PointGroupNRG Manual

Aitor Calvo-Fernández, acalvo049@ehu.eus

June 26, 2024

Contents

1 Overview

This manual describes how to use the `PointGroupNRG` code. Some context and theory are introduced at certain points, but it does not serve as a comprehensive guide to the NRG technique. For a comprehensive review of the NRG method, see Ref. [**bulla2008**]. For a description of the theory and implementation specific to `PointGroupNRG`, see Refs. [**calvo-fernandez2024**] and [new arxiv], which also contain useful references specific to the applications of symmetry in the NRG.

This manual starts in Section 4 by describing the typical steps in the preparation and execution of an NRG calculation. Section 5 describes the format for the Clebsch-Gordan coefficients that have to be provided in order to use finite point or double group symmetries. Sections 6 and 7 deal with the main functions provided by the code and are organized by describing first the required arguments and then the optional ones, introducing the necessary information along the way. Section 6 covers the `compute_multiplets` function of the `PointGroupNRG.MultipletCalculator` submodule, which constitutes the first step in the setup of a model. Section 7 gives information about the `nrgfull` function of the `PointGroupNRG.NRGCalculator` submodule, which constructs the model and solves it using the NRG.

The functions `compute_multiplets` and `nrgfull` as they are defined in their respective submodules have more optional keyword arguments than those described here. The arguments not described here are in general related to features in development and changing their value might cause the program to crash.

2 Julia basics

To make the manual self-contained, here we describe the variable types required to use the code and how to construct them. Usage examples can be found in the `examples/` directory. Nonetheless, we encourage using the official Julia documentation [**juliadoc**].

The main functions in `PointGroupNRG` have typed arguments, meaning that the variables given as input need to be of the required type for the function to recognize them. Since it is not possible in Julia to define a variable to be of a definite type in global scope as one would do inside a function, *e.g.* `n::Int64=1`, here we describe how to initialize variables to match the desired type.

2.1 Simple variables

- `Int64`: Integer numbers with 64-bit precision. For example, `n=1` initializes `n` as an `Int64`.
- `Float64`: Real floating point number with 64-bit precision. For example, `f=1.0` initializes `f` as a `Float64`.
- `ComplexF64`: Complex floating point number with 64-bit precision. For example, `c=1.0+0.0im` or `c=ComplexF64(1.0)` initialize `c` as the same `ComplexF64`.
- `String`: For example, `s="PointGroupNRG"` initializes `s` as a `String`.

2.2 Containers

- `Tuple{T1,T2,...,Tn}`: immutable aggregation of `n` values, each with its own type. For example, `t=(1,"a",1.0)` is a tuple with type `Tuple{Int64,String,Float64}`.
- `Array{T,N}`: `N`-dimensional array containing elements of type `T`. For example, `zeros(ComplexF64,2,3,4)` creates a 3-dimensional array of complex numbers with dimensions 2, 3 and 4 in the indices 1, 2 and 3, respectively.
- `Vector{T}`: 1-dimensional array containing elements of type `T`. For example, `v=[1.0 2.0 3.0]` and `v=Float64[1 2 3]` create equal vectors of `Float64` elements.
- `Matrix{T}`: 2-dimensional array containing elements of type `T`. For example, `m=[1.0+0.0im 0.0im; 0.0im 1.0+0.0im]` and `m=ComplexF64[1 0; 0 1]` create equal matrices containing elements of type `ComplexF64`. One-dimensional matrices can be created as, *e.g.*, `m1d=[1.0+0.0im;;]`.
- `Dict{Tk,Tv}`: Dictionary containing keys of type `Tk` pointing to values of type `Tv`. For example, `d=Dict("a"=>[1.0])` and `d=Dict{String,Vector{Float64}}("a"=>[1.0])` create equal dictionaries of type `Dict{String,Vector{Float64}}`.

3 Symmetry and irreducible representations

The code works for three symmetry types, one for each symmetry-breaking situation and each with its own irreducible representations, although the latter share the same structure:

- Crystal field:

$$G_{\mathrm{UPS}} = U(1)_{\mathrm{C}} \otimes P_{\mathrm{O}} \otimes SU(2)_{\mathrm{S}} \tag{1}$$

  is the combination of the unitary charge symmetry $U(1)_{\mathrm{C}}$ leading to particle conservation, the orbital point group symmetry P_{O}, and the rotational symmetry in spin space $SU(2)_{\mathrm{S}}$ leading to total spin conservation. The irreps are

$$\Gamma_{\mathrm{UPS}} = (N, I, S), \tag{2}$$

  where N is the particle number, I is the point group irrep, and S is the total spin, and they are represented in the code with variables of the type `Tuple{Int64,String,Float64}`, for instance `(2,"A1g",2.0)` represents the irrep of states with $N = 2$ particles, orbital symmetry $I = A_{1g}$ and total spin $S = 2$; this is the same representation that is going to be used for all symmetry groups: the `Int64` slot is used for the particle number, the `String` slot for the finite (point or double) group irrep, and the `Float64` for the total spin or total angular momentum.

- Spin-orbit coupling:

$$G_{\mathrm{UJ}} = U(1)_{\mathrm{C}} \otimes SU(2)_{\mathrm{OS}} \tag{3}$$

  is the combination of the unitary charge symmetry $U(1)_{\mathrm{C}}$ leading to particle conservation and the spin-orbital rotation group $SU(2)_{\mathrm{OS}}$ leading to the conservation of total angular momentum. The irreps are of the type `(2,"anything",3.5)`, which is the irrep of states with $N = 2$ particles and total angular momentum $J = 3.5$; `"anything"` in the `String` slot just means that the slot is unused, so it can be filled with any `String` and the code will internally handle it as an identity irrep.

- Crystal field and spin-orbit coupling:

$$G_{\mathrm{UD}} = U(1)_{\mathrm{C}} \otimes D_{\mathrm{OS}} \tag{4}$$

  is the combination of the unitary charge symmetry $U(1)_{\mathrm{C}}$ leading to particle conservation and the spin-orbital double group D_{OS}. The irreps are of the type `(2,"A1g",0.0)`, which is the irrep of states with $N = 2$ particles and belonging to the double group irrep $I =$ "$A1g$"; `0.0` is the identity irrep of angular momentum (spin S or total angular momentum J), which serves to turn off the continuous rotation symmetry $SU(2)$.

G_{UD} and G_{UJ}.

4 Workflow

The number of steps required for an NRG calculation varies depending on the amount of setup information that is already available. Here we describe all the necessary steps for a full calculation starting from scratch for a model requiring Clebsch-Gordan coefficients for a point or double group. The workflow consists of three main steps:

1. Prepare the directory containing the Clebsch-Gordan coefficients following the format specificed in Section 5. These coefficients will then be read by the functions executed in the next steps. This step is not required for models with total angular momentum conservation, since the Clebsch-Gordan coefficients for the $SU(2)$ spin-orbital symmetry group are computed by the program.

2. Compute the multiplet states for the electronic degrees of freedom of the conduction band and, if necessary, for the impurity (see Section 7.2.2). This is achieved with the `compute_multiplets` function described in Section 6, which calculates the multiplet states and stores them into an appropriate format. This step has to be performed only once for each irrep of the electronic degrees of freedom of a given model, so it is recommended to include it in an independent script separate from step 3.

3. Construct the model and perform the NRG calculation using the `nrgfull` function described in Section 7.2.2.

5 Clebsch-Gordan coefficients

Calculations for models with point or double group symmetries require the Clebsch-Gordan coefficients for the group given in the following format: The coefficients are organized according to the reduction of irrep products. If `A` and `B` are the user-given names of two irreps of the finite group G, then their decomposition is

$$\texttt{A} \boxtimes \texttt{B} = \oplus_{\texttt{C}} \texttt{L(C)C}, \tag{5}$$

where `C` are irreps of G and `L(C)` is the number of times `C` appears in the decomposition. On top of the names, the irreps of G must be arbitrarily numbered by the user, starting from 0. If `A`, `B` and `C` are the `a`-th, `b`-th and `c`-th irreps of G, respectively, then the Clebsch-Gordan coefficients of the irrep decomposition of `A`$\boxtimes$`B` must be contained in a file called `axb_AxB.txt`. That file should be organized in paragraphs separated by blank lines, each paragrah containing the Clebsch-Gordan coefficients associated to the subspaces belonging to each irrep in the following format:

```
c C l
( 1 1 | 1 ) = <( A, 1 ; B, 1 | C, 1 )>
...
```

where `l`= 1,...,`L` labels the distinct subspaces belonging to `C` that arise in the decomposition of `A`⊠`B`, and `<( A, 1 ; B, 1 | C, 1 )>` is meant to be substituted by the value of the Clebsch-Gordan coefficient given in any format that Julia can parse into a complex number. The dots `...` have to be filled with the rest of the Clebsch-Gordan coefficients following the same format,

```
( gA gB | gC ) = <( A, gA ; B, gB | C, gC )
```

where `gA`, `gB` and `gC` are the partner numbers of the irreps `A`, `B` and `C`, respectively. Only the non-zero coefficients are necessary. To compare this with the notation in Refs. [**calvo-fernandez2024**] and [new arxiv], substitute the irreps (partners) following the rule $\Gamma_A \leftrightarrow$`A` ($\gamma_A \leftrightarrow$`gA`), $\Gamma_B \leftrightarrow$`B` ($\gamma_B \leftrightarrow$`gB`) and $\Gamma_C \leftrightarrow$`C` ($\gamma_C \leftrightarrow$`gC`). The files for all the irrep combinations `A`⊠`B` with `a`≥`b` (to avoid redundancy) must be stored into a directory.

6 Multiplet calculation: `compute_multiplets`

The function `compute_multiplets` computes the many-body multiplet states arising from the combination of electrons in states belonging to the same one-electron irrep. If the impurity and conduction electrons occupy states belonging to irreps Γ_1 and Γ_2, for instance, with dimensions $\dim(\Gamma_1)$ and $\dim(\Gamma_2)$, respectively, then `compute_multiplets` has to be run once for Γ_1 and once for Γ_2. This is also the case if there are several multiplet subspaces belonging to Γ_1, for example, because the resulting states are distinguished only by an outer multiplicity label that is taken care of by `nrgfull` (see Section 7). The function is defined as itemize

```
function compute_multiplets(
    symmetry::String ;
    irrep::SF="" ,
    multiplets_path::String="multiplets" ,
    clebschgordan_path::String=""
) where {SF<:Union{String,Float64}}
```

6.1 Necessary arguments

6.1.1 `symmetry::String`

Determines the type of symmetry. The accepted values are:

- `"pointspin"` or `"PS"`: Uses the symmetry type

$$G_{\mathrm{UPS}} = U(1)_{\mathrm{C}} \otimes P_{\mathrm{O}} \otimes SU(2)_{\mathrm{S}}, \tag{6}$$

  which is the outer direct product of unitary charge symmetry $U(1)_{\mathrm{C}}$ (particle conservation), the orbital point group P_{O}, and spin rotation symmetry $SU(2)_{\mathrm{S}}$ (conservation of total spin).

- `"doublegroup"` or `"D"`: Uses the symmetry type

$$G_{\mathrm{UOS}} = U(1)_{\mathrm{C}} \otimes D_{\mathrm{OS}}, \tag{7}$$

  which is the outer direct product of unitary charge symmetry $U(1)_{\mathrm{C}}$ (particle conservation) and the spin-orbital double group D_{OS}.

- `"totalangularmomentum"` or `"J"`: Uses the symmetry type

$$G_{\mathrm{UJ}} = U(1)_{\mathrm{C}} \otimes SU(2)_{\mathrm{OS}}, \tag{8}$$

  which is the outer direct product of unitary charge symmetry $U(1)_{\mathrm{C}}$ (particle conservation) and spin-orbital rotational invariance (conservation of total angular momentum).

6.2 Optional keyword arguments

6.2.1 `irrep::SF=""`

One-electron irrep for which to compute the multiplet states. The specification of the irrep changes depending on the value of `symmetry` (6.1.1):

- If `symmetry=="PS"` or `symmetry=="pointspin"`, then `irrep` must be a `String` and is the name of the name of the point group irrep as it appears in the Clebsch-Gordan coefficient files (see 5).
- If `symmetry=="D"` or `symmetry=="doublegroup"`, then `irrep` must be a `String` and is the name of the name of the double group irrep as it appears in the Clebsch-Gordan coefficient files (see 5).
- If `symmetry=="J"` or `symmetry="totalangularmomentum"`, then `irrep` must be a `Float` and is the value of the total angular momentum J of the electrons, where $\mathbf{J} = \mathbf{L} + \mathbf{S}$.

6.2.2 `clebschgordan_path::String=""`

Absolute or relative path of the directory where the Clebsch-Gordan coefficients are stored (see 5). It must be provided only if `symmetry=="D"`, `symmetry=="doublegroup"`, `symmetry=="PS"`, `symmetry=="pointspin"`, as the Clebsch-Gordan coefficients of $SU(2)$ are computed by the program.

6.2.3 `multiplets_path::String="multiplets"`

Absolute or relative path of the parent directory where the directories containing the multiplet states for the chosen irreps (see 6.2.1) will be stored. It must be the same for irreps intented to be used in the same NRG calculation. If it does not already exist, it is automatically created.

7 NRG calculation: `nrgfull`

The `nrgfull` function constructs the model from the input and the multiplet states computed by `compute_multiplets` (see 6), and it solves it using the NRG method. It can be used to calculate thermodynamic functions (see 7.2.2), spectral functions (see 7.2.15), and/or thermodynamic weights (see)

```
function nrgfull(
    symmetry::String ,
    label::String ,
    L::Float64 ,
    iterations::Int64 ,
    cutoff::IF ,
    shell_config::Dict{SF,Int64} ,
    tunneling::Dict{SF,Matrix{ComplexF64}} ;
    multiplets_path::String="multiplets" ,
    calculation::String="IMP" ,
    impurity_config::Dict{SF,Int64}=Dict{SF,Int64}() ,
    onsite::Dict{SF,Vector{ComplexF64}}=Dict{SF,Vector{ComplexF64}}() ,
    interaction::{Tuple{String,Float64},Matrix{ComplexF64}}=Dict{ Tuple{String,Float64}
    spectrum::Dict{ClearIrrep,Vector{Float64}}=Dict{ClearIrrep,Vector{Float64}}() ,
    lehmann_iaj::Dict{ClearTripleG,Array{ComplexF64,4}}=Dict{ClearTripleG,Array{ComplexF
    # run only until a certain point for partial information
    # - 2-particle multiplets
    # - impurity spectrum
    # - impurity-shell spectrum
    until::String = "",
    clebschgordan_path::String="" ,
    identityrep::String="" ,
    z::Float64=0.0 ,
    max_SJ2::Int64=10 ,
    channels_dos::Dict{ SF , Vector{Function} }=Dict{ SF , Vector{Function} }() ,
    mine::Float64=0.0 ,
    betabar::Float64=1.0 ,
    spectral::Bool=false ,
    broadening_distribution::String="loggaussian" ,
    spectral_broadening::Float64=0.5 ,
    K_factor::Float64=2.0 ,
    compute_impurity_projections::Bool=false ,
    band_width::Float64=1.0 ,
    print_spectrum_levels::Int64=0 ,
) where {SF<:Union{String,Float64},IF<:Union{Int64,Float64}}
```

7.1 Necessary arguments

7.1.1 `symmetry::String`

See 6.1.1.

7.1.2 `label::String`

Label or name given to the system. It will appear in the names of the output files: thermodynamic functions, spectral functions and impurity projections. Output files are ovewritten by subsequent calculations with the same value of `label`.

7.1.3 `L::Float64`

Discretization parameter Λ. See Ref. [**bulla2008**]. Lower values provide more accurate results, larger values result in convergence for lower cutoffs (see 7.1.5). Large values of Λ usually produce low resolution thermodynamic and spectral functions that can contain spurious oscillations. Oscillations can be removed in many cases by averaging over even and odd step results, which is done automatically, resulting in improved spectral functions and often completely satisfactory thermodynamic functions. It is possible to go further by averaging over various discretizations (see 8). In general, it is recommended to use low values $\Lambda \in [2, 3]$ for the spectral functions, while for thermodynamic functions values as large as $\Lambda = 10$ often give satisfactory results.

7.1.4 `iterations::Int64`

Number of iterations in the NRG calculation. Lower temperatures (for thermodynamic function calculations) and lower energies (for spectral function calculations) are reached with more iterations.

7.1.5 `cutoff:IF`

Cutoff imposed on the multiplets: after each iterations, multiplets above the cutoff are discarded. The type of cutoff varies depending on the type of the input `cutoff`:

- If `cutoff` is an `Int`, *e.g.* `300`, then it specifies the number of multiplets kept at each iteration.
- If `cutoff` is a `Float`, *e.g.* `7.0`, then it specifies the cutoff energy: multiplets with energies larger than that are discarded.

In both cases, the program tries to avoid breaking accidental degeneracies by checking for a small energy window above the imposed cutoff and keeping also those states.

7.1.6 `shell_config::Dict{SF,Matrix{ComplexF64}}`

It specifies the configuration of the conduction channels. It has the structure

```
Dict(
    G1 => n1,
    G2 => n2,
    ...
)
```

where `Gi` are the irreps in the format described in 6.2.1 and `ni` specify the number of channels with the symmetry defined by the irrep. The multiplet states for each irrep `Gi` must have been calculated previously with `compute_multiplets` (6).

7.1.7 `tunneling::Dict{SF,Matrix{ComplexF64}}`

It specifies the tunneling amplitudes $V(\Gamma_a)_{r_a r_b}$ as a dictionary with the format

```
Dict(
    G1 => amplitudes1,
    G2 => amplitudes2,
    ...
)
```

where `Gi` are the irreps Γ_a in the format described in 6.2.1 and `amplitudesi` are the amplitudes given as a `MatrixComplexF64` with indices r_a, r_b.

7.2 Optional keyword arguments

7.2.1 `multiplets_path::String="multiplets"`

Same as 6.2.3.

7.2.2 `calculation::String="IMP"`

The value of this variable sets whether the impurity is included in the calculation or not, and is relevant in particular for the calculation of thermodynamic functions, which is performed automatically in every case. These are the possible values:

- `calculation="IMP"`: the impurity is included in the calculation, which is necessary for spectral function calculations (7.2.15) and impurity projection (7.2.19) calculations. The thermodynamic functions for the model are stored in the directory `thermodata/`, which is created automatically if it does not previously exist, and are distinguished by containing `imp` in their names. If a `calculation="CLEAN"` run has been performed (see below), the impurity contribution to the thermodynamic functions is automatically computed by subtracting the `"CLEAN"` results from the `"IMP"`

results (see Ref. [**bulla2008**] for more details), and the resulting functions are stored in `thermodata` and distinguished by containing `diff` in their names.

- `calculation="CLEAN"`: the impurity is excluded for the calculation, leaving only the conduction channels in the model. The resulting thermodynamic functions are stored in `thermodata` and distinguished by containing `clean` in their names. This is typically only used for calculating the impurity contribution to the thermodynamic functions.

7.2.3 `impurity_config::Dict{SF,Int64}`

It defines the degrees of freedom of the impurity for the standard Anderson model (not the ionic model) defined in terms of the on-site energies (7.2.4) and the interaction term (7.2.5). The program assumes that one model or the other is intended based on the value of 7.2.6 and it checks whether the rest of the necessary variables for that model are provided. `impurity_config` follows the same format as 7.1.6 and it also requires the previous calculation of the multiplet states using `compute_multiplets` (6).

7.2.4 `onsite::Dict{SF,Vector{ComplexF64}}=Dict{...}()`

On-site energy term

$$\sum_a \epsilon(\Gamma_a)_{r_a} c_a^\dagger c_a \tag{9}$$

given in the format

```
Dict(
    G1 => energies1,
    G2 => energies2
    ...
)
```

where `Ga` is the irrep Γ_a given in the same format as 6.2.1 and `energiesa` is a vector with index r_a containing the eigenenergies corresponding to the multiplets $m_a = (\Gamma_a, r_a)$. For more information, see Refs. [**calvo-fernandez2024**] and [new arxiv].

7.2.5 `interaction::Dict{Tuple{String,Float64},Matrix{ComplexF64}}=Dict{...}()`

Parameters of the interaction term

$$\sum_{a,b,c,d} U_{abcd} c_a^\dagger c_b^\dagger c_c c_d \tag{10}$$

defined by the set of parameters $U(\Gamma_u)_{r_u r_v}$ from which the rest are constructed following the procedure described in Ref. [**calvo-fernandez2024**]. The argument `interaction` contains the parameters $U(\Gamma_u)_{r_u r_v}$ in the format

```
Dict(
    G1 => matrix_elements1,
    G2 => matrix_elements2
    ...
)
```

where `Gu` are the two-particle irreps Γ_u and `matrix_elements` are the matrix elements $U(\Gamma_u)_{r_u r_v}$ for that irrep Γ_u with r_u, r_v as indices. The irreps `Gu` are given in `Tuple{String,Float64}`, which has to fulfill the following conditions depending on the symmetry of the system:

- First entry (`String`): It is the point or double group irrep. If `symmetry=="J"` or `symmetry=="totalangularmomentum"`, the program will automatically transform the given value to the standard identity irrep value `"A"`.
- Second entry (`Float64`): If `symmetry=="PS"` or `symmetry=="pointspin"`, it is the total spin quantum number. If `symmetry=="J"` or `symmetry=="totalangularmomentum"`, it is the total angular momentum quantum number. Otherwise, the given value will be transformed to the identity irrep value `0.0`.

See also 7.2.6.

7.2.6 `spectrum::Dict{Tuple{Int64,String,Float64},Vector{Float64}}=Dict{...}()`

It is the spectrum of the impurity in the ionic model, which defines the impurity term in the Hamiltonian (see [new arxiv])

$$H_{\text{imp}} = \sum_i \epsilon(\Gamma_i)_{r_i} |i\rangle\langle i|. \tag{11}$$

The spectrum is provided in the format

```
Dict(
    G_1 => energies_1,
    G_2 => energies_2,
    ...
)
```

where `G_i` is the many-body irrep defined as a `Tuple{Int64,String,Float64}`, with the first entry being the particle number and the second and third entries defined in 7.2.5. All together, an irrep `G_i` (Γ_i) takes the full form `G_i=(N_i,I_i,SJ_i)` ($\Gamma_i = (N_i, I_i, S_i)$ or $\Gamma_i = (N_i, I_i, J_i)$), where `N_i` ($N_i$) is the number of particles, `I_i` (I_i) is the point or double group irrep, and `SJ_i` (S_i or J_i) is the total spin or total angular momentum. For each irrep key `G_i` (Γ_i) there is a `Vector{Float64} energies_i` ($\epsilon(\Gamma_i)_{r_i}$ with fixed Γ_i) associated vector with indices r_i

If this argument is set to a value different from the default, the ionic model is used.

7.2.7 `lehmann_iaj::Dict{G3,Array{ComplexF64,4}}=Dict{...}()`

Reduced Lehmann amplitudes (matrix elements of the creation operator), used in the ionic model.

$$\langle \Gamma_i, r_i || f^\dagger_{\Gamma_a, r_a} || \Gamma_j, r_j \rangle \tag{12}$$

Here `G3` means `NTuple{3,Tuple{Int64,String,Float64}}`. The format of `spectrum` is

```
Dict(
    (G_i,G_a,G_j)_1 => amplitudes_1
    (G_i,G_a,G_j)_2 => amplitudes_2
    ...
)
```

where the irreps `G_i` (Γ_i), `G_a` (Γ_a) and `G_j` (Γ_j) are the ones appearing in the reduced matrix elements and `amplitudes_i` are `Array{ComplexF64,4}` with indices α, r_i, r_a, r_j containing the amplitudes for the irreps specified in the key.

7.2.8 `until::String=""`

Setting `until` to a value different from the default `until=""` causes `nrgfull` to stop at a certain stage of the calculation, which varies depending on the value given to `until`:

- `until="2-particle multplets"`: `nrgfull` runs until it computes the impurity part of the Hamiltonian, after which it prints out information about the 2-particle multiplet states and then stops. This is particularly useful in order to know the symmetry-adapted parameters determining the Coulomb interaction that make up the input variable `interaction` (see Ref. [**calvo-fernandez2024**])
- `until="impurity spectrum"`: `nrgfull` runs until it computes the impurity spectrum, after which it prints it out and then stops. This is useful for checking the spectrum resulting from a given choice of symmetry-adapted Hamiltonian parameters `onsite` and `interaction`.
- `until="impurity-shell spectrum"`: `nrgfull` runs until it computes the spectrum of the impurity plus the innermost conduction shell, after which it prints it out and then stops. This is useful for checking the spectrum resulting from the impurity spectrum plus the coupling to the conduction band via tunnneling.

7.2.9 `clebschgordan_path::String=""`

Same as 6.2.2.

7.2.10 `identityrep::String=""`

Identity irrep of the point group or the double group as it appears in the Clebsch-Gordan coefficients given in the directory `clebschgordan_path` (see 6.2.2). It is not necessary if `symmetry=="J"` or `symmetry=="totalangularmomentum"` (see 7.1.1).

7.2.11 `max_SJ2::Int64=10`

Maximum value of twice the total spin, $2S$, (for `symmetry="PS"` or `symmetry="pointspin"`) or twice the total angular momentum, $2J$, (for `symmetry="J"` or `symmetry="totalangularmomentum"`) that is expected to appear in the NRG calculations (see 7.1.1). This is used for computing sums over Clebsch-Gordan coefficients, which is done at the beginning of the calculation. If a value of `max_SJ2` larger than the given one is needed, the program will give an error.

7.2.12 `channels_dos::Dict{SF,VectorFunction}=Dict{SF,VectorFunction}()`

7.2.13 `mine::Float64=0.0`

7.2.14 `betabar::Float64=1.0`

7.2.15 `spectral::Bool=false`

7.2.16 `broadening_distribution::String="loggaussian"`

7.2.17 `spectral_broadening::Float64=0.5`

7.2.18 `K_factor::Float64=2.0`

7.2.19 `compute_impurity_projections::Bool=false`

7.2.20 `band_width::Float64=1.0`

8 Averaging over various discretizations