Slides from FYS4411 Lectures

Morten Hjorth-Jensen

[1]Department of Physics and Center of Mathematics for Applications
University of Oslo, N-0316 Oslo, Norway

Spring 2010

Topics for Week 3, January 18-2

Introduction, Parallelization, MPI and Variational Monte Carlo

- Presentation of topics to be covered and introduction to Many-Body physics (Lecture notes chapter 16, Raimes chapter 1 or Thijssen chapter 4).
- Variational Monte Carlo theory and presentation of project 1. (lecture notes chapter 11, Thijssen chapter 12)
- Introduction to Message Passing Interface (MPI) and parallelization. (lecture notes chapter 7.7)
- Assignment for next week: study chapter 11 of Lecture notes or Chapter 12 of Thijssen.

18 January - 31 May

Course overview, Computational aspects

- Parallelization (MPI), high-performance computing topics and object orientation. Choose between F95 and/or C++ as programming languages. Python also possible as programming language. (all projects)
- Algorithms for Monte Carlo Simulations (multidimensional integrals), Metropolis-Hastings and importance sampling algorithms. Improved Monte Carlo methods (project 1)
- Statistical analysis of data from Monte Carlo calculations, blocking method. (project 1)

18 January - 31 May

Course overview, Computational aspects

- Search for minima in multidimensional spaces (conjugate gradient method) (project 1)
- Object orientation (both projects)
- Solutions of coupled differential equations for Hartree-Fock and density functional calculations. (project 2)
- Alternativ project 2: Lattice quantum chromodynamics or path integral Monte Carlo (Many-body physics at finite temperature)

18 January -31 May, project 1

Quantum Mechanical Methods and Systems

1. Variational Monte Carlo for 'ab initio' studies of quantum mechanical many-body systems.
2. Simulation of atoms like Helium, Beryllium and Neon with extensions to solids. It has also to be extended to two-dimensional systems like quantum dots.
3. Aim of projects 1: understand how to simulate qauntum mechanical systems with many interacting particles using variational Monte Carlo methods.

The methods of projects 1 and 2 are relevant for atomic, molecular,solid state, materials science, nanotechnology, quantum chemistry and nuclear physics.

18 January -31 May, project 2

Quantum Mechanical Methods and Systems

1. Project 2 (standard development) solves much of the same systems as in project 1 but introduces Hartree-Fock theory and density functional theory.
2. The Hartree-Fock solutions are in turn used in the code from project 1 to obtain an ab initio solution for a given system
3. This solution is then used to constrain a density functional (actual research).
4. We will also end up writing a density functional code and use this to compute properties of solids (atoms in a lattice).

DFT and HF are covered by the lectures notes, chapters 4-6 of Thijssen and the articles of Jones on the webpage of the course.

18 January -31 May, project 2

Quantum Mechanical Methods and Systems

1. Project 2 is however not yet determined. Depending on the interest of the participants we may extend project 1 to deal with path integral Monte Carlo methods. This is relevant for studies of quantum mechanical systems at finite temperature and for example lattice quantum chromodynamics. Open for discussions.

18 January -31 May

Projects, deadlines and oral exam

1. Deadline project 1: March 22
2. Deadline project 2: 31 May
3. Oral exam: week 24 (8-12 June), most likley Friday June 11.

The oral exam is based on your presentation of the projects.

18 January -31 May

More on projects

1. Keep a logbook, important for keeping track of all your changes etc etc.
2. The projects should be written as a regular scientific article, with introduction, formalism, codes which have been developed and discussion of results. Conclusions and references should also be included. An example can be found on the webpage of the course.
3. The link with the article example contains also an article on how to use latex and write good scientific articles!

Lectures and ComputerLab

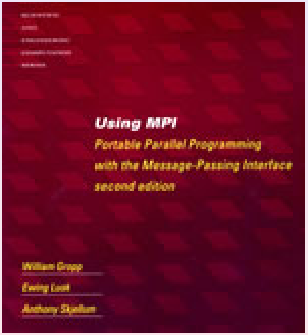
- Lectures: Thursday (14.15-16, room FV329)
- Detailed lecture notes, all programs presented and projects can be found at the homepage of the course.
- Computerlab: 16-19 thursday, room FV329
- Weekly plans and relevant information are on the official webpage.
- Chapters 8, 9, 11 and 16 and 17 of the FYS3150/4150 lecture notes give a good starting point. We recommend also J. M. Thijssen text *Computational Physics* and the text of Raimes as background. For MPI we recommend Gropp, Lusk and Sjellum's text.

Thijssen's text

J. M. Thijssen's text

- Computational Physics
- Chapters 3-6 and 12, possibly also chapter 8-9
- see `http://www.tn.tudelft.nl/tn/People/Staff/Thijssen/comphybook.html`

MPI text


Using MPI
Portable Parallel Programming
with the Message-Passing Interface
second edition
William Gropp
Ewing Lusk
Anthony Skjellum

Gropp, Lusk and Sjellum

- Using MPI
- Chapters 1-5
- see `http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=10761`

Selected Texts and lectures on C/C++

- J. J. Barton and L. R. Nackman,*Scientific and Engineering C++*, Addison Wesley, 3rd edition 2000.
- B. Stoustrup, *The C++ programming language*, Pearson, 1997.
- George Em Karniadakis and Robert M. Kirby II, *Parallel Scientific Computing in C++ and MPI* `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521520805`
- D. Yang, *C++ and Object-oriented Numeric Computing for Scientists and Engineers*, Springer 2000.
- More books reviewed at `http:://www.accu.org/` and `http://www.comeaucomputing.com/booklist/`

Definitions and notations

The Schrödinger equation reads

$$\hat{H}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)\Psi_\lambda(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = E_\lambda \Psi_\lambda(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N), \tag{1}$$

where the vector $\mathbf{r}_i$ represents the coordinates (spatial and spin) of particle i, λ stands for all the quantum numbers needed to classify a given N-particle state and Ψ_λ is the pertaining eigenfunction. Throughout this course, Ψ refers to the exact eigenfunction, unless otherwise stated.

Definitions and notations

We write the Hamilton operator, or Hamiltonian, in a generic way

$$\hat{H} = \hat{T} + \hat{V}$$

where $\hat{T}$ represents the kinetic energy of the system

$$\hat{T} = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m_i} = \sum_{i=1}^{N} \left(-\frac{\hbar^2}{2m_i} \nabla_{\mathbf{i}}{}^2 \right) = \sum_{i=1}^{N} t(\mathbf{r}_i)$$

while the operator $\hat{V}$ for the potential energy is given by

$$\hat{V} = \sum_{i=1}^{N} u(\mathbf{r}_i) + \sum_{ji=1}^{N} v(\mathbf{r}_i, \mathbf{r}_j) + \sum_{ijk=1}^{N} v(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k) + \dots \quad (2)$$

Hereafter we use natural units, viz. $\hbar = c = e = 1$, with e the elementary chargeand c the speed of light. This means that momenta and masses have dimension energy.

Definitions and notations

If one does quantum chemistry, after having introduced the Born-Oppenheimer approximation which effectively freezes out the nucleonic degrees of freedom, the Hamiltonian for $N = n_e$ electrons takes the following form

$$\hat{H} = \sum_{i=1}^{n_e} t(\mathbf{r}_i) - \sum_{i=1}^{n_e} k\frac{Z}{r_i} + \sum_{i<j}^{n_e} \frac{k}{r_{ij}},$$

with $k = 1.44$ eVnm

Definitions and notations

We can rewrite this as

$$\hat{H} = \hat{H}_0 + \hat{H}_1 = \sum_{i=1}^{n_e} \hat{h}_i + \sum_{i<j=1}^{n_e} \frac{1}{r_{ij}}, \tag{3}$$

where we have defined $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ and

$$\hat{h}_i = t(\mathbf{r}_i) - \frac{Z}{r_i}. \tag{4}$$

The first term of eq. (3), H_0, is the sum of the A or n *one-body* Hamiltonians $\hat{h}_i$. Each individual Hamiltonian $\hat{h}_i$ contains the kinetic energy operator of an electron and its potential energy due to the attraction of the nucleus. The second term, H_1, is the sum of the $n_e(n_e - 1)/2$ two-body interactions between each pair of electrons. Note that the double sum carries a restriction $i < j$.

Definitions and notations

The potential energy term due to the attraction of the nucleus defines the onebody field $u_i = u(\mathbf{r}_i)$ of Eq. (2). We have moved this term into the $\hat{H}_0$ part of the Hamiltonian, instead of keeping it in $\hat{V}$ as in Eq. (2). The reason is that we will hereafter treat $\hat{H}_0$ as our non-interacting Hamiltonian. For a many-body wavefunction Φ_λ defined by an appropriate single-particle basis, we may solve exactly the non-interacting eigenvalue problem

$$\hat{H}_0\Phi_\lambda = e_\lambda\Phi_\lambda,$$

with e_λ being the non-interacting energy. This energy is defined by the sum over single-particle energies to be defined below. For atoms the single-particle energies could be the hydrogen-like single-particle energies corrected for the charge Z. For nuclei and quantum dots, these energies could be given by the harmonic oscillator in three and two dimensions, respectively.

Definitions and notations

We will assume that the interacting part of the Hamiltonian can be approximated by a two-body interaction. This means that our Hamiltonian is written as

$$\hat{H} = \hat{H}_0 + \hat{H}_1 = \sum_{i=1}^{N} h_i + \sum_{i<j=1}^{N} V(r_{ij}), \tag{5}$$

with

$$H_0 = \sum_{i=1}^{N} h_i = \sum_{i=1}^{N} \left(t(\mathbf{r}_i) + u(\mathbf{r}_i)\right). \tag{6}$$

The onebody part $u(\mathbf{r}_i)$ is normally approximated by a harmonic oscillator potential or the Coulomb interaction an electron feels from the nucleus. However, other potentials are fully possible, such as one derived from the self-consistent solution of the Hartree-Fock equations.

Definitions and notations

Our Hamiltonian is invariant under the permutation (interchange) of two particles. Since we deal with fermions however, the total wave function is antisymmetric. Let $\hat{P}$ be an operator which interchanges two particles. Due to the symmetries we have ascribed to our Hamiltonian, this operator commutes with the total Hamiltonian,

$$[\hat{H}, \hat{P}] = 0,$$

meaning that $\Psi_\lambda(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N)$ is an eigenfunction of $\hat{P}$ as well, that is

$$\hat{P}_{ij}\Psi_\lambda(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_i, \dots, \mathbf{r}_j, \dots, \mathbf{r}_N) = \Psi_\lambda(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_j, \dots, \mathbf{r}_i, \dots, \mathbf{r}_N).$$

We have introduced the suffix *ij* in order to indicate that we permute particles *i* and *j*. The Pauli principle tells us that the total wave function for a system of fermions has to be antisymmetric. What does that mean for the above permutation?

Definitions and notations

In our case we assume that we can approximate the exact eigenfunction with a Slater determinant

$$\Phi(\mathbf{r}_1,\mathbf{r}_2,\dots,\mathbf{r}_N,\alpha,\beta,\dots,\sigma) = \frac{1}{\sqrt{N!}}\left|\begin{array}{ccccc} \psi_\alpha(\mathbf{r}_1) & \psi_\alpha(\mathbf{r}_2) & \dots & \dots & \psi_\alpha(\mathbf{r}_N) \\ \psi_\beta(\mathbf{r}_1) & \psi_\beta(\mathbf{r}_2) & \dots & \dots & \psi_\beta(\mathbf{r}_N) \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \psi_\sigma(\mathbf{r}_1) & \psi_\sigma(\mathbf{r}_2) & \dots & \dots & \psi_\gamma(\mathbf{r}_N) \end{array}\right|, \quad (7)$$

where $\mathbf{r}_i$ stand for the coordinates and spin values of a particle *i* and $\alpha,\beta,\dots,\gamma$ are quantum numbers needed to describe remaining quantum numbers.

Definitions and notations

The single-particle function $\psi_\alpha(\mathbf{r}_i)$ are eigenfunctions of the onebody Hamiltonian h_i, that is

$$h_i = h(\mathbf{r}_i) = t(\mathbf{r}_i) + u(\mathbf{r}_i),$$

with eigenvalues

$$h_i\psi_\alpha(\mathbf{r}_i) = t(\mathbf{r}_i) + u(\mathbf{r}_i)\psi_\alpha(\mathbf{r}_i) = \varepsilon_\alpha\psi_\alpha(\mathbf{r}_i).$$

The energies ε_α are the so-called non-interacting single-particle energies, or unperturbed energies. The total energy is in this case the sum over all single-particle energies, if no two-body or more complicated many-body interactions are present.

Definitions and notations

Let us denote the ground state energy by E_0. According to the variational principle we have

$$E_0 \leq E[\Phi] = \int \Phi^* \hat{H} \Phi d\tau$$

where Φ is a trial function which we assume to be normalized

$$\int \Phi^* \Phi d\tau = 1,$$

where we have used the shorthand $d\tau = d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_N$.

Definitions and notations

In the Hartree-Fock method the trial function is the Slater determinant of Eq. (7) which can be rewritten as

$$\Psi(\mathbf{r}_1,\mathbf{r}_2,\dots,\mathbf{r}_N,\alpha,\beta,\dots,\nu) = \frac{1}{\sqrt{N!}}\sum_P (-)^P\hat{P}\psi_\alpha(\mathbf{r}_1)\psi_\beta(\mathbf{r}_2)\dots\psi_\nu(\mathbf{r}_N) = \sqrt{N!}\mathcal{A}\Phi_H, \tag{8}$$

where we have introduced the antisymmetrization operator $\mathcal{A}$ defined by the summation over all possible permutations of two nucleons.

Definitions and notations

It is defined as

$$\mathcal{A} = \frac{1}{N!}\sum_{p}(-)^p\hat{P}, \tag{9}$$

with p standing for the number of permutations. We have introduced for later use the so-called Hartree-function, defined by the simple product of all possible single-particle functions

$$\Phi_H(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N, \alpha, \beta, \dots, \nu) = \psi_\alpha(\mathbf{r}_1)\psi_\beta(\mathbf{r}_2)\dots\psi_\nu(\mathbf{r}_N).$$

Definitions and notations

Both $\hat{H}_0$ and $\hat{H}_1$ are invariant under all possible permutations of any two particles and hence commute with $\mathcal{A}$

$$[H_0, \mathcal{A}] = [H_1, \mathcal{A}] = 0. \tag{10}$$

Furthermore, $\mathcal{A}$ satisfies

$$\mathcal{A}^2 = \mathcal{A}, \tag{11}$$

since every permutation of the Slater determinant reproduces it.

Definitions and notations

The expectation value of $\hat{H}_0$

$$\int \Phi^* \hat{H}_0 \Phi d\tau = N! \int \Phi_H^* \mathcal{A} \hat{H}_0 \mathcal{A} \Phi_H d\tau$$

is readily reduced to

$$\int \Phi^* \hat{H}_0 \Phi d\tau = N! \int \Phi_H^* \hat{H}_0 \mathcal{A} \Phi_H d\tau,$$

where we have used eqs. (10) and (11). The next step is to replace the antisymmetrization operator by its definition Eq. (8) and to replace $\hat{H}_0$ with the sum of one-body operators

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{i=1}^{N} \sum_{p} (-)^p \int \Phi_H^* \hat{h}_i \hat{P} \Phi_H d\tau.$$

Definitions and notations

The integral vanishes if two or more particles are permuted in only one of the Hartree-functions Φ_H because the individual single-particle wave functions are orthogonal. We obtain then

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{i=1}^{N} \int \Phi_H^* \hat{h}_i \Phi_H d\tau.$$

Orthogonality of the single-particle functions allows us to further simplify the integral, and we arrive at the following expression for the expectation values of the sum of one-body Hamiltonians

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{\mu=1}^{N} \int \psi_\mu^*(\mathbf{r}) \hat{h} \psi_\mu(\mathbf{r}) d\mathbf{r}. \quad (12)$$

Definitions and notations

We introduce the following shorthand for the above integral

$$\langle \mu | h | \mu \rangle = \int \psi_\mu^*(\mathbf{r}) \hat{h} \psi_\mu(\mathbf{r}) d\mathbf{r},$$

and rewrite Eq. (12) as

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{\mu=1}^{N} \langle \mu | h | \mu \rangle. \tag{13}$$

Definitions and notations

The expectation value of the two-body Hamiltonian is obtained in a similar manner. We have

$$\int \Phi^* \hat{H}_1 \Phi d\tau = N! \int \Phi_H^* \mathcal{A} \hat{H}_1 \mathcal{A} \Phi_H d\tau,$$

which reduces to

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{i \leq j=1}^{N} \sum_p (-)^p \int \Phi_H^* V(r_{ij}) \hat{P} \Phi_H d\tau,$$

by following the same arguments as for the one-body Hamiltonian.

Definitions and notations

Because of the dependence on the inter-particle distance r_{ij}, permutations of any two particles no longer vanish, and we get

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{i<j=1}^{N} \int \Phi_H^* V(r_{ij})(1 - P_{ij}) \Phi_H d\tau.$$

where P_{ij} is the permutation operator that interchanges nucleon i and nucleon j. Again we use the assumption that the single-particle wave functions are orthogonal.

Definitions and notations

We obtain

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)V(r_{ij})\psi_\mu(\mathbf{r}_i)\psi_\nu(\mathbf{r}_j)d\mathbf{r}_i d\mathbf{r}_j \right. \\ \left. - \int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)V(r_{ij})\psi_\nu(\mathbf{r}_j)\psi_\mu(\mathbf{r}_i)d\mathbf{r}_i d\mathbf{r}_j\right]. \tag{14}$$

The first term is the so-called direct term. It is frequently also called the Hartree term, while the second is due to the Pauli principle and is called the exchange term or just the Fock term. The factor $1/2$ is introduced because we now run over all pairs twice.

Definitions and notations

The last equation allows us to introduce some further definitions. The single-particle wave functions $\psi_\mu(\mathbf{r})$, defined by the quantum numbers μ and $\mathbf{r}$ (recall that $\mathbf{r}$ also includes spin degree) are defined as the overlap

$$\psi_\alpha(\mathbf{r}) = \langle \mathbf{r} | \alpha \rangle.$$

Definitions and notations

We introduce the following shorthands for the above two integrals

$$\langle \mu\nu|V|\mu\nu\rangle = \int \psi^*_\mu(\mathbf{r}_i)\psi^*_\nu(\mathbf{r}_j)V(r_{ij})\psi_\mu(\mathbf{r}_i)\psi_\nu(\mathbf{r}_j)d\mathbf{r}_i d\mathbf{r}_j,$$

and

$$\langle \mu\nu|V|\nu\mu\rangle = \int \psi^*_\mu(\mathbf{r}_i)\psi^*_\nu(\mathbf{r}_j)V(r_{ij})\psi_\nu(\mathbf{r}_j)\psi_\mu(\mathbf{r}_i)d\mathbf{r}_i d\mathbf{r}_j.$$

Definitions and notations

The direct and exchange matrix elements can be brought together if we define the antisymmetrized matrix element

$$\langle\mu\nu|V|\mu\nu\rangle_{AS} = \langle\mu\nu|V|\mu\nu\rangle - \langle\mu\nu|V|\nu\mu\rangle,$$

or for a general matrix element

$$\langle\mu\nu|V|\sigma\tau\rangle_{AS} = \langle\mu\nu|V|\sigma\tau\rangle - \langle\mu\nu|V|\tau\sigma\rangle.$$

It has the symmetry property

$$\langle\mu\nu|V|\sigma\tau\rangle_{AS} = -\langle\mu\nu|V|\tau\sigma\rangle_{AS} = -\langle\nu\mu|V|\sigma\tau\rangle_{AS}.$$

Definitions and notations

The antisymmetric matrix element is also hermitian, implying

$$\langle \mu\nu|V|\sigma\tau\rangle_{AS} = \langle \sigma\tau|V|\mu\nu\rangle_{AS}.$$

With these notations we rewrite Eq. (14) as

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\langle \mu\nu|V|\mu\nu\rangle_{AS}. \tag{15}$$

Definitions and notations

Combining Eqs. (13) and (96) we obtain the energy functional

$$E[\Phi] = \sum_{\mu=1}^{N} \langle \mu | h | \mu \rangle + \frac{1}{2} \sum_{\mu=1}^{N} \sum_{\nu=1}^{N} \langle \mu\nu | V | \mu\nu \rangle_{AS}. \tag{16}$$

which we will use as our starting point for the Hartree-Fock calculations later in this course.

Quantum Monte Carlo Motivation

Most quantum mechanical problems of interest in e.g., atomic, molecular, nuclear and solid state physics consist of a large number of interacting electrons and ions or nucleons. The total number of particles N is usually sufficiently large that an exact solution cannot be found. Typically, the expectation value for a chosen hamiltonian for a system of N particles is

$$\langle H \rangle = \frac{\int d\mathbf{R}_1 d\mathbf{R}_2 \dots d\mathbf{R}_N \Psi^*(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N) H(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N) \Psi(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N)}{\int d\mathbf{R}_1 d\mathbf{R}_2 \dots d\mathbf{R}_N \Psi^*(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N) \Psi(\mathbf{R_1}, \mathbf{R}_2, \dots, \mathbf{R}_N)},$$

an in general intractable problem. an in general intractable problem.

This integral is actually the starting point in a Variational Monte Carlo calculation.

Gaussian quadrature: Forget it! given 10 particles and 10 mesh points for each degree of freedom and an ideal 1 Tflops machine (all operations take the same time), how long will it ta ke to compute the above integral? Lifetime of the universe $T \approx 4.7 \times 10^{17}$s.

Quantum Monte Carlo

As an example from the nuclear many-body problem, we have Schrödinger's equation as a differential equation

$$\hat{H}\Psi(\mathbf{r}_1,..,\mathbf{r}_A,\alpha_1,..,\alpha_A)=E\Psi(\mathbf{r}_1,..,\mathbf{r}_A,\alpha_1,..,\alpha_A)$$

where

$$\mathbf{r}_1,..,\mathbf{r}_A,$$

are the coordinates and

$$\alpha_1,..,\alpha_A,$$

are sets of relevant quantum numbers such as spin and isospin for a system of A nucleons ($A=N+Z$, N being the number of neutrons and Z the number of protons).

Quantum Monte Carlo

There are

$$2^A \times \begin{pmatrix} A \\ Z \end{pmatrix}$$

coupled second-order differential equations in $3A$ dimensions.
For a nucleus like ^{10}Be this number is **215040**. This is a truely challenging many-body problem.

Methods like partial differential equations can at most be used for 2-3 particles.

Quantum Many-particle(body) Methods

1. Monte-Carlo methods
2. Renormalization group (RG) methods, in particular density matrix RG
3. Large-scale diagonalization (Iterative methods, Lanczo's method, dimensionalities 10^{10} states)
4. Coupled cluster theory, favoured method in quantum chemistry, molecular and atomic physics. Applications to ab initio calculations in nuclear physics as well for large nuclei.
5. Perturbative many-body methods
6. Green's function methods
7. Density functional theory/Mean-field theory and Hartree-Fock theory

The physics of the system hints at which many-body methods to use. For systems with strong correlations among the constituents, item 5 and 7 are ruled out.

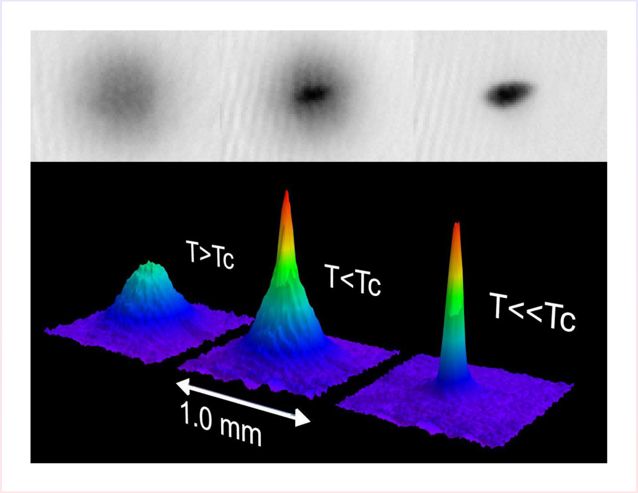
Pros and Cons of Monte Carlo

- Is physically intuitive.
- Allows one to study systems with many degrees of freedom. Diffusion Monte Carlo (DMC) and Green's function Monte Carlo (GFMC) yield in principle the exact solution to Schrödinger's equation.
- Variational Monte Carlo (VMC) is easy to implement but needs a reliable trial wave function, can be difficult to obtain. This is where we will use Hartree-Fock theory to construct an optimal basis.
- DMC/GFMC for fermions (spin with half-integer values, electrons, baryons, neutrinos, quarks) has a sign problem. Nature prefers an anti-symmetric wave function. PDF in this case given distribution of random walkers ($p \geq 0$).
- The solution has a statistical error, which can be large.
- There is a limit for how large systems one can study, DMC needs a huge number of random walkers in order to achieve stable results.
- Obtain only the lowest-lying states with a given symmetry. Can get excited states.

Where and why do we use Monte Carlo Methods in Quantum Physics

- Quantum systems with many particles at finite temperature: Path Integral Monte Carlo with applications to dense matter and quantum liquids (phase transitions from normal fluid to superfluid). Strong correlations.
- Bose-Einstein condensation of dilute gases, method transition from non-linear PDE to Diffusion Monte Carlo as density increases.
- Light atoms, molecules, solids and nuclei.
- Lattice Quantum-Chromo Dynamics. Impossible to solve without MC calculations.
- Simulations of systems in solid state physics, from semiconductors to spin systems. Many electrons active and possibly strong correlations.

Bose-Einstein Condensation of atoms, thousands of Atoms in one State, Project 2 in 2007



Quantum Monte Carlo

Given a hamiltonian H and a trial wave function Ψ_T, the variational principle states that the expectation value of $\langle H \rangle$, defined through

$$E[H] = \langle H \rangle = \frac{\int d\mathbf{R}\Psi_T^*(\mathbf{R})H(\mathbf{R})\Psi_T(\mathbf{R})}{\int d\mathbf{R}\Psi_T^*(\mathbf{R})\Psi_T(\mathbf{R})},$$

is an upper bound to the ground state energy E_0 of the hamiltonian H, that is

$$E_0 \leq \langle H \rangle.$$

In general, the integrals involved in the calculation of various expectation values are multi-dimensional ones. Traditional integration methods such as the Gauss-Legendre will not be adequate for say the computation of the energy of a many-body system.

Quantum Monte Carlo

The trial wave function can be expanded in the eigenstates of the hamiltonian since they form a complete set, viz.,

$$\Psi_T(\mathbf{R}) = \sum_i a_i \Psi_i(\mathbf{R}),$$

and assuming the set of eigenfunctions to be normalized one obtains

$$\frac{\sum_{nm} a_m^* a_n \int d\mathbf{R} \Psi_m^*(\mathbf{R}) H(\mathbf{R}) \Psi_n(\mathbf{R})}{\sum_{nm} a_m^* a_n \int d\mathbf{R} \Psi_m^*(\mathbf{R}) \Psi_n(\mathbf{R})} = \frac{\sum_n a_n^2 E_n}{\sum_n a_n^2} \geq E_0,$$

where we used that $H(\mathbf{R})\Psi_n(\mathbf{R}) = E_n\Psi_n(\mathbf{R})$. In general, the integrals involved in the calculation of various expectation values are multi-dimensional ones. The variational principle yields the lowest state of a given symmetry.

Quantum Monte Carlo

In most cases, a wave function has only small values in large parts of configuration space, and a straightforward procedure which uses homogenously distributed random points in configuration space will most likely lead to poor results. This may suggest that some kind of importance sampling combined with e.g., the Metropolis algorithm may be a more efficient way of obtaining the ground state energy. The hope is then that those regions of configurations space where the wave function assumes appreciable values are sampled more efficiently.

The tedious part in a VMC calculation is the search for the variational minimum. A good knowledge of the system is required in order to carry out reasonable VMC calculations. This is not always the case, and often VMC calculations serve rather as the starting point for so-called diffusion Monte Carlo calculations (DMC). DMC is a way of solving exactly the many-body Schrödinger equation by means of a stochastic procedure. A good guess on the binding energy and its wave function is however necessary. A carefully performed VMC calculation can aid in this context.

Quantum Monte Carlo

- Construct first a trial wave function $\psi_T^{\alpha}(\mathbf{R})$, for a many-body system consisting of N particles located at positions $\mathbf{R} = (\mathbf{R_1}, \dots, \mathbf{R_N})$. The trial wave function depends on α variational parameters $\alpha = (\alpha_1, \dots, \alpha_N)$.
- Then we evaluate the expectation value of the hamiltonian H

$$E[H] = \langle H \rangle = \frac{\int d\mathbf{R}\Psi^{*}_{T_\alpha}(\mathbf{R})H(\mathbf{R})\Psi_{T_\alpha}(\mathbf{R})}{\int d\mathbf{R}\Psi^{*}_{T_\alpha}(\mathbf{R})\Psi_{T_\alpha}(\mathbf{R})}.$$

- Thereafter we vary α according to some minimization algorithm and return to the first step.

Quantum Monte Carlo

Choose a trial wave function $\psi_T(\mathbf{R})$.

$$P(\mathbf{R}) = \frac{\left|\psi_T(\mathbf{R})\right|^2}{\int \left|\psi_T(\mathbf{R})\right|^2 d\mathbf{R}}.$$

This is our new probability distribution function (PDF). The approximation to the expectation value of the Hamiltonian is now

$$E[H] \approx \frac{\int d\mathbf{R}\Psi^*_T(\mathbf{R})H(\mathbf{R})\Psi_T(\mathbf{R})}{\int d\mathbf{R}\Psi^*_T(\mathbf{R})\Psi_T(\mathbf{R})}.$$

Define a new quantity

$$E_L(\mathbf{R}) = \frac{1}{\psi_T(\mathbf{R})}H\psi_T(\mathbf{R}),$$

called the local energy, which, together with our trial PDF yields

$$E[H] = \langle H \rangle \approx \int P(\mathbf{R})E_L(\mathbf{R})d\mathbf{R} \approx \frac{1}{N}\sum_{i=1}^{N} P(\mathbf{R_i})E_L(\mathbf{R_i})$$

with N being the number of Monte Carlo samples.

Quantum Monte Carlo

Algo:

- Initialisation: Fix the number of Monte Carlo steps. Choose an initial **R** and variational parameters α and calculate $\left|\psi_T^{\alpha}(\mathbf{R})\right|^2$.
- Initialise the energy and the variance and start the Monte Carlo calculation (thermalize)
    1. Calculate a trial position $\mathbf{R}_p = \mathbf{R} + r * step$ where r is a random variable $r \in [0, 1]$.
    2. Metropolis algorithm to accept or reject this move
    $$w = P(\mathbf{R}_p)/P(\mathbf{R}).$$
    3. If the step is accepted, then we set $\mathbf{R} = \mathbf{R}_p$. Update averages
- Finish and compute final averages.

Observe that the jumping in space is governed by the variable *step*. Called brute-force sampling. Need importance sampling to get more relevant sampling.

Quantum Monte Carlo

The radial Schrödinger equation for the hydrogen atom can be written as

$$-\frac{\hbar^2}{2m}\frac{\partial^2 u(r)}{\partial r^2}-\left(\frac{ke^2}{r}-\frac{\hbar^2 l(l+1)}{2mr^2}\right)u(r)=Eu(r),$$

or with dimensionless variables

$$-\frac{1}{2}\frac{\partial^2 u(\rho)}{\partial \rho^2}-\frac{u(\rho)}{\rho}+\frac{l(l+1)}{2\rho^2}u(\rho)-\lambda u(\rho)=0,$$

with the hamiltonian

$$H=-\frac{1}{2}\frac{\partial^2}{\partial \rho^2}-\frac{1}{\rho}+\frac{l(l+1)}{2\rho^2}.$$

Use variational parameter α in the trial wave function

$$u_T^{\alpha}(\rho)=\alpha\rho e^{-\alpha\rho}.$$

Quantum Monte Carlo

Inserting this wave function into the expression for the local energy E_L gives

$$E_L(\rho) = -\frac{1}{\rho} - \frac{\alpha}{2}\left(\alpha - \frac{2}{\rho}\right).$$

α	$\langle H \rangle$	σ^2	$\sigma/\sqrt{N}$
7.00000E-01	-4.57759E-01	4.51201E-02	6.71715E-04
8.00000E-01	-4.81461E-01	3.05736E-02	5.52934E-04
9.00000E-01	-4.95899E-01	8.20497E-03	2.86443E-04
1.00000E-00	-5.00000E-01	0.00000E+00	0.00000E+00
1.10000E+00	-4.93738E-01	1.16989E-02	3.42036E-04
1.20000E+00	-4.75563E-01	8.85899E-02	9.41222E-04
1.30000E+00	-4.54341E-01	1.45171E-01	1.20487E-03

Quantum Monte Carlo

We note that at $\alpha = 1$ we obtain the exact result, and the variance is zero, as it should. The reason is that we then have the exact wave function, and the action of the hamiltonan on the wave function

$$H\psi = \mathrm{constant} \times \psi,$$

yields just a constant. The integral which defines various expectation values involving moments of the hamiltonian becomes then

$$\langle H^n \rangle = \frac{\int d\mathbf{R}\Psi_T^*(\mathbf{R})H^n(\mathbf{R})\Psi_T(\mathbf{R})}{\int d\mathbf{R}\Psi_T^*(\mathbf{R})\Psi_T(\mathbf{R})} = \mathrm{constant} \times \frac{\int d\mathbf{R}\Psi_T^*(\mathbf{R})\Psi_T(\mathbf{R})}{\int d\mathbf{R}\Psi_T^*(\mathbf{R})\Psi_T(\mathbf{R})} = \mathrm{constant}.$$

This gives an important information: the exact wave function leads to zero variance! Variation is then performed by minimizing both the energy and the variance.

Quantum Monte Carlo

The helium atom consists of two electrons and a nucleus with charge $Z = 2$. The contribution to the potential energy due to the attraction from the nucleus is

$$-\frac{2ke^2}{r_1} - \frac{2ke^2}{r_2},$$

and if we add the repulsion arising from the two interacting electrons, we obtain the potential energy

$$V(r_1, r_2) = -\frac{2ke^2}{r_1} - \frac{2ke^2}{r_2} + \frac{ke^2}{r_{12}},$$

with the electrons separated at a distance $r_{12} = |\mathbf{r}_1 - \mathbf{r}_2|$.

Quantum Monte Carlo

The hamiltonian becomes then

$$\widehat{\mathbf{H}} = -\frac{\hbar^2\nabla_1^2}{2m} - \frac{\hbar^2\nabla_2^2}{2m} - \frac{2ke^2}{r_1} - \frac{2ke^2}{r_2} + \frac{ke^2}{r_{12}},$$

and Schrödingers equation reads

$$\widehat{\mathbf{H}}\psi = E\psi.$$

All observables are evaluated with respect to the probability distribution

$$P(\mathbf{R}) = \frac{\left|\psi_T(\mathbf{R})\right|^2}{\int \left|\psi_T(\mathbf{R})\right|^2 d\mathbf{R}}.$$

generated by the trial wave function. The trial wave function must approximate an exact eigenstate in order that accurate results are to be obtained. Improved trial wave functions also improve the importance sampling, reducing the cost of obtaining a certain statistical accuracy.

Quantum Monte Carlo

Choice of trial wave function for Helium: Assume $r_1 \to 0$.

$$E_L(\mathbf{R}) = \frac{1}{\psi_T(\mathbf{R})} H\psi_T(\mathbf{R}) = \frac{1}{\psi_T(\mathbf{R})} \left(-\frac{1}{2}\nabla_1^2 - \frac{Z}{r_1} \right) \psi_T(\mathbf{R}) + \text{finite terms.}$$

$$E_L(R) = \frac{1}{\mathcal{R}_T(r_1)} \left(-\frac{1}{2}\frac{d^2}{dr_1^2} - \frac{1}{r_1}\frac{d}{dr_1} - \frac{Z}{r_1} \right) \mathcal{R}_T(r_1) + \text{finite terms}$$

For small values of r_1, the terms which dominate are

$$\lim_{r_1 \to 0} E_L(R) = \frac{1}{\mathcal{R}_T(r_1)} \left(-\frac{1}{r_1}\frac{d}{dr_1} - \frac{Z}{r_1} \right) \mathcal{R}_T(r_1),$$

since the second derivative does not diverge due to the finiteness of Ψ at the origin.

Quantum Monte Carlo

This results in

$$\frac{1}{\mathcal{R}_T(r_1)}\frac{d\mathcal{R}_T(r_1)}{dr_1} = -Z,$$

and

$$\mathcal{R}_T(r_1) \propto e^{-Zr_1}.$$

A similar condition applies to electron 2 as well. For orbital momenta $l > 0$ we have

$$\frac{1}{\mathcal{R}_T(r)}\frac{d\mathcal{R}_T(r)}{dr} = -\frac{Z}{l+1}.$$

Similalry, studying the case $r_{12} \to 0$ we can write a possible trial wave function as

$$\psi_T(\mathbf{R}) = e^{-\alpha(r_1+r_2)}e^{\beta r_{12}}.$$

The last equation can be generalized to

$$\psi_T(\mathbf{R}) = \phi(\mathbf{r}_1)\phi(\mathbf{r}_2)\dots\phi(\mathbf{r}_N)\prod_{i<j} f(r_{ij}),$$

for a system with N electrons or particles.

VMC code for helium, vmc_para.cpp

```
//  Here we define global variables  used in various functions
//  These can be changed by reading from file the different parameters
int dimension = 3; // three-dimensional system
int charge = 2;  //  we fix the charge to be that of the helium atom
int my_rank, numprocs;  //  these are the parameters used by MPI  to
                        //    define which node and how many
double step_length = 1.0;  //  we fix the brute force jump to 1 Bohr r
int number_particles  = 2;  //  we fix also the number of electrons to
```

VMC code for helium, vmc_para.cpp, main part

```
//  MPI initializations
MPI_Init (&argc, &argv);
MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
time_start = MPI_Wtime();

if (my_rank == 0 && argc <= 2) {
  cout << "Bad Usage: " << argv[0] <<
    " read also output file on same line" << endl;
  exit(1);
}
if (my_rank == 0 && argc > 2) {
  outfilename=argv[1];
  ofile.open(outfilename);
}
```

VMC code for helium, vmc_para.cpp, main part

```
// Setting output file name for this rank:
ostringstream ost;
ost << "blocks_rank" << my_rank << ".dat";
// Open file for writing:
blockofile.open(ost.str().c_str(), ios::out | ios::binary);

total_cumulative_e = new double[max_variations+1];
total_cumulative_e2 = new double[max_variations+1];
cumulative_e = new double[max_variations+1];
cumulative_e2 = new double[max_variations+1];

//  initialize the arrays  by zeroing them
for( i=1; i <= max_variations; i++){
   cumulative_e[i] = cumulative_e2[i]  = 0.0;
   total_cumulative_e[i] = total_cumulative_e2[i]  = 0.0;
}
```

VMC code for helium, vmc_para.cpp, main part

```
// broadcast the total number of  variations
MPI_Bcast (&max_variations, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&number_cycles, 1, MPI_INT, 0, MPI_COMM_WORLD);
total_number_cycles = number_cycles*numprocs;
// array to store all energies for last variation of alpha
all_energies = new double[number_cycles+1];
//  Do the mc sampling  and accumulate data with MPI_Reduce
mc_sampling(max_variations, number_cycles, cumulative_e,
              cumulative_e2, all_energies);
//  Collect data in total averages
for( i=1; i <= max_variations; i++){
  MPI_Reduce(&cumulative_e[i], &total_cumulative_e[i], 1, MPI_DOUBLE,
                                             MPI_SUM, 0, MPI_COMM_WORLD);
  MPI_Reduce(&cumulative_e2[i], &total_cumulative_e2[i], 1, MPI_DOUBLE
                                    MPI_SUM, 0, MPI_COMM_WORLD);
}
```

VMC code for helium, vmc_para.cpp, main part

```
blockofile.write((char*)(all_energies+1),
   number_cycles*sizeof(double));
blockofile.close();
delete [] total_cumulative_e; delete [] total_cumulative_e2;
delete [] cumulative_e; delete [] cumulative_e2; delete [] all_energie
// End MPI
MPI_Finalize ();
return 0;
}  //  end of main function
```

VMC code for helium, vmc_para.cpp, sampling

```
alpha = 0.5*charge;
// every node has its own seed for the random numbers
idum = -1-my_rank;
// allocate matrices which contain the position of the particles
r_old =(double **)matrix(number_particles,dimension,sizeof(double));
r_new =(double **)matrix(number_particles,dimension,sizeof(double));
for (i = 0; i < number_particles; i++) {
   for ( j=0; j < dimension; j++) {
     r_old[i][j] = r_new[i][j] = 0;
   }
 }
// loop over variational parameters
```

VMC code for helium, vmc_para.cpp, sampling

```
for (variate=1; variate <= max_variations; variate++){
  // initialisations of variational parameters and energies
  alpha += 0.1;
  energy = energy2 = 0; accept =0; delta_e=0;
  //  initial trial position, note calling with alpha
  for (i = 0; i < number_particles; i++) {
    for ( j=0; j < dimension; j++) {
r_old[i][j] = step_length*(ran2(&idum)-0.5);
    }
  }
  wfold = wave_function(r_old, alpha);
```

VMC code for helium, vmc_para.cpp, sampling

```
// loop over monte carlo cycles
for (cycles = 1; cycles <= number_cycles; cycles++){
   // new position
   for (i = 0; i < number_particles; i++) {
       for ( j=0; j < dimension; j++) {
      r_new[i][j] = r_old[i][j]+step_length*(ran2(&idum)-0.5);
     }
//  for the other particles we need to set the position to the old pos
//  we move only one particle at the time
     for (k = 0; k < number_particles; k++) {
 if ( k != i) {
    for ( j=0; j < dimension; j++) {
      r_new[k][j] = r_old[k][j];
    }
  }
        }
      wfnew = wave_function(r_new, alpha);
// The Metropolis test is performed by moving one particle at the time
      if(ran2(&idum) <= wfnew*wfnew/wfold/wfold ) {
  for ( j=0; j < dimension; j++) {
    r_old[i][j]=r_new[i][j];
  }
  wfold = wfnew;
}
      }  //  end of loop over particles
```

VMC code for helium, vmc_para.cpp, sampling

```
      // compute local energy
      delta_e = local_energy(r_old, alpha, wfold);
      // save all energies on last variate
      if(variate==max_variations){
   all_energies[cycles] = delta_e;
      }
      // update energies
      energy += delta_e;
      energy2 += delta_e*delta_e;
    }   // end of loop over MC trials
    // update the energy average and its squared
    cumulative_e[variate] = energy;
    cumulative_e2[variate] = energy2;
  }    // end of loop over variational  steps
```

VMC code for helium, vmc_para.cpp, wave function

```
// Function to compute the squared wave function, simplest form

double  wave_function(double **r, double alpha)
{
  int i, j, k;
  double wf, argument, r_single_particle, r_12;

  argument = wf = 0;
  for (i = 0; i < number_particles; i++) {
    r_single_particle = 0;
    for (j = 0; j < dimension; j++) {
      r_single_particle  += r[i][j]*r[i][j];
    }
    argument += sqrt(r_single_particle);
  }
  wf = exp(-argument*alpha) ;
  return wf;
}
```

VMC code for helium, vmc_para.cpp, local energy

```
// Function to calculate the local energy with num derivative

double  local_energy(double **r, double alpha, double wfold)
{
  int i, j , k;
  double e_local, wfminus, wfplus, e_kinetic, e_potential, r_12,
    r_single_particle;
  double **r_plus, **r_minus;

  // allocate matrices which contain the position of the particles
  // the function matrix is defined in the progam library
  r_plus =(double **)matrix(number_particles,dimension,sizeof(double))
  r_minus =(double **)matrix(number_particles,dimension,sizeof(double)
  for (i = 0; i < number_particles; i++) {
    for ( j=0; j < dimension; j++) {
      r_plus[i][j] = r_minus[i][j] = r[i][j];
    }
  }
```

VMC code for helium, vmc_para.cpp, local energy

```
    // compute the kinetic energy
    e_kinetic = 0;
    for (i = 0; i < number_particles; i++) {
      for (j = 0; j < dimension; j++) {
        r_plus[i][j] = r[i][j]+h;
        r_minus[i][j] = r[i][j]-h;
        wfminus = wave_function(r_minus, alpha);
        wfplus  = wave_function(r_plus, alpha);
        e_kinetic -= (wfminus+wfplus-2*wfold);
        r_plus[i][j] = r[i][j];
        r_minus[i][j] = r[i][j];
      }
    }
  // include electron mass and hbar squared and divide by wave function
    e_kinetic = 0.5*h2*e_kinetic/wfold;
```

VMC code for helium, vmc_para.cpp, local energy

```
    // compute the potential energy
    e_potential = 0;
    // contribution from electron-proton potential
    for (i = 0; i < number_particles; i++) {
      r_single_particle = 0;
      for (j = 0; j < dimension; j++) {
        r_single_particle += r[i][j]*r[i][j];
      }
      e_potential -= charge/sqrt(r_single_particle);
    }
    // contribution from electron-electron potential
    for (i = 0; i < number_particles-1; i++) {
      for (j = i+1; j < number_particles; j++) {
        r_12 = 0;
        for (k = 0; k < dimension; k++) {
  r_12 += (r[i][k]-r[j][k])*(r[i][k]-r[j][k]);
        }
        e_potential += 1/sqrt(r_12);
      }
    }
```

Going Parallel with MPI

In all projects it will be useful to parallelize the code. **Task parallelism**: the work of a global problem can be divided into a number of independent tasks, which rarely need to synchronize. Monte Carlo simulation or integrations are examples of this. It is almost embarrassingly trivial to parallelize Monte Carlo codes. MPI is a message-passing library where all the routines have corresponding C/C++-binding

```
MPI_Command_name
```

and Fortran-binding (routine names are in uppercase, but can also be in lower case)

```
MPI_COMMAND_NAME
```

What is Message Passing Interface (MPI)? Yet another library!

MPI is a library, not a language. It specifies the names, calling sequences and results of functions or subroutines to be called from C or Fortran programs, and the classes and methods that make up the MPI C++ library. The programs that users write in Fortran, C or C++ are compiled with ordinary compilers and linked with the MPI library.

MPI is a specification, not a particular implementation. MPI programs should be able to run on all possible machines and run all MPI implementetations without change.

An MPI computation is a collection of processes communicating with messages.

See chapter 7.7 of lecture notes for more details.

MPI

MPI is a library specification for the message passing interface, proposed as a standard.

- independent of hardware;
- not a language or compiler specification;
- not a specific implementation or product.

A message passing standard for portability and ease-of-use.
Designed for high performance.
Insert communication and synchronization functions where necessary.

Demands from the HPC community

In the field of scientific computing, there is an ever-lasting wish to do larger simulations using shorter computer time.
Development of the capacity for single-processor computers can hardly keep up with the pace of scientific computing:

- processor speed
- memory size/speed

Solution: parallel computing!

The basic ideas of parallel computing

- Pursuit of shorter computation time and larger simulation size gives rise to parallel computing.
- Multiple processors are involved to solve a global problem.
- The essence is to divide the entire computation evenly among collaborative processors. Divide and conquer.

A rough classification of hardware models

- Conventional single-processor computers can be called SISD (single-instruction-single-data) machines.
- SIMD (single-instruction-multiple-data) machines incorporate the idea of parallel processing, which use a large number of process- ing units to execute the same instruction on different data.
- Modern parallel computers are so-called MIMD (multiple-instruction- multiple-data) machines and can execute different instruction streams in parallel on different data.

Shared memory and distributed memory

- One way of categorizing modern parallel computers is to look at the memory configuration.
- In shared memory systems the CPUs share the same address space. Any CPU can access any data in the global memory.
- In distributed memory systems each CPU has its own memory. The CPUs are connected by some network and may exchange messages.
- A recent trend is ccNUMA (cache-coherent-non-uniform-memory- access) systems which are clusters of SMP (symmetric multi-processing) machines and have a virtual shared memory.

Different parallel programming paradigms

- **Task parallelism**: the work of a global problem can be divided into a number of independent tasks, which rarely need to synchronize. Monte Carlo simulation is one example. Integration is another. However this paradigm is of limited use.
- **Data parallelism**: use of multiple threads (e.g. one thread per processor) to dissect loops over arrays etc. This paradigm requires a single memory address space. Communication and synchronization between processors are often hidden, thus easy to program. However, the user surrenders much control to a specialized compiler. Examples of data parallelism are compiler-based parallelization and OpenMP directives.

Different parallel programming paradigms

- **Message-passing**: all involved processors have an independent memory address space. The user is responsible for partition- ing the data/work of a global problem and distributing the subproblems to the processors. Collaboration between processors is achieved by explicit message passing, which is used for data transfer plus synchronization.
- This paradigm is the most general one where the user has full control. Better parallel efficiency is usually achieved by explicit message passing. However, message-passing programming is more difficult.

SPMD

Although message-passing programming supports MIMD, it suffices with an SPMD (single-program-multiple-data) model, which is flexible enough for practical cases:

- Same executable for all the processors.
- Each processor works primarily with its assigned local data.
- Progression of code is allowed to differ between synchronization points.
- Possible to have a master/slave model. The standard option in Monte Carlo calculations and numerical integration.

Today’s situation of parallel computing

- Distributed memory is the dominant hardware configuration. There is a large diversity in these machines, from MPP (massively parallel pro cessing) systems to clusters of off-the-shelf PCs, which are very cost-effective.
- Message-passing is a mature programming paradigm and widely accepted. It often provides an efficient match to the hardware. It is primarily used for the distributed memory systems, but can also be used on shared memory systems.

In these lectures we consider only message-passing for writing parallel programs.

Overhead present in parallel computing

- **Uneven load balance**: not all the processors can perform useful work at all time.
- **Overhead of synchronization.**
- **Overhead of communication.**
- Extra computation due to parallelization.

Due to the above overhead and that certain part of a sequential algorithm cannot be parallelized we may not achieve an optimal parallelization.

Parallelizing a sequential algorithm

- Identify the part(s) of a sequential algorithm that can be executed in parallel. This is the difficult part,
- Distribute the global work and data among P processors.

Process and processor

- We refer to process as a logical unit which executes its own code, in an MIMD style.
- The processor is a physical device on which one or several processes are executed.
- The MPI standard uses the concept process consistently throughout its documentation.
- However, we only consider situations where one processor is responsible for one process and therefore use the two terms interchangeably.

Bindings to MPI routines

MPI is a message-passing library where all the routines have corresponding C/C++-binding

```
MPI_Command_name
```

and Fortran-binding (routine names are in uppercase, but can also be in lower case)

```
MPI_COMMAND_NAME
```

The discussion in these slides focuses on the C++ binding.

Communicator

- A group of MPI processes with a name (context).
- Any process is identified by its rank. The rank is only meaningful within a particular communicator.
- By default communicator MPI_COMM_WORLD contains all the MPI processes.
- Mechanism to identify subset of processes.
- Promotes modular design of parallel libraries.

The 6 most important MPI routines

- MPI_ Init - initiate an MPI computation
- MPI_Finalize - terminate the MPI computation and clean up
- MPI_Comm_size - how many processes participate in a given MPI communicator?
- MPI_Comm_rank - which one am I? (A number between 0 and size-1.)
- MPI_Send - send a message to a particular pro cess within an MPI communicator
- MPI_Recv - receive a message from a particular pro cess within an MPI communicator

The first MPI C/C++ program

Let every process write ”Hello world” on the standard output.
This is program2.cpp of chapter 7.

```
using namespace std;
#include <mpi.h>
#include <iostream>
int main (int nargs, char* args[])
{
int numprocs, my_rank;
//   MPI initializations
MPI_Init (&nargs, &args);
MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
cout << "Hello world, I have  rank " << my_rank <<
     << numprocs << endl;
//  End MPI
MPI_Finalize ();
```

The Fortran program

```
PROGRAM hello
INCLUDE "mpif.h"
INTEGER:: size, my_rank, ierr

CALL  MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierr)
WRITE(*,*)"Hello world, I've rank ",my_rank," out o
CALL MPI_FINALIZE(ierr)

END PROGRAM hello
```

Note 1

The output to screen is not ordered since all processes are trying to write to screen simultaneously. It is then the operating system which opts for an ordering. If we wish to have an organized output, starting from the first process, we may rewrite our program as in the next example (program3.cpp), see again chapter 7.7 of lecture notes.

Ordered output with MPI_Barrier

```
int main (int nargs, char* args[])
{
 int numprocs, my_rank, i;
 MPI_Init (&nargs, &args);
 MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
 MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
 for (i = 0; i < numprocs; i++) {}
 MPI_Barrier (MPI_COMM_WORLD);
 if (i == my_rank) {
 cout << "Hello world, I have  rank " << my_rank <<
        " out of " << numprocs << endl;}
      MPI_Finalize ();
```

Note 2

Here we have used the *MPI_Barrier* function to ensure that that every process has completed its set of instructions in a particular order. A barrier is a special collective operation that does not allow the processes to continue until all processes in the communicator (here *MPI_COMM_WORLD* have called *MPI_Barrier*. The barriers make sure that all processes have reached the same point in the code. Many of the collective operations like *MPI_ALLREDUCE* to be discussed later, have the same property; viz. no process can exit the operation until all processes have started. However, this is slightly more time-consuming since the processes synchronize between themselves as many times as there are processes. In the next Hello world example we use the send and receive functions in order to a have a synchronized action.

Ordered output with MPI_Recv and MPI_Send

```
.....
int numprocs, my_rank, flag;
MPI_Status status;
MPI_Init (&nargs, &args);
MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
if (my_rank > 0)
MPI_Recv (&flag, 1, MPI_INT, my_rank-1, 100,
           MPI_COMM_WORLD, &status);
cout << "Hello world, I have  rank " << my_rank <<
<< numprocs << endl;
if (my_rank < numprocs-1)
MPI_Send (&my_rank, 1, MPI_INT, my_rank+1,
          100, MPI_COMM_WORLD);
MPI_Finalize ();
```

Note 3

The basic sending of messages is given by the function *MPI_SEND*, which in C/C++ is defined as

```
int MPI_Send(void *buf, int count,
             MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)}
```

This single command allows the passing of any kind of variable, even a large array, to any group of tasks. The variable **buf** is the variable we wish to send while **count** is the number of variables we are passing. If we are passing only a single value, this should be 1. If we transfer an array, it is the overall size of the array. For example, if we want to send a 10 by 10 array, count would be $10 \times 10 = 100$ since we are actually passing 100 values.

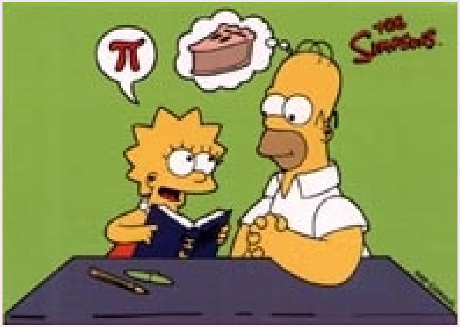
Note 4

Once you have sent a message, you must receive it on another task. The function **MPI_RECV** is similar to the send call.

```
int MPI_Recv( void *buf, int count, MPI_Datatype da
              int source,
              int tag, MPI_Comm comm, MPI_Status *sta
```

The arguments that are different from those in *MPI_SEND* are **buf** which is the name of the variable where you will be storing the received data, **source** which replaces the destination in the send command. This is the return ID of the sender.
Finally, we have used **MPI_Status status;** where one can check if the receive was completed.
The output of this code is the same as the previous example, but now process 0 sends a message to process 1, which forwards it further to process 2, and so forth.
Armed with this wisdom, performed all hello world greetings, we are now ready for serious work.

Integrating π



Examples

- Go to the webpage and click on the programs link
- Go to MPI and then chapter 7
- Look at program5.ccp and program6.cpp. (Fortran version also available).
- These codes compute π using the rectangular and trapezoidal rules.

Integration algos

The trapezoidal rule (example6.cpp)

$$I = \int_a^b f(x)dx = h\left(f(a)/2 + f(a+h) + f(a+2h) + \cdots + f(b-h) + f_b/2\right).$$

Another very simple approach is the so-called midpoint or rectangle method. In this case the integration area is split in a given number of rectangles with length *h* and heigth given by the mid-point value of the function. This gives the following simple rule for approximating an integral

$$I = \int_a^b f(x)dx \approx h\sum_{i=1}^{N} f(x_{i-1/2}),$$

where $f(x_{i-1/2})$ is the midpoint value of f for a given rectangle. This is used in example5.cpp.

Dissection of example 5

```
//    Reactangle rule and numerical integration
using namespace std;
#include <mpi.h>
#include <iostream>

int main (int nargs, char* args[])
{
   int numprocs, my_rank, i, n = 1000;
   double local_sum, rectangle_sum, x, h;
   //   MPI initializations
   MPI_Init (&nargs, &args);
   MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
   MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
```

Dissection of example 5

After the standard initializations with MPI such as MPI_Init, MPI_Comm_size and MPI_Comm_rank, MPI_COMM_WORLD contains now the number of processes defined by using for example

```
mpiexec -np 10 ./prog.x
```

In line 4 we check if we have read in from screen the number of mesh points n. Note that in line 7 we fix $n = 1000$, however we have the possibility to run the code with a different number of mesh points as well. If my_rank equals zero, which correponds to the master node, then we read a new value of n if the number of arguments is larger than two. This can be done as follows when we run the code

```
mpiexec -np 10 ./prog.x  10000
```

Dissection of example 5

```
13     //   Read from screen a possible new vaue of n
14     if (my_rank == 0 && nargs > 1) {
15        n = atoi(args[1]);
16     }
17     h = 1.0/n;
18     //  Broadcast n and h to all processes
19     MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
20     MPI_Bcast (&h, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
21     //  Every process sets up its contribution to the integral
22     local_sum = 0.;
23     for (i = my_rank; i < n; i += numprocs) {
24       x = (i+0.5)*h;
25       local_sum += 4.0/(1.0+x*x);
26    }
27     local_sum *= h;
```

In line 17 we define also the step length *h*. In lines 19 and 20 we use the broadcast function MPI_Bcast. We use this particular function because we want data on one processor (our master node) to be shared with all other processors. The broadcast function sends data to a group of processes.

Dissection of example 5

The MPI routine MPI_Bcast transfers data from one task to a group of others. The format for the call is in C++ given by the parameters of

```
MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);.
MPI_Bcast (&h, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

in a case of a double. The general structure of this function is

```
MPI_Bcast( void *buf, int count, MPI_Datatype datatype, int root, MPI_
```

All processes call this function, both the process sending the data (with rank zero) and all the other processes in MPI_COMM_WORLD. Every process has now copies of *n* and *h*, the number of mesh points and the step length, respectively.

We transfer the addresses of *n* and *h*. The second argument represents the number of data sent. In case of a one-dimensional array, one needs to transfer the number of array elements. If you have an $n \times m$ matrix, you must transfer $n \times m$. We need also to specify whether the variable type we transfer is a non-numerical such as a logical or character variable or numerical of the integer, real or complex type.

Dissection of example 5

```
  if (my_rank == 0) {
    MPI_Status status;
    rectangle_sum = local_sum;
    for (i=1; i < numprocs; i++) {
      MPI_Recv(&local_sum,1,MPI_DOUBLE,MPI_ANY_SOURCE,500,
               MPI_COMM_WORLD,&status);
      rectangle_sum += local_sum;
    }
    cout << "Result: " << rectangle_sum  << endl;
  }  else
    MPI_Send(&local_sum,1,MPI_DOUBLE,0,500,MPI_COMM_WORLD);
  // End MPI
  MPI_Finalize ();
  return 0;
}
```

Dissection of example 5

In lines 23-27, every process sums its own part of the final sum used by the rectangle rule. The receive statement collects the sums from all other processes in case my_rank == 0, else an MPI send is performed. If we are not the master node, we send the results, else they are received and the local results are added to final sum. The above can be rewritten using the MPI_allreduce, as discussed in the next example.

The above function is not very elegant. Furthermore, the MPI instructions can be simplified by using the functions MPI_Reduce or MPI_Allreduce. The first function takes information from all processes and sends the result of the MPI operation to one process only, typically the master node. If we use MPI_Allreduce, the result is sent back to all processes, a feature which is useful when all nodes need the value of a joint operation. We limit ourselves to MPI_Reduce since it is only one process which will print out the final number of our calculation, The arguments to MPI_Allreduce are the same.

MPI_reduce

Call as

```
MPI_reduce( void *senddata, void* resultdata, int count,
     MPI_Datatype datatype, MPI_Op, int root, MPI_Comm comm)
```

The two variables *senddata* and *resultdata* are obvious, besides the fact that one sends the address of the variable or the first element of an array. If they are arrays they need to have the same size. The variable *count* represents the total dimensionality, 1 in case of just one variable, while MPI_Datatype defines the type of variable which is sent and received.
The new feature is MPI_Op. It defines the type of operation we want to do. In our case, since we are summing the rectangle contributions from every process we define MPI_Op = MPI_SUM. If we have an array or matrix we can search for the largest og smallest element by sending either MPI_MAX or MPI_MIN. If we want the location as well (which array element) we simply transfer MPI_MAXLOC or MPI_MINOC. If we want the product we write MPI_PROD.
MPI_Allreduce is defined as

```
MPI_Alreduce( void *senddata, void* resultdata, int count,
          MPI_Datatype datatype, MPI_Op, MPI_Comm comm)}.
```

Dissection of example 6

```
//    Trapezoidal rule and numerical integration usign MPI, example 6
using namespace std;
#include <mpi.h>
#include <iostream>

//     Here we define various functions called by the main program

double int_function(double );
double trapezoidal_rule(double , double , int , double (*)(double));

//   Main function begins here
int main (int nargs, char* args[])
{
  int n, local_n, numprocs, my_rank;
  double a, b, h, local_a, local_b, total_sum, local_sum;
  double  time_start, time_end, total_time;
```

Dissection of example 6

```
//  MPI initializations
MPI_Init (&nargs, &args);
MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
time_start = MPI_Wtime();
//  Fixed values for a, b and n
a = 0.0 ; b = 1.0;  n = 1000;
h = (b-a)/n;    // h is the same for all processes
local_n = n/numprocs;
// make sure n > numprocs, else integer division gives zero
// Length of each process' interval of
// integration = local_n*h.
local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
```

Dissection of example 6

```
  total_sum = 0.0;
  local_sum = trapezoidal_rule(local_a, local_b, local_n,
                               &int_function);
  MPI_Reduce(&local_sum, &total_sum, 1, MPI_DOUBLE,
             MPI_SUM, 0, MPI_COMM_WORLD);
  time_end = MPI_Wtime();
  total_time = time_end-time_start;
  if ( my_rank == 0) {
    cout << "Trapezoidal rule = " <<  total_sum << endl;
    cout << "Time = " <<  total_time
         << " on number of processors: "  << numprocs  << endl;
  }
  // End MPI
  MPI_Finalize ();
  return 0;
}  // end of main program
```

We use MPI_reduce to collect data from each process. Note also the use of the function MPI_Wtime.

Dissection of example 6

```
//  this function defines the function to integrate
double int_function(double x)
{
  double value = 4./(1.+x*x);
  return value;
} // end of function to evaluate
```

Dissection of example 6

Implementation of the trapezoidal rule.

```
//  this function defines the trapezoidal rule
double trapezoidal_rule(double a, double b, int n,
                        double (*func)(double))
{
  double trapez_sum;
  double fa, fb, x, step;
  int    j;
  step=(b-a)/((double) n);
  fa=(*func)(a)/2. ;
  fb=(*func)(b)/2. ;
  trapez_sum=0.;
  for (j=1; j <= n-1; j++){
    x=j*step+a;
    trapez_sum+=(*func)(x);
  }
  trapez_sum=(trapez_sum+fb+fa)*step;
  return trapez_sum;
}  // end trapezoidal_rule
```

Strategies

- Develop codes locally, run with some few processes and test your codes. Do benchmarking, timing and so forth on local nodes, for example your laptop. You can install MPICH2 on your laptop (most new laptos come with dual cores). You can test with one node at the lab.
- When you are convinced that your codes run correctly, you start your production runs on available supercomputers, in our case titan.uio.no.

How do I run MPI on the machines at the lab (MPICH2)

The machines at the lab are all quad-cores

- Compile with mpicxx, mpic++, icc for C++ user and mpif90 or ifort for fortran users.
- Set up collaboration between processes and run

  ```
  mpd --ncpus=4 &
  #  run code with
  mpiexec -n 4 ./nameofprog
  ```

  Here we declare that we will use 4 processes via the $-ncpus$ option and via $-n4$ when running.
- End with

  ```
  mpdallexit
  ```

How do I use the titan.uio.no cluster?

hpc@usit.uio.no

- Computational Physics requires High Performance Computing (HPC) resources
- USIT and the Research Computing Services (RCS) provides HPC resources and HPC support
- Resources: `titan.uio.no`
- Support: 14 people
- Contact: `hpc@usit.uio.no`

Titan

Hardware

- 546 X2200m2, 7 X4200, Magnum 3456 IB switch
- Hugemem nodes of 128 - 256GB RAM
- EVA8K 120 TB storage
- Quad core Intel and AMD ($\sim$ 4000 cores in total)
- Infiniband and ethernet
- Heterogenous cluster!

Titan

Software

- Batch system: SLURM and MAUI
- Message Passing Interface (MPI):
  - OpenMPI
  - Scampi
  - MPICH2
- Compilers: GCC, Intel, Portland and Pathscale
- Optimized math libraries and scientific applications
- All you need may be found under `/site`
- Available software: `http://www.hpc.uio.no/index.php/Titan_software`

Getting started

Batch systems

- A batch system controls the use of the cluster resources
- Submits the job to the right resource
- Monitors the job while executing
- Restarts the job in case of failure
- Takes care of priorities and queues to control execution order of unrelated jobs

Sun Grid Engine

- SGE is the batch system used on Titan
- Jobs are executed either interactively or through job scripts
- Useful commands: `showq`, `qlogin`, `sbatch`
- `http://hpc.uio.no/index.php/Titan_User_Guide`

Getting started

Modules

- Different compilers, MPI-versions and applications need different sets of user environment variables
- The `modules` package lets you load and remove the different variable sets
- Useful commands:
  - List available modules: `module avail`
  - Load module: `module load <environment>`
  - Unload module: `module unload <environment>`
  - Currently loaded: `module list`
- `http://hpc.uio.no/index.php/Titan_User_Guide`

Example

Interactively

```
# login to titan
$ ssh titan.uio.no
# ask for 4 cpus
$ qlogin --account=fys3150 --ntasks=4
# start a job setup, note the punktum!
$ source  /site/bin/jobsetup
# we want to use the scampi module
$ module load scampi
#  or replace scampi with openmpi
$ mkdir -p fys3150/mpiexample/
$ cd fys3150/mpiexample/
# Use program5.cpp from the course pages, see chapter 7
# compile the program
$ mpic++ -O3 -o program5.x program5.cpp
# and execute it
$ mpirun ./program5.x
 4 /opt/scali/bin/mpimon -stdin all  ./program5.x --  compute-9-21 1 compute-9-26 1
       compute-9-26 1 compute-9-31 1
$ Result: 3.14159
```

The job script is called file.slurm

job.slurm

```
#!/bin/sh
# Call this file job.slurm
# 4 cpus with mpi (or other communication)
#SBATCH --ntasks=4
# 10 mins of walltime
#SBATCH --time=0:10:00
# project fys3150
#SBATCH --account=fys3150
# we need 2000 MB of memory per process
#SBATCH --mem-per-cpu=2000M
# name of job
#SBATCH --job-name=program5

source /site/bin/jobsetup

# load the module used when we compiled the program
module load scampi

# start program
mpirun ./program5.x

#END OF SCRIPT
```

Example

Submitting

```
# login to titan
$ ssh titan.uio.no
# we want to use the module scampi
$ module load scampi
$ cd fys3150/mpiexample/
# compile the program
$ mpic++ -O3 -o program5.x program5.cpp
# and submit it
$ sbatch job.slurm
$ exit
```

Example

Checking execution

```
# check if job is running:
$ showq -u mhjensen
ACTIVE JOBS--------------------
JOBNAME           USERNAME      STATE   PROC    REMAINING             STARTTIME

883129            mhjensen    Running      4     10:31:17   Fri Oct  2 13:59:25

     1 Active Job      2692 of 4252 Processors Active (63.31%)
                        482 of  602 Nodes Active      (80.07%)

IDLE JOBS----------------------
JOBNAME           USERNAME      STATE   PROC      WCLIMIT             QUEUETIME


0 Idle Jobs

BLOCKED JOBS----------------
JOBNAME           USERNAME      STATE   PROC      WCLIMIT             QUEUETIME


Total Jobs: 1   Active Jobs: 1   Idle Jobs: 0   Blocked Jobs: 0
```

Tips and admonitions

Tips

- Titan FAQ: `http://www.hpc.uio.no/index.php/FAQ`
- man-pages, e.g. `man sbatch`
- Ask us

Admonitions

- Remember to exit from qlogin-sessions; the resource is reserved for you untill you exit
- Don't run jobs on login-nodes; these are only for compiling and editing files

Topics for Week 4, 25-29 January

Work on project 1

- Repetion from previous weeks
- Only work on project 1

Topics for Week 5, February 1-5

Importance sampling and closed form expressions for the local energy

- Repetition from the last two weeks
- How to compute the local energy, numerically versus closed form expressions
- Importance sampling, the basic philosophy.

Project work this week: finalize 1a and start of 1b.

Structuring the code

During the development of our code we need to make several checks. It is also very instructive to compute a closed form expression for the local energy. Since our wave function is rather simple it is straightforward to find an analytic expressions. Consider first the case of the simple helium function

$$\Psi_T(\mathbf{r}_1, \mathbf{r}_2) = e^{-\alpha(r_1+r_2)}$$

The local energy is for this case

$$E_{L1} = (\alpha - Z)\left(\frac{1}{r_1} + \frac{1}{r_2}\right) + \frac{1}{r_{12}} - \alpha^2$$

which gives an expectation value for the local energy given by

$$\langle E_{L1} \rangle = \alpha^2 - 2\alpha\left(Z - \frac{5}{16}\right)$$

Structuring the code

With analytic formulae we can speed up the computation of the correlation. In our case we write it as

$$\Psi_C = \exp\left\{\sum_{i<j}\frac{ar_{ij}}{1+\beta r_{ij}}\right\},$$

which means that the gradient needed for the quantum force and local energy can be calculated analytically. This will speed up your code since the computation of the correlation part and the Slater determinant are the most time consuming parts in your code. In part 1a of project this needs to be included. It should also be included for part 2a and part 2b.
We will refer to this correlation function as Ψ_C or the *linear Padé-Jastrow*.

Structuring the code

We can test this by computing the local energy for our helium wave function

$$\psi_T(\mathbf{r}_1, \mathbf{r}_2) = \exp\left(-\alpha(r_1 + r_2)\right) \exp\left(\frac{r_{12}}{2(1+\beta r_{12})}\right),$$

with α and β as variational parameters.
The local energy is for this case

$$E_{L2} = E_{L1} + \frac{1}{2(1+\beta r_{12})^2}\left\{\frac{\alpha(r_1+r_2)}{r_{12}}(1 - \frac{\mathbf{r}_1\mathbf{r}_2}{r_1 r_2}) - \frac{1}{2(1+\beta r_{12})^2} - \frac{2}{r_{12}} + \frac{2\beta}{1+\beta r_{12}}\right\}$$

It is very useful to test your code against these expressions. It means also that you don't need to compute derivative numerically as discussed last week. This week you should implement this expression and test the time usage against the code with numerical derivation.

Structuring the code, simple task

- Make another copy of your code.
- Implement the closed form expression for the local energy
- Compile the new and old codes with the -pg option for profiling.
- Run both codes and profile them afterwards using *gprof* $<$ *executable* $>>$ *outprofile*
- Study the time usage in the file *outprofile*

Your tasks till next week

- Implement the closed form expression for the local energy
- Convince yourself that the closed form expressions are correct, see slides from this week.
- Implement the above expressions for systems with more than two electrons.
- Implement importance sampling, see code vmc_be.cpp.
- Finish part 1a and begin part 1b. Blocking will be discussed next week.
- You need to produce random numbers with a Gaussian distribution.
- Reading task: Thijssen's text chapters 8.8 and 12.2.

Efficient calculations of wave function ratios

In the Metropolis/Hasting algorithm, the *acceptance ratio* determines the probability for a particle to be accepted at a new position. The ratio of the trial wave functions evaluated at the new and current positions is given by

$$\boxed{R \equiv \frac{\Psi_T^{new}}{\Psi_T^{cur}} = \underbrace{\frac{\Psi_D^{new}}{\Psi_D^{cur}}}_{R_{SD}} \underbrace{\frac{\Psi_C^{new}}{\Psi_C^{cur}}}_{R_C}.} \tag{17}$$

Here Ψ_D is our Slater determinant while Ψ_C is our correlation function. We need to optimize $\nabla\Psi_T/\Psi_T$ ratio and the second derivative as well, that is the $\nabla^2\Psi_T/\Psi_T$ ratio. The first is needed when we compute the so-called quantum force in importance sampling. The second is needed when we compute the kinetic energy term of the local energy.

$$\frac{\mathbf{\nabla}\Psi}{\Psi} = \frac{\mathbf{\nabla}(\Psi_D\,\Psi_C)}{\Psi_D\,\Psi_C} = \frac{\Psi_C\mathbf{\nabla}\Psi_D + \Psi_D\mathbf{\nabla}\Psi_C}{\Psi_D\Psi_C} = \frac{\mathbf{\nabla}\Psi_D}{\Psi_D} + \frac{\mathbf{\nabla}\Psi_C}{\Psi_C}$$

Efficient calculations of wave function ratios

The expectation value of the kinetic energy expressed in atomic units for electron *i* is

$$\langle \widehat{\mathbf{K}}_i \rangle = -\frac{1}{2}\frac{\langle\Psi|\nabla_i^2|\Psi\rangle}{\langle\Psi|\Psi\rangle}, \tag{18}$$

$$K_i = -\frac{1}{2}\frac{\nabla_i^2\Psi}{\Psi}. \tag{19}$$

$$\begin{aligned}\frac{\nabla^2\Psi}{\Psi} &= \frac{\nabla^2(\Psi_D\,\Psi_C)}{\Psi_D\,\Psi_C} = \frac{\boldsymbol{\nabla}\cdot[\boldsymbol{\nabla}(\Psi_D\,\Psi_C)]}{\Psi_D\,\Psi_C} = \frac{\boldsymbol{\nabla}\cdot[\Psi_C\boldsymbol{\nabla}\Psi_D + \Psi_D\boldsymbol{\nabla}\Psi_C]}{\Psi_D\,\Psi_C}\\ &= \frac{\boldsymbol{\nabla}\Psi_C\cdot\boldsymbol{\nabla}\Psi_D + \Psi_C\nabla^2\Psi_D + \boldsymbol{\nabla}\Psi_D\cdot\boldsymbol{\nabla}\Psi_C + \Psi_D\nabla^2\Psi_C}{\Psi_D\,\Psi_C}\end{aligned} \tag{20}$$

$$\frac{\nabla^2\Psi}{\Psi} = \frac{\nabla^2\Psi_D}{\Psi_D} + \frac{\nabla^2\Psi_C}{\Psi_C} + 2\frac{\boldsymbol{\nabla}\Psi_D}{\Psi_D}\cdot\frac{\boldsymbol{\nabla}\Psi_C}{\Psi_C} \tag{21}$$

Definitions

We define the correlated function as

$$\Psi_C = \prod_{i<j} g(r_{ij}) = \prod_{i<j}^{N} g(r_{ij}) = \prod_{i=1}^{N} \prod_{j=i+1}^{N} g(r_{ij}),$$

with $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$. In our particular case we have

$$\Psi_C = \prod_{i<j} g(r_{ij}) = \exp\left\{\sum_{i<j} f(r_{ij})\right\} = \exp\left\{\sum_{i<j} \frac{ar_{ij}}{1+\beta r_{ij}}\right\},$$

Efficient calculations of wave function ratios

The total number of different relative distances r_{ij} is $N(N-1)/2$. In a matrix storage format, the set forms a strictly upper triangular matrix

$$\mathbf{r} \equiv \begin{pmatrix} 0 & r_{1,2} & r_{1,3} & \cdots & r_{1,N} \\ \vdots & 0 & r_{2,3} & \cdots & r_{2,N} \\ \vdots & \vdots & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & r_{N-1,N} \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}. \tag{22}$$

This applies to $\mathbf{g} = \mathbf{g}(r_{ij})$ as well.

In our algorithm we will move one particle at the time, say the *kth*-particle. Keep this in mind in the discussion to come.

Efficient calculations of wave function ratios

$$\boxed{R_C = \frac{\Psi_C^{\mathrm{new}}}{\Psi_C^{\mathrm{cur}}} = \prod_{i=1}^{k-1} \frac{g_{ik}^{\mathrm{new}}}{g_{ik}^{\mathrm{cur}}} \prod_{i=k+1}^{N} \frac{g_{ki}^{\mathrm{new}}}{g_{ki}^{\mathrm{cur}}}}. \tag{23}$$

For the Padé-Jastrow form

$$\boxed{R_C = \frac{\Psi_C^{\mathrm{new}}}{\Psi_C^{\mathrm{cur}}} = \frac{e^{U_{new}}}{e^{U_{cur}}} = e^{\Delta U},} \tag{24}$$

where

$$\Delta U = \sum_{i=1}^{k-1} \left(f_{ik}^{\mathrm{new}} - f_{ik}^{\mathrm{cur}}\right) + \sum_{i=k+1}^{N} \left(f_{ki}^{\mathrm{new}} - f_{ki}^{\mathrm{cur}}\right) \tag{25}$$

One needs to develop a special algorithm that runs only through the elements of the upper triangular matrix $\boldsymbol{g}$ and have k as an index.

Efficient calculations of wave function ratios

The expression to be derived in the following is of interest when computing the quantum force and the kinetic energy. It has the form

$$\frac{\mathbf{\nabla}_i \Psi_C}{\Psi_C} = \frac{1}{\Psi_C}\frac{\partial \Psi_C}{\partial x_i},$$

for all dimensions and with i running over all particles. For the first derivative only $N-1$ terms survive the ratio because the g-terms that are not differentiated cancel with their corresponding ones in the denominator. Then,

$$\frac{1}{\Psi_C}\frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1}\frac{1}{g_{ik}}\frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^{N}\frac{1}{g_{ki}}\frac{\partial g_{ki}}{\partial x_k}. \tag{26}$$

An equivalent equation is obtained for the exponential form after replacing g_{ij} by $\exp(f_{ij})$, yielding:

$$\frac{1}{\Psi_C}\frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1}\frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^{N}\frac{\partial g_{ki}}{\partial x_k}, \tag{27}$$

with both expressions scaling as $\mathcal{O}(N)$.

Efficient calculations of wave function ratios

Using the identity

$$\frac{\partial}{\partial x_i} g_{ij} = -\frac{\partial}{\partial x_j} g_{ij} \tag{28}$$

on the right hand side terms of Eq. (26) and Eq. (27), we get expressions where all the derivatives acting on the particle are represented by the *second* index of *g*:

$$\boxed{\frac{1}{\Psi_C}\frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1}\frac{1}{g_{ik}}\frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^{N}\frac{1}{g_{ki}}\frac{\partial g_{ki}}{\partial x_i},} \tag{29}$$

and for the exponential case:

$$\boxed{\frac{1}{\Psi_C}\frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1}\frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^{N}\frac{\partial g_{ki}}{\partial x_i}.} \tag{30}$$

Efficient calculations of wave function ratios

For correlation forms depending only on the scalar distances r_{ij} we can use the chain rule. Noting that

$$\frac{\partial g_{ij}}{\partial x_j} = \frac{\partial g_{ij}}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial x_j} = \frac{x_j - x_i}{r_{ij}} \frac{\partial g_{ij}}{\partial r_{ij}}, \tag{31}$$

after substitution in Eq. (29) and Eq. (30) we arrive at

$$\boxed{\frac{1}{\Psi_C} \frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\boldsymbol{r_{ik}}}{r_{ik}} \frac{\partial g_{ik}}{\partial r_{ik}} - \sum_{i=k+1}^{N} \frac{1}{g_{ki}} \frac{\boldsymbol{r_{ki}}}{r_{ki}} \frac{\partial g_{ki}}{\partial r_{ki}}.} \tag{32}$$

Efficient calculations of wave function ratios

Note that for the Padé-Jastrow form we can set $g_{ij} \equiv g(r_{ij}) = e^{f(r_{ij})} = e^{f_{ij}}$ and

$$\frac{\partial g_{ij}}{\partial r_{ij}} = g_{ij}\frac{\partial f_{ij}}{\partial r_{ij}}. \tag{33}$$

Therefore,

$$\boxed{\frac{1}{\Psi_C}\frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1}\frac{\mathbf{r_{ik}}}{r_{ik}}\frac{\partial f_{ik}}{\partial r_{ik}} - \sum_{i=k+1}^{N}\frac{\mathbf{r_{ki}}}{r_{ki}}\frac{\partial f_{ki}}{\partial r_{ki}},} \tag{34}$$

where

$$\mathbf{r}_{ij} = |\mathbf{r}_j - \mathbf{r}_i| = (x_j - x_i)\mathbf{e}_1 + (y_j - y_i)\mathbf{e}_2 + (z_j - z_i)\mathbf{e}_3 \tag{35}$$

is the vectorial distance. When the correlation function is the *linear Padé-Jastrow* we set

$$f_{ij} = \frac{ar_{ij}}{(1+\beta r_{ij})}, \tag{36}$$

which yields the analytical expression

$$\boxed{\frac{\partial f_{ij}}{\partial r_{ij}} = \frac{a}{(1+\beta r_{ij})^2}}. \tag{37}$$

Efficient calculations of wave function ratios

Computing the $\nabla^2\Psi_C/\Psi_C$ ratio

$$\boldsymbol{\nabla}_k\Psi_C = \sum_{i=1}^{k-1}\frac{1}{g_{ik}}\boldsymbol{\nabla}_k g_{ik} + \sum_{i=k+1}^{N}\frac{1}{g_{ki}}\boldsymbol{\nabla}_k g_{ki}.$$

After multiplying by Ψ_C and taking the gradient on both sides we get,

$$\begin{aligned}\nabla_k^2\Psi_C &= \boldsymbol{\nabla}_k\Psi_C\cdot\left(\sum_{i=1}^{k-1}\frac{1}{g_{ik}}\boldsymbol{\nabla}_k g_{ik} + \sum_{i=k+1}^{N}\frac{1}{g_{ki}}\boldsymbol{\nabla}_k g_{ki}\right)\\ &\quad + \Psi_C\nabla_k\cdot\left(\sum_{i=k+1}^{N}\frac{1}{g_{ki}}\boldsymbol{\nabla}_k g_{ki} + \sum_{i=k+1}^{N}\frac{1}{g_{ki}}\boldsymbol{\nabla}_k g_{ki}\right)\\ &= \Psi_C\left(\frac{\boldsymbol{\nabla}_k\Psi_C}{\Psi_C}\right)^2 + \Psi_C\nabla_k\cdot\left(\sum_{i=k+1}^{N}\frac{1}{g_{ki}}\boldsymbol{\nabla}_k g_{ki} + \sum_{i=k+1}^{N}\frac{1}{g_{ki}}\boldsymbol{\nabla}_k g_{ki}\right). \qquad (38)\end{aligned}$$

Efficient calculations of wave function ratios

Now,

$$
\begin{aligned}
\nabla_k \cdot \left(\frac{1}{g_{ik}}\nabla_k g_{ik}\right) &= \nabla_k\left(\frac{1}{g_{ik}}\right)\cdot\nabla_k g_{ik} + \frac{1}{g_{ik}}\nabla_k\cdot\nabla_k g_{ik} \\
&= -\frac{1}{g_{ik}^2}\nabla_k g_{ik}\cdot\nabla_k g_{ik} + \frac{1}{g_{ik}}\nabla_k\cdot\left(\frac{\mathbf{r}_{ik}}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right) \\
&= -\frac{1}{g_{ik}^2}(\nabla_k g_{ik})^2 \\
&+ \frac{1}{g_{ik}}\left[\nabla_k\left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)\cdot\mathbf{r}_{ik} + \left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)\nabla_k\cdot\mathbf{r}_{ik}\right] \\
&= -\frac{1}{g_{ik}^2}\left(\frac{\mathbf{r}_{ik}}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)^2 \\
&+ \frac{1}{g_{ik}}\left[\nabla_k\left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)\cdot\mathbf{r}_{ik} + \left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)d\right] \\
&= -\frac{1}{g_{ik}^2}\left(\frac{\partial g_{ik}}{\partial r_{ik}}\right)^2 \\
&+ \frac{1}{g_{ik}}\left[\nabla_k\left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)\cdot\mathbf{r}_{ik} + \left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)d\right],
\end{aligned}
\tag{39}
$$

with d being the number of spatial dimensions.

Efficient calculations of wave function ratios

Moreover,

$$\nabla_k \left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right) = \frac{\mathbf{r}_{ik}}{r_{ik}}\frac{\partial}{\partial r_{ik}}\left(\frac{1}{r_{ik}}\frac{\partial g_{ik}}{\partial r_{ik}}\right)$$
$$= \frac{\mathbf{r}_{ik}}{r_{ik}}\left(-\frac{1}{r_{ik}^2}\frac{\partial g_{ik}}{\partial r_{ik}} + \frac{1}{r_{ik}}\frac{\partial^2 g_{ik}}{\partial r_{ik}^2}\right).$$

We finally get

$$\nabla_k \cdot \left(\frac{1}{g_{ik}}\nabla_k g_{ik}\right) = -\frac{1}{g_{ik}^2}\left(\frac{\partial g_{ik}}{\partial r_{ik}}\right)^2 + \frac{1}{g_{ik}}\left[\left(\frac{d-1}{r_{ik}}\right)\frac{\partial g_{ik}}{\partial r_{ik}} + \frac{\partial^2 g_{ik}}{\partial r_{ik}^2}\right].$$

Efficient calculations of wave function ratios

Inserting the last expression in Eq. (38) and after division by Ψ_C we get,

$$
\begin{aligned}
\frac{\nabla_k^2\Psi_C}{\Psi_C} &= \left(\frac{\boldsymbol{\nabla}_k\Psi_C}{\Psi_C}\right)^2 \\
&+ \sum_{i=1}^{k-1} -\frac{1}{g_{ik}^2}\left(\frac{\partial g_{ik}}{\partial r_{ik}}\right)^2 + \frac{1}{g_{ik}}\left[\left(\frac{d-1}{r_{ik}}\right)\frac{\partial g_{ik}}{\partial r_{ik}} + \frac{\partial^2 g_{ik}}{\partial r_{ik}^2}\right] \\
&+ \sum_{i=k+1}^{N} -\frac{1}{g_{ki}^2}\left(\frac{\partial g_{ki}}{\partial r_{ki}}\right)^2 + \frac{1}{g_{ki}}\left[\left(\frac{d-1}{r_{ki}}\right)\frac{\partial g_{ki}}{\partial r_{ki}} + \frac{\partial^2 g_{ki}}{\partial r_{ki}^2}\right]. \qquad (40)
\end{aligned}
$$

Efficient calculations of wave function ratios

For the exponential case we have

$$\frac{\nabla_k^2\Psi_C}{\Psi_C} = \left(\frac{\boldsymbol{\nabla}_k\Psi_C}{\Psi_C}\right)^2$$

$$+\sum_{i=1}^{k-1} -\frac{1}{g_{ik}^2}\left(g_{ik}\frac{\partial f_{ik}}{\partial r_{ik}}\right)^2 + \frac{1}{g_{ik}}\left[\left(\frac{d-1}{r_{ik}}\right)g_{ik}\frac{\partial f_{ik}}{\partial r_{ik}} + \frac{\partial}{\partial r_{ik}}\left(g_{ik}\frac{\partial f_{ik}}{\partial r_{ik}}\right)\right]$$

$$+\sum_{i=k+1}^{N} -\frac{1}{g_{ki}^2}\left(g_{ik}\frac{\partial f_{ki}}{\partial r_{ki}}\right)^2 + \frac{1}{g_{ki}}\left[\left(\frac{d-1}{r_{ki}}\right)g_{ki}\frac{\partial f_{ki}}{\partial r_{ki}} + \frac{\partial}{\partial r_{ki}}\left(g_{ki}\frac{\partial f_{ki}}{\partial r_{ki}}\right)\right].$$

Efficient calculations of wave function ratios

Using

$$\begin{aligned}\frac{\partial}{\partial r_{ik}}\left(g_{ik}\frac{\partial f_{ik}}{\partial r_{ik}}\right) &= \frac{\partial g_{ik}}{\partial r_{ik}}\frac{\partial f_{ik}}{\partial r_{ik}} + g_{ik}\frac{\partial^2 f_{ik}}{\partial r_{ik}^2}\\ &= g_{ik}\frac{\partial f_{ik}}{\partial r_{ik}}\frac{\partial f_{ik}}{\partial r_{ik}} + g_{ik}\frac{\partial^2 f_{ik}}{\partial r_{ik}^2}\\ &= g_{ik}\left(\frac{\partial f_{ik}}{\partial r_{ik}}\right)^2 + g_{ik}\frac{\partial^2 f_{ik}}{\partial r_{ik}^2}\end{aligned}$$

and substituting this result into the equation above gives rise to the final expression,

$$\begin{aligned}\frac{\nabla_k^2\Psi_{PJ}}{\Psi_{PJ}} &= \left(\frac{\mathbf{\nabla}_k\Psi_{PJ}}{\Psi_{PJ}}\right)^2\\ &+ \sum_{i=1}^{k-1}\left[\left(\frac{d-1}{r_{ik}}\right)\frac{\partial f_{ik}}{\partial r_{ik}} + \frac{\partial^2 f_{ik}}{\partial r_{ik}^2}\right] + \sum_{i=k+1}^{N}\left[\left(\frac{d-1}{r_{ki}}\right)\frac{\partial f_{ki}}{\partial r_{ki}} + \frac{\partial^2 f_{ki}}{\partial r_{ki}^2}\right]. \qquad (41)\end{aligned}$$

Summing up: Bringing it all together, quantum force

The general derivative formula of the Jastrow factor is

$$\frac{1}{\Psi_C}\frac{\partial \Psi_C}{\partial x_k} = \sum_{i=1}^{k-1}\frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^{N}\frac{\partial g_{ki}}{\partial x_k}$$

However, with our

$$\Psi_C = \prod_{i<j} g(r_{ij}) = \exp\left\{\sum_{i<j}\frac{ar_{ij}}{1+\beta r_{ij}}\right\},$$

the gradient needed for the quantum force and local energy is easy to compute. We get for particle k

$$\frac{\nabla_k \Psi_C}{\Psi_C} = \sum_{j\neq k}\frac{\mathbf{r}_{kj}}{r_{kj}}\frac{a}{(1+\beta r_{kj})^2},$$

which is rather easy to code. Remember to sum over all particles when you compute the local energy.

Summing up: Bringing it all together, Local energy

The second derivative of the Jastrow factor divided by the Jastrow factor (the way it enters the kinetic energy) is

$$\left[\frac{\nabla^2\Psi_C}{\Psi_C}\right]_x = 2\sum_{k=1}^{N}\sum_{i=1}^{k-1}\frac{\partial^2 g_{ik}}{\partial x_k^2} + \sum_{k=1}^{N}\left(\sum_{i=1}^{k-1}\frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^{N}\frac{\partial g_{ki}}{\partial x_i}\right)^2$$

But we have a simple form for the function, namely

$$\Psi_C = \prod_{i<j}\exp f(r_{ij}) = \exp\left\{\sum_{i<j}\frac{ar_{ij}}{1+\beta r_{ij}}\right\},$$

and it is easy to see that for particle *k* we have

$$\frac{\nabla_k^2\Psi_C}{\Psi_C} = \sum_{ij\neq k}\frac{(\mathbf{r}_k-\mathbf{r}_i)(\mathbf{r}_k-\mathbf{r}_j)}{r_{ki}r_{kj}}f'(r_{ki})f'(r_{kj}) + \sum_{j\neq k}\left(f''(r_{kj}) + \frac{2}{r_{kj}}f'(r_{kj})\right)$$

Bringing it all together, Local energy

Using

$$f(r_{ij}) = \frac{ar_{ij}}{1+\beta r_{ij}},$$

and $g'(r_{kj}) = dg(r_{kj})/dr_{kj}$ and $g''(r_{kj}) = d^2g(r_{kj})/dr_{kj}^2$ we find that for particle k we have

$$\frac{\nabla_k^2 \Psi_C}{\Psi_C} = \sum_{ij\neq k} \frac{(\mathbf{r}_k - \mathbf{r}_i)(\mathbf{r}_k - \mathbf{r}_j)}{r_{ki}r_{kj}} \frac{a}{(1+\beta r_{ki})^2} \frac{a}{(1+\beta r_{kj})^2} + \sum_{j\neq k} \left(\frac{2a}{r_{kj}(1+\beta r_{kj})^2} - \frac{2a\beta}{(1+\beta r_{kj})^3} \right)$$

Important feature

For the correlation part

$$\Psi_C = \prod_{i<j} g(r_{ij}) = \exp\left\{\sum_{i<j} \frac{ar_{ij}}{1+\beta r_{ij}}\right\},$$

we need to take into account whether electrons have equal or opposite spins since we have to obey the electron-electron cusp condition as well. For Beryllium you can fix electrons 1 and 2 to have spin up while electrons 3 and 4 have spin down. When the electrons have equal spins

$$a = 1/4,$$

while for opposite spins (as for the ground state of helium)

$$a = 1/2.$$

Importance sampling, what we want to do

We need to replace the brute force Metropolis algorithm with a walk in coordinate space biased by the trial wave function. This approach is based on the Fokker-Planck equation and the Langevin equation for generating a trajectory in coordinate space. This is explained later.
For a diffusion process characterized by a time-dependent probability density $P(x,t)$ in one dimension the Fokker-Planck equation reads (for one particle/walker)

$$\frac{\partial P}{\partial t} = D\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} - F\right) P(x,t),$$

where F is a drift term and D is the diffusion coefficient.
The new positions in coordinate space are given as the solutions of the Langevin equation using Euler's method, namely, we go from the Langevin equation

$$\frac{\partial x(t)}{\partial t} = DF(x(t)) + \eta,$$

with η a random variable, yielding a new position

$$y = x + DF(x)\Delta t + \xi,$$

where ξ is gaussian random variable and Δt is a chosen time step.

Importance sampling, what we want to do

The process of isotropic diffusion characterized by a time-dependent probability density $P(\boldsymbol{x}, t)$ obeys (as an approximation) the so-called Fokker-Planck equation

$$\frac{\partial P}{\partial t} = \sum_i D \frac{\partial}{\partial \boldsymbol{x_i}} \left(\frac{\partial}{\partial \boldsymbol{x_i}} - \boldsymbol{F_i} \right) P(\boldsymbol{x}, t),$$

where $\boldsymbol{F_i}$ is the i^{th} component of the drift term (drift velocity) caused by an external potential, and D is the diffusion coefficient. The convergence to a stationary probability density can be obtained by setting the left hand side to zero. The resulting equation will be satisfied if and only if all the terms of the sum are equal zero,

$$\frac{\partial^2 P}{\partial \boldsymbol{x_i}^2} = P \frac{\partial}{\partial \boldsymbol{x_i}} \boldsymbol{F_i} + \boldsymbol{F_i} \frac{\partial}{\partial \boldsymbol{x_i}} P.$$

Importance sampling, what we want to do

The drift vector should be of the form $\boldsymbol{F} = g(\boldsymbol{x})\frac{\partial P}{\partial \boldsymbol{x}}$. Then,

$$\frac{\partial^2 P}{\partial \boldsymbol{x_i}^2} = P\frac{\partial g}{\partial P}\left(\frac{\partial P}{\partial \boldsymbol{x_i}}\right)^2 + Pg\frac{\partial^2 P}{\partial \boldsymbol{x_i}^2} + g\left(\frac{\partial P}{\partial \boldsymbol{x_i}}\right)^2.$$

The condition of stationary density means that the left hand side equals zero. In other words, the terms containing first and second derivatives have to cancel each other. It is possible only if $g = \frac{1}{P}$, which yields

$$\boxed{\boldsymbol{F} = 2\frac{1}{\Psi_T}\nabla\Psi_T,} \tag{42}$$

which is known as the so-called *quantum force*. This term is responsible for pushing the walker towards regions of configuration space where the trial wave function is large, increasing the efficiency of the simulation in contrast to the Metropolis algorithm where the walker has the same probability of moving in every direction.

Importance Sampling

The Fokker-Planck equation yields a (the solution to the equation) transition probability given by the Green's function

$$G(y,x,\Delta t) = \frac{1}{(4\pi D\Delta t)^{3N/2}} \exp\left(-(y-x-D\Delta tF(x))^2/4D\Delta t\right)$$

which in turn means that our brute force Metropolis algorithm

$$A(y,x) = \min(1, q(y,x))),$$

with $q(y,x) = |\Psi_T(y)|^2/|\Psi_T(x)|^2$ is now replaced by

$$q(y,x) = \frac{G(x,y,\Delta t)|\Psi_T(y)|^2}{G(y,x,\Delta t)|\Psi_T(x)|^2}$$

See program vmc_be.cpp for example. Read more in Thijssen's text chapters 8.8 and 12.2.

Importance sampling, new positions in function vmc_be.cpp

```
for (variate=1; variate <= max_variations; variate
    ++){
    // initialisations of variational parameters
       and energies
    beta += 0.1;
    energy = energy2 = delta_e = 0.0;
    //  initial trial position, note calling with
       beta
    for (i = 0; i < number_particles; i++) {
      for ( j=0; j < dimension; j++) {
        r_old[i][j] = gaussian_deviate(&idum)*sqrt(
            timestep);
      }
    }
    wfold = wave_function(r_old, beta);
    quantum_force(r_old, qforce_old, beta, wfold);
```

Importance sampling, new positions in function vmc_be.cpp

```
// loop over monte carlo cycles
for (cycles = 1; cycles <= number_cycles;
    cycles++){
  // new position
  for (i = 0; i < number_particles; i++) {
    for ( j=0; j < dimension; j++) {
      // gaussian deviate to compute new
          positions using a given timestep
      r_new[i][j] = r_old[i][j] +
          gaussian_deviate(&idum)*sqrt(timestep
          )+qforce_old[i][j]*timestep*D;
```

Importance sampling, new positions in function vmc_be.cpp

```
//  we move only one particle at the time
          for (k = 0; k < number_particles; k++) {
            if ( k != i) {
              for ( j=0; j < dimension; j++) {
                r_new[k][j] = r_old[k][j];
              }
            }
          }
          //          wave_function_onemove(r_new,
             qforce_new, &wfnew, beta);
          wfnew = wave_function(r_new, beta);
          quantum_force(r_new, qforce_new, beta,
             wfnew);
```

Importance sampling, new positions in function vmc_be.cpp

```
//  we compute the log of the ratio of the
    greens functions to be used in the
//  Metropolis-Hastings algorithm
greensfunction = 0.0;
for ( j=0; j < dimension; j++) {
  greensfunction += 0.5*(qforce_old[i][j]+
       qforce_new[i][j])*
     (D*timestep*0.5*(qforce_old[i][j]-
        qforce_new[i][j])-r_new[i][j]+r_old
        [i][j]);
}
greensfunction = exp(greensfunction);
```

Importance sampling, new positions in function vmc_be.cpp

```
// The Metropolis test is performed by
    moving one particle at the time
if(ran2(&idum) <= greensfunction*wfnew*
    wfnew/wfold/wfold ) {
  for ( j=0; j < dimension; j++) {
    r_old[i][j] = r_new[i][j];
    qforce_old[i][j] = qforce_new[i][j];
  }
  wfold = wfnew;
  .....
```

Importance sampling, Quantum force in function vmc_be.cpp

```
void  quantum_force(double **r, double **qforce,
    double beta, double wf)
{
  int i, j;
  double wfminus, wfplus;
  double **r_plus, **r_minus;

  r_plus = (double **) matrix( number_particles,
      dimension, sizeof(double));
  r_minus = (double **) matrix( number_particles,
      dimension, sizeof(double));
  for (i = 0; i < number_particles; i++) {
    for ( j=0; j < dimension; j++) {
      r_plus[i][j] = r_minus[i][j] = r[i][j];
    }
  }
...
```

Importance sampling, Quantum force in function vmc_be.cpp

```
    // compute the first derivative
    for (i = 0; i < number_particles; i++) {
      for (j = 0; j < dimension; j++) {
        r_plus[i][j] = r[i][j]+h;
        r_minus[i][j] = r[i][j]-h;
        wfminus = wave_function(r_minus, beta);
        wfplus  = wave_function(r_plus, beta);
        qforce[i][j] = (wfplus-wfminus)*2.0/wf/h;
        r_plus[i][j] = r[i][j];
        r_minus[i][j] = r[i][j];
      }
    }

  } // end of quantum_force function
```

Topics for Week 6, February 8-12

Importance sampling, Fokker-Planck and Langevin equations

- Repetition from last week
- Derivation of the Fokker-Planck and the Langevin equations (Background material
- Importance sampling, further discussion of codes
- Begin discussion of blocking

Project work this week: finalize 1a and 1b (only importance sampling part). Next week we discuss blocking as a tool to perform statistical analysis of MonteCarlo data.

Your tasks from the previous week plus new tasks

- Implement the closed form expression for the local energy
- Convince yourself that the closed form expressions are correct, see slides from last week.
- Implement the above expressions for systems with more than two electrons.
- Implement importance sampling, see code vmc_be.cpp.
- Finish part 1a and begin part 1b. Blocking will be discussed this week.
- You need to produce random numbers with a Gaussian distribution.
- Reading task: Thijssen's text chapters 8.8 and 12.2. To be discussed today.
- Task to next week: Finish coding importance sampling in 1b.

Importance sampling, Fokker-Planck and Langevin equation

A stochastic process is simply a function of two variables, one is the time, the other is a stochastic variable X, defined by specifying

- the set $\{x\}$ of possible values for X;
- the probability distribution, $w_X(x)$, over this set, or briefly $w(x)$

The set of values $\{x\}$ for X may be discrete, or continuous. If the set of values is continuous, then $w_X(x)$ is a probability density so that $w_X(x)dx$ is the probability that one finds the stochastic variable X to have values in the range $[x, x + dx]$.

Importance sampling, Fokker-Planck and Langevin equation

An arbitrary number of other stochastic variables may be derived from X. For example, any Y given by a mapping of X, is also a stochastic variable. The mapping may also be time-dependent, that is, the mapping depends on an additional variable t

$$Y_X(t) = f(X, t).$$

The quantity $Y_X(t)$ is called a random function, or, since t often is time, a stochastic process. A stochastic process is a function of two variables, one is the time, the other is a stochastic variable X. Let x be one of the possible values of X then

$$y(t) = f(x, t),$$

is a function of t, called a sample function or realization of the process. In physics one considers the stochastic process to be an ensemble of such sample functions.

Importance sampling, Fokker-Planck and Langevin equation

For many physical systems initial distributions of a stochastic variable y tend to equilibrium distributions: $w(y,t) \rightarrow w_0(y)$ as $t \rightarrow \infty$. In equilibrium detailed balance constrains the transition rates

$$W(y \rightarrow y')w(y) = W(y' \rightarrow y)w_0(y),$$

where $W(y' \rightarrow y)$ is the probability, per unit time, that the system changes from a state $|y\rangle$, characterized by the value y for the stochastic variable Y , to a state $|y'\rangle$.

Note that for a system in equilibrium the transition rate $W(y' \rightarrow y)$ and the reverse $W(y \rightarrow y')$ may be very different.

Importance sampling, Fokker-Planck and Langevin equation

Consider, for instance, a simple system that has only two energy levels $\epsilon_0 = 0$ and $\epsilon_1 = \Delta E$.
For a system governed by the Boltzmann distribution we find (the partition function has been taken out)

$$W(0 \rightarrow 1) \exp -\epsilon_0/kT = W(1 \rightarrow 0) \exp -\epsilon_1/kT$$

We get then

$$\frac{W(1 \rightarrow 0)}{W(0 \rightarrow 1)} = \exp -\Delta E/kT,$$

which goes to zero when T tends to zero.

Importance sampling, Fokker-Planck and Langevin equation

If we assume a discrete set events, our initial probability distribution function can be given by

$$w_i(0) = \delta_{i,0},$$

and its time-development after a given time step $\Delta t = \epsilon$ is

$$w_i(t) = \sum_j W(j \rightarrow i) w_j(t = 0).$$

The continuous analog to $w_i(0)$ is

$$w(\mathbf{x}) \rightarrow \delta(\mathbf{x}), \qquad (43)$$

where we now have generalized the one-dimensional position x to a generic-dimensional vector $\mathbf{x}$. The Kroenecker δ function is replaced by the δ distribution function $\delta(\mathbf{x})$ at $t = 0$.

Importance sampling, Fokker-Planck and Langevin equation

The transition from a state j to a state i is now replaced by a transition to a state with position $\mathbf{y}$ from a state with position $\mathbf{x}$. The discrete sum of transition probabilities can then be replaced by an integral and we obtain the new distribution at a time $t+\Delta t$ as

$$w(\mathbf{y},t+\Delta t) = \int W(\mathbf{y},t+\Delta t|\mathbf{x},t)w(\mathbf{x},t)d\mathbf{x}, \tag{44}$$

and after m time steps we have

$$w(\mathbf{y},t+m\Delta t) = \int W(\mathbf{y},t+m\Delta t|\mathbf{x},t)w(\mathbf{x},t)d\mathbf{x}. \tag{45}$$

When equilibrium is reached we have

$$w(\mathbf{y}) = \int W(\mathbf{y}|\mathbf{x},t)w(\mathbf{x})d\mathbf{x}, \tag{46}$$

that is no time-dependence. Note our change of notation for W

Importance sampling, Fokker-Planck and Langevin equation

We can solve the equation for $w(\mathbf{y}, t)$ by making a Fourier transform to momentum space. The PDF $w(\mathbf{x}, t)$ is related to its Fourier transform $\tilde{w}(\mathbf{k}, t)$ through

$$w(\mathbf{x}, t) = \int_{-\infty}^{\infty} d\mathbf{k} \exp\left(i\mathbf{k}\mathbf{x}\right)\tilde{w}(\mathbf{k}, t), \tag{47}$$

and using the definition of the δ-function

$$\delta(\mathbf{x}) = \frac{1}{2\pi} \int_{-\infty}^{\infty} d\mathbf{k} \exp\left(i\mathbf{k}\mathbf{x}\right), \tag{48}$$

we see that

$$\tilde{w}(\mathbf{k}, 0) = 1/2\pi. \tag{49}$$

Importance sampling, Fokker-Planck and Langevin equation

We can then use the Fourier-transformed diffusion equation

$$\frac{\partial \tilde{w}(\mathbf{k},t)}{\partial t} = -D\mathbf{k}^2 \tilde{w}(\mathbf{k},t), \tag{50}$$

with the obvious solution

$$\tilde{w}(\mathbf{k},t) = \tilde{w}(\mathbf{k},0)\exp\left[-(D\mathbf{k}^2 t)\right) = \frac{1}{2\pi}\exp\left[-(D\mathbf{k}^2 t)\right]. \tag{51}$$

Importance sampling, Fokker-Planck and Langevin equation

Using Eq. (47) we obtain

$$w(\mathbf{x},t)=\int_{-\infty}^{\infty}d\mathbf{k}\exp\left[i\mathbf{k}\mathbf{x}\right]\frac{1}{2\pi}\exp\left[-(D\mathbf{k}^2t)\right]=\frac{1}{\sqrt{4\pi Dt}}\exp\left[-(\mathbf{x}^2/4Dt)\right], \qquad (52)$$

with the normalization condition

$$\int_{-\infty}^{\infty}w(\mathbf{x},t)d\mathbf{x}=1. \qquad (53)$$

Importance sampling, Fokker-Planck and Langevin equation

It is rather easy to verify by insertion that Eq. (52) is a solution of the diffusion equation. The solution represents the probability of finding our random walker at position $\mathbf{x}$ at time t if the initial distribution was placed at $\mathbf{x} = 0$ at $t = 0$.
There is another interesting feature worth observing. The discrete transition probability W itself is given by a binomial distribution. The results from the central limit theorem state that transition probability in the limit $n \to \infty$ converges to the normal distribution. It is then possible to show that

$$W(il - jl, n\epsilon) \to W(\mathbf{y}, t + \Delta t|\mathbf{x}, t) = \frac{1}{\sqrt{4\pi D\Delta t}} \exp\left[-((\mathbf{y} - \mathbf{x})^2/4D\Delta t)\right], \qquad (54)$$

and that it satisfies the normalization condition and is itself a solution to the diffusion equation.

Importance sampling, Fokker-Planck and Langevin equation

Let us now assume that we have three PDFs for times $t_0 < t' < t$, that is $w(\mathbf{x}_0, t_0)$, $w(\mathbf{x}', t')$ and $w(\mathbf{x}, t)$. We have then

$$w(\mathbf{x}, t) = \int_{-\infty}^{\infty} W(\mathbf{x}.t|\mathbf{x}'.t')w(\mathbf{x}', t')d\mathbf{x}',$$

and

$$w(\mathbf{x}, t) = \int_{-\infty}^{\infty} W(\mathbf{x}.t|\mathbf{x}_0.t_0)w(\mathbf{x}_0, t_0)d\mathbf{x}_0,$$

and

$$w(\mathbf{x}', t') = \int_{-\infty}^{\infty} W(\mathbf{x}'.t'|\mathbf{x}_0, t_0)w(\mathbf{x}_0, t_0)d\mathbf{x}_0.$$

Importance sampling, Fokker-Planck and Langevin equation

We can combine these equations and arrive at the famous Einstein-Smoluchenski-Kolmogorov-Chapman (ESKC) relation

$$W(\mathbf{x}t|\mathbf{x}_0 t_0) = \int_{-\infty}^{\infty} W(\mathbf{x},t|\mathbf{x}',t')W(\mathbf{x}',t'|\mathbf{x}_0,t_0)d\mathbf{x}'.$$

We can replace the spatial dependence with a dependence upon say the velocity (or momentum), that is we have

$$W(\mathbf{v},t|\mathbf{v}_0,t_0) = \int_{-\infty}^{\infty} W(\mathbf{v},t|\mathbf{v}',t')W(\mathbf{v}',t'|\mathbf{v}_0,t_0)d\mathbf{x}'.$$

Importance sampling, Fokker-Planck and Langevin equation

We will now derive the Fokker-Planck equation. We start from the ESKC equation

$$W(\mathbf{x}, t|\mathbf{x}_0, t_0) = \int_{-\infty}^{\infty} W(\mathbf{x}, t|\mathbf{x}', t')W(\mathbf{x}', t'|\mathbf{x}_0, t_0)d\mathbf{x}'.$$

Define $s = t' - t_0$, $\tau = t - t'$ and $t - t_0 = s + \tau$. We have then

$$W(\mathbf{x}, s+\tau|\mathbf{x}_0) = \int_{-\infty}^{\infty} W(\mathbf{x}, \tau|\mathbf{x}')W(\mathbf{x}', s|\mathbf{x}_0)d\mathbf{x}'.$$

Importance sampling, Fokker-Planck and Langevin equation

Assume now that τ is very small so that we can make an expansion in terms of a small step *xi*, with $\mathbf{x}' = \mathbf{x} - \xi$, that is

$$W(\mathbf{x}, s|\mathbf{x}_0) + \frac{\partial W}{\partial s}\tau + O(\tau^2) = \int_{-\infty}^{\infty} W(\mathbf{x}, \tau|\mathbf{x} - \xi)W(\mathbf{x} - \xi, s|\mathbf{x}_0)d\mathbf{x}'.$$

We assume that $W(\mathbf{x}, \tau|\mathbf{x} - \xi)$ takes non-negligible values only when ξ is small. This is just another way of stating the Master equation!!

Importance sampling, Fokker-Planck and Langevin equation

We say thus that **x** changes only by a small amount in the time interval τ. This means that we can make a Taylor expansion in terms of ξ, that is we expand

$$W(\mathbf{x},\tau|\mathbf{x}-\xi)W(\mathbf{x}-\xi,s|\mathbf{x}_0)=\sum_{n=0}^{\infty}\frac{(-\xi)^n}{n!}\frac{\partial^n}{\partial x^n}\left[W(\mathbf{x}+\xi,\tau|\mathbf{x})W(\mathbf{x},s|\mathbf{x}_0)\right].$$

We can then rewrite the ESKC equation as

$$\frac{\partial W}{\partial s}\tau=-W(\mathbf{x},s|\mathbf{x}_0)+\sum_{n=0}^{\infty}\frac{(-\xi)^n}{n!}\frac{\partial^n}{\partial x^n}\left[W(\mathbf{x},s|\mathbf{x}_0)\int_{-\infty}^{\infty}\xi^n W(\mathbf{x}+\xi,\tau|\mathbf{x})d\xi\right].$$

We have neglected higher powers of τ and have used that for $n=0$ we get simply $W(\mathbf{x},s|\mathbf{x}_0)$ due to normalization.

Importance sampling, Fokker-Planck and Langevin equation

We say thus that $\mathbf{x}$ changes only by a small amount in the time interval τ. This means that we can make a Taylor expansion in terms of ξ, that is we expand

$$W(\mathbf{x},\tau|\mathbf{x}-\xi)W(\mathbf{x}-\xi,s|\mathbf{x}_0) = \sum_{n=0}^{\infty}\frac{(-\xi)^n}{n!}\frac{\partial^n}{\partial x^n}\left[W(\mathbf{x}+\xi,\tau|\mathbf{x})W(\mathbf{x},s|\mathbf{x}_0)\right].$$

We can then rewrite the ESKC equation as

$$\frac{\partial W(\mathbf{x},s|\mathbf{x}_0)}{\partial s}\tau = -W(\mathbf{x},s|\mathbf{x}_0)+\sum_{n=0}^{\infty}\frac{(-\xi)^n}{n!}\frac{\partial^n}{\partial x^n}\left[W(\mathbf{x},s|\mathbf{x}_0)\int_{-\infty}^{\infty}\xi^n W(\mathbf{x}+\xi,\tau|\mathbf{x})d\xi\right].$$

We have neglected higher powers of τ and have used that for $n=0$ we get simply $W(\mathbf{x},s|\mathbf{x}_0)$ due to normalization.

Importance sampling, Fokker-Planck and Langevin equation

We simplify the above by introducing the moments

$$M_n = \frac{1}{\tau}\int_{-\infty}^{\infty} \xi^n W(\mathbf{x}+\xi,\tau|\mathbf{x})d\xi = \frac{\langle[\Delta x(\tau)]^n\rangle}{\tau},$$

resulting in

$$\frac{\partial W(\mathbf{x},s|\mathbf{x}_0)}{\partial s} = \sum_{n=1}^{\infty}\frac{(-\xi)^n}{n!}\frac{\partial^n}{\partial x^n}\left[W(\mathbf{x},s|\mathbf{x}_0)M_n\right].$$

Importance sampling, Fokker-Planck and Langevin equation

When $\tau \to 0$ we assume that $\langle [\Delta x(\tau)]^n \rangle \to 0$ more rapidly than τ itself if $n > 2$. When τ is much larger than the standard correlation time of system then M_n for $n > 2$ can normally be neglected. This means that fluctuations become negligible at large time scales.
If we neglect such terms we can rewrite the ESKC equation as

$$\frac{\partial W(\mathbf{x}, s|\mathbf{x}_0)}{\partial s} = -\frac{\partial M_1 W(\mathbf{x}, s|\mathbf{x}_0)}{\partial x} + \frac{1}{2}\frac{\partial^2 M_2 W(\mathbf{x}, s|\mathbf{x}_0)}{\partial x^2}.$$

Importance sampling, Fokker-Planck and Langevin equation

In a more compact form we have

$$\frac{\partial W}{\partial s} = -\frac{\partial M_1 W}{\partial x} + \frac{1}{2}\frac{\partial^2 M_2 W}{\partial x^2},$$

which is the Fokker-Planck equation! It is trivial to replace position with velocity (momentum).

Langevin equation

Consider a particle suspended in a liquid. On its path through the liquid it will continuously collide with the liquid molecules. Because on average the particle will collide more often on the front side than on the back side, it will experience a systematic force proportional with its velocity, and directed opposite to its velocity. Besides this systematic force the particle will experience a stochastic force $\boldsymbol{F}(t)$. The equations of motion then read

$$\frac{d\boldsymbol{r}}{dt} = \boldsymbol{v},$$

$$\frac{d\boldsymbol{v}}{dt} = -\xi\boldsymbol{v} + \boldsymbol{F}.$$

Langevin equation

From hydrodynamics we know that the friction constant ξ is given by

$$\xi = 6\pi\eta a/m$$

where η is the viscosity of the solvent and a is the radius of the particle.
Solving the second equation in the previous slide we get

$$\boldsymbol{v}(t) = \boldsymbol{v}_0 e^{-\xi t} + \int_0^t d\tau e^{-\xi(t-\tau)} \boldsymbol{F}(\tau).$$

Langevin equation

If we want to get some useful information out of this, we have to average over all possible realizations of $\boldsymbol{F}(t)$, with the initial velocity as a condition. A useful quantity for example is

$$\langle \boldsymbol{v}(t) \cdot \boldsymbol{v}(t) \rangle_{\boldsymbol{v}_0} = v_0^{-\xi 2t} + 2 \int_0^t d\tau e^{-\xi(2t-\tau)} \boldsymbol{v}_0 \cdot \langle \boldsymbol{F}(\tau) \rangle_{\boldsymbol{v}_0}$$

$$+ \int_0^t d\tau' \int_0^t d\tau e^{-\xi(2t-\tau-\tau')} \langle \boldsymbol{F}(\tau) \cdot \boldsymbol{F}(\tau') \rangle_{\boldsymbol{v}_0}.$$

Langevin equation

In order to continue we have to make some assumptions about the conditional averages of the stochastic forces. In view of the chaotic character of the stochastic forces the following assumptions seem to be appropriate

$$\langle \boldsymbol{F}(t) \rangle = 0,$$

$$\langle \boldsymbol{F}(t) \cdot \boldsymbol{F}(t') \rangle_{\boldsymbol{v}_0} = C_{\boldsymbol{v}_0} \delta(t - t').$$

Langevin equation

We omit the subscript $\boldsymbol{v}_0$, when the quantity of interest turns out to be independent of $\boldsymbol{v}_0$. Using the last three equations we get

$$\langle \boldsymbol{v}(t) \cdot \boldsymbol{v}(t) \rangle_{\boldsymbol{v}_0} = v_0^2 e^{-2\xi t} + \frac{C_{\boldsymbol{v}_0}}{2\xi}(1 - e^{-2\xi t}).$$

For large t this should be equal to 3kT/m, from which it follows that

$$\langle \boldsymbol{F}(t) \cdot \boldsymbol{F}(t') \rangle = 6\frac{kT}{m}\xi\delta(t - t').$$

This result is called the fluctuation-dissipation theorem .

Langevin equation

Integrating

$$\boldsymbol{v}(t) = \boldsymbol{v}_0 e^{-\xi t} + \int_0^t d\tau e^{-\xi(t-\tau)} \boldsymbol{F}(\tau),$$

we get

$$\boldsymbol{r}(t) = \boldsymbol{r}_0 + \boldsymbol{v}_0 \frac{1}{\xi}(1 - e^{-\xi t}) + \int_0^t d\tau \int_0^\tau \tau' e^{-\xi(\tau-\tau')} \boldsymbol{F}(\tau'),$$

from which we calculate the mean square displacement

$$\langle (\boldsymbol{r}(t) - \boldsymbol{r}_0)^2 \rangle_{\boldsymbol{v}_0} = \frac{v_0^2}{\xi}(1 - e^{-\xi t})^2 + \frac{3kT}{m\xi^2}(2\xi t - 3 + 4e^{-\xi t} - e^{-2\xi t}).$$

Langevin equation

For very large t this becomes

$$\langle (\boldsymbol{r}(t) - \boldsymbol{r}_0)^2 \rangle = \frac{6kT}{m\xi} t$$

from which we get the Einstein relation

$$D = \frac{kT}{m\xi}$$

where we have used $\langle (\boldsymbol{r}(t) - \boldsymbol{r}_0)^2 \rangle = 6Dt$.

Topics for Week 7, February 15-19

Blocking, Conjugate gradient method and onebody densities

- Repetition from last week
- Importance sampling, further discussion of codes
- Begin discussion of blocking and conjugate gradient method
- Definition of onebody densities.

Project work this week: finalize 1a and 1b till next week. Start with 1c and 1d next week.

Your tasks for this week and next week

- Implement importance sampling, see code vmc_be.cpp.
- Start implementing blocking.
- Programming task for next week: Finish coding 1b.
- Program the onebody density, that is part 1c.
- Read about the conjugate gradient method in Thijssen's text or use the slides. Alternatively, chapter 10 of Numerical Recipes gives a very good overview. The code dfpmin is taken from chapter 10.7 of Numerical Recipes.

Definition of onebody density, needed in 1c of project 1

The hydrogenic like functions for so-called s waves ($l=0$) are rather simple since the angular part given by the spherical harmonics is just $1/\sqrt{4\pi}$.
Our radial part of the wave functions are

$$R_{n0}(r) = \left(\frac{2Z}{n}\right)^{3/2}\sqrt{\frac{(n-1)!}{2n\times n!}}L_{n-1}^{1}(\frac{2Z}{n})\exp{(-\frac{Zr}{n})},$$

with energies $-Z^2/2n^2$ and L being the Laguerre polynomials. This means that if we use just the hydrogen-like wave functions our ground state for helium is given by

$$\Phi(\mathbf{r}_1,\mathbf{r}_2) = \frac{1}{4\pi}R_{10}(r_1)R_{10}(r_2) = \frac{1}{4\pi}4Z^3\exp{(-Z(r_1+r_2))},$$

and the onebody density is defined as

$$\rho(\mathbf{r}_1) = \int d\mathbf{r}_2\left|\frac{1}{4\pi}R_{10}(r_1)R_{10}(r_2)\right|^2.$$

Definition of onebody density, needed in 1c of project 1

Note that $d\mathbf{r}_2 = dx_2dy_2dz_2 = r_2^2dr_2\sin(\theta_2)d\theta_2d\phi_2$ in

$$\rho(\mathbf{r}_1) = \int d\mathbf{r}_2 \left|\frac{1}{4\pi}R_{10}(r_1)R_{10}(r_2)\right|^2.$$

The onebody density is isotropic for the $l = 0$ waves, that is we have no angular dependence.
This result, that is with no Monte Carlo calculation should now be compared with the density obtained with your trial wave function, that is (and now we use Cartesian coordinates only)

$$\rho_T(\mathbf{r}_1) = \int d\mathbf{r}_2 \left|\Psi_T(\mathbf{r}_1, \mathbf{r}_2)\right|^2.$$

When you plot the density it is convenient to plot it as function of r_1 only. With your values of x_1, y_1 and z_1 you can then tabulate the density as function of r_1 only. With your optimal wave function, you can then use say Simpson's method to compute the above density.

What is the charge density?

Why blocking?

Statistical analysis

- Monte Carlo simulations can be treated as *computer experiments*
- The results can be analysed with the same statistical tools as we would use analysing experimental data.
- As in all experiments, we are looking for expectation values and an estimate of how accurate they are, i.e., possible sources for errors.

Why blocking?

Statistical analysis

- As in other experiments, Monte Carlo experiments have two classes of errors:
  - Statistical errors
  - Systematical errors
- Statistical errors can be estimated using standard tools from statistics
- Systematical errors are method specific and must be treated differently from case to case. (In VMC a common source is the step length or time step in importance sampling)

Statistics and blocking

The *probability distribution function (PDF)* is a function $p(x)$ on the domain which, in the discrete case, gives us the probability or relative frequency with which these values of X occur:

$$p(x) = \mathrm{Prob}(X = x)$$

In the continuous case, the PDF does not directly depict the actual probability. Instead we define the probability for the stochastic variable to assume any value on an infinitesimal interval around x to be $p(x)dx$. The continuous function $p(x)$ then gives us the *density* of the probability rather than the probability itself. The probability for a stochastic variable to assume any value on a non-infinitesimal interval $[a, b]$ is then just the integral:

$$\mathrm{Prob}(a \leq X \leq b) = \int_a^b p(x)dx$$

Qualitatively speaking, a stochastic variable represents the values of numbers chosen as if by chance from some specified PDF so that the selection of a large set of these numbers reproduces this PDF.

Statistics and blocking

Also of interest to us is the *cumulative probability distribution function (CDF)*, $P(x)$, which is just the probability for a stochastic variable X to assume any value less than x:

$$P(x) = \mathrm{Prob}(X \leq x) = \int_{-\infty}^{x} p(x')dx'$$

The relation between a CDF and its corresponding PDF is then:

$$p(x) = \frac{d}{dx}P(x)$$

Statistics and blocking

A particularly useful class of special expectation values are the *moments*. The *n*-th moment of the PDF *p* is defined as follows:

$$\langle x^n \rangle \equiv \int x^n p(x)\, dx$$

The zero-th moment $\langle 1 \rangle$ is just the normalization condition of *p*. The first moment, $\langle x \rangle$, is called the *mean* of *p* and often denoted by the letter μ:

$$\langle x \rangle = \mu \equiv \int x p(x)\, dx$$

Statistics and blocking

A special version of the moments is the set of *central moments*, the n-th central moment defined as:

$$\langle (x - \langle x \rangle)^n \rangle \equiv \int (x - \langle x \rangle)^n p(x)\, dx$$

The zero-th and first central moments are both trivial, equal 1 and 0, respectively. But the second central moment, known as the *variance* of p, is of particular interest. For the stochastic variable X, the variance is denoted as σ_X^2 or $\mathrm{Var}(X)$:

$$\begin{aligned}
\sigma_X^2 \;=\; \mathrm{Var}(X) &= \langle (x - \langle x \rangle)^2 \rangle = \int (x - \langle x \rangle)^2 p(x)\, dx \\
&= \int \left(x^2 - 2x\langle x \rangle \; + \langle x \rangle^2 \right) p(x)\, dx \\
&= \langle x^2 \rangle - 2\langle x \rangle \langle x \rangle + \langle x \rangle^2 \\
&= \langle x^2 \rangle - \langle x \rangle^2
\end{aligned}$$

The square root of the variance, $\sigma = \sqrt{\langle (x - \langle x \rangle)^2 \rangle}$ is called the *standard deviation* of p. It is clearly just the RMS (root-mean-square) value of the deviation of the PDF from its mean value, interpreted qualitatively as the “spread” of p around its mean.

Statistics and blocking

Another important quantity is the so called covariance, a variant of the above defined variance. Consider again the set $\{X_i\}$ of n stochastic variables (not necessarily uncorrelated) with the multivariate PDF $P(x_1, \dots, x_n)$. The *covariance* of two of the stochastic variables, X_i and X_j, is defined as follows:

$$\begin{aligned} \mathrm{Cov}(X_i,\, X_j) &\equiv \langle (x_i - \langle x_i \rangle)(x_j - \langle x_j \rangle) \rangle \\ &= \int \cdots \int (x_i - \langle x_i \rangle)(x_j - \langle x_j \rangle)\, P(x_1, \dots, x_n)\, dx_1 \dots dx_n \end{aligned} \tag{55}$$

with

$$\langle x_i \rangle = \int \cdots \int x_i\, P(x_1, \dots, x_n)\, dx_1 \dots dx_n$$

Statistics and blocking

If we consider the above covariance as a matrix $C_{ij} = \mathrm{Cov}(X_i,\, X_j)$, then the diagonal elements are just the familiar variances, $C_{ii} = \mathrm{Cov}(X_i,\, X_i) = \mathrm{Var}(X_i)$. It turns out that all the off-diagonal elements are zero if the stochastic variables are uncorrelated. This is easy to show, keeping in mind the linearity of the expectation value. Consider the stochastic variables X_i and X_j, $(i \neq j)$:

$$\begin{aligned}
\mathrm{Cov}(X_i,\, X_j) &= \langle (x_i - \langle x_i\rangle)(x_j - \langle x_j\rangle)\rangle \\
&= \langle x_i x_j - x_i\langle x_j\rangle - \langle x_i\rangle x_j + \langle x_i\rangle\langle x_j\rangle\rangle \\
&= \langle x_i x_j\rangle - \langle x_i\langle x_j\rangle\rangle - \langle\langle x_i\rangle x_j\rangle + \langle\langle x_i\rangle\langle x_j\rangle\rangle \\
&= \langle x_i x_j\rangle - \langle x_i\rangle\langle x_j\rangle - \langle x_i\rangle\langle x_j\rangle + \langle x_i\rangle\langle x_j\rangle \\
&= \langle x_i x_j\rangle - \langle x_i\rangle\langle x_j\rangle
\end{aligned}$$

Statistics and blocking

If X_i and X_j are independent, we get $\langle x_i x_j\rangle = \langle x_i\rangle\langle x_j\rangle$, resulting in $\mathrm{Cov}(X_i, X_j) = 0 \ \ (i \neq j)$.
Also useful for us is the covariance of linear combinations of stochastic variables. Let $\{X_i\}$ and $\{Y_i\}$ be two sets of stochastic variables. Let also $\{a_i\}$ and $\{b_i\}$ be two sets of scalars. Consider the linear combination:

$$U = \sum_i a_i X_i \qquad V = \sum_j b_j Y_j$$

By the linearity of the expectation value

$$\mathrm{Cov}(U, V) = \sum_{i,j} a_i b_j \mathrm{Cov}(X_i, Y_j)$$

Statistics and blocking

Now, since the variance is just $\mathrm{Var}(X_i) = \mathrm{Cov}(X_i, X_i)$, we get the variance of the linear combination $U = \sum_i a_i X_i$:

$$\mathrm{Var}(U) = \sum_{i,j} a_i a_j \mathrm{Cov}(X_i, X_j) \tag{56}$$

And in the special case when the stochastic variables are uncorrelated, the off-diagonal elements of the covariance are as we know zero, resulting in:

$$\mathrm{Var}(U) = \sum_i a_i^2 \mathrm{Cov}(X_i, X_i) = \sum_i a_i^2 \mathrm{Var}(X_i)$$

$$\mathrm{Var}(\sum_i a_i X_i) = \sum_i a_i^2 \mathrm{Var}(X_i)$$

which will become very useful in our study of the error in the mean value of a set of measurements.

Statistics and blocking

A *stochastic process* is a process that produces sequentially a chain of values:

$$\{x_1, x_2, \dots x_k, \dots\}.$$

We will call these values our *measurements* and the entire set as our measured *sample*. The action of measuring all the elements of a sample we will call a stochastic *experiment* (since, operationally, they are often associated with results of empirical observation of some physical or mathematical phenomena; precisely an experiment). We assume that these values are distributed according to some PDF $p_X(x)$, where X is just the formal symbol for the stochastic variable whose PDF is $p_X(x)$. Instead of trying to determine the full distribution p we are often only interested in finding the few lowest moments, like the mean μ_X and the variance σ_X.

Statistics and blocking

In practical situations a sample is always of finite size. Let that size be n. The expectation value of a sample, the *sample mean*, is then defined as follows:

$$\bar{x}_n \equiv \frac{1}{n}\sum_{k=1}^{n} x_k$$

The *sample variance* is:

$$\mathrm{Var}(x) \equiv \frac{1}{n}\sum_{k=1}^{n}(x_k - \bar{x}_n)^2$$

its square root being the *standard deviation of the sample*. The *sample covariance* is:

$$\mathrm{Cov}(x) \equiv \frac{1}{n}\sum_{kl}(x_k - \bar{x}_n)(x_l - \bar{x}_n)$$

Statistics and blocking

Note that the sample variance is the sample covariance without the cross terms. In a similar manner as the covariance in eq. (55) is a measure of the correlation between two stochastic variables, the above defined sample covariance is a measure of the sequential correlation between succeeding measurements of a sample.

These quantities, being known experimental values, differ significantly from and must not be confused with the similarly named quantities for stochastic variables, mean μ_X, variance $\mathrm{Var}(X)$ and covariance $\mathrm{Cov}(X, Y)$.

Statistics and blocking

The law of large numbers states that as the size of our sample grows to infinity, the sample mean approaches the true mean μ_X of the chosen PDF:

$$\lim_{n\to\infty} \bar{x}_n = \mu_X$$

The sample mean $\bar{x}_n$ works therefore as an estimate of the true mean μ_X.

What we need to find out is how good an approximation $\bar{x}_n$ is to μ_X. In any stochastic measurement, an estimated mean is of no use to us without a measure of its error. A quantity that tells us how well we can reproduce it in another experiment. We are therefore interested in the PDF of the sample mean itself. Its standard deviation will be a measure of the spread of sample means, and we will simply call it the *error* of the sample mean, or just sample error, and denote it by err_X. In practice, we will only be able to produce an *estimate* of the sample error since the exact value would require the knowledge of the true PDFs behind, which we usually do not have.

Statistics and blocking

The straight forward brute force way of estimating the sample error is simply by producing a number of samples, and treating the mean of each as a measurement. The standard deviation of these means will then be an estimate of the original sample error. If we are unable to produce more than one sample, we can split it up sequentially into smaller ones, treating each in the same way as above. This procedure is known as *blocking* and will be given more attention shortly. At this point it is worth while exploring more indirect methods of estimation that will help us understand some important underlying principles of correlational effects.

Statistics and blocking

Let us first take a look at what happens to the sample error as the size of the sample grows. In a sample, each of the measurements x_i can be associated with its own stochastic variable X_i. The stochastic variable $\overline{X}_n$ for the sample mean $\bar{x}_n$ is then just a linear combination, already familiar to us:

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

All the coefficients are just equal $1/n$. The PDF of $\overline{X}_n$, denoted by $p_{\overline{X}_n}(x)$ is the desired PDF of the sample means.

Statistics and blocking

The probability density of obtaining a sample mean $\bar{x}_n$ is the product of probabilities of obtaining arbitrary values $x_1, x_2, \ldots, x_n$ with the constraint that the mean of the set $\{x_i\}$ is $\bar{x}_n$:

$$p_{\overline{X}_n}(x) = \int p_X(x_1) \cdots \int p_X(x_n)\, \delta\left(x - \frac{x_1 + x_2 + \cdots + x_n}{n} \right) dx_n \cdots dx_1$$

And in particular we are interested in its variance $\mathrm{Var}(\overline{X}_n)$.

Statistics and blocking

It is generally not possible to express $p_{\overline{X}_n}(x)$ in a closed form given an arbitrary PDF p_X and a number n. But for the limit $n \to \infty$ it is possible to make an approximation. The very important result is called *the central limit theorem*. It tells us that as n goes to infinity, $p_{\overline{X}_n}(x)$ approaches a Gaussian distribution whose mean and variance equal the true mean and variance, μ_X and σ_X^2, respectively:

$$\lim_{n\to\infty} p_{\overline{X}_n}(x) = \left(\frac{n}{2\pi\mathrm{Var}(X)}\right)^{1/2} e^{-\frac{n(x-\bar{x}_n)^2}{2\mathrm{Var}(X)}} \tag{57}$$

Statistics and blocking

The desired variance $\mathrm{Var}(\overline{X}_n)$, i.e. the sample error squared err_X^2, is given by:

$$\mathrm{err}_X^2 = \mathrm{Var}(\overline{X}_n) = \frac{1}{n^2}\sum_{ij}\mathrm{Cov}(X_i, X_j) \tag{58}$$

We see now that in order to calculate the exact error of the sample with the above expression, we would need the true means μ_{X_i} of the stochastic variables X_i. To calculate these requires that we know the true multivariate PDF of all the X_i. But this PDF is unknown to us, we have only got the measurements of one sample. The best we can do is to let the sample itself be an estimate of the PDF of each of the X_i, estimating all properties of X_i through the measurements of the sample.

Statistics and blocking

Our estimate of μ_{X_i} is then the sample mean $\bar{x}$ itself, in accordance with the the central limit theorem:

$$\mu_{X_i} = \langle x_i \rangle \approx \frac{1}{n}\sum_{k=1}^{n} x_k = \bar{x}$$

Using $\bar{x}$ in place of μ_{X_i} we can give an *estimate* of the covariance in eq. (58):

$$\begin{aligned}
\mathrm{Cov}(X_i, X_j) &= \langle (x_i - \langle x_i \rangle)(x_j - \langle x_j \rangle) \rangle \approx \langle (x_i - \bar{x})(x_j - \bar{x}) \rangle \\
&\approx \frac{1}{n}\sum_{l}^{n} \left(\frac{1}{n}\sum_{k}^{n} (x_k - \bar{x}_n)(x_l - \bar{x}_n) \right) = \frac{1}{n}\frac{1}{n}\sum_{kl} (x_k - \bar{x}_n)(x_l - \bar{x}_n) \\
&= \frac{1}{n}\mathrm{Cov}(x)
\end{aligned}$$

Statistics and blocking

By the same procedure we can use the sample variance as an estimate of the variance of any of the stochastic variables X_i:

$$\begin{aligned}\mathrm{Var}(X_i) &= \langle x_i - \langle x_i \rangle\rangle \approx \langle x_i - \bar{x}_n \rangle \\ &\approx \frac{1}{n}\sum_{k=1}^{n}(x_k - \bar{x}_n) \\ &= \mathrm{Var}(x) \qquad (59)\end{aligned}$$

Now we can calculate an estimate of the error err_X of the sample mean $\bar{x}_n$:

$$\begin{aligned}\mathrm{err}_X^2 &= \frac{1}{n^2}\sum_{ij}\mathrm{Cov}(X_i, X_j) \\ &\approx \frac{1}{n^2}\sum_{ij}\frac{1}{n}\mathrm{Cov}(x) = \frac{1}{n^2}n^2\frac{1}{n}\mathrm{Cov}(x) \\ &= \frac{1}{n}\mathrm{Cov}(x) \qquad (60)\end{aligned}$$

which is nothing but the sample covariance divided by the number of measurements in the sample.

Statistics and blocking

In the special case that the measurements of the sample are uncorrelated (equivalently the stochastic variables X_i are uncorrelated) we have that the off-diagonal elements of the covariance are zero. This gives the following estimate of the sample error:

$$\begin{aligned}
\mathrm{err}_X^2 &= \frac{1}{n^2}\sum_{ij}\mathrm{Cov}(X_i, X_j) = \frac{1}{n^2}\sum_i \mathrm{Var}(X_i) \\
&\approx \frac{1}{n^2}\sum_i \mathrm{Var}(x) \\
&= \frac{1}{n}\mathrm{Var}(x) \qquad (61)
\end{aligned}$$

where in the second step we have used eq. (59). The error of the sample is then just its standard deviation divided by the square root of the number of measurements the sample contains. This is a very useful formula which is easy to compute. It acts as a first approximation to the error, but in numerical experiments, we cannot overlook the always present correlations.

Statistics and blocking

For computational purposes one usually splits up the estimate of err_X^2, given by eq. (60), into two parts:

$$
\begin{aligned}
\mathrm{err}_X^2 &= \frac{1}{n}\mathrm{Var}(x) + \frac{1}{n}(\mathrm{Cov}(x) - \mathrm{Var}(x)) \\
&= \frac{1}{n^2}\sum_{k=1}^{n}(x_k - \bar{x}_n)^2 + \frac{2}{n^2}\sum_{k<l}(x_k - \bar{x}_n)(x_l - \bar{x}_n) \qquad (62)
\end{aligned}
$$

The first term is the same as the error in the uncorrelated case, eq. (61). This means that the second term accounts for the error correction due to correlation between the measurements. For uncorrelated measurements this second term is zero.

Statistics and blocking

Computationally the uncorrelated first term is much easier to treat efficiently than the second.

$$\mathrm{Var}(x) = \frac{1}{n}\sum_{k=1}^{n}(x_k - \bar{x}_n)^2 = \left(\frac{1}{n}\sum_{k=1}^{n} x_k^2\right) - \bar{x}_n^2$$

We just accumulate separately the values x^2 and x for every measurement x we receive. The correlation term, though, has to be calculated at the end of the experiment since we need all the measurements to calculate the cross terms. Therefore, all measurements have to be stored throughout the experiment.

Statistics and blocking

Let us analyze the problem by splitting up the correlation term into partial sums of the form:

$$f_d = \frac{1}{n-d}\sum_{k=1}^{n-d}(x_k - \bar{x}_n)(x_{k+d} - \bar{x}_n)$$

The correlation term of the error can now be rewritten in terms of f_d:

$$\frac{2}{n}\sum_{k<l}(x_k - \bar{x}_n)(x_l - \bar{x}_n) = 2\sum_{d=1}^{n-1} f_d$$

The value of f_d reflects the correlation between measurements separated by the distance d in the sample samples. Notice that for $d = 0$, f is just the sample variance, $\mathrm{Var}(x)$. If we divide f_d by $\mathrm{Var}(x)$, we arrive at the so called *autocorrelation function*:

$$\kappa_d = \frac{f_d}{\mathrm{Var}(x)}$$

which gives us a useful measure of the correlation pair correlation starting always at 1 for $d = 0$.

Statistics and blocking

The sample error (see eq. (62)) can now be written in terms of the autocorrelation function:

$$
\begin{aligned}
\mathrm{err}_X^2 &= \frac{1}{n}\mathrm{Var}(x) + \frac{2}{n}\cdot\mathrm{Var}(x)\sum_{d=1}^{n-1}\frac{f_d}{\mathrm{Var}(x)} \\
&= \left(1 + 2\sum_{d=1}^{n-1}\kappa_d\right)\frac{1}{n}\mathrm{Var}(x) \\
&= \frac{\tau}{n}\cdot\mathrm{Var}(x) \qquad (63)
\end{aligned}
$$

and we see that err_X can be expressed in terms the uncorrelated sample variance times a correction factor τ which accounts for the correlation between measurements. We call this correction factor the *autocorrelation time*:

$$
\tau = 1 + 2\sum_{d=1}^{n-1}\kappa_d \qquad (64)
$$

Statistics and blocking

For a correlation free experiment, τ equals 1. From the point of view of eq. (63) we can interpret a sequential correlation as an effective reduction of the number of measurements by a factor τ. The effective number of measurements becomes:

$$n_{\mathrm{eff}} = \frac{n}{\tau}$$

To neglect the autocorrelation time τ will always cause our simple uncorrelated estimate of $\mathrm{err}_X^2 \approx \mathrm{Var}(x)/n$ to be less than the true sample error. The estimate of the error will be too “good”. On the other hand, the calculation of the full autocorrelation time poses an efficiency problem if the set of measurements is very large.

Can we understand this? Time Auto-correlation Function

The so-called time-displacement autocorrelation $\phi(t)$ for a quantity $\mathcal{M}$ is given by

$$\phi(t) = \int dt' \left[\mathcal{M}(t') - \langle\mathcal{M}\rangle\right] \left[\mathcal{M}(t'+t) - \langle\mathcal{M}\rangle\right],$$

which can be rewritten as

$$\phi(t) = \int dt' \left[\mathcal{M}(t')\mathcal{M}(t'+t) - \langle\mathcal{M}\rangle^2\right],$$

where $\langle\mathcal{M}\rangle$ is the average value and $\mathcal{M}(t)$ its instantaneous value. We can discretize this function as follows, where we used our set of computed values $\mathcal{M}(t)$ for a set of discretized times (our Monte Carlo cycles corresponding to moving all electrons?)

$$\phi(t) = \frac{1}{t_{\max} - t}\sum_{t'=0}^{t_{\max}-t}\mathcal{M}(t')\mathcal{M}(t'+t) - \frac{1}{t_{\max} - t}\sum_{t'=0}^{t_{\max}-t}\mathcal{M}(t')\times\frac{1}{t_{\max} - t}\sum_{t'=0}^{t_{\max}-t}\mathcal{M}(t'+t).$$

Time Auto-correlation Function

One should be careful with times close to $t_{\max}$, the upper limit of the sums becomes small and we end up integrating over a rather small time interval. This means that the statistical error in $\phi(t)$ due to the random nature of the fluctuations in $\mathcal{M}(t)$ can become large.
One should therefore choose $t \ll t_{\max}$.
Note that the variable $\mathcal{M}$ can be any expectation values of interest.

The time-correlation function gives a measure of the correlation between the various values of the variable at a time t' and a time $t' + t$. If we multiply the values of $\mathcal{M}$ at these two different times, we will get a positive contribution if they are fluctuating in the same direction, or a negative value if they fluctuate in the opposite direction. If we then integrate over time, or use the discretized version of, the time correlation function $\phi(t)$ should take a non-zero value if the fluctuations are correlated, else it should gradually go to zero. For times a long way apart the different values of $\mathcal{M}$ are most likely uncorrelated and $\phi(t)$ should be zero.

Time Auto-correlation Function

We can derive the correlation time by observing that our Metropolis algorithm is based on a random walk in the space of all possible spin configurations. Our probability distribution function $\hat{\mathbf{w}}(t)$ after a given number of time steps t could be written as

$$\hat{\mathbf{w}}(t) = \hat{\mathbf{W}}^{\mathbf{t}}\hat{\mathbf{w}}(0),$$

with $\hat{\mathbf{w}}(0)$ the distribution at $t = 0$ and $\hat{\mathbf{W}}$ representing the transition probability matrix. We can always expand $\hat{\mathbf{w}}(0)$ in terms of the right eigenvectors of $\hat{\mathbf{v}}$ of $\hat{\mathbf{W}}$ as

$$\hat{\mathbf{w}}(0) = \sum_i \alpha_i \hat{\mathbf{v}}_i,$$

resulting in

$$\hat{\mathbf{w}}(t) = \hat{\mathbf{W}}^t\hat{\mathbf{w}}(0) = \hat{\mathbf{W}}^t \sum_i \alpha_i \hat{\mathbf{v}}_i = \sum_i \lambda_i^t \alpha_i \hat{\mathbf{v}}_i,$$

with λ_i the i^{th} eigenvalue corresponding to the eigenvector $\hat{\mathbf{v}}_i$.

Time Auto-correlation Function

If we assume that λ_0 is the largest eigenvector we see that in the limit $t \rightarrow \infty$, $\hat{\mathbf{w}}(t)$ becomes proportional to the corresponding eigenvector $\hat{\mathbf{v}}_0$. This is our steady state or final distribution.
We can relate this property to an observable like the mean energy. With the probabilty $\hat{\mathbf{w}}(t)$ (which in our case is the squared trial wave function) we can write the expectation values as

$$\langle \mathcal{M}(t) \rangle = \sum_{\mu} \hat{\mathbf{w}}(t)_{\mu} \mathcal{M}_{\mu},$$

or as the scalar of a vector product

$$\langle \mathcal{M}(t) \rangle = \hat{\mathbf{w}}(t) \mathbf{m},$$

with **m** being the vector whose elements are the values of $\mathcal{M}_{\mu}$ in its various microstates μ.

Time Auto-correlation Function

We rewrite this relation as

$$\langle \mathcal{M}(t) \rangle = \hat{\mathbf{w}}(t)\mathbf{m} = \sum_i \lambda_i^t \alpha_i \hat{\mathbf{v}}_i \mathbf{m}_i.$$

If we define $m_i = \hat{\mathbf{v}}_i \mathbf{m}_i$ as the expectation value of $\mathcal{M}$ in the i^{th} eigenstate we can rewrite the last equation as

$$\langle \mathcal{M}(t) \rangle = \sum_i \lambda_i^t \alpha_i m_i.$$

Since we have that in the limit $t \to \infty$ the mean value is dominated by the the largest eigenvalue λ_0, we can rewrite the last equation as

$$\langle \mathcal{M}(t) \rangle = \langle \mathcal{M}(\infty) \rangle + \sum_{i \neq 0} \lambda_i^t \alpha_i m_i.$$

We define the quantity

$$\tau_i = -\frac{1}{log \lambda_i},$$

and rewrite the last expectation value as

$$\langle \mathcal{M}(t) \rangle = \langle \mathcal{M}(\infty) \rangle + \sum_{i \neq 0} \alpha_i m_i e^{-t/\tau_i}.$$

Time Auto-correlation Function

The quantities τ_i are the correlation times for the system. They control also the auto-correlation function discussed above. The longest correlation time is obviously given by the second largest eigenvalue τ_1, which normally defines the correlation time discussed above. For large times, this is the only correlation time that survives. If higher eigenvalues of the transition matrix are well separated from λ_1 and we simulate long enough, τ_1 may well define the correlation time. In other cases we may not be able to extract a reliable result for τ_1. Coming back to the time correlation function $\phi(t)$ we can present a more general definition in terms of the mean magnetizations $\langle \mathcal{M}(t) \rangle$. Recalling that the mean value is equal to $\langle \mathcal{M}(\infty) \rangle$ we arrive at the expectation values

$$\phi(t) = \langle \mathcal{M}(0) - \mathcal{M}(\infty) \rangle \langle \mathcal{M}(t) - \mathcal{M}(\infty) \rangle,$$

resulting in

$$\phi(t) = \sum_{i,j \neq 0} m_i \alpha_i m_j \alpha_j e^{-t/\tau_i},$$

which is appropriate for all times.

Correlation Time

If the correlation function decays exponentially

$$\phi(t) \sim \exp\left(-t/\tau\right)$$

then the exponential correlation time can be computed as the average

$$\tau_{\text{exp}} = -\langle \frac{t}{log|\frac{\phi(t)}{\phi(0)}|} \rangle.$$

If the decay is exponential, then

$$\int_0^\infty dt\phi(t) = \int_0^\infty dt\phi(0)\exp\left(-t/\tau\right) = \tau\phi(0),$$

which suggests another measure of correlation

$$\tau_{\text{int}} = \sum_k \frac{\phi(k)}{\phi(0)},$$

called the integrated correlation time.

What is blocking?

Blocking

- Say that we have a set of samples from a Monte Carlo experiment
- Assuming (wrongly) that our samples are uncorrelated our best estimate of the standard deviation of the mean $\langle \mathcal{M} \rangle$ is given by

$$\sigma = \sqrt{\frac{1}{n}\left(\langle \mathcal{M}^2 \rangle - \langle \mathcal{M} \rangle^2\right)}$$

- If the samples are correlated we can rewrite our results to show that

$$\sigma = \sqrt{\frac{1 + 2\tau/\Delta t}{n}\left(\langle \mathcal{M}^2 \rangle - \langle \mathcal{M} \rangle^2\right)}$$

where τ is the correlation time (the time between a sample and the next uncorrelated sample) and Δt is time between each sample

What is blocking?

Blocking

- If $\Delta t \gg \tau$ our first estimate of σ still holds
- Much more common that $\Delta t < \tau$
- In the method of data blocking we divide the sequence of samples into blocks
- We then take the mean $\langle \mathcal{M}_i \rangle$ of block $i = 1 \ldots n_{blocks}$ to calculate the total mean and variance
- The size of each block must be so large that sample j of block i is not correlated with sample j of block $i + 1$
- The correlation time τ would be a good choice

What is blocking?

Blocking

- Problem: We don't know τ or it is too expensive to compute
- Solution: Make a plot of std. dev. as a function of block size
- The estimate of std. dev. of correlated data is too low $\rightarrow$ the error will increase with increasing block size until the blocks are uncorrelated, where we reach a plateau
- When the std. dev. stops increasing the blocks are uncorrelated

Implementation

Main ideas

- Do a parallel Monte Carlo simulation, storing all samples to files (one per process)
- Do the statistical analysis on these files, independently of your Monte Carlo program
- Read the files into an array
- Loop over various block sizes
- For each block size n_b, loop over the array in steps of n_b taking the mean of elements $in_b, \dots, (i+1)n_b$
- Take the mean and variance of the resulting array
- Write the results for each block size to file for later analysis

Implementation

Example

- The files vmc_para.cpp and vmc_blocking.cpp contain a parallel VMC simulator and a program for doing blocking on the samples from the resulting set of files
- Will go through the parts related to blocking

Implementation

Parallel file output

- The total number of samples from all processes may get very large
- Hence, storing all samples on the master node is not a scalable solution
- Instead we store the samples from each process in separate files
- Must make sure these files have different names

String handling

```
ostringstream ost;
ost << "blocks_rank" << my_rank << ".dat";
blockofile.open(ost.str().c_str(), ios::out | ios::
    binary);
```

Implementation

Parallel file output

- Having separated the filenames it’s just a matter of taking the samples and store them to file
- Note that there is no need for communication between the processes in this procedure

File dumping

```
all_energies  = new double[number_cycles+1];
mc_sampling(max_variations, number_cycles,
    cumulative_e, cumulative_e2,
               all_energies);

blockofile.write((char*)(all_energies+1),
                    number_cycles*sizeof(double));
blockofile.close();
```

Implementation

Reading the files

- Reading the files is only about mirroring the output
- To make life easier for ourselves we find the filesize, and hence the number of samples by using the C function `stat`

File loading

```
struct stat result;
if(stat("blocks_rank0.dat", &result) == 0){
  local_n = result.st_size/sizeof(double);
  n = local_n*n_procs;
}

double* mc_results = new double[n];
for(int i=0; i<n_procs; i++){
  ostringstream ost;
  ost << "blocks_rank" << i << ".dat";
  ifstream infile;
  infile.open(ost.str().c_str(), ios::in | ios::binary);
  infile.read((char*)&(mc_results[i*local_n]),result.st_size);
  infile.close();
}
```

Implementation

Blocking

- Loop over block sizes $in_b, \ldots, (i+1)n_b$

Loop over block sizes

```
for(int i=0; i<n_block_samples; i++){
  block_size = min_block_size+i*block_step_length;
  blocking(mc_results, n, block_size, res);
  mean  = res[0];
  sigma = res[1];
  outfile << block_size << "\t" << mean << "\t"
          << sqrt(sigma/((n/block_size)-1.0))
          << endl;
}
```

Implementation

Blocking

- The blocking itself is now just a matter of finding the number of blocks (note the integer division) and taking the mean of each block
- Note the pointer aritmetic: Adding a number *i* to an array pointer moves the pointer to element *i* in the array

Blocking function

```
void blocking(double *vals, int n_vals, int
    block_size, double *res){
  int n_blocks = n_vals/block_size;
  double* block_vals = new double[n_blocks];
  for(int i=0; i<n_blocks; i++)
    block_vals[i] = mean(vals+i*block_size,
        block_size);
  meanvar(block_vals, n_blocks, res);
}
```

Topics for Week 8, February 22-26

Blocking, Conjugate gradient method and start discussion of Slater determinants

- Repetition from last week
- Blocking and conjugate gradient method
- Start discussion of Slater determinant (part 2 of Project 1)

Project work this week: Finalize 1b and start with 1c and 1d.

Conjugate gradient (CG) method

The success of the CG method for finding solutions of non-linear problems is based on the theory for of conjugate gradients for linear systems of equations. It belongs to the class of iterative methods for solving problems from linear algebra of the type

$$\hat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{b}}.$$

In the iterative process we end up with a problem like

$$\hat{\mathbf{r}} = \hat{\mathbf{b}} - \hat{\mathbf{A}}\hat{\mathbf{x}},$$

where $\hat{\mathbf{r}}$ is the so-called residual or error in the iterative process.

Conjugate gradient method

The residual is zero when we reach the minimum of the quadratic equation

$$P(\hat{\mathbf{x}}) = \frac{1}{2}\hat{\mathbf{x}}^T\hat{\mathbf{A}}\hat{\mathbf{x}} - \hat{\mathbf{x}}^T\hat{\mathbf{b}},$$

with the constraint that the matrix $\hat{\mathbf{A}}$ is positive definite and symmetric. If we search for a minimum of the quantum mechanical variance, then the matrix $\hat{\mathbf{A}}$, which is called the Hessian, is given by the second-derivative of the variance. This quantity is always positive definite. If we vary the energy, the Hessian may not always be positive definite.

Conjugate gradient method

In the CG method we define so-called conjugate directions and two vectors $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ are said to be conjugate if

$$\hat{\mathbf{s}}^T\hat{\mathbf{A}}\hat{\mathbf{t}} = 0.$$

The philosophy of the CG method is to perform searches in various conjugate directions of our vectors $\hat{\mathbf{x}}_i$ obeying the above criterion, namely

$$\hat{\mathbf{x}}_i^T\hat{\mathbf{A}}\hat{\mathbf{x}}_j = 0.$$

Two vectors are conjugate if they are orthogonal with respect to this inner product.

Being conjugate is a symmetric relation: if $\hat{\mathbf{s}}$ is conjugate to $\hat{\mathbf{t}}$, then $\hat{\mathbf{t}}$ is conjugate to $\hat{\mathbf{s}}$.

Conjugate gradient method

An example is given by the eigenvectors of the matrix

$$\hat{\mathbf{v}}_i^T \hat{\mathbf{A}} \hat{\mathbf{v}}_j = \lambda \hat{\mathbf{v}}_i^T \hat{\mathbf{v}}_j,$$

which is zero unless $i = j$.

Conjugate gradient method

Assume now that we have a symmetric positive-definite matrix $\hat{\mathbf{A}}$ of size $n \times n$. At each iteration $i+1$ we obtain the conjugate direction of a vector

$$\hat{\mathbf{x}}_{i+1} = \hat{\mathbf{x}}_i + \alpha_i \hat{\mathbf{p}}_i.$$

We assume that $\hat{\mathbf{p}}_i$ is a sequence of n mutually conjugate directions. Then the $\hat{\mathbf{p}}_i$ form a basis of R^n and we can expand the solution $\hat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{b}}$ in this basis, namely

$$\hat{\mathbf{x}} = \sum_{i=1}^{n} \alpha_i \hat{\mathbf{p}}_i.$$

Conjugate gradient method

The coefficients are given by

$$\mathbf{Ax} = \sum_{i=1}^{n} \alpha_i \mathbf{Ap}_i = \mathbf{b}.$$

Multiplying with $\hat{\mathbf{p}}_k^T$ from the left gives

$$\hat{\mathbf{p}}_k^T \hat{\mathbf{A}} \hat{\mathbf{x}} = \sum_{i=1}^{n} \alpha_i \hat{\mathbf{p}}_k^T \hat{\mathbf{A}} \hat{\mathbf{p}}_i = \hat{\mathbf{p}}_k^T \hat{\mathbf{b}},$$

and we can define the coefficients α_k as

$$\alpha_k = \frac{\hat{\mathbf{p}}_k^T \hat{\mathbf{b}}}{\hat{\mathbf{p}}_k^T \hat{\mathbf{A}} \hat{\mathbf{p}}_k}$$

Conjugate gradient method and iterations

If we choose the conjugate vectors $\hat{\mathbf{p}}_k$ carefully, then we may not need all of them to obtain a good approximation to the solution $\hat{\mathbf{x}}$. So, we want to regard the conjugate gradient method as an iterative method. This also allows us to solve systems where n is so large that the direct method would take too much time.
We denote the initial guess for $\hat{\mathbf{x}}$ as $\hat{\mathbf{x}}_0$. We can assume without loss of generality that

$$\hat{\mathbf{x}}_0 = 0,$$

or consider the system

$$\hat{\mathbf{A}}\hat{\mathbf{z}} = \hat{\mathbf{b}} - \hat{\mathbf{A}}\hat{\mathbf{x}}_0,$$

instead.

Conjugate gradient method

Important, one can show that the solution $\hat{\mathbf{x}}$ is also the unique minimizer of the quadratic form

$$f(\hat{\mathbf{x}}) = \frac{1}{2}\hat{\mathbf{x}}^T\hat{\mathbf{A}}\hat{\mathbf{x}} - \hat{\mathbf{x}}^T\hat{\mathbf{x}}, \quad \hat{\mathbf{x}} \in \mathbf{R}^n.$$

This suggests taking the first basis vector $\hat{\mathbf{p}}_1$ to be the gradient of f at $\hat{\mathbf{x}} = \hat{\mathbf{x}}_0$, which equals

$$\hat{\mathbf{A}}\hat{\mathbf{x}}_0 - \hat{\mathbf{b}},$$

and $\hat{\mathbf{x}}_0 = 0$ it is equal $-\hat{\mathbf{b}}$. The other vectors in the basis will be conjugate to the gradient, hence the name conjugate gradient method.

Conjugate gradient method

Let $\hat{\mathbf{r}}_k$ be the residual at the k-th step:

$$\hat{\mathbf{r}}_k = \hat{\mathbf{b}} - \hat{\mathbf{A}}\hat{\mathbf{x}}_k.$$

Note that $\hat{\mathbf{r}}_k$ is the negative gradient of f at $\hat{\mathbf{x}} = \hat{\mathbf{x}}_k$, so the gradient descent method would be to move in the direction $\hat{\mathbf{r}}_k$. Here, we insist that the directions $\hat{\mathbf{p}}_k$ are conjugate to each other, so we take the direction closest to the gradient $\hat{\mathbf{r}}_k$ under the conjugacy constraint. This gives the following expression

$$\hat{\mathbf{p}}_{k+1} = \hat{\mathbf{r}}_k - \frac{\hat{\mathbf{p}}_k^T \hat{\mathbf{A}} \hat{\mathbf{r}}_k}{\hat{\mathbf{p}}_k^T \hat{\mathbf{A}} \hat{\mathbf{p}}_k} \hat{\mathbf{p}}_k.$$

Conjugate gradient method

We can also compute the residual iteratively as

$$\hat{\mathbf{r}}_{k+1} = \hat{\mathbf{b}} - \hat{\mathbf{A}}\hat{\mathbf{x}}_{k+1},$$

which equals

$$\hat{\mathbf{b}} - \hat{\mathbf{A}}(\hat{\mathbf{x}}_k + \alpha_k \hat{\mathbf{p}}_k),$$

or

$$(\hat{\mathbf{b}} - \hat{\mathbf{A}}\hat{\mathbf{x}}_k) - \alpha_k \hat{\mathbf{A}}\hat{\mathbf{p}}_k,$$

which gives

$$\hat{\mathbf{r}}_{k+1} = \hat{\mathbf{r}}_k - \hat{\mathbf{A}}\hat{\mathbf{p}}_k,$$

Conjugate gradient method, our case

If we consider finding the minimum of a function f using Newton's method, that is search for a zero of the gradient of a function. Near a point x_i we have to second order

$$f(\hat{\mathbf{x}}) = f(\hat{\mathbf{x}}_i) + (\hat{\mathbf{x}} - \hat{\mathbf{x}}_i)\nabla f(\hat{\mathbf{x}}_i)\frac{1}{2}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_i)\hat{\mathbf{A}}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_i)$$

giving

$$\nabla f(\hat{\mathbf{x}}) = \nabla f(\hat{\mathbf{x}}_i) + \hat{\mathbf{A}}(\hat{\mathbf{x}} - \hat{\mathbf{x}}_i).$$

In Newton's method we set $\nabla f = 0$ and we can thus compute the next iteration point (here the exact result)

$$\hat{\mathbf{x}} - \hat{\mathbf{x}}_i = \hat{\mathbf{A}}^{-1}\nabla f(\hat{\mathbf{x}}_i).$$

Subtracting this equation from that of $\hat{\mathbf{x}}_{i+1}$ we have

$$\hat{\mathbf{x}}_{i+1} - \hat{\mathbf{x}}_i = \hat{\mathbf{A}}^{-1}(\nabla f(\hat{\mathbf{x}}_{i+1}) - \nabla f(\hat{\mathbf{x}}_i)).$$

Codes from numerical recipes

The codes are taken from chapter 10.7 of Numerical recipes. We use the functions *dfpmin* and *lnsrch*. You can load down the package of programs from the webpage of the course, see under project 1. The package is called *NRcgm*107*.tar.gz* and contains the files *dfmin.c*, *lnsrch.c*, *nrutil.c* and *nrutil.h*. These codes are written in C.

```
void dfpmin(double p[], int n, double gtol, int *iter, double *fret,
double(*func)(double []), void (*dfunc)(double [], double []))
```

What you have to provide

The input to *dfpmin*

```
void dfpmin(double p[], int n, double gtol, int *iter, double *fret,
double(*func)(double []), void (*dfunc)(double [], double []))
```

is

- The starting vector *p* of length *n*
- The function *func* on which minimization is done
- The function *dfunc* where the gradient i calculated
- The convergence requirement for zeroing the gradient *gtol*.

It returns in *p* the location of the minimum, the number of iterations and the minimum value of the function under study *fret*.

Simple example and demonstration

For the harmonic oscillator in one-dimension with a trial wave function and probability

$$\psi_T(x) = e^{-\alpha^2 x^2} \qquad , P_T(x)dx = \frac{e^{-2\alpha^2 x^2} dx}{\int dx e^{-2\alpha^2 x^2}}$$

with α as the variational parameter. We have the following local energy

$$E_L[\alpha] = \alpha^2 + x^2 \left(\frac{1}{2} - 2\alpha^2 \right),$$

which results in the expectation value

$$\langle E_L[\alpha] \rangle = \frac{1}{2}\alpha^2 + \frac{1}{8\alpha^2}$$

Simple example and demonstration

The derivative of the energy with respect to α gives

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = \alpha - \frac{1}{4\alpha^3}$$

and a second derivative which is always positive (meaning that we find a minimum)

$$\frac{d^2\langle E_L[\alpha]\rangle}{d\alpha^2} = 1 + \frac{3}{4\alpha^4}$$

The condition

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = 0,$$

gives the optimal $\alpha = 1/\sqrt{2}$.

Simple example and demonstration

In general we end up computing the expectation value of the energy in terms of some parameters $\alpha = \{\alpha_0, \alpha_1, \dots, \alpha_n)$ and we search for a minimum in parameter space. This leads to an energy minimization problem.
The elements of the gradient are (*Ei* is the first derivative wrt to the variational parameter α_i)

$$\begin{aligned} \bar{E}_i &= \left\langle \frac{\psi_i}{\psi} E_{\mathrm{L}} + \frac{H\psi_i}{\psi} - 2\bar{E}\frac{\psi_i}{\psi} \right\rangle && (65) \\ &= 2\left\langle \frac{\psi_i}{\psi}(E_{\mathrm{L}} - \bar{E}) \right\rangle \quad \text{(by Hermiticity)}. && (66) \end{aligned}$$

For our simple model we get the same expression for the first derivative (check it!).

Simple example and demonstration

Taking the second derivative the Hessian is

$$\begin{aligned}\bar{E}_{ij} &= 2\Bigg[\left\langle\left(\frac{\psi_{ij}}{\psi}+\frac{\psi_i\psi_j}{\psi^2}\right)(E_{\mathrm{L}}-\bar{E})\right\rangle \\ &\quad -\left\langle\frac{\psi_i}{\psi}\right\rangle\bar{E}_j-\left\langle\frac{\psi_j}{\psi}\right\rangle\bar{E}_i+\left\langle\frac{\psi_i}{\psi}E_{\mathrm{L},j}\right\rangle\Bigg]. \end{aligned} \tag{67}$$

Note that our conjugate gradient approach does need the Hessian! Check again that the simple models gives the same second derivative with the above expression.

Simple example and demonstration

We can also minimize the variance. In our simple model the variance is

$$\sigma^2[\alpha] = \frac{1}{2}\alpha^4 - \frac{1}{4} + \frac{1}{32\alpha^4},$$

with first derivative

$$\frac{d\sigma^2[\alpha]}{d\alpha} = 2\alpha^3 - \frac{1}{8\alpha^5}$$

and a second derivative which is always positive

$$\frac{d^2\sigma^2[\alpha]}{d\alpha^2} = 6\alpha^2 + \frac{5}{8\alpha^6}$$

Conjugate gradient method, our case

In Newton's method we set $\nabla f = 0$ and we can thus compute the next iteration point (here the exact result)

$$\hat{\mathbf{x}} - \hat{\mathbf{x}}_i = \hat{\mathbf{A}}^{-1}\nabla f(\hat{\mathbf{x}}_i).$$

Subtracting this equation from that of $\hat{\mathbf{x}}_{i+1}$ we have

$$\hat{\mathbf{x}}_{i+1} - \hat{\mathbf{x}}_i = \hat{\mathbf{A}}^{-1}(\nabla f(\hat{\mathbf{x}}_{i+1}) - \nabla f(\hat{\mathbf{x}}_i)).$$

Simple example and demonstration

In our case f can be either the energy or the variance. If we choose the energy then we have

$$\hat{\alpha}_{i+1}-\hat{\alpha}_i=\hat{\mathbf{A}}^{-1}(\nabla E(\hat{\alpha}_{i+1})-\nabla E(\hat{\alpha}_i)).$$

In the simple model gradient and the Hessian $\hat{\mathbf{A}}$ are

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha}=\alpha-\frac{1}{4\alpha^3}$$

and a second derivative which is always positive (meaning that we find a minimum)

$$\hat{\mathbf{A}}=\frac{d^2\langle E_L[\alpha]\rangle}{d\alpha^2}=1+\frac{3}{4\alpha^4}$$

Simple example and demonstration

We get then

$$\alpha_{i+1} = \frac{4}{3}\alpha_i - \frac{\alpha_i^4}{3\alpha_{i+1}^3},$$

which can be rewritten as

$$\alpha_{i+1}^4 - \frac{4}{3}\alpha_i\alpha_{i+1}^4 + \frac{1}{3}\alpha_i^4.$$

Our code does however not need the value of the Hessian since it produces an estimate of the Hessian.

Simple example and code (model.cpp on webpage)

```
#include "nrutil.h"
using namespace std;
//     Here we define various functions called by the main program

double E_function(double *x);
void   dE_function(double *x, double *g);
void   dfpmin(double p[], int n, double gtol, int *iter, double *fret,
    double(*func)(double []), void (*dfunc)(double [], double []));
//   Main function begins here
int main()
{
     int n, iter;
     double gtol, fret;
     double alpha;
     n = 1;
     cout << "Read in guess for alpha" << endl;
     cin >> alpha;
```

Simple example and code (model.cpp on webpage)

```
//   reserve space in memory for vectors containing the variational
//   parameters
     double *p = new double [2];
     gtol = 1.0e-5;
//   now call dfmin and compute the minimum
     p[1] = alpha;
     dfpmin(p, n, gtol, &iter, &fret,&E_function,&dE_function);
     cout << "Value of energy minimum = " << fret << endl;
     cout << "Number of iterations = " << iter << endl;
     cout << "Value of alpha at minimu = " << p[1] << endl;
      delete [] p;
```

Simple example and code (model.cpp on webpage)

```
//  this function defines the Energy function
double E_function(double x[])
{
  double value = x[1]*x[1]*0.5+1.0/(8*x[1]*x[1]);
  return value;
} // end of function to evaluate
```

Simple example and code (model.cpp on webpage)

```
//  this function defines the derivative of the energy
void dE_function(double x[], double g[])
{
  g[1] = x[1]-1.0/(4*x[1]*x[1]*x[1]);
} // end of function to evaluate
```

Using the conjugate gradient method

- Start your program with calling the CGM method (function *dfpmin*).
- This function needs the function for the expectation value of the local energy and the derivative of the local energy. Change the functions *func* and *dfunc* in the codes below.
- Your function *func* is now the Metropolis part with a call to the local energy function. For every call to the function *func* I used 1000 Monte Carlo cycles for the trial wave function
$$\Psi_T(\mathbf{r}_1,\mathbf{r}_2) = e^{-\alpha(r_1+r_2)}$$
- This gave me an expectation value for the energy which is returned by the function *func*.
- When I call the local energy I also compute the first derivative of the expectaction value of the local energy

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = 2\left\langle \frac{\psi_i}{\psi}(E_L[\alpha] - \langle E_L[\alpha]\rangle)\right\rangle.$$

Using the conjugate gradient method

The expectation value for the local energy of the Helium atom with a simple Slater determinant is given by

$$\langle E_L \rangle = \alpha^2 - 2\alpha\left(Z - \frac{5}{16}\right)$$

You should test your numerical derivative with the derivative of the last expression, that is

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = 2\alpha - 2\left(Z - \frac{5}{16}\right).$$

Simple example and code (model.cpp on webpage)

```
#include "nrutil.h"
using namespace std;
//     Here we define various functions called by the main program

double E_function(double *x);
void   dE_function(double *x, double *g);
void   dfpmin(double p[], int n, double gtol, int *iter, double *fret,
    double(*func)(double []), void (*dfunc)(double [], double []));
//   Main function begins here
int main()
{
     int n, iter;
     double gtol, fret;
     double alpha;
     n = 1;
     cout << "Read in guess for alpha" << endl;
     cin >> alpha;
```

Simple example and code (model.cpp on webpage)

```
//   reserve space in memory for vectors containing the variational
//   parameters
     double *p = new double [2];
     gtol = 1.0e-5;
//   now call dfmin and compute the minimum
     p[1] = alpha;
     dfpmin(p, n, gtol, &iter, &fret,&E_function,&dE_function);
     cout << "Value of energy minimum = " << fret << endl;
     cout << "Number of iterations = " << iter << endl;
     cout << "Value of alpha at minimu = " << p[1] << endl;
      delete [] p;
```

Simple example and code (model.cpp on webpage)

```
//  this function defines the Energy function
double E_function(double x[])
{

//  Change here by calling your Metropolis function which
//  returns the local energy

  double value = x[1]*x[1]*0.5+1.0/(8*x[1]*x[1]);



  return value;
} // end of function to evaluate
```

You need to change this function so that you call the local energy for your system. I used 1000 cycles per call to get a new value of $\langle E_L[\alpha] \rangle$.

Simple example and code (model.cpp on webpage)

```
//  this function defines the derivative of the energy
void dE_function(double x[], double g[])
{

//  Change here by calling your Metropolis function.
//  I compute both the local energy and its derivative for every call

  g[1] = x[1]-1.0/(4*x[1]*x[1]*x[1]);
} // end of function to evaluate
```

You need to change this function so that you call the local energy for your system. I used 1000 cycles per call to get a new value of $\langle E_L[\alpha]\rangle$. When I compute the local energy I also compute its derivative. After roughly 10-20 iterations I got a converged result in terms of α.

Topics for Week 9, March 1-6

Slater determinants and correlation factor

- Repetition from last week
- Conjugate gradient method
- Start discussion of Slater determinant (part 2 of Project 1)

Project work this week: start working on 1d

Topics for Week 10, March 8-12

Slater determinants and classes

- Repetition from last week
- Slater determinant (part 2 of Project 1)
- Discussion of codes for Slater determinant and how to write classes.

Project work this week: Finalize 1d and start writing a code for the Slater determinant.

Slater determinants

The potentially most time-consuming part is the evaluation of the gradient and the Laplacian of an N-particle Slater determinant. We have to differentiate the determinant with respect to all spatial coordinates of all particles. A brute force differentiation would involve $N \cdot d$ evaluations of the entire determinant which would even worsen the already undesirable time scaling, making it $Nd \cdot \mathcal{O}(N^3) \sim \mathcal{O}(d \cdot N^4)$. This poses serious hindrances to the overall efficiency of our code.
The efficiency can be improved however if we move only one electron at the time. The Slater determinant matrix $\mathcal{D}$ is defined by the matrix elements

$$d_{ij} \equiv \phi_j(x_i) \tag{68}$$

where $\phi_j(\mathbf{r}_i)$ is a single particle wave function. The columns correspond to the position of a given particle while the rows stand for the various quantum numbers.

Slater determinants

What we need to realize is that when differentiating a Slater determinant with respect to some given coordinate, only one row of the corresponding Slater matrix is changed. Therefore, by recalculating the whole determinant we risk producing redundant information. The solution turns out to be an algorithm that requires to keep track of the *inverse* of the Slater matrix.
Let the current position in phase space be represented by the $(N \cdot d)$-element vector $\mathbf{r}^{\mathrm{old}}$ and the new suggested position by the vector $\mathbf{r}^{\mathrm{new}}$.
The inverse of $\mathcal{D}$ can be expressed in terms of its cofactors C_{ij} and its determinant $|\mathcal{D}|$:

$$d_{ij}^{-1} = \frac{C_{ji}}{|\mathcal{D}|} \tag{69}$$

Notice that the interchanged indices indicate that the matrix of cofactors is to be transposed.

Slater determinants

If $\mathcal{D}$ is invertible, then we must obviously have $\mathcal{D}^{-1}\mathcal{D} = \mathbf{1}$, or explicitly in terms of the individual elements of $\mathcal{D}$ and $\mathcal{D}^{-1}$:

$$\sum_{k=1}^{N} d_{ik}\, d_{kj}^{-1} = \delta_{ij} \tag{70}$$

Consider the ratio, which we shall call R, between $|\mathcal{D}(\mathbf{r}^{\mathrm{new}})|$ and $|\mathcal{D}(\mathbf{r}^{\mathrm{old}})|$. By definition, each of these determinants can individually be expressed in terms of the *i*th row of its cofactor matrix

$$R \equiv \frac{|\mathcal{D}(\mathbf{r}^{\mathrm{new}})|}{|\mathcal{D}(\mathbf{r}^{\mathrm{old}})|} = \frac{\sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{new}})\, C_{ij}(\mathbf{r}^{\mathrm{new}})}{\sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{old}})\, C_{ij}(\mathbf{r}^{\mathrm{old}})} \tag{71}$$

Slater determinants

Suppose now that we move only one particle at a time, meaning that $\mathbf{r}^{\mathrm{new}}$ differs from $\mathbf{r}^{\mathrm{old}}$ by the position of only one, say the *i*th, particle. This means that $\mathcal{D}(\mathbf{r}^{\mathrm{new}})$ and $\mathcal{D}(\mathbf{r}^{\mathrm{old}})$ differ only by the entries of the *i*th row. Recall also that the *i*th row of a cofactor matrix $\mathcal{C}$ is independent of the entries of the *i*th row of its corresponding matrix $\mathcal{D}$. In this particular case we therefore get that the *i*th row of $\mathcal{C}(\mathbf{r}^{\mathrm{new}})$ and $\mathcal{C}(\mathbf{r}^{\mathrm{old}})$ must be equal. Explicitly, we have:

$$C_{ij}(\mathbf{r}^{\mathrm{new}}) = C_{ij}(\mathbf{r}^{\mathrm{old}}) \quad \forall\, j \in \{1, \dots, N\} \tag{72}$$

Slater determinants

Inserting this into the numerator of eq. (71) and using eq. (69) to substitute the cofactors with the elements of the inverse matrix, we get:

$$R = \frac{\sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{new}})\, C_{ij}(\mathbf{r}^{\mathrm{old}})}{\sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{old}})\, C_{ij}(\mathbf{r}^{\mathrm{old}})} = \frac{\sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{new}})\, d_{ji}^{-1}(\mathbf{r}^{\mathrm{old}})}{\sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{old}})\, d_{ji}^{-1}(\mathbf{r}^{\mathrm{old}})} \tag{73}$$

Slater determinants

Now by eq. (70) the denominator of the rightmost expression must be unity, so that we finally arrive at:

$$R = \sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{new}})\, d_{ji}^{-1}(\mathbf{r}^{\mathrm{old}}) = \sum_{j=1}^{N} \phi_j(\mathbf{r}_i^{\mathrm{new}})\, d_{ji}^{-1}(\mathbf{r}^{\mathrm{old}}) \tag{74}$$

What this means is that in order to get the ratio when only the *i*th particle has been moved, we only need to calculate the dot product of the vector $\left(\phi_1(\mathbf{r}_i^{\mathrm{new}}), \ldots, \phi_N(\mathbf{r}_i^{\mathrm{new}})\right)$ of single particle wave functions evaluated at this new position with the *i*th column of the inverse matrix $\mathcal{D}^{-1}$ evaluated at the original position. Such an operation has a time scaling of $\mathcal{O}(N)$. The only extra thing we need to do is to maintain the inverse matrix $\mathcal{D}^{-1}(\boldsymbol{x}^{\mathrm{old}})$.

Slater determinants

If the new position $\mathbf{r}^{\mathrm{new}}$ is accepted, then the inverse matrix can by suitably updated by an algorithm having a time scaling of $\mathcal{O}(N^2)$. This algorithm goes as follows. First we update all but the ith column of $\mathcal{D}^{-1}$. For each column $j \neq i$, we first calculate the quantity:

$$S_j = (\mathcal{D}(\mathbf{r}^{\mathrm{new}}) \times \mathcal{D}^{-1}(\mathbf{r}^{\mathrm{old}}))_{ij} = \sum_{l=1}^{N} d_{il}(\mathbf{r}^{\mathrm{new}})\, d^{-1}_{lj}(\mathbf{r}^{\mathrm{old}}) \tag{75}$$

The new elements of the jth column of $\mathcal{D}^{-1}$ are then given by:

$$d^{-1}_{kj}(\mathbf{r}^{\mathrm{new}}) = d^{-1}_{kj}(\mathbf{r}^{\mathrm{old}}) - \frac{S_j}{R}\, d^{-1}_{ki}(\mathbf{r}^{\mathrm{old}}) \qquad \begin{array}{l} \forall \;\; k \in \{1, \dots, N\} \\ j \neq i \end{array} \tag{76}$$

Slater determinants

Finally the elements of the *i*th column of $\mathcal{D}^{-1}$ are updated simply as follows:

$$d_{ki}^{-1}(\mathbf{r}^{\mathrm{new}}) = \frac{1}{R}\, d_{ki}^{-1}(\mathbf{r}^{\mathrm{old}}) \quad \forall \;\; k \in \{1, \ldots, N\} \tag{77}$$

We see from these formulas that the time scaling of an update of $\mathcal{D}^{-1}$ after changing one row of $\mathcal{D}$ is $\mathcal{O}(N^2)$.

Slater determinants

The scheme is also applicable for the calculation of the ratios involving derivatives. It turns out that differentiating the Slater determinant with respect to the coordinates of a single particle $\mathbf{r}_i$ changes only the *i*th row of the corresponding Slater matrix.

Slater determinants

The gradient and Laplacian can therefore be calculated as follows:

$$\frac{\boldsymbol{\nabla}_i|\mathcal{D}(\mathbf{r})|}{|\mathcal{D}(\mathbf{r})|} = \sum_{j=1}^{N} \boldsymbol{\nabla}_i d_{ij}(\mathbf{r})\, d_{ji}^{-1}(\mathbf{r}) = \sum_{j=1}^{N} \boldsymbol{\nabla}_i \phi_j(\mathbf{r}_i)\, d_{ji}^{-1}(\mathbf{r}) \tag{78}$$

and

$$\frac{\nabla_i^2|\mathcal{D}(\mathbf{r})|}{|\mathcal{D}(\mathbf{r})|} = \sum_{j=1}^{N} \nabla_i^2 d_{ij}(\mathbf{r})\, d_{ji}^{-1}(\mathbf{r}) = \sum_{j=1}^{N} \nabla_i^2 \phi_j(\mathbf{r}_i)\, d_{ji}^{-1}(\mathbf{r}) \tag{79}$$

Slater determinants

Thus, to calculate all the derivatives of the Slater determinant, we only need the derivatives of the single particle wave functions ($\boldsymbol{\nabla}_i\phi_j(\mathbf{r}_i)$ and $\nabla_i^2\phi_j(\mathbf{r}_i)$) and the elements of the corresponding inverse Slater matrix ($\mathcal{D}^{-1}(\mathbf{r}_i)$). A calculation of a single derivative is by the above result an $\mathcal{O}(N)$ operation. Since there are $d \cdot N$ derivatives, the time scaling of the total evaluation becomes $\mathcal{O}(d \cdot N^2)$. With an $\mathcal{O}(N^2)$ updating algorithm for the inverse matrix, the total scaling is no worse, which is far better than the brute force approach yielding $\mathcal{O}(d \cdot N^4)$.
Important note: In most cases you end with closed form expressions for the single-particle wave functions. It is then useful to calculate the various derivatives and make separate functions for them.

Slater determinant: Explicit expressions for various Atoms, beryllium

The Slater determinant takes the form

$$\Phi(\mathbf{r}_1,\mathbf{r}_2,,\mathbf{r}_3,\mathbf{r}_4,\alpha,\beta,\gamma,\delta) = \frac{1}{\sqrt{4!}} \left| \begin{array}{cccc} \psi_{100\uparrow}(\mathbf{r}_1) & \psi_{100\uparrow}(\mathbf{r}_2) & \psi_{100\uparrow}(\mathbf{r}_3) & \psi_{100\uparrow}(\mathbf{r}_4) \\ \psi_{100\downarrow}(\mathbf{r}_1) & \psi_{100\downarrow}(\mathbf{r}_2) & \psi_{100\downarrow}(\mathbf{r}_3) & \psi_{100\downarrow}(\mathbf{r}_4) \\ \psi_{200\uparrow}(\mathbf{r}_1) & \psi_{200\uparrow}(\mathbf{r}_2) & \psi_{200\uparrow}(\mathbf{r}_3) & \psi_{200\uparrow}(\mathbf{r}_4) \\ \psi_{200\downarrow}(\mathbf{r}_1) & \psi_{200\downarrow}(\mathbf{r}_2) & \psi_{200\downarrow}(\mathbf{r}_3) & \psi_{200\downarrow}(\mathbf{r}_4) \end{array} \right|.$$

The Slater determinant as written is zero since the spatial wave functions for the spin up and spin down states are equal. But we can rewrite it as the product of two Slater determinants, one for spin up and one for spin down.

Slater determinant: Explicit expressions for various Atoms, beryllium

We can rewrite it as

$$\Phi(\mathbf{r}_1,\mathbf{r}_2,,\mathbf{r}_3,\mathbf{r}_4,\alpha,\beta,\gamma,\delta) = Det\uparrow(1,2)Det\downarrow(3,4) - Det\uparrow(1,3)Det\downarrow(2,4)$$

$$-Det\uparrow(1,4)Det\downarrow(3,2) + Det\uparrow(2,3)Det\downarrow(1,4) - Det\uparrow(2,4)Det\downarrow(1,3)$$

$$+Det\uparrow(3,4)Det\downarrow(1,2),$$

where we have defined

$$Det\uparrow(1,2) = \frac{1}{\sqrt{2}}\left|\begin{array}{cc} \psi_{100\uparrow}(\mathbf{r}_1) & \psi_{100\uparrow}(\mathbf{r}_2) \\ \psi_{200\uparrow}(\mathbf{r}_1) & \psi_{200\uparrow}(\mathbf{r}_2) \end{array}\right|,$$

and

$$Det\downarrow(3,4) = \frac{1}{\sqrt{2}}\left|\begin{array}{cc} \psi_{100\downarrow}(\mathbf{r}_3) & \psi_{100\downarrow}(\mathbf{r}_4) \\ \psi_{200\downarrow}(\mathbf{r}_3) & \psi_{200\downarrow}(\mathbf{r}_4) \end{array}\right|.$$

The total determinant is still zero!

Slater determinant: Explicit expressions for various Atoms, beryllium

We want to avoid to sum over spin variables, in particular when the interaction does not depend on spin.
It can be shown, see for example Moskowitz and Kalos, Int. J. Quantum Chem. **20** (1981) 1107, that for the variational energy we can approximate the Slater determinant as

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, , \mathbf{r}_3, \mathbf{r}_4, \alpha, \beta, \gamma, \delta) \propto Det \uparrow (1,2) Det \downarrow (3,4),$$

or more generally as

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \dots \mathbf{r}_N) \propto Det \uparrow Det \downarrow,$$

where we have the Slater determinant as the product of a spin up part involving the number of electrons with spin up only (2 in Beryllium and 5 in neon) and a spin down part involving the electrons with spin down.
This ansatz is not antisymmetric under the exchange of electrons with opposite spins but it can be shown that it gives the same expectation value for the energy as the full Slater determinant.

As long as the Hamiltonian is spin independent, the above is correct. Exercise for next week: convince yourself that this is correct.

Code vmc_be.cpp at the webpage

In this code I have included the Slater determinant for the beryllium atom. Note that the implementation is rather brute force like.

```
 for (i = 0; i < number_particles; i++) {
    argument[i] = 0.0;
    r_single_particle = 0;
    for (j = 0; j < dimension; j++) {
      r_single_particle  += r[i][j]*r[i][j];
    }
    argument[i] = sqrt(r_single_particle);
  }
// Slater determinant, no factors as they vanish in Metropolis ratio
wf  = (psi1s(argument[0])*psi2s(argument[1])
       -psi1s(argument[1])*psi2s(argument[0]))*
      (psi1s(argument[2])*psi2s(argument[3])
       -psi1s(argument[3])*psi2s(argument[2]));
```

For beryllium we can easily implement the explicit evaluation of the Slater determinant.

The derivatives of the single-particle wave functions can be computed analytically.

Slater determinants

We will thus factorize the full determinant $|\mathcal{D}|$ into two smaller ones, where each can be identified with $\uparrow$ and $\downarrow$ respectively:

$$|\mathcal{D}| = |\mathcal{D}|_\uparrow \cdot |\mathcal{D}|_\downarrow \tag{80}$$

The combined dimensionality of the two smaller determinants equals the dimensionality of the full determinant. Such a factorization is advantageous in that it makes it possible to perform the calculation of the ratio R and the updating of the inverse matrix separately for $|\mathcal{D}|_\uparrow$ and $|\mathcal{D}|_\downarrow$:

$$\frac{|\mathcal{D}|^{\mathrm{new}}}{|\mathcal{D}|^{\mathrm{old}}} = \frac{|\mathcal{D}|^{\mathrm{new}}_\uparrow}{|\mathcal{D}|^{\mathrm{old}}_\uparrow} \cdot \frac{|\mathcal{D}|^{\mathrm{new}}_\downarrow}{|\mathcal{D}|^{\mathrm{old}}_\downarrow} \tag{81}$$

Slater determinants

This reduces the calculation time by a constant factor. The maximal time reduction happens in a system of equal numbers of $\uparrow$ and $\downarrow$ particles, so that the two factorized determinants are half the size of the original one.
Consider the case of moving only one particle at a time which originally had the following time scaling for one transition:

$$\mathcal{O}_R(N) + \mathcal{O}_{\text{inverse}}(N^2) \tag{82}$$

For the factorized determinants one of the two determinants is obviously unaffected by the change so that it cancels from the ratio R.

Slater determinants

Therefore, only one determinant of size $N/2$ is involved in each calculation of R and update of the inverse matrix. The scaling of each transition then becomes:

$$\mathcal{O}_R(N/2) + \mathcal{O}_{\mathrm{inverse}}(N^2/4) \tag{83}$$

and the time scaling when the transitions for all N particles are put together:

$$\mathcal{O}_R(N^2/2) + \mathcal{O}_{\mathrm{inverse}}(N^3/4) \tag{84}$$

which gives the same reduction as in the case of moving all particles at once.

Updating the Slater matrix

Computing the ratios discussed above requires that we maintain the inverse of the Slater matrix evaluated at the current position. Each time a trial position is accepted, the row number i of the Slater matrix changes and updating its inverse has to be carried out. Getting the inverse of an $N \times N$ matrix by Gaussian elimination has a complexity of order of $\mathcal{O}(N^3)$ operations, a luxury that we cannot afford for each time a particle move is accepted. We will use the expression

$$\boxed{d_{kj}^{-1}(\boldsymbol{x^{new}}) = \begin{cases} d_{kj}^{-1}(\boldsymbol{x^{old}}) - \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R}\sum_{l=1}^{N} d_{il}(\boldsymbol{x^{new}}) d_{lj}^{-1}(\boldsymbol{x^{old}}) & \text{if } j \neq i \\ \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R}\sum_{l=1}^{N} d_{il}(\boldsymbol{x^{old}}) d_{lj}^{-1}(\boldsymbol{x^{old}}) & \text{if } j = i \end{cases}} \quad (85)$$

Updating the Slater matrix

This equation scales as $O(N^2)$. The evaluation of the determinant of an $N \times N$ matrix by standard Gaussian elimination requires $\mathcal{O}(N^3)$ calculations. As there are Nd independent coordinates we need to evaluate Nd Slater determinants for the gradient (quantum force) and Nd for the Laplacian (kinetic energy). With the updating algorithm we need only to invert the Slater determinant matrix once. This can be done by standard LU decomposition methods.

Slater Determinant and VMC

Determining a determinant of an $N \times N$ matrix by standard Gaussian elimination is of the order of $\mathcal{O}(N^3)$ calculations. As there are $N \cdot d$ independent coordinates we need to evaluate Nd Slater determinants for the gradient (quantum force) and $N \cdot d$ for the Laplacian (kinetic energy)

With the updating algorithm we need only to invert the Slater determinant matrix once.

This is done by calling standard LU decomposition methods.

How to compute the Slater Determinant

If you choose to implement the above recipe for the computation of the Slater determinant, you need to LU decompose the Slater matrix. This is described in chapter 4 of the lecture notes.

You need to call the function ludcmp in lib.cpp. You need to transfer the Slater matrix and its dimension. You get back an LU decomposed matrix.

LU Decomposition

The LU decomposition method means that we can rewrite this matrix as the product of two matrices **B** and **C** where

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ b_{21} & 1 & 0 & 0 \\ b_{31} & b_{32} & 1 & 0 \\ b_{41} & b_{42} & b_{43} & 1 \end{pmatrix} \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ 0 & c_{22} & c_{23} & c_{24} \\ 0 & 0 & c_{33} & c_{34} \\ 0 & 0 & 0 & c_{44} \end{pmatrix}.$$

The matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ has an LU factorization if the determinant is different from zero. If the LU factorization exists and **A** is non-singular, then the LU factorization is unique and the determinant is given by

$$det\{\mathbf{A}\} = c_{11}c_{22}\dots c_{nn}.$$

Proof for updating algorithm of the Slater matrix

As a starting point we may consider that each time a new position is suggested in the Metropolis algorithm, a row of the current Slater matrix experiences some kind of perturbation. Hence, the Slater matrix with its orbitals evaluated at the new position equals the old Slater matrix plus a perturbation matrix,

$$d_{jk}(\boldsymbol{x}^{\boldsymbol{new}}) = d_{jk}(\boldsymbol{x}^{\boldsymbol{old}}) + \Delta_{jk}, \tag{86}$$

where

$$\Delta_{jk} = \delta_{ik}[\phi_j(\boldsymbol{x}_i^{\boldsymbol{new}}) - \phi_j(\boldsymbol{x}_i^{\boldsymbol{old}})] = \delta_{ik}(\Delta\phi)_j. \tag{87}$$

Proof for updating algorithm of the Slater matrix

Computing the inverse of the transposed matrix we arrive to

$$d_{kj}(\boldsymbol{x^{new}})^{-1} = [d_{kj}(\boldsymbol{x^{old}}) + \Delta_{kj}]^{-1}. \tag{88}$$

The evaluation of the right hand side (rhs) term above is carried out by applying the identity $(A+B)^{-1} = A^{-1} - (A+B)^{-1}BA^{-1}$. In compact notation it yields

$$\begin{aligned}
[\boldsymbol{D}^T(\boldsymbol{x^{new}})]^{-1} &= [\boldsymbol{D}^T(\boldsymbol{x^{old}}) + \Delta^T]^{-1} \\
&= [\boldsymbol{D}^T(\boldsymbol{x^{old}})]^{-1} - [\boldsymbol{D}^T(\boldsymbol{x^{old}}) + \Delta^T]^{-1}\Delta^T[\boldsymbol{D}^T(\boldsymbol{x^{old}})]^{-1} \\
&= [\boldsymbol{D}^T(\boldsymbol{x^{old}})]^{-1} - \underbrace{[\boldsymbol{D}^T(\boldsymbol{x^{new}})]^{-1}}_{\text{By Eq.88}}\Delta^T[\boldsymbol{D}^T(\boldsymbol{x^{old}})]^{-1}.
\end{aligned}$$

Proof for updating algorithm of the Slater matrix

Using index notation, the last result may be expanded by

$$\begin{aligned}
d_{kj}^{-1}(\boldsymbol{x}^{\boldsymbol{new}}) &= d_{kj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) - \sum_l \sum_m d_{km}^{-1}(\boldsymbol{x}^{\boldsymbol{new}}) \Delta_{ml}^T d_{lj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) \\
&= d_{kj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) - \sum_l \sum_m d_{km}^{-1}(\boldsymbol{x}^{\boldsymbol{new}}) \Delta_{lm} d_{lj}^{-1}(\boldsymbol{x}^{\boldsymbol{cur}}) \\
&= d_{kj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) - \sum_l \sum_m d_{km}^{-1}(\boldsymbol{x}^{\boldsymbol{new}}) \underbrace{\delta_{im}(\Delta\phi)_l}_{\text{By Eq. 87}} d_{lj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) \\
&= d_{kj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) - d_{ki}^{-1}(\boldsymbol{x}^{\boldsymbol{new}}) \sum_{l=1}^{N} (\Delta\phi)_l d_{lj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) \\
&= d_{kj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}) - d_{ki}^{-1}(\boldsymbol{x}^{\boldsymbol{new}}) \sum_{l=1}^{N} \underbrace{[\phi_l(\boldsymbol{r}_{\boldsymbol{i}}^{\boldsymbol{new}}) - \phi_l(\boldsymbol{r}_{\boldsymbol{i}}^{\boldsymbol{old}})]}_{\text{By Eq.87}} D_{lj}^{-1}(\boldsymbol{x}^{\boldsymbol{old}}).
\end{aligned}$$

Proof for updating algorithm of the Slater matrix

Using

$$\boldsymbol{D}^{-1}(\boldsymbol{x}^{old}) = \frac{adj\boldsymbol{D}}{|\boldsymbol{D}(\boldsymbol{x}^{old})|} \quad \text{and} \quad \boldsymbol{D}^{-1}(\boldsymbol{x}^{new}) = \frac{adj\boldsymbol{D}}{|\boldsymbol{D}(\boldsymbol{x}^{new})|},$$

and dividing these two equations we get

$$\frac{\boldsymbol{D}^{-1}(\boldsymbol{x}^{old})}{\boldsymbol{D}^{-1}(\boldsymbol{x}^{new})} = \frac{|\boldsymbol{D}(\boldsymbol{x}^{new})|}{|\boldsymbol{D}(\boldsymbol{x}^{old})|} = R \Rightarrow d_{ki}^{-1}(\boldsymbol{x}^{new}) = \frac{d_{ki}^{-1}(\boldsymbol{x}^{old})}{R}.$$

Therefore,

$$d_{kj}^{-1}(\boldsymbol{x}^{new}) = d_{kj}^{-1}(\boldsymbol{x}^{old}) - \frac{d_{ki}^{-1}(\boldsymbol{x}^{old})}{R} \sum_{l=1}^{N} [\phi_l(\boldsymbol{r}_i^{new}) - \phi_l(\boldsymbol{r}_i^{old})] d_{lj}^{-1}(\boldsymbol{x}^{old}),$$

Proof for updating algorithm of the Slater matrix

or

$$
\begin{aligned}
d_{kj}^{-1}(\boldsymbol{x^{new}}) = d_{kj}^{-1}(\boldsymbol{x^{old}}) \quad & - \quad \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R}\sum_{l=1}^{N}\phi_l(\boldsymbol{r_i^{new}})d_{lj}^{-1}(\boldsymbol{x^{old}}) \\
& + \quad \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R}\sum_{l=1}^{N}\phi_l(\boldsymbol{r_i^{old}})d_{lj}^{-1}(\boldsymbol{x^{old}}) \\
= d_{kj}^{-1}(\boldsymbol{x^{old}}) \quad & - \quad \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R}\sum_{l=1}^{N}d_{il}(\boldsymbol{x^{new}})d_{lj}^{-1}(\boldsymbol{x^{old}}) \\
& + \quad \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R}\sum_{l=1}^{N}d_{il}(\boldsymbol{x^{old}})d_{lj}^{-1}(\boldsymbol{x^{old}}).
\end{aligned}
$$

Proof for updating algorithm of the Slater matrix

In this equation, the first line becomes zero for $j = i$ and the second for $j \neq i$. Therefore, the update of the inverse for the new Slater matrix is given by

$$\boxed{d_{kj}^{-1}(\boldsymbol{x^{new}}) = \begin{cases} d_{kj}^{-1}(\boldsymbol{x^{old}}) - \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R} \sum_{l=1}^{N} d_{il}(\boldsymbol{x^{new}}) d_{lj}^{-1}(\boldsymbol{x^{old}}) & \text{if } j \neq i \\ \frac{d_{ki}^{-1}(\boldsymbol{x^{old}})}{R} \sum_{l=1}^{N} d_{il}(\boldsymbol{x^{old}}) d_{lj}^{-1}(\boldsymbol{x^{old}}) & \text{if } j = i \end{cases}}$$

Topics for Week 11, March 15-19

Slater determinants and classes

- Repetition from last week
- Slater determinant (part 2 of Project 1)
- Discussion of codes for Slater determinant and how to write classes.

Project work this week: Finalize 1d and start writing a code for the Slater determinant. See also the code HeimportanceCGM.cpp at the webpage!

How should we structure our code?

What do you think is reasonable to split into subtasks defined by classes?

- Single-particle wave functions?
- External potentials?
- Operations on r_{ij} and the correlation function?
- Mathematical operations like the first and second derivative of the trial wave function? How can you split the derivatives into various subtasks?
- Matrix and vector operations?

Your task till next week is to figure out how to structure your code in order to compute the Slater determinant for Beryllium. This should be compared with the brute force case. Do not include the correlation factor in the first attempt.

A useful piece of code, distances

```
double r_i(double**, int);
//distance between  nucleus and electron i
double r_ij(double**, int, int);
//distance between electrons i and j
```

A useful piece of code, single-particle functions

```
//Hydrogen-like  single-particle  functions,
    beryllium  case
//phi_j(r_i)
double phi(int j, double** R, int i, double alpha)
    {
  if (j == 0) {
    return exp(-alpha*r_i(R,i));
  }
  if (j == 1) {
    double r = r_i(R,i);
    return (1-alpha*r/2)*exp(-alpha*r/2);
  }
}
```

A useful piece of code, single-particle functions

```
// First derivative of Hydrogen-like orbits
// dphi_j(r_i)/dr_ik
double phi_deriv(int j, double** R, int i, int k,
    double alpha) {
  if (j == 0) {
    double r = r_i(R,i);
    return -alpha*R[i][k]*exp(-alpha*r) / r;
  }
  if (j == 1) {
    double r = r_i(R,i);
    return -alpha*R[i][k]*(2-alpha*r/2)*exp(-alpha*
        r/2) / (2*r);
  }
}
```

A useful piece of code, single-particle functions

```
//second derivative
//d^2phi_j(r_i)/dr_ik^2
double phi_deriv2(int j, double** R, int i, int k,
    double alpha) {
  if (j == 0) {
    double r = r_i(R,i);
    return (pow(alpha*R[i][k]/r,2) -
    alpha*(r*r-R[i][k]*R[i][k])/(r*r*r))*exp(-alpha
        *r);
  }
  if (j == 1) {
    double r = r_i(R,i);
    return -alpha/2 * ( (r*r-R[i][k]*R[i][k])/(r*r*
        r)*(2-alpha*r/2)
    - alpha/2*pow(R[i][k]/r,2)*(3-alpha*r/2) )*exp
        (-alpha*r/2);
  }
}
```

The function to set up a determinant

```
//Determinant function
double determinant(double** A, int dim) {
  if (dim == 2)
    return A[0][0]*A[1][1] - A[0][1]*A[1][0];
  double sum = 0;
  for (int i = 0; i < dim; i++) {
    double** sub = new double*[dim-1];
    for (int j = 0; j < i; j++)
      sub[j] = &A[j][1];
    for (int j = i+1; j < dim; j++)
      sub[j-1] = &A[j][1];
    if(i % 2 == 0)
      sum += A[i][0] * determinant(sub,dim-1);
    else
      sum -= A[i][0] * determinant(sub,dim-1);

    delete[] sub;
  }
  return sum;
}
```

Set up the Slater determinant

```
//Slater-determinant
double slater(double** R, double alpha, double Z,
    double Z2) {
  double** DUp = (double**) matrix(Z2,Z2,sizeof(
      double));
  double** DDown = (double**) matrix(Z2,Z2,sizeof(
      double));
  for (int i = 0; i < Z2; i++) {
    for (int j = 0; j < Z2; j++) {
      DUp[i][j] = phi(j,R,i,alpha);
      DDown[i][j] = phi(j,R,i+Z2,alpha);
    }
  }
  //Returns product of spin up and spin down dets
  double det = determinant(DUp,Z2)*determinant(
      DDown,Z2);
  free_matrix((void**) DUp);
  free_matrix((void**) DDown);
  return det;
}
```

Jastrow factor

```
//Jastrow-faktor
double jastrow(double** R, double beta, double Z,
    double Z2) {
  double arg = 0;
  for (int i = 1; i < Z; i++)
    for (int j = 0; j < i; j++)
      if ((i < Z2 && j < Z2) || (i >= Z2 && j >= Z2
          )) {
        double rij = r_ij(R,i,j);
        arg += .25*rij / (1+beta*rij); //samme
            spinn
      }
      else {
        double rij = r_ij(R,i,j);
        arg += .5*rij / (1+beta*rij); //motsatt
            spinn
      }
  return exp(arg);
}
```

```
//Check of singularity at R = 0
bool Singularity(double** R, int Z) {

  for (int i = 0; i < Z; i++)
    if (r_i(R,i) < 1e-10)
      return true;

  for (int i = 0; i < Z - 1; i++)
    for (int j = i+1; j < Z; j++)
      if (r_ij(R,i,j) < 1e-10)
        return true;
  return false;
}
```

Efficient calculations of wave function ratios

The expectation value of the kinetic energy expressed in atomic units for electron *i* is

$$\langle \widehat{\mathbf{K}}_i \rangle = -\frac{1}{2}\frac{\langle \Psi | \nabla_i^2 | \Psi \rangle}{\langle \Psi | \Psi \rangle}, \tag{89}$$

$$K_i = -\frac{1}{2}\frac{\nabla_i^2 \Psi}{\Psi}. \tag{90}$$

$$\begin{aligned}
\frac{\nabla^2 \Psi}{\Psi} &= \frac{\nabla^2 (\Psi_D \, \Psi_C)}{\Psi_D \, \Psi_C} = \frac{\boldsymbol{\nabla}\cdot[\boldsymbol{\nabla}(\Psi_D \, \Psi_C)]}{\Psi_D \, \Psi_C} = \frac{\boldsymbol{\nabla}\cdot[\Psi_C \boldsymbol{\nabla}\Psi_D + \Psi_D \boldsymbol{\nabla}\Psi_C]}{\Psi_D \, \Psi_C} \\
&= \frac{\boldsymbol{\nabla}\Psi_C \cdot \boldsymbol{\nabla}\Psi_D + \Psi_C \nabla^2 \Psi_D + \boldsymbol{\nabla}\Psi_D \cdot \boldsymbol{\nabla}\Psi_C + \Psi_D \nabla^2 \Psi_C}{\Psi_D \, \Psi_C}
\end{aligned} \tag{91}$$

$$\frac{\nabla^2 \Psi}{\Psi} = \frac{\nabla^2 \Psi_D}{\Psi_D} + \frac{\nabla^2 \Psi_C}{\Psi_C} + 2\frac{\boldsymbol{\nabla}\Psi_D}{\Psi_D} \cdot \frac{\boldsymbol{\nabla}\Psi_C}{\Psi_C} \tag{92}$$

Summing up: Bringing it all together, Local energy

The second derivative of the Jastrow factor divided by the Jastrow factor (the way it enters the kinetic energy) is

$$\left[\frac{\nabla^2\Psi_C}{\Psi_C}\right]_x = 2\sum_{k=1}^{N}\sum_{i=1}^{k-1}\frac{\partial^2 g_{ik}}{\partial x_k^2} + \sum_{k=1}^{N}\left(\sum_{i=1}^{k-1}\frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^{N}\frac{\partial g_{ki}}{\partial x_i}\right)^2$$

But we have a simple form for the function, namely

$$\Psi_C = \prod_{i<j}\exp f(r_{ij}) = \exp\left\{\sum_{i<j}\frac{ar_{ij}}{1+\beta r_{ij}}\right\},$$

and it is easy to see that for particle k we have

$$\frac{\nabla_k^2\Psi_C}{\Psi_C} = \sum_{ij\neq k}\frac{(\mathbf{r}_k-\mathbf{r}_i)(\mathbf{r}_k-\mathbf{r}_j)}{r_{ki}r_{kj}}f'(r_{ki})f'(r_{kj}) + \sum_{j\neq k}\left(f''(r_{kj}) + \frac{2}{r_{kj}}f'(r_{kj})\right)$$

Bringing it all together, Local energy

Using

$$f(r_{ij}) = \frac{ar_{ij}}{1+\beta r_{ij}},$$

and $g'(r_{kj}) = dg(r_{kj})/dr_{kj}$ and $g''(r_{kj}) = d^2g(r_{kj})/dr_{kj}^2$ we find that for particle k we have

$$\frac{\nabla_k^2\Psi_C}{\Psi_C} = \sum_{ij\neq k}\frac{(\mathbf{r}_k-\mathbf{r}_i)(\mathbf{r}_k-\mathbf{r}_j)}{r_{ki}r_{kj}}\frac{a}{(1+\beta r_{ki})^2}\frac{a}{(1+\beta r_{kj})^2}+\sum_{j\neq k}\left(\frac{2a}{r_{kj}(1+\beta r_{kj})^2}-\frac{2a\beta}{(1+\beta r_{kj})^3}\right)$$

Local energy

The gradient and Laplacian can be calculated as follows:

$$\frac{\boldsymbol{\nabla}_i|\mathcal{D}(\mathbf{r})|}{|\mathcal{D}(\mathbf{r})|} = \sum_{j=1}^{N} \boldsymbol{\nabla}_i d_{ij}(\mathbf{r})\, d_{ji}^{-1}(\mathbf{r}) = \sum_{j=1}^{N} \boldsymbol{\nabla}_i \phi_j(\mathbf{r}_i)\, d_{ji}^{-1}(\mathbf{r})$$

and

$$\frac{\nabla_i^2|\mathcal{D}(\mathbf{r})|}{|\mathcal{D}(\mathbf{r})|} = \sum_{j=1}^{N} \nabla_i^2 d_{ij}(\mathbf{r})\, d_{ji}^{-1}(\mathbf{r}) = \sum_{j=1}^{N} \nabla_i^2 \phi_j(\mathbf{r}_i)\, d_{ji}^{-1}(\mathbf{r})$$

Local energy function

```
double E_local(double** R, double alpha, double
    beta, int Z, double** F, double** DinvUp,
                 double** DinvDown, int Z2, double**
                    detgrad, double** jastgrad) {

  //Kinetic energy
  double kinetic = 0;
  //Determinant part
  for (int i = 0; i < Z; i++) {
    for (int j = 0; j < 3; j++) {

      if (i < Z2)
        for (int l = 0; l < Z2; l++)
          kinetic -= phi_deriv2(l,R,i,j,alpha)*
            DinvUp[l][i];
      else
        for (int l = 0; l < Z2; l++)
          kinetic -= phi_deriv2(l,R,i,j,alpha)*
            DinvDown[l][i-Z2];
    }
```

Jastrow part

```
//Jastrow part
double rij,a;
for (int i = 0; i < Z; i++) {
  for (int j = 0; j < 3; j++) {
    kinetic -= jastgrad[i][j]*jastgrad[i][j];
  }
}
for (int i = 0; i < Z-1; i++) {
  for (int j = i+1; j < Z; j++) {
    if ((j < Z2 && i < Z2) || (j >= Z2 && i >= Z2
        ))
      a = .25;
    else
      a = .5;
    rij = r_ij(R,i,j);
    kinetic -= 4*a / (rij*pow(1+beta*rij,3));
  }
}
```

Local energy

```
//"Interference" part
for (int i = 0; i < Z; i++) {
  for (int j = 0; j < 3; j++) {
    kinetic -= 2*detgrad[i][j]*jastgrad[i][j];
  }
}

kinetic *= .5;
```

```
  //Potential energy
  //electron-nucleus potential
  double potential = 0;
  for (int i = 0; i < Z; i++)
    potential -= Z / r_i(R,i);

  //electron-electron potential
  for (int i = 0; i < Z - 1; i++)
    for (int j = i+1; j < Z; j++)
      potential += 1 / r_ij(R,i,j);
  return potential + kinetic;
}
```

Determinant part in quantum force

The gradient for the determinant is

$$\frac{\nabla_i|\mathcal{D}(\mathbf{r})|}{|\mathcal{D}(\mathbf{r})|} = \sum_{j=1}^{N} \nabla_i d_{ij}(\mathbf{r})\, d_{ji}^{-1}(\mathbf{r}) = \sum_{j=1}^{N} \nabla_i \phi_j(\mathbf{r}_i)\, d_{ji}^{-1}(\mathbf{r}).$$

Quantum force

```
void calcQF(double** R, double** F, double alpha,
    double beta,
              int Z, double** DinvUp, double**
                  DinvDown, int Z2, double** detgrad,
                   double** jastgrad) {
  double sum;
  //Determinant part
  for (int i = 0; i < Z; i++) {
    for (int j = 0; j < 3; j++) {
      sum = 0;
      if (i < Z2)
        for (int l = 0; l < Z2; l++)
          sum += phi_deriv(l,R,i,j,alpha)*DinvUp[l
              ][i];
      else
        for (int l = 0; l < Z2; l++)
          sum += phi_deriv(l,R,i,j,alpha)*DinvDown[
              l][i-Z2];
      detgrad[i][j] = sum;
    }
```

Jastrow gradient in quantum force

We have

$$\Psi_C = \prod_{i<j} g(r_{ij}) = \exp\left\{\sum_{i<j} \frac{ar_{ij}}{1+\beta r_{ij}}\right\},$$

the gradient needed for the quantum force and local energy is easy to compute. We get for particle k

$$\frac{\nabla_k \Psi_C}{\Psi_C} = \sum_{j\neq k} \frac{\mathbf{r}_{kj}}{r_{kj}} \frac{a}{(1+\beta r_{kj})^2},$$

which is rather easy to code. Remember to sum over all particles when you compute the local energy.

Jastrow part

```
// Jastrowdel
double ril ,a;
for (int i = 0; i < Z; i++) {
  for(int j = 0; j < 3; j++) {
    sum = 0;
    for (int l = 0; l < Z; l++) {
      if (l != i) {
        if ((l < Z2 && i < Z2) || (l >= Z2 && i
            >= Z2))
          a = .25;
        else
          a = .5;
          ril = r_ij(R,i,l);
          sum += (R[i][j]-R[l][j])*a / (ril*pow(1+
              beta*ril,2));
        }
      }
      jastgrad[i][j] = sum;
    }
  }
  for (int i = 0; i < Z; i++)
    for(int j = 0; j < 3; j++)
      F[i][j] = 2*(detgrad[i][j] + jastgrad[i][j]);
}
```

Metropolis-Hastings part

```
//Initialize positions
double** R = (double**) matrix(Z,3,sizeof(double)
    );
for (int i = 0; i < Z; i++)
  for (int j = 0; j < 3; j++)
    R[i][j] = gaussian_deviate(&idum);


int Z2 = Z/2; //dimension of Slater matrix
```

Metropolis Hastings part

We need to compute the ratio between wave functions, in particular for the Slater determinants.

$$R = \sum_{j=1}^{N} d_{ij}(\mathbf{r}^{\mathrm{new}})\, d_{ji}^{-1}(\mathbf{r}^{\mathrm{old}}) = \sum_{j=1}^{N} \phi_j(\mathbf{r}_i^{\mathrm{new}})\, d_{ji}^{-1}(\mathbf{r}^{\mathrm{old}})$$

What this means is that in order to get the ratio when only the *i*th particle has been moved, we only need to calculate the dot product of the vector $\left(\phi_1(\mathbf{r}_i^{\mathrm{new}}), \ldots, \phi_N(\mathbf{r}_i^{\mathrm{new}})\right)$ of single particle wave functions evaluated at this new position with the *i*th column of the inverse matrix $\mathcal{D}^{-1}$ evaluated at the original position. Such an operation has a time scaling of $\mathcal{O}(N)$. The only extra thing we need to do is to maintain the inverse matrix $\mathcal{D}^{-1}(\boldsymbol{x}^{\mathrm{old}})$.

Jastrow factor in Metropolis Hastings

We have

$$\boxed{R_C = \frac{\Psi_C^{\mathrm{new}}}{\Psi_C^{\mathrm{cur}}} = \frac{e^{U_{new}}}{e^{U_{cur}}} = e^{\Delta U},} \tag{93}$$

where

$$\Delta U = \sum_{i=1}^{k-1} \left(f_{ik}^{\mathrm{new}} - f_{ik}^{\mathrm{cur}}\right) + \sum_{i=k+1}^{N} \left(f_{ki}^{\mathrm{new}} - f_{ki}^{\mathrm{cur}}\right) \tag{94}$$

One needs to develop a special algorithm that runs only through the elements of the upper triangular matrix $\boldsymbol{g}$ and have k as an index.

Metropolis-Hastings part

```
//Initialize inverse Slater matrices for spin up
    and spin down
double** DinvUp = (double**) matrix(Z2,Z2,sizeof(
    double));
double** DinvDown = (double**) matrix(Z2,Z2,
    sizeof(double));
for (int i = 0; i < Z2; i++) {
  for (int j = 0; j < Z2; j++) {
    DinvUp[i][j] = phi(j,R,i,alpha);
    DinvDown[i][j] = phi(j,R,i+Z2,alpha);
  }
}
inverse(DinvUp,Z2);
inverse(DinvDown,Z2);
```

Metropolis-Hastings part

```
//Inverse Slater matrix in new position
double** DinvUp_new = (double**) matrix(Z2,Z2,
    sizeof(double));
double** DinvDown_new = (double**) matrix(Z2,Z2,
    sizeof(double));
for (int i = 0; i < Z2; i++) {
  for (int j = 0; j < Z2; j++) {
    DinvUp_new[i][j] = DinvUp[i][j];
    DinvDown_new[i][j] = DinvDown[i][j];
  }
}
```

Metropolis-Hastings part

```
//Gradients of determinant and and Jastrow factor
double** detgrad = (double**) matrix(Z,3,sizeof(
    double));
double** jastgrad = (double**) matrix(Z,3,sizeof(
    double));
double** detgrad_new = (double**) matrix(Z,3,
    sizeof(double));
double** jastgrad_new = (double**) matrix(Z,3,
    sizeof(double));

//Initialize quantum force
double** F = (double**) matrix(Z,3,sizeof(double)
    );
calcQF(R,F,alpha,beta,Z,DinvUp,DinvDown,Z2,
    detgrad,jastgrad);
```

Metropolis-Hastings part

```
double EL; //Local  energy
double sqrtdt = sqrt(delta_t);
double D = .5; //diffusion constant
//For Metropolis-Hastings algo:
double** R_new = (double**) matrix(Z,3,sizeof(
    double));
double** F_new = (double**) matrix(Z,3,sizeof(
    double));
double greensratio; // Ratio between Green's
    functions
double detratio; //Ratio between Slater
    determinants
double jastratio; //Ratio between Jastrow factors
double rold,rnew,a;
double alphaderiv,betaderiv;
```

Metropolis-Hastings part, inside Monte Carlo loop

```
//Ratio between Slater determinants
if (i < Z2) {
  detratio = 0;
  for (int l = 0; l < Z2; l++)
    detratio += phi(l,R_new,i,alpha) * DinvUp
        [l][i];
}
else {
  detratio = 0;
  for (int l = 0; l < Z2; l++)
    detratio += phi(l,R_new,i,alpha) *
        DinvDown[l][i-Z2];
}
```

Metropolis-Hastings part

```
//Inverse Slater matrix in new position
if (i < Z2) { //Spinn up
  for (int j = 0; j < Z2; j++) {
    if (j != i) {
      Sj = 0;
      for (int l = 0; l < Z2; l++) {
        Sj += phi(l,R_new,i,alpha) * DinvUp[l
            ][j];
      }
      for (int l = 0; l < Z2; l++)
        DinvUp_new[l][j] = DinvUp[l][j] - Sj
            * DinvUp[l][i] / detratio;
    }
  }
  for (int l = 0; l < Z2; l++)
    DinvUp_new[l][i] = DinvUp[l][i] /
        detratio;
}
```

Metropolis-Hastings part

```
else {  //Spinn-ned
  for (int j = 0; j < Z2; j++) {
    if (j != i-Z2) {
      Sj = 0;
      for (int l = 0; l < Z2; l++) {
        Sj += phi(l,R_new,i,alpha) * DinvDown
             [l][j];
      }
      for (int l = 0; l < Z2; l++)
        DinvDown_new[l][j] = DinvDown[l][j] -
             Sj * DinvDown[l][i-Z2] /
             detratio;
    }
  }
  for (int l = 0; l < Z2; l++)
    DinvDown_new[l][i-Z2] = DinvDown[l][i-Z2]
         / detratio;
}
```

Jastrow ratio

```
//Ratio between Jastrow factors
jastratio = 0;
for (int l = 0; l < Z; l++) {
  if (l != i) {
    if ((l < Z2 && i < Z2) || (l >= Z2 && i
        >= Z2))
      a = .25;
    else
      a = .5;
    rold = r_ij(R,l,i);
    rnew = r_ij(R_new,l,i);
    jastratio += a * (rnew/(1+beta*rnew) -
        rold/(1+beta*rold));
  }
}
jastratio = exp(jastratio);
```

Green's functions

```
//quantum force in new position
calcQF(R_new,F_new,alpha,beta,Z,DinvUp_new,
    DinvDown_new,Z2,detgrad_new,jastgrad_new)
    ;

//Ratio between Green's functions
greensratio = 0;
for (int ii = 0; ii < Z; ii++)
  for (int j = 0; j < 3; j++)
    greensratio += .5*(F_new[ii][j]+F[ii][j])
         * (.5*D*delta_t*(F[ii][j]-F_new[ii][
        j]) + R[ii][j] - R_new[ii][j]);
greensratio = exp(greensratio);
```

Metropolis Hastings test

```
//Metropolis-Hastings-test
if (ran2(&idum) < greensratio*detratio*
    detratio*jastratio*jastratio) {
  //Accept move abd update invers Slater
      matrix
  if (i < Z2)
    for (int l = 0; l < Z2; l++)
      for (int m = 0; m < Z2; m++)
        DinvUp[l][m] = DinvUp_new[l][m];
  else
    for (int l = 0; l < Z2; l++)
      for (int m = 0; m < Z2; m++)
        DinvDown[l][m] = DinvDown_new[l][m];
    //Update position, quantum force and
        gradients
    for (int ii = 0; ii < Z; ii++) {
      for (int j = 0; j < 3; j++) {
        R[ii][j] = R_new[ii][j];
        F[ii][j] = F_new[ii][j];
        detgrad[ii][j] = detgrad_new[ii][j];
        jastgrad[ii][j] = jastgrad_new[ii][j];
  ........
} //End loop of electron that has been moved
```

What is object orientation?

- Classes are new types of datatypes specialized for a task that may perform specific operations
- A good way to keep track of what needs to be shared between many functions (alternative: Global variables, massive argument lists)
- Makes debugging and code reuse simpler: Program is composed of several (mostly) independent self-contained pieces.

Use object orientation when:

- You can separate your code into logically separate sections
- When working on the same program for an extended period of time, or collaborating with many people
- When writing a big, complicated program, or etc...

Inheritance, what is it?

- You can make several new classes inherit old classes. They then get copies of the methods and variables in the parent class.
- In addition they may define their own methods and variables, or override methods in the base class.
- You may use a parent pointer to hold any child, while accessing functionality declared in parent.

Programming classes

In Fortran a vector or matrix start with 1, but it is easy to change a vector so that it starts with zero or even a negative number. If we have a double precision Fortran vector which starts at -10 and ends at 10, we could declare it as **REAL**(**KIND**=8):: vector($-$10:10). Similarly, if we want to start at zero and end at 10 we could write **REAL**(**KIND**=8):: vector(0:10). We have also seen that Fortran allows us to write a matrix addition $\mathbf{A} = \mathbf{B} + \mathbf{C}$ as A = B + C. This means that we have overloaded the addition operator so that it translates this operation into two loops and an addition of two matrix elements $a_{ij} = b_{ij} + c_{ij}$.

Programming classes

The way the matrix addition is written is very close to the way we express this relation mathematically. The benefit for the programmer is that our code is easier to read. Furthermore, such a way of coding makes it more likely to spot eventual errors as well. In Ansi C and C++ arrays start by default from $i = 0$. Moreover, if we wish to add two matrices we need to explicitely write out the two loops as

```
for(i=0 ; i < n ; i++) {
    for(j=0 ; j < n ; j++) {
        a[i][j]=b[i][j]+c[i][j]
    }
}
```

Programming classes

However, the strength of C++ over programming languages like C and Fortran 77 is the possibility to define new data types, tailored to some particular problem. Via new data types and overloading of operations such as addition and subtraction, we can easily define sets of operations and data types which allow us to write a matrix addition in exactly the same way as we would do in Fortran. We could also change the way we declare a C++ matrix elements a_{ij}, from $a[i][j]$ to say $a(i,j)$, as we would do in Fortran. Similarly, we could also change the default range from $0:n-1$ to $1:n$.

To achieve this we need to introduce two important entities in C++ programming,

classes and templates.

Programming classes

The function and class declarations are fundamental concepts within C++. Functions are abstractions which encapsulate an algorithm or parts of it and perform specific tasks in a program. We have already met several examples on how to use functions. Classes can be defined as abstractions which encapsulate data and operations on these data. The data can be very complex data structures and the class can contain particular functions which operate on these data. Classes allow therefore for a higher level of abstraction in computing. The elements (or components) of the data type are the class data members, and the procedures are the class member functions.

Programming classes

Classes are user-defined tools used to create multi-purpose software which can be reused by other classes or functions. These user-defined data types contain data (variables) and functions operating on the data.
A simple example is that of a point in two dimensions. The data could be the *x* and *y* coordinates of a given point. The functions we define could be simple read and write functions or the possibility to compute the distance between two points.

Programming classes

C++ has a class complex in its standard template library (STL). The standard usage in a given function could then look like

```
// Program to calculate addition and multiplication
     of two complex numbers
using namespace std;
#include <iostream>
#include <cmath>
#include <complex>
int main()
{
  complex<double> x(6.1,8.2), y(0.5,1.3);
  // write out x+y
  cout << x + y << x*y  << endl;
  return 0;
}
```

where we add and multiply two complex numbers $x = 6.1 + \imath 8.2$ and $y = 0.5 + \imath 1.3$ with the obvious results $z = x + y = 6.6 + \imath 9.5$ and $z = x \cdot y = -7.61 + \imath 12.03$.

Programming classes

We proceed by splitting our task in three files.
We define first a header file complex.h which contains the declarations of the class.
The header file contains the class declaration (data and functions), declaration of stand-alone functions, and all inlined functions, starting as follows

```
#ifndef Complex_H
#define Complex_H
//   various include statements and definitions
#include <iostream>            // Standard ANSI-C++
    include files
#include <new>
#include ....

class Complex
{...
definition of variables and their character
};
//   declarations of various functions used by the
    class
...
#endif
```

Programming classes

Next we provide a file complex.cpp where the code and algorithms of different functions (except inlined functions) declared within the class are written. The files complex.h and complex.cpp are normally placed in a directory with other classes and libraries we have defined.

Finally,we discuss here an example of a main program which uses this particular class. An example of a program which uses our complex class is given below. In particular we would like our class to perform tasks like declaring complex variables, writing out the real and imaginary part and performing algebraic operations such as adding or multiplying two complex numbers.

Programming classes

```
#include "Complex.h"
... other include and declarations
int main ()
{
  Complex a(0.1,1.3);     // we declare a complex
      variable a
  Complex b(3.0), c(5.0,-2.3);  // we declare
      complex variables b and c
  Complex d = b;          //  we declare  a new
      complex variable d
  cout << "d=" << d << ", a=" << a << ", b=" << b
      << endl;
  d = a*c + b/a;  //   we add, multiply and divide
      two complex numbers
  cout << "Re(d)=" << d.Re() << ", Im(d)=" << d.Im
      () << endl;  // write out of the real and
      imaginary parts
}
```

Programming classes

We include the header file complex.h and define four different complex variables. These are $a = 0.1 + \imath 1.3$, $b = 3.0 + \imath 0$ (note that if you don't define a value for the imaginary part this is set to zero), $c = 5.0 - \imath 2.3$ and $d = b$. Thereafter we have defined standard algebraic operations and the member functions of the class which allows us to print out the real and imaginary part of a given variable.

Programming classes

```
class Complex
{
private:
    double re, im; // real and imaginary part
public:
    Complex (); //
        Complex c;
    Complex (double re, double im = 0.0); //
        Definition of a complex variable;
    Complex (const Complex& c); //
        Usage: Complex c(a);   // equate two complex
         variables
    Complex& operator= (const Complex& c); // c = a;
            //  equate two complex variables, same as
         previous
....
```

Programming classes

```
    ~Complex () {}                                    //
       destructor
    double   Re () const;           // double real_part
         = a.Re();
    double   Im () const;           // double imag_part
         = a.Im();
    double   abs () const;          // double m = a.abs
        (); // modulus
    friend Complex operator+ (const Complex&  a,
        const Complex& b);
    friend Complex operator- (const Complex&  a,
        const Complex& b);
    friend Complex operator* (const Complex&  a,
        const Complex& b);
    friend Complex operator/ (const Complex&  a,
        const Complex& b);
};
```

Programming classes

The class is defined via the statement **class Complex.** We must first use the key word **class**, which in turn is followed by the user-defined variable name **Complex.** The body of the class, data and functions, is encapsulated within the parentheses $\{...\};$.

Programming classes

Data and specific functions can be private, which means that they cannot be accessed from outside the class. This means also that access cannot be inherited by other functions outside the class. If we use **protected** instead of **private**, then data and functions can be inherited outside the class.

Programming classes

The key word **public** means that data and functions can be accessed from outside the class. Here we have defined several functions which can be accessed by functions outside the class. The declaration **friend** means that stand-alone functions can work on privately declared variables of the type `(re, im)`. Data members of a class should be declared as private variables.

Programming classes

The first public function we encounter is a so-called constructor, which tells how we declare a variable of type **Complex** and how this variable is initialized. We have chose three possibilities in the example above:

- A declaration like **Complex** c; calls the member function **Complex**() which can have the following implementation

  ```
  Complex :: Complex () { re = im = 0.0; }
  ```

  meaning that it sets the real and imaginary parts to zero. Note the way a member function is defined. The constructor is the first function that is called when an object is instantiated.

Programming classes

- Another possibility is

  ```
  Complex::Complex()     {}
  ```

  which means that there is no initialization of the real and imaginary parts. The drawback is that a given compiler can then assign random values to a given variable.
- A call like **Complex** a(0.1,1.3); means that we could call the member function **Complex**(**double**, **double**)as

  ```
  Complex::Complex(double re_a, double im_a)
  { re = re_a; im = im_a; }
  ```

Programming classes

The simplest member function are those we defined to extract the real and imaginary part of a variable. Here you have to recall that these are private data, that is they invisible for users of the class. We obtain a copy of these variables by defining the functions

```
double Complex:: Re () const { return re; }} //
    getting the real part
double Complex:: Im () const { return im; }  //
    and the imaginary part
```

Note that we have introduced the declaration **const.** What does it mean? This declaration means that a variabale cannot be changed within a called function.

Programming classes

If we define a variable as **const double** p = 3; and then try to change its value, we will get an error when we compile our program. This means that constant arguments in functions cannot be changed.

```
// const arguments (in functions) cannot be changed
    :
void myfunc (const Complex& c)
{ c.re = 0.2; /* ILLEGAL!! compiler error... */  }
```

If we declare the function and try to change the value to 0.2, the compiler will complain by sending an error message.

Programming classes

If we define a function to compute the absolute value of complex variable like

```
double Complex:: abs ()  { return sqrt(re*re + im*
    im);}
```

without the constant declaration and define thereafter a function myabs as

```
double myabs (const Complex& c)
{ return c.abs(); }    // Not ok because c.abs() is
    not a const func.
```

the compiler would not allow the c.abs() call in myabs since **Complex**::abs is not a constant member function.

Programming classes

Constant functions cannot change the object's state. To avoid this we declare the function abs as

```
double Complex:: abs () const { return sqrt(re*re +
      im*im); }
```

Programming classes

C++ (and Fortran) allow for overloading of operators. That means we can define algebraic operations on for example vectors or any arbitrary object. As an example, a vector addition of the type $\mathbf{c} = \mathbf{a} + \mathbf{b}$ means that we need to write a small part of code with a for-loop over the dimension of the array. We would rather like to write this statement as `c = a+b;` as this makes the code much more readable and close to eventual equations we want to code. To achieve this we need to extend the definition of operators.

Programming classes

Let us study the declarations in our complex class. In our main function we have a statement like d = b;, which means that we call d.**operator**= (b) and we have defined a so-called assignment operator as a part of the class defined as

```
Complex& Complex:: operator= (const Complex& c)
{
   re = c.re;
   im = c.im;
   return *this;
}
```

Programming classes

With this function, statements like **Complex** d = b; or **Complex** d(b); make a new object *d*, which becomes a copy of *b*. We can make simple implementations in terms of the assignment

```
Complex::  Complex  (const  Complex& c)
{ *this = c; }
```

which is a pointer to ”this object”, ∗**this** is the present object, so ∗**this** = c; means setting the present object equal to *c*, that is **this**−>**operator**= (c);.

Programming classes

The meaning of the addition operator $+$ for Complex objects is defined in the function **Complex operator**+ (**const Complex**& a, **const Complex**& b); *//a+b* The compiler translates c = a + b; into c = **operator**+ (a, b);. Since this implies the call to function, it brings in an additional overhead. If speed is crucial and this function call is performed inside a loop, then it is more difficult for a given compiler to perform optimizations of a loop.

Programming classes

The solution to this is to inline functions. We discussed inlining in chapter 2 of the lecture notes. Inlining means that the function body is copied directly into the calling code, thus avoiding calling the function. Inlining is enabled by the inline keyword

```
inline Complex operator+ (const Complex& a, const
    Complex& b)
{ return Complex (a.re + b.re, a.im + b.im); }
```

Inline functions, with complete bodies must be written in the header file complex.h.

Programming classes

Consider the case c = a + b; that is,
c.**operator**= (**operator**+ (a,b)); If **operator**+, **operator**= and the constructor **Complex**(r,i) all are inline functions, this transforms to

```
c.re = a.re + b.re;
c.im = a.im + b.im;
```

by the compiler, i.e., no function calls

Programming classes

The stand-alone function **operator+** is a friend of the Complex class

```
class Complex
{
    ...
    friend Complex operator+ (const Complex& a,
        const Complex& b);
    ...
};
```

so it can read (and manipulate) the private data parts *re* and *im* via

```
inline Complex operator+ (const Complex& a, const
    Complex& b)
{ return Complex (a.re + b.re, a.im + b.im); }
```

Programming classes

Since we do not need to alter the re and im variables, we can get the values by Re() and Im(), and there is no need to be a friend function

```
inline Complex operator+ (const Complex& a, const
    Complex& b)
{ return Complex (a.Re() + b.Re(), a.Im() + b.Im())
    ; }
```

Programming classes

The multiplication functionality can now be extended to imaginary numbers by the following code

```
inline Complex operator* (const Complex& a, const
    Complex& b)
{
  return Complex(a.re*b.re - a.im*b.im, a.im*b.re +
      a.re*b.im);
}
```

It will be convenient to inline all functions used by this operator.

Programming classes

To inline the complete expression a∗b;, the constructors and **operator**= must also be inlined. This can be achieved via the following piece of code

```
inline Complex:: Complex () { re = im = 0.0; }
inline Complex:: Complex (double re_, double im_)
{ ... }
inline Complex:: Complex (const Complex& c)
{ ... }
inline Complex:: operator= (const Complex& c)
{ ... }
```

Programming classes

```
// e, c, d are complex
e = c*d;
// first compiler translation:
e.operator= (operator* (c,d));
// result of nested inline functions
// operator=, operator*, Complex(double,double=0):
e.re = c.re*d.re - c.im*d.im;
e.im = c.im*d.re + c.re*d.im;
```

The definitions **operator**− and **operator**/ follow the same set up.

Programming classes

Finally, if we wish to write to file or another device a complex number using the simple syntax cout << c;, we obtain this by defining the effect of << for a Complex object as

```
ostream& operator<< (ostream& o, const Complex& c)
{ o << "(" << c.Re() << "," << c.Im() << ") ";
    return o;}
```

Programming classes, templates

What if we wanted to make a class which takes integers or floating point numbers with single precision? A simple way to achieve this is copy and paste our class and replace **double** with for example **int**.
C++ allows us to do this automatically via the usage of templates, which are the C++ constructs for parameterizing parts of classes. Class templates is a template for producing classes. The declaration consists of the keyword **template** followed by a list of template arguments enclosed in brackets.

Programming classes

We can therefore make a more general class by rewriting our original example as

```
template<class T>
class Complex
{
private:
   T re, im; // real and imaginary part
public:
   Complex (); //
      Complex c;
   Complex (T re, T im = 0); // Definition of a
      complex variable;
   Complex (const Complex& c); //
      Usage: Complex c(a);    // equate two complex
       variables
   Complex& operator= (const Complex& c); // c = a;
         //  equate two complex variables, same as
       previous
```

Programming classes

We can therefore make a more general class by rewriting our original example as

```
  ~Complex () {}                              //
     destructor
   T    Re () const;          // T real_part = a.Re();
   T    Im () const;          // T imag_part = a.Im();
   T    abs () const;         // T m = a.abs(); //
      modulus
   friend Complex operator+ (const Complex&  a,
      const Complex& b);
   friend Complex operator− (const Complex&  a,
      const Complex& b);
   friend Complex operator∗ (const Complex&  a,
      const Complex& b);
   friend Complex operator/ (const Complex&  a,
      const Complex& b);
};
```

Programming classes

What it says is that **Complex** is a parameterized type with *T* as a parameter and *T* has to be a type such as double or float. The class complex is now a class template and we would define variables in a code as

```
Complex<double> a(10.0,5.1);
Complex<int> b(1,0);
```

Programming classes

Member functions of our class are defined by preceding the name of the function with the **template** keyword. Consider the function we defined as
Complex:: **Complex** (**double** re_a, **double** im_a). We would rewrite this function as

```
template<class T>
Complex<T>:: Complex (T re_a, T im_a)
{ re = re_a; im = im_a; }
```

The member functions are otherwise defined following ordinary member function definitions.

Topics for Week 12, March 22-26

Slater determinants and classes

- Repetition from last week
- Discussion of classes, the complex class and a matrix-vector class

Project work this week: Slater determinant!

Useful equations

The $1s$ hydrogen like wave function

$$R_{10}(r) = 2\left(\frac{Z}{a_0}\right)^{3/2}\exp\left(-Zr/a_0\right) = u_{10}/r$$

The total energy for helium (not the Hartree or Fock terms) from the direct and the exchange term should give $5Z/8$.
The single-particle energy with no interactions should give $-Z^2/2n^2$.
The $2s$ hydrogen-like wave function is

$$R_{20}(r) = 2\left(\frac{Z}{2a_0}\right)^{3/2}\left(1-\frac{Zr}{2a_0}\right)\exp\left(-Zr/2a_0\right) = u_{20}/r$$

and the $2p$ hydrogen -like wave function is

$$R_{21}(r) = \frac{1}{\sqrt{3}}\left(\frac{Z}{2a_0}\right)^{3/2}\frac{Zr}{a_0}\exp\left(-Zr/2a_0\right) = u_{21}/r$$

We use $a_0 = 1$.

Topics for Week 14, April 5-9

Hartree-Fock theory and Density functional theory

- Discussion of project 2, Hartree-Fock theory, wave functions and computation of Coulomb matrix elements.
- Project 1: Continuation on how to implement the Slater determinant and correlation part and the conjugate gradient method.
- Discussion of practicalities about the Hartree-Fock equations.
- Formal derivation of the Hartree-Fock equations. Continues next two weeks as well.

The material discussed for the rest of the course is covered by Thijssen's book chapters 4-6. We will use April to cover Hartree-Fock theory and Density Functional theory. May is devoted to finalizing project 2 only. Only Lab work during May.

Structure of project 2

Project 2 deals with the following topics:

- Hartree-Fock calculation of the Beryllium atom. To do that one needs
    1. Choice of basis, discussed today: Slater orbitals and Hydrogen-like orbitals.
    2. Diagonalization of an eigenvalue problem in order to find the coefficients.
    3. Computation of the Coulomb matrix elements. Since we limit ourselves to Beryllium, we need only $l = 0$ waves.

Structure of project 2

Finally:

- The Hartree-Fock solutions are in turn used in the Variational Monte Carlo code developed in project 1. That means that we do not need vary α. The variational calculation is then limited to the Jastrow factor. We will look at different Jastrow factors and see which gives the best solution.

Structure of project 2

The structure of the Hartree-Fock part involves

1. Choice of basis, discussed today: Slater orbitals and Hydrogen-like orbitals. Here we need a function to compute the Laguerre polynomials for Hydrogen-like orbitals. This function is available at the webpage as laguerre.cpp, see under project 2.
2. Diagonalization of an eigenvalue problem in order to find the coefficients. One can use Jacobi's method or Householder's with Givens' transformations, see chapter 12 of lecture notes.
3. Computation of the Coulomb matrix elements. Since we limit ourselves to Beryllium, we need only $l = 0$ waves. Here you need to develop a program which sets up the matrix elements using Gaussian quadrature (chapter 7 of lecture notes).

Finding the Hartree-Fock functional $E[\Phi]$

We rewrite our Hamiltonian

$$\hat{H} = -\sum_{i=1}^{N} \frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N} \frac{Z}{r_i} + \sum_{i<j}^{N} \frac{1}{r_{ij}},$$

as

$$\hat{H} = \hat{H}_0 + \hat{H}_1 = \sum_{i=1}^{N} \hat{h}_i + \sum_{i<j=1}^{N} \frac{1}{r_{ij}},$$

$$\hat{h}_i = -\frac{1}{2}\nabla_i^2 - \frac{Z}{r_i}.$$

Finding the Hartree-Fock functional $E[\Phi]$

Let us denote the ground state energy by E_0. According to the variational principle we have

$$E_0 \leq E[\Phi] = \int \Phi^* \hat{H} \Phi d\tau$$

where Φ is a trial function which we assume to be normalized

$$\int \Phi^* \Phi d\tau = 1,$$

where we have used the shorthand $d\tau = d\mathbf{r}_1 d\mathbf{r}_2 \ldots d\mathbf{r}_N$.

Finding the Hartree-Fock functional $E[\Phi]$

In the Hartree-Fock method the trial function is the Slater determinant which can be rewritten as

$$\Psi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N, \alpha, \beta, \dots, \nu) = \frac{1}{\sqrt{N!}} \sum_P (-)^P P \psi_\alpha(\mathbf{r}_1) \psi_\beta(\mathbf{r}_2) \dots \psi_\nu(\mathbf{r}_N) = \sqrt{N!} \mathcal{A} \Phi_H,$$

where we have introduced the anti-symmetrization operator $\mathcal{A}$ defined by the summation over all possible permutations of two eletrons. It is defined as

$$\mathcal{A} = \frac{1}{N!} \sum_P (-)^P P,$$

with the the Hartree-function given by the simple product of all possible single-particle function (two for helium, four for beryllium and ten for neon)

$$\Phi_H(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N, \alpha, \beta, \dots, \nu) = \psi_\alpha(\mathbf{r}_1) \psi_\beta(\mathbf{r}_2) \dots \psi_\nu(\mathbf{r}_N).$$

Finding the Hartree-Fock functional $E[\Phi]$

Both $\hat{H}_1$ and $\hat{H}_2$ are invariant under electron permutations, and hence commute with $\mathcal{A}$

$$[H_0, \mathcal{A}] = [H_1, \mathcal{A}] = 0.$$

Furthermore, $\mathcal{A}$ satisfies

$$\mathcal{A}^2 = \mathcal{A},$$

since every permutation of the Slater determinant reproduces it.

Finding the Hartree-Fock functional $E[\Phi]$

The expectation value of $\hat{H}_1$

$$\int \Phi^* \hat{H}_0 \Phi d\tau = N! \int \Phi_H^* \mathcal{A} \hat{H}_0 \mathcal{A} \Phi_H d\tau$$

is readily reduced to

$$\int \Phi^* \hat{H}_0 \Phi d\tau = N! \int \Phi_H^* \hat{H}_0 \mathcal{A} \Phi_H d\tau,$$

which can be rewritten as

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{i=1}^{N} \sum_{P} (-)^P \int \Phi_H^* \hat{h}_i P \Phi_H d\tau.$$

Finding the Hartree-Fock functional $E[\Phi]$

The integral vanishes if two or more electrons are permuted in only one of the Hartree-functions Φ_H because the individual orbitals are orthogonal. We obtain then

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{i=1}^{N} \int \Phi_H^* \hat{h}_i \Phi_H d\tau.$$

Orthogonality allows us to further simplify the integral, and we arrive at the following expression for the expectation values of the sum of one-body Hamiltonians

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{\mu=1}^{N} \int \psi_\mu^*(\mathbf{r}_i) \hat{h}_i \psi_\mu(\mathbf{r}_i) d\mathbf{r}_i,$$

or just as

$$\int \Phi^* \hat{H}_0 \Phi d\tau = \sum_{\mu=1}^{N} \langle \mu | h | \mu \rangle.$$

Finding the Hartree-Fock functional $E[\Phi]$

The expectation value of the two-body Hamiltonian is obtained in a similar manner. We have

$$\int \Phi^* \hat{H}_1 \Phi d\tau = N! \int \Phi_H^* \mathcal{A} \hat{H}_1 \mathcal{A} \Phi_H d\tau,$$

which reduces to

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{i \le j=1}^{N} \sum_{P} (-)^P \int \Phi_H^* \frac{1}{r_{ij}} P \Phi_H d\tau,$$

by following the same arguments as for the one-body Hamiltonian. Because of the dependence on the inter-electronic distance $1/r_{ij}$, permutations of two electrons no longer vanish, and we get

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{i < j=1}^{N} \int \Phi_H^* \frac{1}{r_{ij}} (1 - P_{ij}) \Phi_H d\tau.$$

where P_{ij} is the permutation operator that interchanges electrons i and j.

Finding the Hartree-Fock functional $E[\Phi]$

We use the assumption that the orbitals are orthogonal, and obtain

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\mu(\mathbf{r}_i)\psi_\nu(\mathbf{r}_j)d\mathbf{r}_i d\mathbf{r}_j\right.$$

$$\left. -\int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\mu(\mathbf{r}_j)\psi_\nu(\mathbf{r}_i)d\mathbf{r_i}d\mathbf{r_j}\right].$$

The first term is the so-called direct term or Hartree term, while the second is due to the Pauli principle and is called exchange term or Fock term. The factor 1/2 is introduced because we now run over all pairs twice.
The compact notation is

$$\frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\langle\mu\nu|\frac{1}{r_{ij}}|\mu\nu\rangle - \langle\mu\nu|\frac{1}{r_{ij}}|\nu\mu\rangle\right].$$

Variational Calculus and Lagrangian Multiplier, back to Hartree-Fock

Our functional is written as

$$E[\Phi] = \sum_{\mu=1}^{N} \int \psi_{\mu}^{*}(\mathbf{r}_i)\hat{h}_i\psi_{\mu}(\mathbf{r}_i)d\mathbf{r}_i + \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\int \psi_{\mu}^{*}(\mathbf{r}_i)\psi_{\nu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\mu}(\mathbf{r}_i)\psi_{\nu}(\mathbf{r}_j)d\mathbf{r}_i d\mathbf{r}_j \right.$$

$$\left. - \int \psi_{\mu}^{*}(\mathbf{r}_i)\psi_{\nu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\mu}(\mathbf{r}_j)\psi_{\nu}(\mathbf{r}_i)d\mathbf{r}_i d\mathbf{r}_j\right]$$

The more compact version is

$$E[\Phi] = \sum_{\mu=1}^{N}\langle\mu|h|\mu\rangle + \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\langle\mu\nu|\frac{1}{r_{ij}}|\mu\nu\rangle - \langle\mu\nu|\frac{1}{r_{ij}}|\nu\mu\rangle\right].$$

Variational Strategies

With the given functional, we can perform at least two types of variational strategies.

- Vary the Slater determinant by changing the spatial part of the single-particle wave functions themselves. This is what we will do.
- Expand the single-particle functions in a known basis and vary the coefficients, that is, the new function single-particle wave function $|a\rangle$ is written as a linear expansion in terms of a fixed basis (harmonic oscillator, Laguerre polynomials etc)
$$\psi_a = \sum_{\lambda} C_{a\lambda}\psi_{\lambda},$$

Both cases lead to a new Slater determinant which is related to the previous via a unitary transformation.

Small exercise

1. Consider a Slater determinant built up of single-particle orbitals ψ_λ, with $\lambda = 1, 2, \dots, N$.

   The unitary transformation
$$\psi_a = \sum_\lambda C_{a\lambda}\psi_\lambda,$$
   brings us into the new basis. Show that the new basis is orthonormal.

2. Show that the new Slater determinant constructed from the new single-particle wave functions can be written as the determinant based on the previous basis and the determinant of the matrix C.

3. Show that the old and the new Slater determinants are equal up to a complex constant with absolute value unity. (Hint, C is a unitary matrix).

Hartree-Fock: useful expressions for the Coulomb terms

We need to compute the integral for the direct term

$$\langle\alpha\beta|V|\gamma\delta\rangle = \int\int \psi^*_\alpha(\mathbf{r}_1)\psi^*_\beta(\mathbf{r}_2)\frac{1}{r_{12}}\psi_\gamma(\mathbf{r}_1)\psi_\delta(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2$$

and the exchange term

$$\langle\alpha\beta|V|\delta\gamma\rangle = \int\int \psi^*_\alpha(\mathbf{r}_1)\psi^*_\beta(\mathbf{r}_2)\frac{1}{r_{12}}\psi_\delta(\mathbf{r}_1)\psi_\gamma(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2$$

Note well that spin is included in the quantum numbers $\alpha\beta\gamma\delta$, but the interaction does not affect spin.

Hartree-Fock: Expression for Direct Term

We need to break down the single-particle wave functions in radial, angular and spin parts. Note that direct term is diagonal in the spin quantum numbers. The single-particle wave functions are

$$\psi_\alpha(\mathbf{r}) = \psi_{nlm_l sm_s}(\mathbf{r}) = \phi_{nlm_l}(\mathbf{r})\xi_{m_s}(s)$$

with

$$\phi_{nlm_l}(\mathbf{r}) = R_{nl}(r)Y_{lm_l}(\hat{r})$$

Hartree-Fock: Explicit expression for Direct Term

The addition theorem of the spherical harmonics yields

$$\sum_{m_l=-l}^{l} Y^*_{lm_l}(\theta_i,\phi_i)Y_{lm_l}(\theta_j,\phi_j) = \frac{2l+1}{4\pi}P_l(\cos(\theta))$$

with

$$cos(\theta) = cos(\theta_i)cos(\theta_j) + sin(\theta_i)sin(\theta_j)cos(\phi_i - \phi_j).$$

The quantity r_{ij} in the Coulomb interaction depends on the angles of particle i and j.

Hartree-Fock: Explicit expression for Direct Term

The integral over the angles can be performed by expanding $1/r_{ij}$ in terms of spherical harmonics and using the fact that the functions R_{nl} do not depend on the angles. We have

$$\frac{1}{r_{ij}} = \frac{1}{r_i}\sum_{l=0}^{\infty}\left(\frac{r_j}{r_i}\right)^l P_l\cos(\theta)),$$

if $r_i > r_j$ or

$$\frac{1}{r_{ij}} = \frac{1}{r_j}\sum_{l=0}^{\infty}\left(\frac{r_i}{r_j}\right)^l P_l\cos(\theta)),$$

if $r_j > r_i$. In a compact form it reads

$$\frac{1}{r_{ij}} = \sum_{l=0}^{\infty}\left(\frac{(r_<)^l}{(r_>)^{l+1}}\right) P_l\cos(\theta)),$$

with $r_> = max(r_i, r_j)$ and with $r_< = min(r_i, r_j)$. P_l is the Legendre polynomial and

$$\cos(\theta) = \cos(\theta_i)\cos(\theta_j) + \sin(\theta_i)\sin(\theta_j)\cos(\phi_i - \phi_j).$$

Hartree-Fock: Explicit expression for Direct Term

We can use the expression for spherical harmonics to express the interaction as

$$\frac{1}{r_{ij}} = \sum_{l=0}^{\infty} \sum_{m_l=-l}^{l} \frac{4\pi}{2l+1} \left(\frac{(r_<)^l}{(r_>)^{l+1}} \right) Y^*_{lm_l}(\theta_i, \phi_i) Y_{lm_l}(\theta_j, \phi_j)$$

and inserting it we obtain for the direct term

$$\sum_{l=0}^{\infty} \sum_{m_l=-l}^{l} \frac{4\pi}{2l+1} \int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha l_\alpha}(r_1) R^*_{n_\beta l_\beta}(r_2) \frac{(r_<)^l}{(r_>)^{l+1}} R_{n_\gamma l_\gamma}(r_1) R_{n_\delta l_\delta}(r_2)$$

$$\times \int d\Omega_1 Y^*_{lm_l}(\theta_1, \phi_1) Y^*_{l_\alpha m_\alpha}(\theta_1, \phi_1) Y_{l_\gamma m_\gamma}(\theta_1, \phi_1)$$

$$\times \int d\Omega_2 Y_{lm_l}(\theta_2, \phi_2) Y^*_{l_\beta m_\beta}(\theta_2, \phi_2) Y_{l_\delta m_\delta}(\theta_2, \phi_2)$$

Hartree-Fock: Explicit expression for Exchange term

Similarly, we get for the exchange term the following expression

$$\sum_{l=0}^{\infty}\sum_{m_l=-l}^{l}\frac{4\pi}{2l+1}\int r_1^2dr_1\int r_2^2dr_2 R^*_{n_\alpha l_\alpha}(r_1)R^*_{n_\beta l_\beta}(r_2)\frac{(r_<)^l}{(r_>)^{l+1}}R_{n_\gamma l_\gamma}(r_2)R_{n_\delta l_\delta}(r_1)$$

$$\times\int d\Omega_1 Y^*_{lm_l}(\theta_1,\phi_1)Y^*_{l_\alpha m_\alpha}(\theta_1,\phi_1)Y_{l_\delta m_\delta}(\theta_1,\phi_1)$$

$$\times\int d\Omega_2 Y_{lm_l}(\theta_2,\phi_2)Y^*_{l_\beta m_\beta}(\theta_2,\phi_2)Y_{l_\gamma m_\gamma}(\theta_2,\phi_2)$$

Hartree-Fock: Explicit expression for Direct Term of Beryllium

In our case we will only deal with particles with $l_\alpha = 0$. In this case we have that

$$Y_{00} = \frac{1}{\sqrt{4\pi}},$$

and inserting it we get

$$\sum_{l=0}^{\infty}\sum_{m_l=-l}^{l}\frac{4\pi}{2l+1}\int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha 0}(r_1)R^*_{n_\beta 0}(r_2)\frac{(r_<)^l}{(r_>)^{l+1}}R_{n_\gamma 0}(r_1)R_{n_\delta 0}(r_2)$$

$$\times \int d\Omega_1 Y^*_{lm_l}(\theta_1,\phi_1)Y^*_{00}Y_{00}\int d\Omega_2 Y_{lm_l}(\theta_2,\phi_2)Y^*_{00}Y_{00}$$

which becomes

$$\sum_{l=0}^{\infty}\sum_{m_l=-l}^{l}\frac{1}{2l+1}\int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha 0}(r_1)R^*_{n_\beta 0}(r_2)\frac{(r_<)^l}{(r_>)^{l+1}}R_{n_\gamma 0}(r_1)R_{n_\delta 0}(r_2)$$

$$\times \int d\Omega_1 Y^*_{lm_l}(\theta_1,\phi_1)Y_{00}\int d\Omega_2 Y_{lm_l}(\theta_2,\phi_2)Y_{00}$$

Hartree-Fock: Explicit expression for Direct Term of Beryllium

Using the orthogonality relation of spherical harmonics

$$\int d\Omega_1 Y^*_{l' m'_l}(\theta_1,\phi_1) Y_{lm_l}(\theta_1,\phi_1) = \delta_{l,l'}\delta_{m_l,m'_l},$$

we can rewrite

$$\sum_{l=0}^{\infty}\sum_{m_l=-l}^{l}\frac{1}{2l+1}\int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha 0}(r_1)R^*_{n_\beta 0}(r_2)\frac{(r_<)^l}{(r_>)^{l+1}}R_{n_\gamma 0}(r_1)R_{n_\delta 0}(r_2)$$

$$\times \int d\Omega_1 Y^*_{lm_l}(\theta_1,\phi_1)Y_{00}\int d\Omega_2 Y_{lm_l}(\theta_2,\phi_2)Y_{00}$$

as

$$\sum_{l=0}^{\infty}\sum_{m_l=-l}^{l}\frac{1}{2l+1}\int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha 0}(r_1)R^*_{n_\beta 0}(r_2)\frac{(r_<)^l}{(r_>)^{l+1}}R_{n_\gamma 0}(r_1)R_{n_\delta 0}(r_2)\delta_{l,0}\delta_{m_l,0},$$

resulting in

$$\int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha 0}(r_1)R^*_{n_\beta 0}(r_2)\frac{1}{(r_>)}R_{n_\gamma 0}(r_1)R_{n_\delta 0}(r_2)$$

Hartree-Fock: Explicit expression for Direct Term of Beryllium

The direct matrix elements that we need are then simply given by a double integral

$$\int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha 0}(r_1) R^*_{n_\beta 0}(r_2) \frac{1}{(r_>)} R_{n_\gamma 0}(r_1) R_{n_\delta 0}(r_2)$$

- This integral can be solved once and for all using hydrogenic-like wave functions by solving the above double integral using Gaussian quadrature (see chapter 7 of lecture notes).
- These elements should then be stored and looked up every time they are needed in the Hartree-Fock calculation.
- Exercise: find the corresponding matrix element for the exchange part.

Hartree-Fock: Simplification of the direct term

As an exercise, show that you can simplify the integral

$$\int r_1^2 dr_1 \int r_2^2 dr_2 R^*_{n_\alpha 0}(r_1) R^*_{n_\beta 0}(r_2) \frac{1}{(r_>)} R_{n_\gamma 0}(r_1) R_{n_\delta 0}(r_2)$$

as

$$\int_0^\infty r_1^2 dr_1 R^*_{n_\alpha 0}(r_1) R_{n_\gamma 0}(r_1) \left[\frac{1}{(r_1)} \int_0^{r_1} r_2^2 dr_2 R^*_{n_\beta 0}(r_2) R_{n_\delta 0}(r_2) + \int_{r_1}^\infty r_2 dr_2 R^*_{n_\beta 0}(r_2) R_{n_\delta 0}(r_2) \right].$$

Hartree-Fock: Single-particle wave functions

As basis functions for our calculations we will use hydrogenic like functions. In project 2 we need only the radial part since the spherical harmonics for *s*-waves are rather simple. Our radial wave functions are

$$R_{n0}(r) = \left(\frac{2Z}{n}\right)^{3/2}\sqrt{\frac{(n-1)!}{2n\times n!}}L^1_{n-1}(\frac{2Z}{n})\exp{(-\frac{Zr}{n})},$$

with energies $-Z^2/2n^2$. A function for computing the generalized Laguerre polynomials $L^1_{n-1}(\frac{2Z}{n})$ is provided at the webpage of the course under the link of project 2. We will use these functions to solve the Hartree-Fock problem for Beryllium.

Hartree-Fock: Single-particle wave functions

When we have finished the Hartree-Fock calculations, it will be useful to parameterize our solutions in terms of the nodeless Slater-type orbitals (STO). In our case we will use so-called node-dependent solutions given by

$$\Psi_{nlm_l}(r,\theta,\phi) = \mathcal{N} r^{n_{eff}-1} e^{\frac{Z_{eff}\rho}{n_{eff}}} Y_{lm_l}(\theta,\phi).$$

Here $\mathcal{N}$ is a normalization constant, Y_{lm_l} is a spherical harmonic and $\rho = r/a_0$. Such parameterizations exit in the literature. For Beryllium we can use

$$R_{10}^{\mathrm{STO}}(r) = N_{10} \exp\left(-3.6848r\right)$$

and

$$R_{20}^{\mathrm{STO}}(r) = N_{20} r \exp\left(-1.9120r/2\right)$$

Using these values in the Slater determinant for Beryllium, with no Jastrow factor you should get -14.573 a.u. for the ground state energy.

Intermezzo: Variational Calculus and Lagrangian Multiplier

The calculus of variations involves problems where the quantity to be minimized or maximized is an integral.
In the general case we have an integral of the type

$$E[\Phi] = \int_a^b f(\Phi(x), \frac{\partial \Phi}{\partial x}, x) dx,$$

where E is the quantity which is sought minimized or maximized. The problem is that although f is a function of the variables Φ, $\partial\Phi/\partial x$ and x, the exact dependence of Φ on x is not known. This means again that even though the integral has fixed limits a and b, the path of integration is not known. In our case the unknown quantities are the single-particle wave functions and we wish to choose an integration path which makes the functional $E[\Phi]$ stationary. This means that we want to find minima, or maxima or saddle points. In physics we search normally for minima. Our task is therefore to find the minimum of $E[\Phi]$ so that its variation δE is zero subject to specific constraints. In our case the constraints appear as the integral which expresses the orthogonality of the single-particle wave functions. The constraints can be treated via the technique of Lagrangian multipliers

Euler-Lagrange equations

We assume the existence of an optimum path, that is a path for which $E[\Phi]$ is stationary. There are infinitely many such paths. The difference between two paths $\delta\Phi$ is called the variation of Φ.
We call the variation $\eta(x)$ and it is scaled by a factor α. The function $\eta(x)$ is arbitrary except for

$$\eta(a) = \eta(b) = 0,$$

and we assume that we can model the change in Φ as

$$\Phi(x, \alpha) = \Phi(x, 0) + \alpha\eta(x),$$

and

$$\delta\Phi = \Phi(x, \alpha) - \Phi(x, 0) = \alpha\eta(x).$$

Euler-Lagrange equations

We choose $\Phi(x, \alpha = 0)$ as the unkonwn path that will minimize E. The value $\Phi(x, \alpha \neq 0)$ describes a neighbouring path.
We have

$$E[\Phi(\alpha)] = \int_a^b f(\Phi(x, \alpha), \frac{\partial \Phi(x, \alpha)}{\partial x}, x) dx.$$

In the slides I will use the shorthand

$$\Phi_x(x, \alpha) = \frac{\partial \Phi(x, \alpha)}{\partial x}.$$

In our case $a = 0$ and $b = \infty$ and we know the value of the wave function.

Euler-Lagrange equations

The condition for an extreme of

$$E[\Phi(\alpha)] = \int_a^b f(\Phi(x,\alpha), \Phi_x(x,\alpha), x)dx,$$

is

$$\left[\frac{\partial E[\Phi(\alpha)]}{\partial x}\right]_{\alpha=0} = 0.$$

The α dependence is contained in $\Phi(x,\alpha)$ and $\Phi_x(x,\alpha)$ meaning that

$$\left[\frac{\partial E[\Phi(\alpha)]}{\partial \alpha}\right] = \int_a^b \left(\frac{\partial f}{\partial \Phi}\frac{\partial \Phi}{\partial \alpha} + \frac{\partial f}{\partial \Phi_x}\frac{\partial \Phi_x}{\partial \alpha}\right) dx.$$

We have defined

$$\frac{\partial \Phi(x,\alpha)}{\partial \alpha} = \eta(x)$$

and thereby

$$\frac{\partial \Phi_x(x,\alpha)}{\partial \alpha} = \frac{d(\eta(x))}{dx}.$$

Euler-Lagrange equations

Using

$$\frac{\partial \Phi(x,\alpha)}{\partial \alpha} = \eta(x),$$

and

$$\frac{\partial \Phi_x(x,\alpha)}{\partial \alpha} = \frac{d(\eta(x))}{dx},$$

in the integral gives

$$\left[\frac{\partial E[\Phi(\alpha)]}{\partial \alpha}\right] = \int_a^b \left(\frac{\partial f}{\partial \Phi}\eta(x) + \frac{\partial f}{\partial \Phi_x}\frac{d(\eta(x))}{dx}\right) dx.$$

Integrate the second term by parts

$$\int_a^b \frac{\partial f}{\partial \Phi_x}\frac{d(\eta(x))}{dx} dx = \eta(x)\frac{\partial f}{\partial \Phi_x}|_a^b - \int_a^b \eta(x)\frac{d}{dx}\frac{\partial f}{\partial \Phi_x} dx,$$

and since the first term dissappears due to $\eta(a) = \eta(b) = 0$, we obtain

$$\left[\frac{\partial E[\Phi(\alpha)]}{\partial \alpha}\right] = \int_a^b \left(\frac{\partial f}{\partial \Phi} - \frac{d}{dx}\frac{\partial f}{\partial \Phi_x}\right)\eta(x) dx = 0.$$

Euler-Lagrange equations

$$\left[\frac{\partial E[\Phi(\alpha)]}{\partial\alpha}\right] = \int_a^b \left(\frac{\partial f}{\partial\Phi} - \frac{d}{dx}\frac{\partial f}{\partial\Phi_x}\right)\eta(x)dx = 0,$$

can also be written as

$$\alpha\left[\frac{\partial E[\Phi(\alpha)]}{\partial\alpha}\right]_{\alpha=0} = \int_a^b \left(\frac{\partial f}{\partial\Phi} - \frac{d}{dx}\frac{\partial f}{\partial\Phi_x}\right)\delta\Phi(x)dx = \delta E = 0.$$

The condition for a stationary value is thus a partial differential equation

$$\frac{\partial f}{\partial\Phi} - \frac{d}{dx}\frac{\partial f}{\partial\Phi_x} = 0,$$

known as Euler's equation. Can easily be generalized to more variables.

Lagrangian Multipliers

Consider a function of three independent variables $f(x, y, z)$. For the function f to be an extreme we have

$$df = 0.$$

A necessary and sufficient condition is

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} = 0,$$

due to

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz.$$

In physical problems the variables x, y, z are often subject to constraints (in our case Φ and the orthogonality constraint) so that they are no longer all independent. It is possible at least in principle to use each constraint to eliminate one variable and to proceed with a new and smaller set of independent varables.

Lagrangian Multipliers

The use of so-called Lagrangian multipliers is an alternative technique when the elimination of of variables is incovenient or undesirable. Assume that we have an equation of constraint on the variables x, y, z

$$\phi(x, y, z) = 0,$$

resulting in

$$d\phi = \frac{\partial \phi}{\partial x}dx + \frac{\partial \phi}{\partial y}dy + \frac{\partial \phi}{\partial z}dz = 0.$$

Now we cannot set anymore

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} = 0,$$

if $df = 0$ is wanted because there are now only two independent variables! Assume x and y are the independent variables. Then dz is no longer arbitrary.

Lagrangian Multipliers

However, we can add to

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz,$$

a multiplum of $d\phi$, viz. $\lambda d\phi$, resulting in

$$df + \lambda d\phi = (\frac{\partial f}{\partial z} + \lambda\frac{\partial \phi}{\partial x})dx + (\frac{\partial f}{\partial y} + \lambda\frac{\partial \phi}{\partial y})dy + (\frac{\partial f}{\partial z} + \lambda\frac{\partial \phi}{\partial z})dz = 0.$$

Our multiplier is chosen so that

$$\frac{\partial f}{\partial z} + \lambda\frac{\partial \phi}{\partial z} = 0.$$

Lagrangian Multipliers

However, we took *dx* and *dy* as to be arbitrary and thus we must have

$$\frac{\partial f}{\partial x} + \lambda \frac{\partial \phi}{\partial x} = 0,$$

and

$$\frac{\partial f}{\partial y} + \lambda \frac{\partial \phi}{\partial y} = 0.$$

When all these equations are satisfied, $df = 0$. We have four unknowns, x, y, z and λ. Actually we want only x, y, z, λ need not to be determined, it is therefore often called Lagrange's undetermined multiplier. If we have a set of constraints ϕ_k we have the equations

$$\frac{\partial f}{\partial x_i} + \sum_k \lambda_k \frac{\partial \phi_k}{\partial x_i} = 0.$$

Variational Calculus and Lagrangian Multipliers

Let us specialize to the expectation value of the energy for one particle in three-dimensions. This expectation value reads

$$E = \int dxdydz \psi^*(x,y,z)\hat{H}\psi(x,y,z),$$

with the constraint

$$\int dxdydz \psi^*(x,y,z)\psi(x,y,z) = 1,$$

and a Hamiltonian

$$\hat{H} = -\frac{1}{2}\nabla^2 + V(x,y,z).$$

I will skip the variables x, y, z below, and write for example $V(x,y,z) = V$.

Variational Calculus and Lagrangian Multiplier

The integral involving the kinetic energy can be written as, if we assume periodic boundary conditions or that the function ψ vanishes strongly for large values of x, y, z,

$$\int dxdydz\psi^* \left(-\frac{1}{2}\nabla^2\right)\psi dxdydz = \psi^*\nabla\psi| + \int dxdydz\frac{1}{2}\nabla\psi^*\nabla\psi.$$

Inserting this expression into the expectation value for the energy and taking the variational minimum we obtain

$$\delta E = \delta\left\{\int dxdydz\left(\frac{1}{2}\nabla\psi^*\nabla\psi + V\psi^*\psi\right)\right\} = 0.$$

Variational Calculus and Lagrangian Multiplier

The constraint appears in integral form as

$$\int dxdydz \psi^* \psi = \text{constant},$$

and multiplying with a Lagrangian multiplier λ and taking the variational minimum we obtain the final variational equation

$$\delta \left\{ \int dxdydz \left(\frac{1}{2} \nabla\psi^* \nabla\psi + V\psi^*\psi - \lambda\psi^*\psi \right) \right\} = 0.$$

Introducing the function f

$$f = \frac{1}{2} \nabla\psi^* \nabla\psi + V\psi^*\psi - \lambda\psi^*\psi = \frac{1}{2}(\psi_x^*\psi_x + \psi_y^*\psi_y + \psi_z^*\psi_z) + V\psi^*\psi - \lambda\psi^*\psi,$$

where we have skipped the dependence on x, y, z and introduced the shorthand ψ_x, ψ_y and ψ_z for the various derivatives.

Variational Calculus and Lagrangian Multiplier

For ψ^* the Euler equation results in

$$\frac{\partial f}{\partial \psi^*} - \frac{\partial}{\partial x}\frac{\partial f}{\partial \psi^*_x} - \frac{\partial}{\partial y}\frac{\partial f}{\partial \psi^*_y} - \frac{\partial}{\partial z}\frac{\partial f}{\partial \psi^*_z} = 0,$$

which yields

$$-\frac{1}{2}(\psi_{xx} + \psi_{yy} + \psi_{zz}) + V\psi = \lambda\psi.$$

We can then identify the Lagrangian multiplier as the energy of the system. Then the last equation is nothing but the standard Schrödinger equation and the variational approach discussed here provides a powerful method for obtaining approximate solutions of the wave function.

Topics for Week 15, April 12-16

Hartree-Fock theory and Density functional theory

- Discussion of project 2, Hartree-Fock theory, wave functions and computation of Coulomb matrix elements.
- Project 1: Continuation on how to implement the Slater determinant and correlation part and the conjugate gradient method.
- Discussion of practicalities about the Hartree-Fock equations.
- Formal derivation of the Hartree-Fock equations. Continues next two weeks as well.

The material discussed for the rest of the course is covered by Thijssen's book chapters 4-6. We will use April to cover Hartree-Fock theory and Density Functional theory. May is devoted to finalizing project 2 only. Only Lab work during May.

Problems with neon for VMC

In the standard textbook case one uses spherical coordinates in order to get the hydrogen-like wave functions

$$x = rsin\theta cos\phi,$$

$$y = rsin\theta sin\phi,$$

and

$$z = rcos\theta.$$

The reason we introduce spherical coordinates is the spherical symmetry of the Coulomb potential

$$\frac{e^2}{4\pi\epsilon_0 r} = \frac{e^2}{4\pi\epsilon_0\sqrt{x^2+y^2+z^2}},$$

where we have used $r = \sqrt{x^2+y^2+z^2}$. It is not possible to find a separable solution of the type

$$\psi(x,y,z) = \psi(x)\psi(y)\psi(z).$$

However, with spherical coordinates we can find a solution of the form

$$\psi(r,\theta,\phi) = R(r)P(\theta)F(\phi).$$

Problems with neon for VMC

The angle-dependent differential equations result in the spherical harmonic functions as solutions, with quantum numbers l and m_l. These functions are given by

$$Y_{lm_l}(\theta,\phi) = P(\theta)F(\phi) = \sqrt{\frac{(2l+1)(l-m_l)!}{4\pi(l+m_l)!}} P_l^{m_l}(cos(\theta)) \exp{(im_l\phi)},$$

with $P_l^{m_l}$ being the associated Legendre polynomials They can be rewritten as

$$Y_{lm_l}(\theta,\phi) = sin^{|m_l|}(\theta) \times (\mathrm{polynom}(cos\theta)) \exp{(im_l\phi)},$$

Problems with neon for VMC

We have the following selected examples

$$Y_{00} = \sqrt{\frac{1}{4\pi}},$$

for $l = m_l = 0$,

$$Y_{10} = \sqrt{\frac{3}{4\pi}} cos(\theta),$$

for $l = 1$ og $m_l = 0$,

$$Y_{1\pm 1} = \sqrt{\frac{3}{8\pi}} sin(\theta) exp(\pm i\phi),$$

for $l = 1$ og $m_l = \pm 1$.

Problems with neon for VMC

A problem with the spherical harmonics is that they are complex. The introduction of *solid harmonics* allows the use of real orbital wave-functions for a wide range of applications. The complex solid harmonics $\mathcal{Y}_{lm_l}(\mathbf{r})$ are related to the spherical harmonics $Y_{lm_l}(\mathbf{r})$ through

$$\mathcal{Y}_{lm_l}(\mathbf{r}) = r^l Y_{lm_l}(\mathbf{r}).$$

By factoring out the leading r-dependency of the radial-function

$$\mathcal{R}_{nl}(\mathbf{r}) = r^{-l} R_{nl}(\mathbf{r}),$$

we obtain

$$\Psi_{nlm_l}(r,\theta,\phi) = \mathcal{R}_{nl}(\mathbf{r}) \cdot \mathcal{Y}_{lm_l}(\mathbf{r}).$$

Problems with neon for VMC

For the theoretical development of the *real solid harmonics* we first express the complex solid harmonics, C_{lm_l}, by (complex) Cartesian coordinates, and arrive at the real solid harmonics, S_{lm_l}, through the unitary transformation

$$\begin{pmatrix} S_{lm_l} \\ S_{l,-m_l} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} (-1)_l^m & 1 \\ -(-1)_l^m i & i \end{pmatrix} \begin{pmatrix} C_{lm_l} \\ C_{l,-m_l} \end{pmatrix}.$$

Problems with neon for VMC

This transformation will not alter any physical quantities that are degenerate in the subspace consisting of opposite magnetic quantum numbers (the angular momentum l is equal for both these cases). This means for example that the above transformation does not alter the energies, unless an external magnetic field is applied to the system. Henceforth, we will use the solid harmonics, and note that changing the spherical potential beyond the Coulomb potential will not alter the solid harmonics.

Problems with neon for VMC

We have defined

$$\mathcal{Y}_{lm_l}(\mathbf{r}) = r^l Y_{lm_l}(\mathbf{r}).$$

The real-valued spherical harmonics are defined as

$$S_{l0} = \sqrt{\frac{4\pi}{2l+1}}\mathcal{Y}_{l0}(\mathbf{r}),$$

$$S_{lm_l} = (-1)^{m_l}\sqrt{\frac{8\pi}{2l+1}}\mathrm{Re}\mathcal{Y}_{l0}(\mathbf{r}),$$

$$S_{lm_l} = (-1)^{m_l}\sqrt{\frac{8\pi}{2l+1}}\mathrm{Im}\mathcal{Y}_{l0}(\mathbf{r}),$$

for $m_l > 0$.

Problems with neon for VMC

The lowest-order real solid harmonics are listed in here

Real Solid Harmonics

$m_l \backslash l$	0	1	2	3
+3				$\frac{1}{2}\sqrt{\frac{5}{2}}(x^2-3y^2)x$
+2			$\frac{1}{2}\sqrt{3}(x^2-y^2)$	$\frac{1}{2}\sqrt{15}(x^2-y^2)z$
+1		x	$\sqrt{3}xz$	$\frac{1}{2}\sqrt{\frac{3}{2}}(5z^2-r^2)x$
0	1	y	$\frac{1}{2}(3z^2-r^2)$	$\frac{1}{2}(5z^2-3r^2)x$
-1		z	$\sqrt{3}yz$	$\frac{1}{2}\sqrt{\frac{3}{2}}(5z^2-r^2)y$
-2			$\sqrt{3}xy$	$\sqrt{15}xyz$
-3				$\frac{1}{2}\sqrt{\frac{5}{2}}(3x^2-y^2)y$

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

If we generalize the Euler-Lagrange equations to more variables and introduce N^2 Lagrange multipliers which we denote by $\epsilon_{\mu\nu}$, we can write the variational equation for the functional of E

$$\delta E - \sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\epsilon_{\mu\nu}\delta\int\psi_{\mu}^{*}\psi_{\nu} = 0.$$

For the orthogonal wave functions ψ_{μ} this reduces to

$$\delta E - \sum_{\mu=1}^{N}\epsilon_{\mu}\delta\int\psi_{\mu}^{*}\psi_{\mu} = 0.$$

Variational Calculus and Lagrangian Multiplier, back to Hartree-Fock

Our functional is written as

$$E[\Phi] = \sum_{\mu=1}^{N} \int \psi_\mu^*(\mathbf{r}_i)\hat{h}_i\psi_\mu(\mathbf{r}_i)d\mathbf{r}_i + \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\mu(\mathbf{r}_i)\psi_\nu(\mathbf{r}_j)d\mathbf{r}_i d\mathbf{r}_j \right.$$

$$\left. - \int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\mu(\mathbf{r}_j)\psi_\nu(\mathbf{r}_i)d\mathbf{r}_i d\mathbf{r}_j\right]$$

The more compact version is

$$E[\Phi] = \sum_{\mu=1}^{N}\langle\mu|h|\mu\rangle + \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\langle\mu\nu|\frac{1}{r_{ij}}|\mu\nu\rangle - \langle\mu\nu|\frac{1}{r_{ij}}|\nu\mu\rangle\right].$$

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

Variation with respect to the single-particle wave functions ψ_μ yields then

$$
\begin{aligned}
&\sum_{\mu=1}^{N}\int \delta\psi_\mu^*\hat{h}_i\psi_\mu d\mathbf{r}_i + \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\int \delta\psi_\mu^*\psi_\nu^*\frac{1}{r_{ij}}\psi_\mu\psi_\nu d\mathbf{r}_i d\mathbf{r}_j - \int \delta\psi_\mu^*\psi_\nu^*\frac{1}{r_{ij}}\psi_\nu\psi_\mu d\mathbf{r}_i d\mathbf{r}_j\right] \\
&+\sum_{\mu=1}^{N}\int \psi_\mu^*\hat{h}_i\delta\psi_\mu d\mathbf{r}_i + \frac{1}{2}\sum_{\mu=1}^{N}\sum_{\nu=1}^{N}\left[\int \psi_\mu^*\psi_\nu^*\frac{1}{r_{ij}}\delta\psi_\mu\psi_\nu d\mathbf{r}_i d\mathbf{r}_j - \int \psi_\mu^*\psi_\nu^*\frac{1}{r_{ij}}\psi_\nu\delta\psi_\mu d\mathbf{r}_i d\mathbf{r}_j\right] \\
&\qquad -\sum_{\mu=1}^{N}E_\mu\int \delta\psi_\mu^*\psi_\mu d\mathbf{r}_i - \sum_{\mu=1}^{N}E_\mu\int \psi_\mu^*\delta\psi_\mu d\mathbf{r}_i = 0.
\end{aligned}
$$

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

Although the variations $\delta\psi$ and $\delta\psi^*$ are not independent, they may in fact be treated as such, so that the terms dependent on either $\delta\psi$ and $\delta\psi^*$ individually may be set equal to zero. To see this, simply replace the arbitrary variation $\delta\psi$ by $i\delta\psi$, so that $\delta\psi^*$ is replaced by $-i\delta\psi^*$, and combine the two equations. We thus arrive at the Hartree-Fock equations

$$\left[-\frac{1}{2}\nabla_i^2 - \frac{Z}{r_i} + \sum_{\nu=1}^{N}\int \psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\nu(\mathbf{r}_j)d\mathbf{r}_j\right]\psi_\mu(\mathbf{r}_i)$$

$$-\left[\sum_{\nu=1}^{N}\int \psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\mu(\mathbf{r}_j)d\mathbf{r}_j\right]\psi_\nu(\mathbf{r}_i) = \epsilon_\mu\psi_\mu(\mathbf{r}_i).$$

Notice that the integration $\int d\mathbf{r}_j$ implies an integration over the spatial coordinates $\mathbf{r_j}$ and a summation over the spin-coordinate of electron j.

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

The two first terms are the one-body kinetic energy and the electron-nucleus potential. The third or *direct* term is the averaged electronic repulsion of the other electrons. This term is identical to the Coulomb integral introduced in the simple perturbative approach to the helium atom. As written, the term includes the 'self-interaction' of electrons when $i = j$. The self-interaction is cancelled in the fourth term, or the *exchange* term. The exchange term results from our inclusion of the Pauli principle and the assumed determinantal form of the wave-function. The effect of exchange is for electrons of like-spin to avoid each other.

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

A theoretically convenient form of the Hartree-Fock equation is to regard the direct and exchange operator defined through

$$V^{d}_{\mu}(\mathbf{r}_i) = \int \psi^*_{\mu}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\mu}(\mathbf{r}_j)d\mathbf{r}_j$$

and

$$V^{ex}_{\mu}(\mathbf{r}_i)g(\mathbf{r}_i) = \left(\int \psi^*_{\mu}(\mathbf{r}_j)\frac{1}{r_{ij}}g(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\mu}(\mathbf{r}_i),$$

respectively.

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

The function $g(\mathbf{r}_i)$ is an arbitrary function, and by the substitution $g(\mathbf{r}_i) = \psi_\nu(\mathbf{r}_i)$ we get

$$V_\mu^{ex}(\mathbf{r}_i)\psi_\nu(\mathbf{r}_i) = \left(\int \psi_\mu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\nu(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_\mu(\mathbf{r}_i).$$

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

We may then rewrite the Hartree-Fock equations as

$$H_i^{HF}\psi_\nu(\mathbf{r}_i) = \epsilon_\nu\psi_\nu(\mathbf{r}_i),$$

with

$$H_i^{HF} = h_i + \sum_{\mu=1}^{N} V_\mu^{d}(\mathbf{r}_i) - \sum_{\mu=1}^{N} V_\mu^{ex}(\mathbf{r}_i),$$

and where h_i is the one-body part

Hartree-Fock: Explicit expression for Direct Term

We want to show first that

$$V_{\mu}^{d}(\mathbf{r}_i) = \int \psi_{\mu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\mu}(\mathbf{r}_j)d\mathbf{r}_j$$

is

$$V_{\mu}^{d}(\mathbf{r}_i) = V_{nl}^{d}(r_i) = \sum_{n'l'} 2(2l'+1)\int_0^{\infty} |u_{n'l'}(r_j)|^2 \frac{1}{r_>}dr_j$$

with $r_> = max(r_i, r_j)$.

Hartree-Fock: Explicit expression for Direct Term

We need to break down the single-particle wave functions in radial, angular and spin parts. Note that direct term is diagonal in the spin quantum numbers. The single-particle wave functions are

$$\psi_\alpha(\mathbf{r}) = \psi_{nlm_lsm_s}(\mathbf{r}) = \phi_{nlm_l}(\mathbf{r})\xi_{m_s}(s)$$

with

$$\phi_{nlm_l}(\mathbf{r}) = R_{nl}(r)Y_{lm_l}(\hat{r})$$

and we defined $u_{nl}(r) = rR_{nl}(r)$. The direct term is, with a factor two from the spin degrees of freedom gives for a single shell

$$V^d_{n'l'}(r_i) = 2\sum_{m_l'=-l'}^{l}\int_0^\infty |\phi_{n'l'm_l'}(r_j)|^2\frac{1}{r_{ij}}d\mathbf{r}_j$$

which reads

$$V^d_{n'l'}(r_i) = 2\int_0^\infty |u_{n'l'}(r_j)|^2\frac{1}{r_{ij}}dr_j\sum_{m_l'=-l'}^{l}|Y_{lm_l'}(\theta_j,\phi_j)|^2 d\Omega_j,$$

with $d\Omega_j$ the angular part $d(cos\theta_j)d\phi_j$.

Hartree-Fock: Explicit expression for Direct Term

The addition theorem ofthe spherical harmonics yields

$$\sum_{m'_l=-l'}^{l} |Y_{lm'_l}(\theta_j,\phi_j)|^2 = \frac{2l'+1}{4\pi},$$

resulting in

$$V^d_{n'l'}(r_i) = \frac{2(2l'+1)}{4\pi}\int_0^\infty |u_{n'l'}(r_j)|^2 \frac{1}{r_{ij}} dr_j d\Omega_j.$$

The quantity r_{ij} depends on the angles of particle i and j.

Hartree-Fock: Explicit expression for Direct Term

The integral over the angles can be performed by expanding $1/r_{ij}$ in terms of spherical harmonics and using the fact that the functions $u_{n'l'}$ do not depend on the angles. We have

$$\frac{1}{r_{ij}} = \frac{1}{r_i}\sum_{l=0}^{\infty}\left(\frac{r_j}{r_i}\right)^l P_l cos(\theta)),$$

if $r_i > r_j$ or

$$\frac{1}{r_{ij}} = \frac{1}{r_j}\sum_{l=0}^{\infty}\left(\frac{r_i}{r_j}\right)^l P_l cos(\theta)),$$

if $r_j > r_i$. In a compact form it reads

$$\frac{1}{r_{ij}} = \sum_{l=0}^{\infty}\left(\frac{(r_<)^l}{(r_>)^{l+1}}\right) P_l cos(\theta)),$$

with $r_> = max(r_i, r_j)$ and with $r_< = min(r_i, r_j)$. P_l is the Legendre polynomial and

$$cos(\theta) = cos(\theta_i)cos(\theta_j) + sin(\theta_i)sin(\theta_j)cos(\phi_i - \phi_j).$$

Hartree-Fock: Explicit expression for Direct Term

We can use the expression for spherical harmonics to express the interaction as

$$\frac{1}{r_{ij}} = \sum_{l=0}^{\infty} \sum_{m_l'=-l'}^{l} \left(\frac{(r_<)^l}{(r_>)^{l+1}} \right) Y^*_{lm_l'}(\theta_i, \phi_i) Y_{lm_l'}(\theta_j, \phi_j)$$

and inserting it we obtain the final expression as

$$\Phi(r_i) = \sum_{n'l'} 2(2l'+1) \int_0^{\infty} |u_{n'l'}(r_j)|^2 \frac{1}{r_>} dr_j$$

with $r_> = max(r_i, r_j)$.

Hartree-Fock: Explicit expression for Exchange Term

Exercise: show that the exchange part can be written as

$$F_{nl}(r_i) = \sum_{n'l'} \sum_{\lambda=|l-l'|}^{l+l'} \frac{2l'+1}{2\lambda+1} < ll'00|\lambda 0 >^2 \left[\int_0^\infty u^*_{n'l'}(r_j) \frac{(r_<)^\lambda}{(r_>)^{\lambda+1}} u_{nl}(r_j) dr_j \right] u_{n'l'}(r_i)$$

with $r_> = max(r_i, r_j)$ and with $r_< = min(r_i, r_j)$.

Hartree-Fock: Explicit expression for Exchange Term

The Clebsch-Gordan coefficient can be written as

$$\langle ll'00|\lambda 0\rangle = \frac{1}{2}\sqrt{2\lambda+1}(1+(-1)^{2g})(-1)^g\sqrt{\frac{(2g-2l)!(2g-2l')!(2g-2l')!}{(2g+1)!}}$$

$$\times \frac{g!}{(g-l)!(g-l')!(g-\lambda)!}$$

with $2g = 2l+\lambda$. You can write a small function for this expression or use the enclosed programs.

Hartree-Fock: Explicit expressions for various Atoms

We need to discuss how the Hartree-Fock equations look like for helium, beryllium and neon.
The general equations take the form

$$\left[-\frac{1}{2}\nabla_i^2 - \frac{Z}{r_i} + \sum_{\nu=1}^{N}\int \psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\nu(\mathbf{r}_j)d\mathbf{r}_j\right]\psi_\mu(\mathbf{r}_i)$$
$$-\left[\sum_{\nu=1}^{N}\int \psi_\nu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\mu(\mathbf{r}_j)d\mathbf{r}_j\right]\psi_\nu(\mathbf{r}_i) = \epsilon_\mu\psi_\mu(\mathbf{r}_i).$$

Notice that the integration $\int d\mathbf{r}_j$ implies an integration over the spatial coordinates $\mathbf{r_j}$ and a summation over the spin-coordinate of electron j.

Hartree-Fock: Explicit expressions for various Atoms

A theoretically convenient form of the Hartree-Fock equation is to regard the direct and exchange operator defined through

$$V_{\mu}^{d}(\mathbf{r}_i) = \int \psi_{\mu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\mu}(\mathbf{r}_j)d\mathbf{r}_j$$

and

$$V_{\mu}^{ex}(\mathbf{r}_i)g(\mathbf{r}_i) = \left(\int \psi_{\mu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}g(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\mu}(\mathbf{r}_i),$$

respectively.

Hartree-Fock Equations

We may then rewrite the Hartree-Fock equations as

$$H_i^{HF}\psi_\nu(\mathbf{r}_i) = \epsilon_\nu\psi_\nu(\mathbf{r}_i),$$

with

$$H_i^{HF} = h_i + \sum_{\mu=1}^{N} V_\mu^{d}(\mathbf{r}_i) - \sum_{\mu=1}^{N} V_\mu^{ex}(\mathbf{r}_i),$$

and where h_i is the one-body part.

Hartree-Fock: Explicit expressions for various Atoms, helium

The Slater determinant for helium reads with two electrons in the 1*s* state

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \alpha, \beta) = \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_\alpha(\mathbf{r}_1) & \psi_\alpha(\mathbf{r}_2) \\ \psi_\beta(\mathbf{r}_1) & \psi_\beta(\mathbf{r}_2) \end{vmatrix},$$

with $\alpha = nlm_l sm_s = 1001/21/2$ and $\beta = nlm_l sm_s = 1001/2 - 1/2$ or using $m_s = 1/2 = \uparrow$ and $m_s = -1/2 = \downarrow$ as $\alpha = nlm_l sm_s = 1001/2 \uparrow$ and $\beta = nlm_l sm_s = 1001/2 \downarrow$. Writing out the Slater determinant we obtain

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \alpha, \beta) = \frac{1}{\sqrt{2}} \left[\psi_\alpha(\mathbf{r}_1)\psi_\beta(\mathbf{r}_2) - \psi_\beta(\mathbf{r}_1)\psi_\gamma(\mathbf{r}_2)\right],$$

and we see that the Slater determinant is antisymmetric with respect to the permutation of two particles, that is

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \alpha, \beta) = -\Phi(\mathbf{r}_2, \mathbf{r}_1, \alpha, \beta),$$

Hartree-Fock: Explicit expressions for various Atoms, helium

In the derivations of the direct and exchange terms we have not discussed the role of the electron spin.
Let us introduce

$$\psi_{nlm_lsm_s} = \phi_{nlm_l}(\mathbf{r})\xi_{m_s}(s)$$

with s is the spin (1/2 for electrons), m_s is the spin projection $m_s = \pm 1/2$, and the spatial part is

$$\phi_{nlm_l}(\mathbf{r}) = R_{nl}(r)Y_{lm_l}(\hat{\mathbf{r}})$$

with Y the spherical harmonics and $u_{nl} = rR_{nl}$. We have for helium

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \alpha, \beta) = \frac{1}{\sqrt{2}}\phi_{100}(\mathbf{r}_1)\phi_{100}(\mathbf{r}_2)\left[\xi_{\uparrow}(1)\xi_{\downarrow}(2) - \xi_{\uparrow}(2)\xi_{\downarrow}(1)\right],$$

Hartree-Fock: Explicit expressions for various Atoms, helium

The direct term acts on

$$\frac{1}{\sqrt{2}}\phi_{100}(\mathbf{r}_1)\phi_{100}(\mathbf{r}_2)\xi_{\uparrow}(1)\xi_{\downarrow}(2)$$

while the exchange term acts on

$$-\frac{1}{\sqrt{2}}\phi_{100}(\mathbf{r}_1)\phi_{100}(\mathbf{r}_2)\xi_{\uparrow}(2)\xi_{\downarrow}(1).$$

How do these terms get translated into the Hartree and the Fock terms?

Hartree-Fock: Explicit expressions for various Atoms, helium

The Hartree term

$$V_{\mu}^{d}(\mathbf{r}_i) = \int \psi_{\mu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\mu}(\mathbf{r}_j)d\mathbf{r}_j,$$

acts on $\psi_{\lambda}(\mathbf{r}_i) = \phi_{nlm_l}(\mathbf{r}_i)\xi_{m_s}(s_i)$, that is it results in

$$V_{\mu}^{d}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \left(\int \psi_{\mu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\mu}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\lambda}(\mathbf{r}_i),$$

and accounting for spins we have

$$V_{nlm_l\uparrow}^{d}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \left(\int \psi_{nlm_l\uparrow}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{nlm_l\uparrow}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\lambda}(\mathbf{r}_i),$$

and

$$V_{nlm_l\downarrow}^{d}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \left(\int \psi_{nlm_l\downarrow}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{nlm_l\downarrow}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\lambda}(\mathbf{r}_i),$$

Hartree-Fock: Explicit expressions for various Atoms, helium

If the state we act on has spin up, we obtain two terms from the Hartree part.

$$\sum_{\mu=1}^{N} V_{\mu}^{d}(\mathbf{r}_i),$$

and since the interaction does not depend on spin we end up with a total contribution for helium

$$\sum_{\mu=1}^{N} V_{\mu}^{d}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \left(2\int \phi_{100}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{100}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\lambda}(\mathbf{r}_i),$$

one from spin up and one from spin down. Since the energy for spin up or spin down is the same we can then write the action of the Hartree term as

$$\sum_{\mu=1}^{N} V_{\mu}^{d}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \left(2\int \phi_{100}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{100}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{100\uparrow}(\mathbf{r}_i).$$

(the spin in $\psi_{100\uparrow}$ is irrelevant)

Hartree-Fock: Explicit expressions for various Atoms, helium

What we need to code for helium is then

$$\Phi(r_i)u_{10} = 2V_{10}^{d}(r_i)u_{10}(r_i) = 2\int_0^\infty |u_{10}(r_j)|^2 \frac{1}{r_>} dr_j)u_{10}(r_i).$$

with $r_> = max(r_i, r_j)$. What about the exchange or Fock term

$$V_\mu^{ex}(\mathbf{r}_i)\psi_\lambda(\mathbf{r}_i) = \left(\int \psi_\mu^*(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_\lambda(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_\mu(\mathbf{r}_i)?$$

Hartree-Fock: Explicit expressions for various Atoms, helium

We must be careful here with

$$V^{ex}_{\mu}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \left(\int \psi^*_{\mu}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\lambda}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\mu}(\mathbf{r}_i),$$

because the spins of μ and λ have to be the same due to the constraint

$$\langle s_{\mu} m^{\mu}_s | s_{\lambda} m^{\lambda}_s \rangle = \delta_{m^{\mu}_s, m^{\lambda}_s}.$$

This means that if $m^{\mu}_s =\uparrow$ then $m^{\lambda}_s =\uparrow$ and if $m^{\mu}_s =\downarrow$ then $m^{\lambda}_s =\downarrow$. That is

$$V^{ex}_{\mu}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \delta_{m^{\mu}_s, m^{\lambda}_s}\left(\int \psi^*_{\mu}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{\lambda}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\mu}(\mathbf{r}_i),$$

Hartree-Fock: Explicit expressions for various Atoms, helium

The consequence is that for the $1s\uparrow$ (and the same for $1s\downarrow$) state we get only one contribution from the Fock term

$$\sum_{\mu=1}^{N} V_{\mu}^{ex}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i),$$

$$\sum_{\mu=1}^{N} V_{\mu}^{ex}(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) = \delta_{m_s^{\mu},\uparrow}\left(\int \psi_{\mu}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{100\uparrow}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{\mu}(\mathbf{r}_i),$$

resulting in

$$\sum_{\mu=1}^{N} V_{\mu}^{ex}(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) = \left(\int \psi_{100\uparrow}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{100\uparrow}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{100\uparrow}(\mathbf{r}_i).$$

Hartree-Fock: Explicit expressions for various Atoms, helium

The final Fock term for helium is then

$$\sum_{\mu=1}^{N} V_{\mu}^{ex}(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) = \left(\int \psi_{100\uparrow}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\psi_{100\uparrow}(\mathbf{r}_j)d\mathbf{r}_j\right)\psi_{100\uparrow}(\mathbf{r}_i),$$

which is exactly the same as the Hartree term except for a factor of 2. Else the integral is the same. We can then write the differential equation

$$\left(-\frac{1}{2}\frac{d^2}{dr^2} + \frac{l(l+1)}{2r^2} - \frac{2}{r} + \Phi_{nl}(r) - F_{nl}(r)\right)u_{nl}(r) = e_{nl}u_{nl}(r).$$

as

$$\left(-\frac{1}{2}\frac{d^2}{dr^2} + \frac{l(l+1)}{2r^2} - \frac{2}{r} + 2V_{10}^{d}(r)\right)u_{10}(r) - V_{10}^{ex}(r) = e_{10}u_{10}(r),$$

or

$$\left(-\frac{1}{2}\frac{d^2}{dr^2} - \frac{2}{r} + V_{10}^{d}(r)\right)u_{10}(r) = e_{10}u_{10}(r),$$

since $l=0$. The shorthand $V_{10}^{ex}(r)$ contains the $1s$ wave function and can be dangerous later!

Hartree-Fock: Explicit expressions for various Atoms, beryllium

The expression we have obtained are independent of the spin projections and we have skipped them in the equations. Last week's exercise was to derive the corresponding equations for beryllium, with two electrons in 1s as in helium but now also two electrons in 2s.
The Slater determinant takes the form

$$\Phi(\mathbf{r}_1,\mathbf{r}_2,,\mathbf{r}_3,\mathbf{r}_4,\alpha,\beta,\gamma,\delta) = \frac{1}{\sqrt{4!}} \begin{vmatrix} \psi_{100\uparrow}(\mathbf{r}_1) & \psi_{100\uparrow}(\mathbf{r}_2) & \psi_{100\uparrow}(\mathbf{r}_3) & \psi_{100\uparrow}(\mathbf{r}_4) \\ \psi_{100\downarrow}(\mathbf{r}_1) & \psi_{100\downarrow}(\mathbf{r}_2) & \psi_{100\downarrow}(\mathbf{r}_3) & \psi_{100\downarrow}(\mathbf{r}_4) \\ \psi_{200\uparrow}(\mathbf{r}_1) & \psi_{200\uparrow}(\mathbf{r}_2) & \psi_{200\uparrow}(\mathbf{r}_3) & \psi_{200\uparrow}(\mathbf{r}_4) \\ \psi_{200\downarrow}(\mathbf{r}_1) & \psi_{200\downarrow}(\mathbf{r}_2) & \psi_{200\downarrow}(\mathbf{r}_3) & \psi_{200\downarrow}(\mathbf{r}_4) \end{vmatrix},$$

Hartree-Fock: Explicit expressions for various Atoms, beryllium

When we now spell out the Hartree-Fock equations we get two coupled differential equations, one for u_{10} and one for u_{20}.
The 1*s* wave function has the same Hartree-Fock contribution as in helium for the 1*s* state, but the 2*s* state gives two times the Hartree term and one time the Fock term.
That is we get

$$\sum_{\mu=1}^{N} V_{\mu}^{d}(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) = 2\int_0^{\infty} d\mathbf{r}_j \left(\phi_{100}^*(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{100}(\mathbf{r}_j) + \phi_{200}^*(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{200}(\mathbf{r}_j)\right)\psi_{100\uparrow}(\mathbf{r}_i)$$

$$= (2V_{10}^{d}(\mathbf{r}_i) + 2V_{20}^{d}(\mathbf{r}_i))\psi_{100\uparrow}(\mathbf{r}_i)$$

for the Hartree part.

Hartree-Fock: Explicit expressions for various Atoms, beryllium

For the Fock term we get (we fix the spin)

$$\sum_{\mu=1}^{N} V_{\mu}^{ex}(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) = \int_0^{\infty} d\mathbf{r}_j \phi_{100}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{100}(\mathbf{r}_j)\psi_{100\uparrow}(\mathbf{r}_i) +$$

$$\int_0^{\infty} d\mathbf{r}_j \phi_{200}^{*}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{100}(\mathbf{r}_j)\psi_{200\uparrow}(\mathbf{r}_i) = V_{10}^{ex}(\mathbf{r}_i) + V_{20}^{ex}(\mathbf{r}_i).$$

The first term is the same as we have for the Hartree term with 1s except the factor of two. The final differential equation is

$$\left(-\frac{1}{2}\frac{d^2}{dr^2} - \frac{4}{r} + V_{10}^{d}(r) + 2V_{20}^{d}(r)\right)u_{10}(r) - V_{20}^{ex}(r) = e_{10}u_{10}(r).$$

Note again that the $V_{20}^{ex}(r)$ contains the 1s function in the integral, that is

$$V_{20}^{ex}(r) = \int_0^{\infty} d\mathbf{r}_j \phi_{200}^{*}(\mathbf{r}_j)\frac{1}{r - r_j}\phi_{100}(\mathbf{r}_j)\psi_{200\uparrow}(\mathbf{r}).$$

Hartree-Fock: Explicit expressions for various Atoms, beryllium

The $2s$ wave function obtains the following Hartree term (recall that the interaction has no spin dependence)

$$\sum_{\mu=1}^{N} V^d_\mu(\mathbf{r}_i)\psi_{200\uparrow}(\mathbf{r}_i) = 2\int_0^\infty d\mathbf{r}_j \left(\phi^*_{100}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{100}(\mathbf{r}_j) + \phi^*_{200}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{200}(\mathbf{r}_j)\right)\psi_{200\uparrow}(\mathbf{r}_i) =$$

$$(2V^d_{10}(\mathbf{r}_i) + 2V^d_{20}(\mathbf{r}_i))\psi_{200\uparrow}(\mathbf{r}_i)$$

Hartree-Fock: Explicit expressions for various Atoms, beryllium

For the Fock term we get

$$\sum_{\mu=1}^{N} V^{ex}_{\mu}(\mathbf{r}_i)\psi_{200\uparrow}(\mathbf{r}_i) = \int_0^{\infty} d\mathbf{r}_j \phi^*_{100}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{200}(\mathbf{r}_j)\psi_{100\uparrow}(\mathbf{r}_i) +$$

$$\int_0^{\infty} d\mathbf{r}_j \phi^*_{200}(\mathbf{r}_j)\frac{1}{r_{ij}}\phi_{200}(\mathbf{r}_j)\psi_{200\uparrow}(\mathbf{r}_i) = V^{ex}_{10}(\mathbf{r}_i) + V^{ex}_{20}(\mathbf{r}_i)$$

The second term is the same as we have for the Hartree term with $2s$. The final differential equation is

$$\left(-\frac{1}{2}\frac{d^2}{dr^2} - \frac{4}{r} + 2V^d_{10}(r) + V^d_{20}(r)\right)u_{20}(r) - V^{ex}_{10}(r) = e_{20}u_{20}(r).$$

Note again that the $V^{ex}_{10}(r)$ contains the $2s$ function in the integral, that is

$$V^{ex}_{10}(r) = \int_0^{\infty} d\mathbf{r}_j \phi^*_{100}(\mathbf{r}_j)\frac{1}{r - r_j}\phi_{200}(\mathbf{r}_j)\psi_{100\uparrow}(\mathbf{r}).$$

Hartree-Fock: Final expressions for beryllium

We have two coupled differential equations

$$\left(-\frac{1}{2}\frac{d^2}{dr^2}-\frac{4}{r}+V_{10}^{d}(r)+2V_{20}^{d}(r)\right)u_{10}(r)-V_{20}^{ex}(r)=e_{10}u_{10}(r),$$

and

$$\left(-\frac{1}{2}\frac{d^2}{dr^2}-\frac{4}{r}+2V_{10}^{d}(r)+V_{20}^{d}(r)\right)u_{20}(r)-V_{10}^{ex}(r)=e_{20}u_{20}(r).$$

Recall again that the interaction does not depend on spin. This means that the single-particle energies and single-particle function u do not depend on spin. Also

Exercise: derive the equivalent expressions for neon.

Hartree-Fock: Final expressions for neon

Exercise: fill in the missing parts

$$\left(-\frac{1}{2}\frac{d^2}{dr^2}-\frac{10}{r}+V_{10}^{d}(r)+2V_{20}^{d}(r)+?\right)u_{10}(r)-V_{20}^{ex}(r)+?=e_{10}u_{10}(r),$$

and

$$\left(-\frac{1}{2}\frac{d^2}{dr^2}-\frac{10}{r}+2V_{10}^{d}(r)+V_{20}^{d}(r)+?\right)u_{20}(r)-V_{10}^{ex}(r)+?=e_{20}u_{20}(r).$$

and

$$\left(-\frac{1}{2}\frac{d^2}{dr^2}+\frac{2}{2r^2}-\frac{10}{r}+2V_{10}^{d}(r)+2V_{20}^{d}(r)+?\right)u_{21}(r)-V_{10}^{ex}(r)-V_{20}^{ex}(r)+?=e_{21}u_{21}(r).$$

Useful equations

The 1s hydrogen like wave function

$$R_{10}(r) = 2\left(\frac{Z}{a_0}\right)^{3/2}\exp\left(-Zr/a_0\right) = u_{10}/r$$

The total energy for helium (not the Hartree or Fock terms) from the direct and the exchange term should give $5Z/8$.
The single-particle energy with no interactions should give $-Z^2/2n^2$.
The 2s hydrogen-like wave function is

$$R_{20}(r) = 2\left(\frac{Z}{2a_0}\right)^{3/2}\left(1-\frac{Zr}{2a_0}\right)\exp\left(-Zr/2a_0\right) = u_{20}/r$$

and the 2p hydrogen -like wave function is

$$R_{21}(r) = \frac{1}{\sqrt{3}}\left(\frac{Z}{2a_0}\right)^{3/2}\frac{Zr}{a_0}\exp\left(-Zr/2a_0\right) = u_{21}/r$$

We use $a_0 = 1$.

Effective Charge and first Iteration

If we compute the total energy of the helium atom with the function

$$R_{10}(r) = 2\left(\frac{Z}{a_0}\right)^{3/2} \exp\left(-Zr/a_0\right) = u_{10}/r,$$

as a trial single-particle wave fuction, we obtain a total energy (one-body and two-body)

$$E[Z] = Z^2 - 4Z + \frac{5}{8}Z.$$

The minimum is not at $Z = 2$. Take the derivative wrt Z and we find that the minimum is at

$$Z = 2 - \frac{5}{16} = 1.6875$$

and represents an optimal effective charge. When we do the Hartree-Fock calculations and use the optimal single-particle wave function in a variational Monte Carlo calculation, we should have the wave function calculated at the optimal value.

Hartree-Fock by varying the coefficients of a wave function expansion

Another possibility is to expand the single-particle functions in a known basis and vary the coefficients, that is, the new single-particle wave function is written as a linear expansion in terms of a fixed chosen orthogonal basis (for example harmonic oscillator, Laguerre polynomials etc)

$$\psi_a = \sum_\lambda C_{a\lambda}\psi_\lambda. \tag{95}$$

In this case we vary the coefficients $C_{a\lambda}$. If the basis has infinitely many solutions, we need to truncate the above sum. In all our equations we assume a truncation has been made.

The single-particle wave functions $\psi_\lambda(\mathbf{r})$, defined by the quantum numbers λ and $\mathbf{r}$ are defined as the overlap

$$\psi_\lambda(\mathbf{r}) = \langle \mathbf{r}|\lambda\rangle.$$

Hartree-Fock by varying the coefficients of a wave function expansion

We will omit the radial dependence of the wave functions and introduce first the following shorthands for the Hartree and Fock integrals

$$\langle \mu\nu|V|\mu\nu\rangle = \int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)V(r_{ij})\psi_\mu(\mathbf{r}_i)\psi_\nu(\mathbf{r}_j)d\mathbf{r}_i\mathbf{r}_j,$$

and

$$\langle \mu\nu|V|\nu\mu\rangle = \int \psi_\mu^*(\mathbf{r}_i)\psi_\nu^*(\mathbf{r}_j)V(r_{ij})\psi_\nu(\mathbf{r}_i)\psi_\mu(\mathbf{r}_i)d\mathbf{r}_i\mathbf{r}_j.$$

Hartree-Fock by varying the coefficients of a wave function expansion

Since the interaction is invariant under the interchange of two particles it means for example that we have

$$\langle \mu\nu|V|\mu\nu\rangle = \langle \nu\mu|V|\nu\mu\rangle,$$

or in the more general case

$$\langle \mu\nu|V|\sigma\tau\rangle = \langle \nu\mu|V|\tau\sigma\rangle.$$

Hartree-Fock by varying the coefficients of a wave function expansion

The direct and exchange matrix elements can be brought together if we define the antisymmetrized matrix element

$$\langle \mu\nu|V|\mu\nu\rangle_{AS} = \langle \mu\nu|V|\mu\nu\rangle - \langle \mu\nu|V|\nu\mu\rangle,$$

or for a general matrix element

$$\langle \mu\nu|V|\sigma\tau\rangle_{AS} = \langle \mu\nu|V|\sigma\tau\rangle - \langle \mu\nu|V|\tau\sigma\rangle.$$

It has the symmetry property

$$\langle \mu\nu|V|\sigma\tau\rangle_{AS} = -\langle \mu\nu|V|\tau\sigma\rangle_{AS} = -\langle \nu\mu|V|\sigma\tau\rangle_{AS}.$$

The antisymmetric matrix element is also hermitian, implying

$$\langle \mu\nu|V|\sigma\tau\rangle_{AS} = \langle \sigma\tau|V|\mu\nu\rangle_{AS}.$$

Hartree-Fock by varying the coefficients of a wave function expansion

With these notations we rewrite the Hartree-Fock functional as

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \frac{1}{2} \sum_{\mu=1}^{A} \sum_{\nu=1}^{A} \langle \mu\nu | V | \mu\nu \rangle_{AS}. \tag{96}$$

Combining Eqs. (13) and (96) we obtain the energy functional

$$E[\Phi] = \sum_{\mu=1}^{N} \langle \mu | h | \mu \rangle + \frac{1}{2} \sum_{\mu=1}^{N} \sum_{\nu=1}^{N} \langle \mu\nu | V | \mu\nu \rangle_{AS}. \tag{97}$$

which we will use as our starting point for the Hartree-Fock calculations.

Hartree-Fock by varying the coefficients of a wave function expansion

If we vary the above energy functional with respect to the basis functions $|\mu\rangle$, this corresponds to what was done in the previous case. We are however interested in defining a new basis defined in terms of a chosen basis as defined in Eq. (95). We can then rewrite the energy functional as

$$E[\Psi] = \sum_{a=1}^{N} \langle a|h|a\rangle + \frac{1}{2}\sum_{ab=1}^{N} \langle ab|V|ab\rangle_{AS}, \tag{98}$$

where Ψ is the new Slater determinant defined by the new basis of Eq. (95).

Hartree-Fock by varying the coefficients of a wave function expansion

Using Eq. (95) we can rewrite Eq. (98) as

$$E[\Psi] = \sum_{a=1}^{N}\sum_{\alpha\beta} C^*_{a\alpha}C_{a\beta}\langle\alpha|h|\beta\rangle + \frac{1}{2}\sum_{ab=1}^{N}\sum_{\alpha\beta\gamma\delta} C^*_{a\alpha}C^*_{b\beta}C_{a\gamma}C_{b\delta}\langle\alpha\beta|V|\gamma\delta\rangle_{AS}. \qquad (99)$$

Hartree-Fock by varying the coefficients of a wave function expansion

We wish now to minimize the above functional. We introduce again a set of Lagrange multipliers, noting that since $\langle a|b\rangle = \delta_{a,b}$ and $\langle \alpha|\beta\rangle = \delta_{\alpha,\beta}$, the coefficients $C_{a\gamma}$ obey the relation

$$\langle a|b\rangle = \delta_{a,b} = \sum_{\alpha\beta} C^*_{a\alpha} C_{a\beta} \langle \alpha|\beta\rangle = \sum_{\alpha} C^*_{a\alpha} C_{a\alpha},$$

which allows us to define a functional to be minimized that reads

$$E[\Psi] - \sum_{a=1}^{N} \epsilon_a \sum_{\alpha} C^*_{a\alpha} C_{a\alpha}. \tag{100}$$

Hartree-Fock by varying the coefficients of a wave function expansion

Minimizing with respect to $C^*_{k\alpha}$, remembering that $C^*_{k\alpha}$ and $C_{k\alpha}$ are independent, we obtain

$$\frac{d}{dC^*_{k\alpha}}\left[E[\Psi] - \sum_a \epsilon_a \sum_\alpha C^*_{a\alpha} C_{a\alpha}\right] = 0, \tag{101}$$

which yields for every single-particle state k the following Hartree-Fock equations

$$\sum_\gamma C_{k\gamma}\langle\alpha|h|\gamma\rangle + \sum_{a=1}^{N}\sum_{\beta\gamma\delta} C^*_{a\beta}C_{a\delta}C_{k\gamma}\langle\alpha\beta|V|\gamma\delta\rangle_{AS} = \epsilon_k C_{k\alpha}. \tag{102}$$

Hartree-Fock by varying the coefficients of a wave function expansion

We can rewrite this equation as

$$\sum_{\gamma}\left\{\langle\alpha|h|\gamma\rangle+\sum_{a}^{N}\sum_{\beta\delta}C^{*}_{a\beta}C_{a\delta}\langle\alpha\beta|V|\gamma\delta\rangle_{AS}\right\}C_{k\gamma}=\epsilon_{k}C_{k\alpha}. \tag{103}$$

Note that the sums over greek indices run over the number of basis set functions (in principle an infinite number).

Hartree-Fock by varying the coefficients of a wave function expansion

Defining

$$h^{HF}_{\alpha\gamma} = \langle\alpha|h|\gamma\rangle + \sum_{a=1}^{N}\sum_{\beta\delta} C^*_{a\beta}C_{a\delta}\langle\alpha\beta|V|\gamma\delta\rangle_{AS},$$

we can rewrite the new equations as

$$\sum_{\gamma} h^{HF}_{\alpha\gamma}C_{k\gamma} = \epsilon_k C_{k\alpha}. \tag{104}$$

Note again that the sums over greek indices run over the number of basis set functions (in principle an infinite number).

Hartree-Fock by varying the coefficients of a wave function expansion

The advantage of this approach is that we can calculate and tabulate the matrix elements $\alpha|h|\gamma\rangle$ and $\langle\alpha\beta|V|\gamma\delta\rangle_{AS}$ once and for all. If the basis $|\alpha\rangle$ is chosen properly, then the matrix elements can also serve as a good starting point for a Hartree-Fock calculation. Eq. (104) is nothing but an eigenvalue problem. The eigenvectors are defined by the coefficients $C_{k\gamma}$.

The size of the matrices to diagonalize are seldomly larger than 100×100 and can be solved by the standard eigenvalue methods that are discussed in chapter 12 of the lecture notes. Jacobi's method is enough!!

Hartree-Fock by varying the coefficients of a wave function expansion

For closed shell atoms it is natural to consider the spin-orbitals as paired. For example, two $1s$ orbitals with different spin have the same spatial wave-function, but orthogonal spin functions. For open-shell atoms two procedures are commonly used; the *restricted Hartree-Fock* (RHF) and *unrestricted Hartree-Fock* (UHF). In RHF all the electrons except those occupying open-shell orbitals are forced to occupy doubly occupied spatial orbitals, while in UHF all orbitals are treated independently. The UHF, of course, yields a lower variational energy than the RHF formalism. One disadvantage of the UHF over the RHF, is that whereas the RHF wave function is an eigenfunction of S^2, the UHF function is not; that is, the total spin angular momentum is not a well-defined quantity for a UHL wave-function. Here we limit our attention to closed shell RHF's, and show how the coupled HF equations may be turned into a matrix problem by expressing the spin-orbitals using known sets of basis functions.

Hartree-Fock by varying the coefficients of a wave function expansion

In principle, a complete set of basis functions must be used to represent spin-orbitals exactly, but this is not computationally feasible. A given finite set of basis functions is, due to the incompleteness of the basis set, associated with a *basis-set truncation error*. The limiting HF energy, with truncation error equal to zero, will be referred to as the *Hartree-Fock limit*.

Hartree-Fock by varying the coefficients of a wave function expansion

The computational time depends on the number of basis-functions and of the difficulty in computing the integrals of both the Fock matrix and the overlap matrix. Therefore we wish to keep the number of basis functions as low as possible and choose the basis-functions cleverly. By cleverly we mean that the truncation error should be kept as low as possible, and that the computation of the matrix elements of both the overlap and the Fock matrices should not be too time consuming.

Hartree-Fock by varying the coefficients of a wave function expansion

One choice of basis functions are the so-called *Slater type orbitals* (STO). They are defined as

$$\Psi_{nlm_l}(r,\theta,\phi) = \mathcal{N} r^{n_{eff}-1} e^{\frac{Z_{eff}\rho}{n_{eff}}} Y_{lm_l}(\theta,\phi). \tag{105}$$

Here $\mathcal{N}$ is a normalization constant that for the purpose of basis set expansion may be put into the unknown $c_{i\mu}$'s, Y_{lm_l} is a spherical harmonic and $\rho = r/a_0$.

Topics for Week 16, April 19-23

Hartree-Fock theory and Density functional theory

- Discussion of project 2, Hartree-Fock theory, wave functions and computation of Coulomb matrix elements.
- Project 1: Continuation on how to implement the Slater determinant and correlation part and the conjugate gradient method.
- Discussion of Hartree-Fock program
-

Hartree-Fock code

Some useful global variables (brute force coding)

```
int Z = 4;
const int basiscutoff = 20;   // Need to check this
const int nmax = basiscutoff/2;
double coulombIntegrals[nmax][nmax][nmax][nmax];
```

Hartree-Fock code

```
//Radial wave functions with n=1,2.. and  l=0 (s-
    waves)
double radial(int n, double r) {

  return pow(2.0*Z/n,1.5)*sqrt(1.0/(2*n*n))*
      LaguerreGeneral(n-1,1,2*Z*r/n)*exp(-Z*r/n);
}
```

Hartree-Fock code, direct term

$$\int_0^\infty r_1^2 dr_1 R^*_{n_\alpha 0}(r_1) R_{n_\gamma 0}(r_1)$$

$$\times \left[\frac{1}{(r_1)} \int_0^{r_1} r_2^2 dr_2 R^*_{n_\beta 0}(r_2) R_{n_\delta 0}(r_2) + \int_{r_1}^\infty r_2 dr_2 R^*_{n_\beta 0}(r_2) R_{n_\delta 0}(r_2) \right].$$

Hartree-Fock code

```
//Function to compute the inner integrals of the
    Coulomb matrix elements
double innerIntegrals(double r1, int n2, int n4,
    double upperlimit, int N) {

  double sum = 0;
  double x[N]; //Integration points
  double w[N]; //Integration weights
```

$$\frac{1}{(r_1)}\int_0^{r_1} r_2^2 dr_2 R^*_{n_\beta 0}(r_2) R_{n_\delta 0}(r_2)$$

```
  //First integral
  gauleg(0,r1,x,w,N);
  for (int i = 0; i < N; i++)
    sum += w[i]*x[i]*x[i]*radial(n2,x[i])*radial(n4
        ,x[i]);
  sum /= r1;
}
```

Hartree-Fock code

$$\int_{r_1}^{\infty} r_2 dr_2 R^*_{n_\beta 0}(r_2) R_{n_\delta 0}(r_2)$$

```
  //Second integral
  gauleg(r1,upperlimit,x,w,N);
  for (int i = 0; i < N; i++)
    sum += w[i]*x[i]*radial(n2,x[i])*radial(n4,x[i
        ]);

  return sum;
}
```

Hartree-Fock code

```
//Function for the Coulomb integral <n1,n2|V|n3,n4>
//This is specialized to l=0, only the quantum
    number n is needed
double coulombIntegral(int n1, int n2, int n3, int
    n4) {
  int N = 100; //Number of integration points
  double sum = 0;
  double x[N]; //Integration points
  double w[N]; //Integration weights
  double upperlimit = 100; //Upper limit (cutoff,
      in principle infinite)
  //Compute the integral
  gauleg(0,upperlimit,x,w,N);
  for (int i = 0; i < N; i++)
    sum += w[i]*x[i]*x[i]*radial(n1,x[i])*radial(n3
        ,x[i])*innerIntegrals(x[i],n2,n4,upperlimit
        ,N);
  return sum;
}
```

Hartree-Fock code

We need matrix element with greek indices (hydrogen-like wave functions)

$$\langle ab|V|cd\rangle_{AS} = \sum_{\alpha\beta\gamma\delta} C^*_{a\alpha} C^*_{b\beta} C_{a\gamma} C_{b\delta} \langle\alpha\beta|V|\gamma\delta\rangle_{AS}.$$

```
//Function which returns the matrix element <alpha,
    beta|V|gamma,delta>
double matrixElement(int alpha, int beta, int gamma
    , int delta) {

  //Return zero if orthogonality due to spin value
  if (alpha%2 != gamma%2 || beta%2 != delta%2)
    return 0;
  else //Return the relevant Coulomb integral
    return coulombIntegrals[alpha/2][beta/2][gamma
        /2][delta/2];
}
```

Hartree-Fock code

```
//Function that returns the anti-symmetrized matrix
     element
//  <alpha,beta|V|gamma,delta>_AS
double matrixElementAS(int alpha, int beta, int
   gamma, int delta) {

  return matrixElement(alpha,beta,gamma,delta) -
      matrixElement(alpha,beta,delta,gamma);
}
```

Hartree-Fock code

$$E[\Psi] = \sum_{a=1}^{N}\sum_{\alpha\beta} C^*_{a\alpha} C_{a\beta}\langle\alpha|h|\beta\rangle + \frac{1}{2}\sum_{ab=1}^{N}\sum_{\alpha\beta\gamma\delta} C^*_{a\alpha} C^*_{b\beta} C_{a\gamma} C_{b\delta}\langle\alpha\beta|V|\gamma\delta\rangle_{AS}.$$

```
//Function to compute the Hartree-Fock energy
double calcEnergy(double** C) {

  double energy = 0;
  for (int a = 0; a < Z; a++) {
    for (int beta = 0; beta < basiscutoff; beta++)
      energy -= C[a][beta]*C[a][beta] * Z*Z /
           (2.0*(beta/2+1)*(beta/2+1));
  }
}
```

Hartree-Fock code

```
//Function to compute the Hartree-Fock energy,
    continues
  for (int a = 0; a < Z; a++)
    for (int b = 0; b < Z; b++)
      for (int mu = 0; mu < basiscutoff; mu++)
        for (int nu = 0; nu < basiscutoff; nu++)
          for (int beta = 0; beta < basiscutoff;
              beta++)
            for (int delta = 0; delta < basiscutoff
                ; delta++)
              energy += .5*C[a][mu]*C[b][nu]*C[a][
                  beta]*C[b][delta]*matrixElementAS
                  (mu,nu,beta,delta);

  return energy;
}
```

Hartree-Fock code, main part

```
int main() {
.....
  //Compute the Coulomb matrix elements
  for (int n1 = 1; n1 <= nmax; n1++)
    for (int n2 = 1; n2 <= nmax; n2++)
      for (int n3 = n1; n3 <= nmax; n3++)
        for (int n4 = n2; n4 <= nmax; n4++)
          coulombIntegrals[n3-1][n2-1][n1-1][n4-1]
             = coulombIntegrals[n1-1][n4-1][n3-1][
             n2-1]
            = coulombIntegrals[n3-1][n4-1][n1-1][n2
              -1] = coulombIntegrals[n1-1][n2-1][
              n3-1][n4-1] = coulombIntegral(n1,n2
              ,n3,n4);
```

Hartree-Fock code, main part, coefficients *C*

```
  //Declare the matrix C, initialize with
     coefficient as a pure
//  hydrogenic state with no mixing
  double** C = (double**)matrix(Z,basiscutoff,
     sizeof(double));
  for (int i = 0; i < Z; i++)
    for (int j = 0; j < basiscutoff; j++)
      C[i][j] = 0;
  for (int i = 0; i < Z; i++)
    C[i][i] = 1;
```

Hartree-Fock code

```
//Declare the Hartree-Fock matrix
double** hHF = (double**)matrix(basiscutoff,
    basiscutoff,sizeof(double));

//We need two vectors to keep track of the tri-
    diagonal matrix in Householder's //algorithm
    and the final eigenvalues
double* d = new double[basiscutoff];
double* e = new double[basiscutoff];

//Energy at present and previous iterations. If
    self-consistency is reached
// they should be equal within a small chosen
    numerical lower limit
double energy = 0;
double energyprev;
```

Hartree-Fock matrix

$$h^{HF}_{\alpha\gamma} = \langle\alpha|h|\gamma\rangle + \sum_{a=1}^{N}\sum_{\beta\delta} C^*_{a\beta}C_{a\delta}\langle\alpha\beta|V|\gamma\delta\rangle_{AS},$$

we can rewrite the new equations as

$$\sum_{\gamma} h^{HF}_{\alpha\gamma}C_{k\gamma} = \epsilon_k C_{k\alpha}.$$

Hartree-Fock code, the iteration itself

```
do {
  energyprev = energy;
  //Set up the Hartree-Fock matrix
  for (int alpha = 0; alpha < basiscutoff; alpha
      ++) {
    for (int gamma = 0; gamma < basiscutoff;
        gamma++) {
      //Matrix element <alpha|h|gamma>
      if (alpha == gamma)
        hHF[alpha][gamma] = -Z*Z / (2.0*(alpha
            /2+1)*(alpha/2+1)); // E = -Z^2/(2n
            ^2)
```

Hartree-Fock code, the iteration itself

```
        else
          hHF[alpha][gamma] = 0;
        for (int a = 0; a < Z; a++)
          for (int beta = 0; beta < basiscutoff;
               beta++)
            for (int delta = 0; delta < basiscutoff
                 ; delta++)
              hHF[alpha][gamma] += C[a][beta]*C[a][
                 delta]*matrixElementAS(alpha,beta
                 ,gamma,delta);
    }
  }
```

Hartree-Fock code, the iteration

```
    // Find eigenvalues and eigenvectors, no
       sorting
    tred2(hHF, basiscutoff,d,e);
    tqli(d,e, basiscutoff,hHF);
    //Sort eigenvectors in order that lowest energy
        comes first
    sort(hHF,d, basiscutoff);

    //The Eigenvectors are stored in the rows of C
    for (int k = 0; k < Z; k++)
      for (int epsilon = 0; epsilon < basiscutoff;
          epsilon++)
        C[k][epsilon] = hHF[epsilon][k];
    //Compute the energy. Can you think of an
       alternative strategy?
    energy = calcEnergy(C);
    cout << energy << endl;
  } while(abs(energy-energyprev) > 1e-6);
```

Hartree-Fock code

```
ofstream orb1s("orb1s"); ofstream orb2s("orb2s");
//Write 1s and 2s wave functions to files
double sum;
for (double r = 0; r < 5; r += .01) {
  //1s
  sum = 0;
  for (int alpha = 0; alpha < basiscutoff; alpha
      ++)
    sum += C[0][alpha]*radial(alpha/2+1,r);

  orb1s << setprecision(8) << r << " " << sum <<
      endl;
  //2s
  sum = 0;
  for (int alpha = 0; alpha < basiscutoff; alpha
      ++)
    sum += C[2][alpha]*radial(alpha/2+1,r);
  orb2s << setprecision(8) << r << " " << sum <<
      endl;
}
```

Hartree-Fock code, sorting of eigenvalues and vectors

```
void sort(double** A, double* d, int n) {
  double* temp = new double[n];
  double tempval;
  do {
    for (int i = 0; i < n-1; i++) {
      if (d[i] > d[i+1]) {
        tempval = d[i];
        d[i] = d[i+1];
        d[i+1] = tempval;
        for (int j = 0; j < n; j++)
          temp[j] = A[j][i];
        for (int j = 0; j < n; j++)
          A[j][i] = A[j][i+1];
        for (int j = 0; j < n; j++)
          A[j][i+1] = temp[j];
      }
    }
  } while(!isSorted(d,n));
  delete[] temp;
}
```

Hartree-Fock code, sorting of eigenvalues and vectors

```
//Function to sort eigenvalues
bool isSorted(double* d, int n) {

  for (int i = 0; i < n-1; i++)
    if (d[i] > d[i+1])
      return false;

  return true;
}
```

Topics for Week 17, April 26-30

Hartree-Fock theory and Density functional theory

This is the last formal lecture. The rest of the period (May) is devoted to finalizing the projects and to write the report. Only lab in this period.

- Repetition from last week with a discussion of project 2, Hartree-Fock theory, wave functions and computation of Coulomb matrix elements.
- How to write the final report
- Brief overview of density functional theory

The report

What should it contain? A possible structure

- An introduction where you explain the rational for the physics case and what you have done. At the end of the introduction you should give a brief summary of the structure of the report
- Theoretical models and technicalities. This is the methods section.
- Results and discussion
- Conclusions and perspectives
- Appendix with extra material
- Bibliography

The report

What should I focus on? Introduction

You don't need to answer all questions in projects 1 and 2 in a chronological order. When you write the introduction you could focus on the following aspects

- A central aim is to study the role of correlations due to the repulsion between the electrons.
- To do this we have singled out three closed-shell atoms
- We use variational Monte Carlo and try different trial wave functions to see how close we get to experiment/exact result for a given Hamiltonian
- We test also the wave functions by computing onebody densities and compare these with those obtained with a non-interacting wave function.
- We perform also the simplest possible many-body calculations, namely Hartree-Fock and use these wave functions in an improved description of the Slater determinant

The report

What should I focus on? Methods sections

- Describe the methods (quantum mechanical and algorithms)
- You need to explain variational Monte Carlo and Hartree-Fock
- The trial wave functions. Why do you choose the functions you do?
- Why do you do importance sampling? And blocking and Conjugate gradient. You don't need to explain in detail these methods.
- You need to explain how you implemented the methods and also say something about the structure of your algorithm and present some parts of your code (Slater det and Jastrow factor).
- You can also plug in some calculations to demonstrate your code, such as selected runs from exercises 1a-1d for helium

The report

What should I focus on? Results

- Focus on beryllium
- As an example, you should present results for the Δt dependence for beryllium only but keep in the appendix some selected Δt for helium and neon.
- Same applies to the blocking analysis and the conjugate gradient method
- Same for the onebody densities, focus on beryllium and various wave functions.
- Discuss the results from the two many-body methods and the various wave functions. What do we learn?

The report

What should I focus on? Conclusions

- State your main findings and interpretations
- Try as far as possible to present perspectives for future work
- Try to discuss the pros and cons of the methods and possible improvements

The report

What should I focus on? additional material

- Additional calculations used to validate the codes
- Selected calculations for helium and neon, these can be listed with few comments
- Listing of the code if you feel this is necessary

You can consider moving parts of the material from the methods section to the appendix. You can also place additional material on your webpage.

The report

What should I focus on? References

- Give always references to material you base your work on, either scientific articles/reports or books.
- *Wikipedia is not accepted as a scientific reference*. Under no circumstances.
- Refer to articles as: name(s) of author(s), journal, volume (boldfaced), page and year in parenthesis.
- Refer to books as: name(s) of author(s), title of book, publisher, place and year, eventual page numbers

The exam

Dates and structure

- Two days: June 11 and June 14. Are these ok? Send me your wishes for day and time as soon as possible. Actual times are 9-17 both days.
- Duration 45 minutes
- Give a presentation of your report, 30 mins. Slides only.
- Then questions and feedback.
- Your final grade will be based on the report, your presentation and what you have done in total.

DFT: Litterature I

- R. van Leeuwen: *Density functional approach to the many-body problem: key concepts and exact functionals*, Adv. Quant. Chem. **43**, 25 (2003). (Mathematical foundations of DFT)
- R. M. Dreizler and E. K. U. Gross: *Density functional theory: An approach to the quantum many-body problem*. (Introductory book)
- W. Koch and M. C. Holthausen: *A chemist's guide to density functional theory*. (Introductory book, less formal than Dreizler/Gross)
- E. H. Lieb: Density functionals for Coulomb systems, Int. J. Quant. Chem. **24**, 243-277 (1983). (Mathematical analysis of DFT)

DFT: Litterature II

- J. P. Perdew and S. Kurth: In *A Primer in Density Functional Theory: Density Functionals for Non-relativistic Coulomb Systems in the New Century*, ed. C. Fiolhais *et al.* (Introductory course, partly difficult, but interesting points of view)
- E. Engel: In *A Primer in Density Functional Theory: Orbital-Dependent Functionals for the Exchange-Correlation Energy*, ed. C. Fiolhais *et al.* (Introductory lectures, only about orbital-dependent functionals)

Density Functional Theory (DFT)

Hohenberg and Kohn proved that the total energy of a system including that of the many-body effects of electrons (exchange and correlation) in the presence of static external potential (for example, the atomic nuclei) is a unique functional of the charge density. The minimum value of the total energy functional is the ground state energy of the system. The electronic charge density which yields this minimum defines the ground state energy.
In Hartree-Fock theory one works with large basis sets. This poses a problem for large systems. An alternative to the HF methods is DFT. DFT takes into account electron correlations but is less demanding computationally than full scale diagonalization or Monte Carlo methods.

Density Functional Theory

The electronic energy E is said to be a *functional* of the electronic density, $E[\rho]$, in the sense that for a given function $\rho(r)$, there is a single corresponding energy. The *Hohenberg-Kohn theorem* confirms that such a functional exists, but does not tell us the form of the functional. As shown by Kohn and Sham, the exact ground-state energy E of an N-electron system can be written as

$$E[\rho] = -\frac{1}{2}\sum_{i=1}^{N}\int \Psi_i^*(\mathbf{r_1})\nabla_1^2\Psi_i(\mathbf{r_1})d\mathbf{r_1} - \int \frac{Z}{r_1}\rho(\mathbf{r_1})d\mathbf{r_1} + \frac{1}{2}\int \frac{\rho(\mathbf{r_1})\rho(\mathbf{r_2})}{r_{12}}d\mathbf{r_1}d\mathbf{r_2} + E_{EXC}[\rho]$$

with Ψ_i the *Kohn-Sham* (KS) *orbitals*.

Density Functional Theory

The ground-state charge density is given by

$$\rho(\mathbf{r}) = \sum_{i=1}^{N} |\Psi_i(\mathbf{r})|^2,$$

where the sum is over the occupied Kohn-Sham orbitals. The last term, $E_{EXC}[\rho]$, is the *exchange-correlation energy* which in theory takes into account all non-classical electron-electron interaction. However, we do not know how to obtain this term exactly, and are forced to approximate it. The KS orbitals are found by solving the *Kohn-Sham equations*, which can be found by applying a variational principle to the electronic energy $E[\rho]$. This approach is similar to the one used for obtaining the HF equation.

Density Functional Theory

The KS equations reads

$$\left\{-\frac{1}{2}\nabla_1^2 - \frac{Z}{r_1} + \int \frac{\rho(\mathbf{r_2})}{r_{12}} d\mathbf{r_2} + V_{EXC}(\mathbf{r_1})\right\} \Psi_i(\mathbf{r_1}) = \epsilon_i \Psi_i(\mathbf{r_1})$$

where ϵ_i are the KS orbital energies, and where the *exchange-correlation potential* is given by

$$V_{EXC}[\rho] = \frac{\delta E_{EXC}[\rho]}{\delta \rho}.$$

Density Functional Theory

The KS equations are solved in a self-consistent fashion. A variety of basis set functions can be used, and the experience gained in HF calculations are often useful. The computational time needed for a DFT calculation formally scales as the third power of the number of basis functions.
The main source of error in DFT usually arises from the approximate nature of E_{EXC}. In the *local density approximation* (LDA) it is approximated as

$$E_{EXC} = \int \rho(\mathbf{r})\epsilon_{EXC}[\rho(\mathbf{r})]d\mathbf{r},$$

where $\epsilon_{EXC}[\rho(\mathbf{r})]$ is the exchange-correlation energy per electron in a homogeneous electron gas of constant density. The LDA approach is clearly an approximation as the charge is not continuously distributed. To account for the inhomogeneity of the electron density, a nonlocal

Hamiltonian of N *non-interacting* particles:

$$\hat{H}_s = \hat{T} + \hat{V}_s$$

Hohenberg and Kohn $\implies$ $\exists$ unique energy functional

$$E_s[n] = T_s[n] + \int v_s(\mathbf{r})n(\mathbf{r})d^3r$$

s. t. $\delta E_s[n] = 0$ gives GS density $n_s(\mathbf{r})$ corresp. to $\hat{H}_s$

Theorem

Let

$$\begin{aligned} v_s(\mathbf{r}) &= \textit{local single-particle pot.}, \\ n(\mathbf{r}) &= \textit{GS density of interacting system}, \\ n_s(\mathbf{r}) &= \textit{GS density of non-interacting system} \end{aligned}$$

$\Longrightarrow$ *for any interacting system,*

$$\exists \;\; a \;\; v_s(\mathbf{r}) \;\; s.\ t. \;\; n_s(\mathbf{r}) = n(\mathbf{r})$$

Proof in book by Dreizler/Gross, Sec. 4.2

In proof: $F_{HK}[n]$ replaced by $F_L[n]$

Theorem

Let

$$\begin{aligned} v_s(\mathbf{r}) &= \textit{local single-particle pot.,} \\ n(\mathbf{r}) &= \textit{GS density of interacting system,} \\ n_s(\mathbf{r}) &= \textit{GS density of non-interacting system} \end{aligned}$$

$\Longrightarrow$ *for any interacting system,*

$$\exists \quad a \quad v_s(\mathbf{r}) \;\; s.\ t. \;\; n_s(\mathbf{r}) = n(\mathbf{r})$$

Proof in book by Dreizler/Gross, Sec. 4.2

In proof: $F_{HK}[n]$ replaced by $F_L[n]$

Assume nondegenerate GS. Then

$$n(\mathbf{r}) = n_s(\mathbf{r}) = \sum_{i=1}^{N} |\phi_i(\mathbf{r})|^2 \,,$$

where $\phi_i(\mathbf{r})$ are determined by

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_s(\mathbf{r})\right)\phi_i(\mathbf{r}) = \varepsilon_i \phi_i(\mathbf{r}), \qquad \varepsilon_1 \le \varepsilon_2 \le \dots \,.$$

If $\exists\ v_s(\mathbf{r})$, then H-K theorem gives *uniqueness* of $v_s(\mathbf{r})$
Consequently, we may write

$$\phi_i(\mathbf{r}) = \phi_i([n(\mathbf{r})]) \qquad !!$$

Assume nondegenerate GS. Then

$$n(\mathbf{r}) = n_s(\mathbf{r}) = \sum_{i=1}^{N} |\phi_i(\mathbf{r})|^2 \,,$$

where $\phi_i(\mathbf{r})$ are determined by

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_s(\mathbf{r})\right)\phi_i(\mathbf{r}) = \varepsilon_i \phi_i(\mathbf{r}), \qquad \varepsilon_1 \leq \varepsilon_2 \leq \dots \,.$$

If $\exists\ v_s(\mathbf{r})$, then H-K theorem gives *uniqueness* of $v_s(\mathbf{r})$
Consequently, we may write

$$\phi_i(\mathbf{r}) = \phi_i([n(\mathbf{r})]) \qquad \textbf{!!}$$

Assume

$$v_0(\mathbf{r}) = \text{ext. potential}$$
$$n_0(\mathbf{r}) = \text{GS density}$$

of interacting system

- Wanted: single-particle potential $v_s(\mathbf{r})$ of non-interacting system

Many-particle energy functional:

$$
\begin{aligned}
E_{v_0}[n] &= F_L[n] + \int d^3 v_0(\mathbf{r}) n(\mathbf{r}) \\
&= \left(T_s[n] + \frac{1}{2} \iint d^3r d^3r' \, n(\mathbf{r}) w(\mathbf{r}, \mathbf{r}') n(\mathbf{r}') + E_{xc}[n] \right) + \int d^3r v_0(\mathbf{r}) n(\mathbf{r})
\end{aligned}
$$

Here exchange-correlation functional defined:

$$E_{xc}[n] = F_L[n] - \frac{1}{2} \iint d^3r d^3r' \, n(\mathbf{r}) w(\mathbf{r}, \mathbf{r}') n(\mathbf{r}') - T_s[n]$$

The exchange-correlation functional defined:

$$E_{xc}[n] = F_L[n] - \frac{1}{2}\iint d^3r d^3r' n(\mathbf{r}) w(\mathbf{r}, \mathbf{r}') n(\mathbf{r}') - T_s[n]$$

Explicit form of $F_L[n]$ as functional of n unknown

- $E_{xc}[n]$ unknown functional, must be approximated
  Otherwise, Kohn-Sham scheme exact

Definition

Let $F : B \to \mathbb{R}$ be a *functional* from normed function space B to real numbers $\mathbb{R}$.

The functional derivative (Gâteaux derivative)
$\delta F[n] \equiv \delta F[n]/\delta n(\mathbf{r})$ is defined as

$$\frac{\delta F}{\delta n}[\varphi] = \lim_{\varepsilon \to 0} \frac{F[n + \varepsilon\varphi] - F[n]}{\varepsilon}$$

Another useful definition of $\delta F[n]$:

$$\langle \delta F[n], \varphi \rangle = \frac{d}{d\varepsilon} F[n + \varepsilon\phi] \bigg|_{\varepsilon=0},$$

where

$$\langle \delta F[n], \varphi \rangle \equiv \int d\mathbf{r} (\delta F[n(\mathbf{r})]) \varphi(\mathbf{r}),$$

$\varphi =$ test function

Let us derive expression for single-particle potential $v_s(\mathbf{r})$ of non-interacting system:

H-K variational principle:

$$\begin{aligned} 0 = \delta E_{v_0} &= E_{v_0}[n_0 + \delta n] - E_{v_0}[n_0] \\ &= \delta T_s + \int d^3r \delta n(\mathbf{r}) \left[v_0(\mathbf{r}) + \int w(\mathbf{r}, \mathbf{r}') d^3r' + v_{xc}([n_0]; \mathbf{r}) \right], \end{aligned} \tag{106}$$

where exchange-coorelation potential

$$v_{xc}([n_0]; \mathbf{r}) = \left. \frac{\delta E_{xc}[n]}{\delta n(\mathbf{r})} \right|_{n_0},$$

$n_0(\mathbf{r}) =$ GS density

$n_0(\mathbf{r}) + \delta n(\mathbf{r})$ non-interacting v-representable $\implies$ unique representation $\phi_{i,0}(\mathbf{r}) + \delta\phi_i(\mathbf{r})$

$$
\begin{aligned}
\delta T_s &= \sum_i^N \int d^3r \left[\delta\phi_i^*(\mathbf{r}) \left(-\frac{\hbar^2}{2m}\nabla^2 \right) \phi_{i,0}(\mathbf{r}) + \phi_{i,0}^*(\mathbf{r}) \left(-\frac{\hbar^2}{2m}\nabla^2 \right) \delta\phi_i(\mathbf{r}) \right] \\
&= \sum_i^N \int d^3r \left[\delta\phi_i^*(\mathbf{r}) \left(-\frac{\hbar^2}{2m}\nabla^2 \right) \phi_{i,0}(\mathbf{r}) + \delta\phi_{i,0}^*(\mathbf{r}) \left(-\frac{\hbar^2}{2m}\nabla^2 \right) \phi_i(\mathbf{r}) \right] \qquad (107)
\end{aligned}
$$

$\uparrow$

Green's first identity

Green's first identity:

$$\int_V f\,\nabla^2 g\,dV = \oint_S f(\nabla g \cdot n)\,dS - \int_V \nabla f \cdot \nabla g\,dV,$$

where $V \in \mathbb{R}^3$, $S \equiv \partial V \in \mathbb{R}^2$ and f, $g =$ arb. real scalar functions

Let surface ∂V approach infinity w.r.t. origin,
assume $f, g \longrightarrow 0$ on ∂V,
Apply Green's first identity twice $\quad\Longrightarrow$

$$\begin{aligned}\int_V f\,\nabla^2 g\,dV &= 0 - \int_V \nabla f \cdot \nabla g\,dV \\ &= -\left(0 - \int_V \nabla f \cdot \nabla g\,dV\right) \\ &= \int_V g\,\nabla^2 f\,dV\end{aligned}$$

The orbitals $\phi_{i,0}(\mathbf{r})$ in Eq. (107) satisfy

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_{s,0}(\mathbf{r})\right)\phi_{i,0}(\mathbf{r}) = \varepsilon_i\phi_{i,0}(\mathbf{r}), \qquad \varepsilon_1 \geq \varepsilon_2 \geq \ldots\ . \tag{108}$$

Using this relation, we may rewrite Eq. (107) as

$$\begin{aligned} \delta T_s &= \sum_i^N \int d^3r \left[\delta\phi_i^*(\mathbf{r})\left(\varepsilon_i - v_{s,0}(\mathbf{r})\right)\phi_{i,0}(\mathbf{r}) + \delta\phi_i(\mathbf{r})\left(\varepsilon_i - v_{s,0}(\mathbf{r})\right)\phi_i^*(\mathbf{r})\right] \\ &= \sum_{i=1}^N \varepsilon_i \int d^3r\delta|\phi_i(\mathbf{r})|^2 - \sum_{i=1}^N \int d^3r v_{s,0}(\mathbf{r})\delta|\phi_i(\mathbf{r})|^2. \end{aligned} \tag{109}$$

Since

$$\int d^3r\delta|\phi_i(\mathbf{r})|^2 = \int d^3r \left[|\phi_{i,0}(\mathbf{r}) + \delta\phi_{i,0}(\mathbf{r})|^2 - |\phi_{i,0}(\mathbf{r})|^2\right]$$
$$= 1 - 1 = 0, \quad (110)$$

the first term of Eq. (109) vanishes, and we get

$$\delta T_s = -\int d^3r v_{s,0}(\mathbf{r})\delta n(\mathbf{r}). \quad (111)$$

Combine Eqs. (106) and (111): $\Longrightarrow$ total single-particle potential:

$$v_{s,0}(\mathbf{r}) = v_0(\mathbf{r}) + \int d^3r' w(\mathbf{r},\mathbf{r}')n_0(\mathbf{r}') + v_{xc}([n_0];\mathbf{r}) \quad (112)$$

The Kohn-Sham scheme I

The classic Kohn-Sham scheme:

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_{s,0}(\mathbf{r})\right)\phi_{i,0}(\mathbf{r}) = \varepsilon_i\phi_{i,0}(\mathbf{r}), \qquad \varepsilon_1 \geq \varepsilon_2 \geq \dots ,$$

where

$$v_{s,0}(\mathbf{r}) = v_0(\mathbf{r}) + \int d^3r' w(\mathbf{r},\mathbf{r}')n_0(\mathbf{r}') + v_{xc}([n_0];\mathbf{r})$$

The density calculated as

$$n_0(\mathbf{r}) = \sum_{i=1}^{N} |\phi_{i,0}(\mathbf{r})|^2,$$

Equation solved selfconsistently
Total energy:

$$E = \sum_{i=1}^{N}\varepsilon_i - \frac{1}{2}\int d^3r d^3r' n(\mathbf{r})w(\mathbf{r},\mathbf{r}')n(\mathbf{r}') + E_{xc}[n] - \int d^3r v_{xc}([n];\mathbf{r})n(\mathbf{r})$$

The Kohn-Sham scheme I

The classic Kohn-Sham scheme:

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_{s,0}(\mathbf{r})\right)\phi_{i,0}(\mathbf{r}) = \varepsilon_i\phi_{i,0}(\mathbf{r}), \qquad \varepsilon_1 \geq \varepsilon_2 \geq \dots\ ,$$

where

$$v_{s,0}(\mathbf{r}) = v_0(\mathbf{r}) + \int d^3r'\, w(\mathbf{r},\mathbf{r}')n_0(\mathbf{r}') + v_{xc}([n_0];\mathbf{r})$$

The density calculated as

$$n_0(\mathbf{r}) = \sum_{i=1}^{N} |\phi_{i,0}(\mathbf{r})|^2,$$

Equation solved selfconsistently
Total energy:

$$E = \sum_{i=1}^{N}\varepsilon_i - \frac{1}{2}\int d^3r d^3r'\, n(\mathbf{r})w(\mathbf{r},\mathbf{r}')n(\mathbf{r}') + E_{xc}[n] - \int d^3r v_{xc}([n];\mathbf{r})n(\mathbf{r})$$

The Kohn-Sham scheme II

Kohn-Sham scheme for systems with degenerate GS:

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_{s,0}(\mathbf{r})\right)\phi_{i,0}(\mathbf{r}) = \varepsilon_i\phi_{i,0}(\mathbf{r}), \qquad \varepsilon_1 \geq \varepsilon_2 \geq \dots ,$$

where

$$v_{s,0}(\mathbf{r}) = v_0(\mathbf{r}) + \int d^3r'\, w(\mathbf{r},\mathbf{r}')n_0(\mathbf{r}') + v_{xc}([n_0];\mathbf{r})$$

and

$$\begin{aligned} v_{xc}([n];\mathbf{r}) &= \frac{\delta E_{xc}[n]}{\delta n(\mathbf{r})} \\ &= \frac{\delta}{\delta n(\mathbf{r})}\left(F_L[n] - \frac{1}{2}\iint d^3r d^3r'\, n(\mathbf{r})w(\mathbf{r},\mathbf{r}')n(\mathbf{r}') - T_L[n]\right) \end{aligned}$$

The Kohn-Sham scheme II

Density of degen. K-S scheme:

$$n_0(\mathbf{r}) = \sum_{i=1}^{N} \gamma_i |\phi_{i,0}(\mathbf{r})|^2,$$

occupation numbers γ_i satisfy

$$\begin{aligned} \gamma_i = 1 &: \varepsilon_i < \mu \\ 0 \le \gamma_i \le 1 &: \varepsilon_i = \mu \\ \gamma_i = 0 &: \varepsilon_i > \mu \end{aligned}$$

and

$$\sum_{i=1}^{N} \gamma_i = N$$

Hartree-Fock equation:

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_0(\mathbf{r}) + \int d^3r' \, w(\mathbf{r},\mathbf{r}')n(\mathbf{r}')\right)\phi_k(\mathbf{r})$$
$$\underbrace{-\sum_{l=1}^{N}\int d^3r' \, \phi_l^*(\mathbf{r}')w(\mathbf{r},\mathbf{r}')\phi_k(\mathbf{r}')\phi_l(\mathbf{r})}_{\text{exchange term}} = \varepsilon_k\phi_k(\mathbf{r}),$$

Non-local exchange term (Pauli exclusion principle)

Kohn-Sham equation:

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + v_0(\mathbf{r}) + \int d^3r' \, w(\mathbf{r},\mathbf{r}')n(\mathbf{r}') + \underbrace{v_{xc}([n];\mathbf{r})}_{\text{exchange + correlation}}\right)\phi_k(\mathbf{r}) = \varepsilon_k\phi_k(\mathbf{r}),$$

Local exchange-correlation term

Exchange-correlation energy $=$ Exchange energy $+$ Correlation energy

$$E_{xc}[n] = E_x[n] + E_c[n]$$

From earlier:

$$E_{xc}[n] = F_L[n] - T_s[n] - \frac{1}{2}\iint d^3r d^3r' n(\mathbf{r}) w(\mathbf{r},\mathbf{r}') n(\mathbf{r}')$$

We want to show: $E_c[n] \leq 0$

Here we have (assume $F_L[n] = F_{LL}[n]$)

$$\begin{aligned} F_L[n] &\equiv \inf_{\Psi \to n} \Psi \hat{T} + \hat{W} \Psi \\ &= \Psi_n^{min} \hat{T} + \hat{W} \Psi_n^{min}, \end{aligned}$$

and

$$T_s[n] \equiv \inf_{\Psi \to n} \Psi \hat{T} \Psi = \Phi_n^{min} \hat{T} \Phi_n^{min},$$

Ψ = normalized, antisymm. N-particle wavefunction,
Φ_n^{min} lin. komb. of Slater determinants of

single-particle orbitals $\psi_i(r_j)$

Eq. (4.35) in J. M. Thijssen: *Computational Physics*:

$$\Phi_n^{min}\hat{W}\Phi_n^{min} = \frac{1}{2}\sum_{k,l}\Big[\iint d^3r d^3r' n(\mathbf{r})w(\mathbf{r},\mathbf{r}')n(\mathbf{r}') \\ - \iint d^3r d^3r' \psi_l^*(\mathbf{r})\psi_l(\mathbf{r}')w(\mathbf{r},\mathbf{r}')\psi_k^*(\mathbf{r}')\psi_k(\mathbf{r})\Big]$$

By definition,

$$E_x[n] \equiv -\frac{1}{2}\sum_{k,l}\iint d^3r d^3r' \psi_l^*(\mathbf{r})\psi_l(\mathbf{r}')w(\mathbf{r},\mathbf{r}')\psi_k^*(\mathbf{r}')\psi_k(\mathbf{r})$$

Using expressions from previous pages gives

$$\begin{aligned}
E_c[n] &= E_{xc}[n] - E_x[n] \\
&= F_L[n] - T_s[n] - \frac{1}{2}\iint d^3r d^3r' n(\mathbf{r}) w(\mathbf{r},\mathbf{r}') n(\mathbf{r}') \\
&+ \frac{1}{2}\sum_{k,l}\iint d^3r d^3r' \psi_l^*(\mathbf{r})\psi_l(\mathbf{r}') w(\mathbf{r},\mathbf{r}') \psi_k^*(\mathbf{r}')\psi_k(\mathbf{r}) \\
&= \Psi_n^{min}\hat{T} + \hat{W}\Psi_n^{min} - \Phi_n^{min}\hat{T} + \hat{W}\Phi_n^{min}
\end{aligned}$$

Since

$$\Psi_n^{min}\hat{T} + \hat{W}\Psi_n^{min} = \inf_{\Psi\to n} \Psi\hat{T} + \hat{W}\Psi,$$

we see that

$$E_c[n] \leq 0$$