University of Chemistry and Technology, Prague

# Algorithms of Quantum Chemistry

*Author*
Tomáš Jíra, Jiří Janoš, Petr Slavíček

January 3, 2025

# Contents

# Part I.

# Electronic Structure Methods

This part provides an educational exploration into the computational techniques fundamental to understanding molecular electronic structure in quantum chemistry. Beginning with the Hartree–Fock (HF) method, the text introduces this foundational approach for determining molecular orbitals and electronic energies by approximating the interactions of electrons through a mean-field approximation. The HF method forms the basis for subsequent methods and is presented with a practical coding exercise that guides readers in implementing and calculating HF energies in Python.

Moving beyond HF, the text delves into Møller–Plesset Perturbation Theory (MPPT), which improves HF predictions by introducing corrections for electron correlation through a perturbative approach. This section includes exercises on calculating second- and third-order corrections, allowing readers to enhance their understanding of how electron interactions can be more accurately incorporated. Configuration Interaction (CI) theory is then presented as an approach for representing the molecular wavefunction as a combination of electron configurations. Here, readers learn the theoretical basis of CI and engage with practical examples focused on constructing the CI Hamiltonian matrix and solving for molecular energies, particularly emphasizing Full Configuration Interaction (FCI) for high accuracy in small systems.

The part culminates with a discussion on the Coupled Cluster (CC) theory, a highly accurate and computationally efficient method for capturing electron correlation effects, often used for small to medium-sized systems. By introducing truncations such as Coupled Cluster Doubles (CCD) and Coupled Cluster Singles and Doubles (CCSD), the text demonstrates how electron correlation can be systematically included while balancing computational cost. The CC section provides iterative coding exercises for calculating correlation energies, rounding out the document's comprehensive approach to electronic structure methods in computational quantum chemistry. Through this blend of theory, mathematical formulations, and hands-on coding exercises, the document serves as an invaluable resource for building a strong foundational understanding of electronic structure methods.

# 1. Hartree–Fock Method

The HF method is a foundational approach in quantum chemistry, aimed at solving the electronic structure problem in molecules by determining an optimal wavefunction. This method simplifies the complex interactions of electrons through a mean-field approximation, where each electron moves in an average field created by all others. This allows for the use of a single set of orbitals, leading to the construction of the Fock operator and iterative solutions to one-electron equations.

However, HF has notable limitations. Its reliance on a single-determinant wavefunction means it struggles to account for electron correlation. This leads to inaccuracies in energy predictions, particularly for systems with strong electron interactions, such as transition metal complexes or molecules with delocalized electrons.

## 1.1. Theoretical Background

Our primary objective is to solve the Schrödinger equation in the form

$$\hat{\mathbf{H}} \left|\Psi\right\rangle = E \left|\Psi\right\rangle \tag{1.1}$$

where $\hat{\mathbf{H}}$ denotes the molecular Hamiltonian operator, $|\Psi\rangle$ represents the molecular wavefunction, and $E$ is the total energy of the system. The HF approximates the total wavefunction $|\Psi\rangle$ as a single Slater determinant, expressed as

$$|\Psi\rangle = |\chi_1 \chi_2 \cdots \chi_N\rangle \tag{1.2}$$

where $\chi_i$ denotes a spin orbital, and $N$ is the total number of electrons. The goal of the HF method is to optimize these orbitals in order to minimize the system's total energy, thereby providing a reliable estimate of the electronic structure.

In the Restricted Hartree–Fock (RHF) method, we impose a constraint on electron spin, which allows us to work with spatial orbitals instead of spin orbitals. This reformulation expresses the Slater determinant in terms of spatial orbitals as

$$|\Psi\rangle = |\Phi_1 \Phi_2 \cdots \Phi_{N/2}\rangle \tag{1.3}$$

where $\Phi_i$ represents a spatial orbital. Notably, the RHF method requires an even number of electrons to satisfy spin-pairing. In practice, atomic orbitals (whether spin or spatial) are typically expanded in a set of basis functions $\{\phi_i\}$, which are often Gaussian functions, allowing for convenient computation with expansion coefficients. After performing some algebraic manipulations, the HF method leads to the Roothaan equations, which are expressed as

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon} \tag{1.4}$$

where $\mathbf{F}$ is the Fock matrix, $\mathbf{C}$ is the matrix of orbital coefficients, $\mathbf{S}$ is the overlap matrix, and $\boldsymbol{\varepsilon}$ represents the orbital energies. These matrices will be defined in detail later.

## 1.2. Implementation of the Restricted Hartree–Fock Method

To begin, we define the core Hamiltonian, also known as the one-electron Hamiltonian. This component of the full Hamiltonian excludes electron-electron repulsion and is expressed in index notation as

$$H_{\mu\nu}^{\text{core}} = T_{\mu\nu} + V_{\mu\nu} \tag{1.5}$$

where $\mu$ and $\nu$ are indices of the basis functions. Here, $T_{\mu\nu}$ represents a kinetic energy matrix element, while $V_{\mu\nu}$ denotes a potential energy matrix element. These matrix elements are defined as

$$T_{\mu\nu} = \langle \phi_\mu | \hat{T} | \phi_\nu \rangle \tag{1.6}$$
$$V_{\mu\nu} = \langle \phi_\mu | \hat{V} | \phi_\nu \rangle \tag{1.7}$$

Additionally, the overlap integrals, which describe the extent of overlap between basis functions, are given by

$$S_{\mu\nu} = \langle \phi_\mu | \phi_\nu \rangle \tag{1.8}$$

and another essential component are the two-electron repulsion integrals, defined as

$$J_{\mu\nu\kappa\lambda} = \langle \phi_\mu \phi_\mu | \hat{J} | \phi_\kappa \phi_\lambda \rangle \tag{1.9}$$

which play crucial roles in the HF calculation. All of these integrals over (Gausssian) basis functions are usually calculated using analytical expressions.[1] The solution to the Roothaan equations (1.4) requires an iterative procedure, since the Fock matrix defined as

$$F_{\mu\nu} = H_{\mu\nu}^{\text{core}} + D_{\kappa\lambda} \left( J_{\mu\nu\kappa\lambda} - \frac{1}{2} J_{\mu\lambda\kappa\nu} \right) \tag{1.10}$$

depends on the unknown density matrix $\mathbf{D}$, defined as

$$D_{\mu\nu} = 2 C_{\mu i} C_{\nu i} \tag{1.11}$$

This iterative procedure is executed through the Self-Consistent Field (SCF) method. An initial guess is made for the density matrix $\mathbf{D}$ (usually zero), the Roothaan equations (1.4) are solved and the density matix is updated using the equation (1.11). The total energy of the system is then calculated using the core Hamiltonian and the Fock matrix as

$$E = \frac{1}{2} D_{\mu\nu} (H_{\mu\nu}^{\text{core}} + F_{\mu\nu}) + E_{\text{nuc}} \tag{1.12}$$

After convergence of both the density matrix and total energy, the process concludes, yielding the optimized molecular orbitals. The total energy of the system also includes the nuclear repulsion energy, which is given by

$$E_{\text{nuc}} = \sum_A \sum_{B<A} \frac{Z_A Z_B}{R_{AB}} \tag{1.13}$$

where $Z_A$ is the nuclear charge of atom $A$, and $R_{AB}$ is the distance between atoms $A$ and $B$.

### 1.2.1. Direct Inversion in the Iterative Subspace

In the HF method, convergence of the density matrix and energy can be significantly accelerated by employing the Direct Inversion in the Iterative Subspace (DIIS) technique. DIIS achieves this by storing Fock matrices from previous iterations and constructing an optimized linear combination that minimizes the current iteration's error. This approach is especially valuable in HF calculations for large systems, where convergence issues are more common and challenging to resolve.

We start by defining the error vector of $i$-th iteration $\mathbf{e}_i$ as

$$\mathbf{e}_i = \mathbf{S}_i\mathbf{D}_i\mathbf{F}_i - \mathbf{F}_i\mathbf{D}_i\mathbf{S}_i \tag{1.14}$$

Our goal is to transform the Fock matrix $\mathbf{F}_i$ as

$$\mathbf{F}_i = \sum_{j=i-(L+1)}^{i-1} c_j\mathbf{F}_j \tag{1.15}$$

where $c_j$ are the coefficients that minimize the error matrix $\mathbf{e}_i$ and $L$ is the number of Fock matrices we store called the subspace size. To calculate the coefficients $c_j$, we solve the set linear equations

$$\begin{bmatrix} \mathbf{e}_1\cdot\mathbf{e}_1 & \dots & \mathbf{e}_1\cdot\mathbf{e}_{L+1} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{e}_{L+1}\cdot\mathbf{e}_1 & \dots & \mathbf{e}_{L+1}\cdot\mathbf{e}_{L+1} & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_{L+1} \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{1.16}$$

After solving the linear equations, we use the coefficients $c_j$ to construct the new Fock matrix $\mathbf{F}_i$ according to the equation (1.15) and proceed as usual with the HF calculation.

### 1.2.2. Gradient of the Restricted Hartree–Fock Method

If we perform the calculation as described above and get the density matrix $\mathbf{D}$ we can evaluate the nuclear energy gradient as[2]

$$\frac{\partial E}{\partial X_{A,i}} = \sum_{\mu\nu\in\{\phi_A\}} D_{\mu\nu}\frac{\partial H^{\text{core}}_{\mu\nu}}{\partial X_{A,i}} + 2\sum_{\mu\nu\in\{\phi_A\}} D_{\mu\nu}D_{\kappa\lambda}\frac{\partial\left(J_{\mu\nu\kappa\lambda} - \frac{1}{2}J_{\mu\lambda\kappa\nu}\right)}{\partial X_{A,i}} - 2W_{\mu\nu}\frac{\partial S_{\mu\nu}}{\partial X_{A,i}} \tag{1.17}$$

where $A$ is the index of an atom, $i$ is the index of the coordinate, $\{\phi_A\}$ is the set of all basis functions located at atom $A$ and $\mathbf{W}$ is energy weighed density matrix defined as

$$W_{\mu\nu} = 2C_{\mu i}C_{\nu i}\varepsilon_i \tag{1.18}$$

Keep in mind that the indices $\kappa$ and $\lambda$ in the gradient equation (1.17) are summed over all basis functions.

## 1.3. Integral Transforms to the Basis of Molecular Spinorbitals

To carry out most post-Hartree–Fock (post-HF) calculations, it is essential to transform the integrals into the Molecular Spinorbital (MS) basis. We will outline this transformation process here and refer to it in subsequent sections on post-HF methods. The post-HF methods in this document will be presented using the integrals in the MS basis (and in its antisymmetrized form for the two-electron integrals) as this approach is more general.

All the integrals defined in the equations (1.5), (1.8), and (1.9), as well as Fock matrix in the equation (1.10) are defined in the basis of atomic orbitals. To transform these integrals to the MS basis, we utilize the coefficient matrix $\mathbf{C}$ obtained from the solution of the Roothaan equations (1.4). This coefficient matrix $\mathbf{C}$ is initially calculated in the spatial molecular orbital basis (in the RHF calculation).

The first step involves expanding the coefficient matrix $\mathbf{C}$ to the MS basis. This transformation can be mathematically expressed using the tiling matrix $\mathbf{P}_{n\times 2n}$, defined as

$$\mathbf{P} = \begin{pmatrix} e_1 & e_1 & e_2 & e_2 & \dots & e_n & e_n \end{pmatrix} \tag{1.19}$$

where $e_i$ represents the $i$-th column of the identity matrix $\mathbf{I}_n$. Additionally, we define the matrices $\mathbf{M}_{n\times 2n}$ and $\mathbf{N}_{n\times 2n}$ with elements given by

$$M_{ij} = 1 - j \bmod 2, N_{ij} = j \bmod 2 \tag{1.20}$$

The coefficient matrix $\mathbf{C}$ in the MS basis can be then expressed as

$$\mathbf{C}^{\mathrm{MS}} = \begin{pmatrix} \mathbf{CP} \\ \mathbf{CP} \end{pmatrix} \odot \begin{pmatrix} \mathbf{M} \\ \mathbf{N} \end{pmatrix} \tag{1.21}$$

where $\odot$ denotes the Hadamard product. This transformed matrix $\mathbf{C}^{\mathrm{MS}}$ is subsequently used to transform the two-electron integrals $\mathbf{J}$ to the MS basis as

$$J^{\mathrm{MS}}_{pqrs} = C^{\mathrm{MS}}_{\mu p} C^{\mathrm{MS}}_{\nu q} (\mathbf{I}_2 \otimes_K (\mathbf{I}_2 \otimes_K \mathbf{J})^{(4,3,2,1)})_{\mu\nu\kappa\lambda} C^{\mathrm{MS}}_{\kappa r} C^{\mathrm{MS}}_{\lambda s} \tag{1.22}$$

where the superscript $(4,3,2,1)$ denotes the axes transposition and $\otimes_K$ is the Kronecker product. This notation accommodates the spin modifications and ensures adherence to quantum mechanical principles. We also define the antisymmetrized two-electron integrals in physicists' notation as

$$\langle pq||rs\rangle = (J^{\mathrm{MS}}_{pqrs} - J^{\mathrm{MS}}_{psrq})^{(1,3,2,4)} \tag{1.23}$$

For the transformation of the one-electron integrals such as the core Hamiltonian, the overlap matrix and also the Fock matrix, we use the formula

$$A^{\mathrm{MS}}_{pq} = C^{\mathrm{MS}}_{\mu p} (\mathbf{I}_2 \otimes_K \mathbf{A})_{\mu\nu} C^{\mathrm{MS}}_{\nu q} \tag{1.24}$$

where $\mathbf{A}$ is an arbitrary matrix of one-electron integrals. Since many post-HF methods rely on differences of orbital energies in the denominator, we define the tensors

$$\varepsilon^a_i = \varepsilon_i - \varepsilon_a \tag{1.25}$$

$$\varepsilon^{ab}_{ij} = \varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b \tag{1.26}$$

$$\varepsilon^{abc}_{ijk} = \varepsilon_i + \varepsilon_j + \varepsilon_k - \varepsilon_a - \varepsilon_b - \varepsilon_c \tag{1.27}$$

for convenience. These tensors enhance code readability and efficiency, making it easier to understand and work with the underlying mathematical framework. Here and also throughout the rest of the document, the indices $i$, $j$ and $k$ run over occupied orbitals, whereas the indices $a$, $b$ and $c$ run over virtual orbitals.

## 1.4. Hartree–Fock Method and Integral Transform Coding Exercise

This section provides Python code snippets for implementing the HF method and transforming integrals to the MS basis. The code uses the NumPy library for efficient numerical computations, and the exercises are designed to build familiarity with the HF method and the MS integral transformations.

Each exercise includes placeholders for you to fill. It is assumed that the variables `atoms`, `coords`, `S`, `H`, and `J` are defined before the exercises. These variables represent the atomic numbers, atomic coordinates, overlap matrix, core Hamiltonian, and two-electron integrals, respectively. These variables can typically be obtained from the output of a quantum chemistry software package. If you would like to focus solely on coding you can save the molecule file, overlap integrals, core Hamiltonian, and two-electron integrals in the STO-3G basis to the same directory as the exercise code and load the variables using the Listing 1.1 below. The `ATOM` variable is a dictionary that maps the atomic symbols to atomic numbers.

```
# get the atomic numbers and coordinates of all atoms
atoms = np.array([ATOM[line.split()[0]] for line in open("molecule.xyz").readlines()[2:]],
    dtype=int)
coords = np.array([line.split()[1:] for line in open("molecule.xyz").readlines()[2:]],
    dtype=float)

# convert to bohrs
coords *= 1.8897261254578281

# load the integrals from the files
H, S = np.loadtxt("H_AO.mat", skiprows=1), np.loadtxt("S_AO.mat", skiprows=1); J = np.
    loadtxt("J_AO.mat", skiprows=1).reshape(4 * [S.shape[1]])
```

**Listing 1.1:** Example loading of molecule and integrals over atomic basis functions into variables used throughout exercises. The snippet expects the molecule and integral files to b present in the same directory as the script.

With all the variables defined, you can proceed to the HF and integral transform exercises in the Listings 1.2 and 1.3 below.

```
"""
Here are defined some of the necessary variables. The variable "E_HF" stores the Hartree-
    Fock energy, while "E_HF_P" keeps track of the previous iteration's energy to monitor
    convergence. The "thresh" defines the convergence criteria for the calculation. The
    variables "nocc" and "nbf" represent the number of occupied orbitals and the number of
     basis functions, respectively. Initially, "E_HF" is set to zero and "E_HF_P" to one
    to trigger the start of the Self-Consistent Field (SCF) loop. Although you can rename
    these variables, it is important to note that certain sections of the code are
    tailored to these specific names.
"""
E_HF, E_HF_P, nocc, nbf, thresh = 0, 1, sum(atoms) // 2, S.shape[0], 1e-8

"""
These lines set up key components for our HF calculations. We initialize the density
    matrix as a zero matrix, and the coefficients start as an empty array. Although the
    coefficient matrix is computed within the while loop, it's defined outside to allow
    for its use in subsequent calculations, such as the MP energy computation. Similarly,
    the exchange tensor is accurately calculated here by transposing the Coulomb tensor.
    The "eps" vector, which contains the orbital energies, is also defined at this stage
    to facilitate access throughout the script. This setup ensures that all necessary
    variables are ready for iterative processing and further calculations beyond the SCF
    loop.
"""
```

```
K, F, D, C, eps = J.transpose(0, 3, 2, 1), np.zeros_like(S), np.zeros_like(S), np.
    zeros_like(S), np.array(nbf * [0])

"""
This while loop is the SCF loop. Please fill it so it calculates the Fock matrix, solves
    the Fock equations, builds the density matrix from the coefficients and calculates the
     energy. You can use all the variables defined above and all the functions in numpy
    package. The recommended functions are np.einsum and np.linalg.eigh. Part of the
    calculation will probably be calculation of the inverse square root of a matrix. The
    numpy package does not conatin a function for this. You can find a library that can do
     that or you can do it manually. The manual calculation is, of course, preferred.
"""
while abs(E_HF - E_HF_P) > thresh:
    break

"""
In the followng block of code, please calculate the nuclear-nuclear repulsion energy. You
    should use only the atoms and coords variables. The code can be as short as two lines.
     The result should be stored in the "VNN" variable.
"""
VNN = 0

# print the results
print("RHF ENERGY: {:.8f}".format(E_HF + VNN))
```

**Listing 1.2:** HF method exercise code. The important variables like number of occupied orbitals, convergence threshold and matrix containers are already defined. The student is expected to fill the SCF loop and calculate nuclear repulsion energy from the atomic numbers and coordinates. The total energy is then automatically printed.

```
"""
To perform most of the post-HF calculations, we need to transform the Coulomb integrals to
     the molecular spinorbital basis, so if you don't plan to calculate any post-HF
    methods, you can end the eercise here. The restricted MP2 calculation could be done
    using the Coulomb integral in MO basis, but for the sake of subsequent calculations,
    we enforce here the integrals in the MS basis. The first thing you will need for the
    transform is the coefficient matrix in the molecular spinorbital basis. To perform
    this transform using the mathematical formulation presented in the materials, the
    first step is to form the tiling matrix "P" which will be used to duplicate columns of
     a general matrix. Please define it here.
"""
P = np.zeros((nbf, 2 * nbf))

"""
Now, please define the spin masks "M" and "N". These masks will be used to zero out
    spinorbitals, that should be empty.
"""
M, N = np.zeros((nbf, 2 * nbf)), np.zeros((nbf, 2 * nbf))

"""
With the tiling matrix and spin masks defined, please transform the coefficient matrix
    into the molecular spinorbital basis. The resulting matrix should be stored in the "
    Cms" variable.
"""
Cms = np.zeros(2 * np.array(C.shape))

"""
For some of the post-HF calculations, we will also need the Hamiltonian and Fock matrix in
     the molecular spinorbital basis. Please transform it and store it in the "Hms" and "
    Fms" variable. If you don't plan to calculate the CCSD method, you can skip the
    transformation of the Fock matrix, as it is not needed for the MP2 and CI calculations
    .
"""
Hms, Fms = np.zeros(2 * np.array(H.shape)), np.zeros(2 * np.array(H.shape))
```

```
"""
With the coefficient matrix in the molecular spinorbital basis available, we can proceed
    to transform the Coulomb integrals. It is important to note that the transformed
    integrals will contain twice as many elements along each axis compared to their
    counterparts in the atomic orbital (AO) basis. This increase is due to the
    representation of both spin states in the molecular spinorbital basis.
"""
Jms = np.zeros(2 * np.array(J.shape))

"""
The post-HF calculations also require the antisymmetrized two-electron integrals in the
    molecular spinorbital basis. These integrals are essential for the MP2 and CC
    calculations. Please define the "Jmsa" tensor as the antisymmetrized two-electron
    integrals in the molecular spinorbital basis.
"""
Jmsa = np.zeros(2 * np.array(J.shape))

"""
As mentioned in the materials, it is also practical to define the tensors of reciprocal
    orbital energy differences in the molecular spinorbital basis. These tensors are
    essential for the MP2 and CC calculations. Please define the "Emss", "Emsd" and "Emst"
     tensors as tensors of single, double and triple excitation energies, respectively.
    The configuration interaction will not need these tensors, so you can skip this step
    if you don't plan to program the CI method. The MP methods will require only the "Emsd
    " tensor, while the CC method will need both tensors.
"""
Emss, Emsd = np.array([]), np.array([])
```

**Listing 1.3:** Integral transform exercise code. The tiling matrices are predefined here with a correct shape, but the student is expected to fill the with the correct expressions. After that, the student should transform the coefficient matrix, core Hamoltonian, Fock matrix and two-electron integrals to the MS basis. Additionally, orbital energy tensor are also expected to be calculated.

Solution to this exercises can be found in Section A.1 and in Section A.2.

# 2. Møller–Plesset Perturbation Theory

MPPT is a quantum mechanical method used to improve the accuracy of electronic structure calculations within the framework of HF theory. It involves treating electron-electron correlation effects as a perturbation to the reference HF wave function. The method is named after its developers, physicists C. Møller and M. S. Plesset. By systematically including higher-order corrections, MPPT provides more accurate predictions of molecular properties compared to the initial HF approximation.

## 2.1. Theory of the Perturbative Approach

As for the HF method, we start with the Schrödinger equation in the form

$$\hat{\mathbf{H}} |\Psi\rangle = E |\Psi\rangle \tag{2.1}$$

where $\hat{\mathbf{H}}$ is the molecular Hamiltonian operator, $|\Psi\rangle$ is the molecular wave function, and $E$ is the total energy of the system. In the Møller–Plesset perturbation theory we write the Hamiltonian operator as

$$\hat{\mathbf{H}} = \hat{\mathbf{H}}^{(0)} + \lambda \hat{\mathbf{H}}^{'} \tag{2.2}$$

where $\hat{\mathbf{H}}^{(0)}$ is the Hamiltonian used in the HF method (representing electrons moving in the mean field), $\lambda$ is a parameter between 0 and 1, and $\hat{\mathbf{H}}^{'}$ is the perturbation operator representing the missing electron-electron interactions not included in the HF approximation. We then expand the wavefunction $|\Psi\rangle$ and total energy $E$ as a power series in $\lambda$ as

$$|\Psi\rangle = |\Psi^{(0)}\rangle + \lambda |\Psi^{(1)}\rangle + \lambda^2 |\Psi^{(2)}\rangle + \dots \tag{2.3}$$

$$E = E^{(0)} + \lambda E^{(1)} + \lambda^2 E^{(2)} + \dots \tag{2.4}$$

and ask, how how does the total energy change with the included terms. After some algebra, we can show that the first order correction to the total energy is zero, the second order correction is given by

$$E_{\text{corr}}^{\text{MP2}} = \sum_{s>0} \frac{H_{0s}^{'} H_{s0}^{'}}{E_0 - E_s} \tag{2.5}$$

where $s$ runs over all doubly excited determinants, $H_{0s}^{'}$ is the matrix element of the perturbation operator between the HF determinant and the doubly excited determinant, and $E_0$ and $E_s$ are the energies of the reference and doubly excited determinants, respectively.[3, 4] We could express all higher-order corrections in a similar way, using only the matrix elements of the perturbation operator and the energies of the determinants. For practical calculations, we apply Slater-Condon rules to evaluate the matrix elements and use the orbital energies obtained from the Hartree-Fock calculation. The expressions for calculation are summarised below.

## 2.2. Implementation of 2nd and 3rd Order Corrections

Having the antisymmetrized two-electron integrals in the MS basis and physicists' notation defined in Section 1.3, we can now proceed with the calculation of the correlation energy. The 2nd order correlation energy can be expressed as

$$E_{\text{corr}}^{\text{MP2}} = \frac{1}{4} \sum_{ijab} \frac{\langle ab||ij\rangle \langle ij||ab\rangle}{\varepsilon_{ij}^{ab}} \tag{2.6}$$

and the 3rd order correlation energy as

$$\begin{aligned} E_{\text{corr}}^{\text{MP3}} = & \frac{1}{8} \sum_{ijab} \frac{\langle ab||ij\rangle \langle cd||ab\rangle \langle ij||cd\rangle}{\varepsilon_{ij}^{ab} \varepsilon_{ij}^{cd}} + \\ & + \frac{1}{8} \sum_{ijab} \frac{\langle ab||ij\rangle \langle ij||kl\rangle \langle kl||ab\rangle}{\varepsilon_{ij}^{ab} \varepsilon_{kl}^{ab}} + \\ & + \sum_{ijab} \frac{\langle ab||ij\rangle \langle cj||kb\rangle \langle ik||ac\rangle}{\varepsilon_{ij}^{ab} \varepsilon_{ik}^{ac}} \end{aligned} \tag{2.7}$$

To calculate the 4th order correction, we would need to write 39 terms, which is not practical. Higher-order corrections are usually not programmed this way, instead, the diagrammatic approach is used.[4–6]

## 2.3. 2nd and 3rd Order Corrections Code Exercise

Similar to the HF method, Møller–Plesset perturbation theory can also be implemented in Python. The code exercise below provides a self-contained guide to calculating the Møller–Plesset Perturbation Theory of 2nd Order (MP2) and Møller–Plesset Perturbation Theory of 3rd Order (MP3) correlation energies. This exercise is designed to be appended to your existing HF implementation, as the MP2 and MP3 methods build on the results of the HF procedure. You can access the foundational HF method and integral transformation coding exercise in Section 1.4. The exercise is provided in the Listing 2.1 below

```
"""
Since we have everything we need for the MP calculations, we can now calculate the MP2
    correlation energy. The result should be stored in the "E_MP2" variable.
"""
E_MP2 = 0

"""
Let's not stop here. We can calculate MP3 correlation energy as well. Please calculate it
    and store it in the "E_MP3" variable.
"""
E_MP3 = 0

# print the results
print("MP2 ENERGY: {:.8f}".format(E_HF + E_MP2 +       + VNN))
print("MP3 ENERGY: {:.8f}".format(E_HF + E_MP2 + E_MP3 + VNN))
```

**Listing 2.1:** MP2 and MP3 exercise code. The placeholders for the energies are initialized to zero. If you transformed all the necessary integrals in the previous exercise, you should be able to fill the placeholders with correct expressions.

Solution to this exercise can be found in Section A.3.

# 3. Configuration Interaction

CI is a post-HF, utilizing a linear variational approach to address the nonrelativistic Schrödinger equation under the Born–Oppenheimer approximation for multi-electron quantum systems. CI mathematically represents the wave function as a linear combination of Slater determinants. The term "configuration" refers to different ways electrons can occupy orbitals, while "interaction" denotes the mixing of these electronic configurations or states. CI computations, however, are resource-intensive, requiring significant CPU time and memory, limiting their application to smaller molecular systems. While FCI considers all possible electronic configurations, making it computationally prohibitive for larger systems, truncated versions like Configuration Interaction Singles and Doubles (CISD) or Configuration Interaction Singles, Doubles and Triples (CISDT) are more feasible and commonly employed in quantum chemistry studies.

## 3.1. Theoretical Background of General Configuration Interaction

In CI theory, we expand the wavefunction $|\Psi\rangle$ in terms of the HF reference determinant and its excited configurations as

$$|\Psi\rangle = c_0 |\Psi_0\rangle + \left(\frac{1}{1!}\right)^2 c_i^a |\Psi_i^a\rangle + \left(\frac{1}{2!}\right)^2 c_{ij}^{ab} |\Psi_{ij}^{ab}\rangle + \left(\frac{1}{3!}\right)^2 c_{ijk}^{abc} |\Psi_{ijk}^{abc}\rangle + \dots \tag{3.1}$$

where we seek the coefficients $\mathbf{c}$ that minimize the energy. To determine these coefficients, we construct and diagonalize the Hamiltonian matrix in the basis of these excited determinants. The CI Hamiltonian matrix $\mathbf{H}^{\mathrm{CI}}$ is represented as

$$\mathbf{H}^{\mathrm{CI}} = \begin{bmatrix} \langle\Psi_0|\hat{H}|\Psi_0\rangle & \langle\Psi_0|\hat{H}|\Psi_i^a\rangle & \langle\Psi_0|\hat{H}|\Psi_{ij}^{ab}\rangle & \dots \\ \langle\Psi_i^a|\hat{H}|\Psi_0\rangle & \langle\Psi_i^a|\hat{H}|\Psi_i^a\rangle & \langle\Psi_i^a|\hat{H}|\Psi_{ij}^{ab}\rangle & \dots \\ \langle\Psi_{ia}^{jb}|\hat{H}|\Psi_0\rangle & \langle\Psi_{ia}^{jb}|\hat{H}|\Psi_1\rangle & \langle\Psi_{ia}^{jb}|\hat{H}|\Psi_{ia}^{jb}\rangle & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.2}$$

After the Hamiltonian matrix is constructed, we solve the eigenvalue problem

$$\mathbf{H}^{\mathrm{CI}}\mathbf{C}^{\mathrm{CI}} = \mathbf{C}^{\mathrm{CI}}\boldsymbol{\varepsilon}^{\mathrm{CI}} \tag{3.3}$$

where $\mathbf{C}^{\mathrm{CI}}$ is a matrix of coefficients and $\boldsymbol{\varepsilon}^{\mathrm{CI}}$ is a diagonal matrix of eigenvalues. The lowest eigenvalue gives the ground-state energy, and the corresponding eigenvector provides the coefficients that minimize the energy. The elements of the CI Hamiltonian matrix are computed using the Slater–Condon rules, summarized in one function as

$$\mathbf{H}_{ij}^{\mathrm{CI}} = \begin{cases} \sum_k H_{kk}^{\mathrm{core,MS}} + \frac{1}{2}\sum_k \sum_l \langle kl||kl\rangle & D_i = D_j \\ H_{pr}^{\mathrm{core,MS}} + \sum_k \langle pk||lk\rangle & D_i = \{\cdots p\cdots\} \wedge D_j = \{\cdots r\cdots\} \\ \langle pq||rs\rangle & D_i = \{\cdots p\cdots q\cdots\} \wedge D_j = \{\cdots r\cdots s\cdots\} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where $D_i$ and $D_j$ are Slater determinants, $\mathbf{H}^{\text{core,MS}}$ is the core Hamiltonian in the MS basis, and $\langle pk||lk\rangle$ are the antisymmetrized two-electron integrals in MS basis and physicists' notation. The sums extend over all spinorbitals common between the two determinants. These integrals were previously transformed in Section 1.3. Keep in mind, that to apply the Slater-Condon rules, the determinants must be aligned, and the sign of the matrix elements must be adjusted accordingly, based on the number of permutations needed to align the determinants.

An important caveat in CI theory is its lack of size-extensivity, which implies that the energy does not scale linearly with the number of electrons. This drawback stems from the fact that the CI wavefunction is not size-consistent, meaning the energy of a combined system is not simply the sum of the energies of its isolated parts. This limitation restricts the application of CI mainly to small molecular systems.

## 3.2. Full Configuration Interaction Implementation

In FCI, we aim to account for all possible electronic configurations within a chosen basis set, offering the most accurate wavefunction representation for the given basis. Although this method yields highly precise electronic structure information, it is computationally intensive. Its cost scales exponentially with both the number of electrons and basis functions, limiting its feasibility to smaller systems.

The FCI process involves constructing all possible Slater determinants for a system. For simplicity, we'll assume that we want to include both singlet and triplet states in our determinant space. The total number of these determinants $N_D$ can be calculated using binomial coefficients

$$N_D = \binom{n}{k} \tag{3.5}$$

where $k$ is the total number of electrons, and $n$ is the total number of spinorbitals. For practical representation, it's useful to describe determinants as arrays of numbers, where each number corresponds to the index of an occupied orbitals. For example, the ground state determinant for a system with 6 electrons can be represented as $\{0, 1, 2, 3, 4, 5\}$, whereas the determinant $\{0, 1, 2, 3, 4, 6\}$ represents an excited state with one electron excited from orbital 5 to orbital 6. Using the determinants, the CI Hamiltonian matrix (3.2) can be constructed, and the eigenvalue problem (3.3) can be solved to obtain the ground and excited state energies.

## 3.3. Full Configuration Interaction Code Exercise

The FCI example builds on the HF method and demonstrates how to implement a FCI calculation in Python using NumPy. This exercise focuses on generating determinants, constructing the CI Hamiltonian matrix using Slater–Condon rules, and solving the eigenvalue problem to obtain the ground state energy. The code is designed for educational purposes and is based on prior HF results that can be done in Section 1.4. The exercise is provided in the Listing 3.1 below.

```
"""
Since we already calculated the necessary integrals in the MS basis, we can proceed. The
    next step involves generating determinants. We will store these in a simple list, with
     each determinant represented by an array of numbers, where each number corresponds to
     an occupied spinorbital. Since we are programming for Full Configuration Interaction
    (FCI), we aim to generate all possible determinants. However, should we decide to
    implement methods like CIS, CID, or CISD, we could easily limit the number of
    excitations. It is important to remember that for all CI methods, the rest of the code
     remains unchanged. The only difference lies in the determinants used. Don't
    overcomplicate this. Generating all possible determinants can be efficiently achieved
    using a simple list comprehension. I recommend employing the combinations function
    from the itertools package to facilitate this task.
```

```
"""
dets = list()

"""
Now, for your convenince, I define here the CI Hamiltonian.
"""
Hci = np.zeros([len(dets), len(dets)])

"""
Before we begin constructing the Hamiltonian, I recommend defining the Slater-Condon rules
    . Let's consider that the input for these functions will be an array of spinorbitals,
    segmented into unique and common ones. A practical approach might be to arrange this 1
    D array with all unique spinorbitals at the front, followed by the common spinorbitals
    . This arrangement allows you to easily determine the number of unique spinorbitals
    based on the rule being applied, meaning you will always know how many entries at the
    beginning of the array are unique spinorbitals. While you can develop your own method
    for managing this array, I will proceed under the assumption that the Slater-Condon
    rules we use will take a single array of spinorbitals and return an unsigned matrix
    element. The sign of this element will be corrected later in the script. For
    simplicity and flexibility, I'll define these rules using lambda functions, but you're
     welcome to expand them into full functions if you prefer.
"""
slater0 = lambda so: 0
slater1 = lambda so: 0
slater2 = lambda so: 0

"""
We can now proceed to filling the CI Hamiltonian. The loop is simple.
"""
for i in range(Hci.shape[0]):
    for j in range(Hci.shape[1]):

        """
        The challenging part of this process is aligning the determinants. In this step, I
             transfer the contents of the j-th determinant into the "aligned" determinant.
             It's important not to alter the j-th determinant directly within its original
             place, as doing so could disrupt the computation of other matrix elements.
            Instead, we carry out the next steps on the determinant now contained in the "
            aligned" variable. Additionally, the element sign is defined at this stage.
            You probably want to leave this unchanged.
        """
        aligned, sign = dets[j].copy(), 1

        """
        Now it's your turn. Please adjust the "aligned" determinant to match the i-th
            determinant as closely as possible. By "align", I mean you should execute a
            series of spinorbital swaps to minimize the differences between the "aligned"
            and the i-th determinant. It's also important to monitor the number of swaps
            you make, as each swap affects the sign of the determinant, hence the reason
            for the "sign" variable defined earlier. This task is not straightforward, so
            don't hesitate to reach out to the authors if you need guidance.
        """
        aligned = aligned

        """
        After aligning, we end up with two matched determinants: "aligned" and "dets[i]".
            At this point, we can apply the Slater-Condon rules. I suggested earlier that
            the input for these rules should be an array combining both unique and common
            spinorbitals. You can prepare this array now. However, if you've designed your
             Slater-Condon rules to directly accept the determinants instead, you can skip
             this preparatory step.
        """
        so = list()

        """
        Now, you'll need to assign the matrix element. Start by determining the number of
```

```
            differences between the two determinants. Based on this number, apply the
            corresponding Slater-Condon rule. Don't forget to multiply the result by the
            sign to account for any changes due to swaps made during the alignment of the
            determinants.
        """
        H[i, j] = 0

"""
You can finally solve the eigenvalue problem. Please, assign the correlation energy to the
    "E_FCI" variable.
"""
E_FCI = 0

# print the results
print("FCI ENERGY: {:.8f}".format(E_HF + E_FCI + VNN))
```

**Listing 3.1:** CI exercise code. Here, student is expected to calculate the CI ground state energy. The task here is to fill the CI Hamiltonian using the Slater–Condon rules and diagonalize it.

Solution to this exercise can be found in Section A.4.

# 4. Coupled Cluster Theory

CC theory is a post-HF method used in quantum chemistry to achieve highly accurate solutions to the electronic Schrödinger equation, particularly for ground states and certain excited states. It improves upon HF by incorporating electron correlation effects through a systematic inclusion of excitations (singles, doubles, triples, etc.) from a reference wavefunction, usually the HF wavefunction. The method uses an exponential ansatz to account for these excitations, leading to a size-consistent and size-extensive approach, making it one of the most accurate methods available for small to medium-sized molecular systems.

Within CC theory, specific truncations are often applied to manage computational cost. The CCD method considers only double excitations, capturing electron correlation more effectively than simpler methods like HF, but at a lower computational expense than higher-level methods. CCSD extends this approach by including both single and double excitations, offering greater accuracy, particularly for systems where single excitations play a significant role. CCSD is widely used due to its balance between accuracy and computational feasibility, making it a reliable choice for many chemical systems. ## CC Formalism

In the CC formalism, we write the total wavefunction in an exponential form as

$$|\Psi\rangle = e^{\hat{\mathbf{T}}} |\Psi_0\rangle \tag{4.1}$$

where $|\Psi_0\rangle$ is the reference wavefunction, usually the HF wavefunction, and $\hat{T}$ is the cluster operator that generates excitations from the reference wavefunction. The cluster operator is defined as

$$\hat{\mathbf{T}} = \hat{\mathbf{T}}_1 + \hat{\mathbf{T}}_2 + \hat{\mathbf{T}}_3 + \dots \tag{4.2}$$

where $\hat{\mathbf{T}}_1$ generates single excitations, $\hat{\mathbf{T}}_2$ generates double excitations, and so on. For example

$$\hat{\mathbf{T}}_1 |\Psi_0\rangle = \left(\frac{1}{1!}\right)^2 t_i^a |\Psi_i^a\rangle \tag{4.3}$$

where $t_i^a$ are the single excitation amplitudes. These amplitudes are just expansion coefficients that determine the contribution of each excitation to the total wavefunction. In the context of configuration interaction, we denoted these coefficients as $c_i^a$. Now that we have the total wavefunction, we want to solve the Schrödinger equation

$$\hat{\mathbf{H}} |\Psi\rangle = E |\Psi\rangle \tag{4.4}$$

where $\hat{H}$ is the molecular Hamiltonian operator, $E$ is the total energy of the system, and $|\Psi\rangle$ is the total wavefunction. In the CC theory, we usually rewrite the Schrödinger equation in the exponential form as

$$e^{-\hat{\mathbf{T}}} \hat{\mathbf{H}} e^{\hat{\mathbf{T}}} |\Psi_0\rangle = E |\Psi_0\rangle \tag{4.5}$$

because we can then express the CC energy as

$$E = \langle \Psi_0 | e^{-\hat{\mathbf{T}}} \hat{\mathbf{H}} e^{\hat{\mathbf{T}}} | \Psi_0 \rangle \tag{4.6}$$

taking advantage of the exponential form of the wavefunction. We could then proceed to express the total energy for various CC methods like CCD and CCSD, but the equations would be quite lengthy. Instead, we will leave the theory here and proceed to the actual calculations. One thing to keep in mind is that the CC equations are nonlinear and require iterative solution methods to obtain the final amplitudes.

## 4.1. Implementation of Truncated Coupled Cluster Methods

We will not go into the details here, but we will provide the final expressions for the CCD and CCSD methods.[7] The CCD and CCSD methods are the most commonly used CC methods, and they are often used as benchmarks for other methods. All we need for the evaluation of the expressions below are the two-electron integrals in the MS basis and physicists' notation, Fock matrix in the MS basis and the orbital energy tensors obtained from the HF calculation. All these transformations are already explained in Section 1.3. The expressions for the CCD can be written as

$$E_{\mathrm{CCD}} = \frac{1}{4} \langle ij||ab\rangle \, t_{ij}^{ab} \tag{4.7}$$

where the double excitation amplitudes $t_{ij}^{ab}$ are determined by solving the CCD amplitude equation. The CCD amplitude equations are given by

$$\begin{aligned} t_{ij}^{ab} = &\langle ab||ij\rangle + \frac{1}{2} \langle ab||cd\rangle \, t_{cd}^{ij} + \frac{1}{2} \langle kl||ij\rangle \, t_{ab}^{kl} + \hat{P}_{(a/b)}\hat{P}_{(i/j)} \langle ak||ic\rangle \, t_{cb}^{ij} - \\ &- \frac{1}{2}\hat{P}_{(a/b)} \langle kl||cd\rangle \, t_{ac}^{ij} t_{bd}^{kl} - \frac{1}{2}\hat{P}_{(i/j)} \langle kl||cd\rangle \, t_{ab}^{ik} t_{cd}^{jl} + \\ &+ \frac{1}{4} \langle kl||cd\rangle \, t_{cd}^{ij} t_{ab}^{kl} + \hat{P}_{(i/j)} \langle kl||cd\rangle \, t_{ac}^{ik} t_{bd}^{jl} \end{aligned} \tag{4.8}$$

where $\hat{P}_{(a/b)}$ and $\hat{P}_{(i/j)}$ are permutation operators that ensure the correct antisymmetry of the amplitudes. The CCSD energy expression is given by

$$E_{\mathrm{CCSD}} = F_{ia}^{\mathrm{MS}} t_a^i + \frac{1}{4} \langle ij||ab\rangle \, t_{ij}^{ab} + \frac{1}{2} \langle ij||ab\rangle \, t_i^a t_b^j \tag{4.9}$$

where the single and double excitation amplitudes $t_a^i$ and $t_{ij}^{ab}$ are determined by solving the CCSD amplitude equations. To simplify the notation a little bit, we define the the $\mathscr{F}$ and $\mathscr{W}$ intermediates as

$$\mathscr{F}_{ae} = (1 - \delta_{ae}) \, F_{ae} - \frac{1}{2} F_{me} t_m^a + t_m^f \langle ma||fe\rangle - \frac{1}{2} \tilde{\tau}_{mn}^{af} \langle mn||ef\rangle \tag{4.10}$$

$$\mathscr{F}_{mi} = (1 - \delta_{mi}) \, F_{mi} + \frac{1}{2} F_{me} t_i^e + t_n^e \langle mn||ie\rangle + \frac{1}{2} \tilde{\tau}_{in}^{ef} \langle mn||ef\rangle \tag{4.11}$$

$$\mathscr{F}_{me} = F_{me} + t_n^f \langle mn||ef\rangle \tag{4.12}$$

$$\mathscr{W}_{mnij} = \langle mn||ij\rangle + \hat{P}_{(i/j)} t_j^e \langle mn||ie\rangle + \frac{1}{4} \tau_{ij}^{ef} \langle mn||ef\rangle \tag{4.13}$$

$$\mathscr{W}_{abef} = \langle ab||ef\rangle - \hat{P}_{(a/b)} t_m^b \langle am||ef\rangle + \frac{1}{4} \tau_{mn}^{ab} \langle mn||ef\rangle \tag{4.14}$$

$$\mathscr{W}_{mbej} = \langle mb||ej\rangle + t_j^f \langle mb||ef\rangle - t_n^b \langle mn||ej\rangle - \left(\frac{1}{2} t_{jn}^{fb} + t_j^f t_n^b\right) \langle mn||ef\rangle \tag{4.15}$$

and two-particle excitation operators as

$$\tilde{\tau}_{ij}^{ab} = t_{ij}^{ab} + \frac{1}{2}\left(t_i^a t_j^b - t_i^b t_j^a\right) \tag{4.16}$$

$$\tau_{ij}^{ab} = t_{ij}^{ab} + t_i^a t_j^b - t_i^b t_j^a \tag{4.17}$$

The CCSD single excitations amplitude equations are then given by

$$t_i^a = F_{ai}^{\mathrm{MS}} + t_i^e \mathscr{F}_{ae} - t_m^a \mathscr{F}_{mi} t_{im}^{ae} \mathscr{F}_{me} - t_n^f \langle na||if\rangle - -\frac{1}{2} t_{im}^{ef} \langle ma||ef\rangle - \\ - \frac{1}{2} t_{mn}^{ae} \langle nm||ei\rangle \tag{4.18}$$

and the CCSD double excitations amplitude equations are given by

$$t_{ij}^{ab} = \langle ab||ij\rangle + \hat{P}_{(a/b)} t_{ij}^{ae}\left(\mathscr{F}_{be} - \frac{1}{2} t_m^b \mathscr{F}_{ae}\right) - \hat{P}_{(i/j)} t_{im}^{ab}\left(\mathscr{F}_{mi} + \frac{1}{2} t_j^e \mathscr{F}_{me}\right) + \\ + \frac{1}{2}\tau_{mn}^{ab} \mathscr{W}_{mnij} + \frac{1}{2}\tau_{ij}^{ef} \mathscr{W}_{abef} + \hat{P}_{(i/j)}\hat{P}_{(a/b)}\left(t_{im}^{ae} \mathscr{W}_{mbej} - t_i^e t_m^a \langle mb||ej\rangle\right) + \\ + \hat{P}_{(i/j)} t_i^e \langle ab||ej\rangle - \hat{P}_{(a/b)} t_m^a \langle mb||ij\rangle \tag{4.19}$$

The CCSD amplitude equations are, again, nonlinear and require iterative solution methods to obtain the final amplitudes. The initial guess for the amplitudes is often set to zero, and the equations are solved iteratively until convergence is achieved.

## 4.2. Coupled Cluster Singles and Doubles Code Exercise

After completing the HF implementation in Section 1.4, you can proceed with coding the CCSD exercise, which builds on the HF results. The exercise is provided in the Listing 4.1 below.

```
"""
We also have everything we need for the CC calculations. In this exercise, we will
    calculate the CCSD energy. Since the calculation will be iterative, I define here the
    CCSD energy as zero, the "E_CCSD_P" variable will be used to monitor convergence.
"""
E_CCSD, E_CCSD_P = 0, 1

"""
The first step of the calculation is to define the "t1" and "t2" amplitudes. These arrays
    can be initialized as zero arrays with the appropriate dimensions. I will leave this
    task to you.
"""
t1, t2 = np.array([]), np.array([])

"""
Now for the more complicated part. The CCSD calculation is iterative, and the convergence
    criterion is set by the "thresh" variable. The while loop should be filled with the
    appropriate calculations. The calculation of the "t1" and "t2" amplitudes is the most
    challenging part of the CCSD calculation. After convergence, the "E_CCSD" variable
    should store the final CCSD energy.
"""
while abs(E_CCSD - E_CCSD_P) > thresh:
    break

# print the CCSD energy
print("CCSD ENERGY: {:.8f}".format(E_HF + E_CCSD + VNN))
```

**Listing 4.1:** CCSD exercise code. The energy and amplitudes are initialized with default values. The student is expected to fill the loop for the calculation of the excitation amplitudes and ground state energy. After the self-consistency is achieved the result is automatically printed.

Solution to this exercise can be found in Section A.5.

# Appendices

# A. Code Solutions

This section provides the solutions to all of the coding exercises provided in the text. The solutions are written in Python and use the NumPy library for numerical operations. The code snippets are self-contained and can be run in any Python environment. The solutions are organized by the exercise they correspond to and are presented in the same order as in the text. For convenience, the full code solution file resmet.py can be saved here.

## A.1. Hartree–Fock Method

```
# energies, number of occupied and virtual orbitals and the number of basis functions
E_HF, E_HF_P, VNN, nbf, nocc = 0, 1, 0, S.shape[0], sum(atoms) // 2; nvirt = nbf - nocc

# exchange integrals and the guess density matrix
K, D = J.transpose(0, 3, 2, 1), np.zeros((nbf, nbf))

# Fock matrix, coefficient matrix and orbital energies initialized to zero
F, C, eps = np.zeros((nbf, nbf)), np.zeros((nbf, nbf)), np.zeros((nbf))

# the X matrix which is the inverse of the square root of the overlap matrix
SEP = np.linalg.eigh(S); X = SEP[1] @ np.diag(1 / np.sqrt(SEP[0])) @ SEP[1].T

# the scf loop
while abs(E_HF - E_HF_P) > args.threshold:

    # build the Fock matrix
    F = H + np.einsum("ijkl,ij->kl", J - 0.5 * K, D, optimize=True)

    # solve the Fock equations
    eps, C = np.linalg.eigh(X @ F @ X); C = X @ C

    # build the density from coefficients
    D = 2 * np.einsum("ij,kj->ik", C[:, :nocc], C[:, :nocc])

    # save the previous energy and calculate the current electron energy
    E_HF_P, E_HF = E_HF, 0.5 * np.einsum("ij,ij->", D, H + F, optimize=True)

# calculate nuclear-nuclear repulsion
for i, j in ((i, j) for i, j in it.product(range(natoms), range(natoms)) if i != j):
    VNN += 0.5 * atoms[i] * atoms[j] / np.linalg.norm(coords[i, :] - coords[j, :])

# print the results
print("    RHF ENERGY: {:.8f}".format(E_HF + VNN))
```

**Listing A.1:** HF method exercise code solution.

## A.2. Integral Transform

```
# define the occ and virt spinorbital slices shorthand
o, v = slice(0, 2 * nocc), slice(2 * nocc, 2 * nbf)

# define the tiling matrix for the Jmsa coefficients and energy placeholders
```

```python
P = np.array([np.eye(nbf)[:, i // 2] for i in range(2 * nbf)]).T

# define the spin masks
M = np.repeat([1 - np.arange(2 * nbf, dtype=int) % 2], nbf, axis=0)
N = np.repeat([    np.arange(2 * nbf, dtype=int) % 2], nbf, axis=0)

# tile the coefficient matrix, apply the spin mask and tile the orbital energies
Cms, epsms = np.block([[C @ P], [C @ P]]) * np.block([[M], [N]]), np.repeat(eps, 2)

# transform the core Hamiltonian and Fock matrix to the molecular spinorbital basis
Hms = np.einsum("ip,ij,jq->pq", Cms, np.kron(np.eye(2), H), Cms, optimize=True)
Fms = np.einsum("ip,ij,jq->pq", Cms, np.kron(np.eye(2), F), Cms, optimize=True)

# transform the coulomb integrals to the MS basis in chemists' notation
Jms = np.einsum("ip,jq,ijkl,kr,ls->pqrs",
    Cms, Cms, np.kron(np.eye(2), np.kron(np.eye(2), J).T), Cms, Cms, optimize=True
);

# antisymmetrized two-electron integrals in physicists' notation
Jmsa = (Jms - Jms.swapaxes(1, 3)).transpose(0, 2, 1, 3)

# tensor epsilon_i^a
Emss = epsms[o] - epsms[v, None]

# tensor epsilon_ij^ab
Emsd = epsms[o] + epsms[o, None] - epsms[v, None, None] - epsms[v, None, None, None]
```

**Listing A.2:** Integral transform exercise code solution.

## A.3. 2nd and 3rd Order Perturbative Corrections

```python
# energy containers
E_MP2, E_MP3 = 0, 0

# calculate the MP2 correlation energy
if args.mp2 or args.mp3:
    E_MP2 += 0.25 * np.einsum("abij,ijab,abij",
        Jmsa[v, v, o, o], Jmsa[o, o, v, v], Emsd**-1, optimize=True
    )
    print("    MP2 ENERGY: {:.8f}".format(E_HF + E_MP2 + VNN))

# calculate the MP3 correlation energy
if args.mp3:
    E_MP3 += 0.125 * np.einsum("abij,cdab,ijcd,abij,cdij",
        Jmsa[v, v, o, o], Jmsa[v, v, v, v], Jmsa[o, o, v, v], Emsd**-1, Emsd**-1,
        optimize=True
    )
    E_MP3 += 0.125 * np.einsum("abij,ijkl,klab,abij,abkl",
        Jmsa[v, v, o, o], Jmsa[o, o, o, o], Jmsa[o, o, v, v], Emsd**-1, Emsd**-1,
        optimize=True
    )
    E_MP3 += 1.000 * np.einsum("abij,cjkb,ikac,abij,acik",
        Jmsa[v, v, o, o], Jmsa[v, o, o, v], Jmsa[o, o, v, v], Emsd**-1, Emsd**-1,
        optimize=True
    )
    print("    MP3 ENERGY: {:.8f}".format(E_HF + E_MP2 + E_MP3 + VNN))
```

**Listing A.3:** MP2 and MP3 exercise code solution.

## A.4. Full Configuration Interaction

```
# generate the determiants
dets = [np.array(det) for det in it.combinations(range(2 * nbf), 2 * nocc)]

# define the CI Hamiltonian
Hci = np.zeros([len(dets), len(dets)])

# define the Slater-Condon rules, "so" is an array of unique and common spinorbitals
slater0 = lambda so: (
    sum(np.diag(Hms)[so]) + sum([0.5 * Jmsa[m, n, m, n] for m, n in it.product(so, so)])
)
slater1 = lambda so: (
    Hms[so[0], so[1]] + sum([Jmsa[so[0], m, so[1], m] for m in so[2:]])
)
slater2 = lambda so: (
    Jmsa[so[0], so[1], so[2], so[3]]
)

# filling of the CI Hamiltonian
for i in range(0, Hci.shape[0]):
    for j in range(i, Hci.shape[1]):

        # aligned determinant and the sign
        aligned, sign = dets[j].copy(), 1

        # align the determinant "j" to "i" and calculate the sign
        for k in (k for k in range(len(aligned)) if aligned[k] != dets[i][k]):
            while len(l := np.where(dets[i] == aligned[k])[0]) and l[0] != k:
                aligned[[k, l[0]]] = aligned[[l[0], k]]; sign *= -1

        # find the unique and common spinorbitals
        so = np.block(list(map(lambda l: np.array(l), [
            [aligned[k] for k in range(len(aligned)) if aligned[k] not in dets[i]],
            [dets[i][k] for k in range(len(dets[j])) if dets[i][k] not in aligned],
            [aligned[k] for k in range(len(aligned)) if aligned[k] in dets[i]]
        ]))).astype(int)

        # apply the Slater-Condon rules and multiply by the sign
        if ((aligned - dets[i]) != 0).sum() == 0: Hci[i, j] = slater0(so) * sign
        if ((aligned - dets[i]) != 0).sum() == 1: Hci[i, j] = slater1(so) * sign
        if ((aligned - dets[i]) != 0).sum() == 2: Hci[i, j] = slater2(so) * sign

        # fill the lower triangle
        Hci[j, i] = Hci[i, j]

# solve the eigensystem and assign energy
eci, Cci = np.linalg.eigh(Hci); E_FCI = eci[0] - E_HF

# print the results
print("    FCI ENERGY: {:.8f}".format(E_HF + E_FCI + VNN))
```

**Listing A.4:** CI exercise code solution.

## A.5. Coupled Cluster Singles and Doubles

```
# energy containers for all the CC methods
E_CCD, E_CCD_P, E_CCSD, E_CCSD_P = 0, 1, 0, 1

# initialize the first guess for the t-amplitudes as zeros
t1, t2 = np.zeros((2 * nvirt, 2 * nocc)), np.zeros(2 * [2 * nvirt] + 2 * [2 * nocc])

# CCD loop
if args.ccd:
    while abs(E_CCD - E_CCD_P) > args.threshold:
```

```
        # collect all the distinct LCCD terms
        lccd1 = 0.5 * np.einsum("abcd,cdij->abij", Jmsa[v, v, v, v], t2, optimize=True)
        lccd2 = 0.5 * np.einsum("klij,abkl->abij", Jmsa[o, o, o, o], t2, optimize=True)
        lccd3 = 1.0 * np.einsum("akic,bcjk->abij", Jmsa[v, o, o, v], t2, optimize=True)

        # apply the permuation operator and add it to the corresponding LCCD terms
        lccd3 += lccd3.transpose(1, 0, 3, 2) - lccd3.swapaxes(0, 1) - lccd3.swapaxes(2, 3)

        # collect all the remaining CCD terms
        ccd1 = -0.50 * np.einsum("klcd,acij,bdkl->abij",
            Jmsa[o, o, v, v], t2, t2, optimize=True
        )
        ccd2 = -0.50 * np.einsum("klcd,abik,cdjl->abij",
            Jmsa[o, o, v, v], t2, t2, optimize=True
        )
        ccd3 = +0.25 * np.einsum("klcd,cdij,abkl->abij",
            Jmsa[o, o, v, v], t2, t2, optimize=True
        )
        ccd4 = +1.00 * np.einsum("klcd,acik,bdjl->abij",
            Jmsa[o, o, v, v], t2, t2, optimize=True
        )

        # permutation operators
        ccd1 -= ccd1.swapaxes(0, 1);
        ccd2 -= ccd2.swapaxes(2, 3);
        ccd4 -= ccd4.swapaxes(2, 3)

        # update the t-amplitudes
        t2 = (Jmsa[v, v, o, o] + lccd1 + lccd2 + lccd3 + ccd1 + ccd2 + ccd3 + ccd4) / Emsd

        # evaluate the energy
        E_CCD_P, E_CCD = E_CCD, 0.25 * np.einsum("ijab,abij", Jmsa[o, o, v, v], t2)

    # print the CCD energy
    print("    CCD ENERGY: {:.8f}".format(E_HF + E_CCD + VNN))

# CCSD loop
if args.ccsd:
    while abs(E_CCSD - E_CCSD_P) > args.threshold:

        # define taus
        tau, ttau = t2.copy(), t2.copy()

        # add contributions to the tilde tau
        ttau += 0.5 * np.einsum("ai,bj->abij", t1, t1, optimize=True).swapaxes(0, 0)
        ttau -= 0.5 * np.einsum("ai,bj->abij", t1, t1, optimize=True).swapaxes(2, 3)

        # add the contributions to tau
        tau += np.einsum("ai,bj->abij", t1, t1, optimize=True).swapaxes(0, 0)
        tau -= np.einsum("ai,bj->abij", t1, t1, optimize=True).swapaxes(2, 3)

        # define the deltas for Fae and Fmi
        dae, dmi = np.eye(2 * nvirt), np.eye(2 * nocc)

        # define Fae, Fmi and Fme
        Fae, Fmi, Fme = (1 - dae) * Fms[v, v], (1 - dmi) * Fms[o, o], Fms[o, v].copy()

        # add the contributions to Fae
        Fae -= 0.5 * np.einsum("me,am->ae",     Fms[o, v],        t1,   optimize=True)
        Fae += 1.0 * np.einsum("mafe,fm->ae",   Jmsa[o, v, v, v], t1,   optimize=True)
        Fae -= 0.5 * np.einsum("mnef,afmn->ae", Jmsa[o, o, v, v], ttau, optimize=True)

        # add the contributions to Fmi
        Fmi += 0.5 * np.einsum("me,ei->mi",     Fms[o, v],        t1,   optimize=True)
        Fmi += 1.0 * np.einsum("mnie,en->mi",   Jmsa[o, o, o, v], t1,   optimize=True)
```

```
        Fmi += 0.5 * np.einsum("mnef,efin->mi", Jmsa[o, o, v, v], ttau, optimize=True)

        # add the contributions to Fme
        Fme += np.einsum("mnef,fn->me", Jmsa[o, o, v, v], t1, optimize=True)

        # define Wmnij, Wabef and Wmbej
        Wmnij = Jmsa[o, o, o, o].copy()
        Wabef = Jmsa[v, v, v, v].copy()
        Wmbej = Jmsa[o, v, v, o].copy()

        # define some complementary variables used in the Wmbej intermediate
        t12  = 0.5 * t2 + np.einsum("fj,bn->fbjn", t1, t1,  optimize=True)

        # add contributions to Wmnij
        Wmnij += 0.25 * np.einsum("efij,mnef->mnij", tau, Jmsa[o, o, v, v], optimize=True)
        Wabef += 0.25 * np.einsum("abmn,mnef->abef", tau, Jmsa[o, o, v, v], optimize=True)
        Wmbej += 1.00 * np.einsum("fj,mbef->mbej",   t1,  Jmsa[o, v, v, v], optimize=True)
        Wmbej -= 1.00 * np.einsum("bn,mnej->mbej",   t1,  Jmsa[o, o, v, o], optimize=True)
        Wmbej -= 1.00 * np.einsum("fbjn,mnef->mbej", t12, Jmsa[o, o, v, v], optimize=True)

        # define the permutation arguments for Wmnij and Wabef and add them
        PWmnij = np.einsum("ej,mnie->mnij", t1, Jmsa[o, o, o, v], optimize=True)
        PWabef = np.einsum("bm,amef->abef", t1, Jmsa[v, o, v, v], optimize=True)

        # add the permutations to Wmnij and Wabef
        Wmnij += PWmnij - PWmnij.swapaxes(2, 3)
        Wabef += PWabef.swapaxes(0, 1) - PWabef

        # define the right hand side of the T1 and T2 amplitude equations
        rhs_T1, rhs_T2 = Fms[v, o].copy(), Jmsa[v, v, o, o].copy()

        # calculate the right hand side of the CCSD equation for T1
        rhs_T1 += 1.0 * np.einsum("ei,ae->ai",     t1, Fae,              optimize=True)
        rhs_T1 -= 1.0 * np.einsum("am,mi->ai",     t1, Fmi,              optimize=True)
        rhs_T1 += 1.0 * np.einsum("aeim,me->ai",   t2, Fme,              optimize=True)
        rhs_T1 -= 1.0 * np.einsum("fn,naif->ai",   t1, Jmsa[o, v, o, v], optimize=True)
        rhs_T1 -= 0.5 * np.einsum("efim,maef->ai", t2, Jmsa[o, v, v, v], optimize=True)
        rhs_T1 -= 0.5 * np.einsum("aemn,nmei->ai", t2, Jmsa[o, o, v, o], optimize=True)

        # contracted F matrices that used in the T2 equations
        Faet = np.einsum("bm,me->be", t1, Fme, optimize=True)
        Fmet = np.einsum("ej,me->mj", t1, Fme, optimize=True)

        # define the permutation arguments for all terms in the equation for T2
        P1  = np.einsum("aeij,be->abij",    t2,     Fae - 0.5 * Faet, optimize=True)
        P2  = np.einsum("abim,mj->abij",    t2,     Fmi + 0.5 * Fmet, optimize=True)
        P3  = np.einsum("aeim,mbej->abij",  t2,     Wmbej,            optimize=True)
        P3 -= np.einsum("ei,am,mbej->abij", t1, t1, Jmsa[o, v, v, o], optimize=True)
        P4  = np.einsum("ei,abej->abij",    t1,     Jmsa[v, v, v, o], optimize=True)
        P5  = np.einsum("am,mbij->abij",    t1,     Jmsa[o, v, o, o], optimize=True)

        # calculate the right hand side of the CCSD equation for T2
        rhs_T2 += 0.5 * np.einsum("abmn,mnij->abij", tau, Wmnij, optimize=True)
        rhs_T2 += 0.5 * np.einsum("efij,abef->abij", tau, Wabef, optimize=True)
        rhs_T2 += P1.transpose(0, 1, 2, 3) - P1.transpose(1, 0, 2, 3)
        rhs_T2 -= P2.transpose(0, 1, 2, 3) - P2.transpose(0, 1, 3, 2)
        rhs_T2 += P3.transpose(0, 1, 2, 3) - P3.transpose(0, 1, 3, 2)
        rhs_T2 -= P3.transpose(1, 0, 2, 3) - P3.transpose(1, 0, 3, 2)
        rhs_T2 += P4.transpose(0, 1, 2, 3) - P4.transpose(0, 1, 3, 2)
        rhs_T2 -= P5.transpose(0, 1, 2, 3) - P5.transpose(1, 0, 2, 3)

        # Update T1 and T2 amplitudes and save the previous iteration
        t1, t2 = rhs_T1 / Emss, rhs_T2 / Emsd; E_CCSD_P = E_CCSD

        # evaluate the energy
        E_CCSD  = 1.00 * np.einsum("ia,ai",      Fms[o, v],        t1    )
```

```
        E_CCSD += 0.25 * np.einsum("ijab,abij",  Jmsa[o, o, v, v], t2    )
        E_CCSD += 0.50 * np.einsum("ijab,ai,bj", Jmsa[o, o, v, v], t1, t1)

    # print the CCSD energy
    print("    CCSD ENERGY: {:.8f}".format(E_HF + E_CCSD + VNN))
```

**Listing A.5:** CCD and CCSD method exercise code solution.

# Acronyms

**CC** Coupled Cluster. 4, 18, 19

**CCD** Coupled Cluster Doubles. 4, 18, 19, 28

**CCSD** Coupled Cluster Singles and Doubles. 4, 18–21, 28

**CI** Configuration Interaction. 4, 14, 15, 17, 25

**CISD** Configuration Interaction Singles and Doubles. 14

**CISDT** Configuration Interaction Singles, Doubles and Triples. 14

**DIIS** Direct Inversion in the Iterative Subspace. 7

**FCI** Full Configuration Interaction. 4, 14, 15

**HF** Hartree–Fock. 4–7, 9, 10, 12–15, 18–20, 23

**MP2** Møller–Plesset Perturbation Theory of 2nd Order. 13, 24

**MP3** Møller–Plesset Perturbation Theory of 3rd Order. 13, 24

**MPPT** Møller–Plesset Perturbation Theory. 4, 12

**MS** Molecular Spinorbital. 8, 9, 11, 13, 15, 19

**post-HF** post-Hartree–Fock. 8, 14, 18

**RHF** Restricted Hartree–Fock. 5, 8

**SCF** Self-Consistent Field. 6, 10

# Bibliography


(1) Gill, P. M., *Molecular integrals Over Gaussian Basis Functions*; Academic Press: 1994.

(2) Yamaguchi, Y.; Schaefer, H. F., *Analytic Derivative Methods in Molecular Electronic Structure Theory: A New Dimension to Quantum Chemistry and its Applications to Spectroscopy*; John Wiley & Sons, Ltd: 2011.

(3) Cremer, D. *WIREs Computational Molecular Science* **2011**, *1*, 509–530.

(4) Szabo, A.; Ostlund, N. S., *Modern quantum chemistry: introduction to advanced electronic structure theory*; Courier Corporation: 1996.

(5) Paldus, J.; Wong, H. H. *Computer Physics Communications* **1973**, *6*, 1–7.

(6) Wong, H. H.; Paldus, J. *Computer Physics Communications* **1973**, *6*, 9–16.

(7) Stanton, J. F.; Gauss, J.; Watts, J. D.; Bartlett, R. J. *The Journal of Chemical Physics* **1991**, *94*, 4334–4345.