

# NuShellX for Windows and Linux

William D. M. Rae\*

*Knollhouse, Garsington, Oxford.*

NuShellX is a new nuclear shell model code that is based on the code MultiShell by Godwin and Rae [1] with ideas borrowed from the Nathan shell model code of the Strasbourg group, E. Caurier et al. [2] It has been enhanced using ideas suggested by J. Toivanen [3] and used in his shell model code EICODE [4]. It is written in Fortran 95 with some Fortran 2003 additions and uses OpenMP multiprocessing to take advantage of modern multicore processors. It compiles with the Intel Fortran Compiler [5] which is available for Windows and Linux platforms. It is capable of doing calculations with dimensions of over 100,000,000 basis states in the angular momentum coupled scheme on a typical dual core PC with 4-6 GB of RAM. It makes no real demands on IO systems. Higher clock speeds and more cores significantly decrease the execution times particularly for non zero total angular momentum. The latest version Version 4 Release 2 is now a complete replacement for NuShell. NuShellX can now calculate the isospin T of the eigenvectors quite quickly.

## I. INTRODUCTION

As computers have evolved and become more powerful nuclear shell model codes have been able to tackle larger and larger calculations. In addition shell model techniques have also evolved. Two significant break through's were the Glasgow, m-scheme code of Whitehead et al [7] and the codes Nathan and Antoine by the Strasbourg group, Caurier et al [2]. A significant contribution was made by OXBASH, Brown, Etchegoyen and Rae [6]. This code had developed from an earlier code I wrote with Nigel Godwin, The Oxford Shell Model code [1], a semi m-scheme/J-scheme code, developed without any knowledge of the Glasgow code [7], by Godwin and Rae.

Last year I completely rewrote OXBASH [6] from the ground up. In addition I solved some of the problems OXBASH [6] was having with angular momentum projection from high spin j levels. The new code NuShell [8] was tidier, more accurate and significantly faster than OXBASH [6] and has now replaced OXBASH [6].

Nushell [8] is still limited by the fact that it has to store a large number of Hamiltonian matrix elements in memory for maximum efficiency. Although the full square matrix does not have to be stored this problem severely limits the range of the calculations possible.

I realized what was needed was a fast method that did not involve storing the Hamiltonian matrix at all. Antoine [2] was famous for its 'on the fly' method of the calculating matrix elements. (Personally I think the term 'on the fly' is misleading and should not be used.) But Antoine [2] was an m-scheme code and required the use of large m-scheme vectors even although the matrix was never calculated explicitly.

Nathan [2] was a J-scheme code and the J-scheme vectors have much smaller dimensions. I realized that the ideas behind Nathan [2] were similar to the old shell model code I had written with Nigel Godwin called MultiShell [1]. In MultiShell [1] the shell model space was divided up into subspaces. In each subspace shell model calculations were performed with the Oxford Shell Model code [1]. The results of the sub-shell calculations were then coupled in spin and isospin to form a basis for the complete shell model space. Normally a highly truncated MultiShell [1] basis was used formed from only the lowest sub-shell eigenvectors. The Oxford Shell Model code [1] also calculated the operators required to calculate the Hamiltonian in the full shell model space with MultiShell [1].

In fact there was very little difference between Nathan [2] and MultiShell [1]. MultiShell [1] actually calculated a matrix but if instead it had used the Lanczos algorithm it would have been equivalent to Nathan [2] in concept, so some of the Nathan [2] ideas are not original, although MultiShell [1] was never published and gained no popularity and Nathan [2] was developed without knowledge of MultiShell [1].

In the following there are references to version 4 release 2 of NuShellX. The current version is 4. Version 4 Release 2 should be available in April 2008.

---

\*Electronic address: [william.rae@virgin.net](mailto:william.rae@virgin.net); URL: <http://knollhouse.org>; Thanks to Jussi Toivanen, Alex Brown and Mihai Horoi

## II. METHODS

### A. Initial Preparation

Rather than convert MultiShell [1] in its entirety, initially I have borrowed the idea of splitting the shell model space into neutron and proton spaces from Nathan and Antoine [2]. NuShellX is modular and it starts off using NuShell [8] modules. Indeed in the first stage NuShellX starts as an m-scheme calculation like Antoine [2] calculating the m-scheme bases for the protons and neutrons separately. These m-scheme bases are quite small and tractable even for quite large calculations such as  $^{56}\text{Ni}$  in the full  $fp$  shell. Next states with good total angular momentum  $J$  are projected from particular m-scheme Slater determinants to form a complete J-scheme basis for the neutrons and protons separately.

The second stage in the calculation uses the m-scheme basis to calculate matrix elements of the operators  $a_{j_a m_a}^+ a_{j_b m_b}^-$  in the m-scheme. From these, using the J-scheme vectors obtained by angular momentum projection, J-scheme matrix elements can be obtained for protons and neutrons. These can be reduced by inverting the Wigner-Eckart theorem to provide reduced transition densities

$$\frac{\langle J' || [a_{j_a}^+ \tilde{a}_{j_b}]^\lambda || J \rangle}{(2\lambda + 1)^{1/2}}$$

The J-scheme vectors have several good quantum numbers. These are  $\mathcal{N}$  the number of protons or neutrons,  $J$  of course, and a partition label  $P$  giving the number of protons or neutrons in each  $j$  shell. The  $j$  shells are labeled by  $N, n, l, j$  here  $N = 2n + l$  and it uses the convention that  $n$  starts at 0. A further quantum number  $\alpha$  is required to distinguish the different states within the same partition and with the same  $J$  quantum number. Particular importance will be paid to the blocks of states with the same  $J$ , ie the sets of states with fixed  $J$  and all possible partitions and all possible  $\alpha$ . These will be referred to as proton or neutron blocks. Similarly the sub-blocks with a fixed value of  $J$  and a fixed partition  $P$ , ie the sets of states with fixed  $J$  and  $P$  and all possible  $\alpha$  play an important role.

The reduced transition densities are stored in look up tables for use with binary look up or binary sorting. The index is made up of all the initial and final quantum numbers and  $j_a, j_b, \lambda$ . This is compactly stored in a `character(len = 8) :: trda(..)` array and uses the Fortran 95 intrinsic functions for ordering strings and order comparison. Each quantum number or a pointer to a group of quantum numbers is stored in an 8 bit character variable giving a useful range of 0 to 255 for each of these variables. The partition numbers are not stored in these character arrays. An Fortran type structure of integer arrays is defined that can be interpreted as a two dimensional bit or logical array. One array is associated with each pair of initial and final,  $J$  and  $J'$ , blocks  $L_{P,P'}^{J,J'}$ . A bit in this two dimensional array is set (to TRUE) for initial and final partitions connected by the reduced transition matrix elements.

The advantage of splitting the basis according to neutrons and protons is that the reduced transition matrix elements are one of two sets of matrix elements required to do the complete calculations. This result follows from the fact that the number of protons and neutrons is conserved. In MultiShell [1] many more different off-diagonal matrix elements were required. The above reduced transition densities (RTD's) will be used to calculate the n-p interaction terms. The other matrices required are the n-n and p-p interaction matrices. These are easily calculated using NuShell [8] modules and then the final nn and pp matrices are input into NuShellX along with the RTD's.

Thus the third stage in the calculation is to run the NuShell [8] modules to produce the nn and pp matrices. Again this is very fast since the dimensions are small. This preparation work need only be done once for each nucleus. The main calculations for all total angular momenta  $\mathcal{J}$  share these input files. Nevertheless the time taken in initial preparation is always insignificant compared to the time taken to calculate energy levels and eigenfunctions for any  $\mathcal{J}$ .

### B. The Main Calculation

After the preparation is complete the NuShellX Lanczos module is run. This is essentially the core of NuShellX and this has to be the most efficient code. There are two choices a normal Lanczos module using the standard Lanczos iteration scheme and a Thick Restart Lanczos module which uses the Thick Restart iteration scheme.

The Thick Restart iteration scheme is preferred. It often converges in many fewer iterations and it uses smaller disk files. The basic idea behind the Thick Restart method is that it starts in standard Lanczos mode for a limited number of iterations. I have set the limit at around 80 iterations. When the routine has done 80 iterations it diagonalizes the tridiagonal matrix. It uses a formula to determine how many eigenvectors to keep. This is usually about 40. It

constructs these 40 eigenvectors. It can be shown that these 40 eigenvectors satisfy a Lanczos iteration recurrence relationship but with a very sparse but non tridiagonal matrix. Then 39 of the remaining vectors are discarded keeping only the 80th to continue the Lanczos process. A normal Lanczos procedure then generates vectors 41,42.. etc producing tridiagonal elements in the sparse matrix. This process is called a restart. Iteration continues again for a further 40 iterations until approximately 80 vectors are stored and a second restart is performed. Of course the routine often terminates before reaching a second restart. After the first restart a Householder routine is required to diagonalize the sparse matrix.

The standard and Thick Restart Lanczos modules only differ in their Lanczos subroutines. The details that follow apply to both modules. The Lanczos modules start by reading in general information and tables for the calculation. They then start to construct the basis. First they couple the proton and neutron J values to the total angular momentum  $\mathcal{J} = \mathbf{J}_p \times \mathbf{J}_n$  where for an untruncated calculation the proton and neutron J values run over all possible values for the specific model spaces. One can often truncate the model space by putting lower limits on the maximum J values, combined with an upper threshold on the sum of proton single particle energies and separately on the sum of neutron single particle energies. These amount to partition truncations.

The total number of such angular momentum combinations is referred to as BAS and this is printed out by the code. BAS is an important factor for determining the time it will take the code to run particularly as a function of total  $\mathcal{J}$ . Recall that every  $\mathbf{J}_p$  and  $\mathbf{J}_n$  is associated with a block of vectors of dimensions that we will denote by  $\mathbf{D}_p$  and  $\mathbf{D}_n$  respectively. So corresponding to each BAS there is a super-block of  $\mathbf{D}_p \cdot \mathbf{D}_n$  vectors. It is often convenient to think of this super-block of vectors as a matrix of vectors, called a BAS matrix with  $D_p$  rows and  $D_n$  columns. This emphasizes the fact that the basis is factorisable into proton and neutron parts. NuShellX uses this to maximum advantage for speed but it does mean that simple truncations that depend on both neutron and proton partitions cannot be efficiently implemented.

If the sub-matrices of the one body transition operator above for protons are denoted by  $A_{P,P',j_a,j_b}^{J,J',\lambda}$ , these matrices are of dimension  $(\alpha, \alpha')$ . The neutron matrices can similarly be denoted by  $B_{Q,Q',j_c,j_d}^{I,I',\lambda}$  with, I, now representing neutron angular momentum and Q denoting neutron partitions. We will denote the dimensions of these matrices by  $(\beta, \beta')$ . Now it is easy to show that the p-n contribution to the new vector in a Lanczos iteration step  $U^{n+1} = H_{p-n} U^n$  is given by the following formula where we update one BAS matrix on the left from one BAS matrix on the right

$$U^{n+1}_{J,I} = \sum_{j_a,j_b,\lambda,P,P'} A_{P,P',j_a,j_b}^{J,J',\lambda} \times \sum_{j_c,j_d,Q,Q'} F(j_a,j_b,j_c,j_d,\lambda) \Gamma(J,I,J',I',\lambda\mathcal{J}) B_{Q,Q',j_c,j_d}^{I,I',\lambda} U^n_{J',I'}$$

Here  $F$  are the two body n-p interaction matrix elements transformed by a p-h transformation,  $\Gamma$  is essentially just a Racah coefficient, and the sums are over all possible values of the quantum numbers. The B matrices only act on the columns of the right hand side BAS matrix coupling partitions  $Q'$  to partitions  $Q$  on the output BAS matrix columns (or left hand side BAS matrix columns). Similarly the A matrix only couples rows  $P'$  to  $P$ . To speed up the multiplications only those rows and columns of the input BAS matrix which can couple are multiplied by A and B. This is done by projecting the logical OR of the bits in the logical bit arrays  $L_{PP'}^{JJ'}(A)$  and  $L_{QQ'}^{II'}$  corresponding to the full block matrices A and B onto for example the P and  $P'$  axes separately. These projected OR vectors, one dimensional logical vectors, determine which rows and columns, or more specifically which partitions, of the initial and final BAS matrices will be required during the multiplication process.

The appropriate rows and columns of the initial BAS matrix are read into a buffer. First these are multiplied by the sum of B matrices as indicated in the above equation. Then this sum is multiplied by one A matrix and this process is repeated for each A matrix. It is clear that if the dimensions of A and B are unequal the process is most efficient for larger A smaller B thus NuShellX automatically assigns protons to A or B and neutrons to B or A so that when the number of protons and neutrons are different the most efficient choice is made. All multiplications are applied to unit spaced, one dimensional arrays which enables the compiler to invoke the highest level of vectorization. This involves performing multiplications in unusual orders and transposing the BAS matrices for the A matrix multiplication. A further transposition is required afterwards. The result of all this work is a buffer containing only those rows and columns which have to be updated the final BAS matrix. This is repeated for all the initial and final BAS matrices. An OpenMP parallel loop over  $I,J,I'$  is used to speed up the calculations.

Finally the initial BAS matrices are multiplied by the nn and pp matrices. This is a simpler process since the nn matrices only couple different rows of the BAS matrices being independent of the columns, while the pp matrices only couple the columns of the BAS matrices independent of the rows. Similar methods are used but this is a much faster

operation and optimization less important. OpenMP loops are used for these multiplications. The n-p multiplication time dominates the iteration time. Other processes add less than 10% to the iteration time.

There are similarities in the above methods to those used by Jussi Toivanen [3] in his code EICODE [4] and I am working on a paper [3] with him and we have had many private communications [3]. I think our methods are nearly identical except for how we handle and store/access the RDM's and do the matrix multiplications. Jussi has developed a SuperMatrix method which promises to be faster. However I think we achieve iteration times which are comparable to within a factor of 2 at maximum based on his original method.

When the iteration terminates based on energy level comparisons over the last few iterations the Lanczos modules exit writing their vectors to disk. A further module reads these vectors and associated information, re-diagonalizes the final sparse matrix and calculates the required number of eigenvectors and writes these to disk. It also does a simple analysis of the wavefunctions giving average single particle shell occupation numbers, leading partitions, and a breakdown of contributions of different proton and neutron angular momenta and writes this to disk.

### C. Observables

An important part of a shell model code is its ability to predict transition rates, spectroscopic factors, cluster spectroscopic factors etc. NuShellX is no exception. The second most important module generates data from which all these quantities can be calculated. This transition module uses a formalism very similar to the Hamiltonian operator. The formula for a general transition operator reduced matrix element is

$$\langle U_{\mathcal{J}} | \Sigma_{P, P', J, \lambda_a} \mathcal{A}_{P, P', j_a, j_b}^{J, J', \lambda_a} \Sigma_{Q, Q', I, I', \lambda_b} \Gamma(J, J', \lambda_a, I, I', \lambda_b, \mathcal{J}, \mathcal{J}', \lambda) \mathcal{B}_{Q, Q', j_c, j_d}^{I, I', \lambda} | U_{\mathcal{J}'} \rangle$$

Here  $\Gamma$  is now essentially a 9-j coefficient and  $\mathcal{A}$  and  $\mathcal{B}$  are generalizations of the one body density matrices A and B. They can be matrices of any one or two body operator (and in principle 3 body operators but this is not yet supported). They can also be matrices of the identity operator or number operator. Currently supported operators are  $\hat{I}$ ,  $\hat{N}$ ,  $a^+$ ,  $a^-$ ,  $a^+ a^-$ ,  $a^+ a^+$  and  $a^- a^-$ . Any of these 7 operators can be used for both  $\mathcal{A}$  and  $\mathcal{B}$  matrices giving a total of 49 possibilities.

Again NuShell [8] is used to calculate the necessary m-scheme matrix elements then these are converted to J-scheme matrix elements which are reduced by inverting the Wigner-Eckart theorem as described earlier for the A and B matrices. Optimization of the transition module is less easy since many individual matrix elements have to be explicitly stored, but OpenMP loops are used wherever possible.

The actual transition rates and spectroscopic factors are calculated by other modules, one for 3 and 4 particle cluster transfer and the other handles all other cases. These codes do not provide all the multiple options of Alex Brown's 'dens' code [6]. But they cover all the basics and are 'open source' so you can add your own options.

### D. The Control Module

Some of this discussion applies to Version 4 R 2 only.

To get you started I have written a control module that asks questions and will provide help on a question by question basis. Given an interaction file in a very slightly modified OXBASH [6] format it will produce all the input files and run the necessary modules for you. But it is not perfect and you might find there are limitations that do not suit you. If this is true you will need to modify the source code or write your own control program. If you wish to rename the code the convention is NuShellX@Institution. You need to ask my permission to do this and to add your name to mine as a co-author. I will almost always grant permission but you are then responsible for maintaining this version and correcting errors including those of my making although I will try to help while I can ( I have Parkinson's and this is progressive.)

This control module is similar to the easy to use SHELL command in the old OXBASH which was first written by Nigel Godwin and myself in 1977. I think it is much more user friendly than the Antoine interface and you should quickly become familiar with it. It is very similar to the NuShell control program and Alex Brown tells me he has switched from his OXBASH compatible interface to NuShell, to NuShell's own control program which I wrote.

There are five sections to the control program. The (Z)uker [10] option will be discussed separately. The other 4 are:

### 1. (*M*)odel Space

This section allows you to set up the shell model space eg sd shell, fp shell, sd<sub>pf</sub> shells. The input is a bit tedious but it only has to be done once. You are asked for a 5 character model space name, the total number of j orbits and the number of proton orbits. You then input the proton levels first followed by the neutron levels. The order should be identical to the order in the .int file you intend to use. The format for the sd shell is 0d3/2, 0d5/2, 1s1/2. In this section you can also set truncation limits on proton and neutron spaces separately by one of three methods.

1. Set a limit of  $\Sigma 2n + l$  summed over all protons or neutrons. Separate proton and neutron limits are implemented in Version 4 while combined proton + neutron limits are implemented in Version 4 Release 2 but very inefficiently.
2. Set limits on the minimum and maximum nucleons in each shell. Since each level is designated proton or neutron these are independent limits for protons and neutrons.
3. Set limits on the number of protons or neutrons in each major harmonic oscillator shell. Separate proton and neutron limits are implemented in Version 4.

These truncations are saved on disk in the model space file with extension .sps. The easiest way to modify these truncations is to edit the .sps files and with experience you should be able to create your own .sps files directly.

### 2. (*L*)evels

This part of the control program sets up the energy level calculations. You need a model space file with the order of the orbits consistent with an OXBASH [6] or NuShell [8] interaction file extension .int. An OXBASH [6] interaction file needs one comment line, which must be the first comment line, containing the core mass, the mass for which the interaction was defined - usually core + 2 - and the mass scaling power often 0.3, 0.333333, 0.35. If there is no mass scaling put 0.0. Note please also read all of 2. below, both proton and neutron single particle energies need to be added even when using isospin format.

1. The module first asks for a Nucleus name of 5 characters. Normally Ni56 for example is a sensible choice but any number of characters up to 5 will do. Next it asks for an interaction code, you can use any character that will remind you of the interaction. You are then asked for the model space, the number of protons and the number of neutrons. The code generally assigns the A matrix to the larger number, but you can override this using negative numbers for one or the other. Of course absolute values are eventually used.
2. Next the module gives you the option of using either the same interaction file for the p-p, n-p and n-n interactions or separate files. The latter option is useful for heavy nuclei or adding the Coulomb interaction. You are then asked if the interaction files are in standard OXBASH [6]/NuShell [8] isospin format or whether they are np interaction files. The code accepts either. An undocumented feature of the programs that read the interaction files is they will add several interactions with different normalizations and use the sum as the interaction. They can also print out the total interaction to a .txt file. A single interaction file consists of several comment lines marked by an exclamation mark ! . Apart from the first NuShellX ignores these comment lines. The first line after the last comment should contain an integer, which is ignored in single interaction mode, followed by the proton single particle energies followed by the neutron single particle energies. The next line contains the first two body matrix element and this continues to the last two body matrix element. To combine more than one interaction in an editor put all your interaction files in sequence with no blank lines. Make sure there are comments for each interaction. Delete all the single particle energy lines including integer except the first single particle line. On the first single particle energy line change the integer to -n where n is the number of interactions you want to add. Edit the last comment line for each interaction changing only ! to & which stands for 'and'. Finally when each interaction program runs you will be asked for the normalizations for each interaction in order.
3. In addition to model space truncations there are four other truncation options. The first of these options is to put a cut on the sum of the single particle energies (spe) of the protons and neutrons separately. If the sum of the spe energies for a particular proton or neutron partition exceeds the appropriate cut, the partition is discarded. This truncation reduces the dimension and it can produce very accurate results for the first few energy levels and wave functions of the combined system. The second of these truncation options (V4R2) is based on the formalism of Defour and Zuker [10]. This will be discussed separately below in a special section. The third option enables you to truncate the multipolarities used in the n-p interaction  $\lambda$  [2]. You are given the

choice of setting the maximum and step. If these are left as 0, the default is  $0 \ 2j_{max} \ 2$ . This truncation and the second truncation option do not affect the dimension of the basis but reduce the time for each iteration. The fourth option is discussed in 5 below.

4. The module then moves to questions about angular momenta. You are asked for the minimum, maximum and step (delta) for the total angular momentum  $\mathcal{J}$ . All angular momenta are input as integers equal to twice the value of the angular momenta. This caters for half integer  $\mathcal{J}$  values. So an input of  $0 \ 16 \ 4$  would result in the calculation of the states with angular momentum 0,2,4,6,8. Next the module suggests values for the minimum, maximum and step (Version 4) of the proton and neutron NuShell [8] calculations. For an untruncated calculation the minimum and maximum should be the maximum possible allowed by the Pauli principle with a step of 2 and this is what will be printed out. These values will be wrong if either you have truncated the basis above or truncate the basis with either the single particle energy options or the parity options that follow so you have to work the new limits out by hand.
5. Test calculations suggest that the higher proton and neutron J values contribute little to the wave functions of the lowest eigenstates of the total system. So you can use this here to provide another form of truncation. In addition the contributions from unnatural parity states of the neutrons and protons are significantly less than that from natural parity states so you could choose the minimum J and step for the protons and or neutrons to be different from their default values. This truncation like the single particle energy truncation reduces the dimension.
6. Next you are asked for the parity for the proton and neutron parts of the calculation. Positive  $+1$ , negative  $-1$  and both positive and negative  $0$  are allowed for protons and neutrons but only positive or negative was allowed for the combined system in version 3. In Version 4 both positive and negative parities will be allowed for the combined system which will produce the lowest positive and negative parity states being simultaneously diagonalized. For a parity conserving Hamiltonian the Lanczos iteration scheme will produce states which are 100% positive or negative parity.
7. The final questions in the Levels section asks you how many energy levels you require to converge and whether you wish to delete existing energy level files so that the PS graphics program will produce an energy level diagram of levels from the present calculation only. You have the choice of standard Lanczos or Thick Restart Lanczos. Finally you are asked if you want to set up basic tables. Normally the answer to this is always yes, but if you have already done part of a calculation for limited angular momentum values and you have ONLY changed the total angular momentum range but nothing else you do not have to set up tables.
8. It is not documented elsewhere but Thick Restart Lanczos calculations can be restarted to generate more energy levels. Edit the Nucleus//interaction code.spe file eg Ne20w.spe This file contains three lines. The first line specifies the single particle energies. The second line contains the matrix element re-normalization factor. The third line specifies the number of levels. Change the number of levels to a new larger value. Add a fourth line with a non-zero integer eg  $1$ , free format and rerun the Thick Restart Lanczos module followed by the eigenvector module. You can find out how to do this by editing the ShellX.bat file which will also show you how a complete calculation is run.

At the end of the Levels section in Version 4 Release 2 the Control module writes out an .inf file which is a record of the options you have chosen and the input you have given. Keep this file for your own records! But you **must** also keep it because other options use this file ((O)verlap and (T)ransitions)

Once the calculation completes the output files have extensions .xvc, .xva, .eps, .lev, and .lp where the xvc and xva files are unformatted (binary) and the lp and lev files are formatted (ASCII). The xvc file contains the eigenvectors. The xva file contains the complete analysis tables that are partially reproduced in editable form in the lp file. The lev file just contains the spins and parities and is used mainly by the Postscript graphics program that generate the energy level diagram in the eps file. ( All the Lanczos vectors are stored in binary files with either extension .lnz or .trl with information required to read these files being stored in the binary files .abz . )

### 3. (T)ransitions

The transitions section of the control module first asks you details about the initial and final nuclei. You must first calculate the energy levels of both nuclei to obtain eigenvectors. First you are asked about the initial nucleus. Specifically the name you chose for the nucleus eg Ni56, the interaction code you chose, which matrix was assigned to the neutrons ie A(a) or B(b), and the total angular momentum minimum, maximum and step you want for the

transition calculations. Obviously this must be a subset of those you calculated in the Levels section. On the same input line you are asked how many type 'a' and type 'b' nucleons were involved in the energy level calculations. This is just the proton and neutron numbers that were assigned to the A(a) or B(b) matrices. Next the same questions are repeated for the final nucleus, which can be the same nucleus as the initial or a different nucleus. The initial and final states must have been calculated in the same model space. In Version 4 R 2 the module uses the .inf files to enforce this.

You will then be required to choose a name that will identify the transition files. This can be up to 6 characters. All transition files for this calculation will start with this name. This allows you to keep different transition files organized. Using the initial and final proton and neutron numbers the module calculates what operator combinations are allowed. For some cases where proton or neutron numbers differ in the initial and final nuclei by +/-1 or 2 there is only one choice and this will be chosen for you automatically. When there is no difference between proton or neutron numbers in the initial and final nuclei you will be offered the choice of the identity operator, the number operator or reduced transition densities. The latter do not require new calculations since they have already been calculated for the Hamiltonian operator.

- For electromagnetic transitions and moments choose the proton (or the neutron) operator to be a transition density and the neutron (or proton) operator to be the identity operator. The module will automatically calculate the alternative combination. For these calculations you will also be asked for the proton effective charge, the neutron effective charge, the orbital angular momentum  $g$ -factors for protons and neutrons and the spin  $g$ -factors for protons and neutrons. The effective charges are absolute so the default value for the protons is 1.5. If you enter 0 0 0 0 0 0 the program will use the default values that are adequate for most calculations, otherwise enter the values you want. The exception is the neutron effective charge. If you wish to set this to zero entering 0 gives the default value. So to set it to zero enter 999. Finally you will be asked for the mass and charge of the nucleus.
- For spectroscopic factors either one operator will be chosen automatically or both will be chosen automatically in the case of pn transfer, three particle transfer or four particle transfer. Where only one operator is chosen for you the second should always be the identity operator. The three particle and four particle transfer options produce cluster transfer spectroscopic factors equivalent to SU(3) cluster spectroscopic factors. The cluster spectroscopic factors produced by NuShellX sum to the  $G^2(SU(3))$  limit as defined in the literature [9]. Please **note** that the modules that actually calculate particle transfer spectroscopic factors only work with one, two, three or four  $a^+$  operators not with  $a^-$  operators. So choose your initial and final states appropriately.
- Special care has to be taken with charge exchange. The operator combination must be  $a^+ a^-$ , since the coupling order inside the program is  $\lambda_a \times \lambda_b = \lambda$  so the 'a' type nucleons must be associated with the creation operator. The alternative combination is a completely different matrix element. To achieve this you may have to override the default allocations to A and B matrices and this is done by inputting negative proton or neutron numbers (see Levels section). Note that in some cases if you do reverse the A and B matrices you will have the 'b' nucleons associated with the creation operators. If you do not get all the matrix elements in .tra files and .lp files then try changing your initial and final states.

The module then asks you for the minimum and maximum values of  $\lambda$  or Jop. These must be consistent with your operator choices ie odd for half integer values, and even for integer values since you are required to input  $2*Jop$  ie  $2\lambda$ . Then you choose the number of levels in the initial nucleus and the number of levels in the final nucleus for which you require transition data. These must be less than or equal to the number of levels you requested for each nucleus. Finally you have the option of changing the definition of the sign of the radial wave functions at infinity. NuShellX calculates transition rates using Harmonic Oscillator wave functions defined as positive at the origin. However the interaction you use actually defines the convention used for the eigenvectors. Normally interaction files use the same definition and so the standard answer to this question is no. But the possibility of changing it exists if you need to do so. This concludes the input, the necessary files are created, the necessary modules are run for you and a file whose name starts with the characters you chose with an extension .lp contains the results. Other files produced start with the same characters and end with extensions .tra and .trs. The tra is a formatted (ASCII) output from the transition code before final processing and the trs file is an unformatted (binary) version of this file.

#### 4. (O)verlaps V4R2

This section allows you to work out the single particle plus two-body matrix elements of a scalar operator (cf the Hamiltonian) between eigenvectors of the Hamiltonian for a single nucleus. One example might be the matrix

elements of a different Hamiltonian. Another simple example is  $T^2$ . Here the single particle energies are all 0.75 and for  $T=1$  diagonal two body matrix elements are all 0.5, while diagonal  $T=0$  two body matrix elements are all -1.5. All off diagonal two body matrix elements are zero. Make sure you include a complete set of diagonal matrix elements. The diagonal matrix elements of this operator are  $T(T+1)$  for the whole system if the original interaction was of iso type, while the off diagonals will be zero.

To calculate these matrix elements chose option 'O', enter the name of your .inf file for the original calculation, enter the name of your new interaction/operator .int file eg t2fp, the name of your  $T^2$  operator file for the fpshell for example, and if you have not setup files for this Nucleus/operator combination before answer y to setup files. The output files have extension .ovl (text) and .ovx (binary).

There is a separate module which will read the .ovx overlap file, add optional diagonal matrix element from a text file .ovd and diagonalise the result. This can be useful to find the effects of adding small terms to the interaction. Following the example here a trivial case would be to add  $\alpha T^2$  to the original Hamiltonian. But there are many non trivial examples. Note that there can be no mixing with eigenvectors not explicitly requested ie the mixing is only within the small subspace of the no\_levels requested. The output is in a text file with the same name as the .ovx file with extension .lp. It contains both eigenvalues and eigenvectors.

## 5. Notes

- Warning on Single Particle Energies:** In order to work properly the Lanczos routines require the initial energy calculation to produce a negative result. Since the initial vector is just a random vector this requires the diagonal nn and pp matrix elements all to be negative. This can be achieved simply by ensuring that all your single particle energies are negative, or mostly negative. The problem will show up, if it does show up, as extremely negative, or overflowing negative eigenvalues. Subtracting a constant value  $E_{sp}$  from each of your single particle energies will solve this problem. Your new eigenvalues  $\mathcal{E}_n$  will be equal to your original eigenvalues minus the sum of the single particle energies ie  $\mathcal{E}_n = \mathcal{E}_o - E_{sp}N$ , where  $N$  is the total number of nucleons, protons and neutrons.
- Restrictions on Spectroscopic Factors:** Although you can calculate transition operators of the form  $a^-$  and  $a^- a^-$ , only  $a^+$  and  $a^+ a^+$  are supported in the modules NuTrans(NuShell), NuTrx and NuClx (NuShellX). (The names of the modules and their function is discussed in next section.) This does not actually restrict you. It just requires swapping the initial and final states.
- Using the Environment Variables:** Most of the dimensions of the arrays in NuShell and NuShellX are determined by the codes themselves, or from information in the various files produced along the way. So with the exception that you may run out of memory you do not need to worry about array dimensions. There are some exceptions where it is not possible to calculate the required memory before the code has run. These arrays can turn out to be too small for larger calculations. If you do meet this problem you will get an error message which gives you two options. The first option is to modify a parameter in the Fortran source code. The message will give you the name of the parameter, which file it is in, but usually, it still cannot tell you what value to use, but it will give you a lower limit. Using this method you have to compile and rebuild the code. The **alternative easier** option is to set an environment variable to define the dimension. The message gives the environment variable's name which must be entered in UPPER CASE. It will also give the same lower limit for the value. You need to try a value greater than this lower limit. To set the environment variable (in Windows) right click on the 'My Computer' icon, choose 'Properties', then in the new window click on 'Advanced'; then click on 'Environment Variables'. In the 'user' area click 'Add' and in the new window enter the name of the variable exactly as printed in the message and a value about 50% bigger than the minimum. Click OK etc until the 'My Computer' window closes. Before running the code again open a new CMD shell. If it is still too small, go though the same sequence of steps but do not use 'Add', select the variable and click 'Edit'. On Linux the environment variable should be exported in a command added to your shell resource file so that it will be set when any shell starts.
- Directories:** It is advisable to use a fresh directory (folder) for each calculation. If NuTRL fails it may leave a damaged file, that will be used next time you run NuTRL for the same calculation. Alternatiely delete the .trl or .lnz file if you ran NuLnz and .abz It is **definitely not** advisable to run NuTRL and NuLnz in the the same directory, since they share the same .abz file and NuVec will know whether to use the .lnz file or the .trl file. (Module names and purposes are discussed below.)

### III. MODULE NAMES AND PURPOSE

In this section I briefly list all the modules and their purpose and some details on the important input files.

- NuShellX is the control module name. This acquires input from you and using a pre-existing .int file will produce all the input files required to run a calculation. These include .sps, .nus, .spe, .op, .oph and .me files. The control module also runs the calculation for you and produces a ShellX.bat file which will rerun the calculation. If you want to keep this file and avoid it being over written rename it eg ren ShellX.bat MyCalc.bat. NuShellx.f90 is the source code that you may wish to edit and rebuild to suit your own preferences. See section V on Compiling and Building the codes.
- NuBasis is the m-scheme basis module. It requires just one .nus input file. It outputs files containing the m-scheme vectors and other information files.
- NuProj is the angular momentum projection module. It requires the same .nus file as NuBasis. It outputs files containing the J-scheme basis vectors defined in the m-scheme basis, plus other information files.
- NuOper is the program that converts the .int file into an m-scheme operator. It requires an .nus input file, a single character interaction code, a .int interaction file and either 'np' or 'i' to tell it what type of .int file it is using. It outputs a .op file and optionally useful text files. Optional output is controlled by the last entry in the .nus file output\_control in version 3. In version 4 up to 3 more lines may follow, the first with two integers, a second with an ASCII file name and a third tied to the second with two real numbers. The output\_control is a single integer with value 0 after the last long row of multiple integers. Edit this line and enter 3 or higher for additional output from all modules.
- NuMatrix is the module that calculates the pp and nn matrices using the J-scheme vectors expressed in the m-scheme basis. It requires a .nus file and the one character interaction code. It outputs the pp or nn Hamiltonian matrix in a semi-orthogonal basis. The rhs vectors are not orthogonal the lhs vectors are orthogonal. These matrices are stored with the extension .mtx .
- NuOrth is the module that completes the orthogonalization of the matrix produced by module NuMatrix. The module requires a .nus file and interaction code. It outputs a fully orthogonalised J-scheme matrix in a file with extension .mat .
- NuOp takes an identical input to NuOper, but unlike NuOper it does not produce an m-scheme operator file. It produces a J-scheme particle hole transformation of the np interaction and writes this out to a binary file with extension .oph. It also produces a .txt file containing the ph matrix elements. It can also add interactions see NuOper above for details.
- NuOpm is the module that calculates all the m-scheme operators required by NuLnz, NuTRL and NuTra, see below. This includes transition densities etc. It needs the .nus files for the initial nucleus and the final nucleus, a character code of up to 4 characters defining the operator eg, 'i' , 'a+a-' or 'a+a+' , and a one character code to distinguish this operator file from other operator files. The output is a file with extension .trm in binary format.
- NuOpd is a module that converts the m-scheme operator matrix elements to reduced matrix elements in the J-scheme using the projected J-scheme vectors produced by NuProj, the m-scheme matrix elements produced by NuOpm and the Wigner-Eckart theorem. The input must be identical that for NuOpm which is run first. NuOpd produces a binary file with extension .trd
- NuOpdZ is a special version of NuOpdZ (V4R2) designed to make highly truncated Dufour-Zuker calculations faster. In addition to the input for NuOpd it requires the name of the interaction file.
- NuLnz is the standard Lanczos module. This takes a separate .nus file for each total angular momentum eg Ni560, an A RTD matrix file eg Ni56a , a B RTD matrix file eg Ni56b and a .oph pn interaction file eg Ni56g. The output files are binary and have extensions .lnz and .abz .
- NuTRL is the Thick Restart Lanczos module. This takes a separate .nus file for each total angular momentum eg Ni560, an A RTD matrix file eg Ni56a , a B RTD matrix file eg Ni56b and a .oph pn interaction file eg Ni56g. The output files are binary and have extensions .trl and .abz .

- NuTRLZ is the Defour-Zuker version of NuTRL(V4R2). It works directly with the Defour-Zuker eigenvector multipole expansion rather than the standard multipole expansion. In addition to the input for NuTRLZ it requires the name of the interaction file.
- NuXpct is a non iterating version of NuTRLZ for overlap calculations (V4R2). It requires the same information as NuTRLZ.
- NuPrtb diagonalizes any matrices from NuXpct . You run this program by hand at the command line, it is not supported by the Control module. It requires the name of the file output by NuXpct and an optional text file with the same name as the .ovx file but with an .ovd extension. This enables you to add additional diagonal matrix elements to the matrix. In the .ovd text file white space will be ignored on input but you must have at least no\_levels floating point 'f' format numbers in the file (V4R2) eg -92.1 -90.9 -88.1 -85.4 ..... The output is in a text file with the same name but a .lp extension
- NuVec is the eigenvector module which takes exactly the same input was used for NuLnz or NuTRL. It outputs files with extensions .xvc, .xva, .lev and .lp which have been previously described.
- NuTra is the transition module it takes an input of the form Ni572 Ni56p Ni56i Tr1n Ni560 . Here the first input refers to the .nus file for final state the  $\mathcal{J} = 5/2$  states in  $^{57}\text{Ni}$  the next two files are the operator reduced matrix element files, then the .me file and finally the initial nucleus .nus filename for  $^{56}\text{Ni}$   $\mathcal{J} = 0$  states.
- NuTrx is the module that calculates reduced electromagnetic transition rates ie BE(L) and M(L) matrix elements as well as B(GT) and one and two particle spectroscopic factors. It requires as input the .nus files for the final nucleus, the initial nucleus and one or two appropriate .tra files depending on the results required. The electromagnetic transition rate calculations require two .tra files one for protons and one for neutrons. Everything else requires only one .tra file. NuTrx produces one output file with extension .lp which has been discussed above.
- NuClx is a module similar to NuTrx and requires the same input with just one .tra file. NuClx produces one output file with extension .lp which has been discussed above.
- NuDiop is the Defour-Zuker interaction preprocessor for np terms in the interaction only. It is only used when you chose DZ truncation (V4R2). It requires the name of the .oph np particle hole interaction, a real value representing the cut in MeV on the  $|e|$  values in this method (see section IV) below and real value representing the cut in amplitude on the  $|u|$  coefficients of the  $\hat{u}$  vectors (see section IV) below.
- NuDZop is the engine module for the Defour-Zuker interaction processing for full isospin interactions. It has options for removing spurious states, editing multipole strengths and separating the collective part of the interaction from the statistically distributed weaker components. It also give great insight when comparing interactions. It is not designed to be run on its own (V4R2). It requires similar input to NuDiop except that it takes an isospin interaction .int file instead of the .oph file. It also requires an isospin model space file and information on the options you want. You can chose to remove the coupling term in the interaction that couples spurious and non spurious states. You can either apply cuts as in NuDiop or Edit the interaction  $e$  values and  $u$  coefficients with Notepad (supplied with Windows. Linux users will have to alias the name Notepad to some editor in Linux). Before converting the calculation back to standard form it displays the final interaction again in a Notepad window. Editing this second window will have no effect!! It outputs an interaction in OXBASH / NuShell format with a .int extension in either iso or np formalisms.
- NuDZOper is the control program for NuDZop. It can be run from the command line or by selecting option Z in NuShellX (V4R2). It obtains all the information required to run NuDZop then runs it. It has two options (M)odel to create an isospin model space .sps file and (O)per to run NuDZop. Finally e(X)it exits this module and will return you to the console or the Control module NuShellX depending on how you initiated it. Note 'X' does not exit from NuShellX if you are using the 'Z' option.

#### IV. DUFOUR ZUKER

This is a brief description of the formalism of Marianne and Andrés Zuker [10]. The (isospin) interaction is transformed into particle hole formalism. So it can be written as

$$H = \sum_{rstu\gamma} [\gamma]^{1/2} f_{\{rs\}\{tu\}}^{\gamma} (\{a_r^+ \tilde{a}_t^-\}^{\gamma} \{a_s^+ \tilde{a}_u^-\}^{\gamma})^0$$

The matrix  $f$  is diagonalised giving eigenvalues  $e_k^\gamma$  and eigenvectors  $u_{k\{ab\}}^\gamma$ . Defining the following vectors  $\hat{\mathbf{u}}$  in the space spanned by the index  $k$

$$\hat{\mathbf{u}}_\alpha^\gamma = \sum_{k, ab} u_{k\{ab\}}^\gamma \{a_a^+ \tilde{a}_b^-\}^\gamma$$

where  $\alpha$  represents the implicit sum over  $\{ab\}$ , the interaction can be written simply as  $H = \sum_\gamma e_k^\gamma [\gamma]^{1/2} \hat{\mathbf{u}}_\alpha^\gamma \cdot \hat{\mathbf{u}}_\beta^\gamma$  where the dot product is taken with respect to  $k$  only and not all the indices. Reference [10] describes many interesting features of these eigenvalues.

First the eigenvalues are roughly symmetrically distributed about 0.0 MeV. Most of the matrix elements are between -1.0 MeV and 1.0 MeV and they are grouped symmetrically about 0.0 MeV with what appears to be a statistical distribution. Only a finite number of eigenvalues lie outside this range and generally there is only one eigenvalue for each channel defined by  $\gamma = \{\lambda\tau\}$  and parity. The number of such eigenvalues lying outside this range appears to be independent of interaction and model space, ie the number of orbits in the calculation.

Second, after scaling for mass dependence, the eigenvalues from different interactions which fit the data in different model spaces are amazingly similar. Within a particular model space the largest eigenvalues from interactions that fit the data are almost identical as are the eigenvectors. In addition I have compared a recent normalized lo-k Kuo interaction (Kuosd) with the Wildenthal interaction (W). While the eigenvalues disagree quite a lot (Kuosd does not fit the data so well) a large number of the eigenvectors are all but identical. Even the simple MSDI interaction has some eigenvectors in common with realistic interactions. A striking feature of the MSDI in this representation are the multiple degeneracies for the eigenvalues.

Clearly there is something that this representation of the interaction is telling us that is not so obvious by looking at the two body interaction matrix elements which can look quite different for interactions that fit the data. Dufour and Zuker correctly realized that the well separated larger eigenvalues are connected to collective motion or coherent scattering while the smaller statistically distributed eigenvalues correspond to random interactions or incoherent scattering. It has been known for some time that random nuclear matrix elements reproduce many general features of nuclear spectra. What is interesting is that the two types of interaction can be roughly be separated by putting a cut on the absolute value of the eigenvalues  $e$ , except for the monopole terms that are asymmetrically distributed.

NuShellX (V4R2) now comes with a complete package to analyze, edit and apply cuts in this representation. This is done with the Control module NuDZOper which can be run either stand alone or from within the Control program NuShellX. In NuShellX choose option 'Z'. NuDZOper is the most powerful and flexible option -it is applied to the total isospin interaction and outputs either an np or isospin interaction with the same name as the input interaction but with a z appended. But there good reasons to reapply the cuts after the interaction has been transformed to the pn interaction. This can be done to the pn part of the interaction by NuDiop which is invoked if you reply yes to the Defour Zuker option in the Level section of NuShellX. This also invokes NuOpdZ and NuTRLZ which work together to speed up the calculation if you have eliminated most of the incoherent terms, but can be slower and use large amounts of memory if the cuts are less severe. The use of NuDZOper does not result in penalties or significant gains since the interaction is reconstructed back to normal form and can be used in any shell model code!

NuDZOper requires a NuShell isospin .sps file. This can be generated by using the 'M' option in NuDZOper. Do not use the same name for your isospin .sps file and your np .sps file! Next choose the 'O' option. It asks for the Model space. It then asks for the interaction you want to analyze or truncate. You have the option of removing terms in the interaction that couple spurious and non spurious states. This is trivial in this representation (I discuss this next). You can then chose a) an Edit option that presents you with the interaction in terms of eigenvalues and eigenvectors for you to edit by hand or b) the simple cut option which asks for two positive numbers equal to the absolute cut in MeV below which the eigenvectors  $e$  will be set to zero and the absolute cut in amplitude below which individual components in eigenvectors  $u$  will be set to zero. Note that the energies given in the editor are  $[\gamma]^{1/2} e_k^\gamma$ , the cuts will be applied to just the eigenvalue  $e$  where  $\gamma$  stands for  $\lambda\tau$  and the [ ] bracket has its usual meaning [10].

One simple application of this formalism is that it can be used to remove the terms in the interaction that couple spurious and non spurious states. The coupling is confined to the  $\lambda\tau = 10$  channel. The eigenvector that couples spurious and non spurious states can be simply written down in LS coupling. It is given by equation (45) of reference [11]. LS-jj recoupling coefficients enable us to write it in the forms we need. But some care is needed. The vector given by equation (45) of [11] is just the single particle matrix elements of the vector operator  $\mathbf{A}^\dagger + \mathbf{A}$ . The operator  $\mathbf{A}^\dagger$  creates one unit of excitation of the center of mass motion. The operator  $\mathbf{A}$  destroys one unit of excitation in the center of mass motion.

In equation (38) of reference [11] the center of mass coupling term in the Hamiltonian  $H_1$  is witten as  $-\hbar\omega(\mathbf{A}^\dagger \cdot \mathbf{A} + \mathbf{A} \cdot \mathbf{A}^\dagger)/2A$  where  $A$  is the total number of nucleons. This operator is separable and is (ignoring

the constants)

$$(\mathbf{A}^\dagger, \mathbf{A})^\dagger \begin{pmatrix} \mathbf{A}^\dagger \\ \mathbf{A} \end{pmatrix}$$

The single particle matrix elements  $\mathbf{A}^\dagger$  and  $\mathbf{A}$  are  $i \langle \mathbf{A}^\dagger \rangle$  and  $i \langle \mathbf{A} \rangle$  where the bra-ket's are real. So in the Defour Zuker method matrix elements of  $\langle (\mathbf{A}^\dagger, \mathbf{A})^\dagger \rangle$  and  $\langle (\mathbf{A}^\dagger, \mathbf{A}) \rangle$  can be written as two vectors  $u_{cm}^\dagger$  and  $u_{cm}$  with eigenvalues  $e_{cm}^\dagger, e_{cm}$  that include the constants. Without loss of generality, the coefficients of the vectors  $u_{cm}^\dagger$  and  $u_{cm}$  can be defined to be real with real eigenvalues of equal magnitude but opposite sign  $e_{cm}^\dagger = -e_{cm}$ . The opposite signs originate in taking the Hermitian conjugate of  $i$  which is  $-i$ . So clearly to avoid coupling with spurious states all Defour Zuker eigenvectors in the  $\lambda\tau = 10$  channel need to be orthogonal to the vectors  $u_{cm}^\dagger$  and  $u_{cm}$ . Note that the magnitudes of the amplitudes of  $u_{cm}$  are identical to  $u_{cm}^\dagger$  but the phases differ.

So to remove the coupling NuDZOper orthogonalizes all eigenvectors in the  $\lambda\tau = 10$  channel to these two vectors  $u_{cm}^\dagger$  and  $u_{cm}$  and sets  $e_{cm}^\dagger = 0 = -e_{cm}$ . It turns out that this is equivalent the alternative method of adding  $\beta H_{cm}$  to the Hamiltonian where  $\beta$  is large. In this alternative method two vectors can be identified with very large almost equal and opposite eigenvalues in the Defour Zuker analysis of the total interaction. These vectors are almost identical to those identified above. This ensures that all the other vectors are approximately orthogonal to  $u_{cm}^\dagger$  and  $u_{cm}$  and moves the spurious states to high excitation. For the simplest calculation of  $1^-$  states in  $\text{He}^4$  both methods give identical eigenvalues for the nuclear states.

## V. TIMING

I give only a few specific results for timing and discuss some empirical formula for timing. My reference system is a Dell 490 Precision work station with an Intel Xeon 5160 dual core 3.0GHz with 4MB of shared on chip cache and a 1333MHz rear side bus. The memory is 667MHz DDR-2 ecc fully buffered controlled by a quad-access controller. All my 8 memory slots are full with 4x1GB and 4x512MB modules, giving a total of 6GB. In the BIOS all speed enhancements are on. My main C: drive is a Western Digital 10,000rpm 73GB SATA-2 disk. My second disk is a Maxtor 7,200rpm 250GB SATA-1 disk. I also have a 1TB Buffalo RAID-0 system with 4 250GB pata drives. This is where I run the calculations. Its performance for large continuous file transfers is significantly higher than the other drives by up to a factor of 1.8 in both read and write modes. But it has the same performance for smaller files. I use the C: drive as the swap disk.

[My graphics card is an NVIDIA Quadro FX 4600 card with 768MB of memory. It has 128 parallel floating point processors and there is a driver suite available to enable programming preferably in C directly on the card. This comes with subset of parallelized BLAS and FFT packages. There is an interface to Fortran 95 and with Version 1 of NuShellX I successfully ran the matrix multiplications on the graphics card in parallel, but Version 1 made calls for each sub-matrix so any gains were totally wiped out by data transfer overheads to and from the card. A successful implementation would require few transfers and more intensive calculation on the card. It still handles simple screen graphics simultaneously. Bearing in mind it is only single precision and the memory is limited, while expensive as a graphics card, it is a cheap 128 multiprocessor solution and is being used for physics research (see NVIDIA website).

Finally I have parts of this system courtesy of DELL Inc. who supplied the base unit, the graphics card, 2GB of memory and the 73GB disk, to replace an older dual Xeon system I had bought with a three year warranty from their refurbished range at a very reasonable price. Problems had arisen with the old system and they have been very generous in replacing it. I am grateful for their generosity, particularly for the graphics card whose specification exceeds what they were obliged to supply.]

I will only give a few typical times for 10 eigenvalues using NuTRL. NuLnz is generally slower since it takes more iterations to converge. Using this system the time taken to do a calculation for the ground state of  $^{48}\text{Cr}$  is 36 - 42 seconds. this compares with a total time in NuShell on the same machine of around 5 minutes. For the gx1a interaction other the times for  $^{48}\text{Cr}$  are displayed below.

$0^+$	36 secs.	41355	11	1.82	2.2	0.14 secs.
$2^+$	663 secs.	182421	49	2.65	24.5	0.16 secs.
$4^+$	901secs.	246979	69	2.88	40.5	0.19 secs
$6^+$	1192 secs	226259	80	2.99	38.0	0.22 secs
$8^+$	874 secs.	156262	79	2.98	23.4	0.29 secs

There is a significant increase for the spin 2 states and then smaller increases for higher spin states. These times also reflect different numbers of iterations which were 116, 164, 116, 140 and 125 respectively. The increase for higher total angular momentum comes from two factors. First the increase in dimension which is given in the first column of the second table and the increase in the number of BAS matrices which is given in the second column of the second table. Jussi Toivanen [3] has shown that the number of mathematical operations in his method scales as  $d^{1.45}$  where  $d$  is the dimension of spin zero states and for non-zero  $\mathcal{J}$  he tells me [3] that the maximum number of operations is 9 times this limit.

I have found empirically that for  $fp$  shell nuclei with  $N=Z$  that the time per iteration scales as  $(BAS^{0.25} d)^{1.3}$ . This approximation works well from  $^{50}\text{Mn}$  onwards. It is not very good near  $^{48}\text{Cr}$ . But just as an illustration in table two column 3, I give  $BAS^{0.25}$  then in column 4,  $(BAS^{0.25} d)^{1.3}$  divided by  $10^6$ . Finally in column 5, I divide the time in seconds by the product of column 4, and the number of iterations. This gives a re-normalized time per iteration. One can see that the re-normalized times are more regular. For heavier  $N=Z$  nuclei these re-normalized times become identical for all total angular momenta. BAS does not keep increasing with higher spins. It approaches a limit defined by the maximum values of the proton and neutron individual total angular momenta. I should also point out that  $^{48}\text{Cr}$  is about a factor of two slower than calculations for heavier nuclei after re-normalization. A good normalization point is  $^{52}\text{Fe } 0^+$ , dimension 1777116, BAS 14, iterations 152, time 3821.6 seconds. This gives the time per iteration as  $8 \times 10^{-8} \times (BAS^{0.25} d)^{1.3}$  seconds.

It is interesting to compare different codes for this case Jussi Toivanen [3] has quoted an np multiplication time of 83 seconds for his code using an Opteron 2.3GHz processor. He also reports a single iteration time (published) for Antoine on a comparable processor of 670 seconds. My time is 25 seconds. Converting Jussi's np multiplication time into an iteration time by multiplying by 1.1, adjusting for the different processor speeds and assuming he quotes times for a single core and using my single core times EICODE is 60% slower than NuShellX and Antoine is 11.5 times slower than NuShellX.

For  $N \neq Z$  nuclei or protons and neutrons in different shells ie when the dimensions of A are greater than the dimensions of B times for higher total angular angular momentum states are much more favorable. I do not have any specific numbers but during testing these calculations were faster than the empirical formula, this increase in speed increased with increasing total angular momentum.

Just a couple of other time line points. One for  $^{50}\text{Cr } 0^+$  states, where NuTRL takes between 162 secs. and 180 secs. depending on interaction (it is really the number of iterations that causes the difference but the interaction determines the level density and spacing which reflects in the iteration count). Finally some numbers for  $^{56}\text{Ni } 0^+$  state that I used as a benchmark calculation in developing the code. For the full 10 eigenvectors my machine takes 17 - 19 hours. For 5 eigenvectors the times reduce to 14 - 15.5 hours.

I have so far not discussed memory requirements. Some of the preparation modules can require of the order of 1GB in excess of the system overhead. But NuTRL runs the spin zero states in  $^{56}\text{Ni}$  with an overhead of 1.3GB on my two core system. This will increase with more cores. The spin two states in  $^{56}\text{Ni}$  require an overhead of around 1.8GB with two cores. This calculation has a dimension of 71,109,189 states. But the run time is about 2 hours per iteration and a total time of 14 days. So really 16 cores are required. But the memory requirements are very reasonable and the disk requirement is limited to a file containing a maximum of 80 single precision vectors. (A file with 40 vectors is required in addition during a restart.)

## VI. COMPILING AND BUILDING

I recommend compiling with version 9.1 of the Intel compiler on Windows and Linux. For some reason I do not understand version 10 produces slower code on both Windows and Linux. I only have a license for 9.1.036 on Windows but I downloaded 10.1 on a demo license and it produced slower code. The same is true on Linux. Luckily I had kept an old copy of the personal edition for Linux version 9.0.31 and this is faster than the latest release.

### A. Wine

You can also run Windows executables under Wine version 0.9.5-11 released on 11th January 2008. Here are the instructions:

*Instructions for running NuShellX or NuShell using Linux or Unix under Wine*

If Wine is already installed you can try that first. On a Windows PC download the prebuilt code from <http://knollhouse.org> and unzip the install files. Run Setup on the PC. Using Samba or a CD transfer the unpacked '.exe', '.dll', '.sps' and '.int' files to a directory on your Linux or Unix System. You may be able to run Setup.exe under Wine. I have not tried that.

You need to set up some aliases. On SuSe Linux 10.3 I added the following to my '.bashrc' file, `alias del='rm'` and `alias ren='mv'`. Here 'del' is the command to delete a file under Windows and 'rm' a compatible Linux command. Similarly 'ren' and 'mv' are commands that will change the name of a file. Then try a simple example say '20Ne' on NuShellX. ( see NuShellX\_manual.doc ). Just type at the command prompt `wine NuShellX.exe`

If you get virtual memory errors you need to update Wine. The latest release is available free from [www.winehq.org](http://www.winehq.org). I have tested wine-0.9.5-11.1.i586.rpm for Open SuSe 10 and Fedora core 8. Other versions are available for Red Hat, CentOS, other Fedora releases, Mandriva, Slackware, Ubuntu, FreeBSD, PC-BSD and Solaris. If Wine is already installed on your system you need to ask your system administrator to uninstall it and then install the new version. Otherwise you just need to ask him/her to install the latest release.

The shell model codes run about 10% slower under Wine compared to running them in a native Windows environment on the same machine.

## B. Building and Compiling under Windows

Each NuShell module (not to be confused with a Fortran 90 module) has its own directory. There are also common directories NuCommon and NuCommonX. If you have Microsoft Visual Studio and have loaded the software provided by Intel to convert their compiler into Intel Visual Fortran the necessary .sln and .vfproj files are included to you will be up and running in no time. Otherwise I provide a Make.f90 code that will generate make files but it has not been upgraded for a while and needs some work. Finally there is an ..Order.build file which lists the files to be compiled and linked, in the order they should be compiled and linked , giving their relative location and lists any special switches for this module.

The switches common to all modules are `/O3 /Ob2 /reentrancy /assume:buffered_io /Qipo`. Assuming you have a modern multicore processor you should add `/QaxT`. For other processors change T to the letter appropriate for your processor.

Some modules require `/Qopenmp`. Only include this when required. If you do not want to use OpenMP use `/Qopenmp-stubs` instead. Some programs require the linker switch `/STACK:100000000`. Some programs require `/LARGEADDRESSAWARE` to allow access to memory above the 2GB limit. On Windows 32 bit you also have to add the `/3GB` switch to your boot.ini file. Read up on this at Microsoft's Web site to ensure you do it correctly. My line reads `multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="Microsoft Windows XP Professional" /3GB /noexecute=optin /fastdetect`

If you are compiling for 32 bit Windows, create and build as described in the Intel Documentation. No code changes are required. If you are compiling and building for 64 bit Windows you need to change two parameters. Edit NuCommon.f90 in the /NuCommon folder. Very near the top you will find `nbits=32` change this to `nbits=64` and later change `nwords=10` to `nwords=5`. Save the file and build.

## C. Building under Linux.

I have built and run the code under SuSe 10.3 and Fedora Core 8. Much of the above discussion for Windows also applies to Linux so I only note differences here. The equivalent compiler switches are `-O3 -Ob2 -reentrancy -assume buffered_io -xT -ipo -openmp` (where required). There is no stack switch in Linux , you have to use the system command `'ulimit -s 100000000'` once before you run any of the codes. There is no large address switch but you should consider which memory model you use. Since you will probably run in 64 bit mode you must make the changes to NuCommon.f90. You must also name the executables NuBasis NuProj etc with capitalization and no extension.

There are three other editing jobs to be done. 1) Edit ICommon.FI in folder NuCommon. Find the subroutine 'get\_next\_file'. Comment out the code between `! Intel . . . ! end Intel`. Uncomment the code after `! Sun` to before `! end Sun`. Replace the function call `sh` to `SYSTEM` instead. 2) Next edit NuShellx.f90 in the NuShellX folder. In the subroutine Levels find the function call `delet(...)`. Replace by a call to `SYSTEM('rm *.lev')`. 3) Edit TRl.f90 in the NuTRL folder. Find the SYSTEM calls. Change `del` to `rm` and `ren` to `mv`. NuShellX is now converted to Linux and ready to build and run. (This applies to Version 4. In version 3 edit NuTRL.f90 instead.)

### Acknowledgments

Many thanks to Jussi Toivanen for useful email exchanges and particular thanks to Mihai Horoi who checked NuShellX against his own code and Antoine, for nuclei I could not reach with NuShell. This showed up a bug which had crept in during a code revision. It has been fixed and NuShellX now agrees extremely well with other codes including NuShell and Antoine. Many thanks to Alex Brown for continuing to test the code and in particular finding an error in  $1p_{1/2}$  and  $xp_{1/2}$  calculations and labelling errors in NuTra. At his request the code has been speeded up for  $0p$  or  $0n$  calculations.

### References

- 
- [1] N. Godwin, D.Phil. Thesis, Oxford, (1979), W.D.M.Rae, D.Phil. Thesis, Oxford, (1976), unpublished
  - [2] E. Caurier and F. Nowacki, Acta Physica Polonica B Vol. **30** (1999), 705
  - [3] J. Toivanen, arXiv:nucl-th/0610028v1 9 Oct 2006, and J. Toivanen and W. D. Rae to be published and private communications.
  - [4] J. Toivanen, computer code EICODE, JYFL, Finland, 2004
  - [5] Intel Corporation, USA
  - [6] OXBASH for Windows, B. A. Brown, et al. MSU-NSCL report number 1289, (2004).
  - [7] R.R Whitehead et al, Adv. Nucl. Phys. **9**, (1977), 123
  - [8] NuShell for Windows, Solaris and Linux, W.D.M. Rae, <http://knollhouse.org> , unpublished, (2007)
  - [9] See Anyas-Weiss et al, PHYSICS REPORTS (Section C of Physics Letters) **12**, no.3 (1974) 201-272 for a formula for  $G(SU(3))$
  - [10] M. Defour and A. P. Zuker, Physical Review C, **54**, (1996), 1641
  - [11] S. Gartenhaus and C. Schwartz, Physical Review **108**, (1957), 482