

Slides from FYS4410 Lectures

Morten Hjorth-Jensen

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

Spring 2009

Introduction, Parallelization, MPI and Variational Monte Carlo

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

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)

Course overview, Computational aspects

- Search for minima in multidimensional spaces (conjugate gradient method) (project 1)
- Solutions of coupled differential equations for Hartree-Fock and density functional calculations. (project 2)
- Parallelization of quantum molecular dynamics code. (project 3)
- Object orientation (all projects)

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.
- 3 Aim of projects 1: understand how to simulate quantum mechanical systems with many interacting particles using variational Monte Carlo methods.

The methods of project 1, 2 and 3 are relevant for atomic, molecular, solid state, materials science, nanotechnology, quantum chemistry and nuclear physics.

Quantum Mechanical Methods and Systems

- 1 Project 2 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.

Quantum Mechanical Methods and Systems

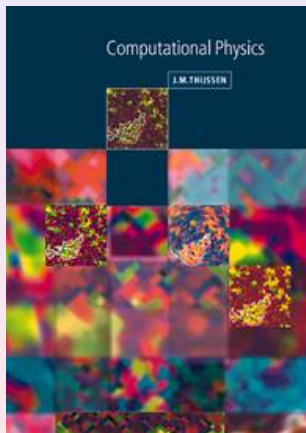
- 1 Project 3 is not yet determined, but will most likely deal with quantum molecular dynamics (QMD). The DFT results will be used to fit a semi-classical potential to be used in QMD calculations. QMD is covered by chapter 8-9 of Thijssen's book plus lecture notes by MHJ.

Projects, deadlines and oral exam

- 1 Deadline project 1: 16 March
- 2 Deadline project 2: 30 April
- 3 Deadline project 3: 31 May
- 4 Oral exam: week 24 (8-12 June)

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

- Lectures: Wednesday (12.15-14.00, room LilleFys) and Thursday (14.15-19, 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 all other information are on the official webpage.
- The thursday lectures will also be used to demonstrate algorithms etc, while the wednesday lectures will be more like classical blackboard sessions.
- 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.



J. M. Thijssen's text






- Computational Physics
- Chapters 3-6 and 8-9
- **see** <http://www.tn.tudelft.nl/tn/People/Staff/Thijssen/comphybook.html>

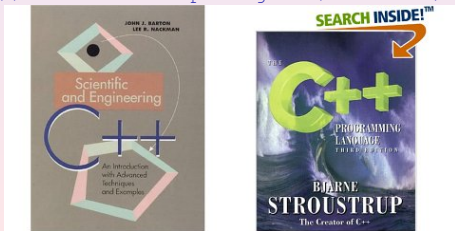


Gropp, Lusk and Sjellum

- 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.
-  H. P. Langtangen INF-VERK3830
<http://heim.ifi.uio.no/~hpl/INF-VERK4830/>
-  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

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

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

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

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

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

Hereafter we use natural units, viz. $\hbar = c = e = 1$, with e the elementary charge and 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 $k = 1.44 \text{ eVnm}$

Definitions and notations

We can rewrite this as

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

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

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

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

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

with

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

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

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

where β is the eigenvalue of \hat{P} . 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, resulting in the eigenvalue $\beta = -1$.

Definitions and notations

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

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

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

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

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

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

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}

$$[H_0, \mathcal{A}] = [H_1, \mathcal{A}] = 0. \quad (10)$$

Furthermore, \mathcal{A} satisfies

$$\mathcal{A}^2 = \mathcal{A}, \quad (11)$$

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

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

and rewrite Eq. (12) as

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

Definitions and notations

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

$$\int \Phi^* \hat{H}_1 \Phi d\tau = N! \int \Phi_H^* \mathcal{A} \hat{H}_2 \mathcal{A} \Phi_H d\tau,$$

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

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

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

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

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

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

Definitions and notations

Combining Eqs. (13) and (85) we obtain the energy functional

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

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 take to compute the above integral? Lifetime of the universe

$$T \approx 4.7 \times 10^{17} \text{s.}$$

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

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

where

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

are the coordinates and

$$\alpha_1, \dots, \alpha_A,$$

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

$$2^A \times \binom{A}{Z}$$

coupled second-order differential equations in $3A$ dimensions.

For a nucleus like ^{10}Be this number is **215040**. This is a truly 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.

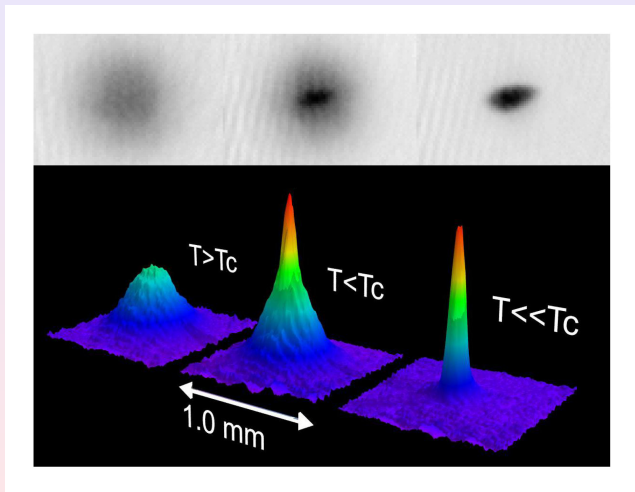
Pros and Cons of Monte Carlo

- 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 homogeneously 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 $\psi_T(\mathbf{R})$.

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

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 \mathbf{R} and variational parameters α and calculate $|\psi_T^\alpha(\mathbf{R})|^2$.
- Initialise the energy and the variance and start the Monte Carlo calculation (thermalize)
 - 1 Calculate a trial position $\mathbf{R}_p = \mathbf{R} + r * \text{step}$ 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.

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	σ/\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 hamiltonian on the wave function

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

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

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

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

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

and Schrödinger's equation reads

$$\hat{H}\psi = E\psi.$$

All observables are evaluated with respect to the probability distribution

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

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 $r_1 \rightarrow 0$.

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

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

For small values of r_1 , the terms which dominate are

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

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

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

Similarly, studying the case $r_{12} \rightarrow 0$ 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:
ostreamstream 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_energies;
// 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;
    }
}
```


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_l2,
        r_single_particle;
    double **r_plus, **r_minus;

    // allocate matrices which contain the position of the particles
    // the function matrix is defined in the program 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);
    }
}
```

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 implementations without change.

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

See chapter 7.7 of lecture notes for more details.

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.

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 processing 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** Â all involved processors have an independent memory address space. The user is responsible for partitioning 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.

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 processing) 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.

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 process within an MPI communicator
- MPI_Recv - receive a message from a particular process 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 of "
CALL MPI_FINALIZE(ierr)

END PROGRAM hello
```


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 ();
}
```

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 ();
```

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.

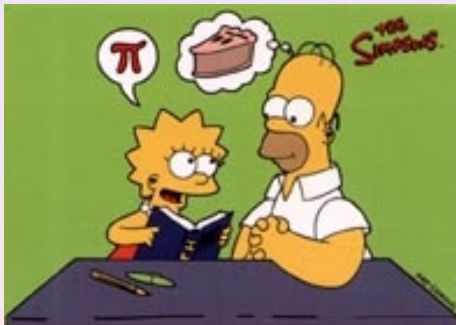
```
int MPI_Recv( void *buf, int count, MPI_Datatype da
             int source,
             int tag, MPI_Comm comm, MPI_Status *sta
```

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.



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)

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

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 height 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

```
1 //    Rectangle rule and numerical integration
2 using namespace std;
3 #include <mpi.h>
4 #include <iostream>

5 int main (int nargs, char* args[])
6 {
7     int numprocs, my_rank, i, n = 1000;
8     double local_sum, rectangle_sum, x, h;
9     //    MPI initializations
10    MPI_Init (&nargs, &args);
11    MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
12    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 corresponds 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

```
28     if (my_rank == 0) {
29         MPI_Status status;
30         rectangle_sum = local_sum;
31         for (i=1; i < numprocs; i++) {
32             MPI_Recv(&local_sum,1,MPI_DOUBLE,MPI_ANY_SOURCE,500,
33                     MPI_COMM_WORLD,&status);
34             rectangle_sum += local_sum;
35         }
36         cout << "Result: " << rectangle_sum << endl;
37     } else
38         MPI_Send(&local_sum,1,MPI_DOUBLE,0,500,MPI_COMM_WORLD);
39     // End MPI
40     MPI_Finalize ();
41     return 0;
42 }
```

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.

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
```

- 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 laptops 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.

`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`

Hardware

- 546 X2200m2, 7 X4200, Magnum 3456 IB switch
- 174 Dell 1425, + 20 HP DL385 servers
- EVA8K 120 TB storage
- Dual core Intel and AMD (\sim 2500 cores in total)
- Quad-core intel
- Infiniband and ethernet
- From 4GB to 64GB RAM
- Heterogenous cluster!

Software

- Batch system: Sun Grid Engine (SGE)
- Message Passing Interface (MPI):
 - Scampi
 - MPICH2
 - OpenMPI
- 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`, `qsub`
- `http://www.hpc.uio.no/index.php/FAQ#QUEUE_SYSTEM`

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://www.hpc.uio.no/index.php/FAQ#MODULES>

Interactively

```
# login to titan
$ ssh titan.uio.no
# ask for 4 cpus
$ qlogin -P fys4410 -pe mpi 4
# we want to use the scampi module
$ module load scampi
$ mkdir -p fys4410/mpiexample/
$ cd fys4410/mpiexample/
# download program5.cpp from the course pages:
$ wget http://tinyurl.com/39hrah/program5.cpp
# compile the program
$ mpic++ -O3 -o program5.x program5.cpp
# and execute it
$ mpiexec ./program5.x
Result: 3.14159
# finished, so logging out
$ exit
```

The job script

job.sge

```
#!/bin/sh
# 4 cpus with mpi (or other communication)
#$ -pe mpi 4
# 10 mins of walltime
#$ -l s_rt=0:10:0
# project fys4410
#$ -P fys4410
# we need 512 MB of memory per process
#$ -l s_vmem=512M
# name of job
#$ -N program5

source /site/bin/jobsetup

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

# start program
/site/bin/mpixec ./program5.x

#END OF SCRIPT
```

Submitting

```
# login to titan
$ ssh titan.uio.no
# we want to use the module scampi
$ module load scampi
$ cd fys4410/mpiexample/
# compile the program
$ mpic++ -O3 -o program5.x program5.cpp
# and submit it
$ qsub job.sge
$ exit
```

Checking execution

```
# check if job is running:
```

```
$ qstat -u mhjensen
```

```
job-ID  prior  name          user              state submit/start at   queue  
      slots ja-task-ID
```

```
379931 2.50073 rectangle mhjensen          r      10/17/2008 06:02:39 normal@compute  
      -13-6.local          4
```

```
# results are in rectangle.o379931
```

```
$ cat rectangle.o379931
```

```
Starting job 379931 on node compute-13-6.local at Thu Jan 17 06:02:39 CET 2008
```

```
Creating /work/379931.undefined.d
```

```
Result: 3.14159
```

```
Terminating job 379931 on compute-13-6.local at Thu Jan 17 06:02:44 CET 2008
```

```
Deleting /work/379931.undefined.d
```

Tips

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

Admonitions

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

Importance sampling, blocking and Slater determinants

- Wednesday:
- Repetition from previous weeks
- Discussion of importance sampling, lecture notes chapter 8 and 16. Part of this is material from FYS3150 (lecture notes chapter 8)
- Thursday:
- Discussion of importance sampling and Slater determinants, MPI and further computing topics.

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 drift term is

$$F = 2 \frac{1}{\Psi_T} \nabla \Psi_T$$

where Ψ_T is our trial wave function. 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

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

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

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.

Importance sampling: Transformation of Variables

The starting point is always the uniform distribution

$$p(x)dx = \begin{cases} dx & 0 \leq x \leq 1 \\ 0 & \text{else} \end{cases}$$

with $p(x) = 1$ and satisfying

$$\int_{-\infty}^{\infty} p(x)dx = 1.$$

All random number generators provided in the program library generate numbers in this domain.

When we attempt a transformation to a new variable $x \rightarrow y$ we have to conserve the probability

$$p(y)dy = p(x)dx,$$

which for the uniform distribution implies

$$p(y)dy = dx.$$

Transformation of Variables

Let us assume that $p(y)$ is a PDF different from the uniform PDF $p(x) = 1$ with $x \in [0, 1]$. If we integrate the last expression we arrive at

$$x(y) = \int_0^y p(y') dy',$$

which is nothing but the cumulative distribution of $p(y)$, i.e.,

$$x(y) = P(y) = \int_0^y p(y') dy'.$$

This is an important result which has consequences for eventual improvements over the brute force Monte Carlo.

Example 1, a general Uniform Distribution

Suppose we have the general uniform distribution

$$p(y)dy = \begin{cases} \frac{dy}{b-a} & a \leq y \leq b \\ 0 & \text{else} \end{cases}$$

If we wish to relate this distribution to the one in the interval $x \in [0, 1]$ we have

$$p(y)dy = \frac{dy}{b-a} = dx,$$

and integrating we obtain the cumulative function

$$x(y) = \int_a^y \frac{dy'}{b-a},$$

yielding

$$y = a + (b - a)x,$$

a well-known result!

Example 2, from Uniform to Exponential

Assume that

$$p(y) = e^{-y},$$

which is the exponential distribution, important for the analysis of e.g., radioactive decay. Again, $p(x)$ is given by the uniform distribution with $x \in [0, 1]$, and with the assumption that the probability is conserved we have

$$p(y)dy = e^{-y}dy = dx,$$

which yields after integration

$$x(y) = P(y) = \int_0^y \exp(-y')dy' = 1 - \exp(-y),$$

or

$$y(x) = -\ln(1 - x).$$

This gives us the new random variable y in the domain $y \in [0, \infty)$ determined through the random variable $x \in [0, 1]$ generated by our favorite random generator.

Example 2, from Uniform to Exponential

This means that if we can factor out $\exp(-y)$ from an integrand we may have

$$I = \int_0^{\infty} F(y) dy = \int_0^{\infty} \exp(-y) G(y) dy$$

which we rewrite as

$$\int_0^{\infty} \exp(-y) G(y) dy = \int_0^{\infty} \frac{dx}{dy} G(y) dy \approx \frac{1}{N} \sum_{i=1}^N G(y(x_i)),$$

where x_i is a random number in the interval $[0,1]$.

Note that in practical implementations, our random number generators for the uniform distribution never return exactly 0 or 1, but we may come very close. We should thus in principle set $x \in (0, 1)$.

Example 2, from Uniform to Exponential

The algorithm is rather simple. In the function which sets up the integral, we simply need the random number generator for the uniform distribution in order to obtain numbers in the interval $[0,1]$. We obtain y by taking the logarithm of $(1 - x)$. Our calling function which sets up the new random variable y may then include statements like

```
.....  
idum=-1;  
x=ran0(&idum);  
y=-log(1.-x);  
.....
```


Example 3, from Uniform to Normal

For the normal distribution, expressed here as

$$g(x, y) = \exp(-(x^2 + y^2)/2) dx dy.$$

it is rather difficult to find an inverse since the cumulative distribution is given by the error function $\text{erf}(x)$.

If we however switch to polar coordinates, we have for x and y

$$r = (x^2 + y^2)^{1/2} \quad \theta = \tan^{-1} \frac{x}{y},$$

resulting in

$$g(r, \theta) = r \exp(-r^2/2) dr d\theta,$$

where the angle θ could be given by a uniform distribution in the region $[0, 2\pi]$.

Following example 1 above, this implies simply multiplying random numbers $x \in [0, 1]$ by 2π .

Example 3, from Uniform to Normal

The variable r , defined for $r \in [0, \infty)$ needs to be related to random numbers $x' \in [0, 1]$. To achieve that, we introduce a new variable

$$u = \frac{1}{2}r^2,$$

and define a PDF

$$\exp(-u)du,$$

with $u \in [0, \infty)$. Using the results from example 2, we have that

$$u = -\ln(1 - x'),$$

where x' is a random number generated for $x' \in [0, 1]$. With

$$x = r\cos(\theta) = \sqrt{2u}\cos(\theta),$$

and

$$y = r\sin(\theta) = \sqrt{2u}\sin(\theta),$$

we can obtain new random numbers x, y through

$$x = \sqrt{-2\ln(1 - x')}\cos(\theta),$$

and

$$y = \sqrt{-2\ln(1 - x')}\sin(\theta),$$

with $x' \in [0, 1]$ and $\theta \in 2\pi[0, 1]$.

Example 3, from Uniform to Normal

A function which yields such random numbers for the normal distribution would include statements like

```
.....  
idum=-1;  
radius=sqrt(-2*ln(1.-ran0(idum)));  
theta=2*pi*ran0(idum);  
x=radius*cos(theta);  
y=radius*sin(theta);  
.....
```

Box-Mueller Method for Normal Deviates, gasdev.cpp under project 1

```
// random numbers with gaussian distribution
double gaussian_deviate(long * idum)
{
    static int iset = 0;
    static double gset;
    double fac, rsq, v1, v2;
    if ( idum < 0) iset =0;
    if (iset == 0) {
        do {
            v1 = 2.*ran0(idum) -1.0;
            v2 = 2.*ran0(idum) -1.0;
            rsq = v1*v1+v2*v2;
        } while (rsq >= 1.0 || rsq == 0.);
        fac = sqrt(-2.*log(rsq)/rsq);
        gset = v1*fac;
        iset = 1;
        return v2*fac;
    } else {
        iset =0;
        return gset;
    }
}
```

Importance Sampling

With the aid of the above variable transformations we address now one of the most widely used approaches to Monte Carlo integration, namely importance sampling. Let us assume that $p(y)$ is a PDF whose behavior resembles that of a function F defined in a certain interval $[a, b]$. The normalization condition is

$$\int_a^b p(y) dy = 1.$$

We can rewrite our integral as

$$I = \int_a^b F(y) dy = \int_a^b p(y) \frac{F(y)}{p(y)} dy.$$

Importance Sampling

Since random numbers are generated for the uniform distribution $p(x)$ with $x \in [0, 1]$, we need to perform a change of variables $x \rightarrow y$ through

$$x(y) = \int_a^y p(y') dy',$$

where we used

$$p(x)dx = dx = p(y)dy.$$

If we can invert $x(y)$, we find $y(x)$ as well.

Importance Sampling

With this change of variables we can express the integral of Eq. (133) as

$$I = \int p(y) \frac{F(y)}{p(y)} dy = \int \frac{F(y(x))}{p(y(x))} dx,$$

meaning that a Monte Carlo evaluation of the above integral gives

$$\int \frac{F(y(x))}{p(y(x))} dx = \frac{1}{N} \sum_{i=1}^N \frac{F(y(x_i))}{p(y(x_i))}.$$

The advantage of such a change of variables in case $p(y)$ follows closely F is that the integrand becomes smooth and we can sample over relevant values for the integrand. It is however not trivial to find such a function p . The conditions on p which allow us to perform these transformations are

- 1 p is normalizable and positive definite,
- 2 it is analytically integrable and
- 3 the integral is invertible, allowing us thereby to express a new variable in terms of the old one.

Importance Sampling

The algorithm for this procedure is

- Use the uniform distribution to find the random variable y in the interval $[0,1]$. $p(x)$ is a user provided PDF.
- Evaluate thereafter

$$I = \int F(x) dx = \int p(x) \frac{F(x)}{p(x)} dx,$$

by rewriting

$$\int p(x) \frac{F(x)}{p(x)} dx = \int \frac{F(x(y))}{p(x(y))} dy,$$

since

$$\frac{dy}{dx} = p(x).$$

- Perform then a Monte Carlo sampling for

$$\int \frac{F(x(y))}{p(x(y))} dy, \approx \frac{1}{N} \sum_{i=1}^N \frac{F(x(y_i))}{p(x(y_i))},$$

with $y_i \in [0, 1]$,

- Evaluate the variance

Demonstration of Importance Sampling

$$I = \int_0^1 F(x) dx = \int_0^1 \frac{1}{1+x^2} dx = \frac{\pi}{4}.$$

We choose the following PDF (which follows closely the function to integrate)

$$p(x) = \frac{1}{3}(4-2x) \quad \int_0^1 p(x) dx = 1,$$

resulting

$$\frac{F(0)}{p(0)} = \frac{F(1)}{p(1)} = \frac{3}{4}.$$

Check that it fulfils the requirements of a PDF. We perform then the change of variables (via the Cumulative function)

$$y(x) = \int_0^x p(x') dx' = \frac{1}{3}x(4-x),$$

or

$$x = 2 - (4 - 3y)^{1/2}$$

We have that when $y = 0$ then $x = 0$ and when $y = 1$ we have $x = 1$.

Simple Code

```
// evaluate the integral with importance sampling
for ( int i = 1; i <= n; i++){
    x = ran0(&idum); // random numbers in [0,1]
    y = 2 - sqrt(4-3*x); // new random numbers
    fy=3*func(y)/(4-2*y); // weighted function
    int_mc += fy;
    sum_sigma += fy*fy;
}
int_mc = int_mc/((double) n );
sum_sigma = sum_sigma/((double) n );
variance=(sum_sigma-int_mc*int_mc);
```

Test Runs and Comparison with Brute Force for

$$\pi = 3.14159$$

The suffix *cr* stands for the brute force approach while *is* stands for the use of importance sampling. All calculations use `ran0` as function to generate the uniform distribution.

N	I_{cr}	σ_{cr}	I_{is}	σ_{is}
10000	3.13395E+00	4.22881E-01	3.14163E+00	6.49921E-03
100000	3.14195E+00	4.11195E-01	3.14163E+00	6.36837E-03
1000000	3.14003E+00	4.14114E-01	3.14128E+00	6.39217E-03
10000000	3.14213E+00	4.13838E-01	3.14160E+00	6.40784E-03

However, it is unfair to study one-dimensional integrals with MC methods!

Multidimensional Integrals

When we deal with multidimensional integrals of the form

$$I = \int_0^1 dx_1 \int_0^1 dx_2 \dots \int_0^1 dx_d g(x_1, \dots, x_d),$$

with x_i defined in the interval $[a_i, b_i]$ we would typically need a transformation of variables of the form

$$x_i = a_i + (b_i - a_i)t_i,$$

if we were to use the uniform distribution on the interval $[0, 1]$. In this case, we need a Jacobi determinant (useful in point b of project 2)

$$\prod_{i=1}^d (b_i - a_i),$$

and to convert the function $g(x_1, \dots, x_d)$ to

$$g(x_1, \dots, x_d) \rightarrow g(a_1 + (b_1 - a_1)t_1, \dots, a_d + (b_d - a_d)t_d).$$

Example, six-dimensional integral (project 1, part a)

We first integrate in a brute force manner a six-dimensional integral which is used to determine the ground state correlation energy between two electrons in a helium atom. We will employ both Gauss-Legendre quadrature and Monte-Carlo integration. We assume that the wave function of each electron can be modelled like the single-particle wave function of an electron in the hydrogen atom. The single-particle wave function for an electron i in the 1s state is given in terms of a dimensionless variable (the wave function is not properly normalized)

$$\mathbf{r}_i = x_i \mathbf{e}_x + y_i \mathbf{e}_y + z_i \mathbf{e}_z,$$

as

$$\psi_{1s}(\mathbf{r}_i) = e^{-\alpha r_i},$$

where α is a parameter and

$$r_i = \sqrt{x_i^2 + y_i^2 + z_i^2}.$$

We will fix $\alpha = 2$, which should correspond to the charge of the helium atom $Z = 2$.

Switch to spherical coordinates

Useful to change to spherical coordinates

$$d\mathbf{r}_1 d\mathbf{r}_2 = r_1^2 dr_1 r_2^2 dr_2 d\cos(\theta_1) d\cos(\theta_2) d\phi_1 d\phi_2,$$

and

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\beta)}}$$

with

$$\cos(\beta) = \cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)\cos(\phi_1 - \phi_2)$$

How do I do importance sampling in spherical coordinates

- $r_{1,2} \in [0, \infty)$, here we use the mapping $r_{1,2} = -\ln(1 - ran)$ with $ran \in [0, 1]$, a uniform distribution point.
- $\theta_{1,2} \in [0, \pi]$, use mapping $\theta_{1,2} = \pi * ran$ with $ran \in [0, 1]$ a uniform distribution point.
- $\phi_{1,2} \in [0, 2\pi]$, use mapping $\phi_{1,2} = 2\pi * ran$ with $ran \in [0, 1]$ a uniform distribution point.
- Be careful with the integrand

$$\frac{\exp(-4(r_1 + r_2)) r_1^2 dr_1 r_2^2 dr_2 d\cos(\theta_1) d\cos(\theta_2) d\phi_1 d\phi_2}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\beta)}}$$

Example, six-dimensional integral

The ansatz for the wave function for two electrons is then given by the product of two 1s wave functions as

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

Note that it is not possible to find an analytic solution to Schrödinger's equation for two interacting electrons in the helium atom.

The integral we need to solve is the quantum mechanical expectation value of the correlation energy between two electrons, namely

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} d\mathbf{r}_1 d\mathbf{r}_2 e^{-2\alpha(r_1+r_2)} \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} = \frac{5\pi^2}{16^2} = 0.192765711. \quad (17)$$

Note that our wave function is not normalized. There is a normalization factor missing, but for this project we don't need to worry about that.

Example, six-dimensional integral, Gauss-Legendre

```
double *x = new double [N];
double *w = new double [N];
// set up the mesh points and weights
gauleg(a,b,x,w, N);

// evaluate the integral with the Gauss-Legendre method
// Note that we initialize the sum
double int_gauss = 0.;
for (int i=0;i<N;i++){
  for (int j = 0;j<N;j++){
    for (int k = 0;k<N;k++){
      for (int l = 0;l<N;l++){
        for (int m = 0;m<N;m++){
          for (int n = 0;n<N;n++){
            int_gauss+=w[i]*w[j]*w[k]*w[l]*w[m]*w[n]
              *int_function(x[i],x[j],x[k],x[l],x[m],x[n]);
          }}}}}
}
```

Example, six-dimensional integral, Gauss-Legendre

```
// this function defines the function to integrate
double int_function(double x1, double y1, double z1, double x2, double
{
    double alpha = 2.;
// evaluate the different terms of the exponential
    double exp1=-2*alpha*sqrt(x1*x1+y1*y1+z1*z1);
    double exp2=-2*alpha*sqrt(x2*x2+y2*y2+z2*z2);
    double deno=sqrt(pow((x1-x2),2)+pow((y1-y2),2)+pow((z1-z2),2));
    if(deno <pow(10.,-6.)) { return 0;}
    else return exp(exp1+exp2)/deno;
} // end of function to evaluate
```

Example, six-dimensional integral, brute force MC

```
double int_mc = 0.; double variance = 0.;
double sum_sigma= 0. ; long idum=-1 ;
double length=1.5; // we fix the max size of the box to L=3
double volume=pow((2*length),6.);

// evaluate the integral with importance sampling
for ( int i = 1; i <= n; i++){
// x[] contains the random numbers for all dimensions
  for (int j = 0; j< 6; j++) {
    x[j]=-length+2*length*ran0(&idum); // Maps U[0,1] to U[-L,L]
  }
  fx=brute_force_MC(x);
  int_mc += fx;
  sum_sigma += fx*fx;
}
int_mc = int_mc/((double) n );
sum_sigma = sum_sigma/((double) n );
variance=sum_sigma-int_mc*int_mc;
....
```

Example, six-dimensional integral, brute force MC

```
double brute_force_MC(double *x)
{
    double alpha = 2.;
    // evaluate the different terms of the exponential
    double exp1=-2*alpha*sqrt(x[0]*x[0]+x[1]*x[1]+x[2]*x[2]);
    double exp2=-2*alpha*sqrt(x[3]*x[3]+x[4]*x[4]+x[5]*x[5]);
    double deno=sqrt(pow((x[0]-x[3]),2)+pow((x[1]-x[4]),2)+pow((x[2]-x[5]),2));
    double value=exp(exp1+exp2)/deno;
    return value;
} // end function for the integrand
```

Example, six-dimensional integral, importance sampling

```
double int_mc = 0.; double variance = 0.;
double sum_sigma= 0. ; long idum=-1 ;
// The 'volume' contains 4 jacobideterminants(pi,pi,2pi,2pi) and a sca
double volume=4*pow(acos(-1.),4.)*1./16;
// evaluate the integral with importance sampling
for ( int i = 1; i <= n; i++){
    for (int j = 0; j < 2; j++) {
y=ran0(&idum);
x[j]=-0.25*log(1.-y);
    }
    for (int j = 2; j < 4; j++) {
x[j] = 2*acos(-1.)*ran0(&idum);
    }
    for (int j = 4; j < 6; j++) {
x[j] = acos(-1.)*ran0(&idum);
    }
fx=gaussian_MC(x);
    ....
}
```

Example, six-dimensional integral, importance sampling

```
// this function defines the integrand to integrate

double gaussian_MC(double *x)
{
double num=x[0]*x[0]*x[1]*x[1]*sin(x[4])*sin(x[5]);
double deno=sqrt(x[0]*x[0]+x[1]*x[1]-2*x[0]*x[1]*
    (sin(x[4])*sin(x[5])*cos(x[2]-x[3])+cos(x[4])*cos(x[5])));
return num/deno;
} // end function for the integrand
```

Test Runs and Comparison with Brute Force and Gauss-Legendre

The suffix *br* stands for the brute force approach while *is* stands for the use of importance sampling.

N	I_{br}	σ_{br}	time(s)	I_{is}	σ_{is}	time(s)
1E6	0.19238	3.85124E-4	0.6	0.19176	1.01515E-4	1.4
10E6	0.18607	1.18053E-4	6	0.192254	1.22430E-4	14
100E6	0.18846	4.37163E-4	57	0.192720	1.03346E-4	138
1000E6	0.18843	1.35879E-4	581	0.192789	3.28795E-5	1372

Gauss-Legendre results:

N	time(s)	I_n	$ I_n - I $
20	31	0.18047	1.123E-2
30	354	0.18501	7.76E-3
40	1999	0.18653	6.24E-3
50	7578	0.18722	5.54E-3

Importance sampling, blocking and Slater determinants

- Wednesday:
- Fokker-Planck and Langevin equation, importance sampling
- Thursday:
- Blocking and Slater determinants.

Importance sampling, blocking and Slater determinants

- Wednesday:
- Fokker-Planck and Langevin equation, importance sampling and blocking
- Thursday:
- Blocking and Slater determinants.

Why blocking?

Statistical analysis

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

Why blocking?

Statistical analysis

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

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:

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

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:

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

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.

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 :

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

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 :

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

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 $\text{Var}(X)$:

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

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:

$$\begin{aligned}\text{Cov}(X_i, X_j) &\equiv \langle (x_i - \langle x_i \rangle)(x_j - \langle x_j \rangle) \rangle \\ &= \int \cdots \int (x_i - \langle x_i \rangle)(x_j - \langle x_j \rangle) P(x_1, \dots, x_n) dx_1 \dots dx_n \quad (18)\end{aligned}$$

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 $C_{ij} = \text{Cov}(X_i, X_j)$, then the diagonal elements are just the familiar variances, $C_{ii} = \text{Cov}(X_i, X_i) = \text{Var}(X_i)$. 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 , ($i \neq j$):

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

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 $\text{Cov}(X_i, X_j) = 0$ ($i \neq j$).

Also useful for us is the covariance of linear combinations of stochastic variables. Let $\{X_i\}$ and $\{Y_j\}$ be two sets of stochastic variables. Let also $\{a_i\}$ and $\{b_j\}$ be two sets of scalars. Consider the linear combination:

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

By the linearity of the expectation value

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

Statistics and blocking

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

$$\text{Var}(U) = \sum_{i,j} a_i a_j \text{Cov}(X_i, X_j) \quad (19)$$

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:

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

$$\text{Var}\left(\sum_i a_i X_i\right) = \sum_i a_i^2 \text{Var}(X_i)$$

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:

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

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

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

Note that the sample variance is the sample covariance without the cross terms. In a similar manner as the covariance in eq. (18) 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 $\text{Var}(X)$ and covariance $\text{Cov}(X, Y)$.

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:

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

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.

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.

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_j can be associated with its own stochastic variable X_j . The stochastic variable \bar{X}_n for the sample mean \bar{x}_n is then just a linear combination, already familiar to us:

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

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

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

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

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

Statistics and blocking

It is generally not possible to express $p_{\bar{X}_n}(x)$ in a closed form given an arbitrary PDF p_X and a number n . But for the limit $n \rightarrow \infty$ 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, $p_{\bar{X}_n}(x)$ approaches a Gaussian distribution whose mean and variance equal the true mean and variance, μ_X and σ_X^2 , respectively:

$$\lim_{n \rightarrow \infty} p_{\bar{X}_n}(x) = \left(\frac{n}{2\pi \text{Var}(X)} \right)^{1/2} e^{-\frac{n(x - \bar{x}_n)^2}{2\text{Var}(X)}} \quad (20)$$

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

$$\text{err}_X^2 = \text{Var}(\bar{X}_n) = \frac{1}{n^2} \sum_{ij} \text{Cov}(X_i, X_j) \quad (21)$$

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 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. (21):

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

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 :

$$\begin{aligned}\text{Var}(X_i) &= \langle x_i - \langle x_i \rangle \rangle \approx \langle x_i - \bar{x}_n \rangle \\ &\approx \frac{1}{n} \sum_{k=1}^n (x_k - \bar{x}_n) \\ &= \text{Var}(x)\end{aligned}\tag{22}$$

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

$$\begin{aligned}\text{err}_X^2 &= \frac{1}{n^2} \sum_{ij} \text{Cov}(X_i, X_j) \\ &\approx \frac{1}{n^2} \sum_{ij} \frac{1}{n} \text{Cov}(x) = \frac{1}{n^2} n^2 \frac{1}{n} \text{Cov}(x) \\ &= \frac{1}{n} \text{Cov}(x)\end{aligned}\tag{23}$$

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:

$$\begin{aligned}\text{err}_X^2 &= \frac{1}{n^2} \sum_{ij} \text{Cov}(X_i, X_j) = \frac{1}{n^2} \sum_i \text{Var}(X_i) \\ &\approx \frac{1}{n^2} \sum_i \text{Var}(x) \\ &= \frac{1}{n} \text{Var}(x)\end{aligned}\tag{24}$$

where in the second step we have used eq. (22). 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.

For computational purposes one usually splits up the estimate of err_X^2 , given by eq. (23), into two parts:

$$\begin{aligned}\text{err}_X^2 &= \frac{1}{n}\text{Var}(x) + \frac{1}{n}(\text{Cov}(x) - \text{Var}(x)) \\ &= \frac{1}{n^2} \sum_{k=1}^n (x_k - \bar{x}_n)^2 + \frac{2}{n^2} \sum_{k < l} (x_k - \bar{x}_n)(x_l - \bar{x}_n)\end{aligned}\quad (25)$$

The first term is the same as the error in the uncorrelated case, eq. (24). 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.

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

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

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, $\text{Var}(x)$. If we divide f_d by $\text{Var}(x)$, we arrive at the so called *autocorrelation function*:

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

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. (25)) can now be written in terms of the autocorrelation function:

$$\begin{aligned}\text{err}_X^2 &= \frac{1}{n} \text{Var}(x) + \frac{2}{n} \cdot \text{Var}(x) \sum_{d=1}^{n-1} \frac{f_d}{\text{Var}(x)} \\ &= \left(1 + 2 \sum_{d=1}^{n-1} \kappa_d \right) \frac{1}{n} \text{Var}(x) \\ &= \frac{\tau}{n} \cdot \text{Var}(x)\end{aligned}\tag{26}$$

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*:

$$\tau = 1 + 2 \sum_{d=1}^{n-1} \kappa_d\tag{27}$$

Statistics and blocking

For a correlation free experiment, τ equals 1. From the point of view of eq. (26) 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:

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

To neglect the autocorrelation time τ will always cause our simple uncorrelated estimate of $\text{err}_X^2 \approx \text{Var}(x)/n$ 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

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

which can be rewritten as

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

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

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

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_j the j^{th} eigenvalue corresponding to the eigenvector $\hat{\mathbf{v}}_j$.

Time Auto-correlation Function

If we assume that λ_0 is the largest eigenvalue 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 probability $\hat{\mathbf{w}}(t)$ (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 \mathbf{m} being the vector whose elements are the values of \mathcal{M}_{μ} 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 $t \rightarrow \infty$ 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

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

then the exponential correlation time can be computed as the average

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

If the decay is exponential, then

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

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

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

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

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

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 $i = 1 \dots n_{blocks}$ 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

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

Example

- The files `vmc_para.cpp` and `vmc_blocking.cpp` contains 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

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

```
ostreamstream ost;  
ost << "blocks_rank" << my_rank << ".dat";  
blockfile.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);  
  
blockfile.write((char*)(all_energies+1),  
                number_cycles*sizeof(double));  
blockfile.close();
```

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();
}
```

Blocking

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

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;
}
```

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 arithmetic: 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);
}
```

Bootstrap and Jackknife Methods, formalities

A good reference is M. C. K. Yang and David H. Robinson, *Understanding and Learning Statistics by Computer*, (World Scientific, Singapore, 1986).

Simple example: suppose we want to estimate the mass of an elementary particle as predicted in a numerical simulation. The mass is obtained by fitting an exponential to a simulation data set as follows:

$$y(t) = a \exp(-mt)$$

where the data are given as a table of y values for integer values of t , as

$$\{y(0), y(1), y(2), \dots, y(L)\}.$$

Actually the simulation spits out a list of such values in one single measurement, runs for a while, and spits out another list, and so on. So our data set looks like

$$\{y_i(0), y_i(1), y_i(2), \dots, y_i(L)\},$$

where $i = 1, \dots, N$ labels a list of measurements.

Bootstrap and Jackknife Methods

We might think all we have to do is to take the raw data and construct means $\bar{y}(t)$ and standard errors $\sigma(t)$ at each time t and then do a standard least chi square fit. We would get the best values for the parameters a and m and we would get the errors from the error matrix. But we have a problem. The standard chi square fit assumes that the fluctuations in the data points are statistically independent. It turns out that with the numerical simulations (also often a problem with experimental data as well) the fluctuations in the data are correlated. That is, if $y(0)$ fluctuates upwards, chances are better that $y(1)$ also fluctuates upwards. So we can't use the standard formula for chi square. Now it is possible to modify the formula for chi square to take proper account of the correlations. But the analysis becomes much more involved, so one would like to develop more confidence in the resulting error in the mass parameter.

Bootstrap and Jackknife Methods, formalities

Starting from a sample of N measurements, the jackknife begins by throwing out the first measurement, leaving a jackknife data set of $N - 1$ “resampled” values. The statistical analysis is done on the reduced sample, giving a measured value of a parameter, say m_{J1} . Then a new resampling is done, this time throwing out the second measurement, and a new measured value of the parameter is obtained, say m_{J2} . The process is repeated for each set i in the sample, resulting in a set of parameter values $\{m_{Ji}, i = 1, \dots, N\}$. The standard error is given by the formula

$$\sigma_{\text{Jmean}}^2 = (N - 1) \sum_{i=1}^N (m_{Ji} - m)^2 / N$$

where m is the result of fitting the full sample.

Bootstrap and Jackknife Methods, formalities

The jackknife method is also capable of giving an estimate of sampling bias. We may have a situation in which a parameter estimate tends to come out on the high side (or low side) of its true value if a data sample is too small. Thus the estimate m derived from a fit to N data points may be higher (or lower) than the true value. When this happens, we might expect that removing a measurement, as we do in the jackknife, would enhance the bias. We measure this effect by comparing the mean of the jackknife values m_{Ji} , call it m_J , with the result m of fitting the full data set. If there is a difference, we can correct for the bias using

$$\tilde{m} = m - (N - 1)(m_J - m)$$

Bootstrap and Jackknife Methods, formalities

To see how the jackknife works, let us consider the much simpler problem of computing the mean and standard deviation of the mean of a random sample $\{x_i\}$. The conventional approach gives

$$\bar{x} = \sum_{j=1}^N x_j / N$$
$$\sigma_{\text{mean}}^2 = \sum_{j=1}^N (x_j - \bar{x})^2 / [N(N - 1)]$$

The jackknife approach computes the jackknife sample means

$$x_{Ji} = \sum_{j \neq i} x_j / (N - 1)$$

for $i = 1, \dots, N$. Then we compute the jackknife error in the mean, which is given by

$$\sigma_{J\text{mean}}^2 = (N - 1) \sum_{i=1}^N (x_{Ji} - \bar{x})^2 / N$$

Bootstrap and Jackknife Methods, formalities

Compare the placement of the factors of N and $N - 1$ here with the expression for σ_{mean} . The reason for the difference is that the jackknife sample means are distributed $N - 1$ times closer to the mean than the original values x_i , so we need a correction factor of $(N - 1)^2$. In fact for this simple example, it is easy to show that

$$x_{Ji} - \bar{x} = (\bar{x} - x_i)/(N - 1).$$

Consequently we can show trivially that

$$\sigma_{J\text{mean}} = \sigma_{\text{mean}}$$

so the jackknife procedure hasn't gained us anything in this simple case. But our example of determining the mass of an elementary particle is not so simple. The error estimate is found from Eq ([*]). This error estimate is not likely to be the same as the error obtained from a full correlated chi square analysis. However, we expect that in the limit of an infinitely large sample, both estimates should agree.

So if we get two error estimates and they don't agree, which should we believe? A conservative approach would take the larger of the two.

Conjugate gradient method and wave functions

- Wednesday:
- Conjugate gradient method (Numerical recipes chapter 10.7)
- Thursday:
- Conjugate gradient method
- Wave functions: Slater determinant and Jastrow part

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

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

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 *Insrch*. You can load down the package of programs from the webpage of the course, see under project 1. The package is called *NRcgm107.tar.gz* and contains the files *dfmin.c*, *Insrch.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 is 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

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

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 (E_i is the first derivative wrt to the variational parameter α_i)

$$\bar{E}_i = \left\langle \frac{\psi_i}{\psi} E_L + \frac{H\psi_i}{\psi} - 2\bar{E} \frac{\psi_i}{\psi} \right\rangle \quad (28)$$

$$= 2 \left\langle \frac{\psi_i}{\psi} (E_L - \bar{E}) \right\rangle \quad (\text{by Hermiticity}). \quad (29)$$

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

$$\begin{aligned} \bar{E}_{ij} = 2 & \left[\left\langle \left(\frac{\psi_{ij}}{\psi} + \frac{\psi_i \psi_j}{\psi^2} \right) (E_L - \bar{E}) \right\rangle \right. \\ & \left. - \left\langle \frac{\psi_i}{\psi} \right\rangle \bar{E}_j - \left\langle \frac{\psi_j}{\psi} \right\rangle \bar{E}_i + \left\langle \frac{\psi_i}{\psi} E_{L,j} \right\rangle \right]. \end{aligned} \quad (30)$$

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

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

Simple example and demonstration

The variance of the local energy,

$$\sigma^2 = \frac{\int d^3\mathbf{R} \psi^2 (E_L - \bar{E})^2}{\int d^3\mathbf{R} \psi^2} = \langle (E_L - \bar{E})^2 \rangle. \quad (31)$$

where $E_L = H\psi/\psi$ is the local energy, $\langle \cdot \rangle$ denotes a ψ^2 -weighted expectation value, and $\bar{E} = \langle E_L \rangle$ is the expectation value of the energy. The derivative of σ^2 with respect to the i^{th} parameter, c_i , is given by

$$\begin{aligned} (\sigma^2)_i &= 2 \left[\langle E_{L,i} (E_L - \bar{E}) \rangle \right. \\ &\quad \left. + \left\langle \left(\frac{\psi_i}{\psi} - \left\langle \frac{\psi_i}{\psi} \right\rangle \right) (E_L - \bar{E})^2 \right\rangle \right] \\ &= 2 \left[\langle E_{L,i} (E_L - \bar{E}) \rangle + \left\langle \frac{\psi_i}{\psi} E_L^2 \right\rangle - \left\langle \frac{\psi_i}{\psi} \right\rangle \langle E_L^2 \rangle \right. \\ &\quad \left. - 2\bar{E} \left\langle \frac{\psi_i}{\psi} (E_L - \bar{E}) \right\rangle \right], \end{aligned} \quad (32)$$

where subscript i denotes derivative with respect to c_i .

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)

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

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

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

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 minimum = " << 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
```

Conjugate gradient method, Wave functions and diffusion Monte Carlo

- Wednesday:
- Wave functions: Slater determinant and Jastrow part
- Thursday:
- Conjugate gradient method, demonstration of code
- Slater determinant code

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 \mathbf{D} is defined by the matrix elements

$$d_{ij} \equiv \phi_j(\mathbf{x}_i) \quad (33)$$

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}^{old} and the new suggested position by the vector \mathbf{r}^{new} .

The inverse of \mathbf{D} can be expressed in terms of its cofactors C_{ij} and its determinant $|\mathbf{D}|$:

$$d_{ij}^{-1} = \frac{C_{ji}}{|\mathbf{D}|} \quad (34)$$

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

Slater determinants

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

$$\sum_{k=1}^N d_{ik} d_{kj}^{-1} = \delta_{ij} \quad (35)$$

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

$$R \equiv \frac{|\mathbf{D}(\mathbf{r}^{\text{new}})|}{|\mathbf{D}(\mathbf{r}^{\text{old}})|} = \frac{\sum_{j=1}^N d_{ij}(\mathbf{r}^{\text{new}}) C_{ij}(\mathbf{r}^{\text{new}})}{\sum_{j=1}^N d_{ij}(\mathbf{r}^{\text{old}}) C_{ij}(\mathbf{r}^{\text{old}})} \quad (36)$$

Slater determinants

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

$$C_{ij}(\mathbf{r}^{\text{new}}) = C_{ij}(\mathbf{r}^{\text{old}}) \quad \forall j \in \{1, \dots, N\} \quad (37)$$

Slater determinants

Inserting this into the numerator of eq. (36) and using eq. (34) to substitute the cofactors with the elements of the inverse matrix, we get:

$$R = \frac{\sum_{j=1}^N d_{ij}(\mathbf{r}^{\text{new}}) C_{ij}(\mathbf{r}^{\text{old}})}{\sum_{j=1}^N d_{ij}(\mathbf{r}^{\text{old}}) C_{ij}(\mathbf{r}^{\text{old}})} = \frac{\sum_{j=1}^N d_{ij}(\mathbf{r}^{\text{new}}) d_{ji}^{-1}(\mathbf{r}^{\text{old}})}{\sum_{j=1}^N d_{ij}(\mathbf{r}^{\text{old}}) d_{ji}^{-1}(\mathbf{r}^{\text{old}})} \quad (38)$$

Slater determinants

Now by eq. (35) the denominator of the rightmost expression must be unity, so that we finally arrive at:

$$R = \sum_{j=1}^N d_{ij}(\mathbf{r}^{\text{new}}) d_{ji}^{-1}(\mathbf{r}^{\text{old}}) = \sum_{j=1}^N \phi_j(\mathbf{r}_i^{\text{new}}) d_{ji}^{-1}(\mathbf{r}^{\text{old}}) \quad (39)$$

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 $(\phi_1(\mathbf{r}_i^{\text{new}}), \dots, \phi_N(\mathbf{r}_i^{\text{new}}))$ of single particle wave functions evaluated at this new position with the i th column of the inverse matrix \mathbf{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 $\mathbf{D}^{-1}(\mathbf{x}^{\text{old}})$.

Slater determinants

If the new position \mathbf{r}^{new} is accepted, then the inverse matrix can be 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 i th column of \mathbf{D}^{-1} . For each column $j \neq i$, we first calculate the quantity:

$$S_j = (\mathbf{D}(\mathbf{r}^{\text{new}}) \times \mathbf{D}^{-1}(\mathbf{r}^{\text{old}}))_{ij} = \sum_{l=1}^N d_{il}(\mathbf{r}^{\text{new}}) D_{lj}^{-1}(\mathbf{r}^{\text{old}}) \quad (40)$$

The new elements of the j th column of \mathbf{D}^{-1} are then given by:

$$d_{kj}^{-1}(\mathbf{r}^{\text{new}}) = d_{kj}^{-1}(\mathbf{r}^{\text{old}}) - \frac{S_j}{R} d_{ki}^{-1}(\mathbf{r}^{\text{old}}) \quad \begin{array}{l} \forall k \in \{1, \dots, N\} \\ j \neq i \end{array} \quad (41)$$

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

$$d_{ki}^{-1}(\mathbf{r}^{\text{new}}) = \frac{1}{R} d_{ki}^{-1}(\mathbf{r}^{\text{old}}) \quad \forall k \in \{1, \dots, N\} \quad (42)$$

We see from these formulas that the time scaling of an update of \mathbf{D}^{-1} after changing one row of \mathbf{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:

$$\frac{\nabla_i |\mathbf{D}(\mathbf{r})|}{|\mathbf{D}(\mathbf{r})|} = \sum_{j=1}^N \nabla_i d_{ij}(\mathbf{r}) d_{ji}^{-1}(\mathbf{r}) = \sum_{j=1}^N \nabla_i \phi_j(\mathbf{r}_i) d_{ji}^{-1}(\mathbf{r}) \quad (43)$$

and

$$\frac{\nabla_i^2 |\mathbf{D}(\mathbf{r})|}{|\mathbf{D}(\mathbf{r})|} = \sum_{j=1}^N \nabla_i^2 d_{ij}(\mathbf{r}) d_{ji}^{-1}(\mathbf{r}) = \sum_{j=1}^N \nabla_i^2 \phi_j(\mathbf{r}_i) d_{ji}^{-1}(\mathbf{r}) \quad (44)$$

Slater determinants

Thus, to calculate all the derivatives of the Slater determinant, we only need the derivatives of the single particle wave functions ($\nabla_i \phi_j(\mathbf{r}_i)$ and $\nabla_i^2 \phi_j(\mathbf{r}_i)$) and the elements of the corresponding inverse Slater matrix ($\mathbf{D}^{-1}(\mathbf{r}_i)$). 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

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4, \alpha, \beta, \gamma, \delta) = \frac{1}{\sqrt{4!}} \begin{vmatrix} \psi_{100\uparrow}(\mathbf{r}_1) & \psi_{100\uparrow}(\mathbf{r}_2) & \psi_{100\uparrow}(\mathbf{r}_3) & \psi_{100\uparrow}(\mathbf{r}_4) \\ \psi_{100\downarrow}(\mathbf{r}_1) & \psi_{100\downarrow}(\mathbf{r}_2) & \psi_{100\downarrow}(\mathbf{r}_3) & \psi_{100\downarrow}(\mathbf{r}_4) \\ \psi_{200\uparrow}(\mathbf{r}_1) & \psi_{200\uparrow}(\mathbf{r}_2) & \psi_{200\uparrow}(\mathbf{r}_3) & \psi_{200\uparrow}(\mathbf{r}_4) \\ \psi_{200\downarrow}(\mathbf{r}_1) & \psi_{200\downarrow}(\mathbf{r}_2) & \psi_{200\downarrow}(\mathbf{r}_3) & \psi_{200\downarrow}(\mathbf{r}_4) \end{vmatrix}.$$

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

$$\begin{aligned}\Phi(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4, \alpha, \beta, \gamma, \delta) = & \text{Det} \uparrow (1, 2) \text{Det} \downarrow (3, 4) - \text{Det} \uparrow (1, 3) \text{Det} \downarrow (2, 4) \\ & - \text{Det} \uparrow (1, 4) \text{Det} \downarrow (3, 2) + \text{Det} \uparrow (2, 3) \text{Det} \downarrow (1, 4) - \text{Det} \uparrow (2, 4) \text{Det} \downarrow (1, 3) \\ & + \text{Det} \uparrow (3, 4) \text{Det} \downarrow (1, 2),\end{aligned}$$

where we have defined

$$\text{Det} \uparrow (1, 2) = \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_{100\uparrow}(\mathbf{r}_1) & \psi_{100\uparrow}(\mathbf{r}_2) \\ \psi_{200\uparrow}(\mathbf{r}_1) & \psi_{200\uparrow}(\mathbf{r}_2) \end{vmatrix},$$

and

$$\text{Det} \downarrow (3, 4) = \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_{100\downarrow}(\mathbf{r}_3) & \psi_{100\downarrow}(\mathbf{r}_4) \\ \psi_{200\downarrow}(\mathbf{r}_3) & \psi_{200\downarrow}(\mathbf{r}_4) \end{vmatrix}.$$

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

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4, \alpha, \beta, \gamma, \delta) \propto \text{Det} \uparrow (1, 2) \text{Det} \downarrow (3, 4),$$

or more generally as

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N) \propto \text{Det} \uparrow \text{Det} \downarrow,$$

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.

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 = (psils(argument[0])*psi2s(argument[1])
      -psils(argument[1])*psi2s(argument[0]))*
      (psils(argument[2])*psi2s(argument[3])
      -psils(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

It is possible to factorize the full determinant $|\mathbf{D}|$ into two smaller ones which each can be identified with \uparrow and \downarrow respectively:

$$|\mathbf{D}| = |\mathbf{D}|_{\uparrow} \cdot |\mathbf{D}|_{\downarrow} \quad (45)$$

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 $|\mathbf{D}|_{\uparrow}$ and $|\mathbf{D}|_{\downarrow}$:

$$\frac{|\mathbf{D}|^{\text{new}}}{|\mathbf{D}|^{\text{old}}} = \frac{|\mathbf{D}|_{\uparrow}^{\text{new}}}{|\mathbf{D}|_{\uparrow}^{\text{old}}} \cdot \frac{|\mathbf{D}|_{\downarrow}^{\text{new}}}{|\mathbf{D}|_{\downarrow}^{\text{old}}} \quad (46)$$

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:

$$\mathcal{O}_R(N) + \mathcal{O}_{\text{inverse}}(N^2) \quad (47)$$

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:

$$\mathcal{O}_R(N/2) + \mathcal{O}_{\text{inverse}}(N^2/4) \quad (48)$$

and the time scaling when the transitions for all N particles are put together:

$$\mathcal{O}_R(N^2/2) + \mathcal{O}_{\text{inverse}}(N^3/4) \quad (49)$$

which gives the same reduction as in the case of moving all particles at once.

Correlation factor

A correlation function brings about the correlational effects of the two-particle interaction that are not incorporated by the determinant. Usually it has a functional form that depends on the positions of multiplets of particles.

Although general n -particle correlations might certainly be worth examining, we limit ourselves to treat only two-particle correlations. These are expected to be the dominating ones in atomic and solid state physics. We define the correlation function to depend only on the distances between pairs of particles, the distance between particles i and j being (in three dimensions, $\mathbf{r}_i = (x_i, y_i, z_i)$) defined as:

$$r_{ij} = |\mathbf{r}_i - \mathbf{r}_j| = \sqrt{(\mathbf{r}_i - \mathbf{r}_j)^2} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad (50)$$

Correlation factor

We assume that two-body correlations are the essential ones (this needs to be changed if we do nuclear physics) and we restrict the correlation function to have the following functional form:

$$J = \prod_{i < j} g(r_{ij}) \quad (51)$$

where $g(r)$ is a function of r . As we seen previously, it is usual to approximate the correlation part with an exponential form due to the cusp condition, namely

$$J = \prod_{i < j} \exp(g(r_{ij})) = \exp\left(\sum_{i < j} g(r_{ij})\right) \quad (52)$$

In the literature, a correlation function of this form is usually called a Padé-Jastrow function. The necessary numerical evaluations of the correlation function are in general less expensive compared to those of the Slater determinant. It is however more difficult to optimize the code for the correlation function than for the determinant.

Correlation factor

From the expressions above we see that the total number of different relative distances r_{ij} that we have to maintain is $N(N - 1)/2$. Viewed as a matrix indexed by (i, j) , the set of all r_{ij} makes up the upper triangular part:

$$\mathbf{r} \equiv \begin{bmatrix} \cdot & & r_{12} & r_{13} & \dots & r_{1N} \\ & \cdot & & r_{23} & \dots & r_{2N} \\ & & & \cdot & \ddots & \vdots \\ & & & & \cdot & r_{(N-1)N} \\ & & & & & \cdot \end{bmatrix} \quad (53)$$

Correlation factor

The entries can be stored row-wise, starting from r_{12} and ending with $r_{(N-1)N}$. The corresponding function g_{ij} receives the argument r_{ij} can be stored in a similar row-wise manner

$$\mathbf{g} \equiv \begin{bmatrix} \cdot & g(r_{12}) & g(r_{13}) & \dots & g(r_{1N}) \\ & \cdot & g(r_{23}) & \dots & g(r_{2N}) \\ & & \cdot & \ddots & \vdots \\ & & & \cdot & g(r_{(N-1)N}) \\ & & & & \cdot \end{bmatrix} \quad (54)$$

For simplicity of notation, we define

$$g_{ij} \equiv g(r_{ij}) \quad (55)$$

Correlation factor

The quantity of interest is the ratio between the old and the new trial wave functions

$$R \equiv \frac{J^{\text{new}}}{J^{\text{old}}} \quad (56)$$

and

$$\frac{\nabla^2 J}{J}, \quad \frac{1}{J} \frac{\partial J}{\partial x_i} \quad (57)$$

for all dimensions (e.g. (x, y, z)) and with i running over all particles.

Correlation factor

Let us take a look at the calculation of the ratio R . When all particles are moved at once, all the g_{ij} have to be reevaluated and there is no way to introduce further optimization. So the evaluation of R scales as $\mathcal{O}(N^2)$. When moving only one particle at a time, say the k th, we only change $N - 1$ of the distances r_{ij} , namely those that have k as one of their indices. In consequence, just as many factors of J^{new} and J^{old} avoid cancellation

$$\frac{J^{\text{new}}}{J^{\text{old}}} = \prod_{i=1}^{k-1} \frac{g_{ik}^{\text{new}}}{g_{ik}^{\text{old}}} \prod_{i=k+1}^N \frac{g_{ki}^{\text{new}}}{g_{ki}^{\text{old}}} \quad (58)$$

Correlation factor

In particular for the exponential form

$$\frac{J^{\text{new}}}{J^{\text{old}}} = e^{\Delta J} \quad (59)$$

where

$$\Delta J = \sum_{i=1}^{k-1} (g_{ik}^{\text{new}} - g_{ik}^{\text{old}}) + \sum_{i=k+1}^N (g_{ki}^{\text{new}} - g_{ki}^{\text{old}}) \quad (60)$$

The time scaling is thus reduced to $\mathcal{O}(N)$. One needs to develop a special algorithm that iterates only through the elements of the upper triangular matrix \mathbf{g} that have k as an index.

Correlation factor

The ratios containing the gradient and the Laplacian (see eq. (57)) require a more involved analysis. We will take advantage of the following relations:

$$\frac{\partial}{\partial x_i} g_{ij} = -\frac{\partial}{\partial x_j} g_{ij} \quad (61a)$$

$$\frac{\partial^2}{\partial x_i^2} g_{ij} = \frac{\partial^2}{\partial x_j^2} g_{ij} \quad (61b)$$

Correlation factor

For the first derivative, again only $N - 1$ terms survive the ratio because the g -terms that are not differentiated cancel with their corresponding ones in the denominator:

$$\frac{1}{J} \frac{\partial J}{\partial x_k} = \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial g_{ki}}{\partial x_k} \quad (62)$$

For the exponential form, replacing g_{ij} by $\exp(g_{ij})$, we get:

$$\frac{1}{J} \frac{\partial J}{\partial x_k} = \sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} + \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_k} \quad (63)$$

Both scale as $\mathcal{O}(N)$. We can simplify these relations slightly by using eq. (61a) on the second sum of each expression.

Correlation factor

That way, all the derivatives act on the particle represented by the *second* index of g :

$$\frac{1}{J} \frac{\partial J}{\partial x_k} = \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial g_{ki}}{\partial x_i} \quad (64)$$

and for the exponential case:

$$\frac{1}{J} \frac{\partial J}{\partial x_k} = \sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_i} \quad (65)$$

Thus we only need to maintain the derivatives with respect to one of the particles of each g .

Correlation factor

The result for the double derivative is more complicated.

$$\begin{aligned} \frac{1}{J} \frac{\partial^2 J}{\partial x_k^2} &= \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial^2 g_{ik}}{\partial x_k^2} + \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial^2 g_{ki}}{\partial x_k^2} + \\ &+ \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} \left(\sum_{\substack{j=1 \\ j \neq i}}^{k-1} \frac{1}{g_{jk}} \frac{\partial g_{jk}}{\partial x_k} + \sum_{\substack{j=k+1 \\ j \neq i}}^N \frac{1}{g_{kj}} \frac{\partial g_{kj}}{\partial x_k} \right) \\ &+ \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial g_{ki}}{\partial x_k} \left(\sum_{\substack{j=1 \\ j \neq i}}^{k-1} \frac{1}{g_{jk}} \frac{\partial g_{jk}}{\partial x_k} + \sum_{\substack{j=k+1 \\ j \neq i}}^N \frac{1}{g_{kj}} \frac{\partial g_{kj}}{\partial x_k} \right) \end{aligned}$$

Correlation factor

For the full Laplacian we must sum together for all particles k and all dimensions (e.g. (x, y, z)). Let us first apply eqs. (61a) and (61b) in the same manner as we did for eqs. (64) and (65), switching the partial derivatives from the first to the second index where necessary. Furthermore, at close inspection it turns out that when summing over all k , the two first terms involving double derivatives in the above expression become equal.

Correlation factor

By this we get (for one dimension):

$$\begin{aligned} \left[\frac{\nabla^2 J}{J} \right]_x &= 2 \sum_{k=1}^N \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial^2 g_{ik}}{\partial x_k^2} \\ &+ \sum_{k=1}^N \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} \left(\sum_{\substack{j=1 \\ j \neq i}}^{k-1} \frac{1}{g_{jk}} \frac{\partial g_{jk}}{\partial x_k} - \sum_{\substack{j=k+1 \\ j \neq i}}^N \frac{1}{g_{kj}} \frac{\partial g_{kj}}{\partial x_j} \right) \\ &- \sum_{k=1}^N \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial g_{ki}}{\partial x_i} \left(\sum_{\substack{j=1 \\ j \neq i}}^{k-1} \frac{1}{g_{jk}} \frac{\partial g_{jk}}{\partial x_k} - \sum_{\substack{j=k+1 \\ j \neq i}}^N \frac{1}{g_{kj}} \frac{\partial g_{kj}}{\partial x_j} \right) \end{aligned}$$

Correlation factor

We are still able to include one additional simplification. Consider the second and the third line, consisting of the first derivatives. For each value of the index i , the index j runs over all values both smaller and greater than i . In total we therefore get a double-counting of the cross multiplied terms. What we ought to do is to restrict the inner sums to run over $j > i$ and multiply the result by 2.

Correlation factor

After all this deliberation we finally arrive at:

$$\begin{aligned} \left[\frac{\nabla^2 J}{J} \right]_x &= 2 \sum_{k=1}^N \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial^2 g_{ik}}{\partial x_k^2} \\ &+ 2 \sum_{k=1}^N \sum_{i=1}^{k-1} \frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} \left(\sum_{j=i+1}^{k-1} \frac{1}{g_{jk}} \frac{\partial g_{jk}}{\partial x_k} - \sum_{j=k+1}^N \frac{1}{g_{kj}} \frac{\partial g_{kj}}{\partial x_j} \right) \\ &+ 2 \sum_{k=1}^N \sum_{i=k+1}^N \frac{1}{g_{ki}} \frac{\partial g_{ki}}{\partial x_i} \left(\sum_{j=i+1}^N \frac{1}{g_{kj}} \frac{\partial g_{kj}}{\partial x_j} \right) \end{aligned} \quad (66)$$

Correlation factor

For the exponential form, things turn out slightly simpler. First of all it is useful to notice that:

$$\frac{\nabla^2 J}{J} = \left(\nabla^2 \sum g_{ij} \right) + \left(\nabla \sum g_{ij} \right)^2 \quad (67)$$

where the sums go over all allowed i and j . Let us first consider the Laplacian term:

$$\left(\nabla^2 \sum g_{ij} \right) = \sum_{k=1}^N \nabla_k^2 \sum_{i < j} g_{ij} = \sum_{k=1}^N \nabla_k^2 \left(\sum_{i=1}^{k-1} g_{ik} + \sum_{i=k+1}^N g_{ki} \right) \quad (68)$$

Correlation factor

The ∇_k^2 is just a partial Laplacian over all coordinates of the k th particle. This expression can be split up into separate terms for each dimension. For clarity we consider just one of those, denoted by x , and apply eq. (61b) as before on the second sum, to switch the derivative from the first to the second index of g_{ik} :

$$\left(\nabla^2 \sum g_{ij}\right)_x = \sum_{k=1}^N \left(\sum_{i=1}^{k-1} \frac{\partial^2 g_{ik}}{\partial x_k^2} + \sum_{i=k+1}^N \frac{\partial^2 g_{ki}}{\partial x_i^2} \right) \quad (69)$$

Correlation factor

As we noticed earlier on with the Laplacian, it turns out that the two resulting double sums are actually equal. We therefore arrive at:

$$\left(\nabla^2 \sum g_{ij}\right)_x = 2 \sum_{k=1}^N \sum_{i=1}^{k-1} \frac{\partial^2 g_{ik}}{\partial x_k^2} \quad (70)$$

Now consider the square of the gradient. Again the dimensions are separable and we consider only one:

$$\left(\nabla \sum g_{ij}\right)_x^2 = \sum_{k=1}^N \left(\frac{\partial}{\partial x_k} \sum_{i < j} g_{ij} \right)^2 = \sum_{k=1}^N \left(\sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_i} \right)^2 \quad (71)$$

In the second of the two inner sums, we have again used eq. (61a) to switch the index of the partial derivative.

Finally we can sum up the result for the Laplacian ratio of the exponential form (for one dimension):

$$\left[\frac{\nabla^2 J}{J} \right]_x = 2 \sum_{k=1}^N \sum_{i=1}^{k-1} \frac{\partial^2 g_{ik}}{\partial x_k^2} + \sum_{k=1}^N \left(\sum_{i=1}^{k-1} \frac{\partial g_{ik}}{\partial x_k} - \sum_{i=k+1}^N \frac{\partial g_{ki}}{\partial x_i} \right)^2 \quad (72)$$

Correlation factor

By now we understand that in order to calculate the desired ratios in Eq. (57) we only need to maintain quantities of the form:

$$\frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} - \frac{1}{g_{ik}} \frac{\partial^2 g_{ik}}{\partial x_k^2} \quad \forall i < k \quad (73)$$

for all dimensions with the index k running over all particles (for the specific case of J having the exponential form, we only need the partial derivatives without the g -term in the denominators).

Correlation factor

To evaluate the desired ratios the task amounts to assembling the partial ratios according to the various expressions given so far. And since the same basic functionality seems to require slightly different calculation schemes we create separate classes for the generic and the exponential case. In addition, since the ratio R (see Eq. (56)) is evaluated differently whether only one particle is moved or all, we have correspondingly made two versions of both the above mentioned classes. This way the user herself can choose the most efficient evaluation schemes by deciding which of the available correlation classes to use.

Correlation factor

One can treat all the required partial ratios to be stored separately for each differentiated dimension in the familiar form of an upper triangular matrix. As an example, the first derivatives in the x -dimension can be estimated via

$$\begin{bmatrix} \cdot & \frac{1}{g_{12}} \frac{\partial g_{12}}{\partial x_2} & \frac{1}{g_{13}} \frac{\partial g_{13}}{\partial x_3} & \cdots & \frac{1}{g_{1N}} \frac{\partial g_{1N}}{\partial x_N} \\ & \cdot & \frac{1}{g_{23}} \frac{\partial g_{23}}{\partial x_3} & \cdots & \frac{1}{g_{2N}} \frac{\partial g_{2N}}{\partial x_N} \\ & & \cdot & \ddots & \vdots \\ & & & \cdot & \frac{1}{g_{(N-1)N}} \frac{\partial g_{(N-1)N}}{\partial x_N} \\ & & & & \cdot \end{bmatrix} \quad (74)$$

and similarly for the second derivatives and all the other dimensions.

Correlation factor

A term from each \mathbf{g} , \mathbf{g}^+ and \mathbf{g}^- is then enough to evaluate one partial ratio. A first derivative partial ratio is via the three-point formula

$$\frac{1}{g_{ik}} \frac{\partial g_{ik}}{\partial x_k} = \frac{1}{g_{ik}} \left(\frac{g_{ik}^+ - g_{ik}^-}{2h} \right) + \mathcal{O}(h^2) \quad (77)$$

and a second derivative partial ratio:

$$\frac{1}{g_{ik}} \frac{\partial^2 g_{ik}}{\partial x_k^2} = \frac{1}{g_{ik}} \left(\frac{g_{ik}^+ - 2g_{ik} + g_{ik}^-}{2h^2} \right) + \mathcal{O}(h^2) \quad (78)$$

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 can be 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 \mathbf{A} is non-singular, then the LU factorization is unique and the determinant is given by

$$\det\{\mathbf{A}\} = c_{11}c_{22} \dots c_{nn}.$$

Conjugate gradient method, Wave functions and diffusion Monte Carlo

- Wednesday:
- Wave functions: Slater determinant and Jastrow part.
- Thursday:
- Summary of project 1
- Discussion of how to implement the Slater determinant and correlation part and the conjugate gradient method.

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

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

When you use the function `dfpmin`, it is useful to test your derivatives against this function.

Structuring the code

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

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

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. You can choose whether you wish to implement this or not.

Structuring the code

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

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

with α and β as variational parameters.

The local energy is for this case

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

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 (see slides on the correlation function and the Slater determinant). You should try the same for the beryllium atom.

Structuring the code

We have defined the Jastrow factor on exponential form

$$J(r_{ij}) = e^{\sum_{i<j} g(r_{ij})},$$

and that the Slater determinant is given by a product of single-particle wave function

$$\phi_\gamma(\mathbf{r}_i).$$

The first derivative is

$$\nabla_k \Psi_T(\mathbf{R}) = \nabla_k \phi(\mathbf{r}_k) \left[\prod_{i \neq k} \phi(\mathbf{r}_i) \right] e^{\sum_{i<j} g(r_{ij})} + \prod_i \phi(\mathbf{r}_i) e^{\sum_{i<j} g(r_{ij})} \sum_{j \neq k} \nabla_k g(r_{ij})$$

The second derivative entering the local energy is

$$\begin{aligned} \frac{1}{\Psi_T(\mathbf{R})} \nabla_k^2 \Psi_T(\mathbf{R}) &= \frac{\nabla_k^2 \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} + \frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} \left(\sum_{j \neq k} \frac{\mathbf{r}_k}{r_k} g'(r_{ij}) \right) + \\ &\sum_{ij \neq k} \frac{(\mathbf{r}_k - \mathbf{r}_i)(\mathbf{r}_k - \mathbf{r}_j)}{r_{ki} r_{kj}} g'(r_{ki}) g'(r_{kj}) + \sum_{j \neq k} \left(g''(r_{kj}) + \frac{2}{r_{kj}} g'(r_{kj}) \right) \end{aligned}$$

Structuring the code

Note well that for example in the expression for the first derivative is

$$\nabla_k \Psi_T(\mathbf{R}) = \nabla_k \phi(\mathbf{r}_k) \left[\prod_{i \neq k} \phi(\mathbf{r}_i) \right] e^{\sum_{i < j} u(r_{ij})} + \prod_i \phi(\mathbf{r}_i) e^{\sum_{i < j} u(r_{ij})} \sum_{j \neq k} \nabla_k u(r_{ij})$$

you need to change it so that you have a Slater determinant!

Structuring the code

Note well that for example in the expression for the first derivative is

$$\nabla_k \Psi_T(\mathbf{R}) = \nabla_k \phi(\mathbf{r}_k) \left[\prod_{i \neq k} \phi(\mathbf{r}_i) \right] e^{\sum_{i < j} u(r_{ij})} + \prod_i \phi(\mathbf{r}_i) e^{\sum_{i < j} u(r_{ij})} \sum_{j \neq k} \nabla_k u(r_{ij})$$

you need to change it so that you have a Slater determinant!

Bringing it all together, Metropolis test

In the Metropolis algorithm we need to compute the ratio (without importance sampling first) between the new and the old wave functions. As a generic example, let the wave function Ψ consist of the two factors ψ and ϕ :

$$\Psi = \psi \cdot \phi$$

To calculate for example the ratio needed by the Metropolis algorithm we can take advantage of the factorized form to group together the factors belonging to ψ and ϕ :

$$\frac{\Psi_{\text{new}}}{\Psi_{\text{old}}} = \frac{\psi_{\text{new}}}{\psi_{\text{old}}} \cdot \frac{\phi_{\text{new}}}{\phi_{\text{old}}}$$

Bringing it all together, Metropolis test

We need a good algorithm for the determinantal ratio

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

Bringing it all together, Metropolis test

If the new position \mathbf{r}^{new} is accepted, then the inverse matrix can be 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 i th column of \mathbf{D}^{-1} . For each column $j \neq i$, we first calculate the quantity:

$$S_j = (\mathbf{D}(\mathbf{r}^{\text{new}}) \times \mathbf{D}^{-1}(\mathbf{r}^{\text{old}}))_{ij} = \sum_{l=1}^N d_{il}(\mathbf{r}^{\text{new}}) D_{lj}^{-1}(\mathbf{r}^{\text{old}}) \quad (79)$$

The new elements of the j th column of \mathbf{D}^{-1} are then given by:

$$d_{kj}^{-1}(\mathbf{r}^{\text{new}}) = d_{kj}^{-1}(\mathbf{r}^{\text{old}}) - \frac{S_j}{R} d_{ki}^{-1}(\mathbf{r}^{\text{old}}) \quad \begin{array}{l} \forall k \in \{1, \dots, N\} \\ j \neq i \end{array} \quad (80)$$

Bringing it all together, Metropolis test

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

$$d_{ki}^{-1}(\mathbf{r}^{\text{new}}) = \frac{1}{R} d_{ki}^{-1}(\mathbf{r}^{\text{old}}) \quad \forall k \in \{1, \dots, N\} \quad (81)$$

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

Bringing it all together, Metropolis test

The final part is then the correlation factor which reads

$$\frac{J^{\text{new}}}{J^{\text{old}}} = e^{\Delta J} \quad (82)$$

where

$$\Delta J = \sum_{i=1}^{k-1} (g_{ik}^{\text{new}} - g_{ik}^{\text{old}}) + \sum_{i=k+1}^N (g_{ki}^{\text{new}} - g_{ki}^{\text{old}}) \quad (83)$$

Bringing it all together, quantum force

The quantum force used in importance sampling takes the form

$$\frac{\nabla\Psi}{\Psi} = \frac{\nabla\psi}{\psi} + \frac{\nabla\phi}{\phi}$$

We need the derivative of the Jastrow factor and the determinant. The derivative of the determinant is

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

Bringing it all together, quantum force

The general derivative formula of the Jastrow factor is

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

However, with our

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

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

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

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

Bringing it all together, Local energy

The local energy

$$E_L = \frac{\nabla^2 \Psi}{\Psi} - V(\mathbf{r})$$

where V is typically a two-particle interaction which normally does not act on the wave function. We can then focus on the kinetic energy part only of the local energy

$$\frac{\nabla^2 \Psi}{\Psi} = \frac{\nabla^2 \psi}{\psi} + \frac{\nabla^2 \phi}{\phi} + 2 \left(\frac{\nabla \psi}{\psi} \cdot \frac{\nabla \phi}{\phi} \right)$$

Bringing it all together, Local energy

The second derivative of the Slater determinant is

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

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

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

But we have a simple form for the function, namely

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

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

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

Bringing it all together, Local energy

Using

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

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

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

Important feature

For the correlation part

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

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 we did for helium)

$$a = 1/2.$$

Problems with neon for VMC

In the standard textbook case one uses spherical coordinates in order to get the hydrogen-like wave functions

$$x = r \sin \theta \cos \phi,$$

$$y = r \sin \theta \sin \phi,$$

and

$$z = r \cos \theta.$$

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

$$Y_{lm_l}(\theta, \phi) = \sin^{|m_l|}(\theta) \times (\text{polynom}(\cos\theta)) \exp(im_l\phi),$$

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

$$S_{l0} = \sqrt{\frac{4\pi}{2l+1}} \mathcal{Y}_{l0}(\mathbf{r}),$$

$$S_{lm_l} = (-1)^{m_l} \sqrt{\frac{8\pi}{2l+1}} \operatorname{Re} \mathcal{Y}_{l0}(\mathbf{r}),$$

$$S_{lm_l} = (-1)^{m_l} \sqrt{\frac{8\pi}{2l+1}} \operatorname{Im} \mathcal{Y}_{l0}(\mathbf{r}),$$

for $m_l > 0$.

Problems with neon for VMC

The lowest-order real solid harmonics are listed in here

Real Solid Harmonics

$m_l \setminus l$	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				$\frac{1}{2}\sqrt{\frac{5}{2}}(3x^2 - y^2)y$

Testing the Slater determinant for beryllium

It is useful to run first the beryllium case with

- No Jastrow factor
- Test your algorithm for the determinant for some selected points by comparing with the analytic expressions below.
- Compute also the derivatives (first and second) analytically with the expressions below and compare these results with your code.

The expression to test is

$$Det \uparrow (1, 2) Det \downarrow (3, 4)$$

where we have defined

$$Det \uparrow (1, 2) = \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_{100\uparrow}(\mathbf{r}_1) & \psi_{100\uparrow}(\mathbf{r}_2) \\ \psi_{200\uparrow}(\mathbf{r}_1) & \psi_{200\uparrow}(\mathbf{r}_2) \end{vmatrix},$$

and

$$Det \downarrow (3, 4) = \frac{1}{\sqrt{2}} \begin{vmatrix} \psi_{100\downarrow}(\mathbf{r}_3) & \psi_{100\downarrow}(\mathbf{r}_4) \\ \psi_{200\downarrow}(\mathbf{r}_3) & \psi_{200\downarrow}(\mathbf{r}_4) \end{vmatrix}.$$

For the Jastrow part, you can check your code against the closed form expressions for a simpler factor, say $e^{\beta r}$

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 expectation 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 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 minimum = " << 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 .

Hartree-Fock theory and Density functional theory

- Wednesday:
- Ground state energy of a single Slater determinant (repetition from the beginning of the semester). Examples from helium and beryllium atoms.
- Repetition of the variational principles and derivation of the Hartree-Fock equations
- Thursday:
- Derivation of the Hartree-Fock equations
- Project 1: Discussion of how to implement the Slater determinant and correlation part and the conjugate gradient method.

Project 2 is presented Wednesday 25 March. The material discussed for the rest of the course is also covered by Thijssen's book chapters 4-6 and 8-9.

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 unknown 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

$$\left[\frac{\partial E[\Phi(\alpha)]}{\partial \alpha} \right]_{\alpha=0} = 0.$$

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

$$\int_a^b \frac{\partial f}{\partial \Phi_x} \frac{d(\eta(x))}{dx} dx = \eta(x) \frac{\partial f}{\partial \Phi_x} \Big|_a^b - \int_a^b \eta(x) \frac{d}{dx} \frac{\partial f}{\partial \Phi_x} dx,$$

and since the first term disappears 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 variables.

Lagrangian Multipliers

The use of so-called Lagrangian multipliers is an alternative technique when the elimination of variables is inconvenient 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 multiple of $d\phi$, viz. $\lambda d\phi$, resulting in

$$df + \lambda d\phi = \left(\frac{\partial f}{\partial x} + \lambda \frac{\partial \phi}{\partial x}\right) dx + \left(\frac{\partial f}{\partial y} + \lambda \frac{\partial \phi}{\partial y}\right) dy + \left(\frac{\partial f}{\partial z} + \lambda \frac{\partial \phi}{\partial z}\right) dz = 0.$$

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

$$E = \int dx dy dz \psi^*(x, y, z) \hat{H} \psi(x, y, z),$$

with the constraint

$$\int dx dy dz \psi^*(x, y, z) \psi(x, y, z) = 1,$$

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 ,

$$\int dx dy dz \psi^* \left(-\frac{1}{2} \nabla^2 \right) \psi dx dy dz = \psi^* \nabla \psi + \int dx dy dz \frac{1}{2} \nabla \psi^* \nabla \psi.$$

Inserting this expression into the expectation value for the energy and taking the variational minimum we obtain

$$\delta E = \delta \left\{ \int dx dy dz \left(\frac{1}{2} \nabla \psi^* \nabla \psi + V \psi^* \psi \right) \right\} = 0.$$

Variational Calculus and Lagrangian Multiplier

The constraint appears in integral form as

$$\int dx dy dz \psi^* \psi = \text{constant},$$

and multiplying with a Lagrangian multiplier λ and taking the variational minimum we obtain the final variational equation

$$\delta \left\{ \int dx dy dz \left(\frac{1}{2} \nabla \psi^* \nabla \psi + V \psi^* \psi - \lambda \psi^* \psi \right) \right\} = 0.$$

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.

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

$$\hat{H} = \hat{H}_1 + \hat{H}_2 = \sum_{i=1}^N \hat{h}_i + \sum_{i<j=1}^N \frac{1}{r_{ij}},$$

$$\hat{h}_i = -\frac{1}{2} \nabla_i^2 - \frac{Z}{r_i}.$$

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 $d\tau = d\mathbf{r}_1 d\mathbf{r}_2 \dots d\mathbf{r}_N$.

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 electrons. 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}

$$[H_1, \mathcal{A}] = [H_2, \mathcal{A}] = 0.$$

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

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

is readily reduced to

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

which can be rewritten as

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{i=1}^N \sum_P (-1)^P \int \Phi_H^* \hat{h}_i P \Phi_H d\tau.$$

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

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{i=1}^N \int \Phi_H^* \hat{h}_i \Phi_H d\tau.$$

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

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{\mu=1}^N \int \psi_{\mu}^*(\mathbf{r}_i) \hat{h}_i \psi_{\mu}(\mathbf{r}_i) d\mathbf{r}_i,$$

or just as

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \sum_{\mu=1}^N \langle \mu | h | \mu \rangle.$$

Finding the Hartree-Fock functional $E[\Phi]$

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

$$\int \Phi^* \hat{H}_2 \Phi d\tau = N! \int \Phi_H^* \mathcal{A} \hat{H}_2 \mathcal{A} \Phi_H d\tau,$$

which reduces to

$$\int \Phi^* \hat{H}_2 \Phi d\tau = \sum_{i < j = 1}^N \sum_P (-1)^P \int \Phi_H^* \frac{1}{r_{ij}} P \Phi_H d\tau,$$

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

$$\int \Phi^* \hat{H}_2 \Phi d\tau = \sum_{i < j = 1}^N \int \Phi_H^* \frac{1}{r_{ij}} (1 - P_{ij}) \Phi_H d\tau.$$

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

$$\int \Phi^* \hat{H}_2 \Phi d\tau = \frac{1}{2} \sum_{\mu=1}^N \sum_{\nu=1}^N \left[\int \psi_{\mu}^*(\mathbf{r}_i) \psi_{\nu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{\mu}(\mathbf{r}_i) \psi_{\nu}(\mathbf{r}_j) d\mathbf{r}_i d\mathbf{r}_j \right. \\ \left. - \int \psi_{\mu}^*(\mathbf{r}_i) \psi_{\nu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{\nu}(\mathbf{r}_i) \psi_{\mu}(\mathbf{r}_j) d\mathbf{x}_i d\mathbf{x}_j \right].$$

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

$$E[\Phi] = \sum_{\mu=1}^N \int \psi_{\mu}^*(\mathbf{r}_i) \hat{h}_i \psi_{\mu}(\mathbf{r}_i) d\mathbf{r}_i + \frac{1}{2} \sum_{\mu=1}^N \sum_{\nu=1}^N \left[\int \psi_{\mu}^*(\mathbf{r}_i) \psi_{\nu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{\mu}(\mathbf{r}_i) \psi_{\nu}(\mathbf{r}_j) d\mathbf{r}_i d\mathbf{r}_j \right. \\ \left. - \int \psi_{\mu}^*(\mathbf{r}_i) \psi_{\nu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{\nu}(\mathbf{r}_i) \psi_{\mu}(\mathbf{r}_j) d\mathbf{r}_i d\mathbf{r}_j \right]$$

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.

We look at the first case first. The second one is one of the topics for project 2.

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: 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

$$\delta E - \sum_{\mu=1}^N \epsilon_{\mu} \delta \int \psi_{\mu}^* \psi_{\mu} = 0.$$

Hartree-Fock: Variational Calculus and Lagrangian Multiplier

Variation with respect to the single-particle wave functions ψ_μ yields then

$$\begin{aligned} & \sum_{\mu=1}^N \int \delta\psi_\mu^* \hat{h}_i \psi_\mu \mathbf{d}\mathbf{r}_i + \frac{1}{2} \sum_{\mu=1}^N \sum_{\nu=1}^N \left[\int \delta\psi_\mu^* \psi_\nu^* \frac{1}{r_{ij}} \psi_\mu \psi_\nu \mathbf{d}\mathbf{r}_i \mathbf{d}\mathbf{r}_j - \int \delta\psi_\mu^* \psi_\nu^* \frac{1}{r_{ij}} \psi_\nu \psi_\mu \mathbf{d}\mathbf{r}_i \mathbf{d}\mathbf{r}_j \right] \\ & + \sum_{\mu=1}^N \int \psi_\mu^* \hat{h}_i \delta\psi_\mu \mathbf{d}\mathbf{r}_i + \frac{1}{2} \sum_{\mu=1}^N \sum_{\nu=1}^N \left[\int \psi_\mu^* \psi_\nu^* \frac{1}{r_{ij}} \delta\psi_\mu \psi_\nu \mathbf{d}\mathbf{r}_i \mathbf{d}\mathbf{r}_j - \int \psi_\mu^* \psi_\nu^* \frac{1}{r_{ij}} \psi_\nu \delta\psi_\mu \mathbf{d}\mathbf{r}_i \mathbf{d}\mathbf{r}_j \right] \\ & - \sum_{\mu=1}^N E_\mu \int \delta\psi_\mu^* \psi_\mu \mathbf{d}\mathbf{r}_i - \sum_{\mu=1}^N E_\mu \int \psi_\mu^* \delta\psi_\mu \mathbf{d}\mathbf{r}_i = 0. \end{aligned}$$

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

$$V_{\mu}^d(\mathbf{r}_i) = \int \psi_{\mu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{\mu}(\mathbf{r}_j) d\mathbf{r}_j$$

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: 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

$$\psi_{\alpha}(\mathbf{r}) = \psi_{nlm_l s}(\mathbf{r}) = \phi_{nlm_l}(\mathbf{r})\xi_{m_s}(s)$$

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

$$V_{n'l'}^d(r_i) = 2 \sum_{m'_l=-l'}^{l'} \int_0^{\infty} |\phi_{n'l'm'_l}(r_j)|^2 \frac{1}{r_{ij}} d\mathbf{r}_j$$

which reads

$$V_{n'l'}^d(r_i) = 2 \int_0^{\infty} |u_{n'l'}(r_j)|^2 \frac{1}{r_{ij}} dr_j \sum_{m'_l=-l'}^{l'} |Y_{lm'_l}(\theta_j, \phi_j)|^2 d\Omega_j,$$

with $d\Omega_j$ the angular part $d(\cos\theta_j)d\phi_j$.

Hartree-Fock: Explicit expression for Direct Term

The addition theorem of the spherical harmonics yields

$$\sum_{m'=-l}^l |Y_{lm'}(\theta_j, \phi_j)|^2 = \frac{2l+1}{4\pi},$$

resulting in

$$V_{n'l'}^d(r_i) = \frac{2(2l'+1)}{4\pi} \int_0^\infty |u_{n'l'}(r_j)|^2 \frac{1}{r_{ij}} dr_j d\Omega_j.$$

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

$$F_{nl}(r_i) = \sum_{n'l'} \sum_{\lambda=|l-l'|}^{l+l'} \frac{2l'+1}{2\lambda+1} \langle ll'00|\lambda 0 \rangle^2 \left[\int_0^\infty u_{n'l'}^*(r_j) \frac{(r_<)^\lambda}{(r_>)^{\lambda+1}} u_{nl}(r_j) dr_j \right] u_{n'l'}(r_i)$$

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

$$\langle l'00|\lambda0\rangle = \frac{1}{2}\sqrt{2\lambda+1}(1+(-1)^{2g})(-1)^g\sqrt{\frac{(2g-2l)!(2g-2l')!(2g-2l')!}{(2g+1)!}} \\ \times \frac{g!}{(g-l)!(g-l')!(g-\lambda)!}$$

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. (Exercise from last week).

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

$$V_{\mu}^{\text{ex}}(\mathbf{r}_i)g(\mathbf{r}_i) = \left(\int \psi_{\mu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} g(\mathbf{r}_j) d\mathbf{r}_j \right) \psi_{\mu}(\mathbf{r}_i),$$

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 1s 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 $\alpha = nlm_l m_s = 1001/21/2$ and $\beta = nlm_l m_s = 1001/2 - 1/2$ or using $m_s = 1/2 = \uparrow$ and $m_s = -1/2 = \downarrow$ as $\alpha = nlm_l m_s = 1001/2 \uparrow$ and $\beta = nlm_l m_s = 1001/2 \downarrow$. Writing out the Slater determinant we obtain

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \alpha, \beta) = \frac{1}{\sqrt{2}} [\psi_\alpha(\mathbf{r}_1)\psi_\beta(\mathbf{r}_2) - \psi_\beta(\mathbf{r}_1)\psi_\alpha(\mathbf{r}_2)],$$

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

$$\psi_{nlm_l s m_s} = \phi_{nlm_l}(\mathbf{r}) \xi_{m_s}(s)$$

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

$$\Phi(\mathbf{r}_1, \mathbf{r}_2, \alpha, \beta) = \frac{1}{\sqrt{2}} \phi_{100}(\mathbf{r}_1) \phi_{100}(\mathbf{r}_2) [\xi_{\uparrow}(1) \xi_{\downarrow}(2) - \xi_{\uparrow}(2) \xi_{\downarrow}(1)],$$

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

$$V_{\mu}^{ex}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \left(\int \psi_{\mu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{\lambda}(\mathbf{r}_j) d\mathbf{r}_j \right) \psi_{\mu}(\mathbf{r}_i),$$

because the spins of μ and λ have to be the same due to the constraint

$$\langle s_{\mu} m_{s}^{\mu} | s_{\lambda} m_{s}^{\lambda} \rangle = \delta_{m_{s}^{\mu}, m_{s}^{\lambda}}.$$

This means that if $m_{s}^{\mu} = \uparrow$ then $m_{s}^{\lambda} = \uparrow$ and if $m_{s}^{\mu} = \downarrow$ then $m_{s}^{\lambda} = \downarrow$. That is

$$V_{\mu}^{ex}(\mathbf{r}_i)\psi_{\lambda}(\mathbf{r}_i) = \delta_{m_{s}^{\mu}, m_{s}^{\lambda}} \left(\int \psi_{\mu}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{\lambda}(\mathbf{r}_j) d\mathbf{r}_j \right) \psi_{\mu}(\mathbf{r}_i),$$

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

$$\sum_{\mu=1}^N V_{\mu}^{\text{ex}}(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) = \left(\int \psi_{100\uparrow}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \psi_{100\uparrow}(\mathbf{r}_j) d\mathbf{r}_j \right) \psi_{100\uparrow}(\mathbf{r}_i),$$

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

$$\left(-\frac{1}{2} \frac{d^2}{dr^2} + \frac{l(l+1)}{2r^2} - \frac{2}{r} + 2V_{10}^d(r) \right) u_{10}(r) - V_{10}^{\text{ex}}(r) = e_{10} u_{10}(r),$$

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 $V_{10}^{\text{ex}}(r)$ 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 1s wave function has the same Hartree-Fock contribution as in helium for the 1s state, but the 2s state gives two times the Hartree term and one time the Fock term.

That is we get

$$\begin{aligned}\sum_{\mu=1}^N V_{\mu}^d(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) &= 2 \int_0^{\infty} d\mathbf{r}_j \left(\phi_{100}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{100}(\mathbf{r}_j) + \phi_{200}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{200}(\mathbf{r}_j) \right) \psi_{100\uparrow}(\mathbf{r}_i) \\ &= (2V_{10}^d(\mathbf{r}_i) + 2V_{20}^d(\mathbf{r}_i))\psi_{100\uparrow}(\mathbf{r}_i)\end{aligned}$$

for the Hartree part.

Hartree-Fock: Explicit expressions for various Atoms, beryllium

For the Fock term we get (we fix the spin)

$$\sum_{\mu=1}^N V_{\mu}^{\text{ex}}(\mathbf{r}_i)\psi_{100\uparrow}(\mathbf{r}_i) = \int_0^{\infty} d\mathbf{r}_j \phi_{100}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{100}(\mathbf{r}_j) \psi_{100\uparrow}(\mathbf{r}_i) +$$
$$\int_0^{\infty} d\mathbf{r}_j \phi_{200}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{100}(\mathbf{r}_j) \psi_{200\uparrow}(\mathbf{r}_i) = V_{10}^{\text{ex}}(\mathbf{r}_i) + V_{20}^{\text{ex}}(\mathbf{r}_i).$$

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

$$\left(-\frac{1}{2} \frac{d^2}{dr^2} - \frac{4}{r} + V_{10}^d(r) + 2V_{20}^d(r) \right) u_{10}(r) - V_{20}^{\text{ex}}(r) = e_{10} u_{10}(r).$$

Note again that the $V_{20}^{\text{ex}}(r)$ contains the 1s function in the integral, that is

$$V_{20}^{\text{ex}}(r) = \int_0^{\infty} d\mathbf{r}_j \phi_{200}^*(\mathbf{r}_j) \frac{1}{r - r_j} \phi_{100}(\mathbf{r}_j) \psi_{200\uparrow}(\mathbf{r}).$$

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)

$$\sum_{\mu=1}^N V_{\mu}^d(\mathbf{r}_i) \psi_{200\uparrow}(\mathbf{r}_i) = 2 \int_0^{\infty} d\mathbf{r}_j \left(\phi_{100}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{100}(\mathbf{r}_j) + \phi_{200}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{200}(\mathbf{r}_j) \right) \psi_{200\uparrow}(\mathbf{r}_i) =$$
$$(2V_{10}^d(\mathbf{r}_i) + 2V_{20}^d(\mathbf{r}_i)) \psi_{200\uparrow}(\mathbf{r}_i)$$

Hartree-Fock: Explicit expressions for various Atoms, beryllium

For the Fock term we get

$$\sum_{\mu=1}^N V_{\mu}^{\text{ex}}(\mathbf{r}_i) \psi_{200\uparrow}(\mathbf{r}_i) = \int_0^{\infty} d\mathbf{r}_j \phi_{100}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{200}(\mathbf{r}_j) \psi_{100\uparrow}(\mathbf{r}_i) +$$
$$\int_0^{\infty} d\mathbf{r}_j \phi_{200}^*(\mathbf{r}_j) \frac{1}{r_{ij}} \phi_{200}(\mathbf{r}_j) \psi_{200\uparrow}(\mathbf{r}_i) = V_{10}^{\text{ex}}(\mathbf{r}_i) + V_{20}^{\text{ex}}(\mathbf{r}_i)$$

The second term is the same as we have for the Hartree term with 2s. The final differential equation is

$$\left(-\frac{1}{2} \frac{d^2}{dr^2} - \frac{4}{r} + 2V_{10}^d(r) + V_{20}^d(r) \right) u_{20}(r) - V_{10}^{\text{ex}}(r) u_{20}(r) = e_{20} u_{20}(r).$$

Note again that the $V_{10}^{\text{ex}}(r)$ contains the 2s function in the integral, that is

$$V_{10}^{\text{ex}}(r) = \int_0^{\infty} d\mathbf{r}_j \phi_{100}^*(\mathbf{r}_j) \frac{1}{r - r_j} \phi_{200}(\mathbf{r}_j) \psi_{100\uparrow}(\mathbf{r}).$$

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

$$R_{10}(r) = 2 \left(\frac{Z}{a_0} \right)^{3/2} \exp(-Zr/a_0) = u_{10}/r$$

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

$$R_{20}(r) = 2 \left(\frac{Z}{2a_0} \right)^{3/2} \left(1 - \frac{Zr}{2a_0} \right) \exp(-Zr/2a_0) = u_{20}/r$$

and the 2p hydrogen -like wave function is

$$R_{21}(r) = \frac{1}{\sqrt{3}} \left(\frac{Z}{2a_0} \right)^{3/2} \frac{Zr}{a_0} \exp(-Zr/2a_0) = u_{21}/r$$

We use $a_0 = 1$.

Effective Charge and first Iteration

If we compute the total energy of the helium atom with the function

$$R_{10}(r) = 2 \left(\frac{Z}{a_0} \right)^{3/2} \exp(-Zr/a_0) = u_{10}/r,$$

as a trial single-particle wave function, 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)

$$\psi_a = \sum_{\lambda} C_{a\lambda} \psi_{\lambda}. \quad (84)$$

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

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

and

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

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

$$\int \Phi^* \hat{H}_1 \Phi d\tau = \frac{1}{2} \sum_{\mu=1}^A \sum_{\nu=1}^A \langle \mu\nu | V | \mu\nu \rangle_{AS}. \quad (85)$$

Combining Eqs. (13) and (85) we obtain the energy functional

$$E[\Phi] = \sum_{\mu=1}^N \langle \mu | h | \mu \rangle + \frac{1}{2} \sum_{\mu=1}^N \sum_{\nu=1}^N \langle \mu\nu | V | \mu\nu \rangle_{AS}. \quad (86)$$

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. (84). We can then rewrite the energy functional as

$$E[\Psi] = \sum_{a=1}^N \langle a|h|a\rangle + \frac{1}{2} \sum_{ab=1}^N \langle ab|V|ab\rangle_{AS}, \quad (87)$$

where Ψ is the new Slater determinant defined by the new basis of Eq. (84).

Hartree-Fock by varying the coefficients of a wave function expansion

Using Eq. (84) we can rewrite Eq. (87) as

$$E[\Psi] = \sum_{a=1}^N \sum_{\alpha\beta} C_{a\alpha}^* C_{a\beta} \langle \alpha | h | \beta \rangle + \frac{1}{2} \sum_{ab=1}^N \sum_{\alpha\beta\gamma\delta} C_{a\alpha}^* C_{b\beta}^* C_{a\gamma} C_{b\delta} \langle \alpha\beta | V | \gamma\delta \rangle_{AS}. \quad (88)$$

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

$$E[\Psi] - \sum_{a=1}^N \epsilon_a \sum_{\alpha} C_{a\alpha}^* C_{a\alpha}. \quad (89)$$

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

$$\frac{d}{dC_{k\alpha}^*} \left[E[\Psi] - \sum_a \epsilon_a \sum_{\alpha} C_{a\alpha}^* C_{a\alpha} \right] = 0, \quad (90)$$

which yields for every single-particle state k the following Hartree-Fock equations

$$\sum_{\gamma} C_{k\gamma} \langle \alpha | h | \gamma \rangle + \sum_{a=1}^N \sum_{\beta\gamma\delta} C_{a\beta}^* C_{a\delta} C_{k\gamma} \langle \alpha\beta | V | \gamma\delta \rangle_{AS} = \epsilon_k C_{k\alpha}. \quad (91)$$

Hartree-Fock by varying the coefficients of a wave function expansion

We can rewrite this equation as

$$\sum_{\gamma} \left\{ \langle \alpha | h | \gamma \rangle + \sum_a^N \sum_{\beta\delta} C_{a\beta}^* C_{a\delta} \langle \alpha\beta | V | \gamma\delta \rangle_{AS} \right\} C_{k\gamma} = \epsilon_k C_{k\alpha}. \quad (92)$$

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

$$h_{\alpha\gamma}^{HF} = \langle \alpha | h | \gamma \rangle + \sum_{a=1}^N \sum_{\beta\delta} C_{a\beta}^* C_{a\delta} \langle \alpha\beta | V | \gamma\delta \rangle_{AS},$$

we can rewrite the new equations as

$$\sum_{\gamma} h_{\alpha\gamma}^{HF} C_{k\gamma} = \epsilon_k C_{k\alpha}. \quad (93)$$

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 $\langle \alpha | h | \gamma \rangle$ 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. (93) 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!! Project 2 deals with a parallelization of Jacobi's method.

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 UHF 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

$$\Psi_{nlm_l}(r, \theta, \phi) = \mathcal{N} r^{n_{\text{eff}} - 1} e^{-\frac{Z_{\text{eff}} \rho}{n_{\text{eff}}}} Y_{lm_l}(\theta, \phi). \quad (94)$$

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 $\rho = r/a_0$.

We will discuss these in more detail in connection with project 2.

Hartree-Fock theory and Density functional theory

- Wednesday:
- Derivation of the Hartree-Fock equations using a Slater basis for the single-particle wave functions. Preparation for project 2.
- Thursday:
- Derivation of the Hartree-Fock equations and methods for diagonalizing eigenvalue problems.
- Project 1: Discussion of how to implement the Slater determinant and correlation part and the conjugate gradient method.

Project 2 is presented Wednesday 25 March.

Hartree-Fock theory and Density functional theory

- Wednesday:
 - Discussion of project 2, Hartree-Fock theory, wave functions and computation of Coulomb matrix elements. Start discussion of density functional theory.
- Thursday:
 - Project 1: Continuation on how to implement the Slater determinant and correlation part and the conjugate gradient method.
 - Discussion of project 2 as well, with practicalities about the Hartree-Fock equations and density functional theory.

The material discussed for the rest of the course is also covered by Thijssen's book chapters 4-6 and 8-9.

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

Project 2 deals with the following topics:

- 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.
- The second part of the project consists of using the wave function from the Monte Carlo calculation to obtain the density of the ground state. This density is then used to define a density functional. We will make comparisons with standard approximations to density functional theory approaches like the local density approximation.

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).

Hartree-Fock: Explicit 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 We need to compute the integral for the direct 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: 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

$$\psi_{\alpha}(\mathbf{r}) = \psi_{nlm_l m_s}(\mathbf{r}) = \phi_{nlm_l}(\mathbf{r})\xi_{m_s}(s)$$

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

$$\begin{aligned} & \sum_{l=0}^{\infty} \sum_{m_l=-l}^l \frac{4\pi}{2l+1} \int r_1^2 dr_1 \int r_2^2 dr_2 R_{n_\alpha l_\alpha}^*(r_1) R_{n_\beta l_\beta}^*(r_2) \frac{(r_{<})^l}{(r_{>})^{l+1}} R_{n_\gamma l_\gamma}(r_1) R_{n_\delta l_\delta}(r_2) \\ & \times \int d\Omega_1 Y_{lm_l}^*(\theta_1, \phi_1) Y_{l_\alpha m_\alpha}^*(\theta_1, \phi_1) Y_{l_\gamma m_\gamma}(\theta_1, \phi_1) \\ & \times \int d\Omega_2 Y_{lm_l}(\theta_2, \phi_2) Y_{l_\beta m_\beta}^*(\theta_2, \phi_2) Y_{l_\delta m_\delta}(\theta_2, \phi_2) \end{aligned}$$

Hartree-Fock: Explicit expression for Exchange term

Similarly, we get for the exchange term the following expression

$$\begin{aligned} & \sum_{l=0}^{\infty} \sum_{m_l=-l}^l \frac{4\pi}{2l+1} \int r_1^2 dr_1 \int r_2^2 dr_2 R_{n_\alpha l_\alpha}^*(r_1) R_{n_\beta l_\beta}^*(r_2) \frac{(r_<)^l}{(r_>)^{l+1}} R_{n_\gamma l_\gamma}(r_2) R_{n_\delta l_\delta}(r_1) \\ & \quad \times \int d\Omega_1 Y_{lm_l}^*(\theta_1, \phi_1) Y_{l_\alpha m_\alpha}^*(\theta_1, \phi_1) Y_{l_\delta m_\delta}(\theta_1, \phi_1) \\ & \quad \times \int d\Omega_2 Y_{lm_l}(\theta_2, \phi_2) Y_{l_\beta m_\beta}^*(\theta_2, \phi_2) Y_{l_\gamma m_\gamma}(\theta_2, \phi_2) \end{aligned}$$

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

$$\int d\Omega_1 Y_{l',m'}^*(\theta_1, \phi_1) Y_{lm_l}(\theta_1, \phi_1) = \delta_{l,l'} \delta_{m_l,m'_l},$$

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

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

with energies $-Z^2/2n^2$. A function for computing the generalized Laguerre polynomials $L_{n-1}^1\left(\frac{2Z}{n}r\right)$ 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

$$\Psi_{nlm_l}(r, \theta, \phi) = \mathcal{N} r^{n_{\text{eff}}} e^{-\frac{Z_{\text{eff}} \rho}{n_{\text{eff}}}} Y_{lm_l}(\theta, \phi).$$

Here \mathcal{N} is a normalization constant, Y_{lm_l} is a spherical harmonic and $\rho = r/a_0$. Such parameterizations exist in the literature. For Beryllium we can use

$$R_{10}^{\text{STO}}(r) = N_{10} \exp(-3.6848r)$$

and

$$R_{20}^{\text{STO}}(r) = N_{20} r \exp(-1.9120r/2)$$

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.

Density functional theory (DFT)

- Wednesday:
 - Repetition of the basics of DFT
 - Proof of Hohenberg and Sham's theorem on unique density functional (Phys. Rev. **136**, B864 (1964)).
 - The local density approximation and the electron gas model
- Thursday:
 - Electron gas, local density approximation and density functional theory.
 - Discussion of project 2, with practicalities about the Hartree-Fock equations and the computation of the Coulomb term.

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 correction involving the gradient of ρ is often added to the exchange-correlation energy.