Building a Shell Model Code: The Pairing Model and the sd-Shell (and comparing results with NuShellX)

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*

August 29, 2017

Abstract

We have first implemented the pairing model which have a analytical solution (to benchmark the code). Then implemented the sd shell —> more general shell-model program that allows you to study general nuclear structure problems.

developing our own shell-model code that can perform shell-model studies of the oxygen isotopes using standard effective interactions (provided by us) using as example the 1s0d shell as model space.

We have also used the NushellX code in order to perform more advanced shell-model studies and compare the results obtained with your own shell-model code to those of NushellX

and found that.....results...

I. Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

II. The pairing problem

The starting point is a pairing model: our system consists of fermions combined together in pairs of two, one with spin up and one with spin down. The assumption 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 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 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

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

The Hamiltonian matrix becomes

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

[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}....

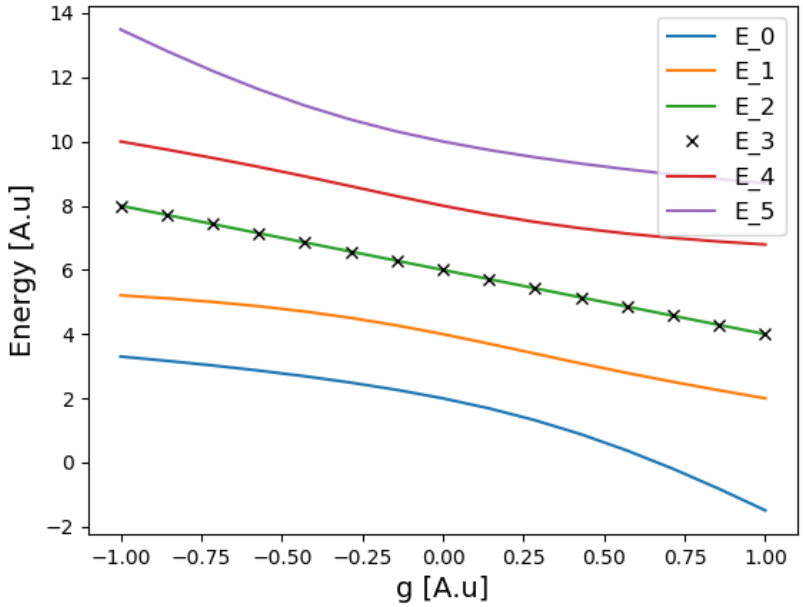

Figure 1: *The energy levels as a function of the strength g for the analytic case of the pairing model.*

In Figure 1 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).

III. Building the shell model code

Applying the technique we learned from the pairig model in section II, we want to extend our code to perform shell model calculation. In the specific case we build a working shell model code to describe the *O* isotopes within the *sd* model space. Our results will be directly compared and benchmarked to the same calculations performed with NushellX.

i. The structure of the code

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

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

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

- `input_func.py`

The `main.py` program runs the code and calls the functions in the other files. The structure of the `main.py` program is as follows:

1. Import files (modules)
2. Ask user to provide inputs 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. Print the eigenvalues and eigenvectors of the problem

To run the code one simply runs the program typing from terminal

```
python main.py
```

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

Need some sort of header or sentence here before the four bullet points below ('details' or something, or a subsection with a name?):

- **Model space:** The single-particle states are the foundamental building block of the shell model, in fact they form the so called "basis". 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). Looking at the energies we can notice a degeneration respect to the m_j quantum number.
- **Slater determinats:** The N-particle wave function, solution of the Hamiltonian H is a linear combination of the Slater determinants constructed inside the model space. A Slater determinant $|\Phi_i\rangle$ is a product of N single-particle states different one each other (we are working with fermions and we need to satisfy Pauli principle). In `create_table_files.py` we calculate all possible Slater determinats and store them in `sd_SlaterD.sd`. We work in m-scheme and we do not impose any condition on the total $M = \sum m_j$. The total number of Slater determinants which can be built with N particles distributed among i single-particle states is

$$d = \left(\begin{array}{c} i \\ N \end{array} \right) = \frac{i!}{(i-N)!N!}. \tag{7}$$

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

- **Hamiltonian matrix:** We use the Slater determinants as basis to set up the Hamiltonian matrix $H_{ij} = \langle \Phi_i | \hat{H} | \Phi_j \rangle$. It is constructed in `ham.py` as sum of 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 two-body interaction

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

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

- **Diagonalization:** The Hamiltonian matrix is finally diagonalized using `numpy`. 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. In our case we obtain degenerate energy levels because our Hamiltonian is not dependent from the total azimutal angular momentum M then all the M-multiplets with same J have the same energy [write it better with the comparison with NushellX that is in J-scheme.]

ii. Non-zero Hamiltonian matrix elements and their phase

It is interesting to point out the physics behind the set-up 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

$$|\Phi_0\rangle = \hat{a}_i^\dagger \hat{a}_j^\dagger ...|0\rangle \tag{10}$$

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

$$|\Phi_i^a\rangle = \hat{a}_a^\dagger \hat{a}_i |\Phi_0\rangle \tag{11}$$

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

$$|\Phi_{ij}^{ab}\rangle = \hat{a}_a^\dagger \hat{a}_b^\dagger \hat{a}_j \hat{a}_i |\Phi_0\rangle \tag{12}$$

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:

$$\langle\Phi_0|\hat{H}|\Phi_0\rangle = \sum_{i\leq F} \langle i|\hat{h}_0|i\rangle + \frac{1}{2} \sum_{i,j\leq F} \langle ij|\hat{v}|ij\rangle_{AS} \tag{13}$$

$$\langle\Phi_0|\hat{H}|\Phi_i^a\rangle = \cancel{\langle i|\hat{h}_0|a\rangle} + \sum_{j\leq F} \langle ij|\hat{v}|aj\rangle_{AS} \tag{14}$$

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

$$\langle\Phi_0|\hat{H}|\Phi_{ij}^{ab}\rangle = \langle ij|\hat{v}|ab\rangle_{AS} \tag{15}$$

$$\langle\Phi_0|\hat{H}|\Phi_{ijk}^{abc}\rangle = 0 \tag{16}$$

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.

Similar expressions for the Hamiltonian matrix elements can be obtained starting from $|\Phi_k\rangle$ with k generic and considering the other Slater determinants 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 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(13),(14) and (15). 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:

$$phase(H_{ij}) = (-1)^{phase(diff)} \tag{17}$$

where

$$phase(diff) = \sum_{k \in diff} position(\hat{a}_{i_k}) + \sum_{m \in diff} position(\hat{a}_{j_m}^\dagger). \tag{18}$$

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 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.*

iii. Unit tests and debugging

We have implemented an unit test that run the pairing model for the case we have derived analytically in Sec.II. If the hamiltonian built by our routine is the same as the analytic one, matrix element per matrix element, the code moves to perform the asked calculation. If the numerical and analytic hamiltonian do not match the code will be exited with an error message.

```
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*

We spent a lot of time in debugging our sd-shell code. In the following some hints to remember.

- 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 allow to control errors in reading the model space and creating the Slated determinants.

- Check that the single-particle energies are read correctly. If we select `N_particles` $= 1$, we should obtain that the eigenvalues of the Hamiltonian matrix are the single-particle energies with a degeneracy 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 first sections of the code to test are the reading routine of the two-body matrix elements and the calculation of the phases.
- Perform analysis of the Hamiltonian matrix on a restricted space, for example with `N_particles` $= 2$ and $M = M_{max} = 4$, for which the matrix elements can be calculated by hand and compared with the code results.

iv. Further considerations

In our code we have explicitly implemented the symmetries of the two-body matrix elements for exchanging of two states. 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:

$$\langle ab|\hat{v}|cd\rangle = -\langle ab|\hat{v}|dc\rangle = -\langle ba|\hat{v}|cd\rangle = \langle ba|\hat{v}|dc\rangle \tag{19}$$

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

$$mass_{corr} = \left(\frac{18}{16+n}\right)^{0.3} \tag{20}$$

where n is the number of valence particles.

In the case of sd-model space, 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 different value of M but same J. When we compare our results with 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* depends only on J and not on its z-projection.

For example, from the output of default inputs, in Code 4, we can see that -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 $J = 2$ ...

A possible improvement of out code will 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.

```
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.*

v. Challenges and proposed solutions / Future work(?)

something about the accuracy of our code somewhere here?

IV. NuShellX

NushellX is a code developed by W. D. M. Rae [ref] that provides an powerful tool for nuclear structure calculations. The shell-model code does Hamiltonian matrix calculations with very large basis dimensions. With NuShellX one can obtain exact energies, eigenvectors and spectrosopic overlaps for low-lying states. 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.

NuShellX uses the m-scheme and jj-coupling. It is written in pn-formalism; it uses a J-coupled proton-neutron the 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 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. [some of these sentences needs rewriting, Gianluca, maybe you know how to write this with same wording as above sections?]

The major disadvantage of NuShellX is that it is very easy to produce unrealistic or inappropriate results. 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 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. Even though the calculation in the full model space may be too computationally intensive, 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 protons (neutrons) in a given orbital. We cannot restrict the total number of nucleons in a given orbital since protons and neutrons are treated separately.

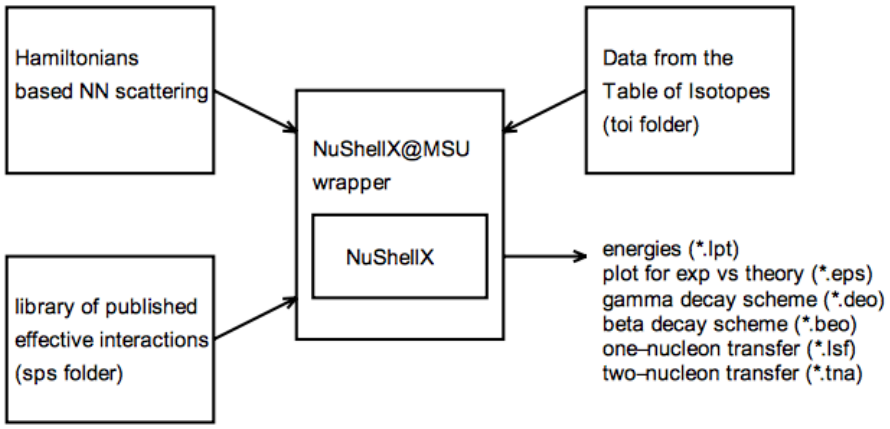

Figure 2: *Schematic layout of the NuShellX@MSU codes from [ref]*

In figure 2 from [ref] 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
- (anything else essential??)

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

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 help file]

i. Benchmark results from NuShellX

The goal of our project was to create a mini-version of the NuxhellX code and use the NushellX code to test our own code when going beyond the pairing model. We have only described the NuShellX calculations we used to benchmark our code in this report. [mention all the other types of calculations possible w NuShellX –> or is the calculations mentioned in the intro enough?]

We have calculated the excitation energies of 18O (and 18F ?). We have used an effective interaction designed for the 1s0d shell (USDA and USDB in the NushellX directory) [did we use both?]

In fig [] we see the sepctra of 18O....

[Insert the results of the NuShellX used to compare with our code. Do we have the plots or do we just 'borrow' from group 6?]

[Did we compare the spectra with available data? Or does NuShellX do that already?]

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

ref: help/help.pdf

V. Results

VI. Discussion

VII. Conclusions

Structure of report (crude first idea):

1. Title
2. Abstract
3. Introduction
    - Why useful
    - Where useful (nuc. chart)
    - (Look at the TALENT proposal)
    - comparing different tools
4. The Pairing problem
    - theory
    - analytic / exact results
5. Building a shell-model code
    - Describe the different parts/blocks of the code (basis, SD, states, int, hamiltonian...)
    - Unit tests / benchmarks (analytic/exact results)

- Psedo code?
- Accuracy of our code
- Where our code fails, why and proposing solutions

6. NuShellX

- describe why useful / 'powerful tool' and what may be calculated
- compare with our own code
- Possible 'future calculations'

7. Conclusions / Summary

References

[Github of the TALENT School] MHJ and Alex Brown *https://github.com/NuclearTalent/NuclearStructure*

[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*