particle collision dynamics (MPCD); lattice Boltzmann (LB).

Monte Carlo approach: faster than MD but dynamics is often incorrect.

Fluid dynamics

$$\rho\left(\frac{\partial \vec{v}}{\partial t}+(\vec{v}\cdot\nabla)\vec{v}\right)=-\nabla p+\eta\nabla^2\vec{v}+\left(\frac{1}{3}\eta+\zeta\right)\nabla(\nabla\cdot\vec{v})+\vec{f}$$

$$\frac{\partial \rho}{\partial t}+\nabla\cdot(\rho\vec{v})=0 \qquad\qquad p=p(\rho)$$

Mostly finite volume methods, finite element also popular. Finite difference used in astrophysics applications.

Incompressible flow. ρ = const. 4 equations, 4 unknowns, but there is no "evolution equation" for *p*. Velocity-pressure coupling. Start with some guess, calculate the velocity, if continuity not satisfied, then if too much flows in, the pressure should be increased, if too much flows out, the pressure should be reduced.

Turbulent flow modeling

At large velocities flow becomes chaotic. This is characterized by the Reynolds number:

$$\mathrm{Re} = \frac{\rho v L}{\eta}$$

Transition occurs at Re ~ 1000. Features present on a multitude of scales.

Discretization has to capture all of these scales. An estimate of the necessary number of nodes is $10^4 \mathrm{Re}^{8/5}$. For cars and trucks Re~10^6-10^7, for planes ~ 10^7-10^8, for ships ~ 10^8-10^9. So doing direct numerical simulation (DNS) of turbulence is totally unrealistic in these cases.

Fortunately, small-scale features are rather universal, so they can be taken into account indirectly. 2 approaches: Reynolds-averaged Navier-Stokes (RANS) and Large Eddy Simulation (LES).

A few additional things that one should be aware of:

Basic algorithms and data structures.

Sorting: the most straightforward algorithms are $O(N^2)$, but there are some (Quicksort, Heapsort) that are O(NlogN). So, if you need to sort a large set of data, keep this in mind.

Data structures: ways to arrange your data so you can use them easily. For instance, one of the simplest data structures is an array. While it is easy to append to an array, it is harder to delete without creating a gap or insert in the middle. This is what (singly or doubly) **linked lists** are for. Various kinds of **trees** are convenient for organizing data for easy sorting and searching.

Code optimization

It used to be the case that you had to keep in mind the relative cost of different operations and avoid more costly ones (strength reduction) or eliminating common subexpressions. These days, compilers are likely to take care of that, so you don't have to do this, if it produces less readable code. BUT you need to turn on optimization flags when compiling! I.e., at least -O.

One area where human optimization is still necessary is memory access. Cache access is fast while main memory access is slow (and disk is even slower). So it may make sense to try to use the data in the cache as much as possible before they are replaced by something else. But it only really matters for large problems.

One curious thing often mentioned in connection with this is different order of storage of multidimensional arrays in Fortran and in C: row-major in C (i.e., a[0][0], a[0][1], ...) vs. column-major in Fortran [a(1,1), a(2,1), ...]

Parallel computing

For some problems, trivial parallelism (e.g., integration). More often, different processors need to share information.

Different types of parallel computers. Multiple instruction, multiple data (MIMD) vs. Single-instruction, multiple data (SIMD). GPUs belong to the latter class.

Distributed-memory vs. shared-memory.

For distributed-memory machines, the standard approach is the message passing paradigm. Message Passing Interface (MPI).

For shared-memory machines, OpenMP.