FINCH: Fitting Isospin Non-Conserving Hamiltonians

Aaron Magilligan

Updated: March 9, 2022

Over the course of the past four years, I have developed from the ground up a Python application to constrain configuration-interaction Hamiltonians for use with NuShellX and similar programs. This code requires an installation of NuShellX@MSU and Python 3 to run properly. The code is called Fitting Isospin Non-Conserving Hamiltonians, or FINCH.

FINCH takes in an initial interaction to ensure the first iteration's nuclear wavefunctions are realistic, and an *ab initio* interaction with which those poorly determined parameter linear combinations will be replaced. FINCH also needs a data set for a given model space. The Singular-Value Decomposition Fitting method is implemented. It comes preconfigured to slowly evolve the interaction and generate a series of solutions for each number of allowed varied linear combinations.

1 How it Works

The initial interaction is broken into several groups of parameters by whichever method the user chooses.

FINCH then calculates the wavefunctions and spectra for each nucleus included in the data set, along with the contribution from each group of parameters and collects this data.

The experimental data and group contributions are then used to generate an error matrix and error vector. The error matrix can then be decomposed into a diagonal matrix containing the **singular values** of the error matrix, along with a rotation matrix that defines the natural basis of the parameter space.

At this stage, we can replace the poorly determined linear combinations with those from the *ab initio* interaction. A choice must be made here as to how many linear combinations to replace. A new interaction can then be determined, and this process is iterated until convergence.

By default, the program will start with a small number of varied linear combinations (VLC) and increase until a full chi-squared fit is completed, and then begin "back propagating" by decreasing the number of varied linear combinations until we simply recreate the *ab initio* interaction. This slow evolution prevents us from wandering too far in the parameter space.

If you run into any error you don't see an immediate cause for, try rerunning in a fresh folder. This fixes 9/10 problems with NuShellX manipulations done by the code.

2 FINCH Answer File

The code is designed to need very little alteration for a new fit with a new model space, a new set of parameter groups, or for the introduction of new data. To this end, the necessary declarations one would want to change in the code are gathered in a single file format *input.fans*. This file is copied over into *fit-inputs.py* and imported as a module in FINCH. The variables in the fans file are

- MODEL_SPACE: the name of the model space
- ZERO_BODY_TERM: the energy contribution from the core in the shell model
- A_MIN, A_MAX: the range of mass values in the model space
- PROTON_ORBITS, NEUTRON_ORBITS: the number of the orbits in order for the .mod file, using the NushellX k labeling
- DATA_FILE_NAME: Name of input file containing level data (*.levels)
- AI and INITIAL interactions: Can be the same or can choose more realistic initial interaction for the first iteration. Defined in parts by,
  - _SPE: name of input file containing single-particle energies and orbit references (*.spe)
  - _STRONG: name of input file containing the TBME for the strong interaction (*.tbme)
  - _COUL: name of input file containing the TBME for the Coulomb interaction (if left as ” then Coulomb is not included)
  - _ISOV: input value for isovector strength (0.01 is 1%) (if left as ” then Coulomb is not included)
  - _ISOT: input value for isotensor strength (0.01 is 1%) (if left as ” then Coulomb is not included)
- DATA_FILE: the file containing the levels/nuclei in the fit. Columns show: element symbol, A, Tz, T, 2J, P, jnum, Energy, and error followed by any relevant comments.
- NEW_RUN: True or False, allows you to rerun a fit in the same folder (by default the last folder ran)
- SKIP_RUNS: how many iterations in the fit you wish to skip over as the calculations had been done before
- PAUSE_BETWEEN: T or F, allows you to have the program pause between iterations to allow debugging/testing.
- fit_to_energies: T or F, include energies in .levels file in the fit
- fit_to_MED: T or F, include mirror energy differences for pairs in .levels file in the fit
- fit_to_TED: T or F, include c coefficients for triplets found in .levels file in the fit
- fit_to_delMED: T or F, (meant for *sd*-shell only) include energies in .levels file in the fit

- STBME_CHOICE: A setting for choosing the grouping method use on the strong interaction TBME
  - 1 - all TBME in one group with one overall strength parameter
  - 2 - all TBME in two groups based on isospin value (T=0,1)
  - 3 - all TBME in three groups evolving separately: pp, nn, and pn
  - 4 - TBME in isospin formalism groups so that pp,nn, and pn evolve together
  - 5 - TBME in isospin formalism groups, but only the diagonal are fit
  - 6 - All TBME in their own groups, each allowed to fit on their own
  - 7 - Specific to the calcium fit in the fp shell UFP-CA
  - 8 - Fit diagonal matrix elements except those involving excluded orbits
  - 9 - Takes group numbers from '*custom.dat*' file (see sample input section)
- EXCLUDE_ORBIT_LIST: a list of integers corresponding to the absolute orbit reference for excluding TBME involving these orbits to fit. Used in conjunction with STBME_CHOICE=8
- DO_NOT_FIT: a list of integers corresponding to the group numbers for the parameters you wish to hold constant in the fit. See "fit_labels.dat"
- max_groups: by default the program will increase VLC until a full fit is reached. However, if the data set is lacking or you wish to save time, you can set a maximum number of VLC the program will iterate up to before "back propagating"
- CALCULATE_ALL: T or F, calculate the overlaps for excluded groups. Set to False for faster iterations, but lose information you may want later.
- VLC_CHOICES: Default is an empty list for a full fit. Or include integers in list to converge only at those VLC.
- FORWARD_ITER_MAX: The maximum number of iterations to spend on any VLC in the forward phase before moving on. The fit may be oscillating between two minima and never actually converge otherwise.
- FORWARD_ITER_MAX: The maximum number of iterations to spend on any VLC in the backward phase before moving on. The fit may be oscillating between two minima and never actually converge otherwise.
- MAX_GROUPS_CHOICE: The maximum number of VLC the fit will try to fit. If it is known that a full SVD will wail, set a reduced number here. A default of 0 sets this to the total number of groups in the fit.
- STORE_EVERY_VLC: T or F, store a copy of each possible VLC choice at each iteration. In cases with a large number of groups, this may cause significant delay between iterations.

3 Sample Inputs

Any new fit requires a handful of inputs. A list of levels to be included in the fit, the single-particle energies, and the two-body matrix elements, and an FINCH answer file to define the Hamiltonian.

3.1 Level Data (*.levels)

The level data includes the ground states and excited states in ascending mass order. It is important that the excited states of a nucleus be listed directly after the ground state.

The columns represent: The chemical symbol, the mass number A, $2T_z$, $2T$, $2J$, P, the J numbered ordering of the level, the experimental energy (important to note that the code assumes a negative ground state energy), and the associated error of the experimental energy. Comments can be added at the end of each line.

The experimental error is also used to include/exclude data from the fitting procedure. In some cases, a user might want to track a level during the fit but there is no well known value for that level, or you know the level will be wrong due to a breakdown of the model space. For this case use an error that is greater than 4 MeV. The output energy rms will include only those levels with less than a 0.3 MeV error, but this can be changed in the code around line 1000.

The below is a sample input for a *fpj4* model space fit to the Calcium isotopes.

```
 Ca 48    8  8    0   0   1   -416.0009    4.0001
 Ca 49    9  9    3   1   1   -421.1474    0.0002
 Ca 49    9  9    1   1   1      2.0232    0.0003
 Ca 49    9  9    5   1   1      3.9910    0.0002
 Ca 50   10 10    0   0   1   -427.5082    0.0016
 Ca 50   10 10    4   0   1      1.0267    0.0001
 Ca 51   11 11    3   1   1   -432.3226    0.0005
 Ca 51   11 11    1   1   1      1.7180    0.0010
 Ca 51   11 11    5   1   1      2.3781    0.0002
 Ca 51   11 11    3   1   2      2.9341    0.0010
 Ca 51   11 11    7   1   1      3.4621    0.0002
 Ca 51   11 11    5   1   2      3.4775    0.0023
 Ca 51   11 11    9   1   1      4.3201    0.0004
 Ca 52   12 12    0   0   1   -438.3278    0.0007
 Ca 52   12 12    4   0   1      2.5630    0.0010
 Ca 53   13 13    1   1   1   -441.5218    0.0424
 Ca 53   13 13    5   1   1      1.7530    0.0150
 Ca 53   13 13    3   1   1      2.2000    0.1000
 Ca 54   14 14    0   0   1   -445.3650    0.0486
 Ca 54   14 14    4   0   1      2.0430    0.0190
 Ca 55   15 15    5   1   1   -446.9255    0.1600
 Ca 56   16 16    0   0   1   -449.8568    0.2500
 Ca 56   16 16    4   0   1      1.4560    0.0120
 Ca 57   17 17    5   1   1   -451.7881    0.9900
 Ca 58   18 18    0   0   1   -454.4447    0.0000
 Ca 58   18 18    4   0   1      1.1150    0.0340
```

```
Ca 59   19 19     5   1   1   -455.8211    0.0000
Ca 59   19 19     9   0   1      1.3700    0.2501
Ca 60   20 20     0   0   1   -458.7272    0.0001
Ca 60   20 20     0   0   2      1.5500    0.2501
```

3.2 Single-Particle Energies *.spe

For a fit in the *fp*-shell with no Coulomb interaction, we set up the following file. fp.spe has columns indicating the SPE type (Coulomb 'c', or strong 's', not important unless adding mass dependence), the orbit label, and the SPE value.

```
s       5    -1.5615
s       6    -5.1465
s       7    -3.1233
s       8     2.6100
```

The lack of values for orbits $k = 1 - 4$ tells FINCH that it is only fitting neutron orbits.

3.3 Two-Body Matrix Elements *.tbme

This format is used for both the Coulomb and strong interaction TBME.

A two-body matrix element input is shown below with columns: k_1, k_2, k_3, k_4, J, T, and v with the k-orbits being labeled as in NuShellX .int files.

```
5  5  5  5   0  1  -1.1858
5  5  5  5   2  1  -0.0139
5  5  5  5   4  1   0.3571
5  5  6  5   2  1  -0.2710
5  5  6  5   4  1  -0.6748
5  5  8  8   4  1   0.0652
```

This is also the format of the '*custom.dat*' file for use with STBME_CHOICE=9 for specifying the strong TBME groups. The difference is instead of values in the seventh column, there is a group name. A group name of '0' will result in the TBME not being fit, and then all other unique group names form their own parameter group. For example, we can define three groups for the above TBME as,

```
5  5  5  5   0  1  1
5  5  5  5   2  1  2
5  5  5  5   4  1  3
5  5  6  5   2  1  0
5  5  6  5   4  1  0
5  5  8  8   4  1  0
```

which places each of the first three matrix elements in their own group and keeps the rest unconstrained.

A copy of '*custom.dat*' with all zeros is generated and placed in the 'groups/' folder each time FINCH is run. To use this, simply place it in the main working directory.

4 Sample Outputs

4.1 Extracted overlap files *.xfit

A csv file containing the calculated energy of a level and then it's overlaps corresponding to the fitting groups in order, example for "Ca59-5-1-1.xfit" (Nucleus-2J-P-jnum.xfit).

```
-445.6423,-7.3953,-20.1276,-6.139,1.0112,-0.7308,-0.0398,1.8355,...
```

4.2 Output Energies

A list of levels showing the input experimental energy and error, along with the interactions prediction for the level. NOTE: the output-energy.dat file in an iteration folder corresponds to the hamil.int files created in the previous iteration.

```
Ca48-0-0-1               -416.0009    4.0001 -416.0009    0.0000
Ca49-3-1-1               -421.1474    0.0002 -421.1474    0.0000
Ca49-1-1-1                  2.0232    0.0003    2.0232    0.0000
Ca49-5-1-1                  3.9910    0.0002    3.5850   -0.4060
Ca50-0-0-1               -427.5082    0.0016 -427.9231   -0.4149
Ca50-4-0-1                  1.0267    0.0001    1.7820    0.7553
Ca51-3-1-1               -432.3226    0.0005 -431.4804    0.8422
Ca51-1-1-1                  1.7180    0.0010    0.6319   -1.0861
Ca51-5-1-1                  2.3781    0.0002    1.7708   -0.6073
Ca51-3-1-2                  2.9341    0.0010    2.4860   -0.4481
Ca51-7-1-1                  3.4621    0.0002    3.8833    0.4212
Ca51-5-1-2                  3.4775    0.0023    2.1706   -1.3069
Ca51-9-1-1                  4.3201    0.0004    3.5787   -0.7414
Ca52-0-0-1               -438.3278    0.0007 -436.7397    1.5881
Ca52-4-0-1                  2.5630    0.0010    1.8999   -0.6631
Ca53-1-1-1               -441.5218    0.0424 -439.3412    2.1806
Ca53-5-1-1                  1.7530    0.0150    1.2991   -0.4539
Ca53-3-1-1                  2.2000    0.1000    1.5365   -0.6635
Ca54-0-0-1               -445.3650    0.0486 -442.6918    2.6732
Ca54-4-0-1                  2.0430    0.0190    1.9621   -0.0809
Ca55-5-1-1               -446.9255    0.1600 -443.4941    3.4314
Ca56-0-0-1               -449.8568    0.2500 -445.7714    4.0854
Ca56-4-0-1                  1.4560    0.0120    1.3203   -0.1357
Ca57-5-1-1               -451.7881    0.9900 -445.5049    6.2832
Ca58-0-0-1               -454.4447    0.0000 -446.9030    7.5417
Ca58-4-0-1                  1.1150    0.0340    1.5160    0.4010
Ca59-5-1-1               -455.8211    0.0000 -445.6423   10.1788
Ca59-9-0-1                  1.3700    0.2501    1.3755    0.0055
Ca60-0-0-1               -458.7272    0.0001 -446.2499   12.4773
Ca60-0-0-2                  1.5500    0.2501    2.9802    1.4302

rms deviation:       3.6262
28
```

The final row corresponds to the number of well known levels included in the rms deviation calculation.

4.3 Output of Fit Scaling

At each iteration, a new interaction is made for every number of varied linear combinations up to the total number of groups. The information for these are stored in ouput-mults.dat in the form of multiplicative scaling factors that need to be applied to the current Hamiltonian to recover the full interaction. This file is used mainly as a guide in determining which parameters are important at different varied linear combination numbers.

Each row represents a single interaction for a varied linear combination number starting at 0 VLC or the *ab initio* interaction. The second half of the row represent the statistical uncertainty introduced by the fit to each parameter group.

5 List of Functions

There are many small functions that are clear in their usage by their name and are short enough to understand by simply reading them. The larger and more complex functions are listed here and their usage is described (this list is still being expanded on).

5.1 General Functions

These functions are very straightforward, and serve to clean up the code by not having to repeat simple (but lengthy) commands. **swap_element_label(z, fixlength=False)**

This function will exchange a proton number Z for the corresponding chemical element symbol, and vice-versa. The option *fixlength* can require that all symbols are returned as two characters in length by appending an underscore to one-letter symbols.

cycle_lbl(x)

NuShellX uses a single character to express the number of protons or neutrons a nucleus has in the model space. This requires using letters for those numbers greater than 9, which is accomplished with this function.

os_path(path)

A simple command to ensure that text strings meant to be interpreted as paths in the file structure are interpreted that way by Finch and NuShellX.

make_folder(path)

Check is a folder exits at *path* and if not, create one.

rms_from_lists(x, y)

Take the rms deviation between any two generic lists.

write_csv_file(fn, list)

Store *list* as a csv formatted file with name/path *fn*.

5.2 The Hamiltonian Class and its Companions

The Hamiltonian class is an object that stores the necessary information to describe an isospin-nonconserving interaction. It contains a number of methods and has associated functions that allow one to compare, change, and write interaction files that can be read by either Finch of NuShellX as needed. To initiate an instance of this class a Finch answer file (*.fans) is needed. To see the form of this input, see Section 5 on Sample Inputs.

__init__(self, ans_file, group_folder='groups')

The initialization of the class, which reads in the data in the answer file (ans_file). In here the mass dependence can be set for the interaction, and the two-body terms are typed and grouped according to conditions set in **type_tbme** and **group_tbme**. The groups are then stored by calling **make_files**. Finally a group number is applied to the two-body terms using **group_number** to aid in manipulation of the parameters in other functions.

strength_list(self)

Returns a python list showing the representative strength of each fitting group in order, to allow for normalization of the error matrix and vector found in **perform_fit**. If there is more than one parameter in a group, the first parameter is taken as the representative strength, otherwise it is just the strength of the single parameter. This can be changed however you like.

comparison_to_bg(self, bg_ham, name)

To better aid in comparisons between the fitted interactions and the original interaction *bg_ham*, this function creates an *.int file showing the original SPE and TBME in comments. The file should still work in NuShellX with these added comments.

make_files(self, a_min, a_max, group_folder)

Takes the total Hamiltonian and creates the necessary *.int files for calculation. These include a hamil.int file that contains the full interaction at a specific mass A for masses between *a_min* and *a_max*, and group*.int files that define the parameter groups at each mass. Additionally, a file is created called parts.nux that lists the group*.int files so that NuShellX can calculate their overlaps with the full wavefunctions.

full_tbme(self)

If the Hamiltonian includes more than one source of two-body terms (such as Coulomb, isotensor, or isovector), than this method will combine them into a single interaction to be written into a file for use in NuShellX.

full_spe(self)

If the Hamiltonian includes more than one source of single-particle energies (such as Coulomb, isotensor, or isovector), than this method will combine them into a single interaction to be written into a file for use in NuShellX.

normalize_scale(self, a)

Universal interactions can contain a mass dependence in the two-body (and in principle the one-body) terms, and this method will normalize the interaction to a specific mass A.

int_at_mass(self, a, folder)

The interaction is collected and normalized to a mass A, and stored as an *.int file in **folder**.

store(self, folder)

A function to store the interaction in *.fans format and in the NuShellX *.int format.

make_isotensor(self, alpha)

A method to create the isotensor interaction in a given strong interaction for a given isotensor strength *alpha*. The isotensor strength is modeled as the scaled increase of the pn $T = 1$ two-body terms over the average of the nn and pp terms.

make_isovector(self, alpha)

A method to create the isovector interaction in a given strong interaction for a given isovector strength *alpha*. The isovector strength is modeled as a scaled increase/decrease of the nn/pp terms so that their average remains the same to avoid interference with **make_isotensor**.

int_file(self, spe, tbme, name, replace_k=True)

The generic interaction file creation method, with options to allow other functions to effectively utilize it in specific cases.

write_groups(self, folder)

This function performs the work described in **make_files** for a given interaction normalized to some mass. It also generates the 'fit_labels.dat' file which contains a description of what is contained in each two-body interaction group.

group_number(row)

Determines the numerical label for a given group name using 'fit_labels.dat' and returns it.

group_tbme(row)

Using the rules defined in this function, a two-body matrix element is sorted into a group and a label corresponding to that group is returned.

type_tbme(row)

Determines the type of two-body matrix element (pp, pn, nn).

read_tbme(name)

Reads in a set of two-body matrix elements in a file located at *name*.

read_spe(name)

Reads in a set of single-particle energies in a file located at *name*.

update_hamiltonian(old_ham, bg_ham, mults)

A new Hamiltonian class instance is initialized, and using the scaling factors (*mults*) generated in **perform_fit** are used to transform the *old_ham* into a new interaction. The background interaction is also sent here so that the rms deviation between the new interaction and the *ab initio* can be calculated.

order_tbme(df)

This function will reorder the two-body dataframe into the standardized ordering used by NuShellX. This aids in readability and comparisons between interactions.

5.3 Running NuShellX and Extracting Results

This section of code is the main gateway to manipulating NuShellX and collecting and formatting its results. The main time save is found here in the form of python multiprocessing, which allows for many nuclei to be calculated at once on a multi-core machine.

setup_dataframe(file, ham)

The nuclear levels included in the data set are read in here and stored in a pandas dataframe. This dataframe is then used to determine the needed calculations.

generate_ans(fit_loc, a, z, j, p, jnum, min_error)

The NuShellX answer file is generated here at *fit_loc* for a nucleus with the properties *a, z, j, p*. The minimum error parameter refers to the minimum experimental error for a level in the nucleus you are trying to calculate. To save time, nuclei with high errors don't need to be fully calculated. See comments in code for further detail.

This function also copies in the interaction to be used for calculation and the parts.nux file.

single_calc(x)

Called from **run_multiproc**, this runs in parallel to calculate all nuclei in the data set and retrieve the results. The parameter *x* is unwrapped into *df_nuc*, *group_count*, and *run_name* which are the subset of the dataframe relevant to a single nucleus at a single spin, the number of groups/parameters in the fit, and the folder in which to store the results.

A path is made for the nucleus to be calculated in, performs those calculations, and extracts the list of overlaps (contributions) to the state energies corresponding to the groups/parameters you are fitting. These are saved for use in the fitting procedure.

run_multiproc(df, r_name, group_count)

Takes in the pandas dataframe containing the nuclear level data, splits it into chunks and initiates the calculations. Once all calculations are complete the code will continue.

read_lpt(location, lineskips=6, corner=False)

The main output file in NuShellX for energy levels has the extension *.lpt. This function reads in these lpt files (check line skips length if this fails, different interactions require different numbers of lines to be skipped), and stores them as a dataframe.

5.4 Performing the Fit

Here the data collected by run_multiproc is used to set up the data error matrix and vector, perform the SVD, and update the Hamiltonian to allow a new iteration to begin. Along with binding and excitation energy inputs as data, this function has the (off by default) ability to also include the mirror energy differences, *c*-coefficients of the IMME, and double energy differences of mirror pairs. This is useful to constrain the isospin non-conserving interactions.

avg_group_ratios(a, b)

Find the average ratio between parameters in the defined groups for two interactions *a* and *b*. This is being phased out in favor of normalizing the error_matrix and error_vectors in **perform_fit** accomplished using the class method **strength_list.**

setup_b_list(df)

Takes the levels dataframe and determines if any mirror pairs are included, and prepares the program to fit on those mirror energy differences.

setup_c_list(df)

Similar to the above, but checks for complete $T = 1$ triplets to allow for fitting on the c-coefficients.

setup_b_diff_list(df)

Similar to the above, but checks for specific differences between mirror pairs, meant for fitting the isovector strength in the *sd* shell data.

subset_groups(group_list, hold, no_energy=False)

As the code allows for groups to be held constant, we need a way to extract a subset of a list that corresponds to those groups that are actually fit.

superset_groups(group_list, hold, dimension, no_energy=False)

As the code allows for groups to be held constant, we need a way to build up a superset of a list that corresponds to those groups that are actually fit.

perform_fit(level_data, folder, cur_ham, bg_ham, vlc, dim, held_groups)

This is the meat of the program, where the fit is actually performed on the data given the NuShellX results. Check back for further documentation.

setup_fit(df, store_output, iterate_fit, multipliers, fit_type)

To allow for multiple data types to be included in a single fit, this sub function was created. It generalizes the generation of the error matrix and error vector.

read_xfit(loc, nuc, k)

Creates a python list from the extracted overlaps for the fitting groups collected by **single_calc**.

b_ovl_list(nuc_1, nuc_2, num)

Creates an overlap list for a mirror pair found in the data.

tbme_plot(r_name, bg_ham, curr_ham, vlc_num)

A very bare-bones plotting function to give quick glances at the changes in the two-body matrix elements during the fitting procedure.