Building a Shell Model Code Using the Pairing Model and NuShellX as Benchmark results

Gianluca Salvioni *[1], Ina K. B. Kullmann *[2], Matthew Shelley[3], and Gilho Ahn[4]

[1]Department of Physics, University of Jyväskylä, *gianlucasalvioni@gmail.com*
[2]Department of Physics, University of Oslo, *i.k.b.kullmann@fys.uio.no*
[3]Department of Physics, University of York, *mges501@york.ac.uk*
[4]Department of Physics, National University of Athens, *gilahn@phys.uoa.gr*

September 11, 2017

Abstract

In this report we outline the steps we made to create a simple shell-model code starting from scratch. At first we face the pairing problem, for which we calculate an analytical solution and build a numerical implementation coded in the programming language Python.
In parallel we practice with a state-of-art code NushellX, to understand the important features of the Configuration Interaction method and the information we can gain from this to study nuclear structure properties.
From the skeleton of the pairing code, we construct a simple shell model code that can perform calculations in 1s0d shell model space for 1 up to 12 valence neutrons (from ^{17}O to ^{28}O). The energy levels obtained with our shell model code for the Oxygen isotopes are exactly equivalent to the results calculated with NushellX.

Contents

I Introduction **2**

II The pairing problem **4**

III NuShellX **7**

IV Building our shell-model code **9**
i The structure of the code 9
ii The model space and the Slater determinants 10
iii The Hamiltonian matrix and the diagonalization 11
iv Unit tests and debugging 13
v Results and benchmarks with NuShellX 15

V Challenges and possible improvements **17**

VI Conclusion **19**

*Need credits for this report.

I. Introduction

The atomic nucleus is a complex system. It has the dimension of few fm (10^{-15}m) so the structure of the nucleus is regulated by quantum mechanics. We are interested in describing the low-energy scale that determine the nuclear configuration. Even though the fundamental particles that form a nucleus are quarks and gluons, the degrees of freedom we explore are protons and neutrons. The number of nucleons can be between one and a couple of hundreds, requiring a many-body treatment.
The sub-atomic system is one of the most attractive laboratories to study physics. Three different natural forces play an active role in the nucleus: the strong, electromagnetic and weak interaction. The Coulomb force acting among protons and the electromagnetic transitions are manifestations of the electromagnetic force. Beta decays are an example of weak interactions. The strong force keeps the nucleons together in a bound state, but this interaction is not yet fully understood. There exist no complete equation for the strong force. Comparing theoretical results of effective interactions with available experimental data help us investigate the importance of the different terms in the interaction models.

One of the aims of nuclear physics is to explain the properties of all the observed nuclei. The properties of the nuclei are also called observables and include the binding energy (mass), spin, radius, deformation, electromagnetic moments as well as life-time, excitation energies, electromagnetic transitions, decays, particle emissions and so on.

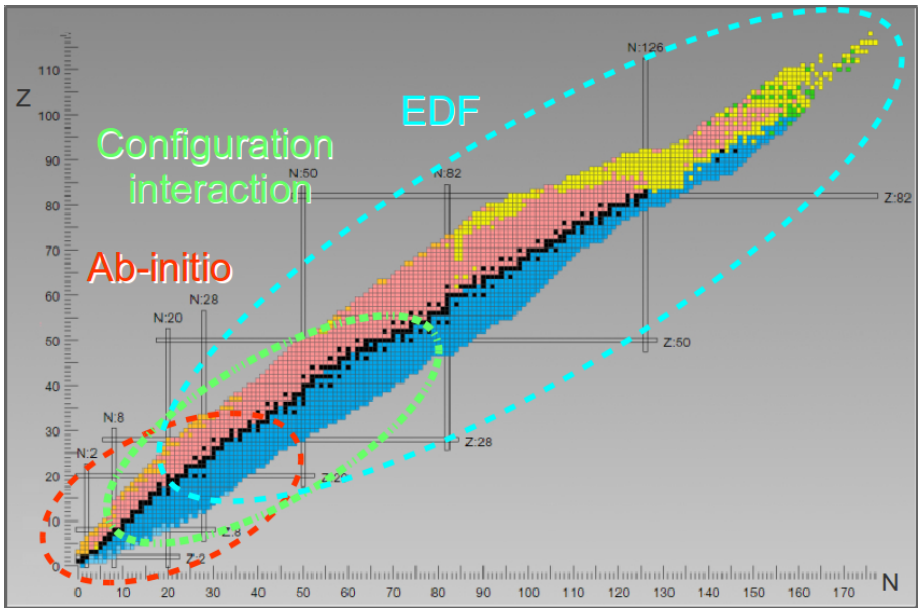

Figure 1: *In this nuclear chart the known nuclei are plot according to the neutron number N (x-axis) and proton number Z (y-axis). The domain of Ab-initio, Configuration Interaction and Energy Density Functional models are pictured. The nuclei represented in black are the stable ones.*

Another goal is to predict the limits of the possible bound systems (driplines). The nuclear chart can be described by three different theoretical models, as shown in Figure 1. Light nuclei, $A \lesssim 20$ is the playground of Ab-initio methods where an interaction derived by first principles (close to quantum chromodinamics) are applied and all the nucleons are taken into account. Medium

mass nuclei, $16 \lesssim A \lesssim 132$ can be studied by Configuration Interaction theory, also known as shell-models. Truncation of the model space is key to study large systems with shell-models due to limitations in computing resources. Heavy nuclei are usually described by Energy Density Functionals that reduce the volume of information needed. The the price for this information reduction is that mainly ground state properties can be calculated.

In this report we consider the Configuration Interaction method. In Sec.II we start to study a basic pairing model to practise with the second quantization formalism and the building block of the shell-model. We will obtain analytical results that we implement in a Python code to be used as benchmark for our own shell model code later.

In Sec.III we learn how to use state-of-the-art Configuration Interaction code NushellX and perform calculations of the Oxygen isotopic chain in the *sd*-model space.

We extend our Python code in Sec.IV to describe, through full the Configuration Interaction method, neutrons in the *sd*-model space with the goal to study the simple isotopic chain ^{17}O-^{28}O. We develop our shell model code beyond the pairing model and compare our energy levels to calculations done with NuShellX.

Further possible developments are discussed in Sec.V and finally our achievements are summarized in Sec.VI.

II. The pairing problem

The starting point for our shell-model code is a pairing model: the system consists of fermions combined together in pairs of two, one with spin up and one with spin down. In this section we will calculate the analytic results of the pairing model that was used to validate our shell-model code at an early stage in the development. The assumption in this model is that the 'breaking of pairs' is not allowed, i.e. the pairs of particles will always be coupled together, forming states with J=0. Of consequences the excited states are obtained from the excitation of two particles at the same time.

The theory of the pairing model. The physics of this system can be described by an Hamiltonian $\hat{H}$ consisting of an unperturbed one-body operator $\hat{H}_0$ and a two-body pertubation $\hat{V}$ defined as 'pairing potential': $\hat{H} = \hat{H}_0 + \hat{V}$. In second quantization we can write:

$$\hat{H}_0 = \sum_{p,\sigma} (\epsilon_p - 1)\hat{a}^\dagger_{p\sigma}\hat{a}_{p\sigma}, \tag{1}$$

$$\hat{V} = -\frac{1}{2} g \sum_{p,q} \hat{a}^\dagger_{p+}\hat{a}^\dagger_{p-}\hat{a}_{q-}\hat{a}_{q+}, \tag{2}$$

where the fermion creation and annihilation operators are given by $\hat{a}^\dagger_p$ and $\hat{a}_q$ respectively and $pqrs$ represent all possible single-particle quantum numbers.

The single-particle states $|p\rangle$ are chosen as eigenfunctions of the one-particle operator $\hat{h}_0$, then the Hamiltonian H acts in turn on various many-body Slater determinants constructed from the single-basis defined by the one-body operator $\hat{h}_0$.

The two-body pairing operator $\hat{V}$ is:

$$\langle q_+ q_- | \hat{V} | s_+ s_- \rangle = -g \tag{3}$$

where it is explicitly shown that for a given matrix element $\langle pq|\hat{V}|rs\rangle$ the states p and q (or r and s) must have opposite spin ($\sigma = \pm 1$). g is the (constant) strenght of the pairing interaction.

Using the formalism of second-quantization, the rules of anticommutation for fermions and the products of commutating and anticommutating operators it can be shown that $\hat{H}_0$ and $\hat{V}$ commute with the spin projection $\hat{J}_z$ and the total spin $\hat{J}^2$. This means that H can be diagonalized in separated blocks. And due to the 'no-broken pair' assumption, only $J = 0$ are allowed.

Constructing the Hamiltonian matrix. We are now interested to consider a system consisting of only four particles to calculate its exact analytic solution as benchmark for our shell-model code. The single-particle space is constructed by the four lowest levels with single-particle level energies $\epsilon_p = 1, 2, 3, 4$. Every level ϵ_p contains two particles, one with spin up $|p+\rangle$ and one with spin down $|p-\rangle$.

The possible Slater determinants that we can build are 6:

$$
\begin{aligned}
|\Phi_0\rangle &= \hat{a}^\dagger_{2+}\hat{a}^\dagger_{2-}\hat{a}^\dagger_{1+}\hat{a}^\dagger_{1-}|0\rangle \\
|\Phi_1\rangle &= \hat{a}^\dagger_{3+}\hat{a}^\dagger_{3-}\hat{a}^\dagger_{1+}\hat{a}^\dagger_{1-}|0\rangle \\
|\Phi_2\rangle &= \hat{a}^\dagger_{4+}\hat{a}^\dagger_{4-}\hat{a}^\dagger_{1+}\hat{a}^\dagger_{1-}|0\rangle \\
|\Phi_3\rangle &= \hat{a}^\dagger_{3+}\hat{a}^\dagger_{3-}\hat{a}^\dagger_{2+}\hat{a}^\dagger_{2-}|0\rangle \\
|\Phi_4\rangle &= \hat{a}^\dagger_{4+}\hat{a}^\dagger_{4-}\hat{a}^\dagger_{2+}\hat{a}^\dagger_{2-}|0\rangle \\
|\Phi_5\rangle &= \hat{a}^\dagger_{4+}\hat{a}^\dagger_{4-}\hat{a}^\dagger_{3+}\hat{a}^\dagger_{3-}|0\rangle,
\end{aligned}
$$

from which we can construct the basis for the Hamiltonian matrix:

$$
|\Phi_0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |\Phi_1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \cdots \quad |\Phi_5\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.
$$

The matrix elements $H_{ij} = \langle\Phi_i|\hat{H}|\Phi_j\rangle$ [1] are obtained using the Hamiltonian of Eq. (1), the Slater determinants and the Wick-theorem. $\hat{H}_0$ acts only on the diagonal and results in terms proportional with $(\epsilon_p - 1)$. The interaction will excite or deexcite a pair of particles from level q to level p.
Using the Wick-theorem, we notice that only the Slater determinants that differ for maximum a pair can interact among them through the pairing potential:

$$
\begin{aligned}
\langle\Phi_i|\hat{V}|\Phi_j\rangle &= \langle 0|\hat{a}_{i_1-}\hat{a}_{i_1+}\hat{a}_{i_2-}\hat{a}_{i_2+}\left[-\frac{1}{2}g\sum_{p,q}\hat{a}^\dagger_{p+}\hat{a}^\dagger_{p-}\hat{a}_{q-}\hat{a}_{q+}\right]\hat{a}^\dagger_{j_2+}\hat{a}^\dagger_{j_2-}\hat{a}^\dagger_{j_1+}\hat{a}^\dagger_{j_1-}|0\rangle \\
&= -\frac{1}{2}g\left(\delta_{p,i_2}\delta_{q,j_2}\delta_{i_1,j_1} + \delta_{p,i_2}\delta_{q,j_1}\delta_{i_1,j_2} + \delta_{p,i_1}\delta_{q,j_2}\delta_{i_2,j_1} + \delta_{p,i_1}\delta_{q,j_1}\delta_{i_2,j_2}\right) \qquad (4)
\end{aligned}
$$

and Eq.(4) is not null only if at least one of $\delta_{i_1,j_1}, \delta_{i_1,j_2}, \delta_{i_2,j_1}$ or δ_{i_2,j_2} among bra and ket states is not zero.
For example the first term in Eq.(4) is obtained from the contractions

$$
\langle 0|\hat{a}_{i_1-}\hat{a}_{i_1+}\hat{a}_{i_2-}\hat{a}_{i_2+}\hat{a}^\dagger_{p+}\hat{a}^\dagger_{p-}\hat{a}_{q-}\hat{a}_{q+}\hat{a}^\dagger_{j_2+}\hat{a}^\dagger_{j_2-}\hat{a}^\dagger_{j_1+}\hat{a}^\dagger_{j_1-}|0\rangle = \delta_{p,i_2}\delta_{q,j_2}\delta_{i_1,j_1}.
$$

Explicitly, the matrix element H_{00} is

$$
\begin{aligned}
\langle\Phi_0|\hat{H}|\Phi_0\rangle &= \langle 0|\hat{a}_{1-}\hat{a}_{1+}\hat{a}_{2-}\hat{a}_{2+}\left[\sum_{p,\sigma}(\epsilon_p - 1)\hat{a}^\dagger_{p\sigma}\hat{a}_{p\sigma}\right]\hat{a}^\dagger_{2+}\hat{a}^\dagger_{2-}\hat{a}^\dagger_{1+}\hat{a}^\dagger_{1-}|0\rangle \\
&+ \langle 0|\hat{a}_{1-}\hat{a}_{1+}\hat{a}_{2-}\hat{a}_{2+}\left[-\frac{1}{2}g\sum_{p,q}\hat{a}^\dagger_{p+}\hat{a}^\dagger_{p-}\hat{a}_{q-}\hat{a}_{q+}\right]\hat{a}^\dagger_{2+}\hat{a}^\dagger_{2-}\hat{a}^\dagger_{1+}\hat{a}^\dagger_{1-}|0\rangle \\
&= 2\cdot 0 + 2\cdot(2-1) + 2\cdot\left(-\frac{1}{2}g\right) = 2 - g. \qquad (5)
\end{aligned}
$$

[1] We use the label 0 to indicate the Slater determinants with the 4 particles in the lowest states, then the first row first column matrix element is H_{00}....

The Hamiltonian matrix becomes

$$\hat{H} = \begin{pmatrix} 2-g & -g/2 & -g/2 & -g/2 & -g/2 & 0 \\ -g/2 & 4-g & -g/2 & -g/2 & 0 & -g/2 \\ -g/2 & -g/2 & 6-g & 0 & -g/2 & -g/2 \\ -g/2 & -g/2 & 0 & 6-g & -g/2 & -g/2 \\ -g/2 & 0 & -g/2 & -g/2 & 8-g & -g/2 \\ 0 & -g/2 & -g/2 & -g/2 & -g/2 & 10-g \end{pmatrix}. \tag{6}$$

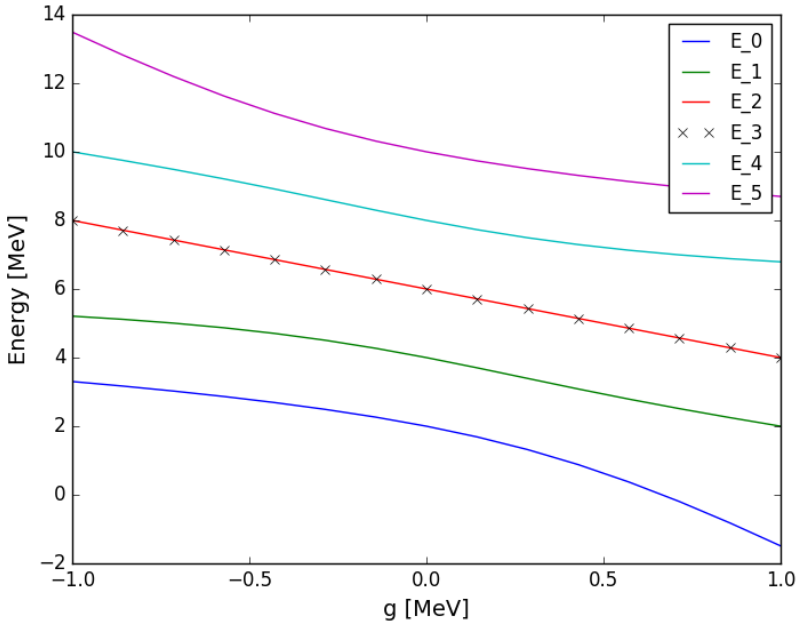

Figure 2: *The energy levels as a function of the strength g for the analytic case of the pairing model.*

In Figure 2 we see the eigenvalues of the Hamiltonian matrix (6) for the pairing model as a function of the strength $g \in [-1, 1]$. [2]. We can observe that the pairing potential increases the gap among the energy levels *E_i*, eigenvalues of Eq.(6). The case $g = 1$, corresponding to a strong attractive pairing, produces the more stable system (lowest energy levels).

[2]The diagonalization was done with `numpy`.

III. NuShellX

NushellX is a code developed by W. D. M. Rae [NushellX] that provides an powerful tool for nuclear structure calculations. This shell-model code is capable of Hamiltonian matrix calculations with very large basis dimensions. With NuShellX one can obtain energies, eigenvectors and spectrosopic overlaps for low-lying states. Here we are interested in extract energies from NuShellX that will be used to benchmark our shell-model code in the development beyond the pairing model.

NushellX is written in *pn*-formalism, i.e. the model space considers neutron and protons states separately, and couples them in *J*-scheme. The single-particle basis, the states for protons and neutrons are created separately. The code can consider *J*-scheme matrix dimensions of up to the order of a 100 million. To take advantage of many cores in modern computers, OpenMP for the Lanczos iterations is used. This way the code can handle quite large model spaces in a reasonable time on a laptop too.

Calculations in an infinite space are not possible and therefore some truncation is required. Ideally one would like to do calculations in the largest possible model spaces and one can argue that the best and most complete results are found with the largest model spaces. The major issue is that the computational time increases exponentially with model space size, so the wish for large model spaces are limited by the computational power and time available. When doing a calculation it may be useful to first truncate severely and quickly get a first crude result. Then one can later relax the truncations until the desired accuracy is reached.

Another challenge when running shell model calculations is that the chosen interaction must be appropriate for the model space. A calculation in the full model space may be too computationally intensive in itself, but it may also be inappropriate for the interaction. To run a shell model calculation one have to make two major decisions: which interaction will we use, and which model space truncation? In NuShellX only subshell truncations can be made. We can only limit the number of valence particles (protons and neutrons separately) in a given orbital.

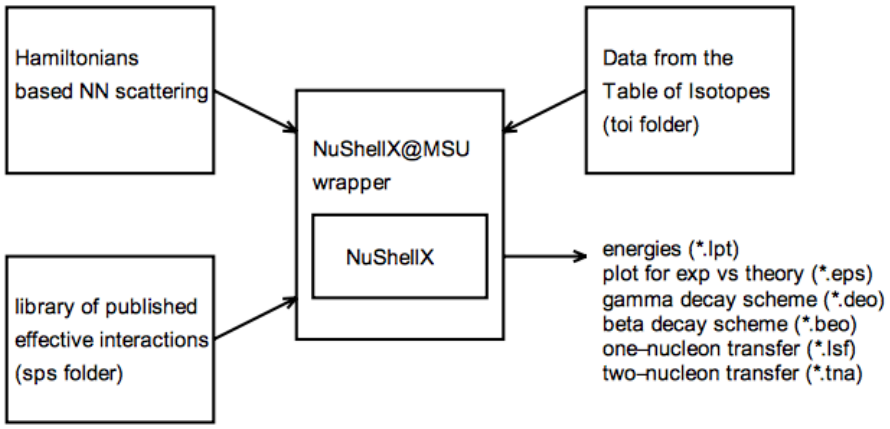

Figure 3: *Schematic layout of the NuShellX@MSU codes from Ref.[NushellX]*

Running a NuShellX calculation. A set of wrapper codes called NuShellX@MSU is written by Alex Brown. These wrapper codes generates input for the NuShellX code and converts the output

into tables and figures for energy levels, gamma decay and beta decay. In figure 3 we see the outline of the in- and output administered by the NuShellX@MSU wrapper code. When `shell` is run the user is asked to provide[3]:

- the batch file name
- the model space name
- restrictions / truncations
- interaction name
- number of protons and nucleons
- min and max J and parity
- extended functionalities

The NuShellX@MSU code provides the model spaces (`*.sp`) and hamiltonians (`*.int`) found in the `sps` folder. The file `label.dat` contains a list of available model space and hamiltonian combinations. For a more detailed description of how to run the NuShellX code see Ref.[NushellX help].

The extended functionality (called by the option `den`) make NushellX a powerful program to study atomic nuclei. This option makes it possible to calculate electromagnetic transitions, neutron and proton emission, beta decay as well as Hartree-Fock calculations in Energy Density Functional style. In this report we will only show the NuShellX calculations that will be used to benchmark our own shell model code.

NuShellX results. We calculated the excitation energies of the isotopes ^{17}O to ^{28}O using the *usdb* effective interaction optimized for the 1s0d shells. In Figure 4 the results for ^{18}O and ^{20}O is shown. The plot is generated by the NuShellX@MSU code, we see the theoretical results (left) and the the experimental available values (right) for each isotope. We observe that the agreement between theory and experiment is good, especially for the lowest excited states.

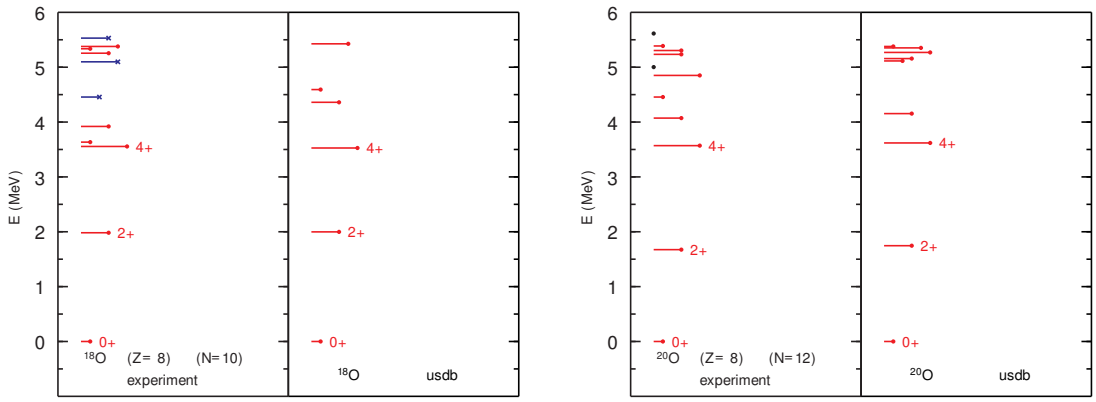

Figure 4: *Example of calculation of the energy levels done with NushellX for ^{18}O and ^{20}O and their relative experimental values. The agreement is good especially for the lowest excited states.*

[3]not all options are given in this example, the input depend on the type of calculation done with NuShellX

IV. Building our shell-model code

Our goal is to describe the specific case of the *O* isotopes within the *sd* model space. In this section we will not describe the code of the pairing model in detail, we simply state that we are building on a working code that reproduce the results given in Sec.II. From this we will extend our code to perform shell-model calculations. Our results will be directly compared and benchmarked to the same calculations performed with NushellX.

i. The structure of the code

We construct our shell-model code step by step, keeping in mind that we are interested in describing a system with N particles in the valence space. The code is organized into blocks of code, or modules, stored in separate files that performs different tasks. The files needed to run the shell-model code is:

- `main.py`
- `input_func.py`
- `create_table_files.py`
- `read_files.py`
- `unit_tests.py`
- `compare.py`
- `ham.py`

The `main.py` program runs the code and calls the functions in the other files. The structure of the `main.py` program is listed[4] below:

1. Import files (modules)
2. Ask user to provide input on the command line. The only inputs needed are
   - `N_particles` - number of valence particles (neutrons)
   - `case` - model space 'sd' (or 'pairing')
   - `g` - strenght of the pairing (needed only in 'pairing' case)
3. Read the single-particle basis
4. Create all the possible the Slater Determinants
5. Read and store the two-body matrix elements (tbme). In the case 'pairing', create all the non-zero pairing matrix elements according to the given strenght g.
6. Build the total Hamiltonian matrix from the one- and two-body terms
7. Check (unit test) if the Hamiltonian for the pairing problem is correct. If not exit the program with an error message
8. Solve the diagonalization problem and print the eigenvalues and eigenvectors of the problem

To run the code one simply type the following into the terminal

```
python main.py
```

and the program will interactively ask the user for input in the terminal (point 2). The user input function is implemented in such a way that the user is not allowed to provide unphysical input

[4] We refer to this 'structure list' and its numbering later in the text (point ...).

values. If invalid input is given the program provides an error message and asks the user to provide the input within the allowed type or interval. When the input is given the program runs and prints the results in the terminal.

ii. The model space and the Slater determinants

The single-particle states are the fundamental building blocks of the shell model, in fact they form the so called 'single-particle basis'. As indicated in the 'structure list' (point 3), reading in the basis is the first step after the input is given. In our case we read the information of the single-particle levels from file. For example `sd_shell.sp` contains the quantum number n,l,$2j$ and $2m_j$ together with their energy levels (eigenvalue of the single-particle Hamiltonian). This file was provided in the first part of the file `sdshellint.dat`[usdb interaction file]. Looking at the energies we can notice a degeneration respect to the m_j quantum number.

The next step in the 'structure list' is to create all the possible Slater Determinants (point 4). A Slater determinant $|\Phi_i\rangle$ is a product of N single-particle states that differ from each other (we are working with fermions and we need to satisfy the Pauli principle). The total number of Slater determinants which can be built with N particles distributed among i single-particle states is

$$d = \begin{pmatrix} i \\ N \end{pmatrix} = \frac{i!}{(i-N)!N!}. \tag{7}$$

The N-particle wave function, the solution of the Hamiltonian H is a linear combination of the Slater determinants constructed inside the model space. In `create_table_files.py` we calculate all possible Slater determinats (see Code 1) and store them in `sd_SlaterD.sd`. We work in m-scheme and we do not impose any condition on the total $M = \sum m_j$.

```
import numpy as np
import itertools

def create_SD_perm(N_particles, nr_sp_states, sp_matrix, SD_filename, restrictions=''):
    ...

    nr_sp_states = int(nr_sp_states)
    sp_list = []
    for i in range(1, nr_sp_states+1):
        sp_list.append(i)
    sp_tuple = tuple(sp_list)
    #print sp_tuple
    index = 0
    SD_list = []
    act_list = []
    for x in itertools.combinations(sp_tuple, N_particles):
        m_tot = 0
        for k in range (N_particles):
            m_tot = m_tot + sp_matrix[x[k]-1,4]
        #PAIRING CASE
        # only N_particles even is considered
        if restrictions == 'pair':
            if N_particles%2 == 0:
                pair_bool = 0
                # check that the single-particle states are in pairs
                for j in range (0,N_particles,2):
                    if np.array_equal(sp_matrix[x[j]-1,1:3], sp_matrix[x[j+1]-1,1:3]):
                        pair_bool = pair_bool
```

```
                    else:
                        pair_bool = pair_bool +1
                # check that M=0 and pairs are coupled
                if m_tot == 0 and pair_bool == 0:
                    index +=1
                    SD_list.append(index)
                    SD_list.extend(list(x))
        #GENERAL CASE
        else:
            index +=1
            SD_list.append(index)
            SD_list.extend(list(x))

    nr_SD = index
    SD_array = np.array(SD_list)
    SD_states = SD_array.reshape(nr_SD,N_particles+1)
    ...
```

Code 1: *The function* `create_SD_perm` *in the file* `create_table_files.py` *uses the Python function* `itertools.combinations(,)` *to create all the possible Slater determinants. In the case of the pairing model the Slater determinants are restricted to the ones with* $M = 0$ *and coupled pairs of single-particle states.*

iii. The Hamiltonian matrix and the diagonalization

Point 6 in the 'structure list' is to use the Slater determinants as basis to set up the Hamiltonian matrix $H_{ij} = \langle \Phi_i | \hat{H} | \Phi_j \rangle$. We construct the Hamiltonian in the module `ham.py` as a sum of the one-body Hamiltonian

$$\hat{H}_0 = \sum_{p,q} \langle p|\hat{h}_0|q\rangle \hat{a}_p^\dagger \hat{a}_q, \tag{8}$$

and the two-body interaction

$$\hat{V} = \frac{1}{4} \sum_{p,q,r,s} \langle pq|\hat{v}|rs\rangle_{AS}\, \hat{a}_p^\dagger \hat{a}_q^\dagger \hat{a}_s \hat{a}_r, \tag{9}$$

where $\langle pq|\hat{v}|rs\rangle_{AS}$ are the two-body antisymmetrized matrix elements for the $\hat{v}$ interaction. We read them from the file `sd_mscheme.int` (point 5)) adapted from Ref.[usdb interaction file] for the interaction *usdb* optimized in the *sd*-model space.

The symmetries of the two-body matrix elements for exchanging of two states are explicitly implemented. When the program read the matrix elements from the `.int` file, it stores it and all the matrix elements related to this for exchange of two states with the correct phase:

$$\langle ab|\hat{v}|cd\rangle = -\langle ab|\hat{v}|dc\rangle = -\langle ba|\hat{v}|cd\rangle = \langle ba|\hat{v}|dc\rangle \tag{10}$$

according to the antisymmetrization rules for fermionic states. Moreover in the case of usdb matrix elements a correction to the mass is automatically added to every matrix elements

$$mass_{corr} = \left(\frac{18}{16+n}\right)^{0.3} \tag{11}$$

where n is the number of valence particles.

It is interesting to point out the physics behind the set-up of the Hamiltonian matrix $H_{ij} = \langle\Phi_i|\hat{H}|\Phi_j\rangle$. We assume $|\Phi_0\rangle$ as an ansatz for the ground state and we consider the other Slater determinants $|\Phi_i\rangle$ as particle-hole (p-h), two particles- two holes (2p-2h), ... excitations of the ground state. In this formalism

$$|\Phi_0\rangle = \hat{a}_i^\dagger \hat{a}_j^\dagger ...|0\rangle \tag{12}$$

has all the single-particle levels $i, j, ...$ filled and we indicate that with $i, j, ... \leq F$ Fermi energy. Of consequence, we define a p-h Slater determinant with

$$|\Phi_i^a\rangle = \hat{a}_a^\dagger \hat{a}_i|\Phi_0\rangle \tag{13}$$

where the particle index is $a > F$ and the hole index is $i \leq F$, and a 2p-2h one with

$$|\Phi_{ij}^{ab}\rangle = \hat{a}_a^\dagger \hat{a}_b^\dagger \hat{a}_j \hat{a}_i|\Phi_0\rangle \tag{14}$$

where $a, b > F$ while $i, j \leq F$.

Using Wick's contractions we can calculate the expectation values of the Hamiltonian operators among the states just built:

$$\langle\Phi_0|\hat{H}|\Phi_0\rangle = \sum_{i\leq F}\langle i|\hat{h}_0|i\rangle + \frac{1}{2}\sum_{i,j\leq F}\langle ij|\hat{v}|ij\rangle_{AS} \tag{15}$$

$$\langle\Phi_0|\hat{H}|\Phi_i^a\rangle = \cancel{\langle i|\hat{h}_0|a\rangle} + \sum_{j\leq F}\langle ij|\hat{v}|aj\rangle_{AS} \tag{16}$$

where $\langle i|\hat{h}_0|a\rangle = 0$ because the single-particles states are orthogonal,

$$\langle\Phi_0|\hat{H}|\Phi_{ij}^{ab}\rangle = \langle ij|\hat{v}|ab\rangle_{AS} \tag{17}$$

$$\langle\Phi_0|\hat{H}|\Phi_{ijk}^{abc}\rangle = 0 \tag{18}$$

where $|\Phi_0\rangle$ and $|\Phi_{ijk}^{abc}\rangle$ (3p-3h state) differ for more than 2-particles, the maximum number that can be contracted with a two-body Hamiltonian.

We can repeat an analogue procedure starting from a generic $|\Phi_j\rangle$ where j can be different from 0, and we will derive similar expressions for the Hamiltonian matrix elements, meaning that for each $|\Phi_j\rangle$ of the basis of Slater determinants there are only 'limited' $\langle\Phi_i|$ that can interact with it through $\hat{H}$. We use this information to construct H_{ij} in a more efficient way, calculating only the non-zero matrix elements.

In our Code 2, we compare the content of the single-particle states in $\langle\Phi_i|$ and $|\Phi_j\rangle$. We store a list ($diff$) of the levels that are different and their position in the Slater determinant. The list is read by the routine that calculate the Hamiltonian matrix. The lenght of the list select the non-zero contributions due to Eq.s(15),(16) and (17). The labels of the levels determine the one-body and two-body matrix elements to be added.

The indices of the position of each levels in the respective Slater determinant, $position(\hat{a}_k^\dagger)$, control the phase of the Hamiltonian matrix element:

$$phase(H_{ij}) = (-1)^{phase(diff)} \tag{19}$$

where

$$phase(diff) = \sum_{k \in diff} position(\hat{a}_{i_k}) + \sum_{m \in diff} position(\hat{a}^{\dagger}_{j_m}). \tag{20}$$

The phase is obtained considering the anticommutation rules of the fermionic annihilation and creation operators that are involved in the Wick's contractions.

```
def beta_alpha_compare(beta_list, alpha_list):

    phase = 0
    diff_list = []
    beta_list_red = list(beta_list)
    alpha_list_red = list(alpha_list)
    #if len(beta_list) == len(alpha_list):
    for i in range(0, len(beta_list)):
        j=0
        j_found = False
        while j < len(alpha_list) and j_found == False:
            #print beta_list[i],i, alpha_list[j],j, diff_list
            if beta_list[i] == alpha_list[j]:
                j_found = True
                alpha_list_red.remove(alpha_list[j])
                beta_list_red.remove(beta_list[i])
                #phase = phase+j+len(beta_list)-i-1
            else:
                j = j+1
    diff_list.extend(beta_list_red)
    diff_list.extend(alpha_list_red)


    for i in range(0,len(diff_list)/2):
      phase = phase + (list(beta_list).index(diff_list[i]))

    for i in range(len(diff_list)/2,len(diff_list)):
        phase = phase + (list(alpha_list).index(diff_list[i]))

    phase = (-1)**phase

    #print diff_list, phase

    return diff_list, phase
```

Code 2: *`beta_alpha_compare` is a function in the module `compare.py` that makes a comparison between $\langle\Phi_\beta|$ and $|\Phi_\alpha\rangle$ to determine their difference in terms of single-particle states.*

The Hamiltonian matrix is finally diagonalized using `numpy` (point 8 in the 'structure list'). The eigenvalues obtained represent the energy of the N-particles state and the eigenvectors are the coefficient of the Slater determinants that form the many-body wave function.

iv. Unit tests and debugging

We have implemented an unit test for the Hamiltonian of the pairing model (point 7). The code built a test Hamiltonian using our routine and if this is the same as the analytic one derived in Sec.II, matrix element per matrix element, the code moves to built the actual Hamiltonian asked from the inputs and to diagonalize it. If the numerical and analytic Hamiltonian do not match, the code will be exited with an error message. See Code 3.

```
def unit_test_hamiltonian_pairing():
    ...
    # numerical hamiltonian
    ...
    numerical_hamiltonian = np.zeros((nr_SD_Utest, nr_SD_Utest))
    numerical_hamiltonian_1body = np.zeros((nr_SD_Utest, nr_SD_Utest))
    numerical_hamiltonian_2body = np.zeros((nr_SD_Utest, nr_SD_Utest))
    numerical_hamiltonian_1body = Hamiltonian_one_body(N_Utest, nr_sp_states_Utest,
    sp_matrix_Utest, SD_filename_Utest)
    numerical_hamiltonian_2body = Hamiltonian_two_body(N_Utest, nr_sp_states_Utest,
    SD_filename_Utest, tbme_filename_Utest, 'pairing')

    numerical_hamiltonian = numerical_hamiltonian_1body+numerical_hamiltonian_2body

    #analytical hamiltonian
    dim = 6

    analytical_hamiltonian = -g_Utest/2.*np.ones([dim,dim])
    np.fill_diagonal(analytical_hamiltonian, 2.-2.*g_Utest/2.)
    np.fill_diagonal(np.rot90(analytical_hamiltonian), 0)

    for n in range(dim):
        analytical_hamiltonian[n,n] += 2*n

    for m in range(5,2,-1):
        analytical_hamiltonian[m,m] = analytical_hamiltonian[m-1,m-1]

    unit_test = np.array_equal(numerical_hamiltonian, analytical_hamiltonian)

    if not unit_test:
        print "\nERROR IN UNIT TEST: The numerical and analytical hamiltonian matrix are
    not equal!\n"
        print 'The numerical hamiltonian matrix (shape):'
        print np.shape(numerical_hamiltonian)
        print numerical_hamiltonian
        print '\n'
        print 'The analytical hamiltonian matrix (shape):'
        print np.shape(numerical_hamiltonian)
        print analytical_hamiltonian
        print '\n'
        sys.exit()

    if unit_test:
        print "UNIT TEST COMPLETED: Hamiltonian matrix set up correctly.\n"
```

Code 3: *`unit_test_hamiltonian_pairing()` is a unit test run by `main.py` before it starts to set up the actual Hamiltonian matrix. The unit test is only for the pairing model and compares the analytic and the numerical Hamiltonian element-wise and exits the code if they do not match.*

To quickly find the source of a bug during the development and debugging of the sd-shell code it is useful to:

- Check that the dimensions of the Hamiltonian matrix are equal to the number of Slater determinants: $dim(H_{ij}) = d \times d$ where d is # of Slater determinants, as in Eq.(7). This controls errors when reading in the model space and creating the Slated determinants.
- Check that the single-particle energies are read correctly. If we select `N_particles` $= 1$, the eigenvalues of the Hamiltonian matrix should be the single-particle energies. The degeneracy should be equal to $2j + 1$ because in this case the two-body interaction does not contribute.

- Check that H_{ij} is a symmetric matrix. If not the bug is most likely in the reading routine of the two-body matrix elements or in the calculation of the phases.
- Perform an analysis of the Hamiltonian matrix on a restricted space. For example the matrix elements of the case `N_particles` $= 2$ and $M = M_{max} = 4$ can be calculated by hand to compare with the code results.

v. Results and benchmarks with NuShellX

Now that our basic Configuration Interaction code for the *sd* model space is completed we can benchmark it with the results obtained from NushellX (see Sec.III). The proper comparison is within *usdb* interaction for the Oxygen isotopes, where varying the number of valence neutrons in our code (`N_particles`) corresponds to varying the number of neutrons on top of the ^{16}O core.

We have built our basis of Slater determinats without restrictions on M, the z-component of the total angular momentum J. This choice is reflected in the eigenvalues of the Hamiltonian matrix. There are degenerate eigenvalues that correspond to a different value of M but same J. When we compare our results with the NushellX code, that works in J-scheme and gives also the expectation value of J for each eigenstates, we can observe that the number of our degenerate levels corresponds to $2J + 1$. This is due to the fact that the interaction *usdb*[usdb 2006] depends only on J and not on its z-projection.
As an example, we consider respectively our code (Code 4) and NushellX (Code 5) results for the energy levels of ^{20}O, 4 valence neutrons. In Code 4, we can see that the eigenvalue -23.632 is repeated only one time it means that $J = 0$ as we can crosscheck in Code 5. For the 5 times degenerate level -21.866 we have in fact $J = 2$ ...

```
Default inputs:
N_particles = 4
case = sd

...calculating...

UNIT TEST COMPLETED: Hamiltonian matrix set up correctly.

model space sd
Number of particles (neutrons): 4


Eigenvalues:
[-23.632 -21.886 -21.886 -21.886 -21.886 -21.886 -20.013 -20.013 -20.013
 -20.013 -20.013 -20.013 -20.013 -20.013 -20.013 -19.478 -19.478 -19.478
 -19.478 -19.478 -18.518 -18.518 -18.518 -18.475 -18.475 -18.475 -18.475
 -18.475 -18.363 -18.363 -18.363 -18.363 -18.363 -18.363 -18.363 -18.363
 -18.363 -18.280 -18.280 -18.280 -18.280 -18.280 -18.280 -18.280 -18.254
 -16.248 -16.248 -16.248 -16.248 -16.248 -16.248 -16.248 -16.248 -16.248
 -16.163 -16.163 -16.163 -16.163 -16.163 -16.163 -16.163 -16.163 -16.163
 -16.163 -16.163 ...]


Eigenvectors:
[[-0.00  0.00 -0.00 ..., -0.00 -0.00 -0.00]
 [-0.00 -0.00  0.00 ..., -0.00  0.00 -0.00]
 [ 0.07  0.01 -0.00 ..., -0.00  0.00 -0.04]
 ...,
 [ 0.00  0.00  0.48 ..., -0.00 -0.00  0.00]
 [ 0.00  0.00 -0.00 ..., -0.00  0.00  0.00]
```

```
[-0.00   0.00  -0.00 ...,  -0.00   0.00  -0.00]]
```

Code 4: *example of the output from* `main.py` *when it is run with default inputs. To be noticed the degenerate eigenvalues.*

```

a =  20 z =  8
usdb            1.00000       0.96889   2.1117  -3.9257  -3.2079  16.0000  18.0000   0.3000

 N     NJ      E(MeV)   Ex(MeV)   J     T_z   p   lowest Ex       name
   1     1    -23.632     0.000   0     2     1     0.000       bb0400.lpe
   2     1    -21.886     1.746   2     2     1     1.746       bb0404.lpe
   3     1    -20.013     3.619   4     2     1     3.619       bb0408.lpe
   4     2    -19.478     4.154   2     2     1                 bb0404.lpe
   5     1    -18.518     5.114   1     2     1     5.114       bb0402.lpe
   6     3    -18.475     5.157   2     2     1                 bb0404.lpe
   7     2    -18.363     5.269   4     2     1                 bb0408.lpe
   8     1    -18.280     5.352   3     2     1     5.352       bb0406.lpe
   9     2    -18.254     5.378   0     2     1                 bb0400.lpe
  10     3    -16.248     7.384   4     2     1                 bb0408.lpe
  11     1    -16.163     7.469   5     2     1     7.469       bb040a.lpe
  12     4    -15.455     8.177   2     2     1                 bb0404.lpe
  ...
```

Code 5: `o_20b.lpt` *output from NushellX.*

All the Oxygen isotopic chain calculations (from ^{17}O to ^{28}O) are summarized in Figure 5, where we can see that the lowest states obtained with our code and with NushellX are identical.
This plot presents interesting features because it show that when increasing the number of neutrons (moving to the right along the x-axis) the system become more bound, i.e. the ground state energy decrease. This trend happens until valence neutron = 8 (^{24}O), then the binding energy starts to increases slightly. The physical phenomena that we can understand here is that around ^{24}O the neutron separation energy S_n, defined as $S_n(N,Z) = BE(N+1,Z) - BE(N,Z)$ becomes negative and the system is not anymore bound. It means that the Oxygen isotopes are approaching the neutron dripline.

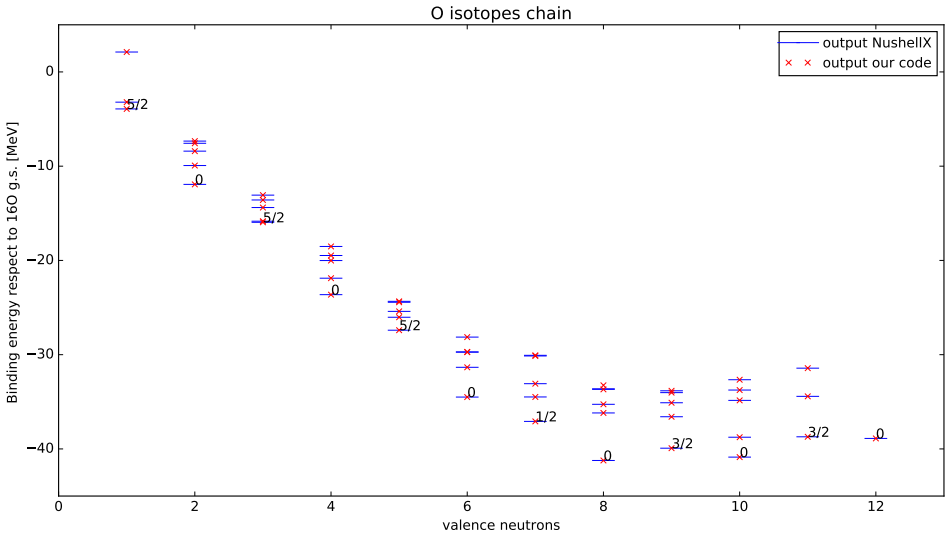

Figure 5: *Comparison between NushellX and our code energy levels for the Oxygen isotopes. The J value of the ground state is shown for each isotope*

V. Challenges and possible improvements

The natural next step in the development of our code would be the introduction of valence protons in our model space. It would require some corrections to the body of the code, but we could increase our calculations until ^{40}Ca (within the *sd* model space). This would make the code more flexible and also a more powerful tool for theoretical calculations.

An useful extension of our code would also be to calculate the expectation value $\langle\Psi_i|\hat{J}^2|\Psi_i\rangle$ for each eigenstate $|\Psi_i\rangle$ of the Hamiltonian matrix, expressing the operator $\hat{J}^2$ in second quantization and applying Wick's theorem. In this way we could label each of the calculated energy levels with the relative total angular momentum.

In NushellX there is a routine `runtrans` that calculate one-body transition densities, i.e. expectation values of the one-body density operator between initial and final states. These densities, multiplied by single-particle matrix elements, in the routine `dens`, give the matrix element for the transition from initial to final state. In our code, with some effort, we also could implement the one-body density operator in second quantization in order to study transitions between different eigenstates of the system (for example between an excited state and the ground state).

Compared to NuShellX our code has a numerical precision of KeV energies. The main sources of theoretical 'errors' are the truncation of the model space and the values of the matrix elements fitted to reproduce the experimental states. We are interested in energy levels and therefore consider the precision of our code at 1 KeV to be good.

A clear issue of our basic code is the execution time. With our code it takes about 10s on a laptop to calculate the eigenvalues for 6 valence neutrons (the slowest possible calculation due to the largest possible number of Slater determinants). Among the total time 0.5s are required for the unit test. Most of the time is spent in building the Hamiltonian matrix and diagonalizing it. The

most efficient way to improve the execution time of our code is therefore to adopt an optimized diagonalization routine (such as Lanczos algorithm) to replace the standard diagonalization done with numpy. The code would also speed up significantly if the technique of bit representation where implemented.

VI. Conclusion

In this report we have seen how useful it is to have good routines for bench marking results during the development of a shell model code. We have extensively compared analytic results to our code before implementing new features. In Subsec. IV.iv we have seen that a lot of time can be saved if unit tests and good debugging routines are implemented early.

The pairing model (Sec. II) proved itself as a very useful stepping stone to a more advanced code. NuShellX (Sec. III) provided us with more advanced results to benchmark our code at a later stage. In Subsec. IV.v we see that the results of our code reproduce calculations done with NuShellX well and with good precision.

There are some major issues with our code, mainly the time spent running the code. NuShellX is a very flexible tool for theoretical calculations that can be done surprisingly fast on a regular laptop. In Sec. V a few suggestions to make our code more flexible are provided, but we can conclude that its major issue is the execution time. This problem may be solved by implementing a bit representation and a better diagonalization routine for finding the eigenvalues of the Hamiltonian matrix.

We consider the development of this simple shell model code to be an absolutely useful tool to practice with the Configuration Interaction method. Coding such routines from scratch has been an highly formative experience for multiple reasons:

- translating the theoretical notions to a code language was a good way to better understand the formalism
- working on our own code provided us with the real picture of the technical difficulties one face when producing physically interesting results
- this summer school [TALENT School 2017] gave a lot of inputs to improve our knowledge about the structure of atomic nuclei

References

[NushellX] *https://people.nscl.msu.edu/ brown/brown-all-papers/525-2014-nds120.115-nushellx.pdf* (and references within)

[NushellX help] from NushellX folder *./help/help.pdf*

[usdb interaction file] *https://github.com/NuclearTalent/NuclearStructure/doc/pub/projects/datafiles/sdshellint.dat*

[usdb 2006] B. Alex Brown and W. A. Richter,Phys. Rev. C **74** (2006) 034315 *https://journals.aps.org/prc/abstract/10.1103/PhysRevC.74.034315*

[TALENT School 2017] A. Brown, A. Gade, R. Grzywacz, M. Hjorth-Jensen, G. Jansen, *https://github.com/NuclearTalent/NuclearStructure*

[Group_10 Github] *https://github.com/mgeshelley/group_10*