The Coupled Cluster Method

Thomas Papenbrock

The University of Tennessee, Knoxville, tpapenbr@utk.edu

Aug 4, 2018

Contents

Chapter 1

The Coupled-Cluster Method

Introduction

The coupled-cluster method is an efficient tool to compute atomic nuclei with an effort that grows polynomial with system size. While this might still be expensive, it is now possible to compute nuclei with mass numbers about $A \approx 100$ with this method. Recall that full configuration interaction (FCI) such as the no-core shell model exhibits an exponential cost and is therefore limited to light nuclei.

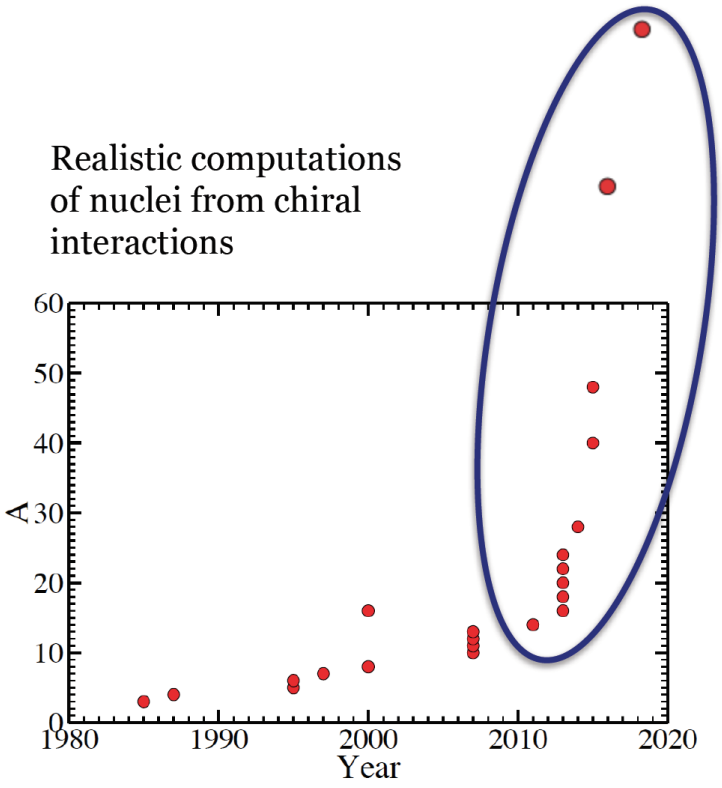


Figure 1.1: Realistic computations of atomic nuclei with interactions from chiral EFT. The slow increase prior to 2015 is based on quantum Monte Carlo and the no-core shell model. These methods are exponentially expensive (in mass number A) and meet with exponentially increasing computer power (Moore's law), thus leading to a progress that is linear in time. Methods such as coupled clusters and in-medium SRG carry a polynomial cost in mass number are transforming the field.

The normal-ordered Hamiltonian

We start from the reference state

$$|\Phi_0\rangle = \prod_{i=1}^{A} a_i^\dagger |0\rangle \tag{1.1}$$

for the description of a nucleus with mass number A. Usually, this reference is the Hartree-Fock state, but that is not necessary. In the shell-model picture, it could also be a product state where the lowest A harmonic oscillator states are occupied. Here and in what follows, the indices $i, j, k, \ldots$ run over hole states, i.e. orbitals occupied in the reference state (1.1), while $a, b, c, \ldots$ run over particle

states, i.e. unoccupied orbitals. Indices p, q, r, s can identify any orbital. Let n_u be the number of unoccupied states, and A is of course the number of occupied states. We consider the Hamiltonian

$$H = \sum_{pq} \varepsilon_q^p a_p^\dagger a_q + \frac{1}{4} \sum_{pqrs} \langle pq|V|rs\rangle a_p^\dagger a_q^\dagger a_s a_r \tag{1.2}$$

The reference state (1.1) is a non-trivial vacuum of our theory. We normal order this Hamiltonian with respect to the nontrivial vacuum state given by the Hartree-Fock reference and obtain the normal-ordered Hamiltonian

$$H_N = \sum_{pq} f_{pq} \left\{ a_p^\dagger a_q \right\} + \frac{1}{4} \sum_{pqrs} \langle pq|V|rs\rangle \left\{ a_p^\dagger a_q^\dagger a_s a_r \right\}. \tag{1.3}$$

Here,

$$f_q^p = \varepsilon_q^p + \sum_i \langle pi|V|qi\rangle \tag{1.4}$$

is the Fock matrix. We note that the Fock matrix is diagonal in the Hartree-Fock basis. The brackets $\{\cdots\}$ in Eq. (1.3) denote normal ordering, i.e. all operators that annihilate the nontrivial vaccum (1.1) are to the right of those operators that create with respect to that vaccum. Normal ordering implies that $\langle \Phi_0 | H_N | \Phi_0 \rangle = 0$.

*

Exercise 1: Practice in normal ordering
Normal order the expression $\sum_{pq} \varepsilon_q^p a_p^\dagger a_q$.

Hint.

$$\sum_{pq} \varepsilon_q^p a_p^\dagger a_q = \sum_{ab} \varepsilon_b^a a_a^\dagger a_b + \sum_{ai} \varepsilon_i^a a_a^\dagger a_i + \sum_{ai} \varepsilon_a^i a_i^\dagger a_a + \sum_{ij} \varepsilon_j^i a_i^\dagger a_j \tag{1.5}$$

Answer. We have to move all operators that annihilate the reference state to the right of those that create on the reference state. Thus,

$$\sum_{pq} \varepsilon_q^p a_p^\dagger a_q = \sum_{ab} \varepsilon_b^a a_a^\dagger a_b + \sum_{ai} \varepsilon_i^a a_a^\dagger a_i + \sum_{ai} \varepsilon_a^i a_i^\dagger a_a + \sum_{ij} \varepsilon_j^i a_i^\dagger a_j \tag{1.6}$$

$$= \sum_{ab} \varepsilon_b^a a_a^\dagger a_b + \sum_{ai} \varepsilon_i^a a_a^\dagger a_i + \sum_{ai} \varepsilon_a^i a_i^\dagger a_a + \sum_{ij} \varepsilon_j^i \left(-a_j a_i^\dagger + \delta_i^j \right) \tag{1.7}$$

$$= \sum_{ab} \varepsilon_b^a a_a^\dagger a_b + \sum_{ai} \varepsilon_i^a a_a^\dagger a_i + \sum_{ai} \varepsilon_a^i a_i^\dagger a_a - \sum_{ij} \varepsilon_j^i a_j a_i^\dagger + \sum_i \varepsilon_i^i \tag{1.8}$$

$$= \sum_{pq} \varepsilon_q^p \left\{ a_p^\dagger a_q \right\} + \sum_i \varepsilon_i^i \tag{1.9}$$

===== =====

We note that $H = E_{HF} + H_N$, where

$$E_{HF} \equiv \langle \Phi_0 | H | \Phi_0 \rangle = \sum_i \varepsilon_i^i + \frac{1}{2} \sum_{ij} \langle ij | V | ij \rangle \tag{1.10}$$

is the Hartree-Fock energy. The coupled-cluster method is a very efficient tool to compute nuclei when a "good" reference state is available. Let us assume that the reference state results from a Hartree-Fock calculation.

*

Exercise 2: What does "good" mean?

How do you know whether a Hartree-Fock state is a "good" reference? Which results of the Hartree-Fock computation will inform you?

Answer. Once the Hartree-Fock equations are solved, the Fock matrix (1.4) becomes diagonal, and its diagonal elements can be viewed as single-particle energies. Hopefully, there is a clear gap in the single-particle spectrum at the Fermi surface, i.e. after A orbitals are filled.

===== =====

If symmetry-restricted Hartree-Fock is used, one is limited to compute nuclei with closed subshells for neutrons and for protons. On a first view, this might seem as a severe limitation. But is it?

*

Exercise 3: How many nuclei are accessible with the coupled cluster method based on spherical mean fields?

If one limits oneself to nuclei with mass number up to mass number $A = 60$, how many nuclei can potentially be described with the coupled-cluster method? Which of these nuclei are potentially interesting? Why?

Answer. Nuclear shell closures are at $N, Z = 2, 8, 20, 28, 50, 82, 126$, and subshell closures at $N, Z = 2, 6, 8, 14, 16, 20, 28, 32, 34, 40, 50, \ldots$.

In the physics of nuclei, the evolution of nuclear structure as neutrons are added (or removed) from an isotope is a key interest. Examples are the rare isotopes of helium (He-8,10) oxygen (O-22,24,28), calcium (Ca-52,54,60), nickel (Ni-78) and tin (Sn-100,132). The coupled-cluster method has the potential to address questions regarding these nuclei, and in a several cases was used to make predictions before experimental data was available. In addition, the method can be used to compute neighbors of nuclei with closed subshells.

===== =====

The similarity transformed Hamiltonian

There are several ways to view and understand the coupled-cluster method. A first simple view of coupled-cluster theory is that the method induces correlations into the reference state by expressing a correlated state as

$$|\Psi\rangle = e^T |\Phi_0\rangle, \tag{1.11}$$

Here, T is an operator that induces correlations. We can now demand that the correlate state (1.11) becomes and eigenstate of the Hamiltonian H_N, i.e. $H_N|\Psi\rangle = E|\Psi\rangle$. This view, while correct, is not the most productive one. Instead, we left-multiply the Schroedinger equation with e^{-T} and find

$$\overline{H_N}|\Phi_0\rangle = E_c|\Phi_0\rangle. \tag{1.12}$$

Here, E_c is the correlation energy, and the total energy is $E = E_c + E_{HF}$. The similarity-transformed Hamiltonian is defined as

$$\overline{H_N} \equiv e^{-T} H_N e^T. \tag{1.13}$$

A more productive view on coupled-cluster theory thus emerges: This method seeks a similarity transformation such that the uncorrelated reference state (1.1) becomes an exact eigenstate of the similarity-transformed Hamiltonian (1.13).

*

Exercise 4: What T leads to Hermitian $\overline{H_N}$?

What are the conditions on T such that $\overline{H_N}$ is Hermitian?

Answer. For a Hermitian $\overline{H_N}$, we need a unitary e^T, i.e. an anti-Hermitian T with $T = -T^\dagger$

===== =====

As we will see below, coupld-cluster theory employs a non-Hermitian Hamiltonian.

*

Exercise 5: Understanding (non-unitary) similarity transformations

Show that $\overline{H_N}$ has the same eigenvalues as H_N for arbitrary T. What is the spectral decomposition of a non-Hermitian $\overline{H_N}$?

Answer. Let $H_N|E\rangle = E|E\rangle$. Thus

$$\begin{aligned} H_N e^T e^{-T}|E\rangle &= E|E\rangle, \\ \left(e^{-T} H_N e^T\right) e^{-T}|E\rangle &= E e^{-T}|E\rangle, \\ \overline{H_N} e^{-T}|E\rangle &= E e^{-T}|E\rangle. \end{aligned}$$

Thus, if $|E\rangle$ is an eigenstate of H_N with eigenvalue E, then $e^{-T}|E\rangle$ is eigenstate of $\overline{H_N}$ with the same eigenvalue.

A non-Hermitian $\overline{H_N}$ has eigenvalues E_α corresponding to left $\langle L_\alpha|$ and right $|R_\alpha\rangle$ eigenstates. Thus

$$\overline{H_N} = \sum_\alpha |R_\alpha\rangle E_\alpha \langle L_\alpha| \tag{1.14}$$

with bi-orthonormal $\langle L_\alpha|R_\beta\rangle = \delta_\alpha^\beta$.

===== =====

To make progress, we have to specify the cluster operator T. In coupled cluster theory, this operator is

$$T \equiv \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i + \cdots + \frac{1}{(A!)^2} \sum_{i_1 \ldots i_A a_1 \ldots a_A} t_{i_1 \ldots i_A}^{a_1 \ldots a_A} a_{a_1}^\dagger \cdots a_{a_A}^\dagger a_{i_A} \cdots a_{i_1}. \tag{1.15}$$

Thus, the operator (1.15) induces particle-hole (p-h) excitations with respect to the reference. In general, T generates up to $Ap - Ah$ excitations, and the unknown parameters are the cluster amplitides t_i^a, t_{ij}^{ab}, ..., $t_{i_1,\ldots,i_A}^{a_1,\ldots,a_A}$.

*

Exercise 6: How many unknowns?

Show that the number of unknowns is as large as the FCI dimension of the problem, using the numbers A and n_u.

Answer. We have to sum up all $np-nh$ excitations, and there are $\binom{n_u}{n}$ particle states and $\binom{A}{A-n}$ hole states for each n. Thus, we have for the total number

$$\sum_{n=0}^{A} \binom{n_u}{n}\binom{A}{A-n} = \binom{A+n_u}{A}. \tag{1.16}$$

The right hand side are obviously all ways to distribute A fermions over $n_0 + A$ orbitals.

===== =====

Thus, the coupled-cluster method with the full cluster operator (1.15) is exponentially expensive, just as FCI. To make progress, we need to make an approximation by truncating the operator. Here, we will use the CCSD (coupled clusters singles doubles) approximation, where

$$T \equiv \sum_{ia} t_i^a a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} t_{ij}^{ab} a_a^\dagger a_b^\dagger a_j a_i. \tag{1.17}$$

We need to determine the unknown cluster amplitudes that enter in CCSD. Let

$$|\Phi_i^a\rangle = a_a^\dagger a_i |\Phi_0\rangle, \tag{1.18}$$
$$|\Phi_{ij}^{ab}\rangle = a_a^\dagger a_b^\dagger a_j a_i |\Phi_0\rangle \tag{1.19}$$

be 1p-1h and 2p-2h excitations of the reference. Computing matrix elements of the Schroedinger Equation (1.12) yields

$$\langle\Phi_0|\overline{H_N}|\Phi_0\rangle = E_c, \tag{1.20}$$
$$\langle\Phi_i^a|\overline{H_N}|\Phi_0\rangle = 0, \tag{1.21}$$
$$\langle\Phi_{ij}^{ab}|\overline{H_N}|\Phi_0\rangle = 0. \tag{1.22}$$

The first equation states that the coupled-cluster correlation energy is an expectation value of the similarity-transformed Hamiltonian. The second and third equations state that the similarity-transformed Hamiltonian exhibits no 1p-1h and no 2p-2h excitations. These equations have to be solved to find the

unknown amplitudes t_i^a and t_{ij}^{ab}. Then one can use these amplitudes and compute the correlation energy from the first line of Eq. (1.20).

We note that in the CCSD approximation the reference state is not an exact eigenstates. Rather, it is decoupled from simple states but $\overline{H}$ still connects this state to 3p-3h, and 4p-4h states etc.

At this point, it is important to recall that we assumed starting from a "good" reference state. In such a case, we might reasonably expect that the inclusion of 1p-1h and 2p-2h excitations could result in an accurate approximation. Indeed, empirically one finds that CCSD accounts for about 90% of the corelation energy, i.e. of the difference between the exact energy and the Hartree-Fock energy. The inclusion of triples (3p-3h excitations) typically yields 99% of the correlation energy.

We see that the coupled-cluster method in its CCSD approximation yields a similarity-transformed Hamiltonian that is of a two-body structure with respect to a non-trivial vacuum. When viewed in this light, the coupled-cluster method "transforms" an A-body problem (in CCSD) into a two-body problem, albeit with respect to a nontrivial vacuum.

*

Exercise 7: Why is CCD not exact?

Above we argued that a similarity transformation preserves all eigenvalues. Nevertheless, the CCD correlation energy is not the exact correlation energy. Explain!

Answer. The CCD approximation does not make $|\Phi_0\rangle$ an exact eigenstate of $\overline{H_N}$; it is only an eigenstate when the similarity-transformed Hamiltonian is truncated to at most 2p-2h states. The full $\overline{H_N}$, with $T = T_2$, would involve six-body terms (do you understand this?), and this full Hamiltonian would reproduce the exact correlation energy. Thus CCD is a similarity transformation plus a truncation, which decouples the ground state only from 2p-2h states.

===== =====

Computing the similarity-transformed Hamiltonian

The solution of the CCSD equations, i.e. the second and third line of Eq. (1.20), and the computation of the correlation energy requires us to compute matrix elements of the similarity-transformed Hamiltonian (1.13). This can be done with the Baker-Campbell-Hausdorff expansion

$$\overline{H_N} = e^{-T} H_N e^T \tag{1.23}$$

$$= H_N + [H_N, T] + \frac{1}{2!}[[H_N, T], T] + \frac{1}{3!}[[[H_N, T], T], T] + \dots . \tag{1.24}$$

We now come to a key element of coupled-cluster theory: the cluster operator (1.15) consists of sums of terms that consist of particle creation and hole annihilation operators (but no particle annihilation or hole creation operators). Thus, all terms that enter T commute with each other. This means that the commutators in the Baker-Campbell-Hausdorff expansion (1.23) can only be non-zero because each T must connect to H_N (but no T with another T). Thus, the expansion is finite.

*

Exercise 8: When does CCSD truncate?

In CCSD and for two-body Hamiltonians, how many nested commutators yield nonzero results? Where does the Baker-Campbell-Hausdorff expansion terminate? What is the (many-body) rank of the resulting $\overline{H_N}$?

Answer. CCSD truncates for two-body operators at four-fold nested commutators, because each of the four annihilation and creation operators in $\overline{H_N}$ can be knocked out with one term of T.

===== =====

We see that the (disadvantage of having a) non-Hermitian Hamiltonian $\overline{H_N}$ leads to the advantage that the Baker-Campbell-Hausdorff expansion is finite, thus leading to the possibility to compute $\overline{H_N}$ exactly. In contrast, the IMSRG deals with a Hermitian Hamiltonian throughout, and the infinite Baker-Campbell-Hausdorff expansion is truncated at a high order when terms become very small.

We write the similarity-transformed Hamiltonian as

$$\overline{H_N} = \sum_{pq} \overline{H}^p_q a^\dagger_q a_p + \frac{1}{4} \sum_{pqrs} \overline{H}^{pq}_{rs} a^\dagger_p a^\dagger_q a_s a_r + \dots \tag{1.25}$$

with

$$\overline{H}^p_q \equiv \langle p | \overline{H_N} | q \rangle, \tag{1.26}$$

$$\overline{H}^{pq}_{rs} \equiv \langle pq | \overline{H_N} | rs \rangle. \tag{1.27}$$

Thus, the CCSD Eqs. (1.20) for the amplitudes can be written as $\overline{H}^a_i = 0$ and $\overline{H}^{ab}_{ij} = 0$.

*

Exercise 9: Compute the matrix element $\overline{H}^{ij}_{ab} \equiv \langle ij|\overline{H_N}|ab\rangle$

Answer. This is a simple task. This matrix element is part of the operator $\overline{H}^{ij}_{ab} a^\dagger_i a^\dagger_j a_b a_a$, i.e. particles are annihilated and holes are created. Thus, no contraction of the Hamiltonian H with any cluster operator T (remember that T annihilates holes and creates particles) can happen, and we simply have $\overline{H}^{ij}_{ab} = \langle ij|V|ab\rangle$.

===== =====

We need to work out the similarity-transformed Hamiltonian of Eq. (1.23). To do this, we write $T = T_1 + T_2$ and $H_N = F + V$, where T_1 and F are one-body operators, and T_2 and V are two-body operators.

Example: The contribution of $[F, T_2]$ to $\overline{H_N}$

The commutator $[F, T_2]$ consists of two-body and one-body terms. Let us compute first the two-body term, as it results from a single contraction (i.e. a single application of $[a_p, a^\dagger_q] = \delta^q_p$). We denote this as $[F, T_2]_{2b}$ and find

$$
\begin{aligned}
[F, T_2]_{2b} &= \frac{1}{4} \sum_{pq} \sum_{rsuv} f_p^q t_{ij}^{ab} \left[a_q^\dagger a_p, a_a^\dagger a_b^\dagger a_j a_i \right]_{2b} \\
&= \frac{1}{4} \sum_{pq} \sum_{abij} f_p^q t_{ij}^{ab} \delta_p^a a_q^\dagger a_b^\dagger a_j a_i \\
&- \frac{1}{4} \sum_{pq} \sum_{abij} f_p^q t_{ij}^{ab} \delta_p^b a_q^\dagger a_a^\dagger a_j a_i \\
&- \frac{1}{4} \sum_{pq} \sum_{abij} f_p^q t_{ij}^{ab} \delta_q^j a_a^\dagger a_b^\dagger a_p a_i \\
&+ \frac{1}{4} \sum_{pq} \sum_{abij} f_p^q t_{ij}^{ab} \delta_q^i a_a^\dagger a_b^\dagger a_p a_j \\
&= \frac{1}{4} \sum_{qbij} \left(\sum_a f_a^q t_{ij}^{ab} \right) a_q^\dagger a_b^\dagger a_j a_i \\
&- \frac{1}{4} \sum_{qaij} \left(\sum_b f_b^q t_{ij}^{ab} \right) a_q^\dagger a_a^\dagger a_j a_i \\
&- \frac{1}{4} \sum_{pabi} \left(\sum_j f_p^j t_{ij}^{ab} \right) a_a^\dagger a_b^\dagger a_p a_i \\
&+ \frac{1}{4} \sum_{pabj} \left(\sum_i f_p^i t_{ij}^{ab} \right) a_a^\dagger a_b^\dagger a_p a_j \\
&= \frac{1}{2} \sum_{qbij} \left(\sum_a f_a^q t_{ij}^{ab} \right) a_q^\dagger a_b^\dagger a_j a_i \\
&- \frac{1}{2} \sum_{pabi} \left(\sum_j f_p^j t_{ij}^{ab} \right) a_a^\dagger a_b^\dagger a_p a_i .
\end{aligned}
$$

Here we exploited the antisymmetry $t_{ij}^{ab} = -t_{ji}^{ab} = -t_{ij}^{ba} = t_{ji}^{ba}$ in the last step. Using $a_q^\dagger a_b^\dagger a_j a_i = -a_b^\dagger a_q^\dagger a_j a_i$ and $a_a^\dagger a_b^\dagger a_p a_i = a_a^\dagger a_b^\dagger a_i a_p$, we can make the expression manifest antisymmetric, i.e.

$$
\begin{aligned}
[F, T_2]_{2b} &= \frac{1}{4} \sum_{qbij} \left[\sum_a \left(f_a^q t_{ij}^{ab} - f_a^b t_{ij}^{qa} \right) \right] a_q^\dagger a_b^\dagger a_j a_i \\
&- \frac{1}{4} \sum_{pabi} \left[\sum_j \left(f_p^j t_{ij}^{ab} - f_i^j t_{pj}^{ab} \right) \right] a_a^\dagger a_b^\dagger a_p a_i .
\end{aligned}
$$

Thus, the contribution of $[F, T_2]_{2b}$ to the matrix element $\overline{H}_{ij}^{ab}$ is

$$\overline{H}_{ij}^{ab} \leftarrow \sum_c \left(f_c^a t_{ij}^{cb} - f_c^b t_{ij}^{ac} \right) - \sum_k \left(f_j^k t_{ik}^{ab} - f_i^k t_{jk}^{ab} \right)$$

Here we used an arrow to indicate that this is just one contribution to this matrix element. We see that the derivation straight forward, but somewhat tedious. As no one likes to commute too much (neither in this example nor when going to and from work), and so we need a better approach. This is where diagramms come in handy.

===== =====

Diagrams. The pictures in this Subsection are taken from Crawford and Schaefer.

By convention, hole lines (labels $i, j, k, \ldots$) are pointing down.

By convention, particle lines (labels $a, b, c, \ldots$) are pointing up.

Let us look at the one-body operator of the normal-ordered Hamiltonian, i.e. Fock matrix. Its diagrams are as follows.

We now turn to the two-body interaction. It is denoted as a horizontal dashed line with incoming and outgoing lines attached to it. We start by noting that the following diagrams of the interaction are all related by permutation symmetry.

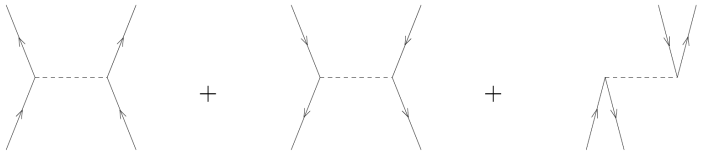
*

Exercise 10: Assign the correct matrix element $\langle pq|V|rs\rangle$ to each of the following diagrams of the interaction

Remember: $\langle \text{left} - \text{out}, \text{right} - \text{out}|\text{V}|\text{left} - \text{in}, \text{right} - \text{in}\rangle$.

aragraph!paragraph>paragraph>-0.5em

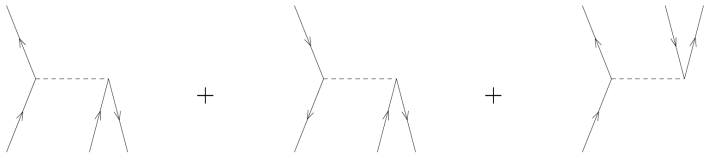
a)



Answer. $\langle ab|V|cd\rangle + \langle ij|V|kl\rangle + \langle ia|V|bj\rangle$

aragraph!paragraph>paragraph>-0.5em

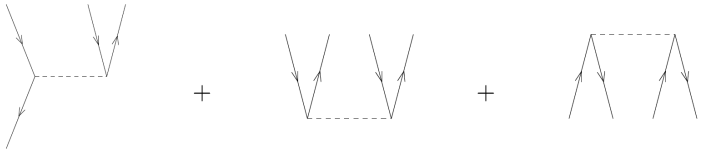
b)



Answer. $\langle ai|V|bc\rangle + \langle ij|V|ka\rangle + \langle ab|V|ci\rangle$

aragraph!paragraph>paragraph>-0.5em

c)



Answer. $\langle ia|V|jk\rangle + \langle ab|V|ij\rangle + \langle ij|V|ab\rangle$

===== =====

Finally, we have the following diagrams for the T_1 and T_2 amplitudes.

We are now in the position to construct the diagrams of the similarity-transformed Hamiltonian, keeping in mind that these diagrams correspond to matrix elements of $\overline{H_N}$. The rules are as follows.

1. Write down all *topologically different* diagrams corresponding to the desired matrix element. Topologically different diagrams differ in the number and type of lines (particle or hole) that connect the Fock matrix F or the interaction V to the cluster amplitudes T, but not whether these connections are left or right (as those are related by antisymmetry). As an example, all diagrams in Fig. 1.8 are topologically identical, because they consist of incoming particle and hole lines and of outgoing particle and hole lines.

2. Write down the matrix elements that enter the diagram, and sum over all internal lines.

3. The overall sign is (-1) to the power of [(number of hole lines) – (number of loops)].

4. Symmetry factor: For each pair of equivalent lines (i.e. lines that connect the same two operators) multiply with a factor 1/2. For n identical vertices, multiply the algebraic expression by the symmery factor $1/n!$ to account properly for the number of ways the diagram can be constructed.

5. Antisymmetrize the outgoing and incoming lines as necessary.

Please note that this really works. You could derive these rules for yourself from the commutations and factors that enter the Baker-Campbell-Hausdorff expansion. The sign comes obviously from the arrangement of creation and annilhilation operators, while the symmetry factor stems from all the different ways, one can contract the cluster operator with the normal-ordered Hamiltonian.

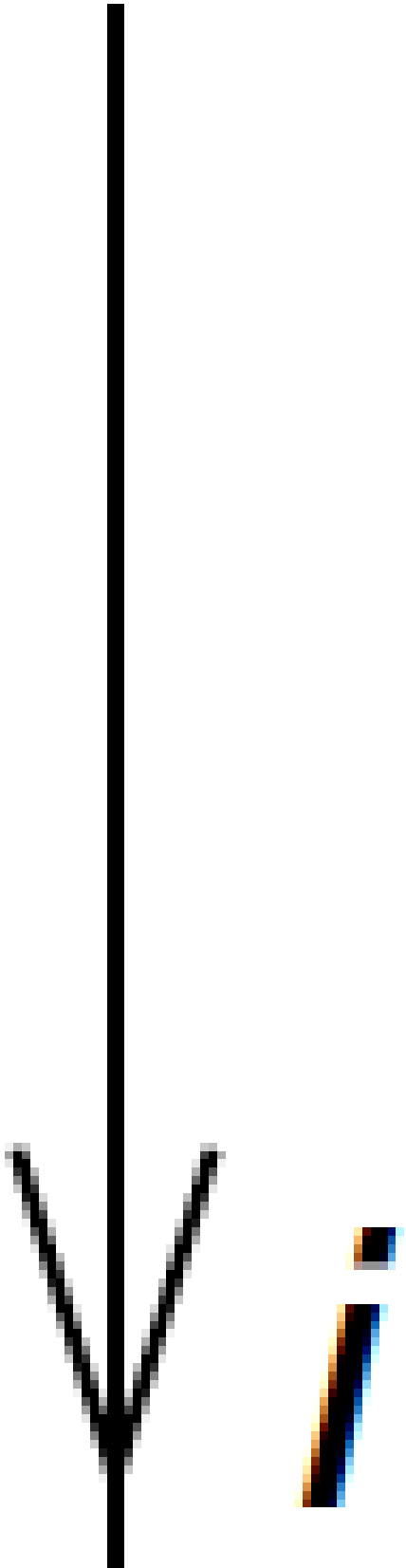
Example: CCSD correlation energy

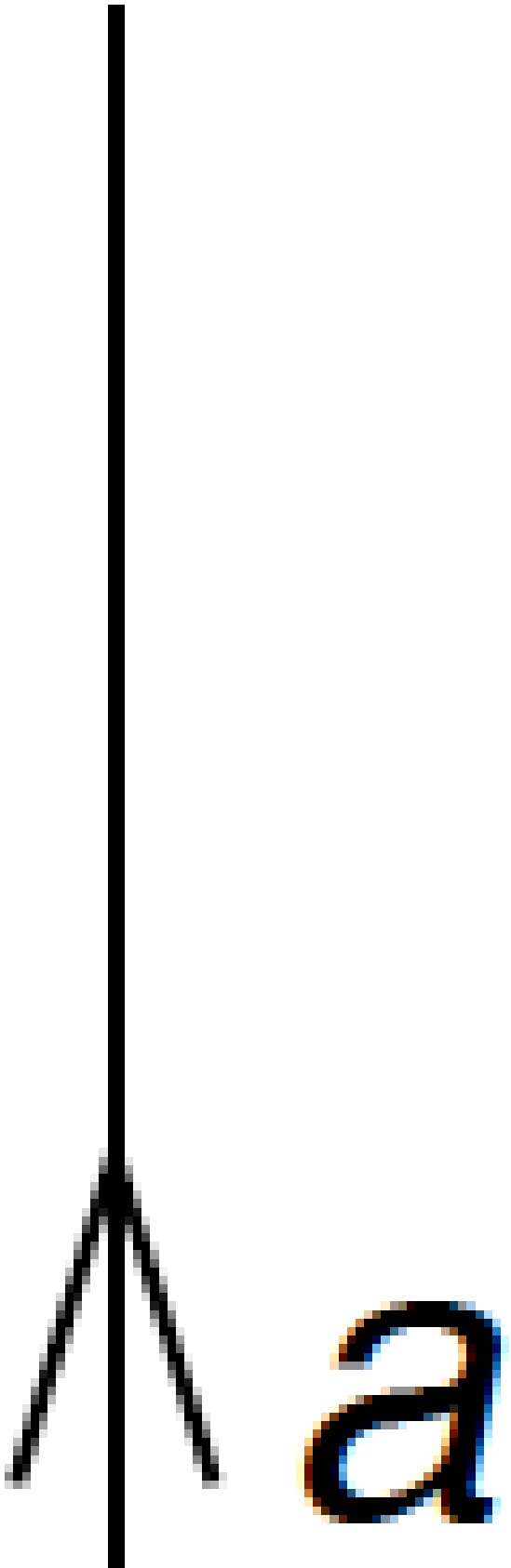
The CCSD correlation energy, $E_c = \langle\Phi_0|\overline{H_N}|\Phi_0\rangle$, is the first of the CCSD equations (1.20). It is a vacuum expectation value and thus consists of all diagrams with no external legs. There are three such diagrams:

The correponding algebraic expression is $E_c = \sum_{ia} f_a^i t_i^a + \frac{1}{4}\sum_{ijab}\langle ij|V|ab\rangle t_{ij}^{ab} + \frac{1}{2}\sum_{ijab}\langle ij|V|ab\rangle t_i^a t_j^b$.

The first algebraic expression is clear. We have one hole line and one loop, giving it a positive sign. There are no equivalent lines or vertices, giving it no symmetry factor. The second diagram has two loops and two hole lines, again leading to a positive sign. We have a pair of equivalent hole lines and a pair of equivalent particle lines, each giving a symmetry factor of 1/2. The third diagram has two loops and two hole lines, again leading to a positive sign. We have two indentical vertices (each connecting to a T_1 in the same way) and thus a symmetry factor 1/2.

===== =====





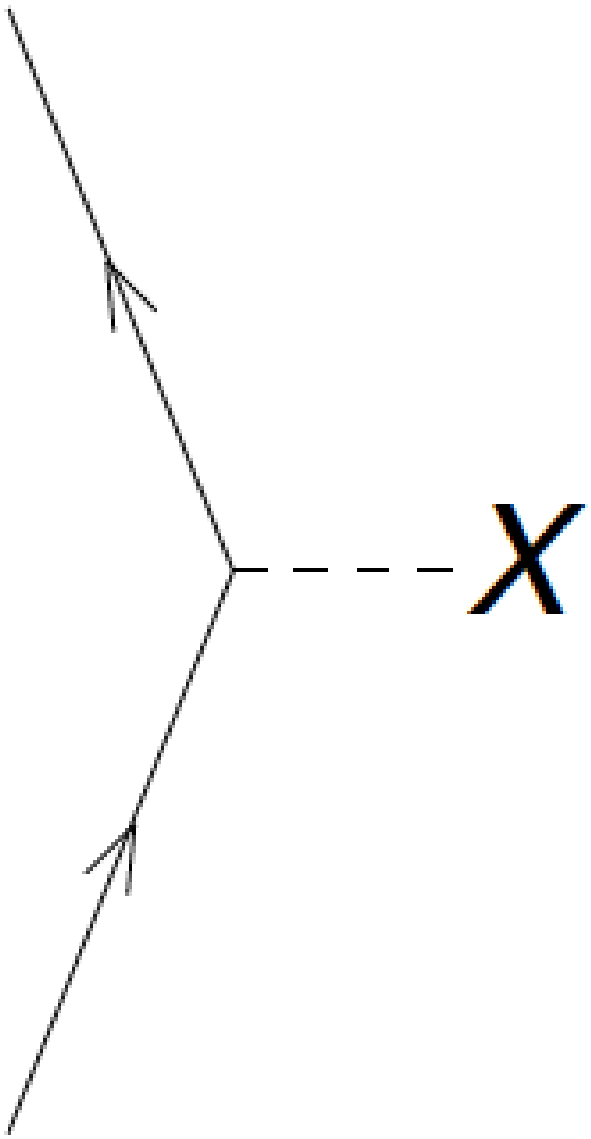


Figure 1.4: The diagrams corresponding to f_a^b. The dashed line with the 'X' denotes the interaction F between the incoming and outgoing lines. The labels a and b are not denoted, but you should label the outgoing and incoming lines accordingly.

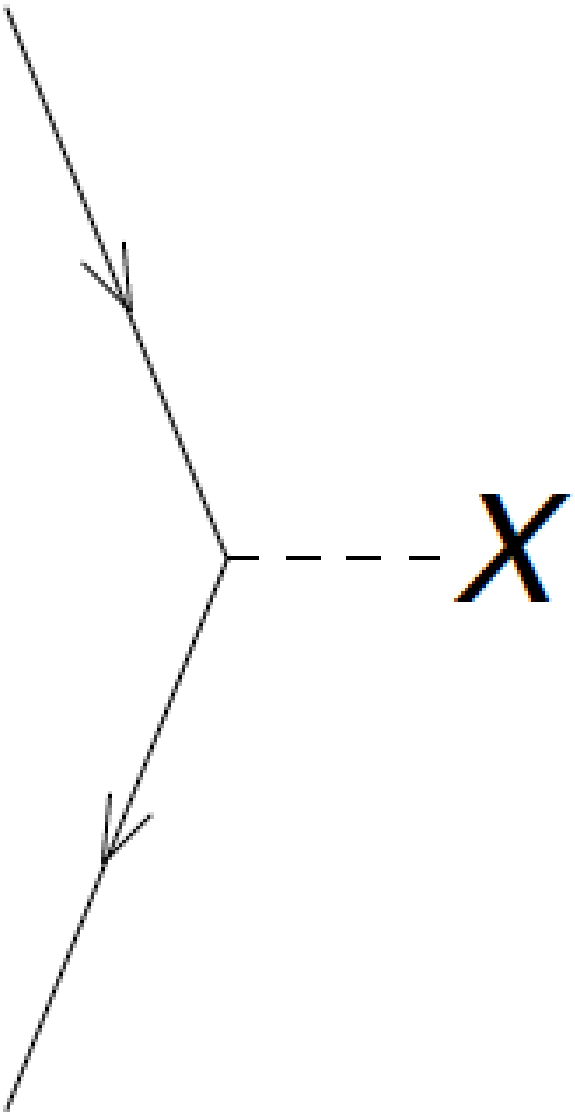


Figure 1.5: The diagrams corresponding to f_i^j. The dashed line with the 'X' denotes the interaction F between the incoming and outgoing lines.

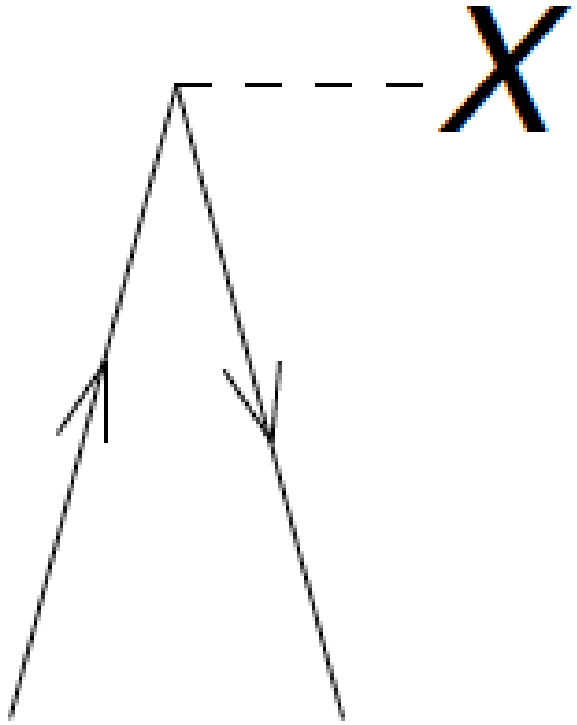


Figure 1.6: The diagrams corresponding to f_a^i. The dashed line with the 'X' denotes the interaction F between the incoming and outgoing lines.

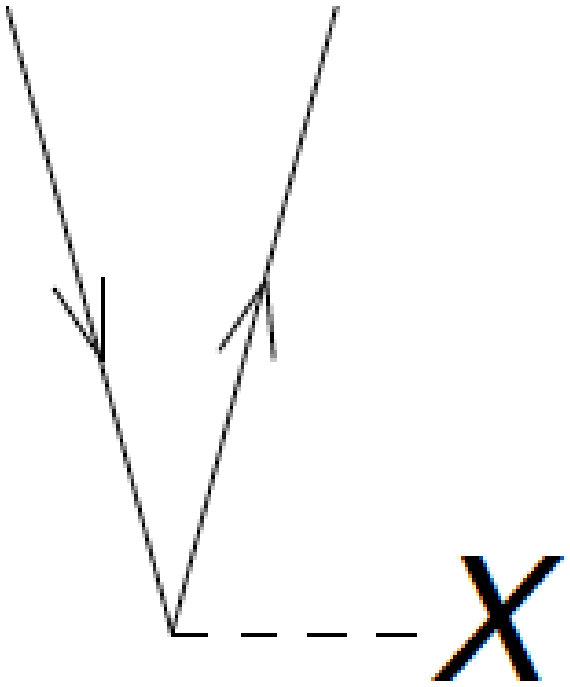


Figure 1.7: The diagrams corresponding to f_i^a. The dashed line with the 'X' denotes the interaction F between the incoming and outgoing lines.

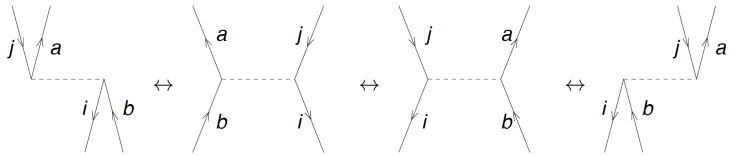


Figure 1.8: The diagrams corresponding to $\langle ai|V|jb\rangle = -\langle ai|V|bj\rangle = -\langle ia|V|jb\rangle = \langle ia|V|bj\rangle$.

$$\hat{T}_1 = \sum_{ia} t_i^a \{a_a^\dagger a_i\} =$$

$$\hat{T}_2 = \frac{1}{4}\sum_{ijab} t_{ij}^{ab} \{a_a^\dagger a_b^\dagger a_j a_i\} =$$

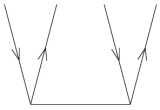


Figure 1.9: The horizontal full line is the cluster amplitude with incoming hole lines and outgoing particle lines as indicated.

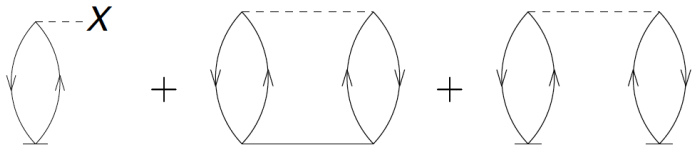


Figure 1.10: Three diagrams enter for the CCSD correlation energy, i.e. all diagrams that leave no external legs.

CCD Approximation

In what follows, we will consider the coupled cluster doubles (CCD) approximation. This approximation is valid in cases where the system cannot exhibit any particle-hole excitations (such as nuclear matter when formulated on a momentum-space grid) or for the pairing model (as the pairing interactions only excites pairs of particles). In this case $t_i^a = 0$ for all i, a, and $\overline{H}_i^a = 0$. The CCD approximation is also of some sort of leading order approximation in the Hartree-Fock basis (as the Hartree-Fock Hamiltonian exhibits no particle-hole excitations).

*

Exercise 11: Derive the CCD equations!

Let us consider the matrix element $\overline{H}_{ij}^{ab}$. Clearly, it consists of all diagrams (i.e. all combinations of T_2, and a single F or V that have two incoming hole lines and two outgoing particle lines. Write down all these diagrams.

Hint. Start systematically! Consider all combinations of F and V diagrams with 0, 1, and 2 cluster amplitudes T_2.

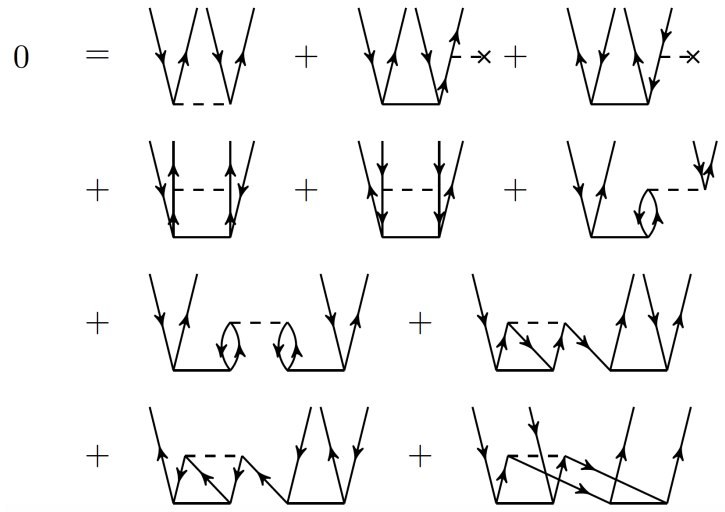


Figure 1.11: The diagrams for the T_2 equation, i.e. the matrix elements of $\overline{H}_{ij}^{ab}$. Taken from Baardsen et al (2013).

Answer. The corresponding algebraic expression is

$$\begin{aligned}\overline{H}_{ij}^{ab} &= \langle ab|V|ij\rangle + P(ab)\sum_c f_c^b t_{ij}^{ac} - P(ij)\sum_k f_j^k t_{ik}^{ab} \\ &+ \frac{1}{2}\sum_{cd}\langle ab|V|cd\rangle t_{ij}^{cd} + \frac{1}{2}\sum_{kl}\langle kl|V|ij\rangle t_{kl}^{ab} + P(ab)P(ij)\sum_{kc}\langle kb|V|cj\rangle t_{ik}^{ac} \\ &+ \frac{1}{2}P(ij)P(ab)\sum_{kcld}\langle kl|V|cd\rangle t_{ik}^{ac} t_{lj}^{db} + \frac{1}{2}P(ij)\sum_{kcld}\langle kl|V|cd\rangle t_{ik}^{cd} t_{lj}^{ab} \\ &+ \frac{1}{2}P(ab)\sum_{kcld}\langle kl|V|cd\rangle t_{kl}^{ac} t_{ij}^{db} + \frac{1}{4}\sum_{kcld}\langle kl|V|cd\rangle t_{ij}^{cd} t_{kl}^{ab}.\end{aligned}$$

===== =====

Let us now turn to the computational cost of a CCD computation.

*

Exercise 12: Computational scaling of CCD

For each of the diagrams in Fig. 1.11 write down the computational cost in terms of the number of occupied A and the number of unoccupied n_u orbitals.

Answer. The cost is $A^2n_u^2$, $A^2n_u^3$, $A^3n_u^2$, $A^2n_u^4$, $A^4n_u^2$, $A^3n_u^3$, $A^4n_u^4$, $A^4n_u^4$, $A^4n_u^4$, and $A^4n_u^4$ for the respective diagrams.

===== =====

Note that $n_u \gg A$ in general. In textbooks, one reads that CCD (and CCSD) cost only $A^2n_u^4$. Our most expensive diagrams, however are $A^4n_u^4$. What is going on?

To understand this puzzle, let us consider the last diagram of Fig. 1.11. We break up the computation into two steps, computing first the intermediate

$$\chi_{ij}^{kl} \equiv \frac{1}{2}\sum_{cd}\langle kl|V|cd\rangle t_{ij}^{cd} \tag{1.28}$$

at a cost of $A^4n_u^2$, and then

$$\frac{1}{2}\sum_{kl}\chi_{ij}^{kl} t_{kl}^{ab} \tag{1.29}$$

at a cost of $A^4n_u^2$. This is affordable. The price to pay is the storage of the intermediate χ_{ij}^{kl}, i.e. we traded memory for computational cycles. This trick is known as "factorization."

*

Exercise 13: Factorize the remaining diagrams of the CCD equation
Diagrams 7, 8, and 9 of Fig. 1.11 also need to be factorized.

Answer. For diagram number 7, we compute

$$\chi_{id}^{al} \equiv \sum_{kc} \langle kl|V|cd\rangle t_{ik}^{ac} \tag{1.30}$$

at a cost of $A^3 n_u^3$ and then compute

$$\frac{1}{2} P(ij)P(ab) \sum_{ld} \chi_{id}^{al} t_{lj}^{db} \tag{1.31}$$

at the cost of $A^3 n_u^3$.

For diagram number 8, we compute

$$\chi_i^l \equiv -\frac{1}{2} \sum_{kcd} \langle kl|V|cd\rangle t_{ik}^{cd} \tag{1.32}$$

at a cost of $A^3 n_u^2$, and then compute

$$-P(ij) \sum_{l} \chi_i^l t_{lj}^{ab} \tag{1.33}$$

at the cost of $A^3 n_u^2$.

For diagram number 9, we compute

$$\chi_d^a \equiv \frac{1}{2} \sum_{kcl} \langle kl|V|cd\rangle t_{kl}^{ac} \tag{1.34}$$

at a cost of $A^2 n_u^3$ and then compute

$$P(ab) \sum_{d} \chi_d^a t_{ij}^{db} \tag{1.35}$$

at the cost of $A^3 n_u^3$.

===== =====

We are now ready, to derive the full CCSD equations, i.e. the matrix elements of $\overline{H}_i^a$ and $\overline{H}_{ij}^{ab}$.

*

Project 14: (Optional) Derive the CCSD equations!

aragraph!paragraph>paragraph>-0.5em

a) Let us consider the matrix element $\overline{H}_i^a$ first. Clearly, it consists of all diagrams (i.e. all combinations of T_1, T_2, and a single F or V that have an incoming hole line and an outgoing particle line. Write down all these diagrams.

$$
\begin{aligned}
0 \;=\; & f_{ai} + \sum_c f_{ac} t_i^c - \sum_k f_{ki} t_k^a + \sum_{kc} \langle ka||ci\rangle t_k^c + \sum_{kc} f_{kc} t_{ik}^{ac} + \frac{1}{2}\sum_{kcd} \langle ka||cd\rangle t_{ki}^{cd} - \\
& \frac{1}{2}\sum_{klc} \langle kl||ci\rangle t_{kl}^{ca} - \sum_{kc} f_{kc} t_i^c t_k^a - \sum_{klc} \langle kl||ci\rangle t_k^c t_l^a + \sum_{kcd} \langle ka||cd\rangle t_k^c t_i^d - \\
& \sum_{klcd} \langle kl||cd\rangle t_k^c t_i^d t_l^a + \sum_{klcd} \langle kl||cd\rangle t_k^c t_{li}^{da} - \frac{1}{2}\sum_{klcd} \langle kl||cd\rangle t_{ki}^{cd} t_l^a - \frac{1}{2}\sum_{klcd} \langle kl||cd\rangle t_{kl}^{ca} t_i^d
\end{aligned}
$$

Figure 1.12: The diagrams for the T_1 equation, i.e. the matrix elements of $\overline{H}_i^a$. Taken from Crawford and Schaefer. Here $\langle pq||rs\rangle \equiv \langle pq|V|rs\rangle$ and $f_{pq} \equiv f_q^p$.

Answer.

aragraph!paragraph>paragraph>-0.5em

b) Let us now consider the matrix element $\overline{H}_{ij}^{ab}$. Clearly, it consists of all diagrams (i.e. all combinations of T_1, T_2, and a single F or V that have two incoming hole lines and two outgoing particle lines. Write down all these diagrams and corresponding algebraic expressions.

Answer. ===== =====

We can now turn to the solution of the coupled-cluster equations.

Solving the CCD equations

The CCD equations, depicted in Fig. 1.11, are nonlinear in the cluster amplitudes. How do we solve $\overline{H}_{ij}^{ab} = 0$? We subtract $(f_a^a + f_b^b - f_i^i - f_j^j)t_{ij}^{ab}$ from both sides of $\overline{H}_{ij}^{ab} = 0$ (because this term is contained in $\overline{H}_{ij}^{ab}$) and find

$$(f_i^i + f_j^j - f_a^a - f_b^b)t_{ij}^{ab} = (f_i^i + f_j^j - f_a^a - f_b^b)t_{ij}^{ab} + \overline{H}_{ij}^{ab}$$

Dividing by $(f_i^i + f_j^j - f_a^a - f_b^b)$ yields

$$t_{ij}^{ab} = t_{ij}^{ab} + \frac{\overline{H}_{ij}^{ab}}{f_i^i + f_j^j - f_a^a - f_b^b} \tag{1.36}$$

This equation is of the type $t = f(t)$, and we solve it by iteration, i.e. we start with a guess t_0 and iterate $t_{n+1} = f(t_n)$, and hope that this will converge

$$
\begin{aligned}
0 \;=\;& \langle ab||ij\rangle + \sum_c \left(f_{bc}t^{ac}_{ij} - f_{ac}t^{bc}_{ij}\right) - \sum_k \left(f_{kj}t^{ab}_{ik} - f_{ki}t^{ab}_{jk}\right) + \\
& \frac{1}{2}\sum_{kl}\langle kl||ij\rangle t^{ab}_{kl} + \frac{1}{2}\sum_{cd}\langle ab||cd\rangle t^{cd}_{ij} + P(ij)P(ab)\sum_{kc}\langle kb||cj\rangle t^{ac}_{ik} + \\
& P(ij)\sum_c \langle ab||cj\rangle t^c_i - P(ab)\sum_k \langle kb||ij\rangle t^a_k + \\
& \frac{1}{2}P(ij)P(ab)\sum_{klcd}\langle kl||cd\rangle t^{ac}_{ik}t^{db}_{lj} + \frac{1}{4}\sum_{klcd}\langle kl||cd\rangle t^{cd}_{ij}t^{ab}_{kl} - \\
& P(ab)\frac{1}{2}\sum_{klcd}\langle kl||cd\rangle t^{ac}_{ij}t^{bd}_{kl} - P(ij)\frac{1}{2}\sum_{klcd}\langle kl||cd\rangle t^{ab}_{ik}t^{cd}_{jl} + \\
& P(ab)\frac{1}{2}\sum_{kl}\langle kl||ij\rangle t^a_k t^b_l + P(ij)\frac{1}{2}\sum_{cd}\langle ab||cd\rangle t^c_i t^d_j - P(ij)P(ab)\sum_{kc}\langle kb||ic\rangle t^a_k t^c_j + \\
& P(ab)\sum_{kc} f_{kc}t^a_k t^{bc}_{ij} + P(ij)\sum_{kc} f_{kc}t^c_i t^{ab}_{jk} - \\
& P(ij)\sum_{klc}\langle kl||ci\rangle t^c_k t^{ab}_{lj} + P(ab)\sum_{kcd}\langle ka||cd\rangle t^c_k t^{db}_{ij} + \\
& P(ij)P(ab)\sum_{kcd}\langle ak||dc\rangle t^d_i t^{bc}_{jk} + P(ij)P(ab)\sum_{klc}\langle kl||ic\rangle t^a_l t^{bc}_{jk} + \\
& P(ij)\frac{1}{2}\sum_{klc}\langle kl||cj\rangle t^c_i t^{ab}_{kl} - P(ab)\frac{1}{2}\sum_{kcd}\langle kb||cd\rangle t^a_k t^{cd}_{ij} -
\end{aligned}
$$

Figure 1.13: The diagrams for the T_2 equation, i.e. the matrix elements of $\overline{H}^{ab}_{ij}$. Taken from Crawford and Schaefer. Here $\langle pq||rs\rangle \equiv \langle pq|V|rs\rangle$, $f_{pq} \equiv f^p_q$, and $P(ab) = 1 - (a \leftrightarrow b)$ antisymmetrizes.

to a solution. We take the perturbative result

$$\left(t^{ab}_{ij}\right)_0 = \frac{\langle ab|V|ij\rangle}{f^i_i + f^j_j - f^a_a - f^b_b} \tag{1.37}$$

as a starting point, compute $\overline{H}^{ab}_{ij}$, and find a new t^{ab}_{ij} from the right-hand side of Eq. (1.36). We repeat this process until the amplitudes (or the CCD energy) converge.

CCD for the pairing Hamiltonian

You learned about the pairing Hamiltonian earlier in this school. Convince yourself that this Hamiltonian does not induce any 1p-1h excitations. Let us solve the CCD equations for this problem. This consists of the following steps

1. Write a function that compute the potential, i.e. it returns a four-indexed array (or tensor). We need $\langle ab|V|cd\rangle$, $\langle ij|V|kl\rangle$, and $\langle ab|V|ij\rangle$. Why is there no $\langle ab|V|id\rangle$ or $\langle ai|V|jb\rangle$?

2. Write a function that computes the Fock matrix, i.e. a two-indexed array. We only need f_a^b and f_i^j. Why?

3. Initialize the cluster amplitudes according to Eq. (1.37), and solve Eq. (1.36) by iteration. The cluster amplitudes T_1 and T_2 are two- and four-indexed arrays, respectively.

Please note that the contraction of tensors (i.e. the summation over common indices in products of tensors) is very user friendly and elegant in python when `numpy.einsum` is used.

*

Project 15: Solve the CCD equations for the pairing problem
The Hamiltonian is

$$H = \delta \sum_{p=1}^{\Omega} (p-1)\left(a_{p+}^\dagger a_{p+} + a_{p-}^\dagger a_{p-}\right) - \frac{g}{2}\sum_{p,q=1}^{\Omega} a_{p+}^\dagger a_{p-}^\dagger a_{q-} a_{q+}. \tag{1.38}$$

Check your results and reproduce Fig 8.5 and Table 8.12 from Lecture Notes in Physics 936.

Answer. Click for IPython notebook for FCI and CCD solutions

```
## Coupled clusters in CCD approximation
## Implemented for the pairing model of Lecture Notes in Physics 936, Chapter 8.
## Thomas Papenbrock, June 2018

import numpy as np


def init_pairing_v(g,pnum,hnum):
    """
    returns potential matrices of the pairing model in three relevant channels

    param g: strength of the pairing interaction, as in Eq. (8.42)
    param pnum: number of particle states
    param hnum: number of hole states

    return v_pppp, v_pphh, v_hhhh: np.array(pnum,pnum,pnum,pnum),
                                   np.array(pnum,pnum,hnum,hnum),
                                   np.array(hnum,hnum,hnum,hnum),
```

```
                                        The interaction as a 4-indexed tensor in three channels.
    """
    v_pppp=np.zeros((pnum,pnum,pnum,pnum))
    v_pphh=np.zeros((pnum,pnum,hnum,hnum))
    v_hhhh=np.zeros((hnum,hnum,hnum,hnum))

    gval=-0.5*g
    for a in range(0,pnum,2):
        for b in range(0,pnum,2):
            v_pppp[a,a+1,b,b+1]=gval
            v_pppp[a+1,a,b,b+1]=-gval
            v_pppp[a,a+1,b+1,b]=-gval
            v_pppp[a+1,a,b+1,b]=gval

    for a in range(0,pnum,2):
        for i in range(0,hnum,2):
            v_pphh[a,a+1,i,i+1]=gval
            v_pphh[a+1,a,i,i+1]=-gval
            v_pphh[a,a+1,i+1,i]=-gval
            v_pphh[a+1,a,i+1,i]=gval

    for j in range(0,hnum,2):
        for i in range(0,hnum,2):
            v_hhhh[j,j+1,i,i+1]=gval
            v_hhhh[j+1,j,i,i+1]=-gval
            v_hhhh[j,j+1,i+1,i]=-gval
            v_hhhh[j+1,j,i+1,i]=gval

    return v_pppp, v_pphh, v_hhhh


def init_pairing_fock(delta,g,pnum,hnum):
    """
    initializes the Fock matrix of the pairing model

    param delta: Single-particle spacing, as in Eq. (8.41)
    param g: pairing strength, as in eq. (8.42)
    param pnum: number of particle states
    param hnum: number of hole states

    return f_pp, f_hh: The Fock matrix in two channels as numpy arrays np.array(pnum,pnum), np.ar
    """
# the Fock matrix for the pairing model. No f_ph needed, because we are in Hartree-Fock basis
    deltaval=0.5*delta
    gval=-0.5*g
    f_pp = np.zeros((pnum,pnum))
    f_hh = np.zeros((hnum,hnum))

    for i in range(0,hnum,2):
        f_hh[i  ,i  ] = deltaval*i+gval
        f_hh[i+1,i+1] = deltaval*i+gval

    for a in range(0,pnum,2):
        f_pp[a  ,a  ] = deltaval*(hnum+a)
        f_pp[a+1,a+1] = deltaval*(hnum+a)

    return f_pp, f_hh


def init_t2(v_pphh,f_pp,f_hh):
    """
```

```
    Initializes t2 amlitudes as in MBPT2, see first equation on page 345

    param v_pphh: pairing tensor in pphh channel
    param f_pp:   Fock matrix in pp channel
    param f_hh:   Fock matrix in hh channel

    return t2: numpy array in pphh format, 4-indices tensor
    """
    pnum = len(f_pp)
    hnum = len(f_hh)
    t2_new = np.zeros((pnum,pnum,hnum,hnum))
    for i in range(hnum):
        for j in range(hnum):
            for a in range(pnum):
                for b in range(pnum):
                    t2_new[a,b,i,j] = v_pphh[a,b,i,j] / (f_hh[i,i]+f_hh[j,j]-f_pp[a,a]-f_pp[b,b])
    return t2_new


# CCD equations. Note that the "->abij" assignment is redundant, because indices are ordered alpha
# Nevertheless, we retain it for transparency.
def ccd_iter(v_pppp,v_pphh,v_hhhh,f_pp,f_hh,t2):
    """
    Performs one iteration of the CCD equations (8.34), using also intermediates for the nonlinia

    param v_pppp: pppp-channel pairing tensor, numpy array
    param v_pphh: pphh-channel pairing tensor, numpy array
    param v_hhhh: hhhh-channel pairing tensor, numpy array
    param f_pp: Fock matrix in pp channel
    param f_hh: Fock matrix in hh channel
    param t2: Initial t2 amplitude, tensor in form of pphh channel

    return t2_new: new t2 amplitude, tensor in form of pphh channel
    """
    pnum = len(f_pp)
    hnum = len(f_hh)
    Hbar_pphh = (  v_pphh
                 + np.einsum('bc,acij->abij',f_pp,t2)
                 - np.einsum('ac,bcij->abij',f_pp,t2)
                 - np.einsum('abik,kj->abij',t2,f_hh)
                 + np.einsum('abjk,ki->abij',t2,f_hh)
                 + 0.5*np.einsum('abcd,cdij->abij',v_pppp,t2)
                 + 0.5*np.einsum('abkl,klij->abij',t2,v_hhhh)
                )

    # hh intermediate, see (8.47)
    chi_hh = 0.5* np.einsum('cdkl,cdjl->kj',v_pphh,t2)

    Hbar_pphh = Hbar_pphh - (  np.einsum('abik,kj->abij',t2,chi_hh)
                             - np.einsum('abik,kj->abji',t2,chi_hh) )

    # pp intermediate, see (8.46)
    chi_pp = -0.5* np.einsum('cdkl,bdkl->cb',v_pphh,t2)

    Hbar_pphh = Hbar_pphh + (  np.einsum('acij,cb->abij',t2,chi_pp)
                             - np.einsum('acij,cb->baij',t2,chi_pp) )

    # hhhh intermediate, see (8.48)
    chi_hhhh = 0.5 * np.einsum('cdkl,cdij->klij',v_pphh,t2)

    Hbar_pphh = Hbar_pphh + 0.5 * np.einsum('abkl,klij->abij',t2,chi_hhhh)
```

```
    # phph intermediate, see (8.49)
    chi_phph= + 0.5 * np.einsum('cdkl,dblj->bkcj',v_pphh,t2)


    Hbar_pphh = Hbar_pphh + (  np.einsum('bkcj,acik->abij',chi_phph,t2)
                             - np.einsum('bkcj,acik->baij',chi_phph,t2)
                             - np.einsum('bkcj,acik->abji',chi_phph,t2)
                             + np.einsum('bkcj,acik->baji',chi_phph,t2) )

    t2_new=np.zeros((pnum,pnum,hnum,hnum))
    for i in range(hnum):
        for j in range(hnum):
            for a in range(pnum):
                for b in range(pnum):
                    t2_new[a,b,i,j] = (  t2[a,b,i,j]
                                       + Hbar_pphh[a,b,i,j] / (f_hh[i,i]+f_hh[j,j]-f_pp[a,a]-f_pp

    return t2_new


def ccd_energy(v_pphh,t2):
    """
    Computes CCD energy. Call as
    energy = ccd_energy(v_pphh,t2)

    param v_pphh: pphh-channel pairing tensor, numpy array
    param t2: t2 amplitude, tensor in form of pphh channel

    return energy: CCD correlation energy
    """
    erg = 0.25*np.einsum('abij,abij',v_pphh,t2)
    return erg

################################
######## Main Program

# set parameters as for model
pnum = 20 # number of particle states
hnum = 10 # number of hole states
delta = 1.0

g = 0.5

print("parameters")
print("delta =", delta, ", g =", g)


# Initialize pairing matrix elements and Fock matrix
v_pppp, v_pphh, v_hhhh = init_pairing_v(g,pnum,hnum)
f_pp, f_hh = init_pairing_fock(delta,g,pnum,hnum)

# Initialize T2 amplitudes from MBPT2
t2 = init_t2(v_pphh,f_pp,f_hh)
erg = ccd_energy(v_pphh,t2)

# Exact MBPT2 for comparison, see last equation on page 365
exact_mbpt2 = -0.25*g**2*(1.0/(2.0+g) + 2.0/(4.0+g) + 1.0/(6.0+g))
print("MBPT2 energy =", erg, ", compared to exact:", exact_mbpt2)
```

```
# iterate CCD equations niter times
niter=200
mix=0.3
erg_old=0.0
eps=1.e-8
for iter in range(niter):
    t2_new = ccd_iter(v_pppp,v_pphh,v_hhhh,f_pp,f_hh,t2)
    erg = ccd_energy(v_pphh,t2_new)
    myeps = abs(erg-erg_old)/abs(erg)
    if myeps < eps: break
    erg_old=erg
    print("iter=", iter, "erg=", erg, "myeps=", myeps)
    t2 = mix*t2_new + (1.0-mix)*t2

print("Energy = ", erg)
```

Chapter 2

Nucleonic Matter

We want to compute nucleonic matter using coupled cluster or IMSRG methods, and start with considering the relevant symmetries.

*

Exercise 16: Which symmetries are relevant for nuclear matter?
aragraph!paragraph>paragraph>-0.5em

a) Enumerate continuous and discrete symmetries of nuclear matter.

Answer. The symmetries are the same as for nuclei. Continuous symmetries: translational and rotational invariance. Discrete symmetries: Parity and time reversal invariance.
aragraph!paragraph>paragraph>-0.5em

b) What basis should we use to implement these symmetries? Why do we have to make a choice between the two continuous symmetries? Which basis is most convenient and why?

Answer. Angular momentum and momentum do not commute. Thus, there is no basis that respects both symmetries simultaneously. If we choose the spherical basis, we are computing a spherical blob of nuclear matter and have to contend with surface effects, i.e. with finite size effects. We also need a partial-wave decomposition of the nuclear interaction. This approach has been done followed in ["Coupled-cluster studies of infinite nuclear matter, " G. Baardsen, A. Ekström, G. Hagen, M. Hjorth-Jensen, arXiv:1306.5681, Phys. Rev. C 88, 054312 (2013)]. If we choose a basis of discrete momentum states, translational invariance can be respected. This also facilitates the implementetation of modern nuclear interactions (which are often formulated in momentum space in effective

field theories). However, we have to think about the finite size effects imposed by periodic boundary conditions (or generalized Bloch waves). This approch was implemented in ["Coupled-cluster calculations of nucleonic matter," G. Hagen, T. Papenbrock, A. Ekström, K. A. Wendt, G. Baardsen, S. Gandolfi, M. Hjorth-Jensen, C. J. Horowitz, arXiv:1311.2925, Phys. Rev. C 89, 014319 (2014)].

===== =====

Basis states

In what follows, we employ a basis made from discrete momentum states, i.e. those states $|k_x, k_y, k_z\rangle$ in a cubic box of size L that exhibit periodic boundary conditions, i.e. $\psi_k(x + L) = \psi_k(x)$.

*

Exercise 17: Determine the basis states

What are the discrete values of momenta admissable in (k_x, k_y, k_z)?

Answer. In 1D position space $\psi_k(x) \propto e^{ikx}$ with $k = \frac{2\pi n}{L}$ and $n = 0, \pm 1, \pm 2, \ldots$ fulfill $\psi_k(x + L) = \psi_k(x)$.

===== =====

Thus, we use a cubic lattice in momentum space. Note that the momentum states e^{ikx} are not invariant under time reversal (i.e. under $k \to -k$), and also do not exhibit good parity ($x \to -x$). The former implies that the Hamiltonian matrix will in general be complex Hermitian and that the cluster amplitudes will in general be complex.

*

Exercise 18: How large should the basis be?

What values should be chosen for the box size L and for the maximum number $n_{\max}$, i.e. for the discrete momenta $k = \frac{2\pi n}{L}$ and $n = 0, \pm 1, \ \pm 2, \ldots, \pm n_{\max}$?

Answer. Usually $n_{\max}$ is fixed by computational cost, because we have $(2n_{\max}+1)^3$ basis states. We have used $n_{\max} = 4$ or up to $n_{\max} = 6$ in actual calculations to get converged results.

The maximum momentum must fulfill $k_{n_{\max}} > \Lambda$, where Λ is the momemtum cutoff of the interaction. This then fixes L for a given $n_{\max}$.

===== =====

Coupled cluster and IMSRG start from a Hartree-Fock reference state, and we need to think about this next. What are the magic numbers of a cubic lattice for neutron matter?

*

Exercise 19: Determine the lowest few magic numbers for a cubic lattice.

Answer. As the spin-degeneracy is $g_s = 2$, we have the magic numbers $g_s N$ with $N = 1, 7, 19, 27, 33, 57, 66, \ldots$ for neutrons.

===== =====

Given $n_{\max}$ and L for the basis parameters, we can choose a magic neutron number N. Clearly, the density of the system is then $\rho = N/L^3$. This summarizes the requirements for the basis. We choose $n_{\max}$ as large as possible, i.e. as large as computationally feasible. Then L and N are constrained by the UV cutoff and density of the system.

Finite size effects

We could also have considered the case of a more generalized boundary condition, i.e. $\psi_k(x + L) = e^{i\theta}\psi_k(x)$. Admissable momenta that fulfill such a boundary condition are $k_n(\theta) = \frac{2\pi n+\theta}{L}$. Avering over the "twist" angle θ removes finite size effects, because the discrete momenta are really drawn form a continuum. In three dimensions, there are three possible twist angles, and averaging over twist angles implies summing over many results corresponding to different angles. Thus, the removal of finite-size effects significantly increases the numerical expense. An example is shown in Figure 2.1, where we compute the kinetic energy per particle

$$T_N(\theta_x, \theta_y, \theta_z) = g_s \sum_{n_x, n_y, n_z \in N} \frac{\hbar^2 \left(k^2_{n_x}(\theta_x) + k^2_{n_y}(\theta_y) + k^2_{n_z}(\theta_z) \right)}{2m} \tag{2.1}$$

and compare with the infinite result $T_{\inf} = \frac{3}{10}\frac{\hbar^2 k_F^2}{m} N$ valid for the free Fermi gas. We clearly see strong shell effects (blue dashed line) and that averaging over the twist angles (red full line) very much reduces the shell oscillations. We also note that the neutron number 66 is quite attractive as it exhibits smaller finite size effects than other of the accessible magic numbers.

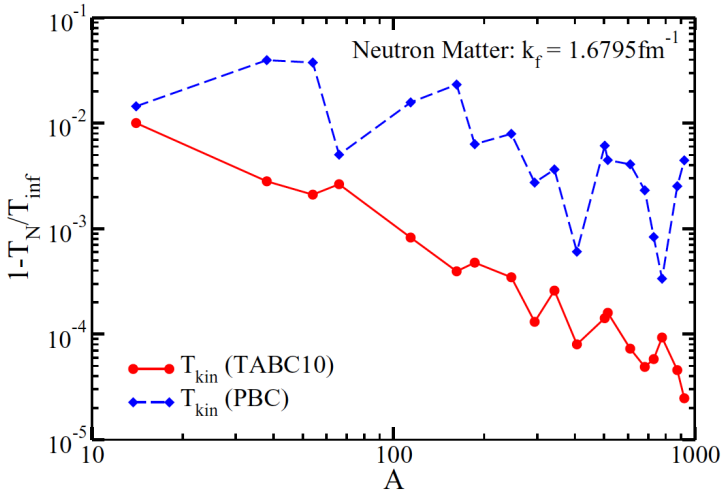


Figure 2.1: Relative finite-size corrections for the kinetic energy in pure neutron matter at the Fermi momentum $k_F = 1.6795\text{fm}^{-1}$ versus the neutron number A. TABC10 are twist-averaged boundary conditions with 10 Gauss-Legendre points in each spatial direction.

Channel structure of Hamiltonian and cluster amplitudes

Good quantum numbers for the nuclear interaction (i.e. operators that commute with the Hamiltonian and with each other) are total momentum, and the number of neutrons and protons, and – for simple interactions – also the spin (this is really spin, not orbital anular momentum or total angular momentum, as the latter two do not commute with momentum). Thus the Hamiltonian (and the cluster amplitudes) will consist of blocks, one for each set of quantum numbers. We call the set of quantum numbers that label each such block as a "channel." As the interaction is block diagonal, a numerically efficient implementation of nuclear matter has to take advantage of this channel structure. In fact, neutron matter cannot be computed in a numerically converged way (i.e. for large enough $n_{\max}$) if one does not exploit the channel structure.

The Hamiltonian is of the structure

$$H = \sum_{\vec{k},\sigma} \varepsilon^{\vec{k},\sigma}_{\vec{k},\sigma} a^\dagger_{\vec{k},\sigma} a_{\vec{k},\sigma} + \sum_{\vec{Q},\vec{p},\vec{k},\sigma_s} V^{\sigma_3\sigma_4}_{\sigma_1\sigma_2}(\vec{p},\vec{k}) a^\dagger_{\vec{Q}/2+\vec{p},\sigma_3} a^\dagger_{\vec{Q}/2-\vec{p},\sigma_4} a_{\vec{Q}/2-\vec{k},\sigma_2} a_{\vec{Q}/2+\vec{k},\sigma_1} \tag{2.2}$$

with $\varepsilon^{\vec{k},\sigma}_{\vec{k},\sigma} = \frac{k^2}{2m}$. In Eq. (2.2) we expressed the single-particle momenta in terms of center-of-mass momentum $\vec{Q}$ and relative momenta $(\vec{k},\vec{p})$, i.e. the incoming momenta $(\vec{k}_1,\vec{k}_2)$ and outgoing momenta $(\vec{k}_3,\vec{k}_3)$ are

$$\vec{k}_1 = \vec{Q}/2 + \vec{k}, \tag{2.3}$$
$$\vec{k}_2 = \vec{Q}/2 - \vec{k}, \tag{2.4}$$
$$\vec{k}_3 = \vec{Q}/2 + \vec{p}, \tag{2.5}$$
$$\vec{k}_4 = \vec{Q}/2 - \vec{p}. \tag{2.6}$$

The conservation of momentum is obvious in the two-body interaction as both the annihilation operators and the creation operators depend on the same center-of-mass momentum $\vec{Q}$. We note that the two-body interaction V depends only on the relative momenta $(\vec{k},\vec{p})$ but not on the center-of-mass momentum. We also note that a local interaction (i.e. an interaction that is multiplicative in position space) depends only on the momentum transfer $\vec{k}-\vec{p}$. The spin projections $\pm 1/2$ are denoted as σ.

Thus, the T_2 operator is

$$T_2 = \frac{1}{4}\sum_{\vec{Q},\vec{p},\vec{k},\sigma_s} t^{\sigma_3\sigma_4}_{\sigma_1\sigma_2}(Q;\vec{p},\vec{k}) a^\dagger_{\vec{Q}/2+\vec{p},\sigma_3} a^\dagger_{\vec{Q}/2-\vec{p},\sigma_4} a_{\vec{Q}/2-\vec{k},\sigma_2} a_{\vec{Q}/2+\vec{k},\sigma_1}. \tag{2.7}$$

We note that the amplitude $t^{\sigma_3\sigma_4}_{\sigma_1\sigma_2}(Q;\vec{p},\vec{k})$ depends on the center-of-mass momentum $\vec{Q}$, in contrast to the potential matrix element $V^{\sigma_3\sigma_4}_{\sigma_1\sigma_2}(\vec{p},\vec{k})$.

In the expressions (2.2) and (2.7) we supressed that $\sigma_1+\sigma_2 = \sigma_3+\sigma_4$. So, a channel is defined by $\vec{Q}$ and total spin projection $\sigma_1+\sigma_2$.

Because of this channel structure, the simple solution we implemented for the pairing problem cannot be really re-used when computing neutron matter. Let us take a look at the Minnesota potential

$$V(r) = \left(V_R(r) + \frac{1}{2}(1+P^\sigma_{12})V_T(r) + \frac{1}{2}(1-P^\sigma_{12})V_S(r)\right)\frac{1}{2}(1-P^\sigma_{12}P^\tau_{12}). \tag{2.8}$$

Here,

$$P^\sigma_{12} = \frac{1}{2}(1+\vec{\sigma}_1\cdot\vec{\sigma}_2),$$
$$P^\tau_{12} = \frac{1}{2}(1+\vec{\tau}_1\cdot\vec{\tau}_2) \tag{2.9}$$

are spin and isospin exchange operators, respectively, and $\vec{\sigma}$ and $\vec{\tau}$ are vectors of Pauli matrices in spin and isospin space, respectively. Thus,

$$\frac{1}{2}(1 - P_{12}^{\sigma} P_{12}^{\tau}) = |S_{12} = 0, T_{12} = 1\rangle\langle S_{12} = 0, T_{12} = 1| + |S_{12} = 1, T_{12} = 0\rangle\langle S_{12} = 1, T_{12} = 0| \tag{2.10}$$

projects onto two-particle spin-isospin states as indicated, while

$$\frac{1}{2}(1 - P_{12}^{\sigma}) = |S_{12} = 0\rangle\langle S_{12} = 0|, \tag{2.11}$$

$$\frac{1}{2}(1 + P_{12}^{\sigma}) = |S_{12} = 1\rangle\langle S_{12} = 1| \tag{2.12}$$

project onto spin singlet and spin triplet combinations, respectively. For neutron matter, two-neutron states have isospin $T_{12} = 1$, and the Minnesota potential has no triplet term V_T. For the spin-exchange operator (and spins $s_1, s_2 = \pm 1/2$), we have $P_{12}^{\sigma}|s_1 s_2\rangle = |s_2 s_1\rangle$. For neutron matter, $P_{12}^{\tau} = 1$, because the states are symmetric under exchange of isospin. Thus, the Minnesota potential simplifies significantly for neutron matter as V_T does not contribute.

We note that the spin operator has matrix elements

$$\langle s_1' s_2'|\frac{1}{2}(1 - P_{12}^{\sigma})|s_1 s_2\rangle = \frac{1}{2}\left(\delta_{s_1}^{s_1'}\delta_{s_2}^{s_2'} - \delta_{s_1}^{s_2'}\delta_{s_2}^{s_1'}\right). \tag{2.13}$$

The radial functions are

$$V_R(r) = V_R e^{-\kappa_R r^2}, \tag{2.14}$$

$$V_S(r) = V_S e^{-\kappa_S r^2}, \tag{2.15}$$

$$V_T(r) = V_T e^{-\kappa_T r^2}, \tag{2.16}$$

$$\tag{2.17}$$

and the parameters are as follows

α	V_α	κ_α
R	+200 MeV	1.487 fm $^{-2}$
S	-91.85 MeV	0.465 fm $^{-2}$
T	-178 MeV	0.639 fm $^{-2}$

Note that $\kappa_\alpha^{1/2}$ sets the momentum scale of the Minnesota potential. We see that we deal with a short-ranged repulsive core (the V_R term) and longer ranged attractive terms in the singlet (the term V_S) and triplet (the term V_T) channels.

A Fourier transform (in the finite cube of length L) yields the momentum-space form of the potential

$$\langle k_p k_q|V_\alpha|k_r k_s\rangle = \frac{V_\alpha}{L^3}\left(\frac{\pi}{\kappa_\alpha}\right)^{3/2} e^{-\frac{q^2}{4\kappa_\alpha}}\delta_{k_p+k_q}^{k_r+k_s}. \tag{2.18}$$

Here, $q \equiv \frac{1}{2}(k_p - k_q - k_r + k_s)$ is the momentum transfer, and the momentum conservation $k_p + k_q = k_r + k_s$ is explicit.

As we are dealing only with neutrons, the potential matrix elements (including spin) are for $\alpha = R, S$

$$\langle k_p s_p k_q s_q | V_\alpha | k_r s_r k_s s_s \rangle = \langle k_p k_q | V_\alpha | k_r k_s \rangle \frac{1}{2} \left(\delta^{s_r}_{s_p} \delta^{s_s}_{s_q} - \delta^{s_s}_{s_p} \delta^{s_r}_{s_q} \right), \tag{2.19}$$

and it is understood that there is no contribution from V_T. Please note that the matrix elements (2.19) are not yet antisymmetric under exchange, but $\langle k_p s_p k_q s_q | V_\alpha | k_r s_r k_s s_s \rangle - \langle k_p s_p k_q s_q | V_\alpha | k_s s_s k_r s_r \rangle$ is.

Example: Channel structure and its usage

We have single-particle states with momentum and spin, namely

$$|r\rangle \equiv |k_r s_r\rangle. \tag{2.20}$$

Naively, two-body states are then

$$|rs\rangle \equiv |k_r s_r k_s s_s\rangle, \tag{2.21}$$

but using the center-of-mass transformation (2.3) we can rewrite

$$|rs\rangle \equiv |P_{rs} k_{rs} s_r s_s\rangle, \tag{2.22}$$

where $P_{rs} = k_r + k_s$ is the total momentum and $k_{rs} = (k_r - k_s)/2$ is the relative momentum. This representation of two-body states is well adapated to our problem, because the interaction and the T_2 amplitudes preserve the total momentum. Thus, we store the cluster amplitudes t^{ab}_{ij} as matrices

$$t^{ab}_{ij} \to [t(P_{ij})]^{|k_{ab} s_a s_b\rangle}_{|k_{ij} s_i s_j\rangle} \equiv t^{|P_{ij} k_{ab} s_a s_b\rangle}_{|P_{ij} k_{ij} s_i s_j\rangle}, \tag{2.23}$$

and the conservation of total momentum is explicit.

Likewise, the pppp, pphh, and hhhh parts of the interaction can be written in this form, namely

$$V^{ab}_{cd} \to [V(P_{ab})]^{|k_{ab} s_a s_b\rangle}_{|k_{cd} s_c s_d\rangle} \equiv V^{|P_{ab} k_{ab} s_a s_b\rangle}_{|P_{ab} k_{cd} s_c s_d\rangle}, \tag{2.24}$$

$$V^{ab}_{ij} \to [V(P_{ij})]^{|k_{ab} s_a s_b\rangle}_{|k_{ij} s_i s_j\rangle} \equiv V^{|P_{ij} k_{ab} s_a s_b\rangle}_{|P_{ij} k_{ij} s_i s_j\rangle}, \tag{2.25}$$

$$V^{kl}_{ij} \to [V(P_{ij})]^{|k_{kl} s_k s_l\rangle}_{|k_{ij} s_i s_j\rangle} \equiv V^{|P_{ij} k_{kl} s_k s_l\rangle}_{|P_{ij} k_{ij} s_i s_j\rangle}, \tag{2.26}$$

and we also have

$$\overline{H}^{ab}_{ij} \to \left[\overline{H}(P_{ij})\right]^{|k_{ab} s_a s_b\rangle}_{|k_{ij} s_i s_j\rangle} \equiv \overline{H}^{|P_{ij} k_{ab} s_a s_b\rangle}_{|P_{ij} k_{ij} s_i s_j\rangle}, \tag{2.27}$$

Using these objects, diagrams (1), (4), and (5) of Figure 1.11 can be done for each block of momentum P_{ij} as a copy and matrix-matrix multiplications, respectively

$$\left[\overline{H}(P_{ij})\right]^{|k_{ab}s_as_b\rangle}_{|k_{ij}s_is_j\rangle} = [V(P_{ij})]^{|k_{ab}s_as_b\rangle}_{|k_{ij}s_is_j\rangle} \tag{2.28}$$

$$+\frac{1}{2}\sum_{|k_{kl}s_ks_l\rangle}[t(P_{ij})]^{|k_{ab}s_as_b\rangle}_{|k_{kl}s_ks_l\rangle}\,[V(P_{ij})]^{|k_{kl}s_ks_l\rangle}_{|k_{ij}s_is_j\rangle} \tag{2.29}$$

$$+\frac{1}{2}\sum_{|k_{cd}s_cs_d\rangle}[V(P_{ij})]^{|k_{ab}s_as_b\rangle}_{|k_{cd}s_cs_d\rangle}\,[t(P_{ij})]^{|k_{cd}s_cs_d\rangle}_{|k_{ij}s_is_j\rangle}\,. \tag{2.30}$$

Similarly, the CCD correlation energy results from

$$E_c = \frac{1}{4}\sum_{P_{ij}}\sum_{|k_{ij}s_is_j\rangle}\sum_{|k_{ab}s_as_b\rangle}[t(P_{ij})]^{|k_{ab}s_as_b\rangle}_{|k_{ij}s_is_j\rangle}\,[V(P_{ij})]^{|k_{ab}s_as_b\rangle}_{|k_{ij}s_is_j\rangle} \tag{2.31}$$

These efficiences cannot be used for the sixth diagram of Figure 1.11. One could either change to a ph formulation, noting that $k_a - k_i = k_k - k_c$ is also a preserved quantity in t^{ac}_{ik} and that $k_k - k_c = k_j - k_b$ is preserved in V^{kb}_{cj}. Thus $\sum_{kc} t^{ac}_{ik}V^{kb}_{cj}$ has a conserved quantity $k_k - k_c$ in the loop, and we can again use matrix-matrix multiplications for this diagram. This requires us to store the T_2 amplitude in a phhp and in the usual pphh formulation. Alternatively, we could simply code this diagram with the loops over single-particle states. If this seems too tedious, one can also limit CCD to the first five diagrams in Figure 1.11 (these are the pp and hh ladders), which gives a good description for neutron matter, see the comparison between this and full CCD in ["Coupled-cluster calculations of nucleonic matter," G. Hagen, T. Papenbrock, A. Ekström, K. A. Wendt, G. Baardsen, S. Gandolfi, M. Hjorth-Jensen, C. J. Horowitz, arXiv:1311.2925, Phys. Rev. C 89, 014319 (2014)].

===== =====

The steps towards the solution of the CCD equations for neutron matter are as follows

1. For a given density and UV cutoff, set up the lattice, i.e. determine the single-particle basis.

2. Determine the channels allowed by the (Minnesota) interaction, i.e. sets of two-body states that are connected by the interaction.

3. Exploit this channel structure when computing the diagrams.

4. Solve the coupled-cluster equations. Here we start first with the pp and hh ladders, i.e. using only the first five diagrams of Figure 1.11.

*

Exercise 20: Write a CCD code for neutron matter, focusing first on ladder approximation, i.e. including the first five diagrams in Figure 1.11.

Answer. Click for IPython notebook

```
import numpy as np

#############################################################
# CCD Program for neutron matter with the Minnesota potential.
#
# Thomas Papenbrock, July/August 2018
#
# License: Free Software following Version 3 of GNU General Public License,
# see https://www.gnu.org/licenses/gpl.html
#######


##########################
# Class for neutron matter
#######

class MomSpaceBasis:
    """
    momentum-space basis class
    The constructor has the form

    MomSpaceBasis(Nmax,kmax)

    param Nmax: Number of lattice points in positive kx-direction
    param kmax: Highest lattice momentum (in 1/fm)

    return: MomSpaceBasis as a single-partcle basis.
    attributes of MomSpaceBasis are

    dk : lattice spacing in 1/fm
    Lbox : linear dimension (in fm) of cubic box
    nvec : lattice vectors (integers)
    kvec : lattice momentum vectors (floats, in 1/fm)
    ngrid : total number of lattice points
    """
    def __init__(self,Nmax,kmax,ordered=True):
        """
        the constructor

        Generates a cubic lattice in momentum space
        param Nmax: Number of lattice points in positive kx-direction
        param kmax: Highest lattice momentum (in 1/fm)
        param ordered: Optional parameter, True by default, will order lattice points by kinetic

        return MomSpaceBasis
        """
        self.Nmax = Nmax
        self.dim = 0
        self.ngrid = 0
        self._kvec=[]
        self._nvec=[]
```

```
        dk = kmax / Nmax
        self.dk = dk
        self.Lbox = 2.0*np.pi/dk

        nx=[]
        nvec=[]
        for i in range(-Nmax,Nmax+1):
            self.dim=self.dim+1
            nx.append(i)

        #print('nx=',nx)

        for i in nx:
            for j in nx:
                for k in nx:
                    nvec.append(np.array([i,j,k], dtype=int))

        #print('nvec=',nvec)
        self.ngrid=len(nvec)

        if ordered:
            #print("ordered")
            norm=np.zeros(self.ngrid,dtype=int)
            for i, vec in enumerate(nvec):
                npvec=np.array(vec,dtype=int)
                norm[i]=np.dot(npvec,npvec)
               # print(i, vec, norm[i])

            index=np.argsort(norm)
            #print(index)
            self._nvec=[]
            for i, ind in enumerate(index):
                #print(i, ind, nvec[ind])
                self._nvec.append(nvec[ind])

        else:
            self._nvec=nvec  # a list

        self._kvec = np.array(self._nvec)*dk # a numpy array


    def kvec(self,indx=-1):
        """
        MomSpaceBasis.kvec(i) returns ith momentum vector
        MomSpaceBasis.kvec() returns all momentum vectors

        param indx: index of k-vector to be returned, optional
        return 3-vector (if index non-negative), or all vectors if no index specified
        """
        if indx == -1:
            return self._kvec
        else:
            return self._kvec[indx]

    def nvec(self,indx=-1):
        """
        MomSpaceBasis.nvec(i) returns ith lattice vector
        MomSpaceBasis.nvec() returns all lattice vectors

        param indx: index of lattice vector to be returned, optional
        return 3-vector (if index non-negative), or all lattice vectors if no index specified
```

```
        """
        if indx == -1:
            return self._nvec
        else:
            return self._nvec[indx]

    def dens(self,num):
        """
        returns density of system if num particles are present
        param num: int, number of particles
        return dens: float
        """
        return num/(self.Lbox)**3

    def update(self,dk):
        """
        Uses dk as new lattice spacing and rescales existing lattice
        param dk: in 1/fm lattice spacing in momentum space
        """
        self.Lbox=2.0*np.pi/dk
        self._kvec = np.array(self._nvec)*dk

    def __len__(self):
        """
        overloading of the 'len' function
        """
        return self.ngrid


############
# useful functions

def magic_numbers(basis):
    """
    param basis: MomSpaceBasis object
    return magic: array of magic numbers
    """
    nvecs = basis.nvec()
    vec=np.array(nvecs[0],dtype=int)
    norm = np.dot(vec,vec)
    magic=[]
    for i in range(1,len(nvecs)):
        vec=np.array(nvecs[i],dtype=int)
        norm2 = np.dot(vec,vec)
        if norm2 > norm:
            magic.append(2*i)
            norm=norm2
    return magic


def get_dk(rho,Num):
    """
    param rho: desired density
    param Num: magic number of particles
    return dk: grid spacing in momentum space (in 1/fm)
    """
    Lbox = (Num/rho)**(1.0/3.0)
    dk = 2.0*np.pi/Lbox
    return dk

def spbasis_from_MomSpaceBasis(lattice_vecs,st_degen):
```

```
    """
    converts a lattice to a single particle basis for spin-isospin degeneracy st_degen
    param lattice_vecs: list of lattice vectors for 1st particle
    param st_degen: spin-isospin degeneracy
    return: basis as a list of momenta
    """
    if st_degen != 2: # for now only neutron matter
        print("Unexpected parameter st_degen")
        return lattice_vecs

    basis=[]
    for vec in lattice_vecs:
        for st in range(st_degen):
            basis.append(np.array(vec,dtype=int))

    return basis



############################################################
# Functions for comparisons with infinite free Fermi gas

def kF_from_density(rho,st_degen=2):
    """
    Computes Fermi momentum for given density and spin/isospin degeneracy.

    param rho: density in inverse fm cubed
    param st_degen: spin-isospin degeneracy; default 2
    return: Fermi momentum in inverse fm
    """
    res = (6.0*(np.pi)**2*rho/st_degen)**(1.0/3.0)
    return res

def EnergyDensity_FermiGas(kF,st_degen=2):
    """
    Computes energy density of free Fermi gas at Fermi momentum and spin/isospin degeneracy
    param kF: Fermi momentum in inverse fm
    param st_degen: spin-isospin degeneracy; default 2
    return: Energy density in MeV/fm**3
    """
    pvec = np.array([kF,0.0,0.0])
    erg = (st_degen*kF**3/(10.0*np.pi**2)) * Tkin(pvec)
    return erg


#############################################################################################
# Functions for CCD of neutron matter
# Implementation uses only pp and hh ladders
#
#############################################################################################


from numba import jit
# compile a few functions to gain speed; should probably done in Fortran or C++,
# and called from Python

@jit(nopython=True)
def minnesota_nn(p_out,s1_out,s2_out,p_in,s1_in,s2_in,Lbox):
    """
    The Minnesota potential between two neutrons, not yet anti-symmetrized
    param p_out: relative out momentum
```

```
    param p_in : relative in momentum
    param s1_out, s2_out: spin projections of out particles 1 and 2
    param s1_in, s2_in  : spin projections of in particles 1 and 2
    Lbox : size of momentum box
    return: value of potential in MeV; not anti-symmetrized!
    """
    # parameters. VT is not active between two neutrons (no triplet)
    VR = 200.0
    VS = -91.85  # sign typo in Lecture Notes Physics 936, Chap. 8
    kappaR = 1.487
    kappaS = 0.465

    qvec=p_out-p_in
    q2=np.dot(qvec,qvec)

    s1_i =spin2spinor(s1_in)
    s2_i =spin2spinor(s2_in)
    s1_o =spin2spinor(s1_out)
    s2_o =spin2spinor(s2_out)

    spin_part = 0.5 * ( np.dot(s1_i,s1_o)*np.dot(s2_i,s2_o)
                       -np.dot(s1_i,s2_o)*np.dot(s2_i,s1_o) )


    pot = spin_part * (  VR*np.exp(-0.25*q2/kappaR) / (Lbox*np.sqrt(kappaR))**3
                       + VS*np.exp(-0.25*q2/kappaS) / (Lbox*np.sqrt(kappaS))**3 )

    pot = pot*(np.sqrt(np.pi))**3


    return pot

@jit
def spin_of_index(i):
    """
    Even indices of the lattive have spin up, odds have spin down
    param i: index of sp_basis
    return: spin as +/- 1
    """
    spin = 1-2*np.remainder(i,2)
    return spin

@jit
def spin2spinor(s):
    """
    Makes a two-component spinor of an integer s
    param s: spin = +/- 1
    return: two-component numpy array [1,0] for up and [0,1] for down
    """
    up  =np.array([1.0,0.0])
    down=np.array([0.0,1.0])
    if s == 1:
        return up
    else:
        return down

@jit
def Tkin(pvec):
    """
    Kinetic energy for a momentum vector
    param pvec: 3-component numpy array in inverse fm
```

```
    return: kinetic energy of that momentum in MeV
    """
    nucleon_mass = 938.92
    hbarc = 197.33
# More precise numbers for neutron mass and hbar follow.
# For N=14, this yields E_HF = 10.3337 MeV per nucleon in HF. Benchmarked with Ragnar Stroberg.
#     nucleon_mass = 939.56563
#     hbarc = 197.3269718
    p2 = np.dot(pvec,pvec)
    res = 0.5*hbarc**2*p2/nucleon_mass
    return res

@jit
def compute_total_Tkin(Nocc,sp_basis,dk):
    """
    Computes total kinetic energy of reference state
    param Nocc, sp_basis, dk: particle number, integer s.p. lattice, delta k
    return: total kinetic energy
    """
    erg=0.0
    for i in range(Nocc):
        mom_vec = sp_basis[i]
        vec=np.array(mom_vec)*dk
        erg=erg+Tkin(vec)

    return erg



@jit
def Fock(pvec,s,sp_basis,Nocc,dk,Lbox):
    """
    Fock matrix of momentum pvec in hh space
    param pvec: 3-component numpy array in inverse fm
    param s: spin as +/- 1 of state
    param_sp_basis, Nocc, dk, Lbox : parameters of s.p. basis and system

    return: Fock matrix element = kinetic energy of that momentum in MeV
    """
    res = Tkin(pvec)

    dum=0.0
    for i in range(Nocc):
        vec=sp_basis[i]*dk
        si=spin_of_index(i)
        p_in = 0.5*(vec-pvec)
        p_out= p_in
        dum = dum + ( minnesota_nn(p_out,s,si, p_in, s,si,Lbox)
                     -minnesota_nn(p_out,s,si,-p_in,si, s,Lbox) )  #antisymmetrized Minnesota

    res = res+dum
    return res

def compute_E_HF_simple(Nocc,sp_basis,dk):
    """
    Computes HF energy of reference state
    param Nocc, sp_basis, dk: particle number, integer s.p. lattice, delta k
    return: total HF energy
    """
    erg=compute_total_Tkin(Nocc,sp_basis,dk)
```

```
    pot=0.0
    for i in range(Nocc):
        momi=sp_basis[i]*dk
        si = spin_of_index(i)
        for j in range(Nocc):
            momj=sp_basis[j]*dk
            sj = spin_of_index(j)
            p_rel = 0.5*(momi-momj)
            pot = pot + 0.5* (  minnesota_nn(p_rel,si,sj, p_rel,si,sj,Lbox)
                              - minnesota_nn(p_rel,si,sj,-p_rel,sj,si,Lbox) )

    erg = erg+pot
    return erg


def get_channels(sp_basis,start1,end1,start2,end2,identical,other_channels=None):
    """
    Returns channels for coupled cluster based on Minnesota potential
    param sp_Basis: A single-particle basis
    param start1: index to start for particle 1
    param end1: index to end for particle 1
    param start2: index to start for particle 2
    param end2: index to end for particle 2
    param identical: True for hh or pp, False for hp
    param other_channels: list of other channels to compare with
    return: channels, p_rel, t2amp. channels is a list of p12, where p12 is a momentum vector;
            p_rel is a nested list with relative momenta and spins for each channel
    """
    channel=[]
    p_rel=[]
    for i, mom_vecs1 in enumerate(sp_basis[start1:end1]):
        #vec1=np.array(mom_vecs1,dtype=int)
        vec1=mom_vecs1
        spin1=spin_of_index(i)

        for j, mom_vecs2 in enumerate(sp_basis[start2:end2]):
            if identical and i==j: continue  #Fortran cycle
            #vec2=np.array(mom_vecs2,dtype=int)
            vec2=mom_vecs2
            spin2=spin_of_index(j)

            p12 = vec1+vec2
            prel= vec1-vec2
            spins=np.array([spin1,spin2],dtype=int)
            ps=[prel,spins]

            new=True
            needed=True
            if other_channels is not None: #check whether we need this channel
                needed=False
                for chan_o in other_channels:
                    if (chan_o==p12).all():
                        needed=True
                        break
            if needed: #check whether this channel exists already
                for ipos, chan in enumerate(channel):
                    if (chan==p12).all():
                        new=False
                        break

            if needed and new:
```

```
                channel.append(p12)
                p_rel.append([ps])

            if needed and not new:
                p_rel[ipos].append(ps)

    return channel, p_rel



def setup_T2_amplitudes(sp_basis,NN,st_degen):
    """
    returns the t2 amplitudes and t2 channels
    param sp_basis: a sp_basis
    param NN: neutron number
    param st_degen: 2 for the moment, spin-isospin degeneracy
    return: hh_channels, pp_channels, p_rel_hh, p_rel_pp, t2amp
            these are the hh and pp channels of T2, lists of the relative momenta,
            and t2amps as a list of numpy arrays set to zero
    """
    num_states = len(sp_basis)

    hh_channels, p_rel_hh = get_channels(sp_basis,0,NN,0,NN,True)
    print('hh channels=', len(hh_channels))

    pp_channels, p_rel_pp = get_channels(sp_basis,NN,num_states,NN,num_states,True,hh_channels)
    print('pp channels=', len(pp_channels))

    if len(pp_channels) != len(hh_channels): print('pp and hh channels do not match')


    ordered_pp_channel=[]
    ordered_p_rel_pp=[]
    for i, chanhh in enumerate(hh_channels):
        for j, chanpp in enumerate(pp_channels):
            if (chanpp==chanhh).all():
                ordered_pp_channel.append(chanpp)
                ordered_p_rel_pp.append(p_rel_pp[j])
                break

    pp_channels = ordered_pp_channel
    p_rel_pp = ordered_p_rel_pp

    # set t2 amplitudes to zero in each channel
    t2amp = fill_pot(Lbox, dk, pp_channels, hh_channels, p_rel_pp, p_rel_hh, True)

    return hh_channels, pp_channels, p_rel_hh, p_rel_pp, t2amp

def fill_pot(Lbox, dk, channels_out, channels_in, p_rel_out, p_rel_in, T2amp=False):
    """
    Fills lists of matrices such as Vhhhh, Vpphh, Vpppp, t2_pphh
    param Lbox: Lbox
    param dk: dk
    param channels_out, channels_in: the channels we have
    param p_rel_out, p_rel_in: the list of [prel, [s1,s2]]
    param T2amp=False: Set to True if t2_pphh needs to be computed
    return: The object of desire as a list of numpy matrices.
            Contain matrix elements for potentials, zeros if T2amp=True is requested.
    """
    Vpot=[]
    for i, chan_in in enumerate(channels_in):
```

```
        dim_in = len(p_rel_in[i])
        dim_out= len(p_rel_out[i])
        Vpot_chan=np.zeros((dim_out,dim_in))
        if not T2amp:
            for ii, ps_i in enumerate(p_rel_in[i]):
                [pii, [s1, s2]] = ps_i
                pii = pii*dk*0.5
                for jj, ps_j in enumerate(p_rel_out[i]):
                    if dim_in == dim_out and jj > ii: continue
                    [pjj, [ss1, ss2]] = ps_j
                    pjj = pjj*dk*0.5
                    Vpot_chan[jj,ii] = ( minnesota_nn( pjj,ss1,ss2, pii,s1,s2,Lbox)
                                        -minnesota_nn(-pjj,ss2,ss1, pii,s1,s2,Lbox) )
                    if dim_in == dim_out : Vpot_chan[ii,jj] = Vpot_chan[jj,ii]

        Vpot.append(Vpot_chan)
    return Vpot


def init_V(Lbox, dk, hhchannels, ppchannels, p_relhh, p_relpp,zeros=False):
    """
    Sets up Vhhhh, Vpphh, and Vpppp.

    return: Vhhhh, Vpphh, Vpppp as a lists of numpy arrays
    """
    Vhhhh = fill_pot(Lbox, dk, hhchannels, hhchannels, p_relhh, p_relhh,zeros)
    Vpphh = fill_pot(Lbox, dk, ppchannels, hhchannels, p_relpp, p_relhh,zeros)
    Vpppp = fill_pot(Lbox, dk, ppchannels, ppchannels, p_relpp, p_relpp,zeros)

    return Vhhhh, Vpphh, Vpppp

@jit
def make_diagram(obj1,obj2,fac):
    """
    Makes diagrams for pp or hh ladders as matrix-matrix multiplications
    """
    hbar_pphh=[]
    dim1=len(obj1)
    for chan in range(dim1):
        mat1 = obj1[chan]
        mat2 = obj2[chan]
        hbar_pphh.append( fac*np.matmul(mat1,mat2) )

    return hbar_pphh


def make_diagrams2_3(t2_pphh,fabij):
    hbar_pphh=[]
    for i, t2_mat in enumerate(t2_pphh):
        f_mat = fabij[i]
        hbar_mat = t2_mat*f_mat
        hbar_pphh.append(hbar_mat)
    return hbar_pphh


def compute_hbar(v_pppp,v_pphh,v_hhhh,t2_pphh,fabij):
    diagram1 = v_pphh.copy()
    diagram23 = make_diagrams2_3(t2_pphh, fabij)
    diagram4 = make_diagram(v_pppp,t2_pphh,0.5)
    diagram5 = make_diagram(t2_pphh,v_hhhh,0.5)
```

```
    hbar_pphh=[]
    for i in range(len(t2_pphh)):
        mat = (  diagram1[i]
               + diagram23[i]
               + diagram4[i]
               + diagram5[i]  )
        hbar_pphh.append(mat)

    return hbar_pphh

def get_energy_denominator(hh_channels,p_rel_pp,p_rel_hh,sp_basis,Nocc,dk,Lbox):
    res=[]
    fabij=[]
    for i, Ptot in enumerate(hh_channels):
        dimhh=len(p_rel_hh[i])
        dimpp=len(p_rel_pp[i])
        res_mat = np.zeros((dimpp,dimhh))
        f_mat   = np.zeros((dimpp,dimhh))
        for ii, psh_rel in enumerate(p_rel_hh[i]):
            [pij, [si, sj]] = psh_rel
            p_i = (Ptot+pij)//2
            p_i = p_i + np.array([Nmax,Nmax,Nmax],dtype=int)
            p_j = (Ptot-pij)//2
            p_j = p_j + np.array([Nmax,Nmax,Nmax],dtype=int)
            ssi = (1-si)//2
            ssj = (1-sj)//2
            fii = fock_mtx4[p_i[0],p_i[1],p_i[2],ssi]
            fjj = fock_mtx4[p_j[0],p_j[1],p_j[2],ssj]
            for jj, psp_rel in enumerate(p_rel_pp[i]):
                [pab, [sa, sb]] = psp_rel
                p_a = (Ptot+pab)//2
                p_a = p_a + np.array([Nmax,Nmax,Nmax],dtype=int)
                p_b = (Ptot-pab)//2
                p_b = p_b + np.array([Nmax,Nmax,Nmax],dtype=int)
                ssa = (1-sa)//2
                ssb = (1-sb)//2
                faa = fock_mtx4[p_a[0],p_a[1],p_a[2],ssa]
                fbb = fock_mtx4[p_b[0],p_b[1],p_b[2],ssb]

                res_mat[jj,ii] = 1.0 / (fii + fjj - faa - fbb)
                f_mat[jj,ii] = faa + fbb - fii - fjj
        res.append(res_mat)
        fabij.append(f_mat)
    return res, fabij


def get_t2_from_mbpt(Vpphh,denom):
    """
    param Vpphh: Vpphh
    param denom: energy denominator in pphh format
    return t2: quotient of both, element for element
    """
    res = []
    for i, vv in enumerate(Vpphh):
        dd = denom[i]
        res_mat = vv*dd  #how simple in python; element by element multiply
        res.append(res_mat)
    return res


def compute_E_CCD(Vpphh,T2pphh):
```

```
    erg=0.0
#     erg2=0.0
    for i, t2mat in enumerate(T2pphh):
        vmat  = Vpphh[i]
        erg = erg + 0.25*np.sum(vmat*t2mat)
    return erg


def compute_Fock_4(sp_basis,Nocc,dk,Lbox,Nmax):
    fock_mtx4=np.zeros(shape=(2*Nmax+1, 2*Nmax+1, 2*Nmax+1, 2))
    for i, vec in enumerate(sp_basis):
        pvec=vec*dk
        spin=spin_of_index(i)
        si = (1 - spin)//2
        px=vec[0]+Nmax
        py=vec[1]+Nmax
        pz=vec[2]+Nmax
        fock_mtx4[px,py,pz,si] = Fock(pvec,spin,sp_basis,Nocc,dk,Lbox)
    return fock_mtx4

#####################################
########### Main Program starts here

from timeit import default_timer as timer
# for timing purposes

progstart=timer()

Nmax=1
kmax=1.0
mbase = MomSpaceBasis(Nmax,kmax)
lattice=mbase.nvec()

## set particle number
NN=14
st_degen=2  # spin up and down
print("chosen N =", NN)
print("magic numbers", magic_numbers(mbase))

## set density
rho=0.08

dk = get_dk(rho,NN)

mbase.update(dk)
Lbox = mbase.Lbox


## get single particle basis

sp_basis = spbasis_from_MomSpaceBasis(lattice,st_degen)
num_states = len(sp_basis)

print('number of s.p. states:', num_states)

# print out a few facts of the reference state
total_Tkin = compute_total_Tkin(NN,sp_basis,dk)
print('total Tkin per particle =', total_Tkin/NN )

k_fermi = kF_from_density(rho)
```

```
print("Fermi momentum =", k_fermi)

E_gas = EnergyDensity_FermiGas(k_fermi)

print("Energy per neutron of infinite free Fermi gas", E_gas/rho)

E_HF = compute_E_HF_simple(NN,sp_basis,dk)
E_HF = E_HF/NN
print("HF energy per neutron =", E_HF)

## now we start our business ...
## get all channels and two-body states within those channels; set T2 to zero
hh_channels, pp_channels, p_rel_hh, p_rel_pp, t2_pphh = setup_T2_amplitudes(sp_basis,NN,st_degen)

# get some insight in how big this all is
count=0
for i, channel in enumerate(p_rel_hh):
    dim=len(p_rel_hh[i])
    count=count+dim

print('hh number of total combinations', count)

count=0
for i, channel in enumerate(p_rel_pp):
    dim=len(p_rel_pp[i])
    count=count+dim

print('pp number of total combinations', count)


print("get v_hhhh, v_pphh, v_pppp")
start = timer()
v_hhhh, v_pphh, v_pppp = init_V(Lbox, dk, hh_channels, pp_channels, p_rel_hh, p_rel_pp)
end = timer()
print("what a hog!", end-start, 'seconds')


print("compute energy denominator")
start = timer()
fock_mtx4 = compute_Fock_4(sp_basis,NN,dk,Lbox,Nmax)
denom_pphh, f_abij = get_energy_denominator(pp_channels,p_rel_pp,p_rel_hh,sp_basis,NN,dk,Lbox)
end = timer()
print("that's faster", end-start, 'seconds')

print("Initialize T2 from MBPT2")
t2_pphh = get_t2_from_mbpt(v_pphh,denom_pphh)

erg = compute_E_CCD(v_pphh,t2_pphh)
print('MBPT2 correlation energy per neutron =', erg/NN)


print("start CCD iterations ...")

niter=200
mix=0.99
erg_old=0.0
eps=1.e-8
for iter in range(niter):

    start = timer()
    hbar_pphh = compute_hbar(v_pppp,v_pphh,v_hhhh,t2_pphh,f_abij)
```

```
        end = timer()
        print("time of making Hbar:", end-start, 'seconds')

        t2_new = get_t2_from_mbpt(hbar_pphh,denom_pphh)

        for i in range(len(t2_new)):
            t2_new[i] = t2_pphh[i] + t2_new[i]

        erg = compute_E_CCD(v_pphh,t2_new)

        myeps = abs(erg-erg_old)/abs(erg)
        if myeps < eps: break
        erg_old=erg

        print("iter=", iter, "Correlation energy per neutron=", erg/NN, ", epsilon=", myeps)

        for i in range(len(t2_pphh)):
            t2_pphh[i] = mix*t2_new[i] + (1.0-mix)*t2_pphh[i]

    print("Correlation energy per neutron= ", erg/NN)

    progend=timer()
    print('total time in seconds', progend-progstart)
```

===== =====

Benchmarks with the Minnesota potential

For the benchmarks, let us use a nucleon mass $m = 938.92$ MeV, $\hbar = 197.33$ MeV fm, and $c = 1$. For $N = 14$ neutrons at a density $\rho = 0.08\text{fm}^{-3}$ one finds $T_{\text{kin}}(|\phi_0\rangle)/N = 22.427553$ MeV, and $E_{HF}/N = 10.3498$ MeV. In model spaces with $N_{\text{max}} = 1$ and $N_{\text{max}} = 2$, and using only the first five diagrams of Figure 1.11 for the CCD calculation, yields the correlation energies per particle of $E_c/N = -0.2118$ MeV and $E_c/N = -0.6923$ MeV, respectively.

Chapter 3

From Structure to Reactions

Nuclear coupled-cluster theory has also been used to describe aspects of nuclear reactions, namely photo reactions and computations of optical potentials. In what follows, we want to discuss these approaches.

Electroweak reactions

Let us assume we probe a nucleus with an electroweak probe (e.g. via photon or Z-boson exchange). The corresponding operator $\hat{\Theta}$ introduces transitions in the nucleus. For photo reactions, the relevant operator is the dipole operator

$$\hat{\Theta} = \sum_{1=1}^{A} q_i(\vec{r}_i - \vec{R}_{CoM}). \tag{3.1}$$

Here q_i is the charge of nucleon i, and $\vec{R}_{CoM}$ is the position of the center of mass. The structure function or response function describing the reaction is

$$S(\omega) \equiv \sum_f \langle\psi_0|\hat{\Theta}^\dagger|\psi_f\rangle\langle\psi_f|\hat{\Theta}|\psi_0\rangle\delta(E_f - E_0 - \omega). \tag{3.2}$$

Here, the sum is over all final states. The structure function is difficult to compute because the sum is over (infinitely many) continuum states, and we seek a simpler formulation. The key idea is that the Lorentz integral transform (LIT) of the structure function

$$L(\omega_0,\Gamma) \equiv \frac{\Gamma}{\pi}\int d\omega \frac{S(\omega)}{(\omega-\omega_0)^2+\Gamma^2} \tag{3.3}$$
$$= \frac{\Gamma}{\pi}\langle\psi_0|\hat{\Theta}^\dagger \frac{1}{H-E_0-\omega_0+i\Gamma}\frac{1}{H-E_0-\omega_0-i\Gamma}\hat{\Theta}|\psi_0\rangle.$$

We note that the LIT $L(\omega_0,\Gamma)$ of the structure function is a ground-state expectation value and thus much easier to compute than the structure function itself. We also note that the LIT is not invertible (mathematically speaking), but making some assumptions about the structure function, and imposing a finite resolution Γ alleviates this problem in practical computation.

We next rewrite the LIT for coupled cluster using the shorthand $z \equiv E_0 + \omega_0 + i\Gamma$ as

$$L(\omega_0,\Gamma) \equiv \frac{\Gamma}{\pi}\langle\tilde{\psi}_L(z^*)|\tilde{\psi}_R(z)\rangle, \tag{3.4}$$

with $|\tilde{\psi}_R(z)\rangle$ and $\langle\tilde{\psi}_L(z^*)|$ fulfilling

$$\left(\overline{H}-z\right)|\tilde{\psi}_R(z)\rangle = \overline{\Theta}|\phi_0\rangle, \tag{3.5}$$
$$\langle\tilde{\psi}_L(z^*)|\left(\overline{H}-z^*\right) = \langle\phi_L|. \tag{3.6}$$

Here, $\langle\phi_L|$ is the left ground state, i.e. the left eigenstate of the similarity-transformed Hamiltonian. Note that in the coupled-cluster formulation we have distinguished between left and right states (as these are not adjoints of each other), and replaced all operators by their similarity transformations. Making the ansatz

$$|\tilde{\psi}_R(z)\rangle = \hat{R}|\phi_0\rangle \tag{3.7}$$
$$= \left(r_0(z) + \sum_{ia} r_i^a(z) a_a^\dagger a_i + \frac{1}{4}\sum_{ijab} r_{ij}^{ab}(z) a_a^\dagger a_b^\dagger a_j a_i + \cdots\right)|\phi_0\rangle$$

for the right state, and

$$\langle\tilde{\psi}_L(z^*)| = \langle\phi_L|\hat{L} \tag{3.8}$$
$$= \langle\phi_L|\left(l_0(z) + \sum_{ia} l_a^i(z) a_i^\dagger a_a + \frac{1}{4}\sum_{ijab} l_{ab}^{ij}(z) a_i^\dagger a_j^\dagger a_b a_a + \cdots\right)$$

for the left state, make the Eq. (3.5) linear systems of equations that can be solved for the parameters r_0, r_i^a, r_{ij}^{ab} and l_0, l_a^i, l_{ab}^{ij} for each value of z. The LIT then becomes

$$L(z) = l_0(z)r_0(z) + \sum_{ia} l_a^i(z)r_i^a(z) + \frac{1}{4}l_{ab}^{ij}(z)r_{ij}^{ab}(z) \tag{3.9}$$

in the CCSD approximation. For details, please see ["Giant and pigmy dipole resonances in 4He, 16,22O, and 40Ca from chiral nucleon-nucleon interactions," S. Bacca, N. Barnea, G. Hagen, M. Miorelli, G. Orlandini, and T. Papenbrock, Phys. Rev. C 90, 064619 (2014)].

Computing optical potentials from microscopic input

The single-particle Green's function

$$G(\alpha,\beta,E) \equiv \langle\psi_0|a_\alpha \frac{1}{E-(H-E_0)+i\eta}a_\beta^\dagger|\psi_0\rangle + \langle\psi_0|a_\beta^\dagger \frac{1}{E-(H-E_0)-i\eta}a_\alpha|\psi_0\rangle \tag{3.10}$$

describes the propagation of particles and holes in the nucleus with a ground state $|\psi_0\rangle$ and energy E_0. The Green's function fulfills the Dyson equation

$$G = G^{(0)} + G^{(0)}\Sigma^* G, \tag{3.11}$$

where Σ^* is the self energy and $G^{(0)}$ is the Hartree-Fock Green's function

$$\begin{aligned} G^{(0)}(\alpha,\beta,E) &\equiv \langle\phi_0|a_\alpha \frac{1}{E-(H_{HF}-E_{HF})+i\eta}a_\beta^\dagger|\phi_0\rangle + \langle\phi_0|a_\beta^\dagger \frac{1}{E-(H_{HF}-E_{HF})-i\eta}a_\alpha|\phi_0\rangle \\ &= \delta_\alpha^\beta\left(\frac{\Theta(\alpha-F)}{E-\varepsilon_\alpha+i\eta} + \frac{\Theta(F-\alpha)}{E-\varepsilon_\alpha-i\eta}\right). \end{aligned} \tag{3.12}$$

Here, $\Theta()$ denotes the unit step function and F labels the index of the Fermi surface. The key point is now that the optical potential Σ' (which describes the reaction of a single nucleon with the nucleus) is related to the self energy and the Hartree-Fock potential U_{HF} by [F. Capuzzi and C. Mahaux, "Projection operator approach to the self-energy," Ann. Phys. (NY) 245, 147 (1996)]

$$\Sigma' \equiv \Sigma^* + U_{HF}. \tag{3.13}$$

The idea is thus as follows. Starting from a Hartree-Fock calculations enables us to compute the Hartree-Fock potential U_{HF} and the Hartree-Fock Green's

function (3.12). Computing the Green's function (3.10) in coupled clusters, and inverting the Dyson equation (3.11) using

$$\Sigma^* = \left(G^{(0)}\right)^{-1} - G^{-1} \tag{3.14}$$

thus allows us to compute the optical potential. We note that the Green's function (3.10) resembles in its structure the LIT (3.3), and we will indeed use a similar approach to compute this object with the coupled-cluster method. Indeed, we compute

$$\left(E - (\overline{H} - E_0) + i\eta\right) |\tilde{\psi}_R\rangle = \overline{a_\beta^\dagger}|\phi_0\rangle, \tag{3.15}$$

$$\langle\tilde{\psi}_L| \left(E - (E_0 - \overline{H}) - i\eta\right) = \langle\phi_L|\overline{a_\beta^\dagger}, \tag{3.16}$$

by making the ansatz (3.7) and (3.8) for the right and left states, respectively, solve the resulting linear systems, and then compute $G(\alpha, \beta, E) = \langle\phi_L|\overline{a_\alpha}|\tilde{\psi}_R\rangle + \langle\tilde{\psi}_L|\overline{a_\alpha}|\phi_0\rangle$.

We note that a high quality optical potential can only be obtained if the structure of the nucleus is computed with a good accuracy, i.e. in good agreement to data on binding and separation energies, and charge and matter radii. For details, please see ["Optical potential from first principles," J. Rotureau, P. Danielewicz, G. Hagen, F. Nunes, and T. Papenbrock, Phys. Rev. C 95, 024315 (2017); arXiv:1611.04554]. We also note that this procedure must be repeated for any nucleus of interest, because the optical potential depends on the nucleus under consideration. Once the optical potential Σ' is computed, the single-particle Schroedinger equation

$$\left(-\frac{\hbar^2\Delta}{2m} + \Sigma'\right)\xi = E\xi \tag{3.17}$$

describes the interaction of a single nucleon (and wave function ξ) with the nucleus.

Finally, we note that this approach does not depend on solving the nucleus A with the coupled-cluster method. Alternatives, such as the IMSRG or Self-Consistent Green's Function methods could also be used.