Towards a new Projection Code

Martin Ueding (ueding@hiskp.uni-bonn.de)

2019-01-12

Abstract

The existing projection code only works for the ρ resonance. We want to extend it to multiple particles and arbitrary isospin. This note contains the architecture and a survey of possible programming languages and libraries for the implementation. Everything that is going to be implemented will be written out here.

Contents

1 Specification

The sLapH contraction code can only work with single particle operators multiplied together. These operators are just Dirac-Γ structure and integer three-

momentum p. The spin and isospin projection code needs to take these and give a GEVP for given total momentum P^2 and irrep Γ.

We have to decide whether and when we want to include baryons. As discussed with Carsten we will stick with the single cover of the octahedral group and therefore just mesons for the time being.

The result that we want from the projection is a data frame with these columns:

- Total momentum P^2
- Irrep Γ
- Irrep row α
- GEVP row index (source operator ID)
- GEVP column index (sink operator ID)
- HDF5 dataset name
- Complex weighting factor

With this we can take the output of the sLapH contraction code and project into the irreps.

In order to get this data frame, we will do the first four of these steps:

1. Group theory
2. Isospin projection
3. Spin projection
4. Analytic Wick contractions
5. Actual projection of the correlators from the HDF5 files.

2 Architecture

Isospin and spin projection should be independent of each other. The current projection code does both concepts at the same time and exploits certain simplifications that come with the ρ-channel. In this new code we want to refrain from any such premature simplifications to use it for any physical process to come.

2.1 Group theory

We will need to have the following of the octahedral group and each of the relevant little groups:

- Irreducible representation matrices D^{Γ}
- Irreducible continuum matrices D^{J}, the Wigner-D matrices
- Cartesian 3D representation matrices R_g

There must be correspondence between the various representations, otherwise we could not sensibly multiply different representations together.

It still needs to be decided whether we want to have all the little groups corresponding to every momentum or whether we do rotations to a reference momentum ourselves.

2.2 Isospin

The isospin projection can be done by hand for the cases we have done so far. For the $I = 0$, $I_3 = 0$ case for instance we construct a two-pion operator as

$$\pi\pi(\vec{p}_1, \vec{p}_2) = \pi^+(\vec{p}_1)\ \pi^-(\vec{p}_2) + \pi^-(\vec{p}_1)\ \pi^+(\vec{p}_2) - \frac{1}{\sqrt{3}}\pi^0(\vec{p}_1)\ \pi^0(\vec{p}_2)\,.$$

For $I = 1$, $I_3 = 0$ the operator is

$$\pi\pi(\vec{p}_1, \vec{p}_2) = \frac{1}{\sqrt{2}}\left(\pi^+(\vec{p}_1)\ \pi^-(\vec{p}_2) - \pi^-(\vec{p}_1)\ \pi^+(\vec{p}_2)\right)\,.$$

For $I = 3$, $I_3 = 3$ the operator is just

$$\pi\pi\pi(\vec{p}_1, \vec{p}_2, \vec{p}_3) = \pi^+(\vec{p}_1)\ \pi^+(\vec{p}_2)\ \pi^+(\vec{p}_2)\,.$$

For scalar particles this is very easy. For vector mesons like the ρ we need to build the various spin states J_3.

As pointed out by Marcus this can also be done automatically. As I plan this to be independent of the remainder, it does not matter so much whether and when we do this automatically.

2.3 Spin

For the spin part we consult Markus' "projection" chapter and use Equation (3.34). For the generalization to N_p particles we have the following expression:

$$\begin{aligned} O^\alpha_\Gamma(\vec{p}_\mathrm{cm})^\dagger = &\sum_{M=-J}^{J} \phi_M \left[\prod_{i=1}^{N_\mathrm{p}} \sum_{M_i=-J}^{J}\right] \langle J, M | J_1, M_1, \ldots, J_{N_\mathrm{P}}, M_{N_\mathrm{P}}\rangle \\ &\times \sum_{\beta=1}^{\mathrm{nrow}(\Gamma)} \tilde{\phi}_\beta \sum_{g\in \mathrm{LG}(\vec{p}_\mathrm{cm})} D^\Gamma_{\alpha\beta}(g)^* \\ &\times \prod_{i=1}^{N_\mathrm{p}} \sum_{M_i'=-J_i}^{J_i} \sum_{M_i''=-J_i}^{J_i} D^{J_i}_{M_i' M_i}(g)\ D^{J_i}_{M_i'' M_i'}(\tilde{g})\ O^{J_i}_{i M_i''}(R^{-1}_{\tilde{g}} R_g R_{\tilde{g}} \vec{p}_i)^\dagger\,. \end{aligned}$$

The variables J and J_i are determined by the physical process that one investigates.

In order to express this in terms of already executed operator products we need to rewrite the last line as

$$\begin{aligned}
&\prod_{i=1}^{N_\mathrm{p}} \sum_{M_i'=-J_i}^{J_i} \sum_{M_i''=-J_i}^{J_i} D^{J_i}_{M_i' M_i}(g)\; D^{J_i}_{M_i'' M_i'}(\tilde{g})\; O^{J_i}_{i M_i''}(R_{\tilde{g}}^{-1} R_g R_{\tilde{g}} \vec{p}_i)^\dagger \\
&\qquad = \left(\prod_{i=1}^{N_\mathrm{p}} \sum_{\mu_i=-J_i}^{J_i} \right) \left[\prod_{i=1}^{N_\mathrm{p}} \sum_{M_i'=-J_i}^{J_i} \sum_{M_i''=-J_i}^{J_i} D^{J_i}_{M_i' M_i}(g)\; D^{J_i}_{M_i'' M_i'}(\tilde{g})\; \delta_{M_i'' \mu_i} \right] \\
&\qquad\quad \times O^{J_1 \ldots J_{N_\mathrm{p}}}_{\mu_1 \ldots \mu_{N_\mathrm{p}}}(R_{\tilde{g}}^{-1} R_g R_{\tilde{g}} (\vec{p}_1, \ldots, \vec{p}_{N_\mathrm{p}}))^\dagger \,.
\end{aligned}$$

The round parentheses are meant to just limit the scope of the $\prod$-sign whereas the square brackets are meant to limit the scope of both $\prod$- and $\sum$-signs. The group element $\tilde{g}$ is defined as $\vec{p}_\mathrm{cm} = R_{\tilde{g}} \vec{p}_\mathrm{ref}$. This likely is not unique because we can always add a rotation around the resulting $\vec{p}_\mathrm{cm}$ without changing it. Perhaps this does not matter? Also reflections might need to be excluded.

2.4 Analytic Wick contraction

The analytic Wick contraction can be done with the "Quark Contraction Tool" in Mathematica. For the $I = 3$, $I_3 = 3$ channel for zero momentum with source at `x` and sink at `y` we have summands like the following, which we identify as `C6cD`:

```
trace[Gamma^5.DE[{up, up}, {x, y}].Gamma^5.DE[{dn, dn}, {y, x}]]^3
```

Also we have the `C6cCD`:

```
trace[Gamma^5.DE[{up, up}, {x, y}].Gamma^5.DE[{dn, dn}, {y, x}]]
trace[Gamma^5.DE[{dn, dn}, {y, x}].Gamma^5.DE[{up, up}, {x, y}].
      Gamma^5.DE[{dn, dn}, {y, x}].Gamma^5.DE[{up, up}, {x, y}]]
```

And also the `C6cC`:

```
trace[Gamma^5.DE[{dn, dn}, {y, x}].Gamma^5.DE[{up, up}, {x, y}].
      Gamma^5.DE[{dn, dn}, {y, x}].Gamma^5.DE[{up, up}, {x, y}].
      Gamma^5.DE[{dn, dn}, {y, x}].Gamma^5.DE[{up, up}, {x, y}]]
```

With the sLapH method at zero momentum we have it quite easy as the three sources and three sinks are indistinguishable. But with general momenta we cannot simplify it as much. The expression from the contraction code with sources `x1`, `x2`, `x3` and sinks `x4`, `x5`, `x6` contains `C6cD` diagrams like this one:

```
trace[Gamma^5.DE[{up, up}, {x1, x4}].Gamma^5.DE[{dn, dn}, {x4, x1}]]
trace[Gamma^5.DE[{up, up}, {x2, x6}].Gamma^5.DE[{dn, dn}, {x6, x2}]]
trace[Gamma^5.DE[{up, up}, {x3, x5}].Gamma^5.DE[{dn, dn}, {x5, x3}]]
```

But also we have this one here:

```
trace[Gamma^5.DE[{up, up}, {x1, x4}].Gamma^5.DE[{dn, dn}, {x4, x1}]]
trace[Gamma^5.DE[{up, up}, {x2, x5}].Gamma^5.DE[{dn, dn}, {x5, x2}]]
trace[Gamma^5.DE[{up, up}, {x3, x6}].Gamma^5.DE[{dn, dn}, {x6, x3}]]
```

The difference is just that the exchange $x_5 \leftrightarrow x_6$ was done.

The output from the Quark Contraction Tool is basically a prescription to a linear combination of diagrams from the contraction code. The positions are equivalent to momentum labels. From this output we learn the way that we have to combine the different diagrams to yield a particular isospin.

We have to convert this output into a data frame with the following columns:

- GEVP row index (source operator ID)
- GEVP column index (sink operator ID)
- HDF5 data set name like `C2c_uu_p010.d000.g5_p010.d000.g5`
- Boolean flag whether the correlator needs to be conjugated to compensate for reversal of quark flow direction
- Complex weight factor

Ideally we also apply certain symmetry simplifications where we can apply say the permutation (13)(24) on a `C4cD` diagram and get the same result.

2.5 Spin and isospin

We have prescriptions for both spin and isospin. They both need to be resolved to yield the wanted linear combination of computed correlation functions.

We have already expressed the spin projection in terms of multi-particle operators. The isospin projected operators can therefore be resolved there. From there we can spin project onto given momentum and irrep and so forth. The result will be multiple-particle operators with definite spin and isospin. Taking these we can do the Wick contractions.

How do we get the dataset names? Just as we would have to parse the Mathematica output into HDF5 dataset name *patterns* after doing the Wick contractions right after the isospin projection, we now need to extract the full dataset name from the symbolic expression that contains the concrete momenta.

That expression will likely have very many summands. Parsing of it needs to be done as part of the program. Either with the PyParsing library in Python or within Mathematica.

3 Choice of languages and libraries

We have a natural interface between the group theory part that provides us with a table of matrices in various irreps and the projection which contains the

physical process.

The projection shall touch the actual numeric correlators as late as possible. I want to have it compute the spin and isospin projection operator as a data frame and only then apply it to the data.

For the choice of programming languages and libraries keep in mind that programming languages are not tools but the very material from which we build things. They stick around once the project is finished.

3.1 Group theory

For the group theory there are a few options:

Copying tables There are various books and papers which list all the things that we need. Copying them is just error-prone and we would have to check for consistency ourselves. Apparently the `Bethe` Maple library also has it copied from a paper.

Maple This is bad because neither the university nor the institute has have a license for it. We could ask the institute to buy one, though I do not believe that the institute can buy the student version for around 150 EUR. The `Bethe` library has been used by Markus for the ρ-project. Perhaps we just ask him to export all we need and hope that we never have to touch it again.

Mathematica The university has a Mathematica campus license, so although that is license riddled as well, at least *we* can use it. The octahedral group is not directly available in Mathematica but I managed to get it via the representation as a permutation group. I can also construct the little groups (also known as stabilizers) but not the irreps.

Sage I have been looking into alternatives a bit yesterday and found that Sage has the group in some form that one can at least get the multiplication and character table, see this snippet.

SciPy This has Clebsch-Gordan coefficients and Wigner-3j symbols, but not really much more.

3.2 Projection operator construction

Also we need Clebsch-Gordan coefficients, but that should be covered in all the packages that we consider.

There are very likely to be cancellations between different terms. The rotation matrices contain factors like $1/\sqrt{2}$ and therefore we likely want to do symbolic algebra. We have these options:

Maple Not a good idea as we do not want to depend on it more than strictly needed.

Mathematica We have a campus license, it is a very powerful language. However it is a proprietary tool and our group has limited experience with it. I personally do not want to invest too much learning into a proprietary tool.

Wigner-D matrices are available as `WignerD`.

Sage For what we need it will likely be overpowered. As it is a pain to install, we should only do that when strictly needed.

SymPy This would be my favorite as I and others know Python and it is free software.

Wigner-D matrices are available as `physics.quantum.spin.Rotation`

3.3 Wick contraction

Cadabra While asking about Wick contractions in SymPy, the author of the Cadabra CAS system reached out to me offering help implementing Wick contractions in that system. It is programmable in Python.

Mathematica We have the Quark Contraction Tool already available.

SymPy There is `physics.secondquant.wicks`, which seems to be able to do Wick contractions. Also there is `physics.secondquant.contraction` which does a contraction between two fermionic operators. I have spend the afternoon of 2019-01-29 looking into it, but the definition of the fermion creation and annihilation operators seem to not support a tensorial structure, at least not with the Wick contractions provided. I have asked a question on Stack Overflow about this.

3.4 Projecting the data

Once we have the data frame with the desired linear combinations of the correlators from the contraction code, we just have to load the HDF5 data and project it. This could be done in various languages.

As HDF5 reading seems to be very slow, we might want to consider adding a re-packaging step just as Markus does by converting the HDF5 files from the sLapH contraction code into Pandas HDF5 files.

C++ We do not really need much for projecting the data, except a HDF5 library. Perhaps the performance will be better if we do this in C++. But the most likely bottleneck is the HDF5 reading, therefore it does not matter which language we use.

Python The current projection code uses Python and Pandas. It can work with HDF5 files.

R As the analysis is in R one can consider R for doing the actual projection. We have HDF5 routines there as well. The advantage would be that the people who do not know Python could work on this part of the code.

4 Implementation

4.1 Group theory

4.1.1 Data format for group elements and irreps

In `sciebo/Lattice/Representations` we have a bunch of text files that contain the group elements and all the irreducible representations for the full group and subgroup.

In the file `Oh-elements.txt` we find a CSV table with elements like the following:

Operator	α/π	β/π	γ/π
`E`	0	0	0
`C2x`	0	1	1
`C2y`	0	1	0
`C2z`	0	0	1
`C31+`	0	0.5	0.5

These are the group elements and their action parametrized in terms of the three Euler angles α, β and γ.

Then there is `Little-group-elements.txt`, a TSV table that contains the subset of operators for each subgroup associated with a momentum vector d. The entries look like this:

$\vec{d}$	LG($\vec{d}$)	Operator	α/π	β/π	γ/π
(0, 0, 1)	`C4v`	`E`	0	0	0
(0, 0, 1)	`C4v`	`C4z+`	0	0	0.5
(0, 0, 1)	`C4v`	`C4z-`	0	0	−0.5
(0, 0, 1)	`C4v`	`sigma_x`	0	1	1
(0, 0, 1)	`C4v`	`sigma_y`	0	1	0
(0, 0, 1)	`C4v`	`sigma_d1`	0	1	0.5
(0, 0, 1)	`C4v`	`sigma_d2`	0	1	−0.5

The representations for the individual subgroups are in files like `C2v-(-1,-1,0)-representations.txt`. There also is `Oh-(0,0,0)-representations.txt` which contains all group elements because the little group is the group itself:

Irrep	Oh	Lg	α/π	β/π	γ/π	row	col	matrix element
`A1g`	`E`	`E`	0	0	0	1	1	1.
`A1g`	`C2x`	`C2x`	0	1	1	1	1	1.
`A1g`	`C2y`	`C2y`	0	1	0	1	1	1.

Irrep	Oh	Lg	α/π	β/π	γ/π	row	col	matrix element
A1g	C2z	C2z	0	0	1	1	1	1.
Eg	E	E	0	0	0	1	1	1.
Eg	E	E	0	0	0	1	2	0.
Eg	E	E	0	0	0	2	1	0.
Eg	E	E	0	0	0	2	2	1.
Eg	C2x	C2x	0	1	1	1	1	1.
Eg	C2x	C2x	0	1	1	1	2	0.
Eg	C2x	C2x	0	1	1	2	1	0.
Eg	C2x	C2x	0	1	1	2	2	1.

The "Irrep" column denotes the irrep name, "Oh" denotes the name of the group element as part of the O_h group and "Lg" the group element name it has within the little group. Little groups belonging to the momenta that are related by a lattice rotation are equivalent to each other but contain different members of the original group.

The irrep matrices are represented in the *long format*. For instance that `C2x` group element in the `Eg` irrep is a 2×2 matrix that is written out as

$$D^{E^+}(C_{2x}) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} .$$

There are two ways that one can represent tensors:

1. A high-dimensional array of the actual values. Each dimension corresponds to one index. Contractions are performed by looping over all indices that stay and then sum up the elements along the contracted dimension.

   The tracking of the index labels it a bit tedious. This also does not scale well when indices are added later as the dimensionality of the arrays has to be increased. Without proper labelling, this will quickly spiral out of control.

   Memory usage is good, though.

2. A data frame in the *long data format*. Contractions are done by *group-by* and *summarize* operations. To contract an index one groups by all the indices that are to stay and summarizes the sum of the values.

   This scheme easily scales with arbitrary many indices and adding indices later on is not a problem. Memory usage is higher, though zeros can be omitted easily.

It seems that Mathematica has the notion of `Dataset`, which means that one can just `Import` a CSV or TSV file and has a data frame as with R or Pandas.