Lab Course of Scientific Computing

Quantum Many-Body Simulations in Haskell using the Hartree-Fock Method

prepared by

David Schlegel

at the Institute of Numerical and Applied Mathematics
Georg-August-Universität Göttingen

Supervisor: Dr. Jochen Schulz

Contents

1 Introduction

Applying numerical methods in quantum mechanics has always been necessary in analyzing complex structures of quantum mechanical systems. The technical progress of computer performance has enabled physicists and mathematicians to simulate complex many-body systems. With these methods tangible progress in quantum physics can be made to analyze quantum phenomena on the level of many-particle interactions.

This article tackles the implementation of the Hartree-Fock method for many-body simulations in the functional programming language `Haskell`. Functional programming languages are getting more and more interesting for physicists through their mathematical way of implementation. In this article simple quantum systems are simulated first to show how a simple one-body system can be simulated straight forward in `Haskell`.

In chapter 3, the main work in this article will be presented, namely the attempt to provide a `Haskell` function library for simulating physical many-body systems. A general overview of how the implementation can be done as well as problems will be treated in this chapter. Here it is important to note that the full implementation still needs to be completed. Thus, within the progress of this work beyond this report, other problems may arise and might be done in a different manner to increase performance, structure, or simplicity. The implementation so far will be presented as well as the problem of calculating necessary Gaussian integrals.

Although a lot of effort still needs to be put in this project, I hope to provide a structural attempt to show that the implementation of the Hartree-Fock Method in `Haskell` stands out in its simplistic way, with the strong advantage of reproducibility – due to its functional operations.

The progress of this project can be tracked on the following repositories:

- GWDG: `https://gitlab.gwdg.de/scientific_practical/project_qm` (University Access only)

- GitHub: `https://github.com/davidschlegel/hartree-fock`

A detailed html documentation about important functions for the Hartree-Fock Method can be found in the `/dist/doc/html/hartree-fock/` directory.

2 Simple Quantum Systems

In this chapter simple quantum systems will be studied. Here, we consider a particle in a three-dimensional potential $V(\vec{x})$. The corresponding wave function $\psi(\vec{x},t)$ is the solution of the Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}\psi(\vec{x},t) = H\psi(\vec{x},t) = -\frac{\hbar^2}{2m}\Delta\psi(\vec{x},t) + V(\vec{x})\psi(\vec{x},t), \tag{1}$$

where Δ is the Laplacian differential operator: $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$. For a time-independent potential $V(\vec{x})$, the Schrödinger equation can be formally solved by

$$\psi(\vec{x},t) = \mathrm{U}(t,t_0)\psi(\vec{x},t_0) = \exp\left\{-\frac{i(t-t_0)}{\hbar}H\right\}\psi(\vec{x},t_0). \tag{2}$$

For a time-dependent potential, like an oscillating laser field, the time evolution of the wave function becomes

$$\begin{aligned}\psi(\vec{x},t) = \mathrm{U}(t,t_0)\psi(\vec{x},t_0) &= \hat{\mathrm{T}}_t \exp\left\{\frac{i}{h}\int_{t_0}^{t} H(\tau)\mathrm{d}\tau\right\}\psi(\vec{x},t_0) \\ &= \sum_{n=0}^{\infty}\frac{1}{n}\left(\frac{-i}{\hbar}\right)^n \int_{t_0}^{t}\mathrm{d}t_1\int_{t_0}^{t}\mathrm{d}t_2\cdots\int_{t_0}^{t}\mathrm{d}t_n\hat{\mathrm{T}}_t\left\{H(t_1)H(t_2)\dots H(t_n)\right\},\end{aligned} \tag{3}$$

where $\hat{\mathrm{T}}_t$ is the time ordering operator. A simple approach is to divide the interval $[0\dots t]$ into a sequence of N steps so that

$$\mathrm{U}(t,t_0) = \mathrm{U}(t,t_{N-1})\dots\mathrm{U}(t_2,t_1)\mathrm{U}(t_1,t_0) \tag{4}$$

and to neglect small deviations of the Hamiltonian in the small interval $\Delta t = t_n - t_{n-1}$.

2.1 Discretization of the kinetic Energy

Dividing the Hamiltonian H into $H = T + V$, the non-local kinetic energy operator can be written as

$$T\psi(\vec{x},t) = -\frac{\hbar^2}{2m}\Delta\psi(\vec{x},t). \tag{5}$$

2.1.1 Method of Finite Differences

There are different methods and approaches to solve the above equation with numerically. Here we will focus on the approach using finite differences, due to its easy

way of implementation, with still very good results. This approach is also computationally stable for many given potential, thus this method can be used in many applications in quantum mechanics./dist/doc/html/hartree-fock/

Taking a grid (k, l, m) in three dimensions, the kinetic energy operator can be approximated by finite differences

$$T\psi(\vec{x},t) \approx -\frac{\hbar^2}{2m}\left(\frac{\psi^n_{(k+1,l,m)} - 2\psi^n_{(k,l,m)} + \psi^n_{(k-1,l,m)}}{\Delta x^2} + \right.$$
$$\left.\frac{\psi^n_{(k,l+1,m)} - 2\psi^n_{(k,l,m)} + \psi^n_{(k,l-1,m)}}{\Delta y^2} + \frac{\psi^n_{(k,l,m+1)} - 2\psi^n_{(k,l,m)} + \psi^n_{(k,l,m-1)}}{\Delta z^2}\right), \qquad (6)$$

with higher order terms $\mathcal{O}(\Delta x^2, \Delta y^2, \Delta z^2)$ where n represents the discrete time index of the wave function. Considering the time independent Schrödinger equation, we can write the operator in one dimension as a matrix satisfying the eigenvalue equation

$$\left[\begin{pmatrix} 2 & -1 & & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & \dots & & -1 & 2 \end{pmatrix} + V_{kk}\right]\begin{pmatrix} \psi_1 \\ \vdots \\ \psi_k \\ \vdots \\ \psi_N \end{pmatrix} = E\begin{pmatrix} \psi_1 \\ \vdots \\ \psi_k \\ \vdots \\ \psi_N \end{pmatrix}, \qquad (7)$$

where $\vec{\psi}$ are the values of the wave function on the evaluation points x_k. V_{kk} represents the potential for each x_k, thus it is a diagonal matrix.

For the example of a particle in a box the potential is

$$V_{kk} = \begin{cases} \infty & \text{for} \quad k = 0,\ k = N \\ 0 & \text{else} \end{cases} . \qquad (8)$$

Solving the above eigenvalue equation yields the eigenfunctions ψ_n with eigenenergies E_n. The boundary conditions $V = \infty$ at $k = 0$ and $k = N$ are satisfied even when the boundary conditions are left out. An example of the first four eigenstates is shown in figure 1. To reduce the problem of a particle in three dimensions to a simple matrix equation, the spatial wave function $\vec{\psi}$ can be stacked, to obtain a N^3

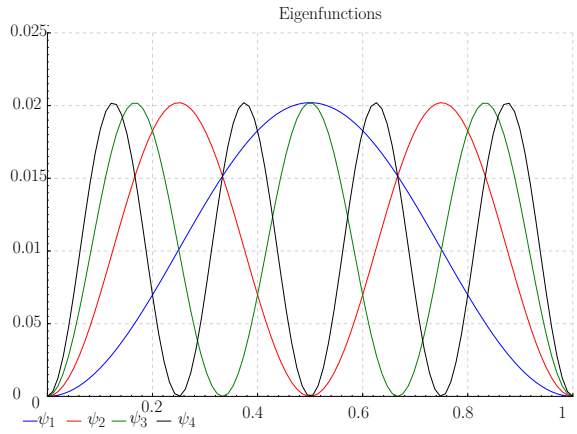


Figure 1: *Probability densities for the calculated eigenfunctions ψ_n, with the method of finite differences and a grid of $N = 100$ for $n = 1, 2, 3$ and 4 in one dimension. For simplicity the pre-factor $\frac{\hbar^2}{2m}$ was set to one, obtaining non-normalized eigenfunctions.*

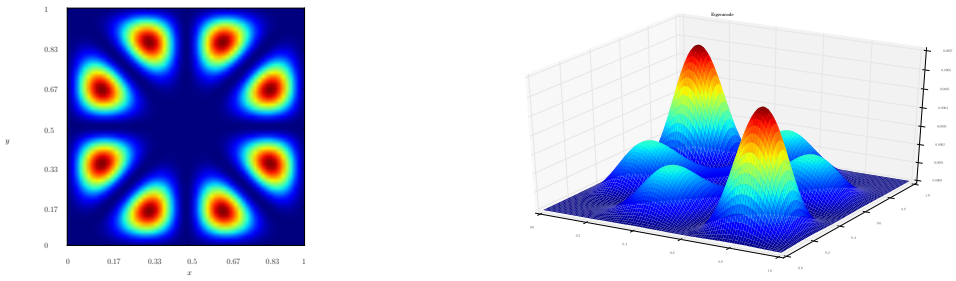
dimensional vector through nested vectors,

$$\vec{\psi}_{k,l,m} = \begin{pmatrix} \vec{\psi}_{1,1,m} \\ \vdots \\ \vec{\psi}_{1,N,m} \\ \vdots \\ \vec{\psi}_{N,1,m} \\ \vdots \\ \vec{\psi}_{N,N,m} \end{pmatrix}. \tag{9}$$

For two dimensions the approximated kinetic energy operator T thus becomes the tridiagonal block matrix

$$T \approx \begin{pmatrix} 4 & -1 & & -1 & & & 0 \\ & \ddots & & & \ddots & & \\ & -1 & 4 & & & -1 & \\ -1 & & & 4 & -1 & & \ddots \\ & \ddots & & & \ddots & & \\ 0 & & -1 & & & -1 & 4 \end{pmatrix}. \tag{10}$$

In figure 2(a) an example of an eigenstate for a particle in a two dimensional box is shown.



(a) ψ_{12}, $N = 60$.

(b) ψ_{34}, $N = 100$.

Figure 2: *Probability densities for the calculated eigenfunctions with the method of finite differences and a grid of* $N = 100$ *and* $N = 60$ *in two dimensions. For simplicity the pre-factor* $\hbar^2/2m$ *was set to one, obtaining non-normalized eigenfunctions.*

An implementation of this eigenvalue problem can be done in the following way:

```
--index function
index :: (Int, Int) -> Int -> Double
index (i,j) n
        | i == j        =  4
        | i == j + n    = -1
        | i == j - n    = -1
        | i == j + 1    = -1
        | i == j - 1    = -1
        | otherwise     =  0

-- Laplace Matrix in 2 dimensions
t :: Matrix Double
t = buildMatrix (n*n) (n*n) (\(i,j) -> index (i,j) n)

--Compute eigenvalues and corresponding eigenvectors
e = sort(toList(mapVector realPart (fst(eig(t)))))
psi = toColumns (mapMatrix realPart (snd(eig(t))))
```

In three dimensions, we obtain a N^3-dimensional Matrix, for example for $N = 2$:

$$T \approx \begin{pmatrix} 6 & -1 & -1 & & -1 & & & 0 \\ -1 & 6 & -1 & -1 & & -1 & & \\ -1 & -1 & 6 & -1 & -1 & & -1 & \\ & -1 & -1 & 6 & -1 & -1 & & -1 \\ -1 & & -1 & -1 & 6 & -1 & -1 & \\ & -1 & & -1 & -1 & 6 & -1 & -1 \\ & & -1 & & -1 & -1 & 6 & -1 \\ 0 & & & -1 & & -1 & -1 & 6 \end{pmatrix}$$

Referring back to the time expansion in chapter 2, the time evolution of an eigenfunction of the Hamiltonian is trivial, since ψ_t only differs in phase from the initial state ψ_0, so $|\psi_t|^2$ is time-independent, although linear combinations of eigenfunctions depend on time, which will not be calculated here.

If we take the computing time into account, it is obvious that even with a small grid size $N < 100$ and parallelized computing the simulation of a particle in a square well potential leads to unexecutable applications. The increase of dimensions with constant grid size results in a growing computing time higher than $\mathcal{O}(t_0^n)$.

To proceed to coupled quantum mechanical systems it seems to be mandatory to use efficient approximations.

3 The Hartree-Fock Method

We consider a real physical system, where particles are not independent, like in atoms, ions, molecules, etc. Assuming the system consists only of K nuclei and N electrons, the Hamiltonian consists of the terms

$$H = T_i + T_n + V_{ii} + V_{in} + V_{nn} \ ,$$

where T_i and T_n are the kinetic energies of the electrons and nuclei, respectively. V_{ii} represents the Coulomb repulsion between electrons, V_{nn} between the nuclei and V_{in} the attraction between electrons and nuclei. So the Hamiltonian reads

$$\begin{aligned} H = &\sum_{i=1}^{N} \frac{p_i^2}{2m} + \sum_{n=1}^{K} \frac{P_n^2}{2M_n} + \frac{1}{4\pi\epsilon_0}\frac{1}{2} \sum_{i,j\neq 1, i\neq j}^{N} \frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} \\ &- \frac{1}{4\pi\epsilon_0} \sum_{i=1}^{K}\sum_{i=1}^{N} \frac{Z_n e^2}{|\mathbf{r}_j - \mathbf{R}_n|} + \frac{1}{4\pi\epsilon_0}\frac{1}{2} \sum_{n,n'=1; n\neq n'}^{K} \frac{Z_n Z_{n'} e^2}{|\mathbf{R}_n - \mathbf{R}_{n'}|}, \end{aligned}$$

where m is the electron mass and M_n the nuclei mass. The index i refers to the electrons, the index n to the nuclei. This Hamiltonian describing the system looks quite complicated and in fact, computing the dynamics of this system seem to be unsolvable even for a few particles and an efficient super computer. Therefore, approximations must be made, still comprising the important information about the system. One approach is the Born-Oppenheimer approximation. It uses the fact that the nuclei are much heavier than the electrons, so the motions of the nuclei are much slower compared to the electrons, justifying to neglect the coulomb repulsion between the nuclei and the kinetic energy of the nuclei, so the approximated Hamiltonian becomes

$$H_{\mathrm{BO}} = \sum_{i=1}^{N} \frac{p_i^2}{2m} + \frac{1}{4\pi\epsilon_0}\frac{1}{2} \sum_{i,j=1, i\neq j}^{N} \frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} - \frac{1}{4\pi\epsilon_0} \sum_{i=1}^{K}\sum_{i=1}^{N} \frac{Z_n e^2}{|\mathbf{r}_j - \mathbf{R}_n|}. \tag{11}$$

The positions of the nuclei can be varied to find the minimum of the total energy.

3.1 The Hartree-Fock Method and Equations

In equation (11) the antisymmetry of the fermion wave functions was not taken into account. Fock extended the so called *Hartree equation* by taking antisymmetry into account. This approach is based on introducing antisymmetry of the wave

function by constructing the so called Slater determinant, which is a determinant from single-particle wave functions:

$$\Psi_{\mathrm{AS}}(\mathbf{x}_1,\ldots,\mathbf{x}_N) = \frac{1}{\sqrt{N!}}\begin{vmatrix} \psi_1(\mathbf{x}_1) & \psi_2(\mathbf{x}_1) & \ldots & \psi_N(\mathbf{x}_1) \\ \psi_1(\mathbf{x}_2) & \psi_2(\mathbf{x}_2) & \ldots & \psi_N(\mathbf{x}_2) \\ \vdots & \vdots & & \vdots \\ \psi_1(\mathbf{x}_N) & \psi_2(\mathbf{x}_N) & \ldots & \psi_N(\mathbf{x}_N) \end{vmatrix}$$

Interchanging spatial coordinates of two particles thus changes the sign of the slater determinant. It is important to note that the slater determinant already imposes a restriction to the full wave function. Further improvements could be made to also take linear combinations of slater determinants into account.

The derivation of the *Hartree* and *Hartree-Fock equations* will not be given here, since the main aim here is the implementation of the method. Therefore see e.g. [1, p. 56-60] and [2, chapter 3]. The *Fock operator* is in natural units given by

$$\begin{aligned}\mathcal{F}\psi_k = &\left[-\frac{1}{2}\nabla^2 - \sum_n \frac{Z_n}{|\mathbf{r}-\mathbf{R}_n|}\right]\psi_k(\mathbf{x}) + \sum_{l=1}^{N}\int \mathrm{d}x' |\psi_l(\mathbf{x}')|^2 \frac{1}{|\mathbf{r}-\mathbf{r}'|}\psi_k(\mathbf{x}) \\ &- \sum_{l=1}^{N}\int \mathrm{d}x' \psi^*(\mathbf{x}')\frac{1}{|\mathbf{r}-\mathbf{r}'|}\psi_k(\mathbf{x}')\psi_l(\mathbf{x}),\end{aligned} \tag{12}$$

satisfying the *Hartree-Fock equation*

$$\mathcal{F}\psi_k = \epsilon_k \psi_k.$$

The fourth term in equation (12) is called the exchange term [1, p. 55] and is nonlocal since the operator is acting on ψ_k, but the value at the position $\mathbf{r}$ is determined by the value assumed by ψ_k at all possible positions $\mathbf{r}'$. The eigenvalues ϵ_k of the Fock operator are related to the total energy by

$$E = \frac{1}{2}\sum_k [\epsilon_k + \langle\psi_k|h|\psi_k\rangle], \text{ with}$$

$$\langle\psi_k|h|\psi_k\rangle = \int \mathrm{d}^3\mathbf{x}\ \psi_k^*(\mathbf{x})\left[-\frac{1}{2}\nabla^2 - \sum_n \frac{Z_n}{|\mathbf{r}-\mathbf{R}_n|}\right]\psi_k(\mathbf{x}).$$

It is obvious that the Fock operator in equation (12) which acts on ψ_k also depends on ψ_k itself.

3.2 The Roothaan Equation

For molecules numerical basis sets are not efficient, so we use atom centered basis sets to describe the molecular orbital

$$\psi_k(\vec{x}) = \sum_{p=1}^{M} C_{pk}\chi_p(\vec{x}), \tag{13}$$

with $k = 1, \ldots, M$, where M is the number of basis states. So the Hartree-Fock equation reads

$$\mathcal{F}(\vec{x})\psi_k(\vec{x}) = \epsilon_k\psi_k(\vec{x}). \quad \text{Using Eq. (13) yields}$$
$$\mathcal{F}(\vec{x})\sum_{p=1}^{M} C_{pk}\chi_p(\vec{x}) = \epsilon_k \sum_{p=1}^{M} C_{pk}\chi_p(\vec{x}).$$

We can now multiply from the left by a specific basis function χ_ν. Integrating over the space yields

$$\sum_{p=1}^{M} C_{pk} \int \chi_\nu^*(\vec{x})\mathcal{F}(\vec{x})\chi_p(\vec{x})\mathrm{d}^3\vec{x} = \epsilon_k \sum_{p=1}^{M} C_{pk} \int \chi_\nu^*(\vec{x})\chi_p(\vec{x})\mathrm{d}^3\vec{x}.$$

This can be written in matrix notation of the Hartree-Fock equations, expressed in the atomic orbital basis:

$$\sum_{p=1}^{M} C_{pk}F_{\nu p} = \epsilon_k \sum_{p=1}^{M} C_{pk}S_{\nu p}$$
$$\mathbf{FC_k} = \epsilon_k \mathbf{SC_k}\ , \tag{14}$$

which is called the *Roothaan equation*. Since the Fock operator itself depends on $\mathbf{C_k}$ we have the problem of self consistency, because $\mathcal{F}$ has the solution already inside. The Self Consistent Field Method (SCF) uses the approach to solve the above equation iteratively, until the solution deviates only small from the next iteration.

Equation (14) looks similar to the Hartree-Fock equation, but here $\mathbf{S}$ is the overlap matrix for the given orbital basis $\chi_p(\mathbf{r})$. The $\mathbf{S}$-matrix does not vanish, since the used orbital basis does not need to be normalized. In comparison with eq.

(12) we can write the Fock matrix as [1]

$$F_{pq} = h_{pq} + \sum_k \sum_{rs} C^*_{rk} C_{sk} \left(2 \langle pr|g|qs\rangle - \langle pr|g|sq\rangle\right),$$

where h_{pq} is identified with the first term of the Fock operator

$$h_{pq} = \langle p|h|q\rangle = \int \mathrm{d}^3 r \chi^*_p(\mathbf{r}) \left[-\frac{1}{2}\nabla^2 - \sum_n \frac{Z_n}{|\mathbf{R}_n - \mathbf{r}|}\right] \chi_q(\mathbf{r}).$$

The terms $\langle pr|g|qs\rangle$ and $\langle pr|g|sq\rangle$ are the two-electron integrals given by

$$\langle pr|g|qs\rangle = \int \mathrm{d}^3 r_1 \mathrm{d}^3 r_2 \chi^*_p(\mathbf{r}_1) \chi^*_r(\mathbf{r}_2) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \chi_q(\mathbf{r}_1) \chi_s(\mathbf{r}_2).$$

Here p, q, r and s label the basis functions and k labels the orbitals ψ_k. For simplicity and implementation it is convenient to define the density matrix as

$$P_{pq} = 2 \sum_k C^*_{pk} C_{qk},$$

where the factor 2 is conventionally due to the spin [1]. Now the Fock matrix reads

$$F_{pq} = h_{pq} + \sum_{rs} P_{sr} \left(2 \langle pr|g|qs\rangle - \langle pr|g|sq\rangle\right).$$

In this orbital representation the energy is given by

$$E = \sum_{pq} P_{pq} h_{pq} + \frac{1}{2} \sum_{pqrs} P_{pq} P_{sr} \left[\langle pr|g|qs\rangle - \frac{1}{2} \langle pr|g|sq\rangle\right].$$

3.3 Closed and open shell systems

Since also the spin plays important role in atomic structures, further restriction on the spin states can be made, leading to closed and open-shell systems, referred to as Unrestricted Hartree-Fock, and Restricted Hartree-Fock. In a closed shell, the orbitals are occupied by two electrons with opposite spin, whereas in an open shell, the energy levels are partially filled where levels contain only one electron. Since the occupied levels in atoms strictly follow the Aufbau principle, the restriction of a closed-shell might lead to unphysical results in some cases. Even if the number of electrons is even, the electrons do not necessarily occupy closed shells – for an odd number of electrons, an open shell is always present. For a closed shell, and thus for

even N, the slater determinant is

$$\Psi_{\mathbf{R}} = \det\left[\alpha\psi_1, \ldots, \alpha\psi_{N/2}, \beta\psi_1, \ldots, \beta\psi_{N/2}\right](\mathbf{x}_1, \ldots, \mathbf{x}_N),$$

which is referred to as **restricted closed shell Hartree-Fock** (RHF). When allowing the number of spin-up and spin-down states to be different, the slater-determinant becomes:

$$\Psi_{\mathbf{R}} = \det\left[\alpha\psi_1, \ldots, \alpha\psi_{N_\alpha}, \beta\psi_1, \ldots, \beta\psi_{N_\beta}\right](\mathbf{x}_1, \ldots, \mathbf{x}_N).$$

This represents the **restricted open shell Hartree-Fock** (ROHF). However, in the general case, also the one-particle wave functions have spin-dependence, requiring two distinct basis sets, for spin-down and for spin-up states, respectively, yielding to the **unrestricted Hartree-Fock** (UHF) method, with slater determinant

$$\Psi_{\mathbf{R}} = \det\left[\alpha\psi_1^\alpha, \ldots, \alpha\psi_{N_\alpha}^\alpha, \beta\psi_1^\beta, \ldots, \beta\psi_{N_\beta}^\beta\right](\mathbf{x}_1, \ldots, \mathbf{x}_N).$$

In this general case, the spin-dependence of the basis set is reflected in the expansion coefficients $\mathbf{C}_k$, since they also depend on the spin, yielding the so-called *Pople-Nesbet* equations:

$$\mathbf{F}^\alpha {\mathbf{C}_\mathbf{k}}^\alpha = \epsilon_k^\alpha \mathbf{S}{\mathbf{C}_\mathbf{k}}^\alpha$$
$$\mathbf{F}^\beta {\mathbf{C}_\mathbf{k}}^\beta = \epsilon_k^\beta \mathbf{S}{\mathbf{C}_\mathbf{k}}^\beta$$

Here, each equation for spin-up and spin-down orbitals has to be solved independently. However, we will restrict ourselves to RHF for simplicity.

3.4 Basis set and basis functions

A key for obtaining reasonable and accurate results in Hartree-Fock theory is the correct choice of the orbital set on which the calculation is based. As mentioned in section. 3.2, the one-particle wave functions are expanded in the form:

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

which is a linear combination of molecular orbitals (LCAO). Since analytic forms of the desired atomic orbitals are only known for the hydrogen atom, these orbitals need to be approximated in a suitable manner. Following the analytic solution for

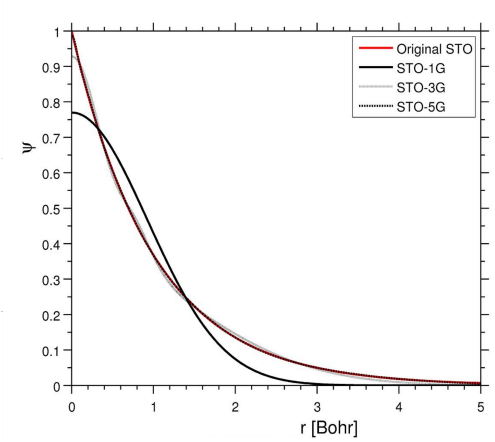


Figure 3: *Approximation of a slater determinant for the 1s orbital by STO-1G, STO-3G and STO-6G basis functions. [3]*

the hydrogen orbital, we get a general form of atomic orbital basis functions, with the nucleus centered at $\mathbf{R}_\mathrm{A}$:

$$\chi_\alpha(\mathbf{r}) = r^m P_l(x,y,z)\mathrm{e}^{-\alpha|\mathbf{r}-\mathbf{R}_\mathrm{A}|},$$

where α defines the range of the orbital. Here, P_l is a polynomial in x, y and z of degree l. This orbital type is also known as a Slater type orbital (STO). However, in Hartree-Fock calculations these orbital types are highly disadvantageous, since integrals containing products of STOs are analytically very difficult and in most cases impossible to compute.

Instead, we can avoid this problem, by replacing the simple exponential by a Gaussian function, yielding:

$$\chi_\alpha(\mathbf{r}) = P_L(x,y,z)\mathrm{e}^{-\alpha(\mathbf{r}-\mathbf{R}_\mathrm{A})^2} = x^l y^m z^n \mathrm{e}^{-\alpha(\mathbf{r}-\mathbf{R}_\mathrm{A})^2}$$

These functions are called *primitive basis functions*. In this case, products of Gaussian type orbitals (GTO) centered at different nuclei are still Gaussian functions. Thus, the necessary integrals, such as the kinetic integrals, the overlap integrals, the nuclear repulsion integrals as well as the electron-electron interaction integrals can be calculated analytically. To accurately approximate a Slater-type orbital one can use a linear combination of Gaussian-type orbitals, which is called a *contracted* Gaussian orbital, with contraction coefficients a_i. Denoting a primitive Gaussian

with $G(\alpha, A, l, m, n)$, a contraction is simply given by a set of primitive Gaussians:

$$G^{\text{Contr.}} = \sum_{i}^{n} a_i G(\alpha_i, A, l, m, n)$$

Here, the variables l, m and n are related to the total angular momentum quantum number L by $L = l + m + n$. In Fig. 3, a comparison of different contractions for an 1s orbital is shown. To represent the cusp of the electron probability at $r = 0$ for an STO, one needs to take more than one primitive Gaussian into account. Choosing and finding the right basis set is a rather difficult part and there is a huge data base of different basis sets available (see, e.g. [4, 5] and the EMSL Basis Set Exchange database). Here, we use the basis set provided by the EMSL Basis Set Exchange database for our calculations.

For a given basis set, the following integrals need to be computed in the Hartree-Fock method:

- **Overlap integral**

$$S_{ij} = \langle \alpha_1, \mathbf{A}, l_1, m_1, n_1 | \alpha_2, \mathbf{B}, l_2, m_2, n_2 \rangle = \int \psi_1^*(\mathbf{r}) \psi_2(\mathbf{r}) d\mathbf{r}$$

- **Kinetic energy integral**

$$T_{ij} = \langle \alpha_1, \mathbf{A}, l_1, m_1, n_1 | -\frac{1}{2}\nabla^2 | \alpha_2, \mathbf{B}, l_2, m_2, n_2 \rangle = \int \psi_1^*(\mathbf{r}) \left(-\frac{1}{2}\nabla^2 \right) \psi_2(\mathbf{r}) d\mathbf{r}$$

- **Nuclear Attraction integrals**

$$V_{ij}^C = \langle \alpha_1, \mathbf{A}, l_1, m_1, n_1 | \frac{Z_C}{r_C} | \alpha_2, \mathbf{B}, l_2, m_2, n_2 \rangle = \int \psi_1^*(\mathbf{r}) \left(\frac{Z_C}{r_C}\nabla^2 \right) \psi_2(\mathbf{r}) d\mathbf{r}$$

- **Electron repulsion integral**

$$\text{ERI} = \langle pr|g|qs \rangle = \int \mathrm{d}^3 r_1 \mathrm{d}^3 r_2 \psi_p^*(\mathbf{r}_1) \psi_r^*(\mathbf{r}_2) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \psi_q(\mathbf{r}_1) \psi_s(\mathbf{r}_2)$$

The last integral is by far the most important and also most difficult part since the calculation involving 4 different primitive Gaussians scales with N^4, where N is the number of primitive Gaussians. Using very efficient state-of-the-art algorithms, the calculation of the electron repulsion integral can be reduced to $\mathcal{O}(N)$. For these integrals, analytic solutions exists. The implementation of these integrals is straightforward but very tedious. For a detailed calculation of Gaussian integral evaluation,

see [6–10]. The analytical computations of these functions are very advantageous, since they are represented by pure functions, i.e. no side-effects or the underlying random system or memory affects the functions, making the computations completely reproducible.

3.5 Program Structure

Here a general overview of the general program structure will be given.

1. **Input data**:
   Here the basis sets as well as the geometry of the atoms in the molecule will be provided and parsed to the program. Furthermore the numbers of electrons N and the atomic charge numbers Z_n are needed as input.

2. **Calculate independent matrices**
   All matrices that are independent of the eigenvectors $\mathbf{C}_k$ can be calculated:
   - The overlap matrix S_pq,
   - The two-electron integrals $\langle pr|g|qs\rangle$,
   - The uncoupled one-electron Hamiltonian h_{pq}.

3. **Make an initial guess for the density matrix P.**
   A possibility is to use $\mathbf{P} = 0$ as an initial guess, which means that the electrons only feel the nuclei but not each other.

4. **Self-consistency procedure**
   This program section is the most important, since in solves the Roothaan equation recursively. It consist of the following steps.
   - Calculate the Coulomb and exchange contributions to the Fock matrix,
   - Construct the Fock matrix,
   - Solve the Roothaan equation by diagonalization of the Fock matrix for the given density matrix $\mathbf{P}$,
   - With the obtained eigenvectors a new density matrix is $\mathbf{P}$ is constructed.

5. **Output data**
   The output is are the converged eigenstates $\mathbf{C}_k$ and its corresponding Fock levels.

In figure 4 a schematic representation of the general program scheme is shown.

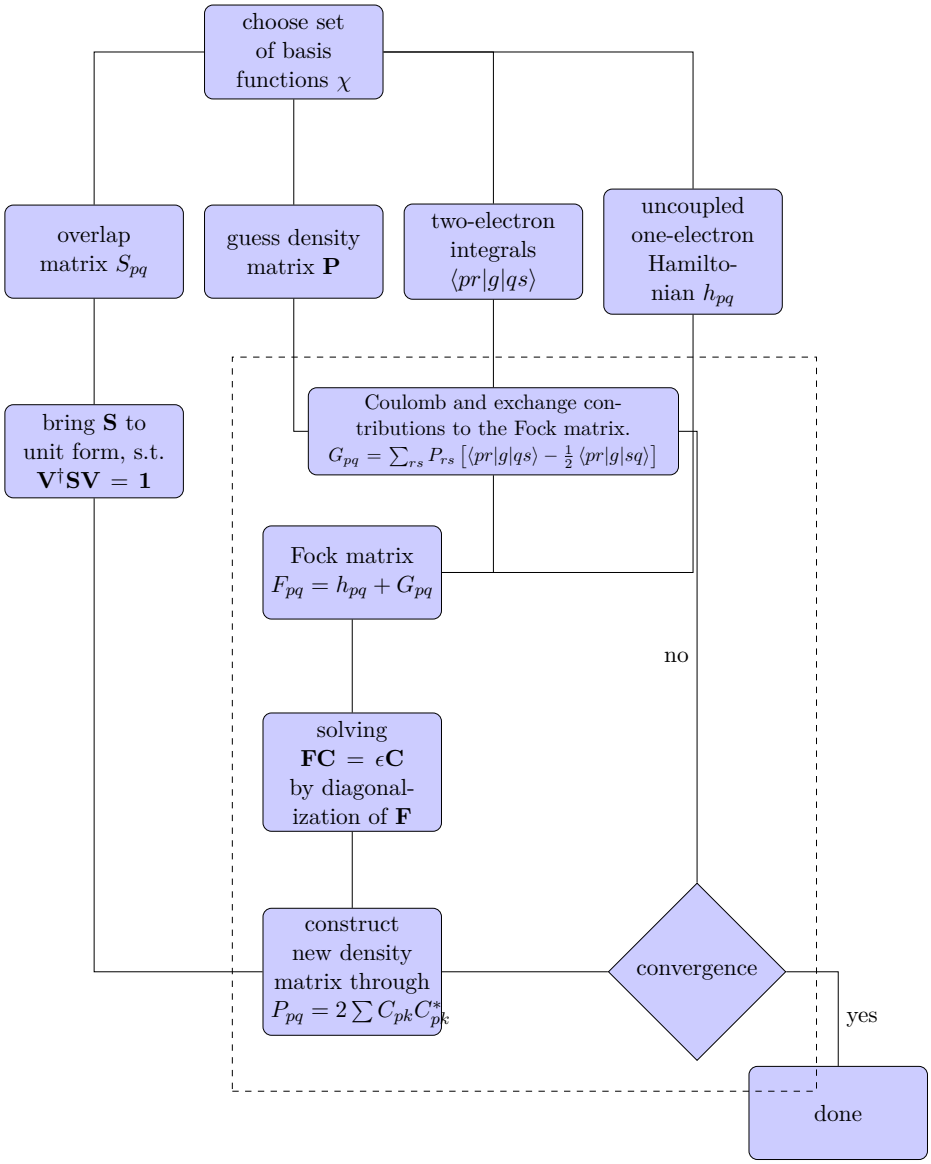


Figure 4: *General scheme of the program structure.*

To solve the Roothaan equation, which is a generalized eigenvalue equation, one can reduce this equation to a normal eigenvalue equation, by finding a matrix $\mathbf{V}$, such that $\mathbf{V}^\dagger\mathbf{S}\mathbf{V} = \mathbf{1}$. In the program, we use build-in functions to solve the entire generalized eigenvalue equation.

3.6 Implementation

Here, an overview about the implementation so far will be given. Not all of the implemented functions are mentioned. A detailed program documentation can be found in the `/dist/doc/html/hartree-fock/` directory. A schematic scheme of the different program modules is shown in Fig. 5.

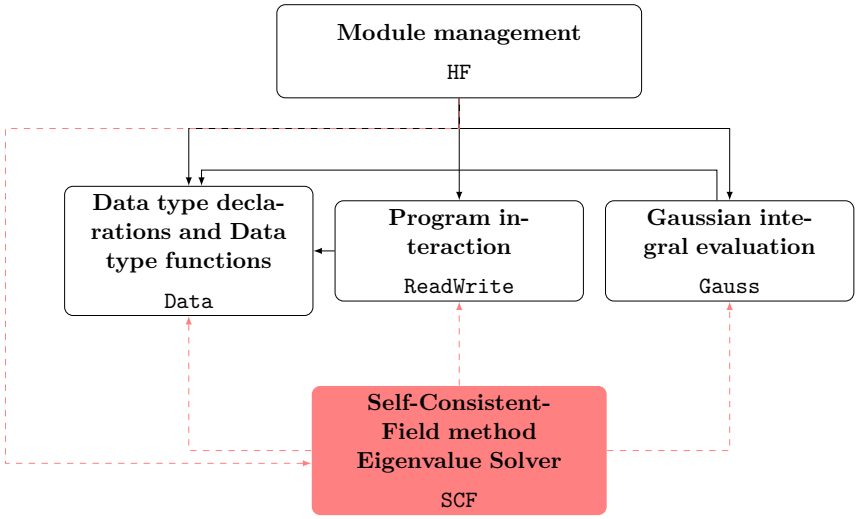


Figure 5: *Schematic modular program structure. Arrows represent module dependencies. The `SCF` module still needs to be implemented.*

3.6.1 Data type declarations and Data type functions

Specific Data types are necessary to maintain an overview about the entire program structure and to simplify calculations. Therefore we define the following data types:

- `Orbital`, defining a certain orbital; dependencies: None
- `Atom`, defining a collection of several (contracted) Orbitals; dependencies: Orbital
- `Mol`, defining a Structure containing multiple Atoms; dependencies: `Atom`, `Orbital`
- `PG`, defining a primitive Gaussian function; dependencies: None
- `Ctr`, defining a set of contractions of primitive Gaussians; dependencies: `PG`

For all data types we use a *record syntax*, to access the information.

`Orbital`

The `Orbital` defines all information about a certain (contracted) Gaussian Orbital. Data type contains the same information as `Ctr`, but it is a canonical data structure for importing data using the basis set database.

```
data Orbital = Orbital {
                        -- | orbital-type (angular momentum) and contraction
                            length  (see also 'momentumdict')
                        description :: (String, Int)
                        -- | numbering of gaussians
                        , numbering :: Vector Double
                        -- | exponents of gaussians
                        , exponents :: Vector Double
                        -- | coefficients of gaussians
                        , coeffs :: Vector Double
                        } deriving (Eq, Show)
```

`description` contains the angular momentum information (e.g. `"S"`, `"P"`, `"D"`) and the length of contraction, i.e. the number of contraction coefficients. The exponents and the contraction coefficients are stored as vectors.

`Atom`

To store all orbital information for a certain atom, we use the data type `Atom`:

```
data Atom = Atom {atomname :: String     -- ^ name of the Atom e.g. \"CARBON\"
                 , orbitals :: [Orbital] -- ^ list of the orbitals
                     corresponding to an atom
                  } deriving (Eq, Show)
```

Here, `atomname` is a `String` used to identify the atom (e.g. `"CARBON"`) and its corresponding atomic number Z.

`Mol`

To construct an entire molecule which is the input data type for the Hartree-Fock calculations, we provide a data type containing multiple `Atom` data types, with corresponding position vectors:

```
data Mol = Mol {  molname :: String              -- ^ Mol name , e.g. \"H20\"
    etc.
```

```
                , config :: [(Atom , Vector Double)] -- ^ configuration
                } deriving (Eq, Show)
```

To easily compute the entire geometry of the system, the number of basis functions and the number of atoms, we provide the functions

```
geometry :: Mol -> [Vector Double]

natoms :: Mol -> Int

nbasisfunctions :: Mol -> Int
```

Additionally, we can merge a list of `Mol` data types into a single `Mol` data type, which makes it easy to construct larger molecular structures, with repeating properties, s.a. benzenes, phenyls, etc.:

```
mergeMols :: [Mol] -- ^ List of Mol data types
                   -> String -- ^ New structure name
                   -> Mol -- ^ final superstructure Mol
```

`PG` and `Ctr`

For matrix calculations including primitive Gaussian functions, it is convenient to use a data type with all necessary information:

```
data PG = PG { lmn :: [Int]     -- ^ \[l,m,n\] angular momentum information
             , alpha :: Double -- ^ exponent
             , position :: Vector Double -- ^ position vector
             } deriving (Eq, Show)
```

For convenience, we define a contraction, `Ctr`, as a list of primitives, with corresponding contraction coefficients:

```
data Ctr = Ctr{ gaussians :: [PG]       -- ^ list of primitives
              , coefflist :: Vector Double -- ^ coefficient vector
              } deriving (Eq, Show)
```

We can convert a `Mol` data type to a list of `Ctr` data types, which is necessary for matrix computations involving Gaussian functions:

```
mol_to_gaussians :: Mol -> [Ctr]
```

3.6.2 Input data

For the input of basis set data, we integrate an open source external `Python` interface, Emsl basis set exchange local provided by Thomas Applencourt (for more information, see `https://github.com/TApplencourt/EMSL_Basis_Set_`

`Exchange_Local.git`). To generate basis set data for a given atom and orbital type, (e.g. `"STO-6G"`), we save the basis set data to a file, for each atom respectively by the function

```
generateFile:: [Char] -- ^ basis, e.g."STO-6G"
               -> [Char] -- ^ elementsymbol, e.g. "C"
               -> [Char] -- ^ output filename
               -> IO (Maybe Handle, Maybe Handle, Maybe Handle, ProcessHandle
                   ) -- ^ output file is saved to @
                   EMSL_Basis_Set_Exchange_Local/data/ @
```

From a given file with basis set information, we can then parse the basis set information from the file into the `Haskell` program. Thus, one file with generated basis set data gives a single `Atom` data type. This is done using the function

```
getAtomData :: [Char] -- ^ filename of the file where the basis set for a
    certain element is stored
               -> Atom -- ^ Atom data structure
```

Additionally, we can read geometry information for given atoms stored in a file. This is especially useful when comparing results with existing Hartree-Fock programs. Here, we use the xyz-formatting, which is also used by the Molden Software. Since constructing a complex molecular structure with a good guess of initial geometry can be rather difficult from scratch, the molecule with its geometry can be constructed via Molden and then used as input in our program. This can be done in the following way.

1. Read a file containing geometry information and output a list of element symbols, e.g. `"C"` with corresponding positions. NOTE: The file should have the Cartesian XYZ-file format that is used in Molden.

   ```
   getgeom :: [Char] -- ^ directory
           -> [([Char], Vector Double)] -- ^ List with element symbols and
               corresponding position vectors
   ```

   example: **`> > > let list = getgeom "CO2.xyz"`**

2. With `constr_set_from_file`, construct basis set data for the different elements which will be saved to files

   ```
   constr_set_from_file :: [([Char], Vector Double)] -> [Char] -> IO ()
   --For a list of element symbols with corresponding positions ([([Char],
       Vector Double)]) and a given basis set, e.g. "STO-3G", save basis
       set data to multiple files.
   ```

   example:**`> > > constr_set_from_file list "STO-6G"`**

3. With `get_mol_from_files`, read the basis set data to construct the complete molecule:

```
get_mol_from_files :: [([Char], Vector Double)] -- ^ list of element
    symbols with corresponding positions
                -> [Char] -- ^ basis set name, e.g. \"STO-3G\"
                -> [Char] -- ^ molname, e.g. \"CO2\"
                -> Mol
```

example:**> > > let mol = get_mol_from_files list "STO-6G" "CO2"**

3.6.3 Output

Information about the constructed `Mol` data type can obtained by the function

```
molInfoPrint :: Mol -> String
{-__Output:__
        Molecule Name
        Number of Atoms
        For each Atom:
                * Atomname
                * Number of Orbitals
                * contracted set descriptions
                * geometry
-}
```

example:

```
> > > let mol = Mol "SINGLE-Carbon" [(getAtomData "C_STO_3G.dat",
fromList [1,0,0])]
> > > putStrLn $ molInfoprint mol
Molecule name:  SINGLE-Carbon
Number of Atoms:  1
Atom:  CARBON
Number of Orbitals:  3
contracted sets:  [("S",3),("S",3),("P",3)]
geometry:  [1.0,0.0,0.0]
```

Additionally, only geometry information can be obtained by

```
molGeomPrint :: Mol -> String
```

3.6.4 Gaussian integral evaluation

The Gaussian integral evaluation is so far the most difficult and tedious part. The analytical expressions for the integrals in Sec. 3.4 are difficult to compute

and involves double factorial and error function evaluation. If the angular quantum number L is larger than zero, which is the case for most of the orbitals in atoms, the calculation become somewhat difficult. A general summation of functions used for Gaussian integral evaluation:

- **Normalization**
  For primitive Gaussians as well as for an entire Contraction, the normalization constant needs to be computed:

```
normCtr :: Ctr -> Double
--Calculates normalization factor of contracted Gaussian with arbitrary
    angular momentum
normPG :: PG -> Double
--Calculates normalization factor of primitive Gaussian with arbitrary
    angular momentum
```

- **Contraction operations**

```
-- |Evaluates a given function for two contractions. The operation
    is symmetric in contraction arguments.
zipContractionWith :: (PG -> PG -> Double) -- ^ Function of two
    primitive Gaussians
                      -> Ctr -- ^ Contraction
                      -> Ctr -- ^ Contraction
                      -> Double

-- |Construct a matrix out of a list of Contractions and a function
    for two primitive gaussians
constr_matrix :: [Ctr] -- ^ List of Contraction
              -> (PG -> PG -> Double) -- ^ Function of two
                  primitive Gaussians
              -> Matrix Double -- ^ Matrix of dimensions (length [
                  Ctr]) x (length [Ctr])
```

- **Gaussian integrals**
  - `s_12`: Overlap integral

$$S_{ij} = \langle \alpha_1, \mathbf{A}, l_1, m_1, n_1 | \alpha_2, \mathbf{B}, l_2, m_2, n_2 \rangle$$

```
-- |Calculates overlap integral of two primitive gaussians
s_12 :: PG -> PG -> Double
```

– **Kinetic energy integral**

$$T_{ij} = \langle \alpha_1, \mathbf{A}, l_1, m_1, n_1 | -\frac{1}{2}\nabla^2 | \alpha_2, \mathbf{B}, l_2, m_2, n_2 \rangle$$

The calculation of the kinetic energy integral is based on the calculation of the overlap integral.

```
-- |Calculates kinetic energy integral of two primitive gaussians
t_12 :: PG -> PG -> Double
```

– **Nuclear attraction integral**

$$V_{ij}^{C} = \langle \alpha_1, \mathbf{A}, l_1, m_1, n_1 | \frac{Z_C}{r_C} | \alpha_2, \mathbf{B}, l_2, m_2, n_2 \rangle$$

Note: The calculation of the nuclear attraction integral is at the present state not complete. For the calculation of the nuclear attraction integral, also the atomic number Z_C as well as the distance to the nucleus r_C are needed.

```
-- |Calculates nuclear attraction integral of two primitive
    gaussians
-- | Here, the second argument is the position vector of the nucleus
    .
v_12 :: Double -> Vector Double -> PG -> PG -> Double
```

– **Electron repulsion integrals**

$$\langle pr|g|qs\rangle = \int \mathrm{d}^3 r_1 \mathrm{d}^3 r_2 \psi_p^*(\mathbf{r}_1)\psi_r^*(\mathbf{r}_2)\frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}\psi_q(\mathbf{r}_1)\psi_s(\mathbf{r}_2)$$

Note: The calculation of the electron repulsion integrals is no yet complete. Different algorithms and recurrence relations can be used to make the calculation simpler and faster. However, the implementation of the electron repulsion integral is one of the most tedious and difficult parts in this program.

3.7 Problems

Looking at the Hartree-Fock method as a whole, it is fair to say that this project is an endeavor which is far to complex to implement alone. During the process of program implementation, several problems arose:

- The first problem of getting suitable basis set data could be nicely solved by making use of the open source `Python` program "Emsl basis set exchange local". This made it possible to directly parse the basis set information into `Haskell` data types.

- The most difficult part after all is the implementation of the Gaussian integrals. The analytic expressions for these evaluations are difficult to find in literature and far more difficult than expected. The implementation of the Gaussian integral calculations consumed most of the effort in comparison to the other implemented modules. At the present state, the implementation of the nuclear attraction integral and the electron repulsion integrals still needs to be done. A possible solution to this difficulty is to rely on existing implementations in imperative programming languages. Doing so, the functional purity would be lost but more effort could be put in implementing the self-consistent field scheme. Due to the fact that `Haskell` is still a relatively young programming language, the amount of existing `Haskell` libraries, which could possibly used in this case, is rather small in comparison to e.g. `Python` or `C/C++`. Another compromise solution could be a plain numerical calculation of the given Gaussian integrals instead of implementing analytical solutions.

- Another problem lies in the difficulty of debugging implemented functions, since currently the program is not able to produce scientifically comparable results. The disparity of imperative and functional programming languages makes it hard to compare calculation steps in existing programs with the program presented here, although still possible, to find implementation errors.

4 Outlook

We have seen that the functional programming language `Haskell` is a good choice for studying quantum mechanical systems. First we have looked at single-particle systems and calculated, as an example, the two-dimensional system, with an electron confined in an infinitely deep square potential using the method of finite differences. Here, an implementation in `Haskell` stands up in its pure functional properties and concise coding possibilities.

The main aim of this work was to investigate quantum many-body systems and how this can be done by implementing numerical methods in `Haskell`. As one of the most fundamental theories in many-electron systems, the Hartree-Fock method was chosen to simulate atomic structures, such as molecules and others. We first outlined the key concepts and equations in Hartree-Fock theory with special focus on computational applications. On this basis, a general scheme, independent of functional or imperative programming languages, was outlined. We further developed a general program structure, with keeping the advantages and properties of `Haskell` in mind. Therefore, all functions involving `IO`-functions are concentrated in a single module, all data type declarations in another. The modular approach allows it to keep track of the entire program development without loosing a good overview. As outlined in Sec. 3.4 especially basis functions and basis sets play an important role in the implementation, since their computation is a difficult part. We provided a user friendly interface, to construct atomic structures with all their necessary information. An external `Python` program was used to provide all basis set data which can be parsed to the `Haskell` program and converted into suitable data structures. As outlined in the previous section, unforeseen difficulties arose when implementing the Gaussian integrals analytically.

Although the program presented here is not yet ready to execute electronic many-body calculation, a big step was already made. Programming in `Haskell` essentially contributed not loosing track about the whole issue. Implementing the Hartree-Fock method in `Haskell` shows, that this functional programming language is well suited for scientific methods. Although for researchers with a background of imperative programming languages, starting to implement scientific methods in `Haskell` might be a bit cumbersome, the advantages of reproducibility and purity of functions as well as the coding structure makes

it up for it.

For the future work on this project it is highly preferable to consult other researchers that are familiar with the topic, since it quickly turned out that implementing electronic many-body systems from scratch is a very difficult task to tackle alone.

References

[1] Jos Thijssen. *Computational Physics* -. Cambridge University Press, Cambridge, 2. edition, 2007.

[2] Frank Jensen. *Introduction to Computational Chemistry* -. John Wiley & Sons, New York, 2013.

[3] Wikimedia Commons. Comparison of different sto-ng and sto, 2013.

[4] Karen L Schuchardt, Brett T Didier, Todd Elsethagen, Lisong Sun, Vidhya Gurumoorthi, Jared Chase, Jun Li, and Theresa L Windus. Basis set exchange: a community database for computational sciences. *Journal of chemical information and modeling*, 47(3):1045–1052, 2007.

[5] David Feller. The role of databases in support of computational chemistry calculations. *Journal of computational chemistry*, 17(13):1571–1586, 1996.

[6] Justin T Fermann and Edward F Valeev. Fundamentals of molecular integrals evaluation. Technical report, Tech. rep, 1997.

[7] Tomas Petersson and Bo Hellsing. A detailed derivation of gaussian orbital-based matrix elements in electron structure calculations. *European journal of physics*, 31(1):37, 2009.

[8] Peter MW Gill, Martin Head-Gordon, and John A Pople. An efficient algorithm for the generation of two-electron repulsion integrals over gaussian basis functions. *International Journal of Quantum Chemistry*, 36(S23):269–280, 1989.

[9] Sigeru Huzinaga. Basis sets for molecular calculations. *Computer Physics Reports*, 2(6):281–339, 1985.

[10] Richard C Raffenetti. General contraction of gaussian atomic orbitals: Core, valence, polarization, and diffuse basis sets; molecular integral evaluation. *The Journal of Chemical Physics*, 58(10):4452–4458, 1973.

[11] Pablo Echenique and J. L. Alonso. A mathematical and computational review of Hartree-Fock SCF methods in Quantum Chemistry. pages 105(23–24), 2007.