Martin-Luther-Universität
Halle-Wittenberg
Naturwissenschaftliche Fakultät II
Institut für Physik

Master Thesis

A thesis submitted in fulfillment of the requirements

Master of Science

Medical Physics

Machine Learning Optimization of Gaussian Basis Sets

Submitted by:
Jacob Schmieder
Matriculation Number: 215207342

Submission Date:
6. October 2022

Primary Supervisor: Prof. Dr. Miguel Marques
Second Supervisor: PD Dr. Jürgen Henk

Abstract

In this Thesis, machine learning algorithms are used to reduce the number of functions in a Gaussian basis set for ab initio DFT calculations. In theoretical quantum chemistry, the Density functional theory (DFT) is heavily used. For an accurate calculation, suitable Gaussian basis sets are indispensable. Large basis sets are required to do state-of-the-art ab initio DFT calculations. At the same time, the calculated systems are enormous. Therefore, numerous Gaussian basis sets have been created over the last several decades. Combining large systems with large basis sets leads to enormous computation time, even using large computer clusters. Machine learning algorithms can reduce the number of required basis functions for specific molecules. Therefore, different basis sets were optimized for over one hundred molecules. The results are discussed from the perspectives of physics and machine learning. While the machine learning process works, the basis sets do not significantly advance ab initio DFT calculations.

"It has been said that the tabulation of a function of one variable requires a page, of two variables a volume, and of three variables a library; but the full specification of a single wave function of neutral Fe is a function of seventy-eight variables. It would be rather crude to restrict to 10 the number of values of each variable at which to tabulate this function, but even so, full tabulation of it would require 10^{78} entries, and even if this number could be reduced somewhat from considerations of symmetry, there would still not be enough atoms in the whole solar system to provide the material for printing such a table."

- Dougals Rayner Hartree 1948[1]

Contents

Acronyms

AD automatic differentiation. 9, 31, 33–38

Adam adaptive moment estimation. 31, 42, 48

AI Artificial Intelligence. 27, 63

ao atomic orbital. 25

ASE Atomic Simulation Environment. 41

au atomic units. 11

bse basis set exchange. 41, 42

CGF contracted Gaussian function. 25

DFT Density functional theory. 2, 8, 11, 16, 17, 20, 21, 24–27, 33, 41, 42, 47, 48, 53, 63

DQC Differentiable Quantum Chemistry. 41, 42

gd gradient descent. 29–32, 48, 58, 62

GGA generalized gradient approximation. 21

GTO Gaussian-type-orbital. 24, 25, 40

HF Hartree-Fock Approximation. 12–18, 21, 22

KS Kohn-Sham Approach. 8, 17, 19, 21, 23

LCAO linear combination of atomic orbitals. 8, 21, 23

LDA local density approximation. 20, 21

LSD local spin-density. 20

LSDA local spin-density approximation. 20, 21

ML Machine Learning. 27, 28, 31, 33, 37, 39–42, 47–49, 57, 61, 62, 77

RHF restricted Hartree-Fock. 15, 16

STO Slater-type-orbitals. 24, 25, 47

UHF unrestricted Hartree-Fock. 15, 16

xc exchange-correlation potential. 18–21

Introduction

The quote from the beginning of this Thesis (page 3) originates from Dougals Rayner Hartree after he and Wladimir Alexandrowitsch Fock invented their first approximation to solve the Schrödinger equation. Furthermore, to find the Ground State Energy of a given System of Particles. The method was titled, by their inventors, the Hartree-Fock Method. It was one of the first approaches to solve the Schrödinger equation not exactly but throw a well-chosen Approximation. Hartree made it possible to solve complex quantum mechanical systems. While only for the hydrogen molecule does an analytical solution is known. Afterward, Walter Kohn and Lu Jeu Sham invented their Kohn-Sham Approach (KS), and the modern Density functional theory (DFT) was born. By applying the linear combination of atomic orbitals (LCAO) approach, which John Lennard-Jones invented in 1929, it was possible to use computers to run ab initio DFT calculations. With the invention of modern computer science, it was possible to solve more complex structures, from larger molecules to Protein structures. Nowadays, solving Density functional theory (DFT) calculations with thousands of atoms is possible. Using the LCAO approach requires an approximation of the exact wave function. These approximations are constructed by combining multiple functions of a particular type, called a basis set. A popular approach to the basis sets in chemistry is a combination of Gaussian-type functions. The first Gaussian basis set was invented by Hehre, Stewart, and Pople and constructed out of three Gaussian functions. Since the first invention of Gaussian basis sets, scientists creating entire libraries of basis sets like basis set exchange [3]. However, one problem does exist with all of these basis sets: Simultaneously by increasing the precession of the approaches, the amount of mandatory functions is also increasing rapidly. Consequently, more precise basis sets consist of more functions and require more calculation time. A small basis set consists of less primitive Gaussian functions than a more extensive basis set.
At the time of Hartree's statement, Computers were not much more than gigantic calculators, which required an entire factory hall to do simple operations. Nevertheless, until now, Computers are way more than just calculators. Today, computers can learn by themself due to machine learning algorithms. Which are invented by humans.
Therefore, the basic idea of this Thesis is:

Attempt to improve the quality of Gaussian basis sets using machine learning methods.

Therefore, the idea is to use two different basis sets. One reference basis set with many Gaussian functions as a basis set with a high degree of accuracy, and one initial basis set which is smaller and leads to a less accurate result in the DFT calculation. Those two basis sets are compared using a projection. The machine learning algorithm should try to improve the quality of the smaller basis set by comparing it to the larger one by

minimizing the Projection.
In Hartree's spirit, the goal is to reduce the amount of mandatory primitive Gaussian functions as he reduced the total amount of wave functions.

In this Thesis, modern machine learning frameworks that can utilize automatic differentiation (AD) methods are used and further developed. These Frameworks then will try to improve different basis sets for particular molecules provided by different databases, including over one hundred molecules. After thousands of conducted optimizations, the results will be evaluated. In the basics part and during the evaluation process, the reader will encounter information bout machine learning, Informatics, physics, and chemistry.

1 Basics

This chapter should give a good foundation for the results and the working code in the second part of this Thesis. It has to be said that this will be a breath introduction to all of these topics so that it will be possible to bring the results into a good perspective relative to their scientific background. Therefore, we want to split the basics up into three major sections. First, the basics of the density functional theory will be discussed and should provide the physics and chemistry background. After building these foundations, the topics will change to the informatics and machine learning side of this Thesis. In the machine learning part, the first thing will be a brief introduction about machine learning in general and what makes the code used during this Thesis so unique by describing how automatic differentiation works and whether it improves machine learning. We now want to begin with the basics of DFT.

1.1 Density functional theory

Before jumping directly into the density functional theory, let us start with the introduction to quantum chemistry, e.g., physics.

The Ansatz in quantum chemistry is simple: Try to find something that approximates the solution of the non-relativistic Schrödinger equation well enough. The stationary Schrödinger equation and therefore the Hamilton $\hat{H}$ describes a system consisting of M nuclei as well as N electrons in the form of:

$$\hat{H}\Psi_i(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N, \vec{R}_1, \vec{R}_2, \ldots, \vec{R}_M) = E_i\Psi_i(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N, \vec{R}_i, \vec{R}_2, \ldots, \vec{R}_M) \tag{1.1}$$

$$\hat{H} = \underbrace{-\frac{1}{2}\sum_{i=1}^{N}\nabla^2}_{\hat{T}_e} \underbrace{-\frac{1}{2}\sum_{A=1}^{M}\frac{1}{M_A}\nabla_A^2}_{\hat{T}_n} \underbrace{-\sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}}}_{\hat{U}_{en}} \underbrace{+\sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}}}_{\hat{U}_{nn}} \underbrace{+\sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A\,Z_B}{R_{AB}}}_{\hat{U}_{ee}} \tag{1.2}$$

Here i, j runs over the N (number of electrons) and A, B runs over M (the individual Nuclei)[4, p.3]. This system can be everything from a single particle, like an electron, over an Atom like hydrogen, or iron, to smaller or larger molecules like H_2 or ethanol (C_2H_5OH). Even large solids, amino acids, proteins, or anything else consisting of quantum mechanical partials can be described as such a system. In this Thesis, in particular, the small molecules will be relevant. To clarify equation 1.2 further, it is useful to split up the terms:

- $\hat{T}_e$ kinetic operator for all electrons in the system [4]
- $\hat{T}_n$ kinetic operator for all nuclei in the system [4]
- $\hat{U}_{en}$ total potential energy for the electron-nucleus Coulombic attraction [4]
- $\hat{U}_{ee}$total potential energy for the Coulombic electron-electron repulsion [4]
- $\hat{U}_{nn}$ total potential energy for the Coulombic nuclei-nuclei repulsion [4]

Ψ_i represents the wave function for the i'th state of the system. Ψ_i depends on the 3N spatial coordinates r_i as well as N spin coordinates s_i. Equation 1.2 is written in a very compact form without any physical constants. Instead of using physical constants to describe the system, in DFT, the atomic units (au) (also called Hartree atomic units) are used. The Hartree Units are displayed in figure 1.1.

Quantity	Atomic unit	Value in SI units	Symbol (name)
mass	rest mass of electron	9.1094 x 10^{-31} kg	m_e
charge	elementary charge	1.6022 x 10^{-19} C	e
action	Planck's constant/2π	1.0546 x 10^{-34} J s	$\hbar$
length	$4\pi\varepsilon_0\,\hbar$ / m_e e^2	5.2918 x 10^{-11} m	a_0 (bohr)
energy	$\hbar^2$ / m_e a_0^2	4.3597 x 10^{-18} J	E_h (hartree)

Figure 1.1: Tabular with the relevant Hartree atomic units [4]

The Schrödinger equation can further simplify, as there is a considerable difference between the masses of the nuclei and the electrons. Even the lightest ^{1}H has about 1800 times the mass of a single electron. Consequently, the nuclei are way slower than the electrons, and the system can be approximated by fixed nuclei and moving electrons [4]. To omit the movement of the nuclei is called *Born-Oppenheimer-* or *clamped-nuclei* approximation. The kinetic energy of the nucleus thus vanishes, and the potential energy of the nucleus-nucleus repulsion is more or less equal for every nucleus within a constant. The Hamiltonian reduces to the so-called electronic Hamiltonian [4, p. 5].

$$\hat{H}_{elec} = -\frac{1}{2}\sum_{i=1}^{V}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} = \hat{T} + \hat{V}_{Ne} + \hat{V}_{ee} \tag{1.3}$$

The solution of equation 1.3 is the electronic Energy, E_{elec} the nuclear coordination depends only on parameters, not explicitly on Ψ_{elec}. So the total energy is given by the sum of E_{elec} and the constant nuclear repulsion term:

$$E_{nuc} = \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{r_{AB}} \tag{1.4}$$

The Operator $\hat{V}_{Ne}$ converts into an external potential and may include magnetic or electronic problems (for example, an external magnetic field).
The wave function Ψ_i is not observable. But its squares represents the probability to find the electrons $1, 2, \ldots, N$ simultaneously in their volume $dV^N = dV_1 \ldots dV_N$, [4, p. 6]. Remember that electrons are fermions. By nature, they are indistinguishable (eq. 1.5) as well as antisymmetric with spin = 1/2 (eq. 1.6).

$$|\Psi(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_i, \vec{x}_j, \ldots, \vec{x}_N)|^2 = |\Psi(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_j, \vec{x}_i, \ldots, \vec{x}_N)|^2 \tag{1.5}$$

$$\Psi(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_i, \vec{x}_j, \ldots, \vec{x}_N) = -\Psi(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_j, \vec{x}_i, \ldots, \vec{x}_N) \tag{1.6}$$

Integrate over all possible states of the wave functions over the full space equals 1.

$$\int \cdots \int |\Psi(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)|^2 d\vec{x}_1 \, \vec{x}_2 \ldots \vec{x}_N = 1 \tag{1.7}$$

Then this is called normalized to one. From now on, all wave functions are normalized.

1.1.1 The Variational Principle

Up to this point, all is standard quantum mechanics. Unfortunately, there is no known way to solve the Schrödinger equation for large N. Nevertheless, there is a way to at least find an approximation using the *variational principle*. Therefore, a guess Ψ_{trial} for the expected ground state of the Hamilton Operator $\hat{H}$ represents the upper bound to the real ground state energy.

$$\langle \Psi_{trial} | \hat{H} | \Psi_{trial} \rangle = E_{trial} \geq E_0 = \langle \Psi_0 | \hat{H} | \Psi_0 \rangle \tag{1.8}$$

If $\Psi_{trial} \equiv \Psi_0$ equality holds. It's important to mention that Ψ_{trial} is a *functional* [1]. The task is to minimize $\mathbf{E}[\Psi]$ by going through all relevant (all functions that make physical sense, which means at least they should be normalized and square-integrable) N-electron wave functions [4, p.8].

$$E_0 = \min_{\Psi \to N} E[\Psi] = \min_{\Psi \to N} \langle \Psi | \hat{T} + \hat{V}_{Ne} + \hat{V}_{ee} | \Psi \rangle \tag{1.9}$$

Of course, it is not possible to calculate all allowed functions. Instead, use a subset of the given wave functions and find the best approximation for E_0 within this subset of Ψ. Which is the basic idea of the Hartree-Fock Approximation (HF) [4].

1.1.2 Hartree-Fock Approximation

One of the fundamental concepts of DFT is the Hartree-Fock Approximation (HF). A simple, practical way for a many-electron system is to define an *antisymmetric* product of N one-electron wave functions. Usually referred to as a *Slater determinant*[4]:

[1] for more information, check out [5]

$$\Psi_0 \approx \Phi_{SD} = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\vec{x}_1) & \chi_2(\vec{x}_1) & \dots & \chi_N(\vec{x}_1) \\ \chi_1(\vec{x}_2) & \chi_2(\vec{x}_2) & \dots & \chi_N(\vec{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_1(\vec{x}_N) & \chi_2(\vec{x}_N) & \dots & \chi_N(\vec{x}_N) \end{vmatrix} \tag{1.10}$$

$$= \frac{1}{\sqrt{N!}} \det\{\chi_1(\vec{x}_1)\chi_2(\vec{x}_2)\dots\chi_N(\vec{x}_N)\} \tag{1.11}$$

The Slater determinant is asymmetric, which means it changes its sign upon exchange of two rows or two columns. $\chi_i(\vec{x}_i)$ is called *spin orbital* and is an effective one-particle problem. $\chi_i(\vec{x}_i)$ can be constructed through spatial orbitals ϕ_i and one of the two spin functions $\alpha(s)$ and $\beta(s)$ where $s = \pm\frac{1}{2}$.

$$\chi(x) = \phi(r)\,\sigma(s) \quad , \sigma = \alpha, \beta \tag{1.12}$$

$\alpha(s)$ and $\beta(s)$ has to be orthogonal, that means:

$$\langle\alpha|\alpha\rangle = \langle\beta|\beta\rangle = 1$$
$$\langle\alpha|\beta\rangle = \langle\beta|\alpha\rangle = 0$$

Consequently, also the spin orbitals are primarily defined as orthogonal:

$$\int \chi_i^*(\vec{x})\chi_i(\vec{x})d\vec{x} = \langle\chi_i|\chi_j\rangle = \delta_{ij} \tag{1.13}$$

Where δ_{ij} refers to the Kronecker delta symbol. In the HF scheme, the spin orbitals are chosen analogous to the wave function in quantum physics. To summarize, $|\chi(\vec{x})|^2 d\vec{x}$ is the probability to find the electron with spin σ in a given volume element $d\vec{x}^3$. The prefactor gives the normalization $\frac{1}{\sqrt{N!}}$. So an N-electron wave function Ψ_{exact} is replaced by a single Slater determinant Φ_{SD}. Now the variational principle can be used to find the best Slater determinant. For the Hartree-Fock Energy, this looks like this:

$$E_{HF} = \min_{\Phi_{SD}\to N} E[\Phi_{SD}] \tag{1.14}$$

Hence the expectation of the Hamilton Operator is given by:

$$E_{HF} = \langle\Phi_{SD}|\,\hat{H}\,|\Phi_{SD}\rangle = \sum_i^N (i|\hat{h}|i) + \frac{1}{2}\sum_i^N\sum_j^N (ii|jj) - (ij|ji) \tag{1.15}$$

Where:

$$(i|\hat{h}|i) = \int\int |\chi^*(\vec{x_i})|\left\{ -\frac{1}{2}\nabla^2 - \sum_A^M \frac{Z_A}{r_{1A}} \right\}|\chi(\vec{x_i})|d\vec{x}_1 \tag{1.16}$$

Equation 1.16 defines the contribution due to the kinetic energy and the electron-nucleus attraction. The last term in (1.14) can be split into the so-called *Coulomb* (eq. 1.17)and *exchange* (eq. 1.18) integrals.

$$(ii|jj) = \int\int |\chi_i(\vec{x}_1)|^2 \frac{1}{r_{12}} |\chi_j(\vec{x}_2)|^2 d\vec{x}_1 d\vec{x}_2 \tag{1.17}$$

$$(ij|ji) = \int\int \chi_i(\vec{x}_1)\chi_j^*(\vec{x}_1)\frac{1}{r_{12}}\chi_j(\vec{x}_2)\chi_i^*(\vec{x}_2) d\vec{x}_1 d\vec{x}_2 \tag{1.18}$$

E_{HF} is again a functional of the set of $\{\chi_i\}$, which has to be orthogonal. To ensure the orthogonality, the *Lagrangian multipliers* ϵ_i will be introduced. The physical interpretation of the ϵ_i is the orbital energy. The best spin Orbitals can be described by the *Hartree-Fock equations* (where $\{\chi_i\}$, E_{HF} is minimal)[4]

$$\hat{f}\chi_i = \epsilon_i \chi_i \, , \, i = 1, 2, \ldots, N \tag{1.19}$$

$$\hat{f}_i = -\frac{1}{2}\nabla_i^2 - \sum_A^M \frac{Z_A}{r_{iA}} + V_{HF}(i) \tag{1.20}$$

The Fock Operator $\hat{f}$ is an effective one-electron operator. The first two terms of $\hat{f}_i$ are already known. The new introduced $V_{HF}(i)$ is the HF potential: It is the average repulsive potential experienced by the i'th electron due to the remaining N-1 electrons.[4, p. 11] As a consequence, the two-electron repulsion operator $\frac{1}{r_{ij}}$ simplifies to a one-electron operator, where the repulsion takes place just in an average way. Explicitly:

$$V_{HF}(\vec{x}_1) = \sum_j^N \left(\hat{J}_j(\vec{x}_1) - \hat{K}_j(\vec{x}_1) \right) \tag{1.21}$$

Where $\hat{J}_j$ is the Coulomb Operator:

$$\hat{J}_j(\vec{x}_1) = \int |\chi_j(\vec{x}_2)|^2 \frac{1}{r_{12}} \, d\vec{x}_2 \tag{1.22}$$

The electron on position $\vec{x}_1$ experiences the average charge distribution of another electron χ_j. $\hat{J}_j(\vec{x}_1)$ is just weighed by the probability that another electron is at the same point in space. Consequently, $\hat{J}_j(\vec{x}_1)$ only depends on one spin orbital $\chi_j(\vec{x}_1)$ and his position, $\vec{x}_1$ is called *local*. The second term $\hat{K}_j(\vec{x}_1)$ is the exchange contribution and has no classical interpretation. However, it can be defined by the effect when an operation on a spin-orbital:

$$\hat{K}_j(\vec{x}_1)\chi_i(\vec{x}_1) = \int \chi_j^*(\vec{x}_2)\frac{1}{r_{ij}}\chi_i(\vec{x}_2) \, d\, \vec{x}_2 \chi_j(\vec{x}_1) \tag{1.23}$$

because $\hat{K}_j(\vec{x}_1)$ depends not only on a single spin orbital χ_i but on another spin orbital, χ_j consequently it's called *non-local*. Since the spin orbitals are orthogonal, only parallel spins exist. In case of antiparallel spins the therms, $\langle\alpha(s_2)|\beta(s_2)\rangle$ e.g., $\langle\beta(s_2)|\alpha(s_2)\rangle$ are zero and vanishes the integral. The $\frac{1}{r_{12}}$ operator is spin independent. It is important to mention that the term $i = j$ is allowed in eq. 1.21. Therefore, the result of a single electron repulsion with itself is more significant than zero. This is called *self-interaction*.

Nevertheless, even then, the equation takes care. The exchange term for $i = j$ reduces to, $\int\int|\chi_i(\vec{x}_1)|^2 \frac{1}{r_{12}}|\chi_i(\vec{x}_2)|^2 d\vec{x}_1 d\vec{x}_2$ so the self-interaction is canceled by eq. 1.21. Because of eq. 1.19 is not a regular *eigenvalue problem*, rather it's a *pseudo-eigenvalue problem* and has solved iteratively. Therefore, it's called *self-consistent field* (SCF) [4]. We guess a set of finite basis sets to solve this iteratively to expand the molecular orbitals (shown in more detail in the KS scheme like figure 1.3).

1.1.2.1 Electron Correlation

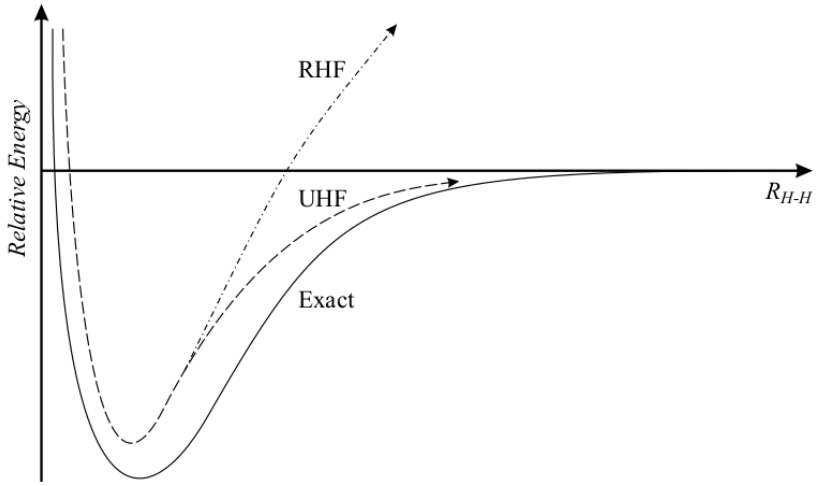


Figure 1.2: Difference in the relative Energy in Hartree of H_2, for different Models. The exact value comes by solving the two-electron Schrödinger equation. unrestricted Hartree-Fock (UHF) describe the unrestricted HF method and is more precise than the restricted HF (restricted Hartree-Fock (RHF)), further away.

$$E_C^{HF} = E_0 - E_{HF} \tag{1.24}$$

As the Hartree-Fock method is an approximation, the *correlation energy* E_C^{HF} declares the difference between the calculated minimal HF Energy and the actual ground state Energy (done by Löwdin [6] and within the Born- Oppenheimer approximation). E_C^{HF} has to be always negative since E_0 and E_{HF} are smaller than zero as well as $|E_0| > |E_{HF}|$. In figure 1.2 the difference caused by the electron correlation is shown. The primary reason for the difference in the energy is caused by instantaneous repulsion by the electrons in the effective HF potential. This effect is directly connected with the *dynamic* correlation caused by the $1/r_{12}$ term. On the other hand, there is a *non-dynamical* term that results from the relatively bad approximation of the ground state by the Slater determinants. As shown in fig. 1.2 the RHF yields a good approximation for the H2 molecule at the ground. In the RHF case, the electrons build up a system that can be handled as a singlet system (e.g., water). The most common solution is to use

doubly occupied spatial orbital ϕ_a by opposite spins χ_a and χ_b. [4, p.14] The dynamic correlation error (the major part for H_2) is just $0.04E_h$. If the distance increases, the RHF diverges. A better approach then is the UHF where different orbitals, for example α and β experience different potentials, V_{HF}^{α} e.g., V_{HF}^{β}. The Slater determinant does not yield an eigenfunction of the total spin operator, $\langle S^2 \rangle$, so it is less physical meaningful [4, p.14]

1.1.3 Hohenberg-Kohn Theorems

The Paper of Hohenberg and Kohn from 1964 [7] laid the foundation for the later discovered Density functional theory (DFT). Where in the HF method, the actual position of the electrons is necessary, Hohenberg and Kohn introduce the electronic density and claim that the electronic density could fully define the ground state of the system:

> the external potential $V_{ext}(\vec{r})$ is (to within a constant) a unique functional of $\rho(r)$; since, in turn $V_{ext}(\vec{r})$ fixes, $\hat{H}$ we see that the full many particle ground state is a unique functional of $\rho(r)$. [4, 7]

The proof can be done by reductio ad absurdum, see [4, 7]. $\rho(r)$ is the electron density and can be defined as:

$$\rho(\vec{r}) = N \int \cdots \int \left|\Psi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_N})\right|^2 d\vec{s_1} d\vec{x_2} \ldots d\vec{x}_N \tag{1.25}$$

> $\rho(r)$ determines the probability of finding any of the N electrons in the given volume element $d\vec{r}_1$ but with arbitrary spin, while the other N-1 electrons have arbitrary positions and spin in the state represented by Ψ. [4]

The electronic density was first introduced simultaneously but independent by Enrico Fermi and Llewellyn Thomas in 1927. According to Hohenberg-Kohn, it is possible to separate the energy expression into two different parts. The first part can be expressed as:

$$E_0[\rho_0] = \underbrace{\int \rho_0(\vec{r}) V_{Ne} d\vec{r}}_{\text{system dependend}} + \underbrace{T[\rho_0] + E_{ee}[\rho_0]}_{\text{universal valid}} \tag{1.26}$$

further, it is possible to define the universal *Hohenberg-Kohn functional*:

$$F_{HK}[\rho] = T[\rho] + E_{ee}[\rho] = \langle \Psi | \hat{T} + \hat{V}_{ee} | \Psi \rangle \tag{1.27}$$

To solve the Hohenberg-Kohn functional is unfortunately nearly impossible for neither $\langle \Psi | \hat{T} | \Psi \rangle$ nor $\langle \Psi | \hat{V}_{ee} | \Psi \rangle$ for molecules bigger than hydrogen, it would be the *holy grale* of DFT to find the exact Functional [4, 8]. To overcome this problem, the second Hohenberg-Kohn Theorem uses the variational principle.

$$\langle \Psi | \hat{H} | \Psi \rangle = \langle \Psi | T[\rho] + E_{ee}[\rho] + v_{ext} | \Psi \rangle = \int \rho_0(\vec{r}) V_{Ne} d\vec{r} + T[\rho_0] + E_{ee}[\rho_0] \geq E_0[\rho_0] \tag{1.28}$$

Therefore, the calculated ground state energy using the electronic density will always be higher than the actual ground state energy.

1.1.4 Kohn-Sham Approach

The Hohenberg-Kohn Theorems are building up the foundation of the Kohn-Sham Approach Approach and what will be called Density functional theory in modern days. The basic idea of Kohn-Sham DFT is to map a system of interaction electrons onto a system with non-interacting electrons and express the difference between those two systems by an effective potential, including the unknown exchange-correlation potential.

Therefore, Hohenberg-Kohn already defines the ground state of an atomic or a molecular system as:

$$E_0 = \min_{\rho \to N}(F[\rho] + \int \rho(\vec{r}) V_{Ne} d\vec{r}) \tag{1.29}$$

$$F[\rho] = T[\rho(\vec{r})] + J[\rho(\vec{r})] + E_{ncl}[\rho(\vec{r})]] \tag{1.30}$$

In the spirit of the Hartree-Fock Approximation, it is possible to define a *non-interacting reference system* with a Hamiltonian in the form of:

$$\hat{H}_S = -\frac{1}{2}\sum_i^N \nabla_i^2 + \sum_i^N V_S(\vec{r}_i) \tag{1.31}$$

where $V_S(\vec{r}_i)$ is an effective local potential. The ground state can be constructed by a single *Slater determinant* (e.g., equation 1.10):

$$\Theta_S = \frac{1}{\sqrt{N!}} \begin{vmatrix} \varphi_1(\vec{x}_1) & \varphi_2(\vec{x}_1) & \dots & \varphi_N(\vec{x}_1) \\ \varphi_1(\vec{x}_2) & \varphi_2(\vec{x}_2) & \dots & \varphi_N(\vec{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(\vec{x}_N) & \varphi_2(\vec{x}_N) & \dots & \varphi_N(\vec{x}_N) \end{vmatrix} \tag{1.32}$$

For clarity in eq. 1.32 the nomenclature has changed as the ϕ_i now the *Kohn-Sham orbitals*. In analogy to HF the one electron, the Kohn-Sham operator can be defined as:

$$\hat{f}^{KS}\varphi_i = \epsilon_i \varphi_i \tag{1.33}$$

$$\hat{f}^{KS} = -\frac{1}{2}\nabla^2 + V_S(\vec{r}) \tag{1.34}$$

The density is expressed throw the Kohn-Sham orbitals by a summation of the different squared orbitals φ_i. By the summation over all orbitals, the density exactly equals the ground state density of the real system of electrons[4, p. 43].

$$\rho_S(\vec{r}) = \sum_i^N \sum_s |\varphi_i(\vec{r}, s)|^2 = \rho_0(\vec{r}) \tag{1.35}$$

The Kohn Shams electron is a non-interacting particle sharing the same density as its interacting counterparts. The solution of a non-interacting system is not equal to an interacting one, so the kinetic energy is not equal.

$$T_{KS} \neq T_{real} \tag{1.36}$$

To overcome this problem, put everything unknown into a new potential and call it exchange-correlation potential (xc) potential. The potential itself is not known explicitly but yields the relevant information [8].

$$E_{XC}[\rho] \equiv (T[\rho] - T_S[\rho]) + (E_{ee}[\rho] - J[\rho]) = T_c[\rho] = E_{ncl}[\rho] \tag{1.37}$$

As expected, E_{XC} is a functional of the density, ρ. T_S is the kinetic energy of the none interaction Kohn-Sham orbitals:

$$T_S = -\frac{1}{2}\sum_i^N \langle\varphi_i|\,\nabla^2\,|\varphi_i\rangle \tag{1.38}$$

The kinetic energy does not directly dependent on the density but the Kohn-Sham orbitals. Which are determined by the density (eq. 1.35). The effective local potential V_S (e.g. 1.34) now includes the exchange potential.

$$\begin{aligned}\Big(-\frac{1}{2}\nabla^2 + \Big[\int\frac{\rho(\vec{r}_2)}{r_{12}} + V_{XC}(\vec{r}_1) - \sum_A^M\frac{Z_A}{r_{1A}}\Big]\Big)\varphi_i \\ = \Big(-\frac{1}{2}\nabla^2 = V_S(\vec{r}_1)\Big)\varphi_i = \epsilon_i\varphi_i\end{aligned} \tag{1.39}$$

Which is again eq. 1.34 but now V_S has been defined throw the exchange-correlation potential.
Consequently, the Kohn-Sham orbitals can be calculated in analogy to the HF using an iterative procedure (as shown in fig. 1.3).

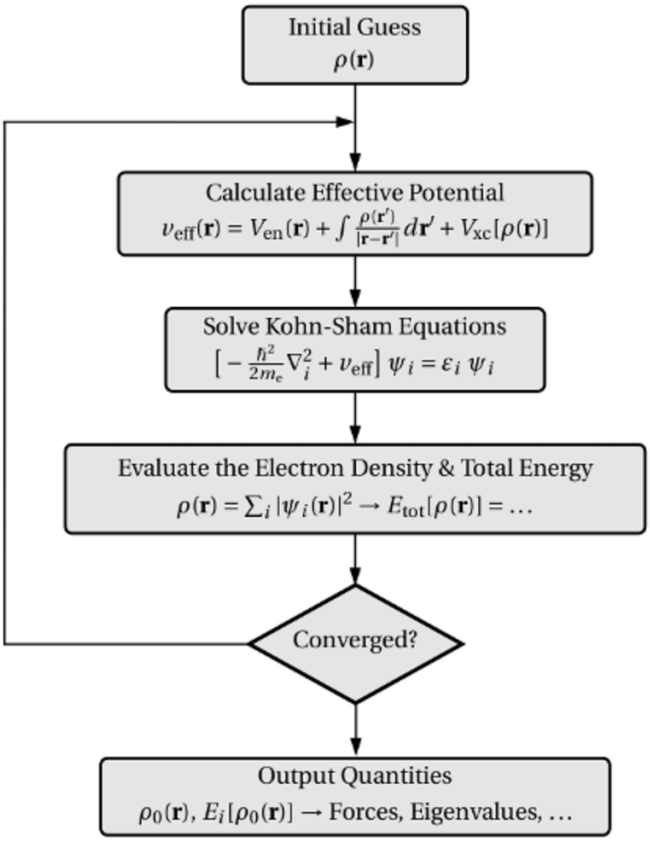


Figure 1.3: Self-consistency cycle in DFT calculations. Starting with an initial guess, then calculating the effective potential and solving the Kohn-Sham Approach (KS) equation. Evaluating the outcome and, if it is converged, output the result. If it is not converged, begin the cycle and use the current result as a new input.[9]

The only thing left is the exact Functional of V_{XC} respectively E_{XC}. Since the term is still unknown, it has just been defined as a functional derivative of E_{XC}.

$$V_{XC} \equiv \frac{\delta E_{XC}}{\delta \rho} \tag{1.40}$$

If $E_{XC}(\rho(\vec{r}))$ it were known, it would be possible to determine the ground state [4]. Hence, V_{XC} is not precisely known. Several theoreticians published various versions of approximations to the exchange-correlation potential [10].

1.1.5 The exchange-correlation potential

Since the basic idea of inventing the xc Functional was to put everything unknown, it is impossible to express it precisely. There are a couple of methods known to approximate the xc potential.

The xc only depends on the electron density in a homogenous electron gas. In more realistic approaches, the V_{xc} depends on the density of position $\vec{r}$ itself but their derivatives.

$$V_{xc}[\rho]|_{\vec{r}} = X_{xc}[\rho, \nabla\rho, \nabla^2\rho, \dots]|_{\vec{r}} \tag{1.41}$$

Here the different methods will be discussed very briefly, as in the most DFT codes, it is possible to use most methods out of the box, and no further information is mandatory to handle these approximations. Therefore, in more detail, only the LDA and the B3LYP potential will be discussed in this Thesis. The simplest method is the local density approximation (LDA) Here V_{xc} only depends on the density at the position $\vec{r}$ but not on the gradients [11].

In, LDA the energy can express as:

$$E_{xc}[\rho] = \int \rho(\vec{r})\epsilon_{xc}[\rho]|_r dr^3 \tag{1.42}$$

where:

$$\epsilon_{xc}[\rho] = \epsilon_x^{Dirac}[\rho] = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{\frac{1}{3}} \rho^{1\backslash 3} \tag{1.43}$$

Does Dirac introduce electronic density? Often ϵ_{xc} is reffed to the average inter-electronic distance r_s

$$\epsilon_x^{Dirac} \propto -\frac{1}{r_s} \qquad r_s = \left(\frac{3}{4\pi\rho}\right)^{\frac{1}{3}} \tag{1.44}$$

As found by Slater, the exchange interaction of an electron with a particular spin creates a surrounding volume where the probability of finding another electron with an equal spin is reduced [11]. This so-called **exchange hole** leads in the **local spin-density (LSD)** which takes it into account. Int the LSD describes a uniform density inside a sphere with the radius r_0 and zero elsewhere. Then r_0 has the size of the area, where the probability of finding a second electron with equal spin is lowered [11].

$$r_0 = \left(\frac{3}{4\pi\rho_\uparrow}\right)^{\frac{1}{3}} \tag{1.45}$$

The correlation part is usually obtained from quantum Monte Carlo simulations. One of the simplest approaches is given by J. P. Perdew, and A. Zunger [12]

$$\epsilon_c^{PZ} = \begin{cases} A\,\ln(r_s) + B + C\,r_s\,\ln(r_s) + D\,r_s & r \le 1 \\ \frac{\gamma}{1+\beta_1\sqrt{r_s}=\beta_2 r_s} & r_s > 1 \end{cases} \tag{1.46}$$

As a further improvement to the LDA the local spin-density approximation (LSDA) should be mentioned.

$$E_{LSDA}[\rho_\alpha, \rho_\beta] = \int d\vec{r}\rho(\vec{r})\epsilon_{xc}\left[\rho_\alpha, \rho_\beta\right] \tag{1.47}$$

where the spin density is by $\rho = \rho_\alpha + \rho_\beta$.

If the density is not homogenous, the LDA gets on its limits. Then

$$\rho(\vec{r}) = \rho_0 = \Delta\rho(\vec{r}) \qquad \left|\Delta\rho(\vec{(}r)\right| \ll \rho_0 \tag{1.48}$$

is no longer fulfilled. Here are the **generalized gradient approximation (GGA)** steps in, and the xc energy now depends also on the gradients of ρ.

$$E_{xc}^{GGA}[\rho] = \int \epsilon_{xc}[\rho, \nabla\rho, \nabla^2\rho]|_{\vec{r}} dr^3 \tag{1.49}$$

The most common one is the PBE potential from Perdew, Burke, and Ernzerhof [13]. GGA can be improved by including the kinetic-energy density, called **Meta-GGA's**. For this Thesis, a Hybrid functional is used.

Hybrid functionals combining the Hartree-Fock and DFT by using correlation taken from LDA and the exchange from Hartree-Fock Approximation (HF) [11]. The most common Hybrid functional is the **B3LYP** Potential by Becke and Lee–Yang–Parr [4].

$$E_{xc}^{B3LYP} = (1-a)\, E_X^{LSDA} + a\, E_x^{HF} + b\, \Delta E_x^{B88} + (1-c)\, E_c^{LSDA} + c\, E_c^{LYP} \tag{1.50}$$

where:

$a = 0.20$, $b = 0.72$, $c = 0.81$

E_x^{LSDA} Becke 1993 exchange functional (LSDA)

E_x^{HF} exact Hartree-Fock potential

E_x^{B88} is the Becke 88 exchange functional (GGA) [14]

E_c^{LSDA} S. H. Vosko; L. Wilk; M. Nusair correlation potential [15]

E_c^{LYP} the Lee, Yang and Parr correlation functional [16]

With B3LYP, the absolute error concerning the G2 database is approximately $2 kcal/mol$, which is quite good. The G2 database will be used in this Thesis; it will be explained later [17]. As for simplicity, the functional is still heavily used in modern days, although there are a bunch of better but more complex functional which lead to much more necessary computational power [4].

1.1.6 The LCAO Ansatz in the Kohn-Sham Equations

After discussing the KS scheme and how to handle the exchange-correlation potential, it is time to think about actually using this theory. In order to use the KS approach, it is mandatory to create an efficient algorithm that is easy to compute, which leads to the linear combination of atomic orbitals (LCAO) Ansatz. Other mentionable approaches are, for example, the tight binding method or full numerical atomic orbitals. The one-electron Kohn-Sham Approach equation looks like (compare equation 1.39):

$$\Big(-\frac{1}{2}\nabla^2 + \Big[\sum_j^N \int \frac{|\varphi(\vec{r}_2)|^2}{r_{12}} + V_{XC}(\vec{r}_1) - \sum_A^M \frac{Z_A}{r_{1A}} \Big] \Big) \varphi_i = \hat{\mathrm{f}}^{KS} \varphi_i = \epsilon_i \varphi_i \tag{1.51}$$

The HF exchange operator (eq. 1.23) is known too.

Accordingly, a molecular orbital can be expressed by a linear combination of L predefined basis functions $\{\eta_\mu\}$.

$$\varphi_i = \sum_{\mu=1}^{L} c_{\mu i}\, \eta_\mu \tag{1.52}$$

Usually called basis sets. If L were infinity, the set was complete. However, for obvious reasons, this will never be the case in real-world applications. The nonlinear problem is now simplified to linear, with unknown coefficients $c_{\mu i}$. The first basis sets, in this sense, were inspired by the real eigenfunctions functions of the hydrogen atom. Furthermore, it explains why these are called "atomic orbitals." Inserting equation 1.52 into the 1.51 leads to [4]:

$$\begin{aligned} \hat{\mathrm{f}}^{KS} \varphi_i &= \varepsilon_i \varphi_i \\ \hat{\mathrm{f}}^{KS} \sum_{\mu=1}^{L} c_{\mu i}\, \eta_\mu &= \varepsilon_i \sum_{\mu=1}^{L} c_{\mu i}\, \eta_\mu \end{aligned} \tag{1.53}$$

which is closely related to the HF scheme. by multiply the left side with an arbitrary function η_ν and integrate over space, L equations are received [4].

$$\sum_{\nu=1}^{L} c_{\nu i} \underbrace{\int \eta_\mu(\vec{r}_1) \hat{\mathrm{f}}^{KS}(\vec{r}_1) \eta_\nu(\vec{r}_1) d\vec{r}_1}_{\hat{\mathbf{F}}^{\mathbf{KS}}_{\mu\nu}\textbf{- Kohn-Sham matrix}} = \varepsilon_i \sum_{\nu=1}^{L} c_{\nu i} \underbrace{\int \eta_\mu(\vec{r}_1) \eta_\nu(\vec{r}_1) d\vec{r}_1}_{\hat{\mathbf{S}}_{\mu\nu}\textbf{- overlap matrix}} \quad , 1 \le i \le L \tag{1.54}$$

On both sides, the integrals are defining a matrix ($\hat{F}^{KS}_{\mu\nu}$ e.g., $\hat{S}_{\mu\nu}$) with dimension $L \times L$ and symmetry $M_{\mu\nu} = M_{\nu\mu}$ as well as self-adjoint or hermitian. Using $\mathbf{S}$ and $\mathbf{F}$, the coefficients of linear combinations can be expressed as a Matrix of dimension $L \times L$. ε is a diagonal Matrix containing the orbital energies

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1L} \\ c_{21} & c_{22} & \dots & c_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ c_{L1} & c_{L2} & \dots & c_{LL} \end{pmatrix} \qquad \varepsilon = \begin{pmatrix} \varepsilon_1 & 0 & \dots & 0 \\ 0 & \varepsilon_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \varepsilon_L \end{pmatrix} \tag{1.55}$$

The preceding Matrices can be combined into a compact equation [4]:

$$\mathbf{F^{KS}}\, \mathbf{C} = \mathbf{S}\, \mathbf{C}\, \varepsilon \tag{1.56}$$

This generalized eigenproblem is solved using standard algorithms from linear algebra. The Kohn-Sham matrix F^{KS} become [4]:

$$\begin{aligned} F_{\mu\nu}^{KS} &= \int \eta_\mu(\vec{r}_1)\Big(-\frac{1}{2}\nabla^2 - \sum_A^M \frac{Z_A}{r_{1A}} + \int \frac{\rho(\vec{r}_2)^2}{r_{12}} + V_{XC}(\vec{r}_1)\Big)\eta_\nu(\vec{r}_1)d\vec{r}_1 \\ &= -\frac{1}{2}\int \eta_\mu(\vec{r}_1)\nabla^2\eta_\nu(\vec{r}_1)d\vec{r}_1 - \int \eta_\mu(\vec{r}_1)\sum_A^M \frac{Z_A}{r_{1A}}\eta_\nu(\vec{r}_1)d\vec{r}_1 \\ &\quad + \int\int \eta_\mu(\vec{r}_1)\frac{\rho(\vec{r}_2)^2}{r_{12}}\eta_\nu(\vec{r}_1) + \int \eta_\mu(\vec{r}_1)V_{XC}(\vec{r}_1)\eta_\nu(\vec{r}_1)d\vec{r}_1 \end{aligned} \tag{1.57}$$

The first and the second part of equation 1.57 are electronic kinetic energy and the electron-nuclear interaction. These only depend on the coordinates of one electron and can be combined into a single integral.

$$h_{\mu\nu} = \int \eta_\mu(\vec{r}_1)\Big(-\frac{1}{2}\nabla^2 - \sum_A^M \frac{Z_A}{r_{1A}}\Big)\eta_\nu(\vec{r}_1)d^3r_1 \tag{1.58}$$

The one-electron contribution $h_{\mu\nu}$ can be calculated relatively easily using specific, well-known fast algorithms. In the LCAO scheme, the density takes the form:

$$\rho(\vec{r}) = \sum_i^N \left|\varphi_i(\vec{r})\right|^2 = \sum_i^N \sum_\mu^L \sum_\nu^L c_{\mu i}\, c_{\nu i}\, \eta_\mu(\vec{r}_1)\, \eta_\nu(\vec{r}_1) \tag{1.59}$$

As the expansion coefficients carry all relevant information, equation 1.59 can be reduced to:

$$P_{\mu\nu} = \sum_i^N c_{\mu i}\, c_{\nu i} \tag{1.60}$$

$P_{\mu\nu}$ is the *density matrix*. For each element $\mu\nu$. The Coulomb contribution of equation 1.57 leads to four-center-two-electron integrals (since four basis functions are necessary for two electrons, e.g. atoms).

$$J_{\mu\nu} = \sum_\lambda^L \sum_\sigma^L P_{\lambda\sigma} \int\int \eta_\mu(\vec{r}_1)\, \eta_\nu(\vec{r}_1)\, \frac{1}{r_{12}}\, \eta_\lambda(\vec{r}_2)\, \eta_\sigma(\vec{r}_2)\, d\vec{r}_1\, d\vec{r}_2 \tag{1.61}$$

Up to this point, all the previous can similarly apply to the Hartree-Fock case. For the correlation part, which is relevant for the Kohn-Sham Approach (KS), the exchange integral looks like this:

$$V_{\mu\nu}^{XC} = \int \eta_\mu(\vec{r}_1)V_{XC}(\vec{r}_1)\eta_\nu(\vec{r}_1)d\vec{r}_1 \tag{1.62}$$

Now, the important question is which basis sets exist and can be used in the LCAO approach [4].

1.1.7 Basis Sets

Since DFT is a heavily used theory in quantum chemistry and physics, many approaches to basis functions were introduced. Multiple different types of basis functions for various use cases exist. In solid state physics, it is practical to use plane wave functions [10]. In chemistry, the most used basis sets are the Gaussian-type-orbital (GTO) [18]. In this Thesis, only the GTO basis is going to be discussed in more detail, but for completeness, the difference between the GTO and the plane waves listed in a short tabular 1.1.

	GTO	plane waves
(+)	atomic orbital-like Compact basis set Analytic integration possible for many operators Optimal for regular grids. Fourier transform again a Gaussian	orthogonal independent of atomic positions
(-)	Non-orthogonal basis Linear dependencies for larger basis sets. Complicated to generate and no easy way to improve. Basis set superposition error (BSSE) Molecules (wavefunction tails), solids have different requirements	naturally periodic many functions needed

Table 1.1: Difference between plane waves and GTO [19] plane waves are usually used in solid state physics. GTO basis sets describe molecules or proteins in chemistry.

The goal when using a Gaussian basis is to simulate an atomic orbital function. Therefore, from a historical and physical standpoint, the first basis was inspired by orbitals quite similar to the orbitals of the hydrogen atom [4].

$$\eta^{STO} = N\, r^{n-1} \exp\{-\zeta r\} Y_{lm}(\theta, \phi) \tag{1.63}$$

where $Y_{lm}(\theta, \phi)$ are the spherical harmonics. ζ controls the width of the orbital. The main advantage of Slater-type-orbitals (STO) is their correct behavior for short- and long-range interaction. Unfortunately, the many-center integrals (like in section 1.1.6) are notoriously hard to calculate for STO functions. Because this is a significant disadvantage, Gaussian-type-orbital (GTO) basis functions are invented. A GTO function looks like:

$$\eta^{GTO} = N x^l y^m z^n \exp\{-\zeta r^2\} \tag{1.64}$$

where N is the normalization factor to ensure $\left|\langle \eta_\mu | \eta_\mu \rangle\right|^2 = 1$ (where in general $\langle \eta_\mu | \eta_\nu \rangle \neq 0$ for $\mu \neq \nu$). ζ represents the width of the orbital again. $L = l+m+n$ defines the orbital (*s, p, d*). For $L > 1$ the number of required function exceeds by $2L+1$. The main advantage

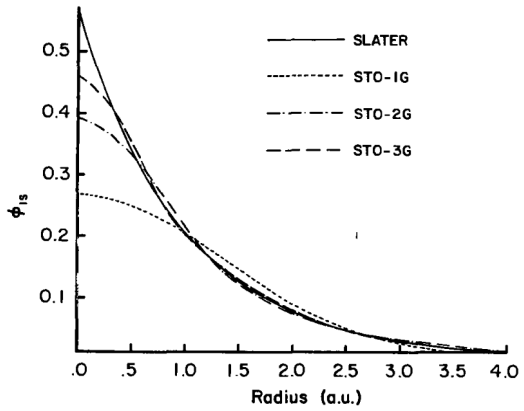


Figure 1.4: Comparison of the quality of the least-square fit of a 1s Slater function ($\zeta = 1.0$) obtained at the STO-1G, STO-2G, STO-3G level. [20, p. 158]

of GTO basis leads to the *Gaussian product theorem*, which says that a product of two Gaussians is again a Gaussian. The disadvantage of GTO basis is the inaccurate long-range behavior. A solution for this problem is relatively simple. Combining multiple Gaussian functions by a linear combination mimics a STO basis.

$$\eta_\tau^{CGF} = \sum_a^A d_{a\tau} \eta_a^{GTO} \tag{1.65}$$

η_τ^{CGF} then called contracted Gaussian function (CGF) [4] and is the primarily used type of basis set in modern Kohn-Sham DFT [18].

The first, simplest and least accurate contracted Gaussian function (CGF) is the *STO-nG* (where n is an integer number) basis set (as it is a CGF to mimic a STO basis). In figure 1.4, three different STO-nG basis sets are shown. As more Gaussians are in an CGF as better an approximation will be. Going more into detail on which type of Gaussian basis functions there are.

A **minimal** basis set contains one basis function (CGF of multiple Gaussians) for each atomic orbital (ao) of the atom (STO, GTO, CGTO)[18, 4]. In the Lithium atom, for example, just two functions are mandatory to describe the atom (one 1s and one 2s function). If there are multiple basis functions to describe an atomic orbital these are called *zeta-* functions. For example, if there are two basis functions which describe one ao it is called a *double-zeta* basis. If three basis functions describe one ao it is called *triple-zeta* basis. Beyond this also quadrupole-, quintuple-zeta up to the 7th degree exist. In the Lithium example, a *double-zeta* consists of four basis functions. For a *triple-zeta* basis, already six functions are required again for each orbital. For larger and more complex systems, the number of basis functions increases rapidly with the higher zeta value (the required computational power). A **split-valence** basis set combines the advantage of the zeta and minimal type basis (3-21G or 6-31G) [4]. In this case, only one basis function is used to describe the core atomic orbital (because these are usually

irrelevant for chemistry). More extensive basis sets are used for the valence orbitals. Counting them using 3-21G as an example basis set (a CGTO basis set): Carbon has one contracted Gaussian function as a core function, a linear combination of three Gaussian functions.
To simulate two interacting atoms, *polarization functions* are invented. A polarization function adds one more atomic orbital to the basis set than would be necessary. This function then is used as an angular momentum function [4]. The polarized basis set for the hydrogen atom carries p-functions, where typically only s-function were required. Again, these can be combined with the previously mentioned basis sets, like *polarized double-zeta* basis. Note that there is one unique behavior for d and f orbitals in polarized basis sets. For the d functions, sometimes 5 (called pure angular momentum functions) and sometimes 6 functions (called 6 Cartesian d functions) are obtained to describe the polarization (the sixth one looks like a 1s orbital and is not mandatory) [18]. In the f-orbital, it is similar, using 7, e.g., ten functions.
At last, there are *diffuse function,* which have a very small ζ exponent, which held the electron far from the actual nucleus. There are mandatory to describe anions or very electronegative atoms as fluorine or when calculating van der Waals complexes.
With all these mentioned basis sets, it is possible to describe more or less everything using DFT methods. For the individual problem, selecting a used basis set is relevant. In most DFT simulations, different basis sets can be applied for different particles in the system. Particles that are less relevant for the chemical problem get allocated smaller basis sets, and the more relevant particles (for example, for a chemical reaction) get allocated larger basis sets (for example, with diffuse functions). Using this method can drastically reduce the computation time and needed computational power and makes calculations for molecules in fluids only possible. So to reduce the number of mandatory functions to a functional level is a real problem in the applied DFT science.

1.2 Optimization using Machine Learning

In the first half of the last century, in 1959, Arthur Samuel, an IBM employee, was the first one who describe some algorithms as machine learning and coined the term [21]. At the same time, the DFT were invented. Since then, many things have changed, computers are way more powerful, and the methods used are way more advanced. As in this Thesis, Machine Learning (ML) optimization methods are used; this will be a short digression to the Machine Learning field. This thesis uses the Machine Learning for optimizations. Therefore, machine learning is just a tiny part of the huge field of Artificial Intelligence (AI), which is also shown in figure 1.5. The AI is everything where a computer program is used to understand or mimic human behavior.

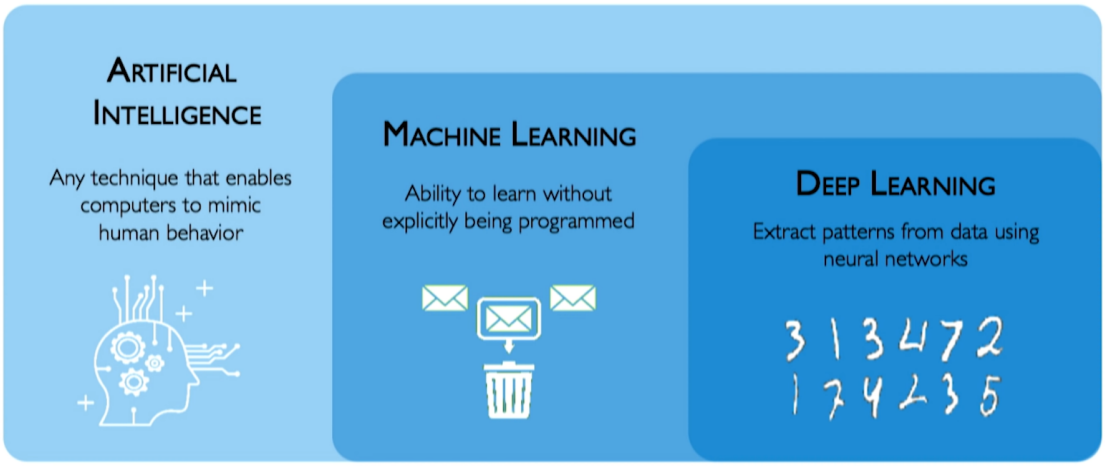


Figure 1.5: Overview of modern machine learning algorithms. Artificial intelligence is the most complex one and tries to mimic human behavior. Machine learning is the ability to select the best result. deep learning can be used to extract data for a machine earning algorithm.[22]

Some famous examples are AI programs that create customized advertising based on search results or customized video recommendations based on previously watched videos. Part of AI is the Machine Learning where the goal is that the program learns by itself without an explicit way. Machine learning gets very famous for all other nature science. Figure 1.6 shows the rapid increase of papers published in which machine learning has been used. In particular, at a certain point where the way to go is to improve some parameters manually, the ML can be very effective. In the same way, the ML is used in this thesis to improve some parameters with machine learning which can variate the parameters automatically, and not only that, the ML is also capable of making educated guesses for the parameter selection.
For the optimization using, Machine Learning it is important to of some function which will be optimized further on [22]. More precisely, to optimize the parameters (of the function) to minimize the loss relative to an exact result. In this thesis, the unbounded

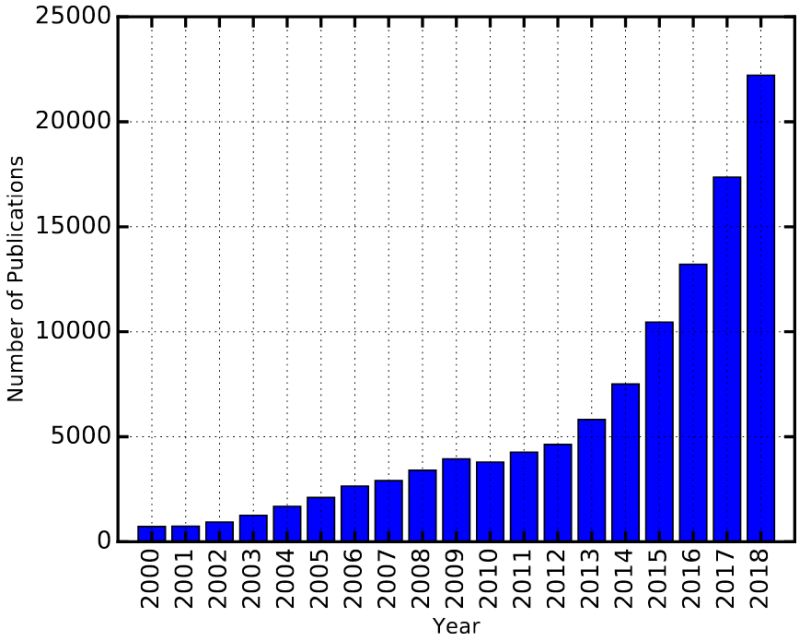


Figure 1.6: Number of ML-related publications as since 2000. [23]

minimization problem will be solved using **xitorch** [24]. xitorch is a fully differentiable machine learning framework based on the popular PyTorch framework [25].

$$\mathbf{y}^* = \arg\min_{\mathbf{y}} f(\mathbf{y}, \theta) \tag{1.66}$$

The goal is to find the best $\mathbf{y}$ to minimize the output of function f [24]. For the actual minimization, two different Optimizers can be used in xitorch. An Optimizer, in, ML is an algorithm or method to minimize the so-called loss function [26]. In machine learning, this function is usually called the loss function. The loss function can take a hundred multidimensional inputs so that a hyperspace can be created for two different input parameters, displayed in figure 1.7.

A loss function maps multiple parameters onto a real number, representing some "cost" concerning the initial parameters [27]. Besides the loss function, every optimizer needs a learning rate. The learning rate can be imagined as a step size in a specific direction. In the best case, this direction point toward the minimum[26]. It is essential to choose the learning rate wisely. Therefore, the learning rate is one of the last things in ML, which is mainly chosen manually. There are approaches to using a neuronal network to select the learning rate, but they are not used in this thesis. If the optimizer is on the right way to find the minimum, this is also called the optimizer does converge . If the optimizer is on the wrong way to find the minimum, this is called the optimizer does diverge . The converging optimizer is shown in the left and middle plot of figure 1.8 the diverging case is shown on the right plot. Figure 1.9 is displayed how The learning rate affects the loss function in dependency of the done iterations of the optimization.

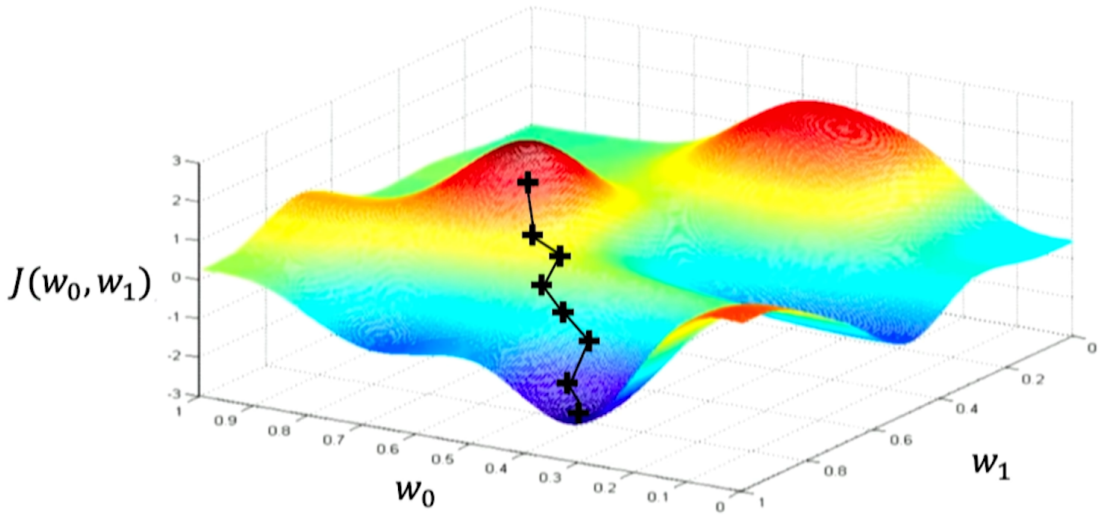


Figure 1.7: Principle of how gradient descent (gd) is working. The distance between the cross symbols equals the learning rate. While the surface spanned the multidimensional loss function in dependency of both input parameters, w_0 e.g. w_1[22]

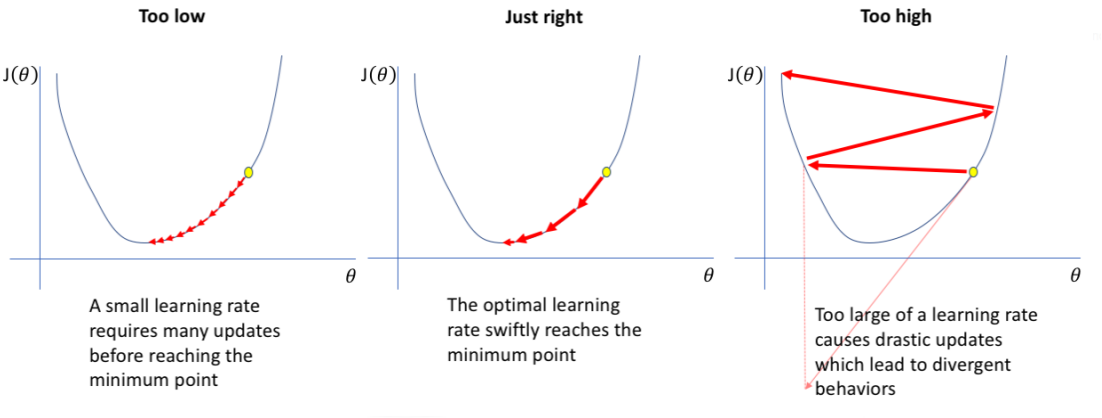


Figure 1.8: Selection of the learning rate, of a loss function $J(\theta)$.[26]

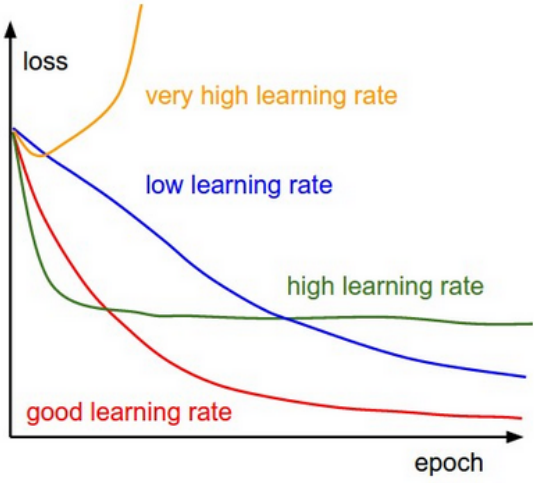


Figure 1.9: Loss function in dependency of done iterations (epoch) for different learning rates [28]

1.2.1 Gradient Descent

The first and simplest optimizer, is the gradient descent (gd) optimizer. gd is one of the most famous optimizers to optimize neuronal networks [29]. gd is simple to implement and very robust in terms of the input data. The major weakness is the weak performance per learning step. Compared to other Optimizers, the time to find a minimum is by far the longest. For the vanilla gd method, it will compute the gradient of a loss function concerning the parameters θ for all the training data:

$$\theta = \theta - \eta \dot{\nabla}_\theta J(\theta) \tag{1.67}$$

Here η is the learning rate. Note that the gradient for the whole dataset will be performed in just one update. In pseudocode, the method looks like this:

for i in range(epochs) **do**
    params_grad = evaluate_gradient(loss_function, data, params)
    params = params − learning_rate ∗ params_grad
end for

In xitorch the gd method enhances by adding a momentum to, gd which is shown in figure 1.10.

The gd algorithm has problems navigating in areas where the surface curves are much more steeply in one dimension than in the other one [29]. Momentum accelerates the algorithm by adding a fraction γ of the update vector in the previous time step to the current update vector

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{1.68}$$

$$\theta = \theta - v_t \tag{1.69}$$

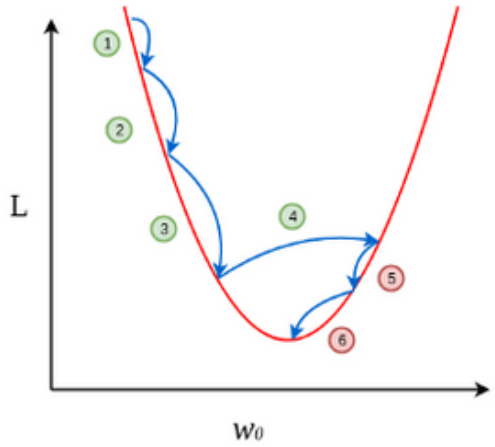


Figure 1.10: Schema of gradient descent using momentum, which leads to faster convergence. [30]

γ is usually set to 0.9. In figure 1.7 the way how the gd optimizer works on a hyperspace is shown.

1.2.2 Adam

Another much newer algorithm is the adaptive moment estimation (Adam) algorithm, introduced in 2014 by Kingma and Lei Ba [31]. The pseudocode for adaptive moment estimation (Adam) is shown in figure 1.11. Adam can interpret as a combination of RMSprop, and stochastic gradient descent method [29]. RMSprop is another optimizer that is heavily used in ML but will not be discussed here for more information [32]. On a given time step (mostly simulated by the iterator), the gradient is calculated conventionally using gd. Note again that in the case of this thesis, the automatic differentiation (AD) is used. The algorithm gets updated through an exponential moving average of the gradient (m_t) as well as the squared gradient (v_t) (where $\beta_1, \beta_2 \in [0, 1)$). Note that the moments are biased towards zero. Consequently, it is necessary to correct the bias by calculate, $\hat{m}_t$ e.g., $\hat{v}_t$ [31]. In Adam, it is crucial to carefully choose the learning rate α, as the learning rate gets adjusted during the actual learning. [31]. The main advantage of Adam is the much higher convergent performance compared to other optimization methods, as shown in figure 1.12 [31].

The previous advantages combined make the Adam Optimizer one of the most famous optimizers in modern deep learning.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 1.11: Adam pseudocode for stochastic optimization. Good default values for machine learning problems are $\alpha = 0.001$, $\beta = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are done element-wise. β_1^t, β_2^t denotes β_1, β_2 to the power of t. m_0, v_o initialize the mandatory momentum vectors. **t** is the iteration number or time step.[31]

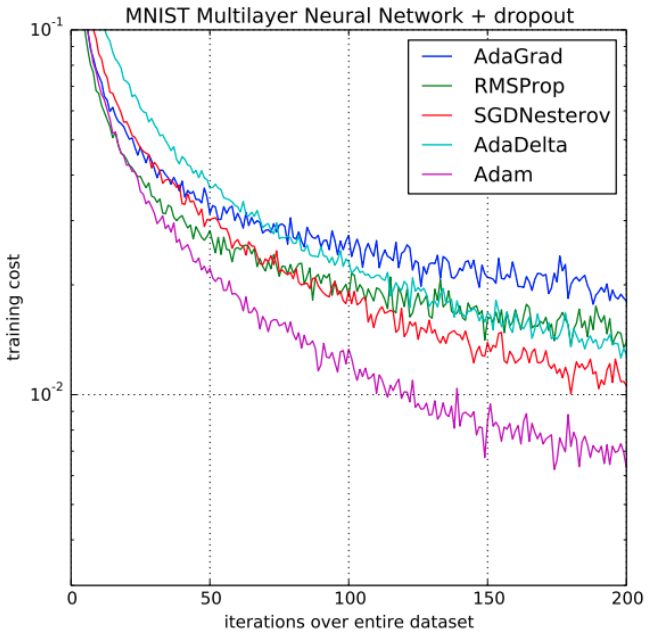


Figure 1.12: Performance of Adam, on multilayer neural networks on MNIST images using stochastic dropout regularization. Compared to other optimizers. AdaGrad, SGDNesterov, AdaDela are gd based methods. [31]

1.3 Automatic differentiation

In Machine Learning as well as in Density functional theory, it is crucial to use an efficient way to calculate the required gradients.

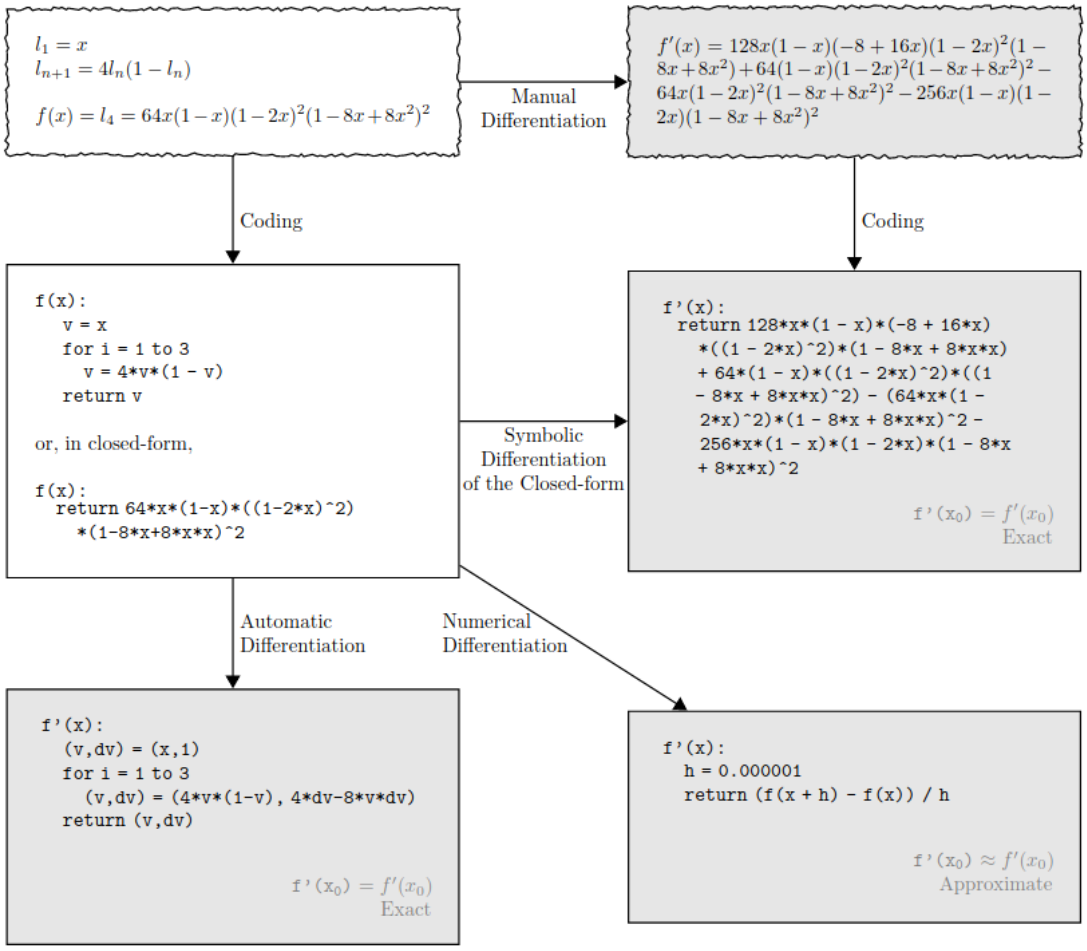


Figure 1.13: Difference and similarities between different derivation approaches [33]. Manual differentiation is fast and exact, but not useable for an arbitrary function. The automatic differentiation shows his major advantage. The results are exact and faster to compute than using symbolic differentiation. The numerical differentiation is fast, but does not yield an exact result.

In general, there are multiple methods known which can be used to calculate derivatives:

1. manually working out derivatives and coding them
2. *numerical differentiation* using finite difference approximations,
3. *symbolic differentiation* using expression manipulation in computer algebra (like Mathematica)
4. automatic differentiation (AD)

The significant differences are shown in figure 1.13. Nevertheless, every method has its advantages and disadvantages and specific use cases. The worst one is obvious method 1. as calculation by hand is notorious for human error. Also, it is incapable of numerous functions or complex functions. But if used correctly, it can bring up fast and accurate results. Method 2. the numerical differentiation is easy to implement, but highly inaccurate in higher degrees of derivations (for more details [34]). However, numerical differentiation has its advantages; it is straightforward to compute and fast. One example where the numeric differentiation is used is by analyzing discrete time series, like in this Thesis [35]. Method 3. the symbolic differentiation combines methods one and two but mostly results in very complex and cryptic expressions. Which can lead to an expression swell (to contain an exact result, the number of needed calculation steps raises drastically during the calculation). Note that there are also easy ways to handle the expression swell in symbolic differentiation [36], which makes symbolic differentiation also a powerful tool for science, which also shows the heavy use of Mathematica (which uses symbolic differentiation).
The Ansatz used in this Thesis is the automatic differentiation (AD). The automatic differentiation is also known for quite a while but was not used in machine learning [33]. Nowadays, there are multiple frameworks as **PyTorch** [25], **xitorch** [24] (which uses PyTorch) or **Jax** [37] that are capable of handling automatic differentiation. The idea of AD is similar to symbolic differentiation (but not equal!). The function

$$f_{example}(x_1, x_2) = ln(x1) + x_1\, x_2 - sin(x_2) \tag{1.70}$$

will serve as an example [33]. By applying the chain rule, it is possible to split up the function $f(x_1, x_2)$ into multiple functions whose derivations are already well known and implemented in every modern math code (like python math or NumPy). For example $\frac{d\sin(x_2)}{dx_2} = \cos x_2$. Using this technique for each chain function yields a correct analytical result for the derivation.
More general, constructing a function $f : \mathbb{R}^n \to \mathbb{R}^m$ using so-called intermediate variables v_i such that:

- variables $v_{i-n} = x_i$, $i = 1, \ldots, n$ are the input variables
- variables v_i , $i = 1, \ldots, l$ are the working (intermediate) variables

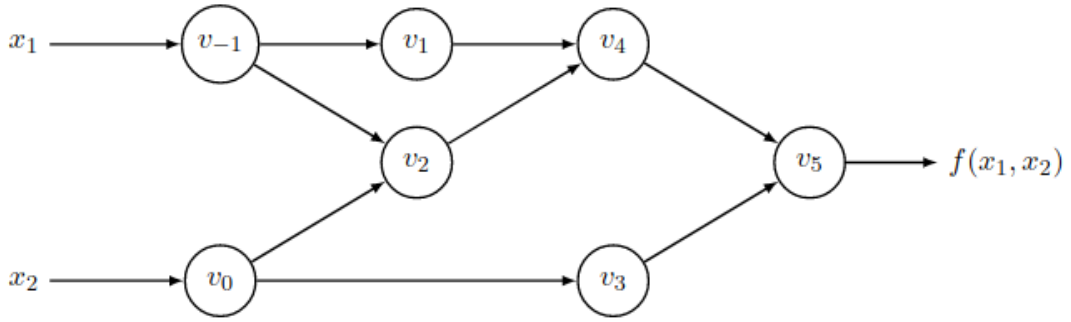


Figure 1.14: Graph of the example $f(x_1, x_2)$. the associated primal traces are shown in 1.15 e.g. 1.16 [33].

- variables $y_{m-i} = v_{l-i}$, $i = m-1, \ldots, 0$ are output variables

The trace of the elementary operations shown in 1.14 [33].

It should be noted that AD uses various algorithms to control the expression which comes out [33]. The symbolic differentiation can perform such algorithms in modern software [36] as well. AD can be split up into two different cases actually to calculate the derivative.

1.3.1 Forward Mode

Forward Primal Trace			Forward Tangent (Derivative) Trace		
v_{-1}	$= x_1$	$= 2$	$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
v_0	$= x_2$	$= 5$	$\dot{v}_0$	$= \dot{x}_2$	$= 0$
v_1	$= \ln v_{-1}$	$= \ln 2$	$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$	$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
v_3	$= \sin v_0$	$= \sin 5$	$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$	$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$	$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
y	$= v_5$	$= 11.652$	$\dot{y}$	$= \dot{v}_5$	$= 5.5$

Figure 1.15: Trace example of forward AD from function $f(x_1, x_2)$ (eq. 1.70) at point $(x_1, x_2) = (2\,,\,5)$ and setting $\dot{x}_1 = 1$ to compute $\frac{\partial f(x_1,x_2)}{\partial x_1}$. The primal forward evaluation is done on the left side. On the right side, the derivative trace is shown.

The concept of the forward method is more straightforward and is shown in figure 1.15. To compute f' within respect to x_1, it is necessary to associate x_1 with each v_i which the chain rule leads in:

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1} \tag{1.71}$$

Applying the chain rule to each operation lead to 5 more operations to finally calculate the derivative through $\dot{v}_5 = \frac{\partial f(x_1,x_2)}{\partial x_1}$. This generalizes to the Jacobian of a function $f : \mathbb{R}^n \to \mathbb{R}^m$. In this case, $\dot{x}_i = 1$ and zero else and represents the i-th unit vector. If $\mathbf{x} = \mathbf{a}$ is a specific input and, y the function:

$$\dot{y}_j = \left.\frac{\partial y_i}{\partial x_1}\right|_{x=a} \tag{1.72}$$

returns one column of the Jacobian matrix. The full Jacobian for n evaluations, given by:

$$\mathbf{J}_f \, \mathbf{r} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} \quad , where \, \dot{\mathbf{x}} = \mathbf{r} \tag{1.73}$$

Furthermore, it can be calculated by just one forward pass.
If $f : \mathbb{R}^n \to \mathbb{R}$, the result can be obtained directly by the partial derivatives.

$$\nabla f \cdot \mathbf{r} \tag{1.74}$$

Note that the forward AD works good for $f : \mathbb{R} \to \mathbb{R}^m$ but it is not the preferred method if $f : \mathbb{R}^n \to \mathbb{R}^m$ $(n \gg m)$ [33]. The AD can generalized to dual numbers [33].

1.3.2 Reverse Mode

Forward Primal Trace			Reverse Adjoint (Derivative) Trace			
$v_{-1} = x_1$		$= 2$	$\bar{x}_1 = \bar{v}_{-1}$			$\mathbf{= 5.5}$
$v_0 = x_2$		$= 5$	$\bar{x}_2 = \bar{v}_0$			$\mathbf{= 1.716}$
$v_1 = \ln v_{-1}$		$= \ln 2$	$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$	
$v_2 = v_{-1} \times v_0$		$= 2 \times 5$	$\bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$	
			$\bar{v}_{-1} = \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$	
$v_3 = \sin v_0$		$= \sin 5$	$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$	
$v_4 = v_1 + v_2$		$= 0.693 + 10$	$\bar{v}_2 = \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$	
			$\bar{v}_1 = \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$	
$v_5 = v_4 - v_3$		$= 10.693 + 0.959$	$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$	
			$\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$	
$y = v_5$		$= 11.652$	$\bar{v}_5 = \bar{y}$		$= 1$	

Figure 1.16: Trace example of backward AD from function $f(x_1, x_2)$ (eq. 1.70) at point $(x_1, x_2) = (2\,,\,5)$ But the point on the right are calculated in reverse. Both $\frac{\partial y}{\partial x_1}$ and $\frac{\partial y}{\partial x_2}$ are calculated equal $\overline{v}_5 = \overline{y} = \frac{\partial y}{\partial y} = 1$. Again, the primal forward trace is shown on the left side, and the derivative trace is done on the left. [33]

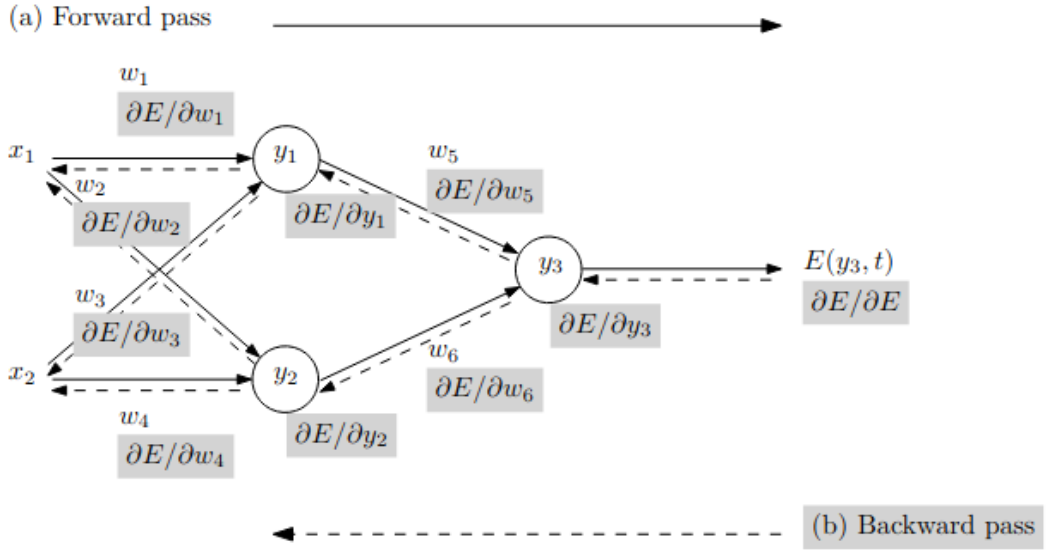


Figure 1.17: Back propagation overview: (a)inputs x_i generating activations y_i. An error E is calculated concerning the output y_3. (b) the error is adjoint backward, giving the gradient weights $\nabla_{\omega_i} E = (\frac{\partial E}{\partial \omega_1}, \ldots, \frac{\partial E}{\partial \omega_6})$.[33]

If $f : \mathbb{R}^n \to \mathbb{R}^m$ $(n \gg m)$ the backpropagation as a valid method, steps in. Especially this behavior is the standard problem of ML where there are many input parameters, but a result is a real number. The backpropagation is done by complementing each v_i with an adjoint:

$$\overline{v}_i = \frac{\partial y_i}{\partial v_i} \tag{1.75}$$

$\overline{v}_i$ represents the sensitivity of y_i concerning v_i, which is displayed as E in fig. 1.17. In reverse, AD can split up into two phases:

1. forward calculations of the original function to calculate v_i
2. backward adjoins the $\overline{v}_i$ form the outputs to the inputs.

Using example eq. 1.70 again to calculate the values given in tabular 1.16. The task now is to calculate the contribution $\overline{v}_i = \frac{\partial y}{\partial v_i}$. As shown in figure 1.14 of the variable v_0 affect the output y throw, v_2 e.g., v_3.

$$\frac{\partial y}{\partial v_0} = \frac{\partial y}{\partial v_2}\frac{\partial v_2}{\partial v_0} + \frac{\partial y}{\partial v_3}\frac{\partial v_3}{\partial v_0} \quad \text{or} \quad \overline{v}_0 = \overline{v}_2\frac{\partial v_2}{\partial v_0} + \overline{v}_3\frac{\partial v_3}{\partial v_0} \tag{1.76}$$

The main advantage of backward AD is the significantly less calculation effort for significant inputs. Particular in the case of $f : \mathbb{R}^n \to \mathbb{R}$ as there is just one application of reverse mode is made to calculate $\nabla f(x_1, \ldots, x_n)$ instead of n in forward mode. Again,

especially in machine learning, this can be heavily used. The outputs are a scalar loss function calculated by many input parameters. In general, for an arbitrary $f : \mathbb{R}^n \to \mathbb{R}^m$ backwards performs better for $m \ll n$.

$$\mathbf{J}_f^T \mathbf{r} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} r_1 \\ \vdots \\ r_m \end{bmatrix} \quad , where\, \overline{\mathbf{y}} = \mathbf{r} \tag{1.77}$$

The disadvantages of backward AD are the increased storage requirement [33]. However, reverse automatic differentiation is still a promising theory regarding speed and accuracy. Furthermore, this also shows the evolution of machine learning since every larger popular machine learning framework is on the way to properly integrate the AD as a standard differentiation method.

2 Optimize Basis Set

This chapter describes the actual process of optimizing a particular basis set. Therefore, the previously explained foundations will be combined, and new definitions will be made. In order to optimize a basis set, a second basis set is necessary. The first or initial basis set is the initial guess for the fully differentiable machine learning framework. The second basis serves as a reference basis set. Both basis sets will be labeled as "first" and "second", respectively, "initial" and "reference," or just "small" and "large" basis sets. The combination of two basis sets will be called "basis set variation" or "basis set combination". The basis set will be shortened to "basis" if it is evident that the basis set is meant out of context. Therefore, the initial basis set must be smaller than the reference. As the initial basis set has fewer parameters as input, the resulting ground state energy will always be higher than the energy calculated throw the reference basis set, which consequently always has more parameters. Besides the basis sets themselves also, the chosen systems are essential. In the case of this Thesis, these systems will be molecules. Every basis set variation is optimized concerning the selected molecule. This might sound surprising, as the regular basis set only depends on an Atom. However, in large chemical structures, different smaller molecule structures are included. Therefore, a basis set optimized for these smaller structures can be used. By utilizing the approach, the calculation time and mandatory calculation power can be reduced. So it makes sense to optimize a basis set to a particular small molecule. It is necessary to define a projection between two basis sets and a loss function for the ML algorithm.

2.1 Projection between two basis sets

The projection between both Basis Sets must be defined to optimize the initial basis. By projecting the basis sets onto each other, the atomic orbitals will also be projected as the basis set defines the atomic orbitals. According to, [38, 39, 40, 41] the projection can be defined as:

$$|\tilde{\Psi}_n\rangle = \hat{P}\,|\Psi_n\rangle \tag{2.1}$$

Where the projection for the initial basis is defined as:

$$\hat{P} = \sum_{\mu\nu} |\phi_\mu\rangle\, S^{-1}_{\mu\,\nu}\, \langle\phi_\nu| = \sum_{\mu\nu} |\phi_\mu\rangle\, \langle\phi_\mu|\phi_\nu\rangle^{-1}\, \langle\phi_\nu| \tag{2.2}$$

Ψ_n is the wave function of the n-th electron ins the system 1.1.6:

$$\langle\Psi_n| = \sum_A c^*_{A\,n}\, \langle\phi_A| \quad \text{and} \quad |\Psi_n\rangle = \sum_B c_{B\,n}\, |\phi_B\rangle \tag{2.3}$$

$\langle\phi_A|$ and $|\phi_B\rangle$ are the A'th and B'th GTO function of the large reference basis. C is the coefficient Matrix. Now using the inner product, since it is invariant under basis change:

$$\langle\Psi_n|\tilde{\Psi}_m\rangle = \langle\Psi_n|\,\hat{P}\,|\Psi_m\rangle \tag{2.4}$$
$$= \langle\Psi_n|\sum_{\mu\nu}|\phi_\mu\rangle\, S^{-1}_{\mu\,\nu}\,\langle\phi_\nu|\Psi_m\rangle \tag{2.5}$$
$$= \sum_A\sum_B\sum_{\mu\nu} c_{A\,n}\,\langle\phi_A|\phi_\mu\rangle\, S^{-1}_{\mu\,\nu}\,\langle\phi_\nu|\phi_B\rangle\, c_{B\,m} \tag{2.6}$$
$$= \sum_A\sum_B\sum_{\mu\nu} c_{A\,n}\, S_{A\mu}\, S^{-1}_{\mu\,\nu}\, S_{\nu B}\, c_{B\,m} \tag{2.7}$$

Now define this inner product for the whole system of molecules and combine the Therms which either correspond to the small, e.g., the more extensive reference basis:

$$\langle\Psi|\,\hat{P}\,|\Psi\rangle = C_1^T\, S_{1\,2}\, S^{-1}_{2\,2}\, S_{2\,1}\, C_1 \tag{2.8}$$

Equation 2.8 builds up the dissimilarity between two basis sets. The projection is a big part of the loss function, which has to be specified for the Machine Learning algorithm. One simple and at the same time apparent reason for this is that the output of the projection is a matrix, not an actual number. Nevertheless, more things can be done to enhance the loss function.

2.2 The loss functions

For this, Thesis, two different loss functions have been defined. Both are relatively similar and consist of the same core operations. Accordingly, the projection defined in equation 2.8 is multiplied with the occupied orbital matrix (θ). Each element of the one-dimensional matrix presents the number of atoms in the orbital. The number of electrons in the particular system will emerge by summation over this matrix. Building the trace of the projection matrix and dividing it by the number of electrons, followed by the multiplication with -1, the loss function is normalized to -1.

$$f_{loss}^{non\,weighted} = -\left(\mathbf{Tr}(\, C_1^T S_{1\,2}\, S^{-1}_{2\,2}\, S_{2\,1}\, C_1\,)/\sum_i^N \theta_i\right) \tag{2.9}$$

In forecast of the following loss function, equation 2.9 is called **non weighed**, because the second loss function used in this thesis is weighed by the molecular orbital energies ϵ. ϵ is a one-dimensional matrix containing every orbital's calculated ground state energy. So equation 2.9 becomes:

$$f_{loss}^{weighted} = \left(\mathbf{Tr}(\, C_1^T S_{1\,2}\, S^{-1}_{2\,2}\, S_{2\,1}\, C_1\,)/\sum_i^N \theta_i\,\epsilon_i\right) \tag{2.10}$$

The multiplication with -1 drops out since the orbital energies are consistently negative. In the following, the used loss function will be marked as "non weighed" if equation 2.9 is used and "weighed" if equation 2.10 is used.

2.3 The optb module

The optb module was created to run the optimization of a particular basis set for a specific molecule[42]. Creating a fully functional python module is much simpler at a certain point. The optb module is not published on PyPI, but can be installed using pip by adding the prefix "git+" before the GitHub URL [43]. At this point, only the abilities and limitations of the optb module will be discussed. An actual optimization run will be explained in the next section. The using the pip installer, the module's requirements will also be installed in their correct version. The requirements are:

- **Linux** as operating system, which is mandatory for PySCF
- **Python** version 3.9
- **PySCF** [44, 45] to do the ab initio DFT calculations.
- **Differentiable Quantum Chemistry (DQC)** the Differentiable Quantum Chemistry module [46] to handle DFT related calculations inside the optimization, like calculating the overlap matrix for the optimized basis set.
- **basis set exchange (bse)** [3] to provide the basis set data.
- **Atomic Simulation Environment (ASE)** the Atomic Simulation Environment [47] which contains the data for the **G2** database [17].
- **xitorch** a differentiable scientific computing library [24] to run the actual optimization.
- **PyTorch** [25] to run xitorch.
- **Tensor Board** [48] a graphical interface to visualize the machine learning process created by TensorFlow and then merged in PyTorch.
- **NumPy** [49], **SciPy** [50], **pandas** [51],**plotly** [52], **Matplotlib** [53] to do simple math task outside the optimization as well as plotting and analyze the outcome results.

As shown in the upper listing, PySCF and DQC handle every DFT related work, while PyTorch Tensor Board and xitorch do all the Machine Learning related work. PySCF uses the B3LYP potential to run an ab initio Kohn-Sham DFT calculation (can be changed manually to every potential of Libxc [54]). The input for PySCF is the atomic structure and the basis set; the remains will be automatically handled by optb. The atomic structure can be contained by two built-in databases or as manual input. The two database are the **G2** database [17] and the **W4-17** [55] database. Both possess information about smaller molecules, like their charge, the total angular moment, and the atomic structure. Therefore, the atomic structure is the position of each atom inside the molecule in real space. Note that the G2 database is mainly used for test reasons since it is the most used database in physics. The W4-17 is done with contemporary

molecular dynamic simulations, which leads to more authentic atomic structures. The molecules which can be easily accessed are preselected. The avail molecules, which can be found in Appendix 4, are uncharged, and the multiplicity ($2S = 1$ where S is the total spin angular momentum) is equal to one. Deviating values would change the actual DFT algorithm and would otherwise have to be adjusted. The preselection retains 119 out of the 148 molecules in the G2 database and 149 out of 183 molecules in the W4-17 database.
As training data, every basis variation can be used, for which bse provides the unique basis sets. Only the previously discussed limitations have to be considered, like the size of both basis sets 2.1.
Optimizing the picked basis set is done using the xitorch minimizer, which offers Adam and gradient descent with momentum as the optimizer. It is also possible to control the optimization by setting parameters like the minimizer's learning rate or a maximal number of iterations (also called epochs). For this thesis, a divergence control has been added to xitorch, which lets the minimizer break up if the divergence is stable over a long period. Other break-up conditions can be specified; a minimal iteration number can be added, which blocks the check-up of the break-up prerequisites for the designated iteration number. In this manner, it is ensured that the minimizer can learn more freely at the beginning. Everything inside the minimizer has to be fully differentiable PyTorch tensors. Accordingly, DQC handles the operations inside the loss function as computing the overlap matrix in every iteration. The overlap matrix and, thus, the projection has to be computed since the initial basis changes in each learning epoch 2.1. During the optimization, the Tensor Board framework can plot the loss function and evaluate the optimization in real time. The optimizer's output is an optimized basis set, which is stored in a ".json" file and can be used for further DFT calculations from now on. Besides that, it is doable to output a misc file where all relevant information related to the Machine Learning (ML) process is stowed. Using python build-ins and Pandas, a folder structure can be created to handle the output, making sense if multiple optimization processes are running.
To sum up, what are the abilities and limitations of the optb module?
Initiating with the abilities, optimizing the majority of Gaussian basis sets for a specified molecule should be possible. Assuming that an initial basis set does exist for every atom in the molecule and the molecule itself does not carry a charge or multiplicity other than one. Two different optimizers to solve the minimization problem are open. Simultaneously, observing the learning progress and creating a folder structure to store large amounts of data is possible.
On the other hand, there exist a couple of restrictions. The most obvious limitation is that optimizing the basis set for a specific Atom independent of a molecule's presence is impossible. It is also not possible to optimize an arbitrary molecule. It is also impossible to use other modern optimizers besides the two implemented ones, like AdaGrad or RMSProp 1.2, and implementing a new optimizer is not easy. Further, no method was implemented that automatically searches for the correct learning rate. The following section shows how a basis set gets optimized for a specific molecule for the H_2 molecule.

2.4 Optimization example of a basis set for molecule

After illustrating the principles of the optb module, the essential use of the code itself will be explained briefly. The presented example looks quite similar to a Jupyter notebook, and indeed there is an executable example, which can be discovered in the optb GitHub [43].

Before using the code, a python 3.9 environment must be set up to run the opt module. If this were done, the optb module could be installed by using:
pip install git+https://github.com/Jaikinator/OptBasisSets
For this example, the STO-3G basis set should be optimized for the H_2 molecule. The cc-pvtz basis set, therefore, serves as a reference.
The first step is importing all relevant functions accessible by simply importing the OPTBASIS class. The molecules which can be optimized have to satisfy two conditions. The molecule's charge has to be zero, and the multiplicity has to be one. In addition, the two lists of avail molecules from the G2 and the W4-17 database will be added (a list of all molecules can be found in the Appendix 4).

```
In [1]:  "
         import relevant class object":
         from optb.optimize_basis import OPTBASIS
         "import all preselected available molecules:  "
         from optb.data.preselected_avdata import elw417 , elg2
```

The second step is to set up all parameters relevant to the ab initio DFT calculation.

```
In [2]:  molecule = "h2" # molecule name
         init_basis = "STO-3G" #initial basis set for the optimization
         ref_basis = "cc-pvtz" #reference basis set for the optimization
```

The H_2 molecule is part of the W4-17 database and can be called by its chemical name (the same molecule from the G2 database is referred to as" H2"). The basis sets can be called natively by their name since basis set exchange and PySCF and DQC can handle these types of inputs. Every basis set available on `https://www.basissetexchange.org` can be used for optimization. The basis sets take their name as input. The initial basis set has to be less precise than the reference one.
In addition to the DFT-related parameter, the machine learning-related parameter has to be defined. The only mandatory parameter, therefore, is the learning rate, which has no default value. The chosen loss function is the non-weighed loss function, as the values are normalized to a minimal value of -1. The maximal iteration number has been set to a usable amount for this example. The maximal iteration value must be multiple magnitudes higher in an actual optimization process to achieve decent results. The same applies to the optimizer. By default, Adam will be used.

```
In [3]:  lr = 2e-4 # selected learning rate

         weighed = False # using the non-weighed loss function
         maxiter = 50 # maximal number of iterations
         miniter = int(maxiter/10) # minimal number of iteration
         method  = "Adam" # optimizer method which will used
```

The next step is applying these parameters to the OPTBASIS class.

```
In [4]:  opt = OPTBASIS(init_basis, ref_basis, molecule, lr,
                        maxiter = maxiter, miniter = miniter,
                        method = method, weighed = weighed)
```

```
Out[4]:  OPTBASIS(basis=STO-3G, basis_ref=cc-pvtz, molecule=h2,
         step=0.0002, maxiter=50, miniter=5, method=Adam,
         weighed = False, diverge=-1.0, maxdivattempts=50,
         output_path=None, get_misc=True, mol_kwargs={},
         minimize_kwargs={}, out_kwargs={})
```

For exemplary purposes, the class object is equipped with a representor, which can ensure that everything is set up correctly. The representor also prints the other optional parameters and their default values, but these will not discuss here. Further information can be found in the documentation inside the code [43].

Now, the code is already ready to run the optimization.

```
In [5]:  opt.optimize_basis(save_out = False) # run the optimization
```

```
Out[5]:  overwrite output file: ./output/scf_cc-pvtz_h2.out
         overwrite output file: ./output/scf_STO-3G_h2.out

         start optimization of STO-3G Basis for the Molecule h2
         and learning-rate 0.0002

         The initial guess is:
         tensor([3.4253, 0.6239, 0.1689, 0.9817, 0.9495, 0.2959],
         dtype=torch.float64)

            #:              f |         dx,         df
            1: -9.928184e-01 | 4.899e-04, 9.928e-01
         minimal number of iterations is reached now
         beginning to check for divergence
           10: -9.934040e-01 | 4.889e-04, 6.222e-05
           20: -9.939863e-01 | 4.861e-04, 5.497e-05
           30: -9.944970e-01 | 4.820e-04, 4.793e-05
           40: -9.949399e-01 | 4.770e-04, 4.141e-05
           50: -9.953213e-01 | 4.716e-04, 3.556e-05
         output file: .//scf_optB_h2_014.out
         /nfs/data-013/jaikinator/anaconda3/envs/OptBasis/lib/python3.9
         /site-packages/xitorch/_impls/optimize/minimizer.py:359:
         UserWarning: The minimizer does not converge after 49
         iterations.
         Best |dx|=4.7160e-04, |df|=3.5562e-05, f=-9.9532e-01
           warnings.warn(msg)
```

```
Out[5]:  opt basis: {'H': [[0, [3.4154923778335884, 0.18382277260481714],
         [0.6337203965479148, 0.6161942841978492], [0.17803968905593073,
         0.4866653691497605]]]}

         initial energy: -0.9248530539549455
         ref energy: -0.963763850056754
         opt energy: -0.9328197564857001
```

Now the optimization is done. As expected, there is no convergence after just 50 iterations, which explains the warning. Three output files for the ab initio DFT calculations were created by PCF. These files contain all information about the DFT calculation itself.

1. scf_cc-pvtz_h2.out DFT calculation using the reference basis set
2. scf_STO-3G_h2.out DFT calculation using the initial basis set
3. scf_optB_h2_001.out DFT calculation using the optimized basis set

Before the run starts, all relevant pieces of information for the learning and the initial parameters are displayed again. Then the run starts and prints updates every ten iterations. After the 5.th iteration, the optimizer allows breaking up due to divergence, which can save time in a more extended optimization. Therefore, optb ensures that the divergence is not just a temporary phenomenon and continues over a long period of iterations. The displayed learning parameters are the value of the loss function. If the value of the loss function gets closer to minus one, this can be interpreted as an improvement. *df* describes the difference between the previous iteration's loss function value and the loss function's current value. $ df = f_i - f_{i-1}$ *dx*, on the other hand, is the difference in the norm of all input parameters of the previous and the current iteration. After all 50 iterations were done, and the optimized basis set values were printed. Here it is good to see that the number of parameters does not change during the optimization. Afterward, the difference in energies is displayed:

- The energy using the initial basis (STO-3G) ≈ -0.925 Hartree
- The energy using the optimized basis ≈ -0.934 Hartree
- The energy using the reference basis (cc-pvtz) ≈ -0.964 Hartree

This shows that already, after just 50 iterations, there is a significant improvement on the STO-3G basis set. Which now can be replicated for other molecules and other basis sets. In an actual optimization process, it will be helpful to use a much higher maximal iteration number (or the default of 10e6) and try different learning rates. Since the input parameters of the basis sets and the molecule were just passed to the optimizer by their names (as they are used in science), at this point, they will be displayed as the optimizer uses them internally:

```
In [6]:   opt.atomstruc # atomic structure of the H2 molecule

Out[6]:  list(['H', 0.0, 0.0, 0.37095], ['H', 0.0, 0.0, -0.37095])

In [7]:   opt.init_basis #values of the STO-3G basis set

Out[7]:  dict('H': [[0,
           [3.425250914, 0.1543289673],
           [0.6239137298, 0.5353281423],
           [0.168855404, 0.4446345422]]])

In [8]:   opt.ref_basis #values of the cc-pvtz basis set

Out[8]:  dict('H': [[0,
           [33.87, 0.0, 0.006068, 0.0],
           [5.095, 0.0, 0.045308, 0.0],
           [1.159, 0.0, 0.202822, 0.0],
           [0.3258, 1.0, 0.503903, 0.0],
           [0.1027, 0.0, 0.383421, 1.0]],
           [1, [1.407, 1.0, 0.0], [0.388, 0.0, 1.0]],
           [2, [1.057, 1.0]]])
```

Therefore, it is possible to input just the atomic structure of a molecule. The same behavior does not apply as quickly to the basis sets; however, it is possible to create an individual basis set. Therefore create an NWChem file and designate it inside the modules basis set folder. At this point, everyone with at least a bit of knowledge about basis sets and molecules should be able to optimize a basis set for a specific molecule. Therefore, we want to leave the part about the optb module and continue with the evaluation results, which optb can output for different molecules and different basis sets.

3 Evaluation

After finalizing the optb module, it was time to use it on the built-in molecules for optimizing some basis sets. The result of these optimizations will be discussed from now on. For this thesis, all avail molecules of the W4 -17 and the G2 database are optimized for various basis set variations. The G2 dataset was mainly used for testing, so most results lie on the W4-17 database. We are starting from the physical and chemical point of view, most significant part. How much does enhance an optimized basis set actual DFT calculation? In the second part, we want to assess the Machine Learning procedure. Which delivers the actual capabilities of the optb module in real applications.
In total, 26098 optimizations were done. All avail molecules (which again can be found in the Appendix 4) were trained for the following basis combinations:

initial basis	**reference basis**
STO-3G	3-21G
STO-3G	cc-pVTZ
3-21G	cc-pVTZ
cc-pVDZ	cc-pVTZ
cc-pVDZ	cc-pVQZ
cc-pCVDZ	cc-pCVTZ
cc-pVDZ	aug-pc-2
aug-cc-pVDZ	aug-pc-2

Table 3.1: Different basis variations for which the molecules have been trained.

These different basis sets were selected due to their different complexities. The STO-3G basis is part of the STO-nG basis set [2]. It is a minimal basis set where Slater-type-orbitals (STO) is approximated by three primitive Gaussian per orbital.
The 3-21G basis is the famous Pople basis [56], which is a split-valence double-zeta basis set.
The cc-pVNZ are correlation-consistent polarized basis sets done by Thom H. Dunning, Jr. [57]. The *V* stands for "valence only", and *NZ* is the number of Zeta functions in the basis set. The aug-cc-pVDZ are extended augmented Dunning basis sets.
The aug-pc-2 was invented by Frank Jensen in 2002 [58, 59, 60, 61]. It is an augmented polarization consistent with, in this case, two angular momentum functions.
The selected basis sets should ensure that the optb module can optimize simple basis sets like STO-3G and handle cutting-edge basis sets, which have additional functions like diffuse or polarisation functions. In addition, is it possible to remove the amount of the necessary ζ- functions for specific molecules or basis sets?

The optimization for each molecule and each basis set combinations were done with both methods (Adam and gradient descent). Since no automated learning rate search was implemented, various learning rates were chosen. The learning rates were 2×10^{-1}, 2×10^{-2}, 2×10^{-3}, 2×10^{-4}, 2×10^{-5}, 2×10^{-6}, 2×10^{-7}, 2×10^{-8}, 2×10^{-9}, 2×10^{-10}. As a maximal number of iteration for the optimization, 10^6 were used. The maximal number of iterations was chosen empirically, as most optimizations were almost converged or diverged at this point. For all runs, a minimal number of iterations were set to 10^4 Of course, not all runs lead to a well-optimized basis set; therefore, the results must be selected.

3.1 Select Optimized basis sets

In ML, the waste majority of the produced data is usually not serviceable. Figure 3.1 shows a small part of the raw output data to illustrate how an actual output looks. Since miscellany learning rates were used, most training will diverge, and merely a few appropriate learning rates will obtain good results. Accordingly, all results were dropped where the best iteration is zeroth (or in the single-digits) iteration. Here the optimization diverges from the origin. Occasionally the learning does converge, but to an unreasonable result for the energy of the optimized basis set, which can happen if the optimizer diverges over a long period but finds a local minimum close to the current diverged loss function value. The outcoming basis sets were dropped if the optimized energy was not in the initial and reference energy range. Mention that this behavior was occasional and only discovered for distinct molecules. But not only the optimization can fail to converge, PySCF [44] can experience the same problem during the DFT calculation. Especially for larger basis sets, this can be an issue since the calculated results are over-determined and linear depend. All the previous problems were solved similarly for the results by dropping every optimized basis set if something went wrong, since we know why a particular behavior has appeared. While the data is already proceeding, the best basis sets for every molecule and reference were stored.

molecule	file number	initial_energy	ref_energy	opt_energy	learning rate	maxiter	miniter	method	best_f	best_df	best_dxnorm	best_i	max_i	f_rtol	basis	ref. basis
1h2	1	-0.92485305395	-0.944415477	-0.924853054	0.002	1000000	10000	adam	-0.51349	0.5134928	0.00489897942	0	774	0	STO-3G	3-21G
2h2	6	-0.92485305395	-0.944415477	-0.924853054	2E-08	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-08	0	24601	0	STO-3G	3-21G
0h2	3	-0.92485305395	-0.944415477	-0.924853054	2E-05	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-05	0	34601	0	STO-3G	3-21G
1h2	8	-0.92485305395	-0.944415477	-0.924853054	2E-10	1000000	10000	adam	-0.51349	0.5134928	4.8989798E-10	0	24601	0	STO-3G	3-21G
2h2	4	-0.92485305395	-0.944415477	-0.924853054	2E-06	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-06	0	24601	0	STO-3G	3-21G
3h2	7	-0.92485305395	-0.944415477	-0.924853054	2E-09	1000000	10000	adam	-0.51349	0.5134928	4.8989793E-09	0	24601	0	STO-3G	3-21G
4h2	10	-0.92485305395	-0.944415477	-0.924853054	2E-12	1000000	10000	adam	-0.51349	0.5134928	4.8990071E-12	0	24601	0	STO-3G	3-21G
5h2	9	-0.92485305395	-0.944415477	-0.924853054	2E-11	1000000	10000	adam	-0.51349	0.5134928	4.8989799E-11	0	24601	0	STO-3G	3-21G
6h2	5	-0.92485305395	-0.944415477	-0.924853054	2E-07	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-07	0	24601	0	STO-3G	3-21G
7h2	2	-0.94441547653	-0.96376385	-0.9444154765	0.0002	1000000	10000	adam	-0.52278	0.5227814	0.00044721359	0	999999	0	3-21G	cc-pvtz
8h2	1	-0.94441547653	-0.96376385	-0.9444154765	0.002	1000000	10000	adam	-0.52278	0.5227814	0.00447213595	0	999999	0	3-21G	cc-pvtz
9h2	6	-0.94441547653	-0.96376385	-0.9444154765	2E-08	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-08	0	24601	0	3-21G	cc-pvtz
0h2	3	-0.94441547653	-0.96376385	-0.9444154765	2E-05	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-05	0	24601	0	3-21G	cc-pvtz
1h2	8	-0.94441547653	-0.96376385	-0.9444154765	2E-10	1000000	10000	adam	-0.52278	0.5227814	4.4721362E-10	0	24601	0	3-21G	cc-pvtz
2h2	4	-0.94441547653	-0.96376385	-0.9444154765	2E-06	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-06	0	24601	0	3-21G	cc-pvtz
3h2	7	-0.94441547653	-0.96376385	-0.9444154765	2E-09	1000000	10000	adam	-0.52278	0.5227814	4.472136E-09	0	24601	0	3-21G	cc-pvtz
4h2	10	-0.94441547653	-0.96376385	-0.9444154765	2E-12	1000000	10000	adam	-0.52278	0.5227814	4.4721612E-12	0	24601	0	3-21G	cc-pvtz
0h2	9	-0.94441547653	-0.96376385	-0.9444154765	2E-11	1000000	10000	adam	-0.52278	0.5227814	4.4721363E-11	0	24601	0	3-21G	cc-pvtz
1h2	5	-0.94441547653	-0.96376385	-0.9444154765	2E-07	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-07	0	24601	0	3-21G	cc-pvtz
2h2	2	-0.93199724014	-0.96376385	-0.959738276	0.0002	1000000	10000	adam	-0.52337	2.293E-10	0.00038187637	2114	24601	0	cc-pvdz	cc-pvtz
3h2	1	-0.93199724014	-0.96376385	-0.9593918135	0.002	1000000	10000	adam	-0.5232	1.737E-07	0.00398756636	89	999999	0	cc-pvdz	cc-pvtz
4h2	6	-0.93199724014	-0.96376385	-0.9319972401	2E-08	1000000	10000	adam	-0.521	0.5210002	6.3245553E-08	0	24601	0	cc-pvdz	cc-pvtz
5h2	3	-0.93199724014	-0.96376385	-0.9606459823	2E-05	1000000	10000	adam	-0.52348	1.09E-10	5.7595008E-05	9600	34601	0	cc-pvdz	cc-pvtz
6h2	8	-0.93199724014	-0.96376385	-0.9319972401	2E-10	1000000	10000	adam	-0.521	0.5210002	6.3245558E-10	0	24601	0	cc-pvdz	cc-pvtz
7h2	4	-0.93199724014	-0.96376385	-0.9590759587	2E-06	1000000	10000	adam	-0.5235	7.242E-13	6.3721342E-06	74638	219601	0	cc-pvdz	cc-pvtz
0h2	7	-0.93199724014	-0.96376385	-0.9319972401	2E-09	1000000	10000	adam	-0.521	0.5210002	6.3245554E-09	0	24601	0	cc-pvdz	cc-pvtz
1h2	10	-0.93199724014	-0.96376385	-0.9319972401	2E-12	1000000	10000	adam	-0.521	0.5210002	6.3245646E-12	0	24601	0	cc-pvdz	cc-pvtz
2h2	9	-0.93199724014	-0.96376385	-0.9319972401	2E-11	1000000	10000	adam	-0.521	0.5210002	6.3245558E-11	0	24601	0	cc-pvdz	cc-pvtz
3h2	5	-0.93199724014	-0.96376385	-0.9319972401	2E-07	1000000	10000	adam	-0.521	0.5210002	6.3245553E-07	0	24601	0	cc-pvdz	cc-pvtz

Figure 3.1: Example for the raw output data out. The first two columns are the molecule's name and the file number in the folder structure. Then the initial energy, the reference energy (textitref_energy), as well as the energy using the optimized basis *opt_energy*, are listed. After this, the ML related parameters are listed. The learning rate column is self-speaking. On the right, the *maxiter* column gives the number of maximal possible iterations possible in the training run. *miniter*, on the other hand, the minimal iterations a training has to do before checking on set termination conditions. The method is the optimizer, which was used for the training. The columns *best_f, best_df, best_dxnorm* are the best value of the loss function. In this case, the non-weighed loss function. *df* is the difference between the current loss function value and the value in the previous epoch. The *dxnorm* is the difference between the norms of the current parameter values and the norm of the parameter values in the previous epoch. *best_i, max_i* are the maximum number of iterations done during the training, and best_i marks the epoch on which the best minima were found. *f_rtol* is a termination condition based on the output's relative tolerance. *basis* and *ref_basis* are the names of the initial, the reference basis set. A larger version of this picture can be found in the Appendix 4, 4.3

3.2 About the successful optimizations

The next step is to make the data better comparable between all outcomes. Since every molecule has a different number of Atoms in the system, the total energies are not comparable between the molecules. Therefore, the ground state energy is displayed in Hartree per Atom. The energy dependencies due to the number of atoms in the molecule were removed this way. In addition, the absolute difference between the reference energy calculated with the reference basis set and the energy using the initial basis set was determined in Hartree per Atom. In addition, the absolute difference between the reference and the energy using the optimized basis set was computed. The evaluation will cover both loss functions, since optb [43] has implemented two.

3.3 Non-weighted loss function optimization

We are beginning with the non-weighed loss function. As a reminder, this means the minimum value of the loss function can never be smaller than -1. However, in the ideal case, it would be -1 as the smaller basis leads to the exact same energy as the larger basis. The calculations were done without termination conditions and with a specific termination condition besides the divergence control. The termination condition was the relative error of the projection function between two iterations. It was set to 10^{-16}. Unfortunately, xitorch used other termination conditions that bring the learning to break up by a total error of 10^{-8}, which was removed in later calculations without any termination conditions.

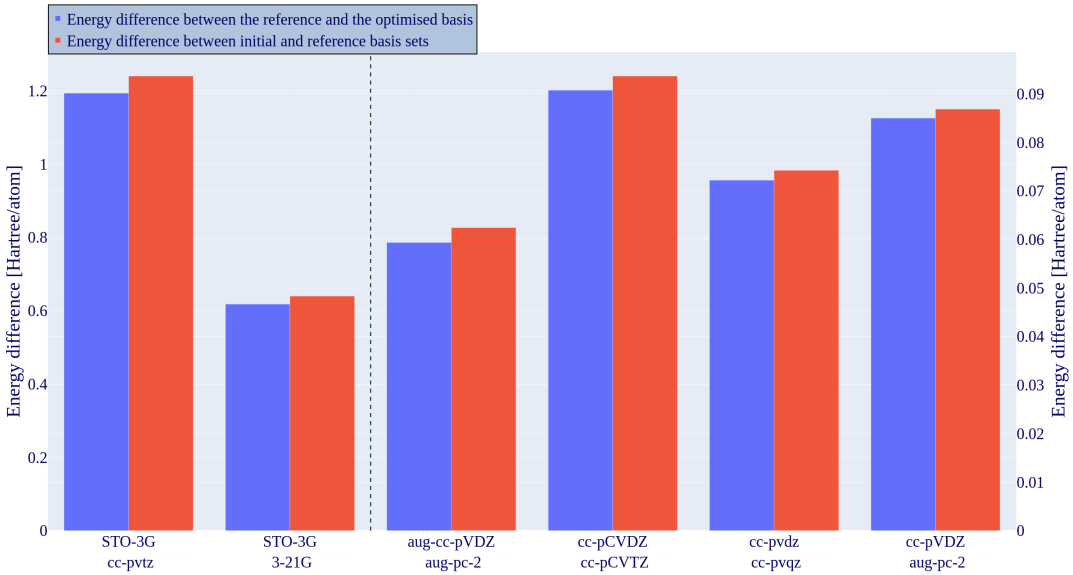


Figure 3.2: In blue, the energy difference between the energy using the reference (larger) basis set and the energy calculated throw the optimized basis. In red, the energy difference between the energy using the reference (larger) basis set and the energy using the initial (smaller) basis set. $\Delta E = E_{ref} - E_{opt}$ The difference between the red and the blue bar for the given basis variation on the x-Axis displays the improvement done by the learning.

In figure 3.2, the energy differences regarding the reference energy are displayed for an average over all molecules and for different basis variations. Figure 3.3 shows energy difference concerning the initial basis set of both optimized and reference basis sets. An important fact that can be extracted from both figures is that there is undoubtedly an improvement in the ground state energy by using the optimized basis set. The gain depends on the chosen basis variations. While the highest energy improvements were achieved using the STO-3G basis set, it does not mean that this is the best optimization.

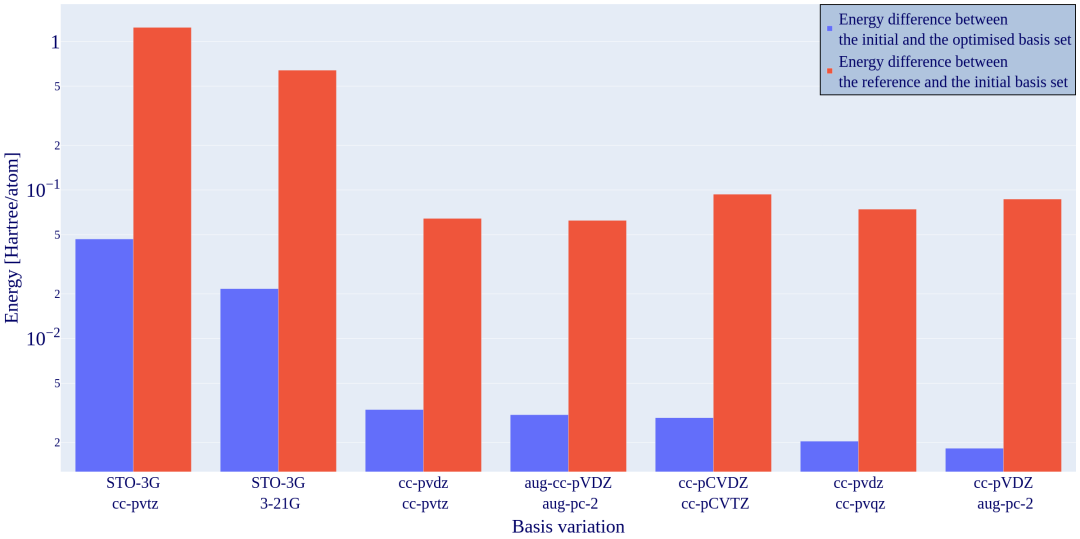


Figure 3.3: Average Energy difference between the energy using the optimized basis set and the initial energy in Hartree per Atom, in blue. $\Delta E = E_{opt} - E_{init}$ In red is the energy difference between the reference and the initial energy. The y-axis is displayed logarithmically.

The STO-3G basis is a small basis set, so the initial energy difference regarding the reference is quite significant from the beginning. Enhance this initial energy discrepancy will be by its total value relatively high. However, the relative improvement related to the initial energy difference is not as high for the STO-3G basis as other basis set variations. The relative improvement is defined as:

$$\Delta E[\%] = \frac{100}{E_{ref} - E_{init}}(E_{optb} - E_{init}) \tag{3.1}$$

Nevertheless, the proposed STO-3G basis set is far superior to the other variations regarding the overall energy improvement. While at the same time, the total energy difference between the initial basis set and the reference basis set is much higher. If the energy difference between the initial and the reference basis is already relatively small, then it is questionable whether it is worth optimizing. In this case, these results can be considered proof of concept. This proof concludes that it is possible to advance large basis sets further. It should be reiterated that the calculated energy differences in Figures 3.3 and 3.2 are based on the average over more than a hundred different molecules. Each molecule has different chemical behaviors and different atoms in it. Remember that the basis set functions depend on individual atoms, even if the basis set type is the same. Another thing to remark is that heavier atoms have more energy per atom than lighter ones. The displayed energy difference for the average overall molecules is consequently shifted towards the molecules that consist of heavier atoms. A heavier atom's basis set consists of more functions than the basis set of a lighter atom. Therefore, they are more

challenging to optimize (see section 1.1.7). This being said, it makes sense to display the optimization success in dependence of the molecule or, more precisely, the number of atoms inside the molecule.

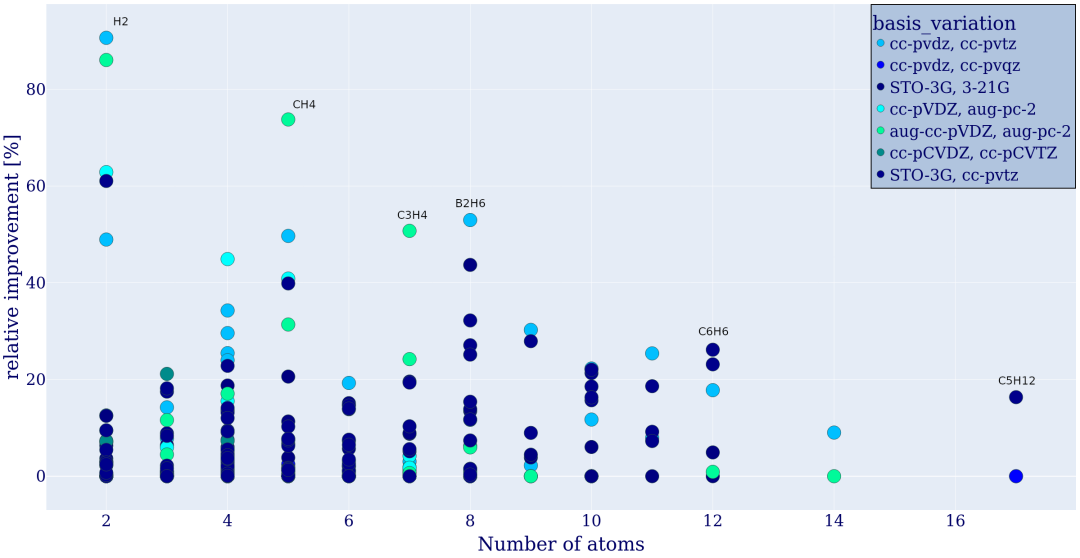


Figure 3.4: Relative improvement of an optimization $\frac{100}{E_{ref.-init}} * E_{optb-init}$ depends on the number of Atoms in a certain molecule. Displayed for different basis variations. A more detailed version of this figure can be found in the Appendix 4.1.

Accordingly, figure 3.4 shows the relative improvement for the best result (regarding all optimizations with atom structure and basis set variation) in dependency of the number of Atoms in the molecule. To mention again, the number of atoms in a molecule only indirectly relates to the number of electrons. The number of electrons inside an Atom is not considered in this plot. The H_2 molecule has the best relative improvement, which can be found in the upper left of the plot. However, the optimization was also going well for some molecules with more atoms. Unlikely, the overall trend shows that the optimization success is lower for larger molecules, which was anticipated since larger molecules are usually more complex than smaller molecules. We want to go into more detail here, utilizing the principles of how basis sets are constructed. A larger basis set can represent the same physical property as a smaller one by increasing the number of the ζ functions, 1.1.7 for every molecular orbital. If this is the circumstance, the task of the optimizer is to mimic the contacted Gaussian of a higher degree using fewer functions, which can be relatively straightforward or highly complicated. Imagine utilizing one function instead of three functions to describe an s-orbital; this will be relatively easy. However, describing a p-orbital with one instead of three functions is nearly impossible. The consequence is that the optimization gets significantly more complex with every additional orbital. The more extensive basis sets do not only add extra ζ-functions to

their collection. Furthermore, they add additional functions to include more physical effects. Examples of these functions are polarization or diffuse functions. Both functions are localized in an additional orbital. Now the optimizer's task is to use fewer functions (from the small basis) to imitate more functions for the same orbital and to describe additional orbitals (can be one or more) without changing the number of parameters given by the initial basis set. For example, complicated molecules that profit from polarization functions are significantly more complex to optimize than molecules that do not need these functions. The optimizer indirectly gets the information over the actual number of molecular orbitals from the occupied orbital matrix, which is part of the loss function and an output of the DFT calculation. All functions of the basis set will still be optimized, but for smaller molecules, most functions are just not as relevant as for larger systems and, therefore, easier to optimize. In figure 3.4 one molecule stands out at the five atoms mark this molecule is CH_4. A plot with a hoverable plot of figure 3.4 can be found at GitHub [43]. So four out of five atoms are hydrogens, from which it is already known that it can be optimized quite well since the required number of functions to describe all necessary orbitals is relatively tiny. In addition, the peak is caused by the basis variation of aug-cc-pCDZ and aug-pc-2, which both describe the same physical behaviors by default. Therefore, it makes sense that the relative improvement is relatively high since the only task is to fit the same amount of parameters onto each other. The success of the basis variation of cc-pvdz and cc-pvtz can also be attributed to the difference between the two basis sets, which is only one ζ-function per orbital. The previously explained behaviors also clarify why most results are between zero and ten percent of improvement. It is difficult to significantly improve smaller basis sets if the molecule requires all the extra functions provided by the larger basis sets. In comparison, it is like fitting a quadratic function with a linear one. It is possible to get a decent fit in a specific small area, but if the area extends, the fit will fail.
Now focus on the best-optimized molecule of all optimized molecules, the H_2 molecule.

In figure 3.5 the different Energies are shown for the H_2 molecule. Here the progress is clear to witness for every basis set variation. The reason that, in particular, this molecule can be optimized so well is due to its simplicity. Only s-orbitals are necessary to describe, H_2 therefore, the most additional functions provided by more extensive basis sets do not significantly enhance the DFT result. Even the solution to the Schrödinger equation is known for the hydrogen molecule. Therefore, the ground state energy depends less on the actual number of functions than the parameters of used functions. To conclude, the optimizer can improve the most basis set using the non-weighted loss function. The improvement is noticeable for specific molecules, such as H_2 or CH_4. The progress would be relatively minor for most other molecules and would not be helpful in applications outside the optimization process. The weighted loss function was created to improve further the optimization process, which will now be evaluated.

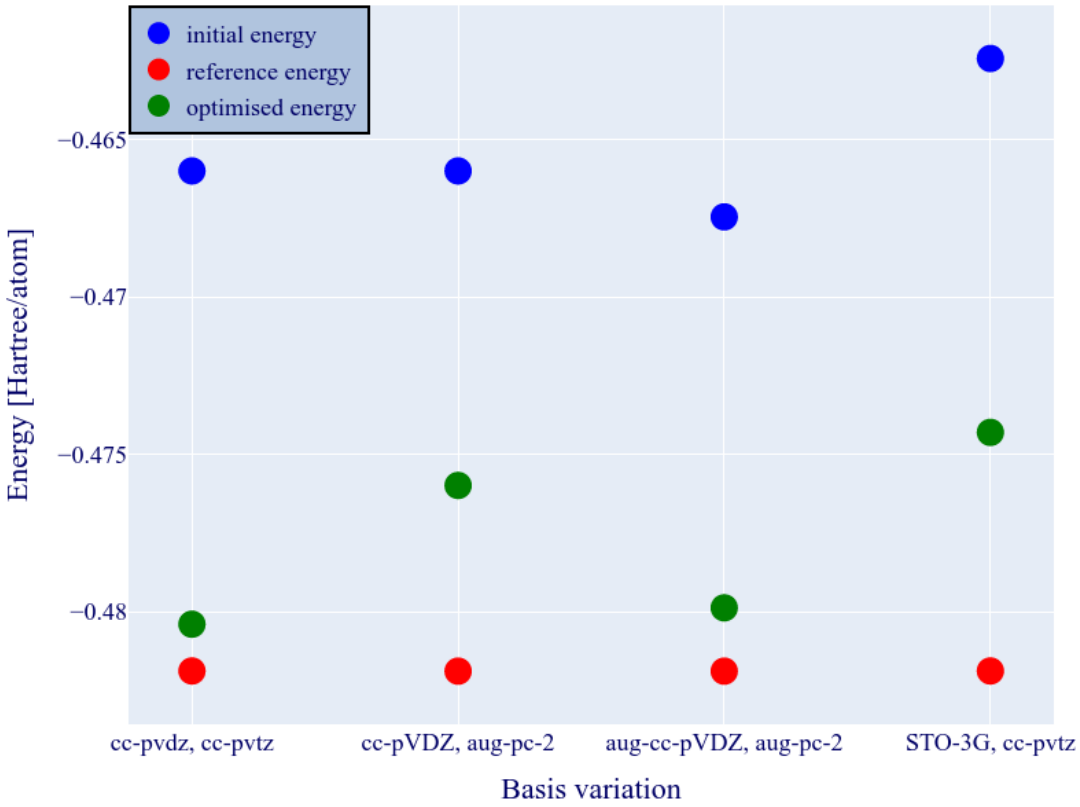


Figure 3.5: Energies of the H_2 for different basis set variations. The initial basis (blue), the reference basis (red), and the optimized basis (green).

3.4 Evaluation of the weighted optimization

Now the weight loss function will be discussed. Since the results are from the same type (for example, energy differences), the display graph looks familiar. As explained in Section 2.2, the weighed loss function considers the molecular orbital energies. The loss function should prefer the outer atomic orbitals because these are the orbitals that are more relevant for chemical interactions. The projection is no longer normalized to one, which is a disadvantage.

From figures 3.6 and 3.7 the relevant information can be extracted again. Unfortunately, the results do not show the intended improvement. All molecules were optimized for the basis variations shown in Figure 3.6. The total and the relative improvements are similar to the non-weighed loss function. Regardless, the weighted loss function has one major disadvantage: it results in a much higher failure rate than the non-weighted loss function.

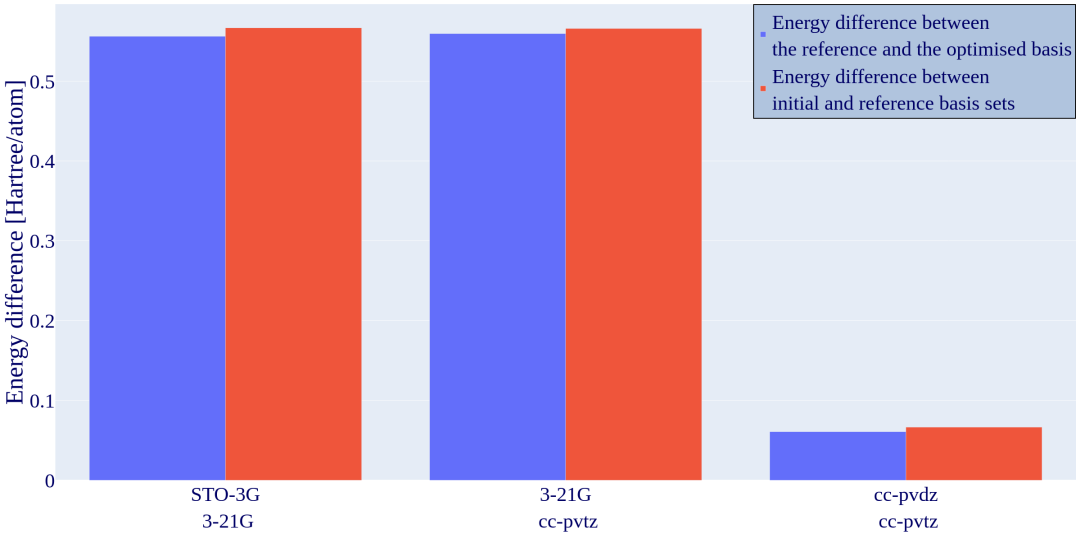


Figure 3.6: In blue, the energy difference between the energy using the reference (larger) basis set and the energy calculated throw the optimized basis. In red, the energy difference between the energy using the reference (larger) basis set and the energy using the initial (smaller) basis set. $\Delta E = E_{ref} - E_{opt}$ The difference between the red and the blue bar for the given basis variation on the x-Axis displays the improvement done by the learning.

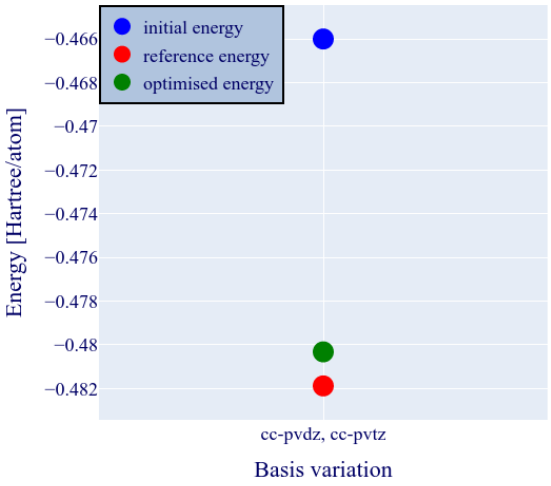


Figure 3.8: Energy's of the initial basis (blue), the reference basis (red), as well as the optimized basis (green) for the H_2 molecule.

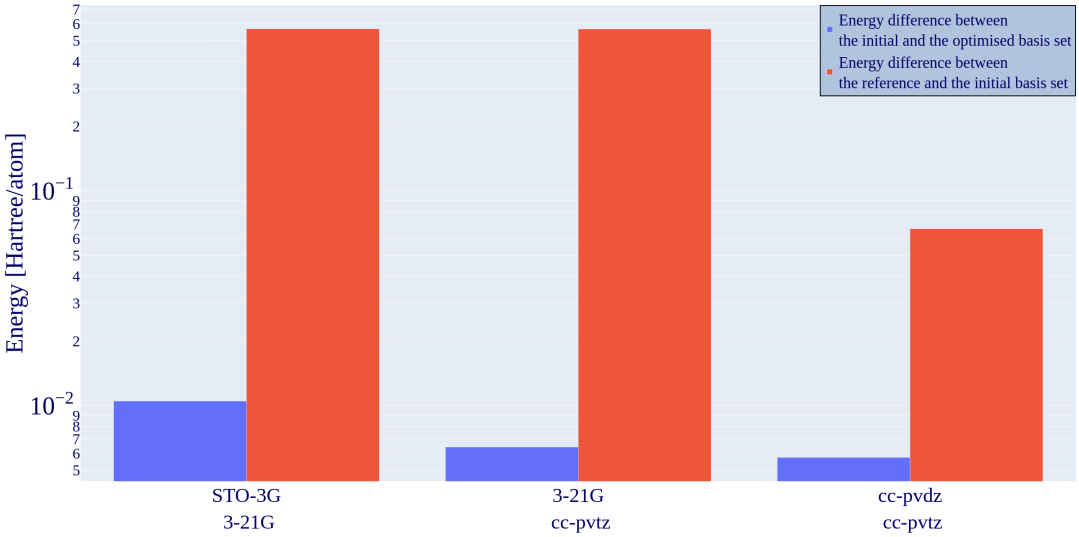


Figure 3.7: Absolute Energy difference between the basis set using the optimized basis set and the initial energy using the small basis set, in Hartree per Atom, in blue. The energy difference between the reference and the initial basis set is red. $\Delta E = E_{opt} - E_{init}$ The y-axis is displayed logarithmically.

For example, for the H_2 molecule, only the one variation shown in the figure 3.8 were possible. All other results for H_2 the molecule were discarded because they diverged from the beginning. The same behavior was discovered not only in the H_2 molecule, but was experienced for the vast majority of other molecules.

We want to look at Figure 3.9, which shows there are still decent results for different molecules, but these can be mainly attributed to the basis variation of cc-pvdz and cc-pvtz. It is known that these are pretty similar, only differing in one ζ-function. In the case of the weight loss function, the focus of the optimization should move to the outer orbitals, which are known to be attributed to diffuse or polarization functions in larger basis sets. Since these functions are not defined in smaller basis sets, optimizing them will be even more complex or nearly impossible. However, if an optimization finishes successfully, the results should be better since the relevant functions will be considered. Comparing the improvements achieved by the two different loss functions contradicts this assumption. The choice of the loss function depends on the evaluation of the machine learning process that will take place in the next part.

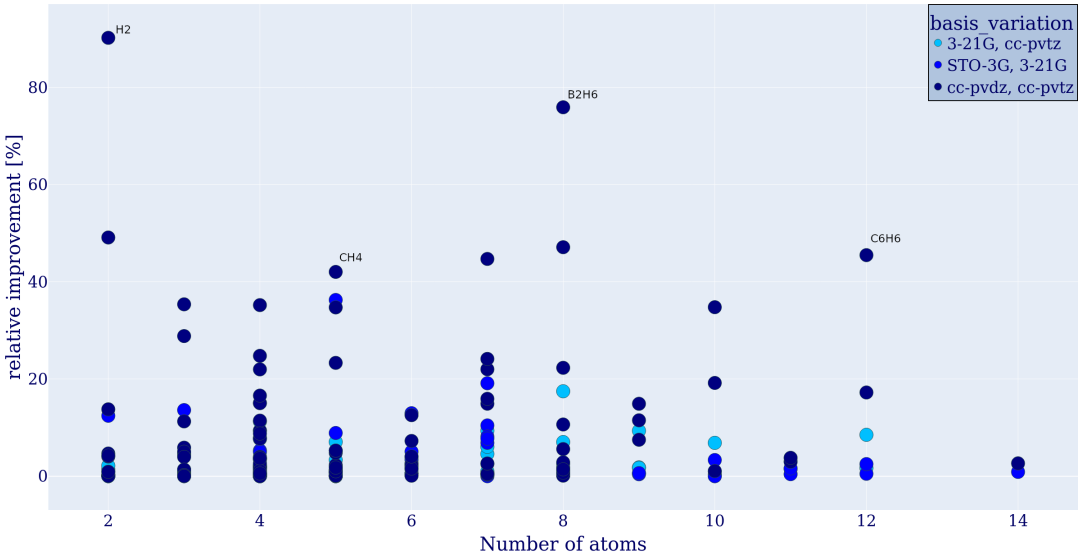


Figure 3.9: Relative improvement of an optimization $\frac{100}{E_{ref.-init}} * E_{optb-init}$ depends on the number of Atoms in a certain molecule. Displayed for the optimized basis variations. A more detailed version of this figure can be found in the Appendix 4.2.

3.5 The machine learning

After evaluating the relevant results from a physics point of view now, the machine learning process will be discussed in more detail. Therefore, the different optimizers will be discussed, and their dependency on the learning rate will be reviewed. The Machine Learning process should also be evaluated concerning both used loss functions.

3.5.1 Comparison of Adam and gradient descent

The best result was chosen for every molecule and basis set variation, which leads to 362 optimized basis sets for the W4-17 using the non-weighed loss function. Regarding tabular 3.2, 111 optimized basis sets were provided by the gradient descent optimizer and the remaining 251 by the Adam optimizer. The success rate gives the percentage for the number of results, leading to an improved basis for all results.

	w417	G2
total gradient descent results	3108	/
total Adam results	9310	9800
usable gradient descent results	395	/
usable Adam results	2133	1286
Success rate gradient descent	12.71 %	/
Success rate Adam	22.911%	13.22%
Number of gradient descent results in the best results	111	/
Number of gradient Adam results in the best results	251	256
Success rate of gradient descent in the best results	30.66%	/
Success rate of Adam in the best results	69.34%	100%

Table 3.2: Data created by optimizers and their success rate using the non-weighed loss function. The "best result" is the set of the best learning for each molecule and basis variation.

The "best result" is the set of parameters with the best optimization outcome for each molecule and basis variation. The success rate favors the Adam optimizer, which makes sense since Adam is the more powerful optimizer invented in 2015. Nevertheless, the gradient descent optimizer is also capable of getting good results. Even though the total number of calculations done with the gradient descent optimizer is less than using Adam, more than 30 percent of the best result leads to gd. Only the Adam optimizers were used for the weighted loss function due to his better performance. Therefore, comparing the two optimizers for the weighed loss function is impossible. Nevertheless, there is no doubt that Adam is generally the more powerful optimizer. However, optimizing the basis sets is also possible using older, much simpler methods like gradient descent.

3.5.2 Learning rate dependency

As familiar with the basics of Adam, the optimizer heavily depends on the chosen learning rate. Therefore, every optimization collection (one molecule multiple basis set variations) was optimized with multiple learning rates. The different learning rates are shown in their corresponding tabular, which will be discussed now. The results for the G2 database can be found in the Appendix, since they were done to proof of concept. Here, only the results for the W4-17 databases will be examined.

learning rate	lr counts unfiltered	lr counts filtered	lr counts in best results
2e-09	2052	1	
2e-10	2018	1	
2e-08	1697	9	1.0
2e-07	1605	94	22.0
2e-06	1548	292	73.0
2e-05	1359	328	93.0
0.0002	1139	348	113.0
0.002	1000	169	60.0

Table 3.3: dependency of the learning rate for both Adam and gradient descent combined. The counts are the total number of the specific results. The filter drops out all wrong learning results. The data is calculated to throw the non-weighted data.

In table 3.3, the results combined for both optimizers are displayed for the non-weighed loss function. The optimizers prefer a larger learning rate between $0,002$ and 2×10^{-6}. It ends with a value of $0,002$ because previous results in the program setup diverged every time, so optimizing the basis set with more effective learning rates does not make sense. Split this up between the gradient descent optimizer and the Adam optimizer, leads to the tabular 3.4 and 3.5. Here one of the weak points of Adam appears. According to the basics, the Adam optimizer is sensitive to the carefully chosen learning rate. However, the gradient descent optimizer also gets results more likely using more effective learning rates.

learning rate	lr counts adam unfiltered	lr counts filtered	lr counts in best results
2e-09	1482		
2e-10	1462		
2e-08	1268	2.0	1.0
2e-07	1219	28.0	5.0
2e-06	1153	153.0	30.0
2e-05	1018	222.0	60.0
0.0002	890	286.0	100.0
0.002	818	156.0	55.0

Table 3.4: dependency of the learning rate for Adam. The counts are the total number of the specific results. The filter drops out all wrong learning results. The data is calculated to throw the non-weighted data.

learning rate	lr counts gd unfiltered	lr counts filtered	lr counts in best results
2e-09	570	1	
2e-10	556	1	
2e-08	429	7	
2e-06	395	139	43.0
2e-07	386	66	17.0
2e-05	341	106	33.0
0.0002	249	62	13.0
0.002	182	13	5.0

Table 3.5: dependency of the learning rate for gradient descent. The counts are the total number of the specific results. The filter drops out all wrong learning results. The data is calculated to throw the non-weighed data.

For comparison, tabular 3.6 showing the result using the weighed loss function. This optimization was just run with Adam due to the runtime efficiency. As expected, the results look similar to the non-weighed results, as the initial parameters and reference data are equal. Higher learning rates seem to increase the chance of failure; in contrast, lower learning rates do not improve significantly. Which also makes sense, remembering figure 1.9 from section 1.2. Lower learning rates are less likely to diverge, but there are also not capable of finding deeper minima in the hyperspace.

learning rate	lr counts adam unfiltered	lr counts filtered	lr counts in best results
0.002	440	86	46.0
0.0002	440	173	53.0
2e-05	439	182	39.0
2e-06	432	183	38.0
2e-07	410	213	56.0
2e-08	388	211	8.0
2e-09	358	207	3.0
2e-10	339	205	
2e-11	321	205	
2e-12	310	204	

Table 3.6: dependency of the learning rate for Adam. The counts are the total number of the specific results. The filter drops out all wrong learning results. The data is calculated to throw the non-weighted data.

After discussing the optimizer and its learning rates now, let us take a closer look at the actual learning of a specific case. Therefore, the leanings from table 3.10 will be used again. Recognize that the H_2 was the molecule where the best gain overall was accomplished. The H_2 molecule here was computed using the non-weighed loss function, which can be noticed on the *best_f* column, which is close to one.

color	learning rate	maxiter	method	f_rtol	best_f	best_df
pink	2E-06	1000000	adam	0	-0.999375	4.9036330551644E-12
green	2E-07	1000000	adam	0	-0.999378	9.7255536957164E-14
grey	2E-08	1000000	adam	0	-0.996543	9.3097229925121E-10
	best_dcnorm	**best_i**	**max_i**	**STO-3G_energy**	**cc-pvtz_energy**	**STO-3G_opt_energy**
pink	4.5936396E-06	39937	84601	-0.924853	-0.963764	-0.938351
green	4.6145201E-07	394235	734601	-0.924853	-0.963764	-0.947299
grey	4.8948997E-08	999999	999999	-0.924853	-0.963764	-0.948607

Figure 3.10: Tabular with the results used in the plots done by the tensor board.

The figure 3.10 represents the results of the learning displayed in figure 3.11. Gray has the lowest learning rate of $2e-8$. Here the previously mentioned problem gets more precise. The optimizer can find better parameters for the initial guess. However, as the step size in the hyperspace is limited, the optimizer never finds the actual deep minima as the optimizer with a more significant learning rate. The green function show learning with a learning rate of $2e-7$ and the pink curve with a learning rate of $2e-6$. Both again mirror the results of previous discussions about the learning rates. Using the learning rates is getting a significant improvement for the initial parameters. Both get similar results, but the lower learning rate needs a much longer time to get the best result. After searching for the best results and still doing so, the optimizer tries to find other minima. However, it is no longer possible; the trace of the projection and, therefore, the value of the loss function gets higher again and starts to diverge, which is called overfitting. This behavior is a common problem of ML. Then the optimizer tries to fit some parameters better and better by ignoring other input parameters. Therefore, the learning for the specific parameter gets better in every epoch while the value of the learning rate diverges. Going to an even larger learning rate, the loss function's value immediately diverges as it "over jumps" all the minima.

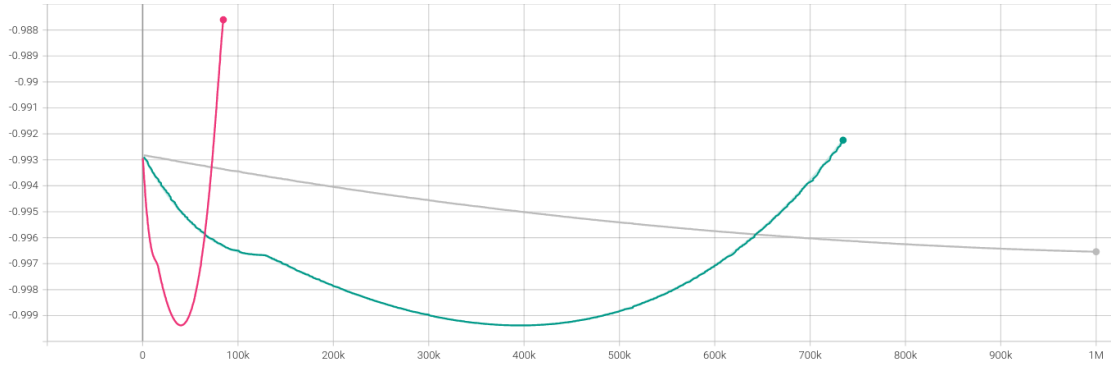


Figure 3.11: Loss of function due to the learnings from figure 3.10.

Comparing the learning rates used for the H_2 molecule with the general learning rate dependency might be surprising, as they are pretty small compared to the average results from tabular 3.4. Since the initial value of the loss function is already close to one (for the non-weighed loss function), it makes sense that a lower learning rate will be more

effective because a higher learning rate will quickly begin to overfit the loss function. Now it is time to conclude the ML topic. Since the non-weighted loss function brings up more valuable results, this will be the preferred one. For the comparison between Adam and gd, the clear favorite is Adam for the much faster convergence behavior and that they bring up more of the best results measured. Selecting the right learning rat is not as easy as specifying a specific one, but higher learning rates between $0,002$ and 2×10^{-6} pretend to be performing better than smaller or larger ones. However, every new input data set must be re-evaluated in this case. The essential outcome of this valuation regarding the ML process is that the framework works and can be used to improve basis sets for particular basis sets. As the evaluation is finished, it is time to do a Résumé and provide an outlook from the current point.

4 Résumé and Outlook

Now it is time to look back and evaluate what has been done and what can be done with this Thesis's outcome. As the results show, optimizing basis sets in principle is possible. Moreover, for elementary molecules like H_2, the optimized basis set is excellent compared to the initial basis set. For more complex molecules, the improvement is less significant but measurable. If we look at the basis set exchange website [3] we will expire dozens of different basis sets with a variety of complexity for every possible atom. They all have a significant adaptation that can be used outside the specific molecule. Which led to a clear result: optimizing basis sets for a specific molecule is possible. Nevertheless, since the created basis sets are limited in their use case the vast majority of the time, it just does not make sense to optimize a basis set to a specific molecule to use later in a larger system. That may sound not very pleasant, but there is hope. Over the last decades, all the basis sets used as initial and reference were very well-adjusted. So even a minor improvement can be evaluated as success. Since there is no actual application for this by the direct use of the out coming basis sets, this learning may be used in other ways. The optimization Process can be implemented in a neuronal network. The neuronal network can be trained using the optb module for way more basis set variations and molecules than it was done for this Thesis. Then the neuronal network can use this information to create a new basis set for standalone an atom without binding on a particular molecule. In the best case, the trained neural network will be capable of outputting a specific basis set depending on the input. For example, if the user wants a basis set that includes diffuse functions, a different amount of ζ functions, or polarization functions. This way, getting the best basis set for the specific system within seconds would be possible. Appropriately, this can speed up DFT calculations while simultaneously achieving better results. Especially for heavier atoms, this would be considerable furtherance, since these basis sets are usually quite large and take much computational power. Another more straightforward use case for the framework is to build a AI, which does not propose a particular new basis set but offers the best basis sets for a particular system. A framework like this could help inexperienced scientists get easier access to ab initio DFT calculation using Gaussian basis sets. Especially chemists, who are the primary user of gaussian basis sets, would be able to run DFT without the theoretical physics background. Another helpful method would be to optimize a basis set for a specific molecule and use this now smaller basis set in a more complex system. It is common to split up large structures like proteins into different molecular substructures and use different basis sets depending on the specific substructure. Therefore, substructures that are more relevant to the chemical interactions are defined by larger basis sets, and smaller basis sets will describe less relevant substructures. At this point, the optimized basis sets can describe the less relevant substructures and enhance the overall accuracy.

In the end, we want to return to Hartree's initial quote, which opened this Thesis. Hartree first makes it possible to solve systems of complex molecules by reducing the number of the required wave functions using a sophisticated Ansatz. It would be arrogant to try to accomplish such a world-changing task in a Master's thesis. In his manner, we tried to reduce the number of required basis functions for specific molecules. Which indeed were successful, though not practical. Nevertheless, achieving this knowledge has to be tried, and it was tried in this Thesis.

Bibliography

[1] Charles Galton Darwin. "Douglas Rayner Hartree, 1897-1958". In: *Biographical Memoirs of Fellows of the Royal Society* 4 (1958). _eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsbm.1958.0010, pp. 102–116. DOI: 10.1098/rsbm.1958.0010. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsbm.1958.0010.

[2] W. J. Hehre, R. F. Stewart, and J. A. Pople. "Self-Consistent Molecular-Orbital Methods. I. Use of Gaussian Expansions of Slater-Type Atomic Orbitals". In: *J. Chem. Phys.* 51 (1969), pp. 2657–2664. DOI: 10.1063/1.1672392.

[3] Benjamin P. Pritchard et al. "A New Basis Set Exchange: An Open, Up-to-date Resource for the Molecular Sciences Community". In: *J. Chem. Inf. Model.* 59 (2019), pp. 4814–4820. DOI: 10.1021/acs.jcim.9b00725.

[4] Wolfram Koch and Max C. Holthausen. *A chemist's guide to density functional theory / Wolfram Koch, Max C. Holthausen.* Second edition. Wiley-VCH, 2001. ISBN: 3-527-60004-3. URL: https://search.ebscohost.com/login.aspx?direct=true&db=cat06365a&AN=ulb.1679709372&lang=de&site=eds-live&scope=site.

[5] Dirk Werner. *Einführung in die höhere Analysis : topologische Räume, Funktionentheorie, gewöhnliche Differentialgleichungen, Maß- und Integrationstheorie, Funktionalanalysis / Dirk Werner.* Jan. 1, 2009. ISBN: 978-3-540-79599-5. URL: http://www.loc.gov/catdir/enhancements/fy1409/2009920037-d.html (visited on 10/06/2021).

[6] Per-Olov Löwdin. "Quantum Theory of Many-Particle Systems. III. Extension of the Hartree-Fock Scheme to Include Degenerate Systems and Correlation Effects". In: *Phys. Rev.* 97.6 (Mar. 1955). Publisher: American Physical Society, pp. 1509–1520. DOI: 10.1103/PhysRev.97.1509. URL: https://link.aps.org/doi/10.1103/PhysRev.97.1509.

[7] P. Hohenberg and W. Kohn. "Inhomogeneous Electron Gas". In: *Phys. Rev.* 136.3 (Nov. 1964). Publisher: American Physical Society, B864–B871. DOI: 10.1103/PhysRev.136.B864. URL: https://link.aps.org/doi/10.1103/PhysRev.136.B864.

[8] Nicola Marzari. "A gentle introduction to DFT calculations - April 2020". URL: https://www.materialscloud.org/learn/sections/VNL7RL/a-gentle-introduction-to-dft-calculations-april-2020 (visited on 04/13/2022).

[9] Eric Welch. "Eric Welch PhD Dissertation - DFT modeling of halide perovskites for optoelectronic applications". PhD thesis. Oct. 2019. DOI: 10.13140/RG.2.2.22423.52646.

[10] Materials Cloud. *Intro to DFT - Day 2: Density-functional practice - Nicola Marzari*. Apr. 16, 2020. URL: https://www.youtube.com/watch?v=6tLx-alUZ8 (visited on 06/01/2022).

[11] Jürgen Henk. "Concepts and Methods in Electronic Structure Calculations". Feb. 6, 2009.

[12] J. P. Perdew and Alex Zunger. "Self-interaction correction to density-functional approximations for many-electron systems". In: *Phys. Rev. B* 23.10 (May 1981). Publisher: American Physical Society, pp. 5048–5079. DOI: 10.1103/PhysRevB.23.5048. URL: https://link.aps.org/doi/10.1103/PhysRevB.23.5048.

[13] John P. Perdew, Kieron Burke, and Matthias Ernzerhof. "Generalized Gradient Approximation Made Simple". In: *Phys. Rev. Lett.* 77.18 (Oct. 1996). Publisher: American Physical Society, pp. 3865–3868. DOI: 10.1103/PhysRevLett.77.3865. URL: https://link.aps.org/doi/10.1103/PhysRevLett.77.3865.

[14] A. D. Becke. "Density-functional exchange-energy approximation with correct asymptotic behavior". In: *Phys. Rev. A* 38.6 (Sept. 1988). Publisher: American Physical Society, pp. 3098–3100. DOI: 10.1103/PhysRevA.38.3098. URL: https://link.aps.org/doi/10.1103/PhysRevA.38.3098.

[15] S. H. Vosko, L. Wilk, and M. Nusair. "Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis". In: *Canadian Journal of Physics* 58.8 (Aug. 1, 1980). Publisher: NRC Research Press, pp. 1200–1211. ISSN: 0008-4204. DOI: 10.1139/p80-159. URL: https://doi.org/10.1139/p80-159 (visited on 05/24/2022).

[16] Chengteh Lee, Weitao Yang, and Robert G. Parr. "Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density". In: *Phys. Rev. B* 37.2 (Jan. 1988). Publisher: American Physical Society, pp. 785–789. DOI: 10.1103/PhysRevB.37.785. URL: https://link.aps.org/doi/10.1103/PhysRevB.37.785.

[17] Larry A. Curtiss et al. "Assessment of Gaussian-2 and density functional theories for the computation of enthalpies of formation". In: *The Journal of Chemical Physics* 106.3 (1997). Eprint: https://doi.org/10.1063/1.473182, pp. 1063–1079. DOI: 10.1063/1.473182. URL: https://doi.org/10.1063/1.473182.

[18] David Sherrill. *Basis Sets part 1*. Feb. 14, 2021. URL: https://www.youtube.com/watch?v=Hk4YRb4okC4 (visited on 06/01/2022).

[19] Juerg Hutter. "Gaussian and Plane Waves Method (GPW)". In: (2022). ISSN: 17590876. URL: https://pc2.uni-paderborn.de/fileadmin/pc2/cp2k_workshop/gpw.pdf.

[20] A. Szabo and N.S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Dover Books on Chemistry. Dover Publications, 1996. ISBN: 978-0-486-69186-2. URL: https://books.google.de/books?id=6mV9gYzEkgIC.

[21] Alexander L. Fradkov. "Early History of Machine Learning". In: *IFAC-PapersOnLine*. 21st IFAC World Congress 53.2 (Jan. 1, 2020), pp. 1385–1390. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.1888. URL: https://www.sciencedirect.com/science/article/pii/S2405896320325027 (visited on 09/21/2022).

[22] Alexander Amini and Ava Soleimany. *MIT Deep Learning 6.S191*. MIT Deep Learning 6.S191. URL: http://introtodeeplearning.com (visited on 06/15/2022).

[23] Michael Frank, Dimitris Drikakis, and Vassilis Charissis. "Machine-Learning Methods for Computational Science and Engineering". In: *Computation* 8 (Mar. 2020), p. 15. DOI: 10.3390/computation8010015.

[24] *xitorch: differentiable scientific computing library*. original-date: 2020-09-25T12:19:45Z. May 4, 2022. URL: https://github.com/xitorch/xitorch (visited on 06/09/2022).

[25] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[26] Musstafa. *Optimizers in Deep Learning*. MLearning.ai. Feb. 12, 2022. URL: https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0 (visited on 09/21/2022).

[27] S. Raschka and V. Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-Learn, and TensorFlow 2, 3rd Edition*. Expert insight. Packt Publishing, 2019. ISBN: 978-1-78995-575-0. URL: https://books.google.de/books?id=n1cjyAEACAAJ.

[28] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/neural-networks-3/ (visited on 09/21/2022).

[29] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (2016). DOI: 10.48550/ARXIV.1609.04747. URL: https://arxiv.org/abs/1609.04747.

[30] Daksh Trehan. *Gradient Descent Explained*. Medium. May 21, 2021. URL: https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c (visited on 09/21/2022).

[31] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2014). DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[32] Vitaly Bushaev. *Understanding RMSprop — faster neural network learning*. Medium. Sept. 2, 2018. URL: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a (visited on 09/21/2022).

[33] Atilim Gunes Baydin et al. "Automatic differentiation in machine learning: a survey". In: (2015). Publisher: arXiv. DOI: 10.48550/ARXIV.1502.05767. URL: https://arxiv.org/abs/1502.05767.

[34] Rubin H. Landau gnd_133312925, Cristian C. Bordeianu, and Manuel J. Páez. *Computational physics : problem solving with Python / Rubin H. Landau, Manuel J. Páez, Cristian C. Bordeianu*. Jan. 1, 2015. ISBN: 978-3-527-68467-0. URL: https://zbmath.org/?q=an:1341.70001 (visited on 06/09/2022).

[35] Jacob Schmieder. "EEG Analyse zum Test von Bewusstsein". Bachelor Thesis. Martin-Luther-Universität Halle-Wittenberg, 2019. URL: http://www.physik.uni-halle.de/Fachgruppen/kantel/Bachelorarbeit_Schmieder.pdf.

[36] Soeren Laue. *On the Equivalence of Forward Mode Automatic Differentiation and Symbolic Differentiation*. 2019. DOI: 10.48550/ARXIV.1904.02990. URL: https://arxiv.org/abs/1904.02990.

[37] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[38] Peize Lin, Xinguo Ren, and Lixin He. "Strategy for constructing compact numerical atomic orbital basis sets by incorporating the gradients of reference wavefunctions". In: *Phys. Rev. B* 103.23 (June 2021). Publisher: American Physical Society, p. 235131. DOI: 10.1103/PhysRevB.103.235131. URL: https://link.aps.org/doi/10.1103/PhysRevB.103.235131.

[39] Daniel Sanchez-Portal, Emilio Artacho, and Jose M. Soler. "Projection of plane-wave calculations into atomic orbitals". In: *Solid State Communications* 95.10 (1995), pp. 685–690. ISSN: 0038-1098. DOI: https://doi.org/10.1016/0038-1098(95)00341-X. URL: https://www.sciencedirect.com/science/article/pii/003810989500341X.

[40] Daniel Sánchez-Portal, Emilio Artacho, and José M. Soler. "Analysis of atomic orbital basis sets from the projection of plane-wave results". In: *Journal of Physics: Condensed Matter* 8.21 (May 1996). Publisher: IOP Publishing, pp. 3859–3880. DOI: 10.1088/0953-8984/8/21/012. URL: https://doi.org/10.1088/0953-8984/8/21/012.

[41] Emilio Artacho and Lorenzo Miláns del Bosch. "Nonorthogonal basis sets in quantum mechanics: Representations and second quantization". In: *Physical Review A* 43.11 (June 1, 1991), pp. 5770–5777. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.43.5770. URL: https://link.aps.org/doi/10.1103/PhysRevA.43.5770 (visited on 09/17/2022).

[42] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[43] Jacob Schmieder. *OptBasisSets*. URL: https://github.com/Jaikinator/OptBasisSets (visited on 07/21/2022).

[44] Qiming Sun et al. "PySCF: the Python-based simulations of chemistry framework". In: *WIREs Computational Molecular Science* 8.1 (2018). _eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.1340, e1340. DOI: https://doi.org/10.1002/wcms.1340. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1340.

[45] Qiming Sun et al. "Recent developments in the PySCF program package". In: *The Journal of Chemical Physics* 153.2 (2020). Eprint: https://doi.org/10.1063/5.0006074, p. 024109. DOI: 10.1063/5.0006074. URL: https://doi.org/10.1063/5.0006074.

[46] M. F. Kasim and S. M. Vinko. "Learning the Exchange-Correlation Functional from Nature with Fully Differentiable Density Functional Theory". In: *Phys. Rev. Lett.* 127.12 (Sept. 2021). Publisher: American Physical Society, p. 126403. DOI: 10.1103/PhysRevLett.127.126403. URL: https://link.aps.org/doi/10.1103/PhysRevLett.127.126403.

[47] Ask Hjorth Larsen et al. "The atomic simulation environment—a Python library for working with atoms". In: *Journal of Physics: Condensed Matter* 29.27 (July 12, 2017), p. 273002. ISSN: 0953-8984, 1361-648X. DOI: 10.1088/1361-648X/aa680e. URL: https://iopscience.iop.org/article/10.1088/1361-648X/aa680e (visited on 07/21/2022).

[48] Martín Abadi et al. *TensorFlow, Large-scale machine learning on heterogeneous systems*. Nov. 2015. DOI: 10.5281/zenodo.4724125.

[49] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020). Publisher: Springer Science and Business Media LLC, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[50] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[51] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[52] Plotly Technologies Inc. *Collaborative data science*. Place: Montreal, QC Publisher: Plotly Technologies Inc. 2015. URL: https://plot.ly.

[53] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007). Publisher: IEEE COMPUTER SOC, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[54] Susi Lehtola et al. “Recent developments in libxc — A comprehensive library of functionals for density functional theory”. In: *SoftwareX* 7 (Jan. 1, 2018). Publisher: Elsevier, pp. 1–5. ISSN: 2352-7110. DOI: 10.1016/j.softx.2017.11.002. URL: https://doi.org/10.1016/j.softx.2017.11.002 (visited on 09/25/2022).

[55] Amir Karton, Nitai Sylvetsky, and Jan M. L. Martin. “W4-17: A diverse and high-confidence dataset of atomization energies for benchmarking high-level electronic structure methods”. In: *Journal of Computational Chemistry* 38.24 (2017). Eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.24854, pp. 2063–2075. DOI: https://doi.org/10.1002/jcc.24854. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.24854.

[56] J. Stephen Binkley, John A. Pople, and Warren J. Hehre. “Self-consistent molecular orbital methods. 21. Small split-valence basis sets for first-row elements”. In: *J. Am. Chem. Soc.* 102 (1980), pp. 939–947. DOI: 10.1021/ja00523a008.

[57] Thom H. Dunning. “Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen”. In: *J. Chem. Phys.* 90 (1989), pp. 1007–1023. DOI: 10.1063/1.456153.

[58] Frank Jensen. “Polarization Consistent Basis Sets. 4: The Elements He, Li, Be, B, Ne, Na, Mg, Al, and Ar”. In: *J. Phys. Chem. A* 111 (2007), pp. 11198–11204. DOI: 10.1021/jp068677h.

[59] Frank Jensen. “Polarization consistent basis sets. III. The importance of diffuse functions”. In: *J. Chem. Phys.* 117 (2002), pp. 9234–9240. DOI: 10.1063/1.1515484.

[60] Frank Jensen. “Polarization consistent basis sets. II. Estimating the Kohn-Sham basis set limit”. In: *J. Chem. Phys.* 116 (2002), pp. 7372–7379. DOI: 10.1063/1.1465405.

[61] Frank Jensen. “Polarization consistent basis sets: Principles”. In: *J. Chem. Phys.* 115 (2001), pp. 9113–9125. DOI: 10.1063/1.1413524.

Appendix

Used Molecules from the G2 Database

PH3	SiF4	CH3S
P2	H3CNH2	CS2
CH3CHO	methylenecyclopropane	SiH2_s1A1d
H2COH	CH3CH2OH	C4H4S
CS	F	N2H4
OCHCHO	NaCl	OH
C3H9C	CH3Cl	CH3OCH3
CH3COF	CH3SiH3	C5H5N
CH3CH2OCH3	AlF3	H2O
HCOOH	C2H3	HCl
HCCl3	ClF	CH2_s1A1d
HOCl	PF3	CH3CH2SH
H2	PH2	CH3NO2
SH2	CH3CN	Cl
C2H2	cyclobutene	Be
C4H4NH	CH3ONO	BCl3
CH3SCH3	SiH3	C4H4O
SiH2_s3B1d	C3H6_D3h	Al
CH3SH	CO2	CH3O
CH3CO	NO	CH3OH
CO	trans-butane	C3H7Cl
ClF3	H2CCHCl	isobutane
SiH4	LiH	Na
C2H6CHOH	NH2	CCl4
CH2NHCH2	CH	CH3CH2O
isobutene	CH2OCH2	H2CCHF
HCO	C6H6	C3H7
bicyclobutane	CH3CONH2	CH3
LiF	H2CCHCN	O3
Si	butadiene	P
C2H6	C	C2H4
CN	H2CO	NCCN
ClNO	CH3COOH	S2
S	HCF3	AlCl3

SiCl4
SiO
C3H4_D2d
H
COF2
2-butyne
C2H5
BF3
N2O
F2O
SO2
H2CCl2
CF3CN
HCN
C2H6NH
OCS
B
ClO
C3H8
HF
O2
SO
NH
C2F4
NF3
CH2_s3B1d
CH3CH2Cl
CH3COCl
NH3
C3H9N
CF4
C3H6_Cs
Si2H6
HCOOCH3
O
CCH
N
Si2
C2H6SO
C5H8
H2CF2
Li2
CH2SCH2
C2Cl4
C3H4_C3v
CH3COCH3
F2
CH4
SH
H2CCO
CH3CH2NH2
Li
N2
Cl2
H2O2
Na2
BeH
C3H4_C2v
NO2

Used Molecules from the W417 Database

acetaldehyde
acetic
alcl3
alcl
alf3
alf
alh3
alh
allene
allyl
b2h6
benzene
beta-lactim
bf3
bf
bh3
bhf2
bh
bn3pi
borole
c2cl2
c2cl4
c2cl6
c2clh3
c2clh5
c2clh
c2f4
c2f6
c2h2
c2h3f
c2h4
c2h5f
c2h6
cch
ccl2h2
ccl2
ccl2o
ccl3h
ccl4
cclh3
cf2cl2
cf2
cf4
cf
ch2ch
ch2clf
ch2c
ch2f2
ch2nh2
ch2nh
ch2-sing
ch2-trip
ch3f
ch3
ch3nh2
ch3nh
ch3ph2
ch4
c-hcoh
chf3
ch
c-hono
c-hooo
cis-c2f2cl2
cl2
clcn
clcof
clf
clno
clo
c-n2h2
cn
co2
co
cs2
cs
cyclobutadiene
cyclobutane
cyclobutene
cyclopentadiene

cyclopropane
cyclopropene
cyclopropyl
dioxetan2one
dioxetane
dioxirane
dithiotane
ethanol
f2co
f2
fccf
fno
formamide
formic-anhydride
formic
furan
glyoxal
h2ccn
h2cn
h2co
h2
h2no
h2o
h2s
hccf
hcl
hclo4
hcnh
hcn
hcno
hcof
hco
hf
hnc
hnco
hnnn
hno
hocl
hoclo2
hoclo
hocn
hof
honc
hooh
hoo
hs
ketene
methanol
n2h4
n2h
n2
n2o4
n2o
n-butane
nccn
nh2cl
nh2f
nh2
nh2oh
nh3
nh
no2
no
n-pentane
o2
ocs
oh
oxadiazole
oxetane
oxirane
oxirene
p2
p4
pf3
pf5
ph3
propane
propene
propyne
pyrrole
s2
s2o
sf6
si2h6
sif4
sif
sih3f
sih4
sih
silole
sio
so2
so3
so
ssh
t-butadiene
tetrahedrane
t-hcoh
thiophene
t-hono
t-hooo
t-n2h2
trans-c2f2cl2

Learning Rate Results for the G2 Database

learning rate	lr counts unfiltered	lr counts filtered	lr counts in best results
2e-10	1612		
2e-09	1546		
2e-07	1276	64.0	9.0
2e-08	1272	3.0	
2e-06	1186	259.0	31.0
2e-05	1089	355.0	70.0
0.0002	975	374.0	106.0
0.002	844	231.0	40.0

Table 4.1: Dependency of the learning rate for both Adam and gradient descent combined. The counts are the total number of the specific results. The filter drops out all wrong learning results. The Data is calculated throw the non weighed data.

learning rate	lr counts adam unfiltered	lr counts filtered	lr counts in best results
2e-10	1612		
2e-09	1546		
2e-07	1276	64.0	9.0
2e-08	1272	3.0	
2e-06	1186	259.0	31.0
2e-05	1089	355.0	70.0
0.0002	975	374.0	106.0
0.002	844	231.0	40.0

Table 4.2: Dependency of the learning rate for Adam. The counts are the total number of the specific results. The filter drops out all wrong learning results. The Data is calculated throw the non weighed data.

Relative improvement of the basis optimization in dependency of the number of atoms

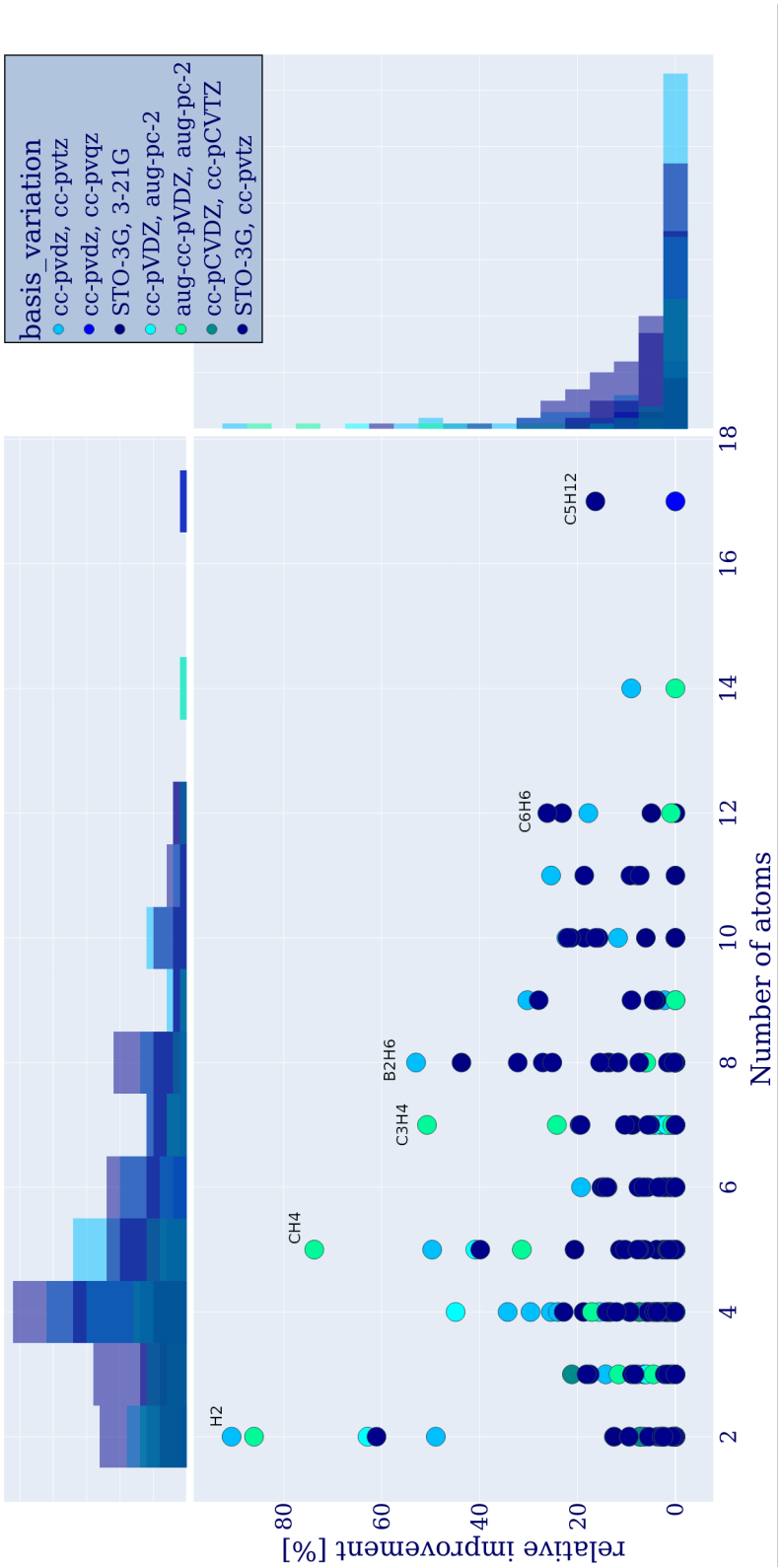


Figure 4.1: Relative improvement of an optimization $\frac{100}{E_{ref.-init}-E_{optb-init}}$ depends on the number of Atoms in a certain molecule. Displayed for different basis variations using the non weighed loss functions.

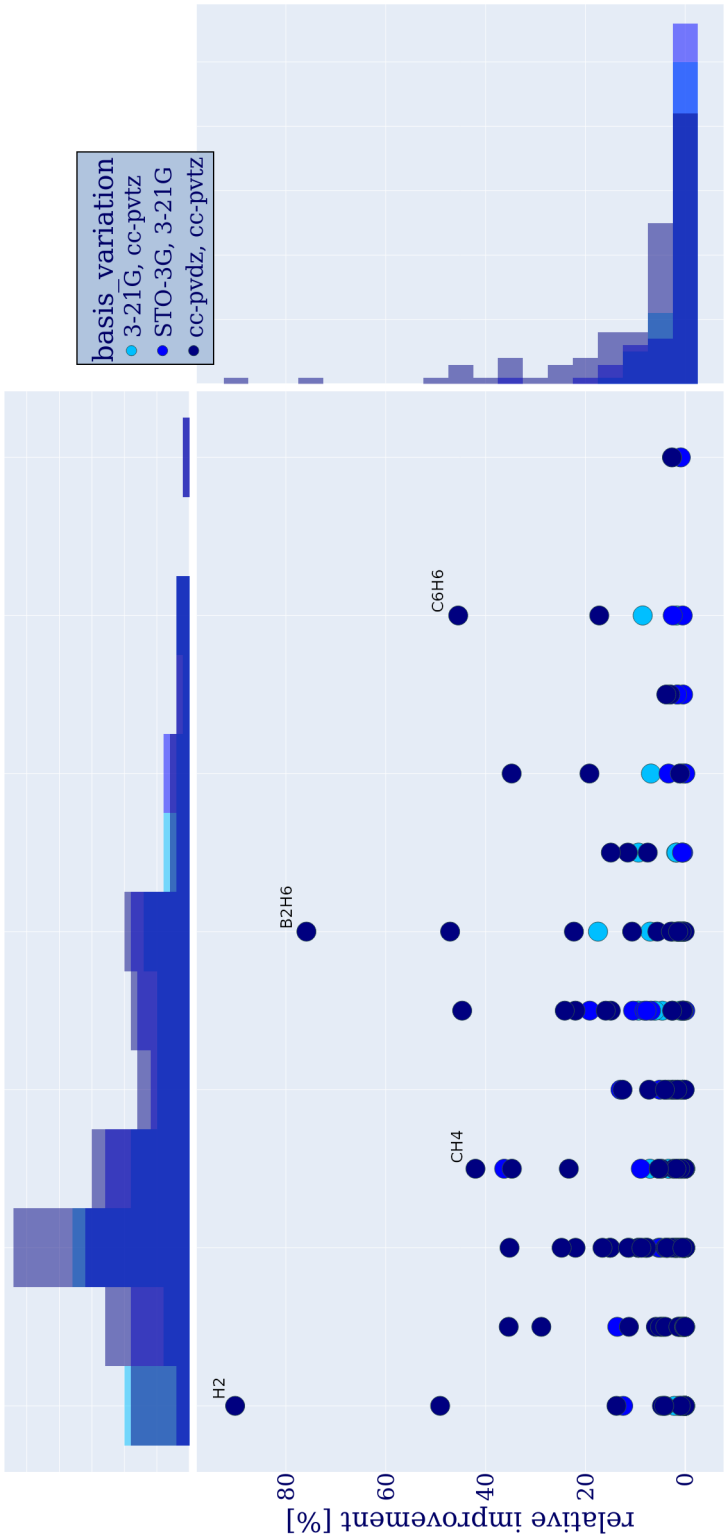


Figure 4.2: Relative improvement of an optimization $\frac{100}{E_{ref.-init}-E_{optb-init}}$ depends on the number of Atoms in a certain molecule. Displayed for the optimized basis variations using the weighed loss functions.

	molecule	file number	initial_energy	ref_energy	opt_energy	learning rate	maxiter	miniter	method	best_f	best_df	best_dxnorm	best_i	max_i	f_rtol	basis	ref. basis
1	h2	1	-0.92485305395	-0.944415477	-0.924853054	0.002	1000000	10000	adam	-0.51349	0.5134928	0.00489897942	0	774	0	STO-3G	3-21G
2	h2	6	-0.92485305395	-0.944415477	-0.924853054	2E-08	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-08	0	24601	0	STO-3G	3-21G
0	h2	3	-0.92485305395	-0.944415477	-0.924853054	2E-05	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-05	0	34601	0	STO-3G	3-21G
1	h2	8	-0.92485305395	-0.944415477	-0.924853054	2E-10	1000000	10000	adam	-0.51349	0.5134928	4.8989798E-10	0	24601	0	STO-3G	3-21G
2	h2	4	-0.92485305395	-0.944415477	-0.924853054	2E-06	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-06	0	24601	0	STO-3G	3-21G
3	h2	7	-0.92485305395	-0.944415477	-0.924853054	2E-09	1000000	10000	adam	-0.51349	0.5134928	4.8989793E-09	0	24601	0	STO-3G	3-21G
4	h2	10	-0.92485305395	-0.944415477	-0.924853054	2E-12	1000000	10000	adam	-0.51349	0.5134928	4.8990071E-12	0	24601	0	STO-3G	3-21G
5	h2	9	-0.92485305395	-0.944415477	-0.924853054	2E-11	1000000	10000	adam	-0.51349	0.5134928	4.8989799E-11	0	24601	0	STO-3G	3-21G
6	h2	5	-0.92485305395	-0.944415477	-0.924853054	2E-07	1000000	10000	adam	-0.51349	0.5134928	4.8989794E-07	0	24601	0	STO-3G	3-21G
7	h2	2	-0.94441547653	-0.96376385	-0.9444154765	0.0002	1000000	10000	adam	-0.52278	0.5227814	0.00044721359	0	999999	0	3-21G	cc-pvtz
8	h2	1	-0.94441547653	-0.96376385	-0.9444154765	0.002	1000000	10000	adam	-0.52278	0.5227814	0.00447213595	0	999999	0	3-21G	cc-pvtz
9	h2	6	-0.94441547653	-0.96376385	-0.9444154765	2E-08	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-08	0	24601	0	3-21G	cc-pvtz
0	h2	3	-0.94441547653	-0.96376385	-0.9444154765	2E-05	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-05	0	24601	0	3-21G	cc-pvtz
1	h2	8	-0.94441547653	-0.96376385	-0.9444154765	2E-10	1000000	10000	adam	-0.52278	0.5227814	4.4721362E-10	0	24601	0	3-21G	cc-pvtz
2	h2	4	-0.94441547653	-0.96376385	-0.9444154765	2E-06	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-06	0	24601	0	3-21G	cc-pvtz
3	h2	7	-0.94441547653	-0.96376385	-0.9444154765	2E-09	1000000	10000	adam	-0.52278	0.5227814	4.472136E-09	0	24601	0	3-21G	cc-pvtz
4	h2	10	-0.94441547653	-0.96376385	-0.9444154765	2E-12	1000000	10000	adam	-0.52278	0.5227814	4.4721612E-12	0	24601	0	3-21G	cc-pvtz
0	h2	9	-0.94441547653	-0.96376385	-0.9444154765	2E-11	1000000	10000	adam	-0.52278	0.5227814	4.4721363E-11	0	24601	0	3-21G	cc-pvtz
1	h2	5	-0.94441547653	-0.96376385	-0.9444154765	2E-07	1000000	10000	adam	-0.52278	0.5227814	4.4721359E-07	0	24601	0	3-21G	cc-pvtz
2	h2	2	-0.93199724014	-0.96376385	-0.959738276	0.0002	1000000	10000	adam	-0.52337	2.293E-10	0.00038187637	2114	24601	0	cc-pvdz	cc-pvtz
3	h2	1	-0.93199724014	-0.96376385	-0.9593918135	0.002	1000000	10000	adam	-0.5232	1.737E-07	0.00398756636	89	999999	0	cc-pvdz	cc-pvtz
4	h2	6	-0.93199724014	-0.96376385	-0.9319972401	2E-08	1000000	10000	adam	-0.521	0.5210002	6.3245553E-08	0	24601	0	cc-pvdz	cc-pvtz
5	h2	3	-0.93199724014	-0.96376385	-0.9606459823	2E-05	1000000	10000	adam	-0.52348	1.09E-10	5.7595008E-05	9600	34601	0	cc-pvdz	cc-pvtz
6	h2	8	-0.93199724014	-0.96376385	-0.9319972401	2E-10	1000000	10000	adam	-0.521	0.5210002	6.3245558E-10	0	24601	0	cc-pvdz	cc-pvtz
7	h2	4	-0.93199724014	-0.96376385	-0.9590759587	2E-06	1000000	10000	adam	-0.5235	7.242E-13	6.3721342E-06	74638	219601	0	cc-pvdz	cc-pvtz
0	h2	7	-0.93199724014	-0.96376385	-0.9319972401	2E-09	1000000	10000	adam	-0.521	0.5210002	6.3245554E-09	0	24601	0	cc-pvdz	cc-pvtz
1	h2	10	-0.93199724014	-0.96376385	-0.9319972401	2E-12	1000000	10000	adam	-0.521	0.5210002	6.3245646E-12	0	24601	0	cc-pvdz	cc-pvtz
2	h2	9	-0.93199724014	-0.96376385	-0.9319972401	2E-11	1000000	10000	adam	-0.521	0.5210002	6.3245558E-11	0	24601	0	cc-pvdz	cc-pvtz
3	h2	5	-0.93199724014	-0.96376385	-0.9319972401	2E-07	1000000	10000	adam	-0.521	0.5210002	6.3245553E-07	0	24601	0	cc-pvdz	cc-pvtz

Figure 4.3: Example for the raw output data out. The first two columns are the molecule's name and the file number in the folder structure. Then the initial energy, the reference energy (textitref_energy), as well as the energy using the optimized basis *opt_energy*, are listed. After this, the ML related parameters are listed. The learning rate column is self-speaking. On the right, the *maxiter* column gives the number of maximal possible iterations possible in the training run. *miniter*, on the other hand, the minimal iterations a training has to do before checking on set termination conditions. The method is the optimizer, which was used for the training. The columns *best_f, best_df, best_dxnorm* are the best value of the loss function. In this case, the non-weighed loss function. *df* is the difference between the current loss function value and the value in the previous epoch. The *dxnorm* is the difference between the norms of the current parameter values and the norm of the parameter values in the previous epoch. *best_i, max_i* are the maximum number of iterations done during the training, and best_i marks the epoch on which the best minima were found. *f_rtol* is a termination condition based on the output's relative tolerance. *basis* and *ref_basis* are the names of the initial, the reference basis set.

1 Dimensional DFT Code using JAX

June 24, 2022

1 1 Dimensional DFT Code

- This code is bade on the code of the Yohei Tamura: https://github.com/tamuhey/python_1d_dft/blob/master/numpy_1ddft.ipynb

The goal is to adapt a one dimensinal DFT Code based on numpy, to the Jax framework done by google.

- Jax: https://github.com/google/jax
- Matrix mechanics (for further information) :
  - https://www.ph.tum.de/academics/bsc/break/2008s/fk_PH0007_03_course.pdf

For this we using the Hamiltonian:

$$\hat{H} = -\frac{1}{2}\frac{d}{dx} + v_{HA}(x) + v_x^{LDA} + x$$

- where v_x^{LDA} stands for *local density approximation*
- $v_{HA}(x)$ stands for the Coulomb Potential
- x^2 represents the harmonmic oscillator

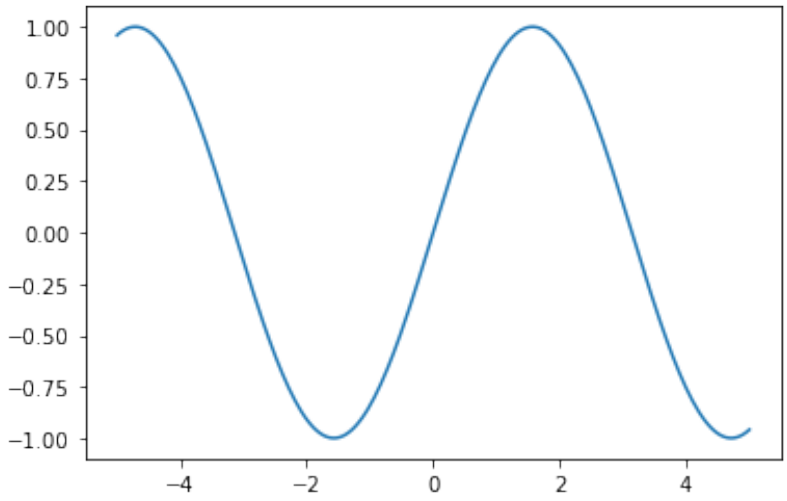
[1]:
```
from jax.config import config
config.update("jax_enable_x64", True)
config.update('jax_platform_name', 'cpu')
import numpy as np
import matplotlib.pyplot as plt
import jax.numpy as jnp
import jax.scipy as jsci
from jax import grad, jit, vmap
from jax.ops import index, index_add, index_update
from jax import random
from jax.tree_util import partial

%matplotlib inline
```

1.1 Try to realize a Differential operator

bevor starting use a $f(x) = sin(x)$ to cacualte first and second derivation of $f(x)$

[2]:
```
n_grid = 200
x = jnp.linspace(-5,5, n_grid, dtype=jnp.float64)
y = jnp.sin(x)
plt.plot(x,y)
```

[2]: `[<matplotlib.lines.Line2D at 0x7f91e82abdc0>]`



1.1.1 first derivation:

- in one dimension we can aproximate a differentiation by

$$(\frac{dy}{dx})_i = \frac{y_{i+1} - y_i}{h}$$

- so further the Matrix is:

$$D_{ij} = \frac{\delta_{i+1,j} - \delta_{i,j}}{h}$$

- the reason of a Matix is here that there are Elements in the Dimension of $D_{i+1,j}$ so
- we could write as follows:

$$(\frac{dy}{dx})_i = D_{ij} y_j$$

- The derivative may not be well defined at the end of the grid.
- δ_{ij} is Kronecker delta

- Einstein summation is used for last equation

[3]:
```
@jit
def diffOP_first(x):
    n_grid_loc = len(x)
    h=x[1]-x[0]
    delta_ip1_j = jnp.diagflat(jnp.ones(n_grid_loc-1),1) #create the Kronecker␣
 ↪i+1, j
    # jnp.diagflat Create a two-dimensional array with the flattened input as a␣
 ↪diagonal.
    #    np.diagflat([1,2], 1)
    #    ==>    array([[0, 1, 0],
    #                  [0, 0, 2],
    #                  [0, 0, 0]])'

    delta_ij = jnp.eye(n_grid_loc) #crate the diagonal Elements

    # 1 in diagonal else 0 if k= 0 else k shift used diagonal
    #>>> np.eye(3, k=1)
    #       array([[0.,  1.,  0.],
    #              [0.,  0.,  1.],
    #              [0.,  0.,  0.]])

    D = (delta_ip1_j -delta_ij)/ h #calc Differential operator D_ij
    return D

D = diffOP_first(x)
```

1.1.2 Second order differentiation

- In the same way as the first order:

$$D^2_{ij} = \frac{\delta_{i+1,j} - 2\delta_{i,j} + \delta_{i-1,j}}{h^2}$$

- This could be written with the first order D_{ij}, as follows (take care of the transpose):

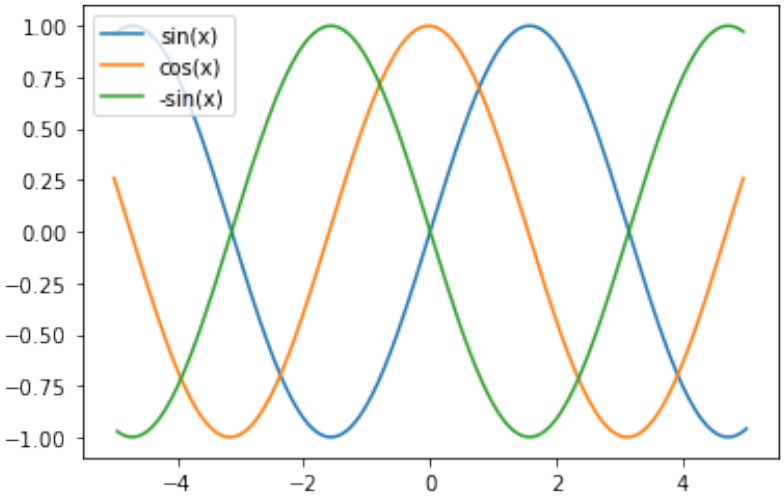
$$D^2_{ij} = -D_{ik}D_{jk}$$

- The derivative may not be well defined at the end of the grid.

[4]:
```
@jit
def diffOP_second(x, diffOP1 = False ):
    if diffOP1 is False:
        D = diffOP_first(x)
    else:
        D = diffOP1
    D2 = D.dot(-D.T)
```

```
    D2_new = index_update(D2, index[-1,-1], D2[0,0])
    return D2_new

D2 = diffOP_second(x, D)

diff_y = D.dot(y)
diff2_y = D2.dot(y)

plt.plot(x,y,label = 'sin(x)')
plt.plot(x[:-1],diff_y[:-1], label = 'cos(x)')
plt.plot(x[1:-1], diff2_y[1:-1] , label = '-sin(x)')
plt.legend(loc = "upper left")
```

[4]: <matplotlib.legend.Legend at 0x7f91e813cfa0>



2 Non-interacting electrons

- This is the Hamiltonian of non-interacting free particles in a box given by the size of grid:

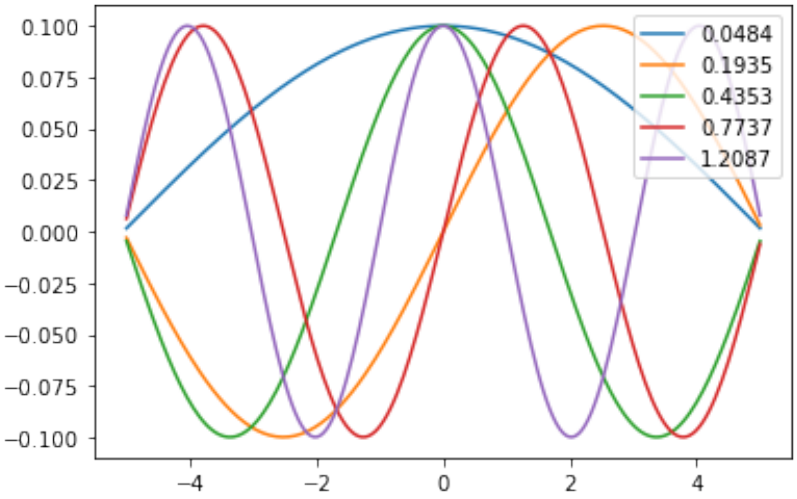
$$\hat{H} = \hat{T} = -\frac{1}{2}\frac{d^2}{dx^2}$$

- We could solve the KS equation by solving the eigenvalue problem
- using jsci.linalg.eigh() makes this pretty easy:

[5]:
```
eig_non_e, psi_non_e = jsci.linalg.eigh(-D2/2)
# eig_non_e is the eingenvalue vom H psi = E psi
```

```
# psi_non_e is the wavefunktion psi
```

[6]:
```
# ploted wavefunction (energies are shown in the label)
for i in range(5):
    plt.plot(x,psi_non_e[:,i], label=f"{eig_non_e[i]:.4f}")
    plt.legend(loc=1)
```



3 Harmonic oscillator

- include the external potential $v_{ext} = x^2$:

$$\hat{H} = \hat{T} = -\frac{1}{2}\frac{d^2}{dx^2} + x^2$$

- we can write the potential as a matrix X, as follows:

$$X = x^2 = \delta_{ij} x \cdot x$$

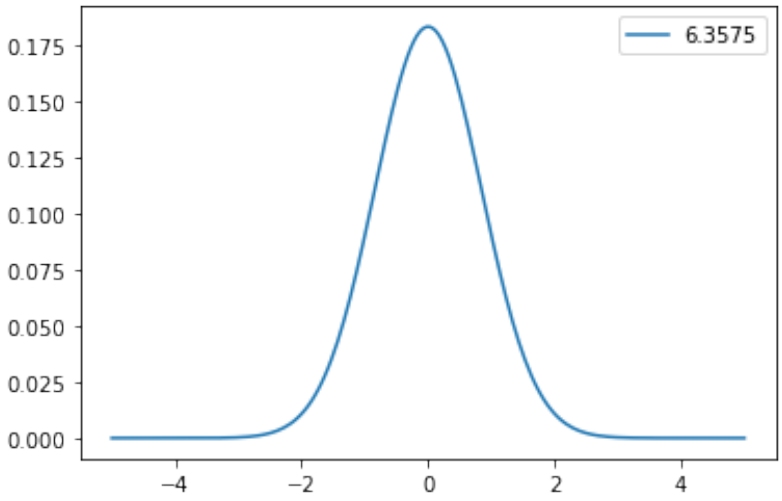
- after that we can again solfe the KS eq as eigenvalue Problem

[7]:
```
X = jnp.diagflat(x*x)
eig_harm, psi_harm = np.linalg.eigh(-D2/2+X)
```

[8]:
```
# ploted wavefunction (energies are shown in the label)

plt.plot(x,psi_harm[:,0], label=f"{eig_harm[i]:.4f}")
```

```
plt.legend(loc=1)
```

[8]: <matplotlib.legend.Legend at 0x7f91e8052c40>

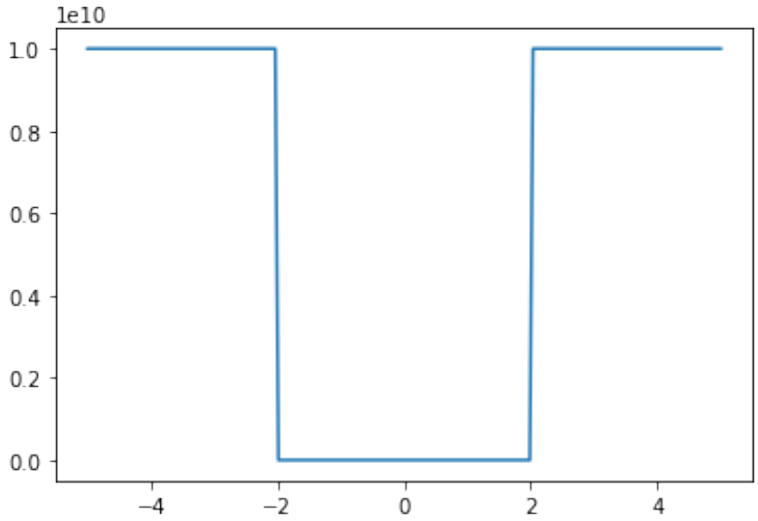


4 Well potential

(Potentialtopf)

[9]:
```
w_old =jnp.full_like(x,1.0e10) #creat array of len of array x
w = index_update(w_old, index[jnp.logical_and(x>-2,x<2)], 0.)
plt.plot(x, w)
```

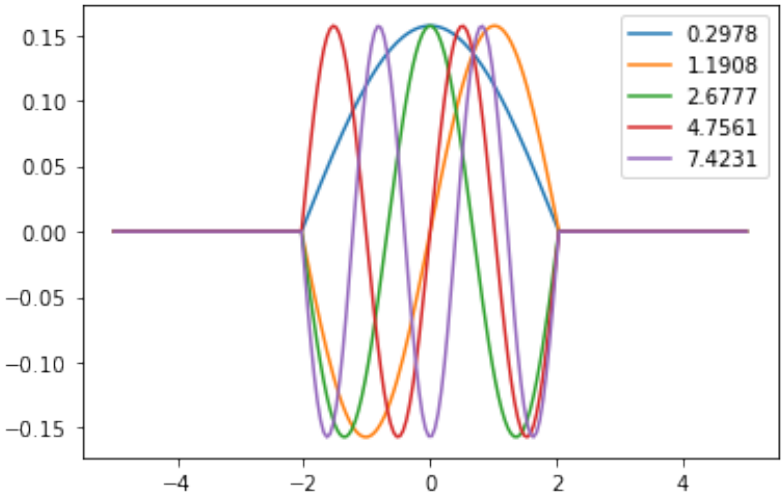
[9]: [<matplotlib.lines.Line2D at 0x7f91e0763b80>]



- Solve KS again adding the Well Potential but but without the harmonic part

[10]:
```
eig_well, psi_well= jsci.linalg.eigh(-D2/2+jnp.diagflat(w))
```

[11]:
```
# ploted wavefunction (energies are shown in the label)
for i in range(5):
    plt.plot(x,psi_well[:,i], label=f"{eig_well[i]:.4f}")
    plt.legend(loc=1)
```



5 Density

- We will want to include the Coulomb or Hatree interacion as well as LDA exchange
- Both of which are density functinals
- So we need to calculate the electron density
- Each state should be normalized:

$$\int |\psi|^2 dx = 1$$

- let f_n be occupation numbers, the density $n(x)$ can be written as follows:

$$n(x) = \sum_n f_n |\psi(x)|^2$$

- Note:
  - Each state fits up to two electrons: one with spin up, and one with spin down.
  - In DFT, we calculate the ground state.

[12]:
```
# integral
@jit
def integral(x,y):
    dx = x[1]- x[0]
    return jnp.sum(y*dx, axis = 0)
```

- number of electrons

[13]:
```
num_electrons = 17
```

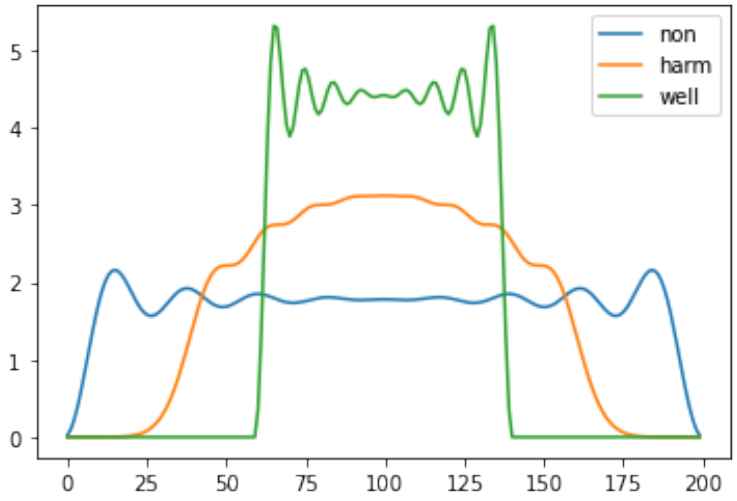
- density

[14]:
```
#@partial(jit , static_argnums = (1,))
@jit
def density(num_electron, psi , x):
    #norm the wave function:
    I = integral(x,psi**2)
    normed_psi = psi / jnp.sqrt(I)
    #follow the Hundschen rules to set up the 2 spins on the orbitals
    new_orb = False
    num_orb = num_electrons // 2

    if num_electrons % 2:
        new_orb = True

    fn_0 = jnp.full(num_orb + new_orb,2.)

    if new_orb == True:
        fn = index_update(fn_0, index[-1], 1)

    fnT = fn.reshape((-1,1))
    used_wavefunc = normed_psi.T[0:len(fn), :]

    def dens(orb, wavefunc):
        return orb * (wavefunc**2)


    return jnp.sum(dens(fnT, used_wavefunc), axis = 0)



%timeit density(num_electrons,psi_non_e, x)
```

```
62.9 µs ± 1.58 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

[15]:
```
plt.plot(density(num_electrons,psi_non_e, x), label="non")
plt.plot(density(num_electrons,psi_harm, x), label="harm")
plt.plot(density(num_electrons,psi_well, x), label="well")
plt.legend(loc=1)
```

[15]:
```
<matplotlib.legend.Legend at 0x7f91e0711ac0>
```



6 Exchange energy

- Consider the exchange functional in the LDA: (to remember *local density approximation*)
  - ignore the correlation for simplicity.

$$E_X^{LDA}[n] = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{1/3}\int n^{4/3}dx$$

- The potential is given by the derivative of the exchange energy with respect to the density:

$$v_X^{LDA}[n] = \frac{\partial E_X^{LDA}}{\partial n} = -\left(\frac{3}{\pi}\right)^{1/3} n^{1/3}$$

- the code is the following

[16]:
```
@jit
def get_exchange(nx,x):
    energy=-3./4.*(3./jnp.pi)**(1./3.)*integral(x,nx**(4./3.))
    potential=-(3./jnp.pi)**(1./3.)*nx**(1./3.)
    return energy, potential
```

7 coulomb potential

- Electrostatic energy or Hatree energy
- The expression of 3D-Hatree energy is not converged in 1D.

- Hence we cheat and use a modified as follows:

$$E_{Ha} = \frac{1}{2} \iint \frac{n(x)n(x')}{\sqrt{(x-x')^2+\varepsilon}} dx dx'$$

  - where ε is a small positive constant

- The potential is given by:

$$v_{Ha} = \int \frac{n(x')}{\sqrt{(x-x')^2+\varepsilon}} dx'$$

- In a matirx expression:

$$E_{Ha} = \frac{1}{2} \frac{n_i n_j h^2}{\sqrt{(x_i-x_j)^2+\varepsilon}}$$

$$v_{Ha,i} = \frac{n_j h}{\sqrt{(x_i-x_j)^2+\varepsilon}}$$

[17]:
```
@jit
def get_hatree(nx,x, eps=1e-1):
    h = x[1]-x[0]

    energy = jnp.sum(nx[None,:]*nx[:,None]*h**2/jnp.sqrt((x[None,:]-x[:
↪,None])**2+eps)/2)

    potential = jnp.sum(nx[None,:]*h/jnp.sqrt((x[None,:]-x[:
↪,None])**2+eps),axis=-1)

    return energy, potential
```

8 Solve the KS equation: Self-consistency loop

0. initialize the density (you can take an arbitrary constant) (beliebige constante kann verwendet werden)
1. Calculate the Exchange and Hatree potentials
2. Calculate the Hamiltonian
3. Calculate the wavefunctions and eigen values
4. If not converged, calculate the density and back to 1.

[18]:
```
def print_log(i,log):
    print(f"step: {i:<5} energy: {log['energy'][-1]:<10.4f} energy_diff:␣
↪{log['energy_diff'][-1]:.10f}")
```

[19]:
```
max_iter=1000
energy_tolerance=1e-5
log={"energy":[float("inf")], "energy_diff":[float("inf")]}
```

[20]:
```
@jit
def Energy_min(dens,x):
```

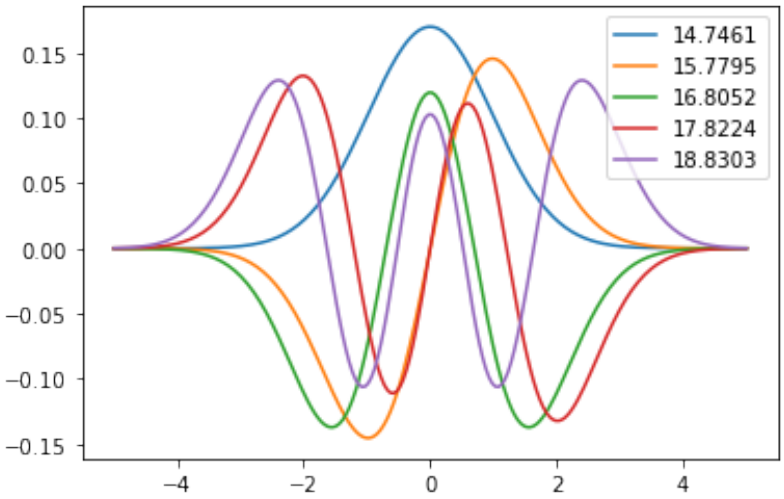
```
        ex_energy, ex_potential = get_exchange(dens, x)
        ha_energy, ha_potential = get_hatree(dens,x)

        # Hamiltonian
        H = -D2/2 + jnp.diagflat(ex_potential+ha_potential+x*x)

        energy, psi = jnp.linalg.eigh(H)
        return energy, psi
```

[21]:
```
dens=jnp.zeros(n_grid)

for i in range(max_iter):

    energy, psi = Energy_min(dens,x)

    # log
    log["energy"].append(energy[0])
    energy_diff = energy[0]-log["energy"][-2]
    log["energy_diff"].append(energy_diff)
    print_log(i,log)
        # convergence

    if np.abs(energy_diff) < energy_tolerance:
        print("converged!")
        break

    # update density
    dens = density(num_electrons,psi,x)
else:
    print("not converged")
```

```
step: 0     energy: 0.7069     energy_diff: -inf
step: 1     energy: 16.3625    energy_diff: 15.6555321919
step: 2     energy: 13.8021    energy_diff: -2.5603559494
step: 3     energy: 15.3002    energy_diff: 1.4980525863
step: 4     energy: 14.4119    energy_diff: -0.8882287680
step: 5     energy: 14.9470    energy_diff: 0.5350438262
step: 6     energy: 14.6242    energy_diff: -0.3228271880
step: 7     energy: 14.8201    energy_diff: 0.1959328656
step: 8     energy: 14.7011    energy_diff: -0.1190355457
step: 9     energy: 14.7735    energy_diff: 0.0724651058
step: 10    energy: 14.7294    energy_diff: -0.0441312736
step: 11    energy: 14.7563    energy_diff: 0.0268946713
step: 12    energy: 14.7399    energy_diff: -0.0163922405
step: 13    energy: 14.7499    energy_diff: 0.0099933983
step: 14    energy: 14.7438    energy_diff: -0.0060926001
step: 15    energy: 14.7475    energy_diff: 0.0037147279
step: 16    energy: 14.7452    energy_diff: -0.0022649307
```

```
step: 17    energy: 14.7466    energy_diff: 0.0013810031
step: 18    energy: 14.7458    energy_diff: -0.0008420446
step: 19    energy: 14.7463    energy_diff: 0.0005134280
step: 20    energy: 14.7460    energy_diff: -0.0003130574
step: 21    energy: 14.7462    energy_diff: 0.0001908842
step: 22    energy: 14.7461    energy_diff: -0.0001163900
step: 23    energy: 14.7461    energy_diff: 0.0000709679
step: 24    energy: 14.7461    energy_diff: -0.0000432721
step: 25    energy: 14.7461    energy_diff: 0.0000263849
step: 26    energy: 14.7461    energy_diff: -0.0000160880
step: 27    energy: 14.7461    energy_diff: 0.0000098095
converged!
```

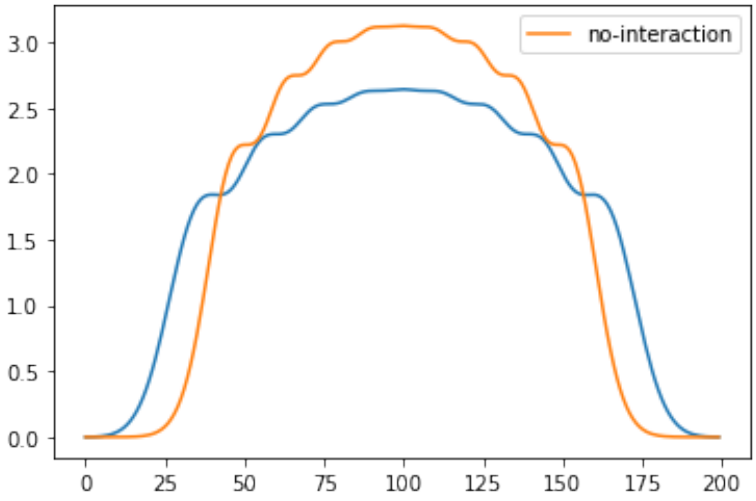
[22]:
```
for i in range(5):
    plt.plot(x,psi[:,i], label = f"{energy[i]:.4f}")
    plt.legend(loc = 1)
```



- compare the density to free particles

[23]:
```
plt.plot(dens)
plt.plot(density(num_electrons,psi_harm,x), label="no-interaction")
plt.legend()
```

[23]:
```
<matplotlib.legend.Legend at 0x7f91e0504a90>
```


no-interaction
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175
200

Declaration of authorship

I hereby declare that I have achieved the following module performance/partial module performance in the module:

with the title:

independently and only using the sources and aids indicated. The passages taken directly or indirectly from external sources are marked as such.

The work has not yet been submitted in the same or similar form to any other examining authority, and has not been published.

The digitized version of the work corresponds word for word with the version submitted in written form.

Name, first name: _____
Number of the matriculation: _____

Date: _____
Signed: _____

Acknowledgement

At this point, I want to thank the many people who make it possible for me to do this Thesis. I want to thank Professor Miguel A.L. Marques and Doctor Jonathan Schmidt, who proposed the project to me. They also proposed to me the Frameworks used in this Thesis. Espesscylie Jonathan Schmidt helped me a lot in evolving the Code. I also want to thank Ahmad W. Huran, who helped me a lot with every question due to the density functional theory and other chemistry-related questions. In addition, I want to thank every other member in the group of professor Miguel Marques for their valuable suggestions. Further on, I want to thank Susi Lehtola for his help in evolving the projection function and for suggestions for the training data. For his enormous help in writing and improving the Master Thesis itself, I want to thank Doctor Jürgen Henk. He was not only my second assessor but also massive support for me during the writing process.

I also want to thank my family, which supports me financially. Furthermore, I want to thank them for their trust and support over all these years. Further, I want to thank all my friends who supported me in the creation process of this Thesis and the entire study of physics. A special thanks should go to Matteo Tabusso. He is not only a close friend but also a companion on this Thesis as he does his Master Thesis in this group simultaneously with me. We often tried to help each other if there were any problems.

Last but not least, I want to thank the Institution of Martin-Luther-Universität Halle-Wittenberg for the opportunity to study physics and do this Thesis.