A COMPUTATIONAL ENVIRONMENT FOR HYPERNUCLEAR STRUCTURE CALCULATIONS

by

GUSTAV R. JANSEN

THESIS

for the degree of

MASTER OF SCIENCE

(Master's degree in Computational Science)

Faculty of Mathematics and Natural Sciences
University of Oslo

May 2008

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Acknowledgements

I would like to thank my wife, Ingunn, without whom this thesis would not have been finished.

I would also like to thank my supervisor, Morten Hjorth-Jensen, for introducing me to the world of nuclear physics and for all the help he has given me during the past two years. I hope we can continue to work together.

In Barcelona, I would like to thank Artur Polls and Isaac Vidana for invaluable help with the completion of my work.

Finally, I would like to thank my friends and family for all their support.

Etterstad, May 2008

Contents

List of Figures

List of Tables

1 Introduction

In this thesis I present a computational environment for hypernuclear structure calculations, consisting of an extensive library as well as several applications and other tools to demonstrate the wide range of capabilities provided by this library. The library has been developed using a combination of a dynamic high-level programming language, Python [1], and compiled low-level programming languages like Fortran77/90 and C/C++. Advantages of both types of languages have been exploited, yielding a library that is both computationally efficient and highly flexible at the same time. As it is implemented in a fully object-oriented manner, individual pieces of the library can easily be substituted and extended, without major revisions. This will prove very useful, when at a later stage, the library is extended to include other interaction models and renormalization procedures, capable of delivering data in a variety of basis sets.

The library is based on an existing program package [2] for nuclear structure calculations. One of the main goals of this thesis, has been to extend this package to include hypernoc degrees of freedom. This has been a long and strenuous process and is by far complete. As of now, this hypernuclear library is capable of the following

- Calculating an effective two-particle interaction, using the V_{low-k} [3] renormalization technique.
- Transform the two-particle interaction to a three-dimensional harmonic oscillator basis.
- Calculate a self-consistent single-particle potential for hypernuclei with closed nucleon shells.

This paves the way for a more rigorous treatment of the many-particle problem with many-particle effective interactions, shell-model calculations and coupled cluster calculations of hypernuclear properties. But as we will see in this thesis, the quality of the hyperon-nucleon interaction models leave rather large uncertainties, resulting in widely different behaviours when used in many-particle calculations.

When preparing this thesis I have:

- Extended existing software for nuclear structure to be able to handle hypernuclear structure.
- Written several Python extension modules, supplying Python with a library structure for hypernuclear structure calculations. This includes baryon-baryon (BB) interactions, renormalization techniques, transformations between basis sets and arbitrary coupling schemes.
- Written a graphical application capable of creating plots of the bare and renormalized interaction for a selected channel. Both diagonal and off-diagonal plots are possible as well as plots of the energy eigenvalues.
- Written a parallellized application to create a renormalized two-particle interaction. The parallellization is at the variation level, where several calculations can be run in parallell for different configurations.

- Written an application for calculating a self-consistent basis to second order in the two-particle interaction. The Python framework is capable of handling arbitrary baryons, but only nucleons, Λ and Σ are implemented in the Fortran specific parts of the library.
- Written scripts to extract and process data.
- Written scripts to create visual representations of several types of data.
- Calculated a self-consistent basis for five doubly magic nuclei with a Λ in an s-shell orbital, using the developed application framework. Results for single-particle energies are presented and the medium dependence of this quantity is shown.

This thesis is divided into six sections, with this introduction being the first section. In addition, an extensive appendix is supplied, containing definitions, some derivations, background information and a complete application programming interface (API) for the library created.

Section two presents an overview of hypernuclear physics, with a presentation of the particles used as effective degrees of freedom. The underlying fundamental interaction is discussed, as well as the effective theory used to model the two-particle interactions. In addition, symmetries of the strong interaction are discussed and how these can facilitate the numerical calculations.

Section three presents the many-particle problem, with a brief overview of what kind of numerical methods are used to solve this problem. A general theory for effective interaction is presented, as well as a detailed discussion of the low-momentum two-particle effective interaction used in this thesis. This section also contains a detailed discussion of the self-consistent procedure used to calculate a new basis for a specified hypernucleus, as well as the numerical expressions needed.

Section four contains a presentation of the program library written, as well as an introduction to some of the applications that have been deveolped using this library. A complete reference to the Python extension modules are referred to the appendix.

Section five presents the single particle energies calculated for ${}^{17}_{\Lambda}$O, ${}^{41}_{\Lambda}$Ca, ${}^{91}_{\Lambda}$Zr, ${}^{133}_{\Lambda}$Sn and ${}^{209}_{\Lambda}$Pb.

Section six contains the concluding remarks.

2 Hypernuclear physics

Hypernuclear physics refers to the study of baryonic many-particle systems containing not only nucleons, but also so-called strange baryons named hyperons. As such, it is really an extension of nuclear physics. Although the baryons and mesons are composite particles - in nuclear physics, the baryons are viewed as the relevant degrees of freedom, with the mesons as the force mediating particles.

To understand the structure of baryons and mesons, it is necessary to look at the underlying fundamental interaction. We will do this in section 2.1.

In order to solve the hypernuclear many-particle problem, we will need a realistic baryon-baryon (BB) interaction. Because of the repulsive properties of the strong interaction, it cannot be used directly in calculating the BB interaction. Instead we will use BB interactions derived from meson exhange theory. This will be discussed in section 2.2.

We will be concerned with finite hypernuclear matter or hypernuclei, where we are aiming for a fully microscopic, self-consistent calculation of properties of hypernuclei. Since this problem is currently too large to handle numerically, we will justify our approximations by the shell structure exhibited by nuclei. The nuclear shell model has been very successful in calculating properties of nuclei, using only interactions between valence nucleons and a closed core. We will reduce the dimensionality of the hypernuclear problem in the same way, defining a closed core that interacts with a hyperon. We will take a closer look at the nuclear shell structure in section 2.3

When studying hypernuclear matter, we come across two distinct types. Infinite hypernuclear matter can describe the high density core of neutron stars, yielding a possibly charge neutral matter. This will not be the focus of this thesis, but the interested reader is referred to [4] and references therein.

Finite hypernuclear matter or hypernuclei, which is our primary focus, consists of an ordinary nucleus with additional hyperons bound to the nucleus. These hypernuclei have been observed and produced with a range of different masses and relatively long lifetimes compared to the timescale of the strong nuclear interaction. They do decay, however, but only as a result of the weak nuclear interaction. We will not treat the weak nuclear interaction in this thesis. We will use a time-independent approach, where we study the hypernucleus at a certain time after the hyperon has been bound to the nucleus, but before the hyperon decays. At this time, the system is considered as a stable object and our calculation will reflect this.

2.1 The strong interaction

There are four observed fundamental interactions in Nature.

- Strong nuclear interaction
- Weak nuclear interaction

- Electromagnetic interaction
- Gravity

The standard model of particle physics combines the theories of the strong nuclear, weak nuclear and electromagnetic interactions into a common framework to describe interactions between subatomic particles. Much work has been and is being done to incorporate gravity into a complete theory, but currently this has not been entirely successful [5].

The theories of the standard model are quantum field theories, so they are compatible with both quantum mechanincs and the special theory of relativity. The components are the quantum theory of electroweak interactions, unifying the electromagnetic and the weak nuclear interactions, and Quantum Chromo Dynamics (QCD), describing the strong nuclear interaction.

The standard model, although its predictions are powerful and accurate, does not give the complete picture, even for the fundamental interactions it describes. Particle mass and coupling constants, describing the strength of the interactions, are still needed as input to the theory. A fully fundamental theory should be able to derive these parameters from first principle.

In short, the standard model claims that all matter is made up of quarks and leptons which are fermions, while the forces are mediated by photons, $W^{\pm}$, Z^0 and gluons, which are bosons. In addition to these two particle groups, there is also the Higgs boson, needed to give the particles mass. This boson has never been observed, but this may change when the LHC (Large Hadron Collider) comes online [6].

Table 2.1 and 2.2 list the properties of the matter particles and force mediating particles respectively, limited to the properties that are essential to the work presented in this thesis. For completeness their antiparticles are also listed.

The matter particles are devided into three generations, reflecting the similarities between particles in different generations. For each particle in a generation, there is a sibling particle in another generation with similar properties.

The strong nuclear interaction is responsible for the interaction between baryons. The fundamental matter constituents are quarks, the force is mediated by gluons and the quantum field theory that describes their dynamics is QCD (Quantum Chromo Dynamics).

The quarks come in six different flavours: up, down, strange, charm, top and bottom. A convenient subdivision is into light quarks (up, down and strange) and heavy quarks (charm, top and bottom). We will only be concerned with baryons made up of light quarks. Selected properties of the quarks are listed in table 2.1.

Each flavour quark has an additional degree of freedom called color, which was introduced to tackle the apparent violation of the Pauli exclusion principle, for the observed Δ^{++} state which consists of three up quarks in a $J^P = \frac{3}{2}^+$ configuration. This state has even angular momentum, so the space part of the wavefunction is symmetric, the flavour part of the wavefunction is symmetric

Generation 1

Name	Symbol	Electric charge	Strangeness	Mass (MeV)[1]
Electron	e^-	-1	0	0.510998918
Positron	e^+	$+1$	0	0.510998918
Electron neutrino	ν_e	0	0	$< 2 \cdot 10^{-6}$
Up quark	u	$\frac{2}{3}$	0	1.5 - 3.0
Up anti-quark	$\bar{u}$	$-\frac{2}{3}$	0	1.5 - 3.0
Down quark	d	$-\frac{1}{3}$	0	1.5 - 3.0
Down anti-quark	$\bar{d}$	$\frac{1}{3}$	0	3 - 7

Generation 2

Name	Symbol	Electric charge	Strangeness	Mass (MeV)[1]
Muon	μ^-	-1	0	105.6583692
Antimuon	μ^+	$+1$	0	105.6583692
Muon neutrino	ν_μ	0	0	$< 2 \cdot 10^{-6}$
Charm quark	c	$\frac{2}{3}$	0	1250 ± 90
Charm anti-quark	$\bar{c}$	$-\frac{2}{3}$	0	1250 ± 90
Strange quark	s	$-\frac{1}{3}$	-1	95 ± 25
Strange anti-quark	$\bar{s}$	$\frac{1}{3}$	-1	95 ± 25

Generation 3

Name	Symbol	Electric charge	Strangeness	Mass (MeV)[1]
Tau lepton	τ^-	-1	0	1776.99
Antitau lepton	τ^+	$+1$	0	1776.99
Tau neutrino	ν_τ	0	0	$< 2 \cdot 10^{-6}$
Top quark	t	$\frac{2}{3}$	0	$1.742 \cdot 10^5 \pm 3.3 \cdot 10^3$
Top anti-quark	$\bar{t}$	$-\frac{2}{3}$	0	$1.742 \cdot 10^5 \pm 3.3 \cdot 10^3$
Bottom quark	b	$-\frac{1}{3}$	0	$4.2 \cdot 10^3 \pm 70$
Bottom anti-quark	$\bar{b}$	$\frac{1}{3}$	0	$4.2 \cdot 10^3 \pm 70$

[1]Particle masses from [7]

Table 2.1: Table of elementary matter particles

Name	Symbol	Force mediated	Mass (MeV)[1]
Photon	γ	Electromagnetic	0
Gluon	g	Strong Nuclear	0
W boson	W	Weak nuclear	$8.0403 \cdot 10^4 \pm 29$
Z boson	Z	Weak nuclear	$9.11876 \cdot 10^4 \pm 2.1$

[1]Particle masses from [7]

Table 2.2: Table of elementary force mediating particles

with three quarks of the same flavour and the spin part of the wavefunction is symmetric with three spins in the same direction. This gives a total symmetric wavefunction with regard to the interchange of two quarks. By introducing the colour wavefunction, with three different colours, the total wavefunction could be made antisymmetric by making the color wavefunction antisymmetric.

QCD is a non-abelian gauge field theory, with SU(3) as the gauge group acting on the color degree of freedom. This gives us eight non-commutating generators, recognized as the gluons, mediating the strong interaction.

2.1.1 Hadrons

Although the quarks are the fundamental matter particles in QCD, a free quark has never been observed. It is believed that the strong interaction has a property named confinement, which means that only combinations of quarks can be observed. These combinations, which will interact strongly, are named hadrons.

So far, only two types of hadrons have been observed. Baryons are made up of three valence quarks, while mesons consist of a quark/antiquark pair. Other combinations have been predicted, with the pentaquark beeing the most likely candidate. But since it was proclaimed in 2003 [8], there have been many conflicting reports. The existence of this particle is therefore still in question. For details see [7] and references therein.

With three quarks, each being spin 1/2 particles, the baryons can couple to both spin 1/2 and spin 3/2. When looking at baryons made up of only light quarks, we have the famous baryon octet (spin 1/2) and decuplet (spin 3/2) depicted in figure 2.1 and 2.2 respectively. The strangeness is read horizontally and the charge is read diagonally downwards and to the right.

Our current implementation only include the spin 1/2 baryons with strangeness $S \geq -1$. This gives a total of six baryons, tabulated in table 2.3.

Name	Symbol	Electric charge	Strangeness	Isospin $t(t_z)$	Mass (MeV)
Proton	p	1	0	1/2 (1/2)	938.272029
Neutron	n	0	0	1/2 ($-1/2$)	939.565360
Λ	Λ	0	-1	0 (0)	1115.683
Σ^+	Σ^+	1	-1	1 (1)	1189.37
Σ^0	Σ^0	0	-1	1 (0)	1192.642
Σ^-	Σ^-	-1	-1	1 (-1)	1197.449

Table 2.3: Table of baryons with strangeness $S \geq -1$. [7]

When using the isospin formalism, the nucleons make up an isospin doublet with isospin $T = 1/2$, the Λ an isospin singlet with isospin $T = 0$ and the Σ an isospin triplet with $T = 1$. When we set up the isospin part of the two-particle wavefunction, it behaves like an angular momentum wavefunction (see appendix C). We can couple the isospin of the two particles to a total isospin, using the rules of angular momentum coupling (see appendix C.1.1). In appendix C.2 the coefficients coupling the two-particle wavefunctions to a total isospin are presented for the different nucleon-nucleon and hyperon-nucleon wavefunctions.

Name	Spin	Parity	Strangeness	Isospin	Mass (MeV)
$\pi^{\pm}$	0	$-$	0	1	139.57018
π^{0}	0	$-$	0	1	134.9766
η	0	$-$	0	0	547.51
$\rho(770)$	1	$-$	0	1	775.5
$\omega(782)$	1	$-$	0	0	782.65
$\eta'(958)$	0	$-$	0	0	757.78
$f_0(980)$	0	$+$	0	0	980
ϕ	1	$-$	0	0	1019.460
$K^{\pm}$	0	$-$	± 1	1/2	493.677
K^{0}	0	$-$	1	1/2	497.648
$K_0^*(\kappa)$	0	$+$	1	1/2	841

Table 2.4: Table of selected strange and non-strange light mesons. [7]

For the mesons, consisting of a quark/antiquark pair, the situation is similar. Two quarks can couple to a total spin $s = 0$ or $s = 1$, giving rise to the two nonets depicted in figures 2.3 and 2.4. As for the baryons, the strangeness is read horizontally, while the charge is read diagonally.

The most relevant mesons used in the meson exchange models we employ are tabulated in table 2.4. However, one look at the list of mesons in [7] will reveal a multitude of different mesons, especially in the mass region above 1 GeV. Modern interaction models incorporate many of these into their calculations, but they have only minor effects. These models also incorporate other mesonic degrees of freedom like the multi-meson exhange.

When discussing the spin and isospin of bosons, it is common to label the spin $s = 0$ bosons for scalar particles, while the spin $s = 1$ bosons are labelled vector particles. The same nomenclature applies to isospin, where the particles are called isoscalar and isovector respectively. In addition, if the parity of the particle is negative, the label "pseudo" is prepended to the label. A meson with spin 0, isospin 1 and negative parity will by this standard be labelled as a pseudoscalar isovector meson. These labels will be used in setting up the lagrangian densities when discussing meson exchange theories in section 2.2.

2.1.2 Symmetries

Symmetry under a transformation $\hat{U}$, means that the Hamiltonian is invariant under the similarity transformation $\hat{U}\hat{H}\hat{U}^{\dagger} = \tilde{H}$. This gives the commutation relations

$$\begin{aligned} \left[\hat{H}, \hat{U}\right] &= 0 \\ \left[\hat{H}, \hat{\Gamma}_i\right] &= 0 \end{aligned} \tag{2.1.1}$$

where $\hat{\Gamma}_i$ are the generators of the group that $\hat{U}$ belongs to. For a proper introduction to the use of group theory in particle physics, see for example [9].

The commutation relations in equation 2.1.1 mean that $\hat{\Gamma}_i$ is a constant of

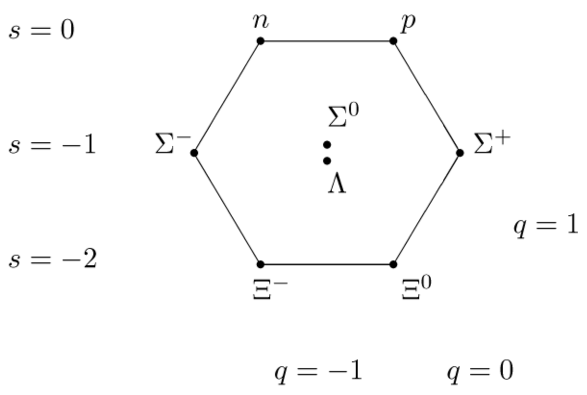


Figure 2.1: Octet of baryons coupled to spin 1/2 [10]

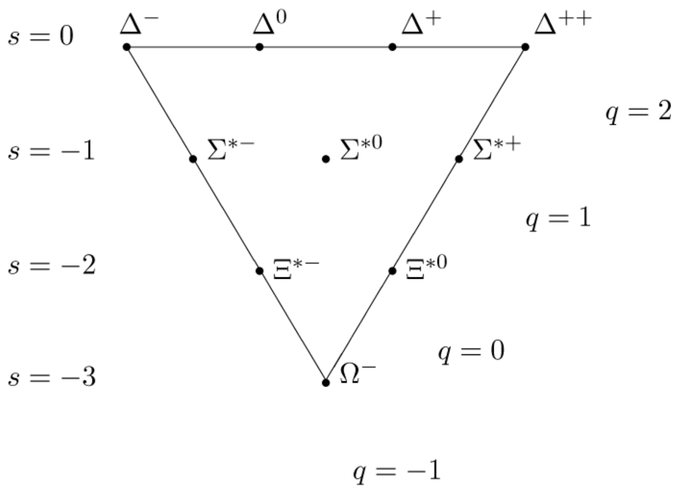


Figure 2.2: Decuplet of baryons coupled to spin 3/2 [10]

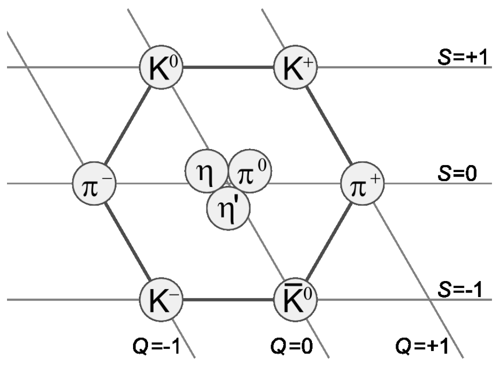


Figure 2.3: Nonet of mesons coupled to spin 0 [10]

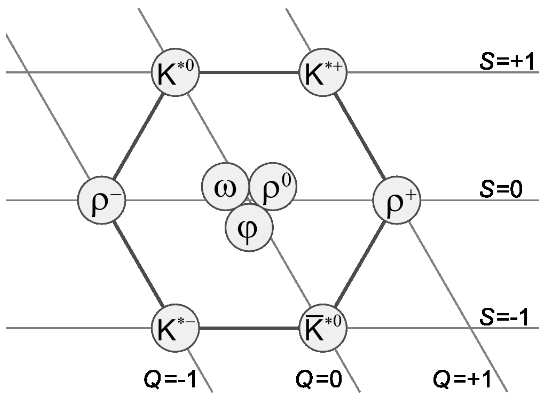


Figure 2.4: Nonet of mesons coupled to spin 1 [10]

motion and if $|\omega\rangle$ is an energy eigenstate, so is $\hat{\Gamma}_i|\omega\rangle$. All these state are degenerate with the same energy.

The interaction between baryons and the underlying fundamental interaction, exhibits invariance under rotation or rotational symmetry. This means that the total angular momentum of a state cannot change as a result of the strong interaction. In the case of the BB interaction, the total angular momentum $\hat{j}$, is the sum of the intrinsic spin and orbital momentum of the two baryons.

$$\hat{j} = \hat{j}_1 + \hat{j}_2 = \hat{l} + \hat{s} = \hat{s}_1 + \hat{s}_2 + \hat{l}_1 + \hat{l}_2 \tag{2.1.2}$$

See appendix C.1.1 for details on how to couple multiple angular momenta to a total angular momenta and details on angular momentum operators in general.

We will use the total angular momenta j as a good quantum number so we can solve the Schrödinger equation for different values of j separately. With j as a good quantum number, we can also put constraints on the total intrinsic spin $\hat{s} = \hat{s}_1 + \hat{s}_2$ and total orbital momentum $\hat{l} = \hat{l}_1 + \hat{l}_2$. We will solve the relative Schrödinger equation in spherical coordinates in a partial wave basis, for specific values of j, l and s separately. This will be discussed in detail in section 3.4.

In 1937, isospin was suggested as a symmetry of the strong interaction [9]. Experiments showed a strong interaction between protons, similar to that between protons and neutrons. The symmetry group was established to be SU(2), isomorphic to the three-dimensional rotation group. The generators $\hat{T}_i$ satisfied all the same relations as the angular momentum operators listed in appendix C. With this new formalism, particles were represented by degenerate multiplets with isospin t. This gives $2t + 1$ components labelled by their projections t_z.

The nucleons make an isospin doublet with isospin $t = 1/2$.

Although it is customary in nuclear physics to label the proton with a negative projection, in hypernuclear physics this is no longer productive. We therefore adopt the particle physics convention, with $t_z = 1/2$ for protons and $t_z = -1/2$ for neutrons.

The Λ is an isospin singlet with $t = 0$, while Σ is an isospin triplet with $t = 1$, where Σ^+, Σ^0 and Σ^- have projections $m = 1, 0, -1$, respectively. The isospin projection of a particle is closely related to its charge. If we define a new quantum number B - Baryon number, where $B = 1$ for baryons and $B = 0$ for mesons, this relation can be written as

$$q = t_z + (B + S)/2, \tag{2.1.3}$$

with S being the particle's strangeness. Isospin invariance is also closely related to charge conservation.

In this thesis we will use the isospin projection t_z as a good quantum number.

In the early 1950's, new tools for discovering particles like the bubble and spark chambers, spurred the discovery of a host of new isospin multiplets of the baryon and meson type. A new quantum number called strangeness was introduced by Gell-Mann and others [11, 12, 13]. This quantum number was conserved in the strong nuclear and electromagnetic interaction, but not in the weak nuclear interaction and explained the long lifetimes of the newly discovered particles.

The strangeness quantum number is simply additive and needs no special treatment when adding up to the total strangeness of a many-particle state.

We will treat strangeness as a good quantum number, reducing the Schrödinger equation to even smaller dimensions.

Gell-Mann also proposed that the strong interaction was invariant under SU(3) symmetry, that tied isospin and strangeness of hadrons into the baryon octet and decuplet, as well as the meson nonets discussed in section 2.1.1. This eventually led to the discovery of the quarks and the developement of QCD.

In addition, the strong interaction is invariant under space inversion or parity. The parity of a state is expressed as $(-1)^l$, where l is the orbital momentum. A transition between states with different parity is not allowed, which is used to reduce the dimension of the numerical calculations we have to perform. This will be discussed in detail in section 3.4.

2.1.3 Numerical approximations

In many-particle physics, perturbation theory is commonly used to approximate the many-particle interaction. The interaction is expanded in a power series in terms of the coupling constant with more and more complicated terms. In QED, the fundamental theory of the electromagnetic interaction, this procedure has been very successful, in part due to the size of the coupling constant. The perturbation expansion converges quickly into the relevant expectation values.

When dealing with the strong interaction, the picture becomes more complicated. An important property of the strong interaction is asymptotic freedom. This means that the coupling constant of the theory is not constant. For small distances, up to about 1 fm, the coupling constant is small and in this region, perturbation theory can be used to approximate the interaction. It is around this distance that hadrons form, so a perturbative expansion in powers of the coupling constant cannot be used to describe interactions between hadrons. The region where this expansion does not converge, is therefore called the non-perturbative region. Interested readers are referred to textbooks on quantum field theory (see for example [14, 9]) for details.

A promising numerical model is Lattice QCD. It is based on Feynman's path integral method, where the functional integral is replaced by a discrete grid or lattice (hence the name). Numerical calculations are done by Monte Carlo Methods (see for example [15]), but the demands on the grid spacing, requires a large number of grid points. The lattice calculations uses periodic boundary conditions, resulting in a finite hypercube. The size of the hypercube needs to be much larger than the Compton wavelength of the object studied and also, the grid spacing has to be much smaller than the Compton wavelength. Since the Compton wavelength is inversely proportional to the mass of the object studied, the number of grid points depends on the mass of this object. The smaller the mass, the more grid points are needed for a realistic calculation.

Recently Ishii *et al* [16] was able to demonstrate important features of the nucelon-nucleon interation, using Lattice QCD. Limitations on computer power did, however, force the authors to use slightly large quark masses, resulting in a pion mass of 530 MeV. According to the authors, within the next five years enough computing power will be available to use real quark masses in the calculations. This will allow them to calulate partial wave contributions to the nucleon-nucleon interaction, constraining the off-shell character from the fundamental QCD Lagrangian. At present, the largest uncertainties in the many-particle calculations, are in the derivations of the bare interactions.

2.2 Meson exchange theory

Due to the non-perturbative nature of QCD at the hadronic level and to our current computational capacity, QCD is currently not a feasible model to use in quantitative numerical calculations of nuclear structure. We need effective theories, where the quark degrees of freedom are frozen, while nucleon and hyperon degrees of freedom are fundamental. We find these in the framework of meson exchange theory.

To justify the use of meson exhange as an approximation of the strong interaction, we will look at the famous "Yukawa potential".

The first efforts in deriving a model of the nuclear force based on the idea of massive particle exchange, was done by Yukawa in 1935 [17] using only classical field theory. The mesons satisfies the Klein-Gordon equation, which is a relativistic version of the Schrödinger equation for a scalar(spin $s = 0$) particle. In the approximation where the baryons are infinitely heavy we get

$$\phi(\mathbf{r}) = \frac{g}{4\pi}\frac{e^{-mr}}{r}, \tag{2.2.1}$$

where g is the coupling constant, describing the strength of the interaction, while m is the mass of the exhanged particle. This gives a mass dependence on the range of the potential. An exchange of a heavier particle, gives a potential with shorter range. Thus the exchange of massive particles, explains the short range nature of the strong interaction.

When going from classical field theory to quantum field theory, the fields need to be quantized. This was first developed for QED where perturbation theory was applied to develop a converging series-expansion of the Schrödinger equation. In QED this can be done because of the size of the coupling constant. But in meson theory, which was orginally thought to be the field theory for strong interactions, the coupling constants are large, so a perturbation expansion will not necessarily converge.

However, it is customary to apply perturbation theory to meson exchange theory, which can be justified by [18], where terms of higher order are of shorter and shorter range, so the long and intermediate parts of the interaction can be described by perturbation theory. At short range, the energies involved are so high, that this low-energy approximation is not longer valid. But since the interaction is strongly repulsive at short range, it can be approximated by a repulsive wall.

Formally the interaction is derived in the framework of relativistic quantum field theory, following the convention of appendix B.1.

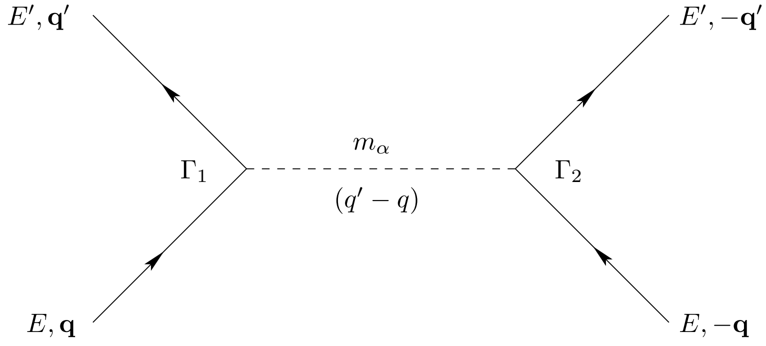


Figure 2.5: Feynman diagram representing a one-boson exchange contribution to BB scattering in the CoM frame. Solid line represents baryons, while dashed line represents meson.

The Feynman diagram in figure 2.5 represents the one-boson exchange contribution to the NN scattering in the CoM frame. According to the Feynman rules defined in for example [19], this diagram corresponds to the amplitude

$$\frac{\bar{u}_1(\mathbf{q}'\Gamma_i u_1(\mathbf{q}) P_\alpha \bar{u}_2(-\mathbf{q}')\Gamma_2 u_2(-\mathbf{q})}{(q'-q)^2 - m_\alpha^2}, \tag{2.2.2}$$

where P_α divided by the denominator is the meson propagator and the Γ_i are the vertices representing the meson-baryon exchange. These are defined for the different meson fields in appendix B.3. The u_i are Dirac spinors representing the baryon fields as defined in appendix B.2. For simplicity, the helicity indices are suppressed. For isovector meson fields, the amplitude must be multiplied by $\tau_1 \cdot \tau_2$, τ_i being the usual Pauli matrices.

The task of evaluating these interactions is a lengthy process and can be found in [20] and references therein.

The final potential is a sum over contributions from the different meson fields. The contribution corresponding to a scalar meson exchange, is given in momentum space in appendix B.4. For pseudoscalar, vector, pseudovector and coordinate space interactions, see [20].

As the scattering phase shifts are only defined in partial waves, it is convenient to express the amplitudes in this basis. The formal developement of the potential in this basis, can be found in [21] and [22].

In addition, this potential can be used in a relativistic framework, as input to a three-dimensional reduction of the Bethe-Salpeter equation. This, however, is beyond the scope of this thesis and will not be discussed further.

We are going to use three different interaction models in this thesis all based on meson exchange. For NN interactions, we will use the N3LO model [23]. This is based on chiral perturbation theory and has proven well when used in nuclear structure calculations. For the YN interactions, we will use two different models. The first model, which we will call J04, is developed by the Jülich group [24] while the second is developed by the Nijmegen group [25] and is named NSC97. This actually consists of six different models, labelled from A to F, each with a different set of coupling constants, where all models reproduce available scattering data.

2.3 Nuclear shell structure

A microscopic self-consistent calculation should reproduce both global and local properties. In this thesis we are concerned with local properties like single-particle wavefunctions and energies. We start with a realistic two-particle interaction derived from meson theory, that reproduce available scattering data in the energy region of 0–500 MeV. Due to the strong repulsion at short range, we have to use an effective two-particle interaction in our calculations.

In order to describe hypernuclear properties at low energies, we choose a restricted space of configurations to perform our calculations. The hypernucleus at low energies, as the nucleus, is best described in spherical coordinates, where the Schrödinger equation is separable in a radial part and an angular part. Formally we write the spatial wavefunction

$$\Psi(\mathbf{r}) = R_{nl}(r)Y_{lm}(\theta, \phi), \tag{2.3.1}$$

where we identify n as the radial quantum number, which can take non-negative integer values, l is identified as the orbital momentum quantum number. This can also take any non-negative integer value. m is called the magnetic quantum number and is identified as the projection of the orbital momentum to a chosen axis. It can take the values $-l \leq m \leq l$, $2l + 1$ values in total.

We also identify $R_{nl}(r)$ as the radial wavefunction, while for a spherically symmetric potential, $Y_{lm}(\theta, \phi)$ are the spherical harmonics.

We will use the labels n and l to describe a single-particle orbit, together with the total angular momentum $j = l + s$, where s is the particle's intrinsic spin. For the particles of interest in this thesis, $s = 1/2$ in units of $\hbar$.

With this nomenclature, the single-particle orbits are labelled n, l and j, where the type of particle is implied and l is expressed in spectroscopic notation. A state with identical n, l and j, will be called an orbital or an orbit in addition to a state.

Due to the Pauli exclusion principle, the number of identical particles that can occupy an orbital is $2j + 1$. We will call a set of orbitals with similar energies a shell, while a full shell will be called closed.

Already in the 1960's [26], there were clear indications of shell structure in the nucleus. The proton and neutron separation energies exhibited clear discontinuities at certain so-called *magic* numbers. It took significantly more energy to remove a nucleon from the nucleus when the number of nucleons matched the *magic* numbers 2, 8, 20, 28, 50, 82 and 126.

An appearence of such a nuclear shell structure can indicate that the motion of the nucleons can be described by an average one-body interaction analogous to the electrons around the nucleus.

Furthermore, experiments showed[26] that many global properties of the nucleus could be described by the interactions of a small number of nucleons outside a closed shell.

We will utilize this picture when we calculate the single-particle energies. We will look at doubly magic nuclei, where both proton and neutron shells are closed, with a single Λ bound to this nucleus in the $0s_{1/2}$ orbital. We will find an average potential the Λ will experience, as a result of the particles in the nucleus.

3 The many-particle problem

In quantum mechanics, the many-particle problem may be defined as the study of interactions between particles and its effects on many-particle systems, such as hypernuclei. The number of degrees of freedom are typically too small to allow for a statistical approach, while being too large too allow for a fully microscopic approach.

There are only a few analytically solvable many-particle problems, with at most three particles, whereas most of the microscopic world at the atomic and molecular level, consists of many-particle systems that have no solutions in a closed form. We need therefore reliable numerical methods in order to approximate these problems.

Typical examples of popular many-particle methods are coupled-cluster methods [27], various types of Monte Carlo methods [28, 29, 30, 31, 32, 33], perturbative expansions [34] and large-scale diagonalization methods [35].

Of these, large-scale diagonalization methods try to tackle the problem head on, diagonalizing the many-particle Hamiltonian directly, using a basis as large as possible. The dimension of this diagonalization problem is $\binom{d}{A}$, where d is the number of available single particle states and A is the number of particles. For ^{16}O, using a small model space constraining all particles to move in the $0s$, $0p$ and $1s0d$ shells, the number of possible Slater determinants is of the order 10^{12}. The number of floating point operations required to solve the diagonalization problem, is proportional to n^3, with n the matrix dimension. Even if we could store the full Hamiltonian matrix in volatile memory and have access to High Performance Clusters(HPC) in the peta-FLOPS(Floating Point Operations Per Second) range, this problem would take more than 10^{13} years to solve. It is clear that this method is unfeasible for problems other than very light nuclei.

For coupled cluster methods, the number of floating point operations scales as $n_o^2 n_u^4$, where n_o is the number of occupied orbitals while n_u is the number of unoccupied orbitals. With this method it is possible to use ab-initio descriptions of nuclei up to the medium and heavy mass region, starting with only two- and three-particle interactions [35].

The problem is complicated further by the fact that the strong interaction is strongly repulsive at short distances. This means that a perturbative calculation, starting with the bare interaction, will converge very slowly, if it converges at all, to the expectation values we are studying. To overcome this problem, we need to define effective two-particle operators that incorporate the short range repulsion, while keeping the matrix elements finite. This process is called renormalization and is discussed in section 3.2.

The renormalized two-particle interaction is used to calculate an appropriate self-consistent basis of single-particle orbitals. These can be used in a perturbative calculation to obtain a many-particle effective interaction in the chosen model space.

In this thesis, we obtain a set of single-particle orbitals, using a self-consistent mean-field approach discussed in section 3.3. This will be done in several model spaces, using different interaction models, where we will start from a harmonic oscillator basis, treating the oscillator parameter as a variational parameter. The final result will be the minima of the single-particle energy for Λ in the $0s_{1/2}$ orbital, when viewed as a function of the oscillator parameter. This will be presented for several doubly magic nuclei for each model space and interaction model, showing the medium dependence of this orbital.

The harmonic oscillator basis will be presented in section 3.4, together with numerical expressions for selected operators. We will also present the different transformation coefficients needed to transform between the basis sets in use.

3.1 Effective interactions

The many-body Hamiltonian for N particles can be written

$$\hat{H}_N = \sum_i^N \hat{t}_i + \sum_i^N \hat{u}_i + \sum_{i\leq j}^N \hat{v}_{ij} + \sum_{i\leq j\leq k}^N \hat{v}_{ijk} + \ldots, \tag{3.1.1}$$

where $\hat{t}_i$ is the kinetic energy operator, $\hat{u}_i$ is a single particle potential, while $\hat{v}_{ij}$ is a two-particle operator. We may add three-particle ($\hat{v}_{ijk}$) interactions or more complicated man-particle interactions.

The Hamiltonian acts on the N-particle wavefunction Ψ_N in the N-particle Schrödinger equation

$$\hat{H}_N|\Psi_N\rangle = E|\Psi_N\rangle, \tag{3.1.2}$$

where E denotes the total energy of the system.

In nuclear physics, the three-body interaction amounts to about ten percent of the total interaction, but in this thesis we are going to truncate the N-particle Hamiltonian so that we only include one- and two-particle interactions. Our new Hamiltonian becomes

$$\hat{H}_N \approx \hat{H}_N^{(2)} = \hat{H}_N^0 + \hat{V}_N^{(2)}, \tag{3.1.3}$$

where we have defined

$$\hat{H}_N^0 = \sum_i^N \hat{t}_i + \hat{u}_i, \tag{3.1.4}$$

$$\hat{V}_N^{(2)} = \sum_{i\leq j}^N \hat{v}_{ij}. \tag{3.1.5}$$

However, our new Schrödinger equation

$$\hat{H}_N^{(2)}|\Psi_N\rangle = \tilde{E}|\Psi_N\rangle, \tag{3.1.6}$$

is as difficult to solve as the original.

The baryons are fermions, so the wavefunction $|\Psi_N\rangle$ needs to be fully antisymmetric. The usual way of ensuring this, is to expand the wavefunction in a basis of so-called Slater determinants (see for example [36]). The Slater determinants $|\Phi_i\rangle$, are fully antisymmetric wavefunctions, consisting of linear combinations of products of single particle orbitals

$$|\Phi_i\rangle = C\sum_{\hat{P}}(-1)^P\hat{P}\prod_{i=1}^{A}|\phi_i\rangle, \tag{3.1.7}$$

where C is a normalization constant, $\hat{P}$ is a permutation operator, acting on the product of single-particle wavefunctions

$$\left(\hat{t}_i + \hat{u}_i\right)|\phi_i\rangle = \epsilon_i|\phi_i\rangle. \tag{3.1.8}$$

The single particle wavefunctions are typically known solutions of a single-particle problem, like plane waves or harmonic oscillator wavefunctions. The dimension of this basis is the number of Slater determinants we can construct for a system. From combinatorics we know that this depends on the number of particles and the number of states(including multiplicity) available. Since we have six types of particles, we have a product of six different Slater determinants, one for each particle. If we let d_i denote the number of states available for a baryon of type i, and n_i the number of baryons of type i, the dimension D of the problem is

$$D = \prod_{i=1}^{6}\binom{d_i}{n_i}, \tag{3.1.9}$$

which quickly becomes very large.

We can now expand the wavefunction in a linear combination of these Slater determinants

$$|\Psi_N\rangle = \sum_{i=1}^{D}\alpha_i|\Phi_i\rangle. \tag{3.1.10}$$

Formally, we need to limit the space in which we do our calculation, but we still have to accomodate for the space left out. We define our model space and complement space by the $\hat{P}$ and $\hat{Q}$ projection operators

$$\hat{P} = \sum_{i=1}^{D}|\Phi_i\rangle\langle\Phi_i|, \tag{3.1.11}$$

$$\hat{Q} = \sum_{i=D+1}^{\infty}|\Phi_i\rangle\langle\Phi_i|. \tag{3.1.12}$$

As projection operators, they have the properties

$$\begin{aligned}\hat{P}+\hat{Q} &= \hat{1},\\ \hat{P}^2 &= \hat{P},\\ \hat{Q}^2 &= \hat{Q}.\end{aligned}$$

Due to the orthogonality of these operators, we can write the Schrödinger equation as a block matrix equation

$$\hat{H}_N^{(2)}|\Psi_N\rangle = \begin{bmatrix}\hat{P}\hat{H}_N^{(2)}\hat{P} & \hat{P}\hat{H}_N^{(2)}\hat{Q}\\ \hat{Q}\hat{H}_N^{(2)}\hat{P} & \hat{Q}\hat{H}_N^{(2)}\hat{Q}\end{bmatrix}\begin{bmatrix}\hat{P}|\Psi_N\rangle\\ \hat{Q}|\Psi_N\rangle\end{bmatrix} \tag{3.1.13}$$

We want an effective interaction operating only in the model space, but still reproducing D eigenvalues of the full Schrödinger equation. We need to find a similarity transformation $\hat{S}$, that decouples the model space from the complement space by solving the so-called decoupling equation.

$$\hat{Q}\tilde{H}_N^{(2)}\hat{P} = \hat{Q}\tilde{H}_N^{(2)}\hat{P} = 0, \tag{3.1.14}$$

with the definition

$$\tilde{H}_N^{(2)} = \hat{S}^{-1}\hat{H}_N^{(2)}\hat{S}. \tag{3.1.15}$$

This gives us an N-particle effective interaction

$$\hat{H}_N^{eff} = \hat{P}\tilde{H}_N^{(2)}\hat{P}. \tag{3.1.16}$$

Solving the decoupling equation directly, can lead to a perturbative expansion of the Brillioun-Wigner type (see section F). An alternative procedure, creates an energy independent interaction using an iterative approach. Several methods exists for solving this problem and the interested reader is referred to [34, 35] for details related to nuclear physics. We will present a brief overview of one of these methods in section 3.2.1.

3.2 Renormalization

The strong interaction is strongly repulsive at short distances. In order to gain the full advantages of perturbation theory, we need to create an effective two-particle interaction that account for this divergent behaviour. This effective two-particle interaction is sometimes called the renormalized interaction and the process of obtaining the effective interaction is called renormalization.

Although recent results by Hagen *et al* [37] shows that coupled-cluster calculations can converge when using the bare two-particle interaction, an effective interaction is still needed for other types of calculation.

There are several methods of creating a renormalized interaction. The G-matrix method [34] is obtained by solving the Lippman-Schwinger equation in a medium, yielding an energy-dependent interaction acting in a truncated Hilbert space.

The no-core interaction [39] is based on diagonalizing the two-particle Schrödinger equation in a large harmonic oscillator space, consisting of several hundred shells. A projection to a smaller Hilbert space of only a few shells is achieved by a similarity transformation.

We have chosen to implement a third renormalization option, based on a similarity transformation to a smaller model space obtained by defining a cutoff in relative momentum space. This method is called V_{low-k} in the literature [3] and is presented in section 3.2.1.

3.2.1 Similarity transformation in momentum-space

Although the formalism we describe below can be applied to an N-particle problem, solving equation 3.1.16 for N particles in order to obtain an effective N-particle interaction, is as difficult as solving the full N-particle problem. Therefore we choose a strategy where we derive an effective two-particle interaction, by solving the full two-particle Schrödinger equation. This two-particle interaction is then used in an N-particle environment.

This method of renormalization, called V_{low-k}, introduces a model space limited by a cutoff in relative momentum space. It consists of two parts - first a diagonalization of the two-particle Schrödinger equation in the domain $k \in [0, \infty)$, then a similarity transformation to a smaller space defined by $k \in [0, \lambda]$, where λ typically is around 2.0 in units of fm^{-1}.

We use a similarity transformation introduced by Suzuki, Okamoto and collaborators [40, 41, 42, 43], which gives an energy independent effective interaction.

Continuing from section 3.1 write up the Schrödinger equation equation in block matrix form

$$\begin{pmatrix} \hat{P}\tilde{H}\hat{P} & \hat{P}\tilde{H}\hat{Q} \\ \hat{Q}\tilde{H}\hat{P} & \hat{Q}\tilde{H}\hat{Q} \end{pmatrix} \begin{pmatrix} \hat{P}\psi \\ \hat{Q}\psi \end{pmatrix} = E_n \begin{pmatrix} \hat{P}\psi \\ \hat{Q}\psi \end{pmatrix}, \tag{3.2.1}$$

where we have simplified the notation by dropping the sub- and superscripts and defined our new two-particle wavefunction

$$|\psi\rangle = |\Psi_2\rangle. \tag{3.2.2}$$

The effective Hamiltonian is defined as

$$\tilde{H} = \hat{S}^{-1}\hat{H}\hat{S} = e^{-\omega}\hat{H}e^{\omega}, \tag{3.2.3}$$

where ω is defined by $\omega = \hat{Q}\omega\hat{P}$. Since the P and Q space do not not overlap, ω has the properties $\omega^2 = \omega^3 \ldots = 0$. By using a Taylor expansion of the operator e^{ω} used in the similarity transformation, we get $\hat{S} = e^{\omega} = \hat{P} + \hat{Q} + \omega$ and $\hat{S}^{-1} = e^{-\omega} = \hat{P} + \hat{Q} - \omega$.

In order for the effective Hamiltonian to have the required properties, the decoupling condition, $P\tilde{H}Q = 0$, must be satisfied. This leads to a non-linear equation for ω

$$\hat{Q}\hat{H}\hat{P} + \hat{Q}\hat{H}\hat{Q}\omega - \omega\hat{P}\hat{H}\hat{P} - \omega\hat{P}\hat{H}\hat{Q}\omega = 0, \tag{3.2.4}$$

where ω is defined by [40, 41, 42, 43]

$$\langle\tilde{\alpha}_Q|\phi_i\rangle = \sum_{\alpha_P}\langle\tilde{\alpha}_Q|\omega|\tilde{\alpha}_P\rangle\langle\tilde{\alpha}_P|\phi_i\rangle, \tag{3.2.5}$$

where we have expanded the wavefunction in a chosen basis

$$|\psi\rangle = \sum_i \alpha_i|\phi_i\rangle, \tag{3.2.6}$$

and defined $|\tilde{\alpha}_P\rangle$ as the states we wish to reproduce in the model space and $|\tilde{\alpha}_Q\rangle$ as the states in the complement space. The V_{low-k} interaction is called a state dependent interaction, because it depends on the exact states reproduced.

We define the matrices A, B and ω by their elements

$$A_{Q,i} = \langle\tilde{\alpha}_Q|\phi_i\rangle, \tag{3.2.7}$$

$$B_{P,i} = \langle\tilde{\alpha}_P|\phi_i\rangle, \tag{3.2.8}$$

and

$$\omega_{Q,P} = \langle\tilde{\alpha}_Q|\omega|\tilde{\alpha}_P\rangle. \tag{3.2.9}$$

The solution for ω can now be found by a matrix inversion and a matrix multiplication.

$$\omega = AB^{-1}. \tag{3.2.10}$$

When a solution for ω is found, the effective interaction is given by

$$\hat{R} = \hat{P}\tilde{H}\hat{P} - \hat{P}\hat{H}_0\hat{P} = \hat{P}\hat{V}\hat{P} + \hat{P}\hat{V}\hat{Q}\omega, \tag{3.2.11}$$

where we have assumed that the projection operators commute with the unperturbed Hamiltonian $\hat{H}_0$.

The effective interaction $\hat{R}$ is generally non-hermitian. We can rewrite this as a hermitian interaction via the orthogonal transformation

$$\hat{V}_{\text{eff}} = \hat{U}^{-1}(\hat{H}_0 + \hat{V})\hat{U} - \hat{H}_0, \tag{3.2.12}$$

where $\hat{U}$ is defined by

$$\hat{U} = \begin{pmatrix} \hat{P}(1+\omega^\dagger\omega)^{-1/2}\hat{P} & -\hat{P}\omega^\dagger(1+\omega\omega^\dagger)^{-1/2}\hat{Q} \\ \hat{Q}\omega(1+\omega^\dagger\omega)^{-1/2}\hat{P} & \hat{Q}(1+\omega\omega^\dagger)^{-1/2}\hat{Q} \end{pmatrix}, \tag{3.2.13}$$

and

$$\hat{U}^T\hat{U} = \hat{U}\hat{U}^T = 1, \;\; \hat{U}^T = \hat{U}^{-1}. \tag{3.2.14}$$

We can then define a Hermitian effective two-body interaction

$$\hat{V}_{\text{eff}} = (\hat{P} + \omega^{\mathrm{T}}\omega)^{1/2}(\hat{P}\hat{H}\hat{P} + \hat{P}\hat{H}\hat{Q}\omega)(\hat{P} + \omega^{\mathrm{T}}\omega)^{-1/2} - \hat{H}_0. \tag{3.2.15}$$

To determine $\hat{V}_{\text{eff}}$, one has to find the square root of the matrix $C = (\hat{P} + \omega^{\mathrm{T}}\omega)$. In the case of C being real and positive definite, the method based on eigenvector decomposition gives generally a stable solution. The final effective interaction is energy independent. For details, see [44] and references therein.

For a two-particle problem, with a centrally symmetric interaction, this can be solved as a one-particle problem in relative coordinates. In this basis we can diagonalize the Hamiltonian directly, obtaining the exact eigenstates and eigenvalues needed for solving the decoupling equation. For a more complex problem, an iterative procedure is used, as outlined in [3].

We start by solving the Schrödinger equation in relative momentum space in a partial wave expansion(see D.4 for the formalism used)

$$\frac{p^2}{2m}\langle p|\psi_\alpha\rangle + \frac{2}{\pi}\int_0^\infty \mathrm{d}k k^2 \langle p|V|k\rangle\langle k|\psi_\alpha\rangle = \epsilon\,\langle p|\psi_\alpha\rangle,$$

where m is the reduced mass of the two-particle state and α represents all other quantum numbers. The last equation is used to obtain the eigenstates and eigenvalues needed to solve the decoupling equation 3.2.4 for ω. We discretize this equation using the following substitutions

$$\begin{aligned} k \in [0,\infty) &\Longrightarrow k_i \in [k_1 \dots k_n], \\ |k\rangle \in [0,\infty) &\Longrightarrow |k_i\rangle \in [|k_1\rangle \dots |k_n\rangle], \\ \langle k|\psi_\alpha\rangle &\Longrightarrow (\psi_1 \dots \psi_n)^T, \qquad \psi_i = \langle k_i|\psi_\alpha\rangle, \\ \langle p|V|k\rangle &\Longrightarrow V_{i,j} = \langle p_i|V|k_j\rangle, \\ \int dk k^2 &\Longrightarrow \sum_{i=1}^{n} k_i^2 w_i, \\ \epsilon &\Longrightarrow \epsilon_i, \end{aligned}$$

where k_i and w_i are the meshpoints and corresponding quadrature weights. This give a system of n equations in n unknowns

$$\begin{aligned} \left(\frac{k_1^2}{2m} + \frac{2}{\pi}k_1^2 w_1 V_{1,1}\right)\psi_1 + \frac{2}{\pi}k_2^2 w_2 V_{1,2}\psi_2 + \dots + \frac{2}{\pi}k_n^2 w_n V_{1,n}\psi_n &= \epsilon_1\ \psi_1 \\ \frac{2}{\pi}k_1^2 w_1 V_{2,1}\psi_1 + \left(\frac{k_2^2}{2m} + \frac{2}{\pi}k_2^2 w_2 V_{2,2}\right)\psi_2 + \dots + \frac{2}{\pi}k_n^2 w_n V_{2,n}\psi_n &= \epsilon_2\ \psi_2 \\ \vdots &= \vdots \\ \frac{2}{\pi}k_1^2 w_1 V_{n,1}\psi_1 + \frac{2}{\pi}k_2^2 w_2 V_{n,2}\psi_2 + \dots + \left(\frac{k_n^2}{2m} + \frac{2}{\pi}k_n^2 w_n V_{n,n}\right)\psi_n &= \epsilon_n\ \psi_n. \end{aligned} \tag{3.2.16}$$

This can be written as an eigenvalue problem

$$D\psi = \epsilon\psi,$$

where

$$D = \begin{bmatrix} \frac{k_1^2}{2m} + \frac{2}{\pi}k_1^2 w_1 V_{1,1} & \frac{2}{\pi}k_2^2 w_2 V_{1,2} & \dots & \frac{2}{\pi}k_n^2 w_n V_{1,n} \\ \frac{2}{\pi}k_1^2 w_1 V_{2,1} & \frac{k_2^2}{2m} + \frac{2}{\pi}k_2^2 w_2 V_{2,2} & \dots & \frac{2}{\pi}k_n^2 w_n V_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{\pi}k_1^2 w_1 V_{n,1} & \frac{2}{\pi}k_2^2 w_2 V_{n,2} & \dots & \frac{k_n^2}{2m} + \frac{2}{\pi}k_n^2 w_n V_{n,n} \end{bmatrix},$$

$\psi = [\psi_1, \psi_2, \dots \psi_n]^T$ are the eigenvectors of D and ϵ are the corresponding eigenvalues.

Numerically, a hermitian matrix is more efficient to diagonalize, so we introduce $|\bar{k}_i\rangle = k_i\sqrt{w_i}|k_i\rangle$ with the properties

$$\sum_{i=1}^{n} |\bar{k}_i\rangle\langle\bar{k}_i| = \mathbb{I}, \quad \langle\bar{k}_i|\bar{k}_j\rangle = \delta_{i,j}.$$

The matrix elements of D, which becomes our Hamiltonian matrix, can now be expressed in a plane wave basis

$$H_{i,j} = \langle\bar{k}_i|H|\bar{k}_j\rangle = \frac{k_i^2}{2m}\delta_{i,j} + \sqrt{w_i w_j}k_i k_j V_{i,j}, \tag{3.2.17}$$

where we for simplicity have incorporated the $2/\pi$ factor in the interaction elements. We arrive at the hermitian matrix

$$H = \begin{bmatrix} \frac{k_1^2}{2m} + k_1^2 w_1 V_{1,1} & k_1 k_2 \sqrt{w_1 w_2} V_{1,2} & \dots & k_1 k_n \sqrt{w_1 w_n} V_{1,n} \\ k_2 k_1 \sqrt{w_2 w_1} V_{2,1} & \frac{k_2^2}{2m} + k_2^2 w_2 V_{2,2} & \dots & k_2 k_n \sqrt{w_2 w_n} V_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k_n k_1 \sqrt{w_n w_1} V_{n,1} & k_n k_2 \sqrt{w_n w_2} V_{n,2} & \dots & \frac{k_n^2}{2m} + k_n^2 w_n V_{n,n} \end{bmatrix}.$$

We diagonalize this matrix to obtain the eigenvectors $\tilde{\alpha}_i$, which are linear combinations of our free spherical waves $\tilde{\alpha}_i = \sum_j \beta_{ij}\phi_j$. To decide which eigenstates are in the model space and which are in the complement space, we calculate the probability of finding each eigenstate in the model space. The eigenstates with the highest probability of being in the model space are $\tilde{\alpha}_P$ while the rest are $\tilde{\alpha}_Q$. We can now find ω from equation 3.2.10 and $\hat{V}_{eff}$ from equation 3.2.15.

We find the effective interaction in the original basis

$$\langle k|V_{\text{eff}}|k'\rangle = \frac{\langle \bar{k}_i|V_{\text{eff}}|\bar{k}_j\rangle}{\sqrt{\omega_i\omega_j}k_i k_j}, \tag{3.2.18}$$

From now on, we will drop the subscript for $\hat{V}_{\text{eff}}$ and call the effective interaction for $\hat{V}$.

This method is selected because of its relatively easy implementation and is therefore an ideal candidate for a test implementation. Because of the momentum space cutoff, high-momentum intermediate states will not be accounted for in the many-particle problem. This can be remedied by including three or many-particle forces explicitly, but this has not been done in this thesis.

The V_{low-k} method will introduce a strong dependence on the chosen cutoff, which cannot be excluded by including more complicated two-particle correlations. Therefore, the problem is typically solved for different cutoffs. A completely realistic computation will have to include many-particle forces generated by different cutoffs.

3.3 Mean-field potential and single-particle energies

In this thesis we will study how a single hyperon behaves in the presence of a doubly magic nucleus. Specifically, we will look at the single-particle energy of a Λ in the $0s_{1/2}$ orbital. Physically, this can be realized by low energy scattering between a hyperon and a nucleus where the hyperon is momentarily trapped in the potential set up by the nucleus.

We will approximate this problem by a single-particle potential or mean-field potential, to second order in perturbation theory. This potential is sometimes called the self-energy and represents the effective energy of a particle in a medium.

We will describe the potential by a set of Goldstone diagrams, depicted to second order in figure 3.1 a). We will adopt the particle/hole formalism. Single-particle states with energy below the Fermi energy are called hole states, while states with energy above are called particle states.

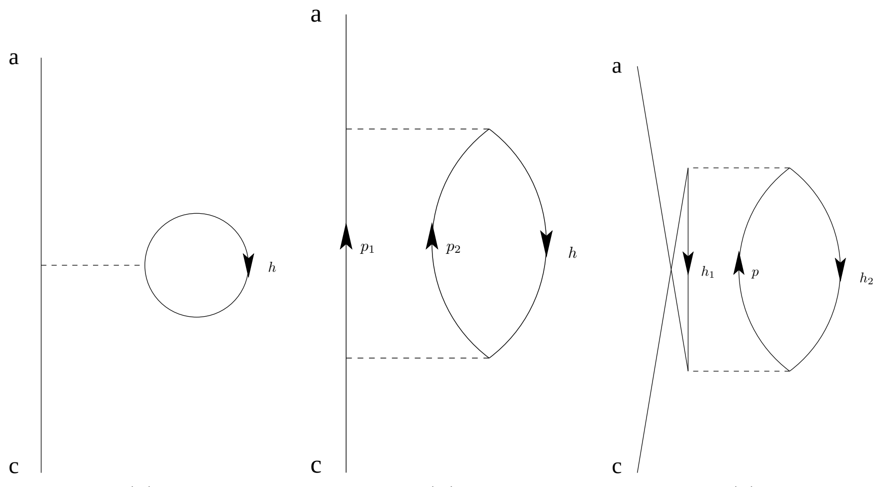


Figure 3.1: Diagrams to second order in the interaction. Diagram (a) is the first order Hartree-Fock term, while diagrams (b) and (c) are the 2p–1h and 2h–1p corrections respectively. The labels a and c represent final and initial states. The particle states are labelled as p, p_1 and p_2, while the intermediate hole states are labelled h, h_1 and h_2. The dotted line represents the interaction.

The first order diagram in figure 3.1 a), is the so-called Hartree-Fock diagram. This represents a particle interacting with a nucleus, where none of the nucleons are allowed in an exited state. The second order diagrams in figure 3.1 b) and 3.1 c), represent so-called one- particle-two-hole and two-particle-one-hole exitations. This means that a particle interacts with a nucleus, where only one of the nucleons can be exited to a particle state. Higher order corrections can also be implemented, but this is a topic for future work.

The expression for the diagrams in figure 3.1, can be expressed in the uncoupled representation using standard diagram rules. We will use single-particle basis states, coupled to a total angular momentum, so it is useful to adopt the angular momentum coupled representation of the diagrams [45].

The expression for the first order diagram is

$$\langle a|u|c\rangle = \frac{1}{2j_a+1}\sum_{J,h}(2J+1)\langle ah|V|ch\rangle, \tag{3.3.1}$$

where $|j_a - j_h| \leq J \leq |j_a + j_h|$ and h is a hole state as defined for the specific nucleus.

The second order diagrams are

$$\langle a|u|c\rangle = \frac{1}{4(2j_a+1)}\sum_{J,h,p_1,p_2}(2J+1)\frac{\langle ah|V|p_1p_2\rangle\langle p_1p_2|V|ch\rangle}{e_c+e_h-e_{p_1}-e_{p_2}}, \tag{3.3.2}$$

where $|j_a - j_h| \leq J \leq |j_a + j_h|$ and

$$\langle a|u|c\rangle = -\frac{1}{4(2j_a+1)} \sum_{J,p,h_1,h_2} (2J+1)\frac{\langle ap|V|h_1h_2\rangle\langle h_1h_2|V|cp\rangle}{e_{h_1}+e_{h_2}-e_p-e_a}, \tag{3.3.3}$$

where $|j_a - j_p| \leq J \leq |j_a + j_p|$. In the above two expressions, p_i and h_i represents particle states and hole states respectively, as defined for the specified nucleus. e_α are the single particle energies of the respective states and j_α are their total angular momenta.

When calculating the single-particle potential from these expressions, the energies and wavefunctions depend on the basis set chosen for the calculation. These will in general not be the same as the solutions to the Schrödinger equation, when solved with this potential. We would like to make this calculation self-consistent, meaning that the wavefunctions and energies used, are solutions to the Schrödinger equation, when calculated with the single-particle potential u. This will be achieved by using an iterative procedure consisting of two steps.

1. Calculate the single particle potential u, using all diagrams up to the specified order.
2. Solve the single particle Schrödinger equation using u as the potential, acquiring new wavefunctions and energies as linear combinations of our chosen basis states.

For each iteration the difference between the old and the new single particle energies are calculated and the iteration continues until this difference is below a threshold value. Formally, the algorithm is presented in figure 3.2.

ϕ_i	Chosen basis of single particle wavefunctions
ψ_i^k	New single particle wavefunction after k iterations.

ψ_i^k is a linear combination of the original basis states

$$\psi_i^k = \sum_{j=0}^{N} C_{ij}^k \phi_j. \tag{3.3.4}$$

In order to keep track of the coefficients C_{ij}^k, we define the matrices C^k with C_{ij}^k as its elements. We can now write

$$\begin{bmatrix} C_{00}^k & C_{01}^k & \cdots & C_{0n}^k \\ C_{10}^k & C_{11}^k & \cdots & C_{1n}^k \\ \vdots & & \ddots & \vdots \\ C_{n0}^k & C_{n1}^k & \cdots & C_{nn}^k \end{bmatrix} \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} \psi_0^k \\ \psi_1^k \\ \vdots \\ \psi_n^k \end{bmatrix}, \tag{3.3.5}$$

shortened to

$$\psi^{\mathbf{k}} = C^k \phi, \tag{3.3.6}$$

when $\phi = [\phi_0, \phi_1, \ldots, \phi_n]^T$ and $\psi^{\mathbf{k}} = [\psi_0^k, \psi_1^k, \ldots, \psi_n^k]^T$. The initial conditions are

$$\psi^0 = \phi, \tag{3.3.7}$$

giving C^0 as the identity matrix.

The single-particle states used when calculating the diagrams in equations 3.3.1, 3.3.2 and 3.3.3 are now linear combinations of the chosen basis states. To express these corrections using the basis states, they will have to be modified slightly.

The first order correction, which can also be derived from the Hartree-Fock variational procedure, becomes

$$\langle a|u|c\rangle = \frac{1}{2j_a+1}\sum_{J,h,h',h''}(2J+1)\langle ah'|V|ch''\rangle, \tag{3.3.8}$$

where j_α are defined as before, h is now the wavefunction of the new hole state, which is a linear combination of basis states, represented by the primed letters

$$|h\rangle = \sum_j C_{hj}|\phi_j\rangle = \sum_{h'}|h'\rangle. \tag{3.3.9}$$

The algorithm for calculating this diagram is presented in figure 3.3

The second order corrections, which would correspond to an extended Hartree-Fock procedure, are modified similarily

$$\langle a|u|c\rangle = \frac{1}{4(2j_a+1)}\sum_{\substack{J\\ h,h',h''\\ p_1,p_1',p_1''\\ p_2,p_2',p_2'}}(2J+1)\frac{\langle ah'|V|p_1'p_2'\rangle\langle p_1''p_2''|V|ch''\rangle}{e_c+e_h-e_{p_1}-e_{p_2}} \tag{3.3.10}$$

and

$$\langle a|u|c\rangle = -\frac{1}{4(2j_a+1)}\sum_{\substack{J\\ p,p',p''\\ h_1,h_1',h_1''\\ h_2,h_2',h_2''}}(2J+1)\frac{\langle ap'|V|h_1'h_2'\rangle\langle h_1''h_2''|V|cp''\rangle}{e_{h_1}+e_{h_2}-e_p-e_a}, \tag{3.3.11}$$

where the algorithms for calculating these diagrams are presented in figure 3.4 and 3.5 respectively. The particle and hole states are defined similar to h, as a sum over the corresponding primed states as in equation 3.3.9.

We will be using a translationally invariant Hamiltonian, as is customary in nuclear physics,

$$H = \sum_{i=1}^{A}\frac{\mathbf{k_i}^2}{2m_i} - \frac{\mathbf{K}^2}{2M} + \sum_{i<j}^{A}V(i,j), \tag{3.3.12}$$

where A is the number of particles, $\mathbf{k}_i$ is the momentum of particle i, m_i is its mass, M is the total mass of the system while $\mathbf{K}$ is the center of mass momentum defined by

$$\mathbf{K} = \sum_{i=1}^{A}\mathbf{k}_i. \tag{3.3.13}$$

This can be organized as

$$H = \sum_{i=1}^{A} \left[\frac{\mathbf{k_i}^2}{2m_i} - \frac{\mathbf{k_i}^2}{2M} \right] + \sum_{i<j}^{A} \left[V(i,j) - \frac{\mathbf{k_i} \cdot \mathbf{k_j}}{M} \right] \tag{3.3.14}$$

$$= \sum_{i=1}^{A} \mathbf{k_i}^2 \left[\frac{1}{2m_i} - \frac{1}{2M} \right] + \sum_{i<j}^{A} \left[V(i,j) - \frac{\mathbf{k_i} \cdot \mathbf{k_j}}{M} \right]. \tag{3.3.15}$$

To solve the Schrödinger equation in this form, we will need the single particle elements $\langle a|\mathbf{k}_i^2|c\rangle$ and the two-particle elements $\langle ab|V|cd\rangle$ and $\langle ab|\mathbf{k_i} \cdot \mathbf{k_j}|cd\rangle$ in our chosen basis.

```
for $a = 1, \ldots, n$
  if a is outside modelspace; continue
  for $c = 1, \ldots, n$
    if c is outside modelspace; continue
    if symmetries are not preserved; continue
    $u \leftarrow 0$
    for d in included diagrams
      $u \leftarrow u + d(a, c)$
    end for
    $k \leftarrow \langle a|t|c\rangle$
    $H(a, c) \leftarrow k + u$
  end for
end for
```

Figure 3.2: Algorithm for calculating the self-energy corrections

```
$result \leftarrow 0$
for $h = 1, \ldots, n$
  if h is not hole; continue
  $j_{min} \leftarrow |j_a - j_h|$
  $j_{max} \leftarrow |j_a + j_h|$
  for $j = j_{min}, \ldots, j_{max}$
    $factor \leftarrow \frac{2j+1}{2j_a+1}$
    $result \leftarrow result + factor * \langle ah|V_j|ch\rangle$
  end for
end for
```

Figure 3.3: 1. order diagram

```
$result \leftarrow 0$
for $h = 1, \ldots, n$
  if h is not hole; continue
  $j_{min} \leftarrow |j_a - j_h|$
  $j_{max} \leftarrow |j_a + j_h|$
  for $p_1 = 1, \ldots, n$
    if $p_1$ is not a particle state; continue
    for $p_2 = 1, \ldots, n$
      if $p_2$ is not a particle state; continue
      for $j = j_{min}, \ldots, j_{max}$
        $factor \leftarrow \frac{2j+1}{4(2j_a+1)}$
        $denom \leftarrow e_c + e_h - e_{p_1} - e_{p_2}$
        $nom \leftarrow \langle ah|V_j|p_1p_2\rangle\langle p_1p_2|V_j|ch\rangle$
        $result \leftarrow result + factor * \frac{nom}{denom}$
      end for
    end for
  end for
end for
```

Figure 3.4: 2. order diagram

```
$result \leftarrow 0$
for $p = 1, \ldots, n$
  if p is not a particle state; continue
  $j_{min} \leftarrow |j_a - j_p|$
  $j_{max} \leftarrow |j_a + j_p|$
  for $h_1 = 1, \ldots, n$
    if $h_1$ is not a hole state; continue
    for $h_2 = 1, \ldots, n$
      if $h_2$ is not a hole state; continue
      for $j = j_{min}, \ldots, j_{max}$
        $factor \leftarrow \frac{2j+1}{4(2j_a+1)}$
        $denom \leftarrow e_{h_1} + e_{h_2} - e_p - e_a$
        $nom \leftarrow \langle ap|V_j|h_1h_2\rangle\langle h_1h_2|V_j|cp\rangle$
        $result \leftarrow result - factor * \frac{nom}{denom}$
      end for
    end for
  end for
end for
```

Figure 3.5: 2. order exchange diagram

3.4 Harmonic oscillator basis

Until now, the calculations have been independent of the chosen basis, except for the calculation of the two-particle effective interaction using the V_{low-k} procedure, which was defined in momentum space.

The interaction models implemented, delivers matrix elements in a relative partial wave decomposition in relative momentum space, see appendix D.3 and D.4 for details. This is also the basis we will use for the V_{low-k} procedure.

We will choose the three-dimensional spherical harmonic oscillator as our two-particle basis for calculating the self-consistent single particle energies. This is a suitable choice for the bound systems we will be studying.

In this section we will define the terms associated with this choice of basis, as well as the basis sets used by the renormalization procedure and interaction models. Also, the equations and operators will be expressed in the appropriate basis sets and the transformations between the basis sets are presented.

3.4.1 Notations

In nuclear physics is is customary to work in a relative so-called partial wave basis, where a spherical wave is expanded in a linear combination of spherical Bessel functions (see appendix A.1) and Legendre polynomials (see appendix A.2). The total orbital momentum l of the relative state is coupled to the total intrinsic spin s, to give a total angular momentum $j = l+s$ (See C.1.1 for details on coupling angular momentum operators).

As part of the V_{low-k} procedure, the momentum space Schrödinger equation is solved for a discrete set of momenta as a matrix diagonalization problem. As diagonalization algorithms scale the number of floating point operations as n^3[46], where n is the dimension of the matrix, we will split the matrix into as many separately solvable parts as possible.

We will accomplish this by exploiting the symmetries of the strong interaction presented in section 2.1.2.

The strong interaction conserves the total angular momentum j, strangeness S, charge q and parity π. This means that no transitions can occur between states with different j, S, q and π and the matrix elements in the Hamiltonian representing these transitions are 0. The Hamiltonian is block diagonal in these sets of quantum numbers.

We can further reduce the size of the block matrix by exploiting the constraints on l and s given by the conservation of j and parity.

Since we are only including particles with $s_i = 1/2$ in units of $\hbar$, a two-particle state can only couple to total intrinsic spin $s = 0, 1$. l can take any non-negative integer value, so for a given j, the possible transitions are

$$\langle s = 0, 1; l = j|V|s = 0, 1; l = j\rangle, \tag{3.4.1}$$

$$\langle s = 1; l = j \pm 1|V|s = 1; l = j \pm 1\rangle. \tag{3.4.2}$$

The parity is given by $\pi = -1^l$, so parity conservation forbids transitions between $l = j$ and $l = j \pm 1$.

We can further reduce the size of the problem by considering conservation of strangeness and charge. We will only consider possible strangeness values to be $S = 0, -1, -2$.

Tables 3.1, 3.2 and 3.3 tabulate the possible particle combinations for different charges, together with the BB label used to identify this channel for strangeness channels $S = 0$, $S = -1$ and $S = -2$, respectively.

q	Label	# subchannels	sub 1	sub 2	sub 3	sub 4
0	nn	1	nn			
1	pn	1	pn			
2	pp	1	pp			

Table 3.1: Table of possible particle combinations for strangeness 0

q	Label	# subchannels	sub 1	sub 2	sub 3	sub 4
-1	$\Sigma^- n$	1	$\Sigma^- n$			
0	Λn	3	Λn	$\Sigma^- p$	$\Sigma^0 n$	
1	Λp	3	Λp	$\Sigma^0 p$	$\Sigma^+ n$	
2	$\Sigma^+ p$	1	$\Sigma^+ p$			

Table 3.2: Table of possible particle combinations for strangeness -1

q	Label	# subchannels	sub 1	sub 2	sub 3	sub 4
-2	$\Sigma^-\Sigma^-$	1	$\Sigma^-\Sigma^-$			
-1	$\Lambda\Sigma^-$	2	$\Lambda\Sigma^-$	$\Sigma^0\Sigma^-$		
0	$\Lambda\Sigma^0$	4	$\Lambda\Sigma^0$	$\Lambda\Lambda$	$\Sigma^0\Sigma^0$	$\Sigma^+\Sigma^-$
1	$\Lambda\Sigma^+$	2	$\Lambda\Sigma^+$	$\Sigma^0\Sigma^+$		
2	$\Sigma^+\Sigma^+$	1	$\Sigma^+\Sigma^+$			

Table 3.3: Table of possible particle combinations for strangeness -2

We can now categorize the channels into the following categories, labelled in spectroscopic notation, prepended by the appropriate baryon-baryon label.

- tensor coupled channel – A channel where transitions between states with different orbital momenta are allowed. Labelled as $BB\ {}^{2s+1}l_j$–${}^{2s+1}l'_j$.
- spin coupled channel – A channel where transitions between states with different intrinsic spins are allowed. Labelled as $BB\ {}^{0-1}l_j$.
- baryon coupled channel – A channel where transitions between different baryon-baryon configurations are allowed. Labelled as $BB\ {}^{2s+1}l_j$.
- uncoupled channel – A channel that is neither tensor coupled, spin coupled nor baryon coupled. Labelled as $BB\ {}^{2s+1}l_j$.

In spectroscopic notation, the orbital momentum label is given by a letter, where the $l = 0$ state is labelled S, the $l = 1, 2$ states are labelled P and D, respectively, while higher orbital momentum states are labelled alphabetically, starting at F.

The Hamiltonian matrices used in the V_{low-k} procedure, are displayed in appendix D.4.

A state in the relative and center of mass (CoM) partial wave basis in momentum space can now be labelled

$$|(B_1B_2)klKLsSqT_zj\alpha\rangle, \tag{3.4.3}$$

where K and L are the CoM momentum and orbital momentum, T_z is the isospin projection, included only to simplify the merger of developed code and existing code. For completeness, α represents all other quantum numbers. The other labels are already defined.

3.4.2 Transformations

The two-particle matrix elements needed to calculate the single-particle energies are given in a spherical harmonic oscillator basis in the laboratory frame, labelled by

$$|(ab)JT_zS\alpha\rangle, \tag{3.4.4}$$

where ab is a shorthand for $(B_an_al_aj_atz_a)(B_bn_bl_bj_btz_b)$, $J = j_1 + j_2$, $T_z = tz_a + tz_b$ and S identifies the strangeness channel. n_i, l_i are the radial and orbital momentum quantum numbers for particle i, while j_i and tz_i denotes its total angular momentum and isospin projection. B_i is the particle type. Since all particles included have spin $s_i = 1/2$, this label is excluded, but still used in the calculations.

The transformation from the relative and CoM frame to the laboratory frame for a two-particle state can be written

$$\begin{aligned} |(ab)JT_zS\alpha\rangle = \frac{1}{\sqrt{1+\delta_{ab}}} &\times \sum_{\lambda sj}\sum_{nNlL} F \times \langle ab|\lambda sJ\rangle \\ &\times (-1)^{\lambda+j-L-s}\sqrt{2\lambda+1}\left\{\begin{array}{ccc} L & l & \lambda \\ s & J & j \end{array}\right\} \\ &\times \langle nlNL\lambda|n_al_al_bl_b\lambda\rangle|(B_aB_b)nlNLsSqT_zj\rangle, \end{aligned} \tag{3.4.5}$$

where $\langle nlNL\lambda|n_al_an_bl_b\lambda\rangle$ is the Moshinsky bracket [47], $|l - L| \leq \lambda \leq |l + L|$ and also $|l_a - l_b| \leq \lambda \leq |l_a + l_b|$. We have defined

$$\langle ab|\lambda sJ\rangle = \hat{j_a}\hat{j_b}\hat{\lambda}\hat{s}\left\{\begin{array}{ccc} l_a & s_a & j_a \\ l_b & s_b & j_b \\ \lambda & s & J \end{array}\right\}, \tag{3.4.6}$$

where $\hat{x} = \sqrt{2x+1}$. The factor F is defined as $F = \frac{1-(-1)^{l+s+T}}{\sqrt{2}}$ if a and b represents identical particles, $F = 1$ otherwise. J is the total angular momentum of the two-particle state.

In addition we will need the transformation from the relative and CoM momentum basis, to the relative and CoM harmonic oscillator basis.

$$|(B_1B_2)nlNLsSqT_zJ\rangle = \int k^2K^2dkdKR_{nl}(k)R_{NL}(K)|(B_1B_2)klKLsSqT_zJ\rangle, \tag{3.4.7}$$

where P_{nl} and P_{NL} are the radial harmonic oscillator functions for the relative and CoM frame respectively, defined in appendix E.1.

3.4.3 Coulomb interaction operator

After the renormalization procedure, the Coulomb interaction is added to the matrix elements of the interaction in the relative and CoM harmonic oscillator basis. We will need the Coulomb operator in this basis. The Coulomb operator is only dependent on the relative motion, so we only need the elements

$$\langle nlm|V_c|n'l'm'\rangle = \frac{q_1q_2}{4\pi\epsilon_0}\int \mathrm{d}rrR_{nl}(r)^*R_{n'l}(r), \tag{3.4.8}$$

from appendix E.3. Here $R_{nl}(r)$ are the radial harmonic oscillator wavefunction, defined in appendix E.1.

We discretize this equation by making the following substitutions

$$\begin{aligned} R_{(n,l)}(r) &\implies (R_1^{(n,l)}\dots R_n^{(n,l)})^T, \qquad R_i^{(n,l)} = R_{(n,l)}(r_i), && (3.4.9)\\ r\in[0,\infty) &\implies r_i\in[r_1\dots r_n], && (3.4.10)\\ \int drr &\implies \sum_{i=1}^{n} r_iw_i, && (3.4.11)\end{aligned}$$

where r_i and w_i are the meshpoints and corresponding quadrature weights given by a Legendre polynomial. The mespoints and weights are mapped to the appropriate domain by using a tangent mapping on the interval $r_i \in [-1,1]$.

This give the following equation for the matrix elements of the coulomb interaction

$$\langle(B_1B_2)nlNLsSqT_zJ|V_C|(B_1B_2)n'lNLsSqT_zJ\rangle = \frac{q_1q_2}{4\pi\epsilon_0}\sum_{i=1}^{n} r_iw_iR_i^{nl}R_i^{n'l}. \tag{3.4.12}$$

3.4.4 Kinetic energy operator

For the kinetic energy operator in the self-consistent calculation, we will need the expression for the kinetic energy in a harmonic oscillator basis in the laboratory frame. This is a single particle operator and diagonal in all quantum numbers, except n.

From equation E.2.5 we have

$$\langle a|\hat{\mathbf{t}}|c\rangle = \frac{1}{2\mu}\int_0^\infty \mathrm{d}p\,p^4P_{n_al_a}(p)P_{n_cl_c}(p), \tag{3.4.13}$$

where μ is the mass of the particle in question, $P_{nl}(p)$ is the radial harmonic oscillator wavefunction in momentum space from appendix E.1.

We discretize this equation by making the following substitutions

$$\begin{aligned} P_{(n,l)}(k) &\implies (P_1^{(n,l)} \dots P_n^{(n,l)})^T, \qquad P_i^{(n,l)} = P_{(n,l)}(k_i), && (3.4.14)\\ k \in [0,\infty) &\implies k_i \in [k_1 \dots k_n], && (3.4.15)\\ \int dk k^4 &\implies \sum_{i=1}^{n} k_i^4 w_i, && (3.4.16)\end{aligned}$$

where k_i and w_i are the meshpoints and corresponding quadrature weights given by a Legendre polynomial. The mespoints and weights are mapped to the appropriate domain by using a tangent mapping on the interval $r_i \in [-1,1]$.

This gives the following expression for the kinetic energy.

$$\langle a|t|c\rangle = \sum_{i=1}^{n} k_i^2 w_i P_i^{(n_a,l_c)} P_i^{(n_c,l_c)} \frac{k_i^2 \hbar^2}{2\mu}. \qquad (3.4.17)$$

3.4.5 Two-particle momentum operator $\mathbf{k}_i \cdot \mathbf{k}_j$

In the calculation of the single particle energies, the two-particle matrix elements $\langle ab|\mathbf{k}_i \cdot \mathbf{k}_j|cd\rangle$ are needed. The expressions for these are given by [44]

$$\langle ab|\mathbf{k}_i \cdot \mathbf{k}_j|cd\rangle = -\hbar^2(-1)^{j_c+j_a+J} \begin{Bmatrix} j_a & j_b & J \\ j_d & j_c & 1 \end{Bmatrix} \langle a||\nabla_i||c\rangle\langle b||\nabla_j||d\rangle, \qquad (3.4.18)$$

where the reduced matrix elements can be found in [38]. As usual the state a denotes all quantum numbers of a single-particle state.

4 Software

The library and applications that make up the environment for hypernuclear structure calculations presented in this thesis, are based on an existing program package for nuclear structure, named CENS (Compuational Environment for Nuclear Structure)[2]. This package contains code for renormalizing the two-particle interaction in a variety of ways, creating effective many-particle interactions and using these in shell-model calculations. It consists of several different applications, each with its own library, written in Fortran 77/90 and C/C++. A script, written in Python [1], combine the different applications into a common user interface. The compiled Fortran and C/C++ parts will be called the applications, while the Python user interface will be called the interface.

The major goals of this thesis, were to extend CENS to include hyperonic degrees of freedom, redesign it to be a complete library for the Python programming language and use this library to create specialized applications providing the same capabilities as the original application.

The original application could be executed in one of two ways. Either the application was executed directly from the command line, or it was executed as a seperate process from within the Python interface. After the application has been started, no communication is possible between the user(programmer) and the application, other than what would be available when started from the command line.

In the software developed for this thesis, we use a different approach. Instead of running an entire application from within Python, the different Fortran subroutines are made available to Python as extension modules or ordinary library functions. This gives the user total control over what is being done and, depending on the level of detail made available to Python, the applications can be rapidly extended and modified without the need for recompilation. The data is available to the user at every step of the process, making analysis of intermediate data very easy, even for a user with no previous experience with Fortran or C/C++.

Only a subset of the capabilities provided by the orginal application is implemented for this thesis, specified in section 3. The numerical issues are already presented, so this section will focus on the provided library.

The library was intended to be very general, and be used for many different applications. Three distinct applications are presented here, giving an idea of what types of calculation the library enables.

One of the major goals of this thesis, was to extend a high-level language with this library. Since the existing software was written in a mix of Fortran77/90 and C/C++, Python [1] was chosen for this task. Section 4.1 discuss Python and the process of creating extensiton modules for this language.

In section 4.2 the library modules are presented, while section 4.3 presents the applications developed.

4.1 Extending Python

Python, with the Numerical Python (numpy) extension [48], is an open-source, high-level object-oriented programming language, capable of handling contiguous array allocation in both row-major and column-major order, used by C/C++ and Fortran respectively. It can be extended with additional libraries using both modules written in Python and externally compiled object files. The internal workings of the Python interpreter will, however, dictate how these external objects are written and compiled.

The common procedure is to write, or generate using a supplied tool, special interface functions in C/C++, to access the structures of the original library. When implementing a shared library written in Fortran, numpy supplies a tool named *f2py* to generate the interface to Python that would make every public variable, subroutine, module etc. available to Python. As Python is written in C/C++, it is easier to make libraries written in these languages available to Python, so we will rely on hand-written interfaces for these libraries.

In order to understand how extension modules are written, it is necessary to understand how programs written in Python are executed. As Python is an interpreted language, Python programs are not run as normally compiled programs in Fortran or C/C++. An interpreter is executed, which reads the Python source code in a linear fashion, parsing the statements one by one. When started, the interpreter knows only about the built-in structures, representing the core of the Python language. (For details on how the interpreter parses the file, see the Python Reference manual [49]). When encoutering a name that is not recognized as a built-in structure, the interpreter looks it up to see if the name is defined. If not, the execution aborts with an exception.

Before an extension module can be used in a Python script, the interpreter needs to be made aware of the module and the names it provide. This is done by the import statement. When the interpreter finds this statement, it automatically looks for an initialization function defined within the module, if the module exists. There are several demands on this function, but the bottom line is that the interpreter is made aware of what the module provides, especially any callable objects or functions.

Later, when the interpreter encounters a call to one of the functions defined in the module, the parameters are placed in a parameter structure and passed to the function. All C/C++ functions that are callable by the Python interpreter, must have this specific signature, taking the parameter structure as its parameter. The Python implementation ships with a C/C++ header-file where all the Python structures are defined. This is needed to extract the parameters passed to the function.

In order to write an extension module for Python, the compiled object must include the Python header-file, define an initialization function with a specific name and format, providing a list of all functions available in the module. Each of these functions must have a specific signature to be callable from the Python interpreter.

Once this is compiled and linked, the extension module can be used as any other

Python module.

The process of creating a Python module from a Fortran library, is now the same as gaining access to structures written in Fortran from C/C++. All features of the Fortran language needs to be translated to C/C++ specific features, in order for a completely seamless implementation. In most cases, only a small subset of the features need to be implemented.

Creating interface functions for Fortran subroutines and datastructures proved more complex than was originally intended, so major parts of the Fortran library had to be rewritten.

One of the major issues, was that derived types in Fortran 90, was not supported by *f2py*. The original Fortran library relies upon derived types to organize the transformations between the different basis sets. This meant that a large part of the program control needed to be implemented in Fortran and that the transformations between basis sets was not available to Python.

Another issue, solved by creating hand-written interface code in Fortran 90, was that *f2py* didn't support allocatable objects. This meant that Python had to know the size of all arrays in advance and more importantly, all data needed to be allocated on the stack-segment in the Fortran code. This represented a serious issue, as the size of the stack segment is limited by the operating system to usually around 2 Mb. In addition, when full, the application isn't necessarily interupted by a segmentation fault. In the worst case, the application performs completely as expected, exept that the data is corrupted.

When using allocatable arrays, this is no longer a problem. The operating system handles the memory management and the application has access to the entire memory space.

The solution to this problem was by no means elegant and involved a two step process. First the size of the allocated array was made available to Python, before the allocated data was copied to an new Python array. Care was taken not to change the data in the Python applications developed, but no safeguards was implemented to stop an unsuspecting user from doing just that.

A new version of *f2py* is now available and is supposed to handle the complete set of Fortran 90 features. This means that the library can be rewritten with the original design in mind.

4.2 Modules

There are four python extension modules that implement the interface to the Fortran library. These were designed to be independent of any user interface, so that the user can build both a streamlined version for high performance computing or a graphical user interface when ease of use and presentation is more important than fast results.

More often than not, it is not only the final results that are important. It has therefore been a priority to facilitate the extraction of data at any point in the process. In this case we can study the effects of interaction models and renormalization techniques at each intermediate step, to explain what elements are important to the final result.

The four modules are named configuration, interaction, renormalization and effective and they are presented in sections 4.2.1, 4.2.2, 4.2.3 and 4.2.4 respectively. A fifth module, responsible for converting raw data into visual representations, is named visualization and is presented in section 4.2.5.

We only give a brief introduction to each module, as a complete reference can be found in appendix G.

4.2.1 configuration

The configuration module was designed to hold information about the basis sets available. Currently the only configuration implemented is the relative partial wave in the physical basis. This is the form in which most interaction models deliver their matrix elements, so it is a natural place to start. In addition to the physical basis, the isospin basis is also frequently used, but this has not been implemented. Interaction models that come in this basis, are translated to the physical basis inside the class specific for the interaction model.

The main component of the configuration model, is the Channel class. This class holds all information about a specific channel, defined as the set of quantum numbers that make up a separately diagonizable part of the full two-particle Hamiltonian. In the physical basis, this class also holds a SubChannel object, which contains information about the different baryon-baryon couplings that are available in this specific channel. A SubChannel object consists of two Particle objects, as well as convenience methods to extract information about a specific particle combination, such as the effective mass, the total mass etc.

The configuration module also holds a list of Particle objects that are available to the module. A Particle object typically holds all relevant information about a specific particle. The particle information is imported from a separate datafile, where the data has been taken from [7].

The module is designed to be completely general, it does not depend on the specific particles included. This way, when we later want to include more general interaction models and incorporate additional particles and perhaps resonances, this module will need minimal changes.

There are two typical entrypoints to this module. A specific channel may be requested by the methods get_channel or get_channel_by_charge. They each take a set of quantum numbers as arguments, returning an appropriate Channel object. The former method uses isospin projection to specify a channel, while the latter uses the total charge to specify the channel.

The second entrypoint is the static method setup_channels. It takes the minimum and maximum relative angular momentum as arguments, returning a list of all channels in the given range. Such a list is typically needed when transforming from the relative and CoM frame, to the laboratory frame.

4.2.2 interaction

The interaction module is designed to give a unified interface to all interaction models included. Although the possibilty to include transformation to and from different basis sets are present, this is not implemented in a consistent manner. For now, the module only gives matrix elements in a relative, physical partial wave basis in momentum space. The elements are returned in units of MeV^{-2} while the input is a set of momenta in units of fm^{-1} as well as a Channel object.

The different interaction models are implemented in a hierarchical fashion. A specific interaction model is implemented as a separate class, or set of classes, which inherit all the properties of the parent class - IntModel. The parent class is basically present to insure a common programming interface to all the interaction models.

Presently there are three different models implemented - the N3LO [23] model, the NSC97 [25] models and the J04 [24] model, as presented in section 2.2. The N3LO model is purely a nucleon-nucleon(NN) potential and was originally included only as a reference. However, as we will see in section 5.2.2, the NSC97 models do not reproduce the deuteron binding energy. We will therefore use only the N3LO interaction model to create the NN interaction for the production runs.

J04 is purely a hyperon-nucleon(YN) potential and originally comes in an isospin basis. These elements are converted to the physical basis inside the core of the model, before being returned.

The NSC97 model, which actually includes six different models fitting the off-shell behaviour slightly different, provide both NN, YN and hyperon-hyperon(YY) potential up to strangeness $S = -4$ and is the most versatile of the included models.

All of the included interaction models are provided as a separate Fortran 77 file. In order to make the extraction of matrix elements as effecient as possible, interface functions were created in Fortran 90, that collect matrix elements for all momenta for a specific channel into a single two-dimensional array. These interface functions were then compiled using $f2py$, from Numerical Python(see [48]), in order to make them accessible to Python. As Python supports contiguous arrays stored in both row-major and column-major order, no manipulation is needed before handing the array to another function/subroutine written in Fortran. If, however, the elements need to be manipulated by a function written in a language that supports only row-major storage, the array need to be transposed before passed to this function. Efficient methods for converting to and from row-major and column-major order are provided with numpy.

For the interaction models supporting YN or YY interactions, there is an additional complication by the baryon coupled channels. The matrix elements for the different particle combinations are placed in a specific order in the resulting array, which is not necessarily the same order that the combinations are presented in the Channel class. In addition, the interaction models may give matrix elements for more particles in specific channels, than we wish to include. We need a way to select only the appropriate elements. This is especially important for the $S \leq -2$ channels supported by the NSC97 models, that include not only the Λ and Σ, but also the Cascade Ξ. To resolve this, translation arrays are created inside each interaction class, to make sure that the correct matrix elements are always indexed. For convenience, methods for reordering the list of SubChannels in the Channel object to match the ordering of the interaction models are also included.

4.2.3 renormalization

The renormalization module is the largest module of those included. In addition to the actual renormalization procedure, this module also include the code to transform the renormalized interaction from the relative and CoM frame to the laboratory frame as presented in section 3.4.

In regards to the initial design, this module deviates the most. Largely, this is beacuse of limitation with $f2py$ as explained in section 4.1.

When extending the existing renormalization codes for the NN case, the decision was made to keep the existing data structures in the transformation subroutines. When trying to implement the design, we found out that these Fortran data structures could not be made visible to python in any convenient way. Faced with the alternative of rewriting the entire existing renomalization package, we decided to keep the original structures and forego some design criterias in the process.

The renormalization module consists of a parent renormalization class which consists of all the properties and methods shared by the different renormalization procedures. Each procedure is then implemented as a subclass of the renormalization class.

The compiled Fortran library made available through this module, contain not only the renormalization procedure, but also the transformation from relative momentum space to the relative harmonic oscillator space and the transformation from the relative and CoM frame to the laboratory frame in a two-particle three-dimensional harmonic oscillator basis.

The renormalization module reflects this and contain methods both for renormalizing a specific channel and for setting up a range of channels to transform to the laboratory frame.

When a channel is set up in the Fortran specific part of the module, the initialization procedure takes care of setting up the number of integration points, allocating all appropriate array structures. If only renormalization of a single channel is needed, the Hamiltonian for this channel is set up and renormalized using the specified procedure. By design, it is possible to return the renormalized interaction in the basis needed, but as only the V_{low-k} interaction is implemented, only the realtive momentum basis is implemented.

If the object is to do a transformation to the lab system, several channels are needed, and the relative matrix elements needs to be transformed to a relative harmonic oscillator basis. After acquiring a renormalized interaction for all channels in the relative harmonic oscillator basis, the laboratory transformation is set up using the final_vlowk subroutine.

Depending on the model space selected, this procedure can take from minutes to days and is by far the most computationally intensive part of the framework. Since the full many-particle Hamiltonian also consists of the two-particle operator $\mathbf{p}_i \cdot \mathbf{p}_j$, these matrix elements are also calculated in the same basis.

When setting up the interaction in the laboratory frame, the end result is a file of single particle orbits for the selected model space and a datafile of the renormalized two-particle matrix elements.

When setting up an renormalized interaction for a specific channel, the following parameters defines the behaviour.

Renormalization procedure What prodcedure to use to calculate the renormalized interaction.

Channel Which channel should be calculated, meaning the set up quantum numbers making up the channel, if it is a coupled channel and the properties of the particles making up the channel.

Interaction model Which interaction model should be used to create the bare interaction.

Grid Size of the modelspace and complement space. In case of the V_{low-k} procedure, this defines the number of integration points in each space and the cutoffs in each space.

When setting up the renormalized interaction in laboratory space, additional parameters are necessary. The following configuration parameters are recognized. These can be read from file or specified directly as input to the constructor of the renormalization class.

type_of_renormv Renormalization procedure to use. Only the low-momentum type is available as the configuration value: vlowk

coulomb_included Should the coulomb interaction be included in the final effective interaction: (yes/no)

output_run File where the application stores information about the run. The file will be created or overwritten.

renorminteraction_file The file where the matrix elements of the final effective two-body interaction in the lab frame will be stored.

spdata_file File where the single particle orbits in laboratory space will be stored for the selected model space, defined by the four parameters lab_lmax, lab_nmax, hyp_lab_lmax and hyp_lab_nmax to be specified below.

hbar_omega The oscillator parameter to be used for the ho wavefunction.

jmin Start with partial waves with relative total angular momentum jmin.

jmax Include all partial waves with relative total angular momentum up to jmax

type_of_nn_pot Which interaction should be used for the NN interaction.

type_of_yn_pot Which interaction should be used for the YN interaction.

type_of_yy_pot Which interaction should be used for the YY interaction.

include_yn Should channels with Strangeness $S \geq -1$ be included.

include_yy Should channels with Strangeness $S \geq -2$ be included.

lab_lmax Include single particle orbitals for nucleons with orbital momentum up to lab_lmax.

lab_nmax Include single particle orbitals for nucleons with radial number lab_nmax, but only if 2*n + l is not larger than l.

hyp_lab_lmax Include single particle orbitals for hyperons with orbital momentum up to lab_lmax.

hyp_lab_nmax Include single particle orbitals for hyperons with radial quantum number up to hyp_lab_nmax. Unlike the nucleon case we would like to include all orbitals satisfying with n up to hyp_lab_nmax.

n_k1 Number of integration points for the modelspace when using V_{low-k}.

n_k2 Number of integration points for the complement space when using V_{low-k}.

k_cutoff Momentum cutoff for the modelspace when using V_{low-k}.

k_max Momentum cutoff for the complementspace when using V_{low-k}.

4.2.4 effective

This module implements the procedure specified in section 3.3. The primary purpose of the effective module is to create an effective many-particle interaction using a perturbative approach. In this thesis however, we stop at creating a self-consistent basis of three-dimensional harmonic oscillator wavefunctions at the Hartree-Fock level.

For the moment, it is possible to calculate a core of filled orbitals, with a Λ or Σ in an orbit around this core. Later we will look at the energies calculated for these orbitals and derive their interaction model and parameter dependence.

As input, this module takes a single-particle data file as supplied by the renormalization module. The orbitals in this file are modified to represent the nucleus we wish to do calculations for. We supply also a tool for choosing the nucleus, so this file does not have to be changed manually.

The following parameters are needed for a self-consistent calculation.

order_of_interaction To what order is the self-consistent potential to be calculated: first, second

output_run File where the application stores information about the run. The file will be created or overwritten.

renorminteraction_file File where the renormalized interaction elements are stored.

spdata_file File where the single-particle orbits are read.

HFrenorminteraction_file File where the new interaction elements are stored after the self-consistent calculation.

HFspdata_file File where the new single-particle orbitals are stored after the self-consistent procedure.

4.2.5 visualization

The visualization module consiste of methods to generate different types of plots. Current capabilities include the plotting of the diagonal elements of a matrix, off-diagonal elements constrained by a single momenta and an eigenvalue plot. These functions can handle several datasets with separate legends.

The plots can be both displayed on screen and saved to file in postscript format.

4.3 Application

To show the flexibility of the python extension modules created, three applications have been created. The first two are barely worth mentioning, as the resulting Python scripts only consists of a couple of lines. The first script, set up the renormalized matrix elements in the laboratory frame, by reading a configuration file and passing the options to the renormalization class.

The entire python script is written

```
filename = 'renorm.ini'
r = selector(filename)
r.transform_lab()
```

The second application calculates the self-consistent basis and is similarily simple

```
filename = 'bhf.ini'
eff = Effective(filename)
eff.setup_hf_orbits()
```

A third application was created to show how the data could be extracted at intermediate steps in the calculation process. This is a graphical application to study the bare and renormalized interaction in a relative partial wave basis.

Figure 4.1 shows a screenshow of the full application.

The interaction models to study are set by pressing the appropriate checkboxes in figure 4.2. There is no limitations on the number of models to include in each plot.

The baryon-baryon configuration to study is selected by pressing the appropriate radiobutton in figure 4.3.

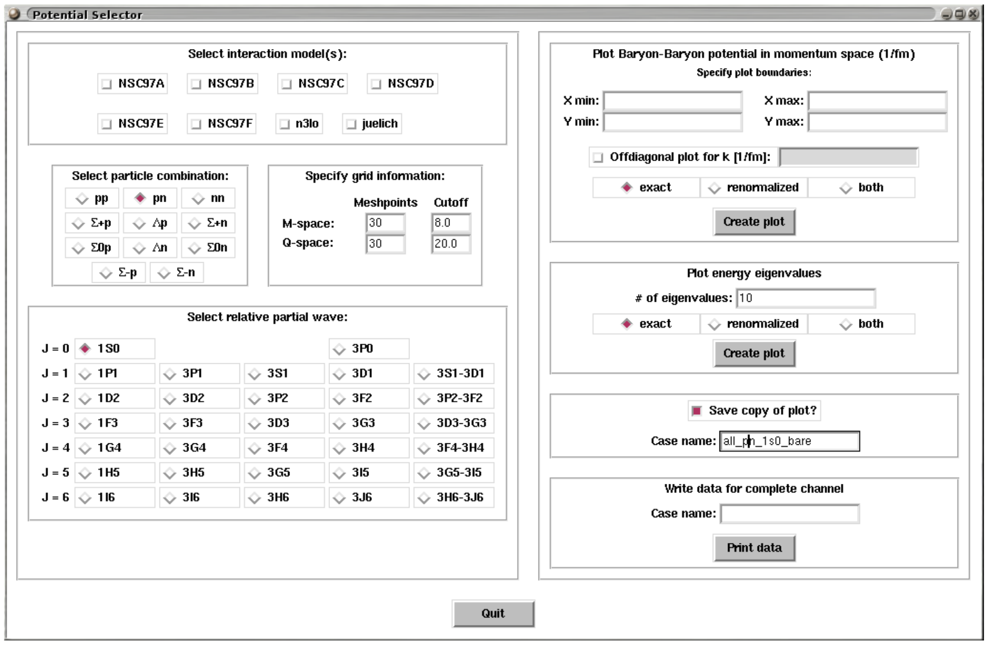
As the V_{low-k} procedure is the only renormalization procedure implemented, there is no option to select this. The size of the low momentum model space is, however, specified in the entry fields in figure 4.4.

The partial wave to plot is chosen by pressing the appropriate radiobutton in figure 4.5.

In figure 4.6 the plot boundaries are specified, together with the appropriate row to plot if an off-diagonal plot is needed. The options also include plotting a bare or renormalized potential or both in the same figure.

In figure 4.7 the eigenvalue plot is selected. The user must choose whether to plot the energy eigenvalues using the bare or renormalized interaction when setting up the Hamiltonian. Also, the number of eigenvalues to plot is specified.

In figure 4.8 and 4.9, the user may choose to save a copy of the plot to a file



Select particle combination: [✓ pn]

Specify grid information:

	Meshpoints	Cutoff
M-space:	30	8.0
Q-space:	30	20.0

J = 0 [✓ 1S0]

J = 1, J = 2, J = 3, J = 4, J = 5, J = 6

Plot Baryon-Baryon potential: [✓ exact]

# of eigenvalues: 10

Plot energy eigenvalues: [✓ exact]

[✓ Save copy of plot?]

Case name: all_pn_1s0_bare

Figure 4.1: Full view of application.

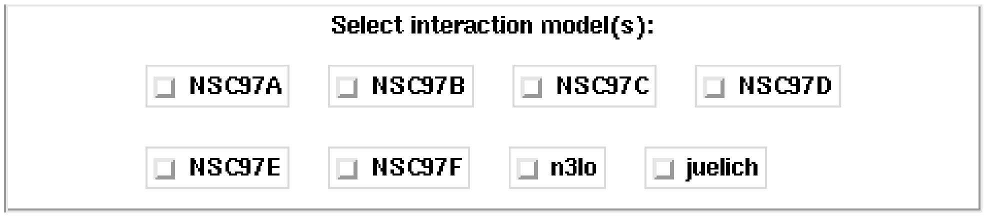
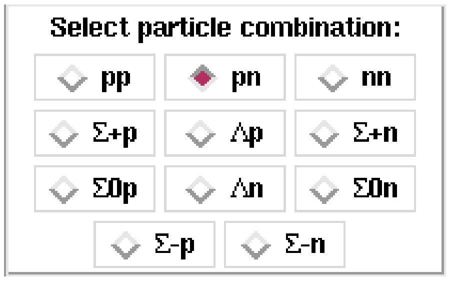
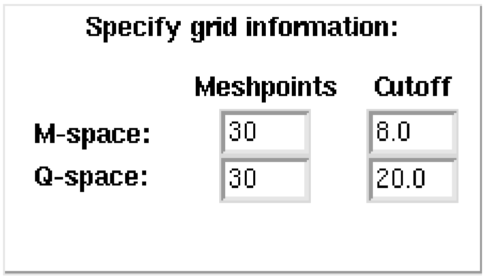


Figure 4.2: Selecting the interaction model.



[✓ pn]

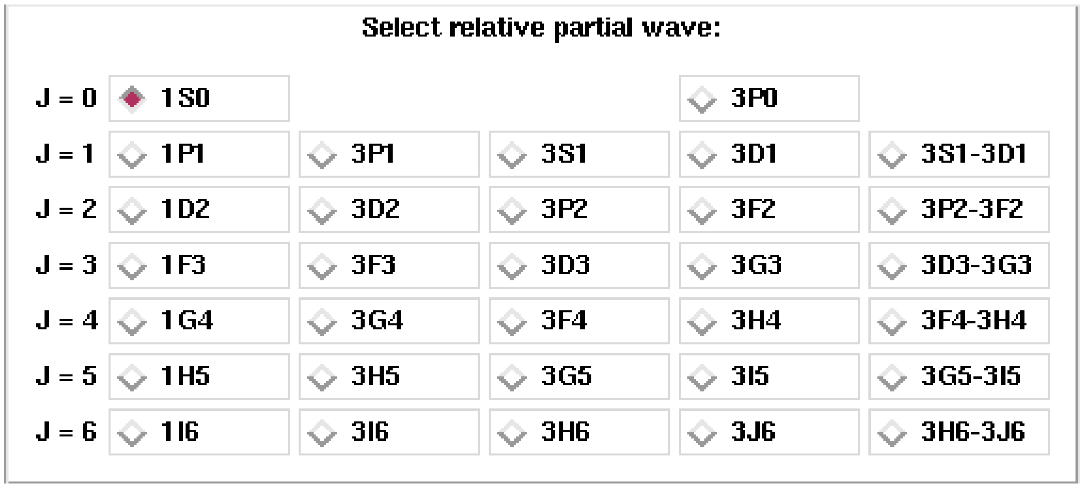
Figure 4.3: Selecting the Baryon Baryon channel



Specify grid information:

	Meshpoints	Cutoff
M-space:	30	8.0
Q-space:	30	20.0

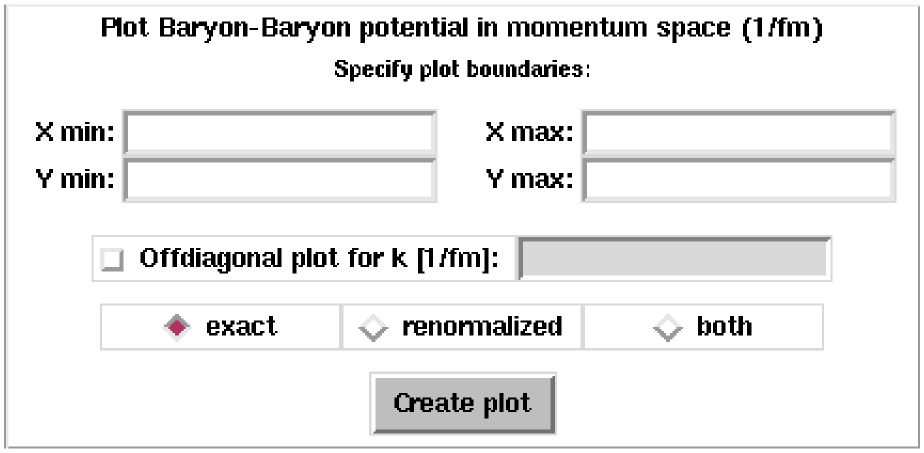
Figure 4.4: Specifying grid information for the model space and the complement space.



Select relative partial wave:

J = 0	[✓ 1S0]			3P0	
J = 1	1P1	3P1	3S1	3D1	3S1-3D1
J = 2	1D2	3D2	3P2	3F2	3P2-3F2
J = 3	1F3	3F3	3D3	3G3	3D3-3G3
J = 4	1G4	3G4	3F4	3H4	3F4-3H4
J = 5	1H5	3H5	3G5	3I5	3G5-3I5
J = 6	1I6	3I6	3H6	3J6	3H6-3J6

Figure 4.5: Selecting the partial wave.



Plot Baryon-Baryon potential in momentum space (1/fm)

Specify plot boundaries:

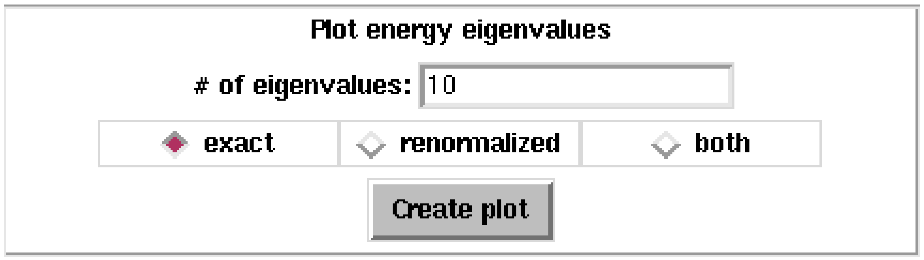
X min: X max:

Y min: Y max:

Offdiagonal plot for k [1/fm]:

[✓ exact] renormalized both

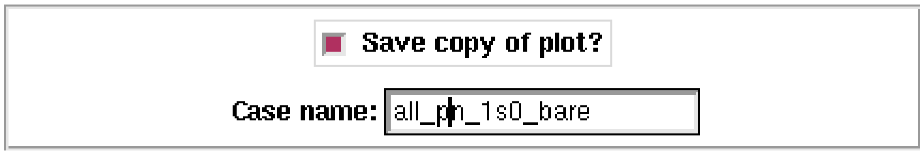
Figure 4.6: Specifying information for plotting the potential



# of eigenvalues: 10

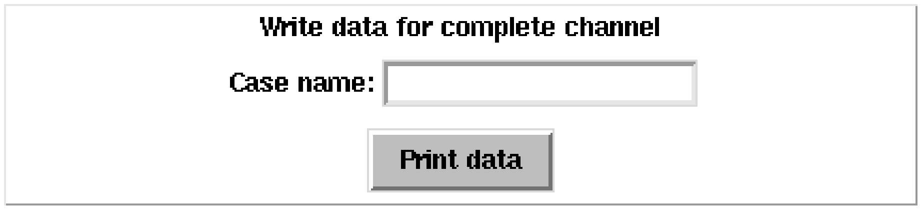
[✓ exact] renormalized both

Figure 4.7: Specifying information for plotting the eigenvalues.


[✓ Save copy of plot?]

Case name: all_pn_1s0_bare

Figure 4.8: Save a copy of the plot to file.



Case name:

Figure 4.9: Save all data for entire coupled channel.

and save the data used to generate the plot, respectively.

The graphical elements are programmed using the Tkinter module.

5 Results

5.1 Setup

Our goal in this thesis, is to study how the energy of the single-particle orbital $0s_{1/2}$ for a Λ hyperon behaves as a function of the oscillator parameter $\beta = \hbar\omega$ and of the V_{low-k} momentum cutoff λ. This will be studied for a range of different hypernuclei, all with filled proton and neutron shells.

The momentum cutoff λ will range from $2.0 - 3.0$, while β will range from $5.0 - 25.0$. We will sample 5 times for β and 3 times for λ. This will generate 15 variations for each interaction model we wish to investigate.

There are a couple of parameters we wish to fix from the start. We want the renormalized interaction to be independent of the number of meshpoints selected, as well as the maximum momenta in the integral to infinity involved in the V_{low-k} procedure.

The NSC97 models come in six different version but we will, however, only choose two of these versions in our production runs.

We will also have to select the proper cutoff for the harmonic oscillator space, to decide how many matrix elements we wish to generate. We will look at how the results vary as a function of the size of this model space, with samples from $N = 2n + l \in 3, 5, 7, 9$ for the nucleon shells.

Because there are differences in which orbitals are dominant for the nucleons and hyperons, the harmonic oscillator space is chosen differently for the two types of particles. As the hyperons are suspected to be loosely bound, we only investigate the effects of low lying orbits with zero orbital momentum. We will however let the radial quantum number n run from 0 through 3.

In the library, there is currently no option to include only a single type of hyperon. Matrix elements for combinations of all 4 kinds of hyperons are always generated. The number of elements and the time needed to calculate them, increase rapidly with the number of available hyperon orbits. Limitations on time and resources has been a factor when choosing such a small hyperon model space.

5.1.1 Grid size

The V_{low-k} renormalization procedure, as well as the transformation from a basis in relative momentum wavefunctions, to relative harmonic oscillator wavefunctions, both requires an integration over momentum k with $k \in [0, \infty]$. Numerically, this would normally be accomplished by selecting the meshpoints by a tangent mapping in the domain. When doing V_{low-k}, we will be projecting the problem to a smaller space defined by $k \in [0, \lambda]$, where λ ranges from 2 fm^{-1} to 3 fm^{-1} and we will therefore need a certain amount of meshpoints inside this model space. A tangent mapping will generate too few points in the model space, with too many in the excluded space.

By exploiting that the interaction models are only valid on a limited domain, we can introduce a cutoff k_{max} in the complement space and integrate over the domain $k \in [0, k_{max}]$ instead of $k \in [0, \infty)$. We would have to select k_{max} large enough, so that the results converge and are independent of this choice of cutoff.

We can then choose a set of integration points inside the model space, and another set of integration points in the complement space. The number of meshpoints will be selected so that the results are independent of the number of meshpoints.

Figure 5.1 and 5.2 shows the dependence of the lowest energy eigenvalue on the number of meshpoints in the tensor coupled channel ${}^3S_1$–${}^3D_1$, for the pn and Λp cases respectively. Note that the number of meshpoints indicated is for an uncoupled channel, so that the dimension of the problem in a tensor coupled channel is twice the number of meshpoints. The latter channel is also a baryon coupled channel with three subchannels, so the dimension of the Hamiltonian is six times the number of meshpoints for this problem.

Both these figures show relatively rapid convergence with the number of meshpoints. For the production runs, we will use 60 meshpoints, where 30 meshpoints are located in the model space and 30 are located in the complement space.

This choice of meshpoints also ensures that the norm is one to six significant digits when normalizing the overlap between the relative momentum wavefunction and the harmonic oscillator wavefunction. This is more than acceptable.

Figure 5.3 and 5.4 shows the eigenvalues of the ${}^3S_1$–${}^3D_1$ tensor coupled channel in the pn case and the baryon coupled Λp case as a function of the momentum cutoff of the complement space, k_{max}. It is clear that a reasonable cutoff can be chosen to be $k_{max} = 20$ fm^{-1} to be sure that the results are independent on the choice of cutoff.

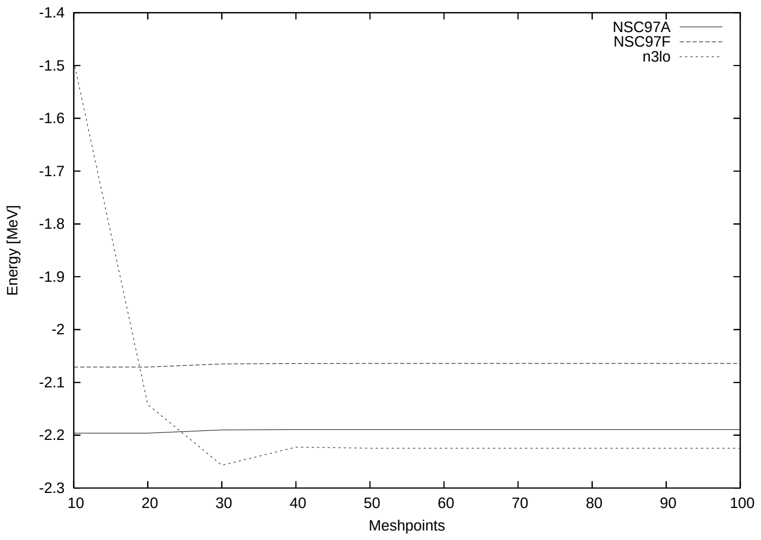


Figure 5.1: Lowest energy eigenvalue for the pn $^3S_1$–$^3D_1$ channel as a function of the number of meshpoints.

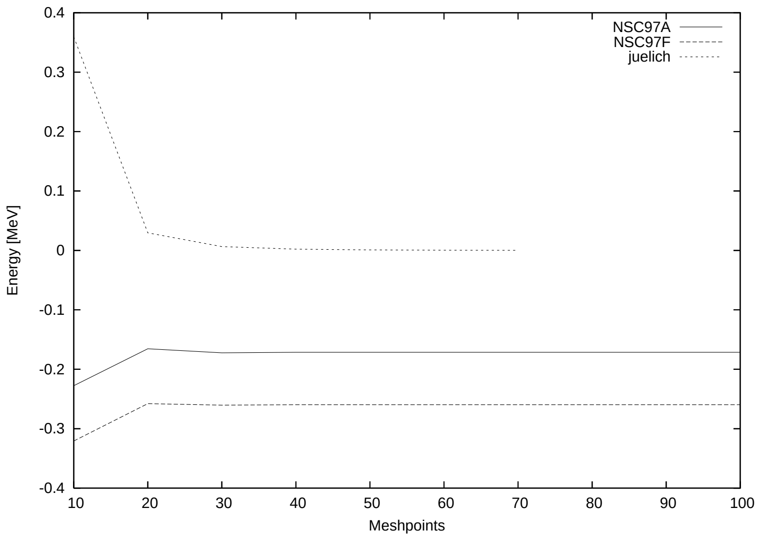


Figure 5.2: Lowest energy eigenvalue for the Λp $^3S_1$–$^3D_1$ channel as a function of the number of meshpoints. Note that the implementation of the J04 interaction doesn't support more than 70 meshpoints.

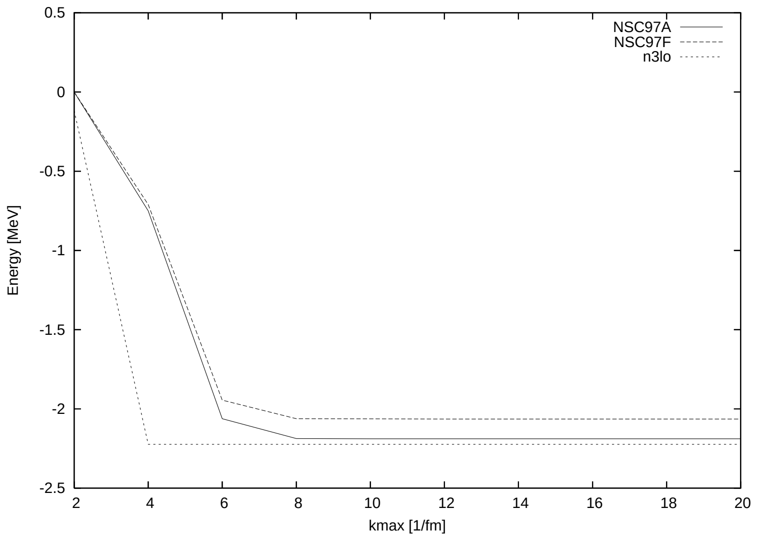


Figure 5.3: Lowest energy eigenvalue for the pn $3S_1$–$^3D_1$ channel as a function of the maximum momenta.

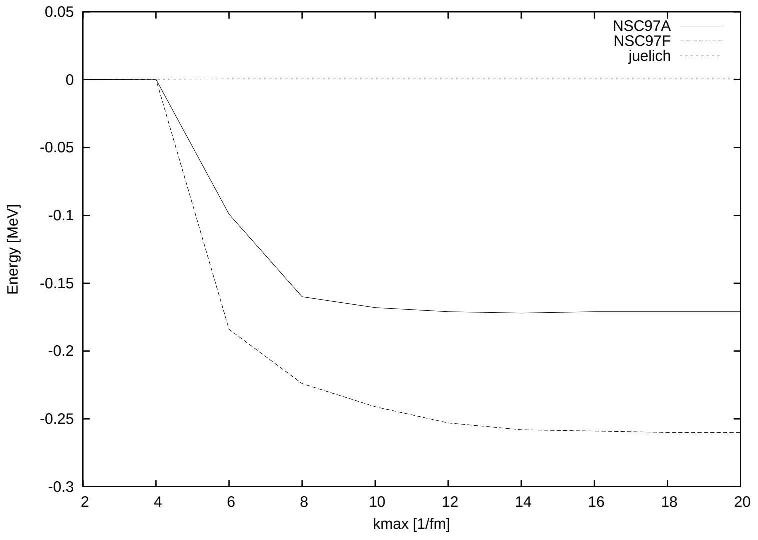


Figure 5.4: Lowest energy eigenvalue for the Λp $^3S_1$–$^3D_1$ channel as a function of the maximum momenta.

5.1.2 NSC97 models

We will calculate matrix elements for 15 variations for each interaction model included in the calculation. Each variation will generate approximately 1Gb of data, giving 15Gb of data for each interaction model. With limited time and resources, we select to use only three different hyperon-nucleon models. Of these three, one is the J04 model [24], while two are available for the NSC97 models [25].

Figure 5.5 shows the diagonal elements for both the bare interaction and the renormalized interaction for the $\Sigma^+ p\ ^1S_0$ channel, as well as the eigenvalues obtained when diagonalizing the Hamiltonian. It is an uncoupled channel and as suspected, there is not much difference between the different models.

For the coupled channels, we expect the results to differ more. Figure 5.6, 5.7 and 5.8 shows the same plots for a tensor coupled channel, a baryon coupled channel and finally a tensor and baryon coupled channel. For the tensor coupled channel, the interaction elements are different, but the energy eigenvalues are the same for all models. In the baryon coupled case, even the eigenvalues are different.

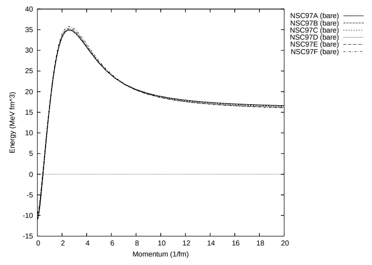
We will first focus on the NSC97F model. The plots show that the diagonal elements generated using this model is in one end of the range for all three plots. The same is true when we look at the single particle energies for the different hypernuclei in table 5.1, where the NSC97F model gives the least amount of binding for each hypernucleus.

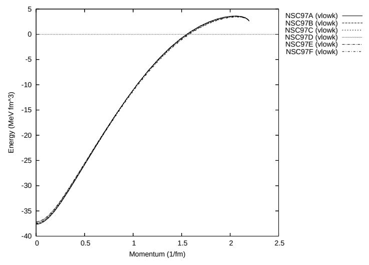
As to the second model to include, the choice is not so obvious. There is no clear candidate in the other end of the range, as we would have hoped, but NSC97A seems to be a good representative. NSC97A, NSC97F and J04 will be the interaction models studied.

A troubling aspect, is that the different models generate so widely different results for the single particle energies. We will not discuss this yet, as the results could depend on our choice of model space and oscillator parameter, which in this case is $\beta = 10.0$ and $\lambda = 2.5$ with 60 meshpoints. Unfortunately it turns out that this effect is independent of both the model space and variational parameters.

Hypernuclei	NSC97A	NSC97B	NSC97C	NSC97D	NSC97E	NSC97F
${}^{17}_{\Lambda}$O	-17.6048	-17.9954	-19.1783	-19.3929	-18.9891	-16.6459
${}^{41}_{\Lambda}$Ca	-33.0701	-33.1851	-34.6602	-34.0715	-32.7319	-27.7015
${}^{91}_{\Lambda}$Zr	-49.6889	-49.1103	-50.5352	-48.3865	-45.5555	-36.8208
${}^{133}_{\Lambda}$Sn	-54.0225	-53.2739	-54.6748	-52.1088	-48.8951	-39.2667
${}^{209}_{\Lambda}$Pb	-61.1422	-59.8580	-61.0152	-57.3759	-53.2460	-41.5609

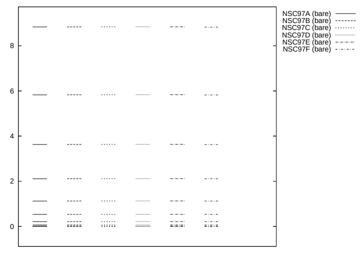
Table 5.1: Single particle energies for Λ in the $0s_{1/2}$ orbital for different hypernuclei.

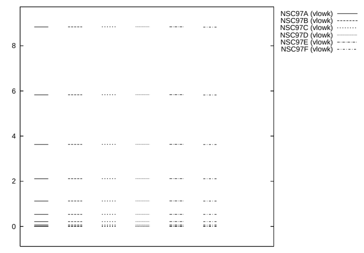




(a) Diagonal elements of the bare potential

(b) Diagonal elements of the renormalized potential

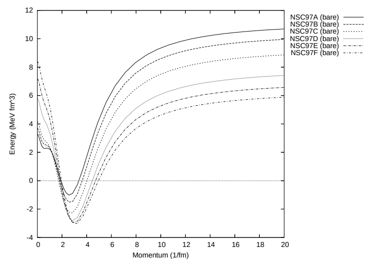


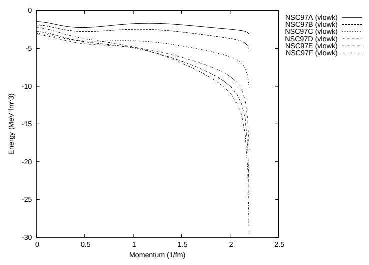


(c) Eigenvalues obtained with the bare potential

(d) Eigenvalues obtained with the renormalized potential

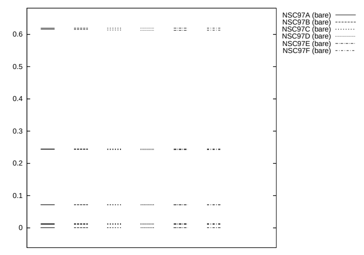
Figure 5.5: Potential and eigenvalues for the Σ^+p $^1S_0$ channel.

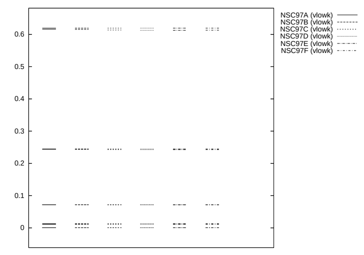




(a) Diagonal elements of the bare potential

(b) Diagonal elements of the renormalized potential

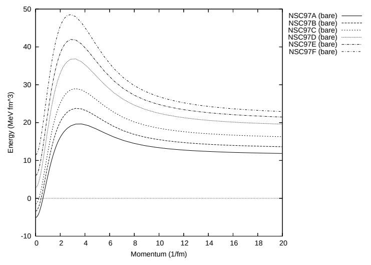




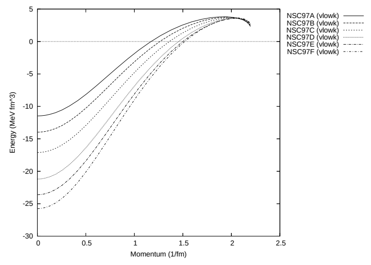
(c) Eigenvalues obtained with the bare potential

(d) Eigenvalues obtained with the renormalized potential

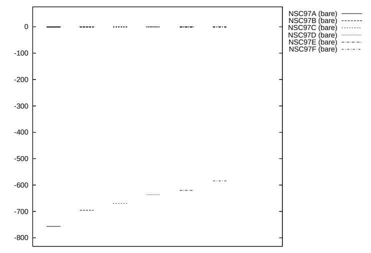
Figure 5.6: Potential and eigenvalues for the Σ^+p $^3S_1$–$^3D_1$ channel. The potential is plotted for the $^3S_1$ block matrix only.



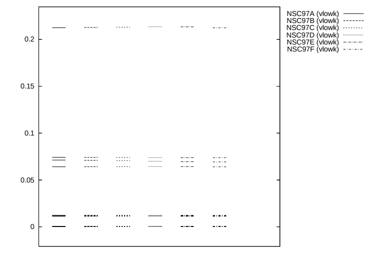
(a) Diagonal elements of the bare potential



(b) Diagonal elements of the renormalized potential

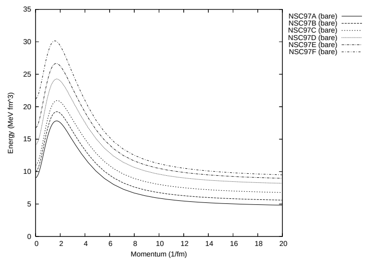


(c) Eigenvalues obtained with the bare potential

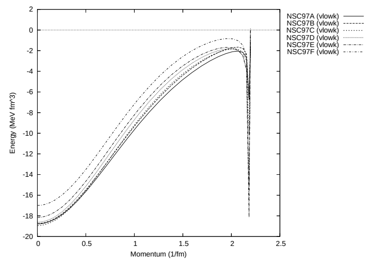


(d) Eigenvalues obtained with the renormalized potential

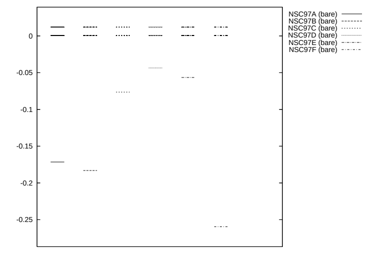
Figure 5.7: Potential and eigenvalues for the $\Lambda p\ {}^1S_0$ channel.



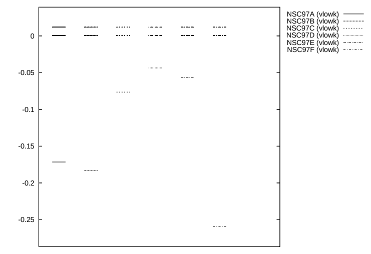
(a) Diagonal elements of the bare potential



(b) Diagonal elements of the renormalized potential



(c) Eigenvalues obtained with the bare potential



(d) Eigenvalues obtained with the renormalized potential

Figure 5.8: Potential and eigenvalues for the $\Lambda p\ {}^3S_1$–${}^3D_1$ channel. The potential is plotted for the $\Lambda p\ {}^3S_1$ block matrix only.

5.2 Test cases

5.2.1 Energy eigenvalues

A relatively easy test of the numerical implementation of V_{low-k}, is to see if the eigenvalues of the selected eigenstates are reproduced after the renormalization procedure. We select the eigenstate with the largest components in the model space, as discussed in section 3.2.1. We show this for two cases. The first one is the pn case, which has a bound state in the ${}^3S_1$–${}^3D_1$ tensor coupled channel.

Table 5.2 shows the original eigenvalues and the reproduced eigenvalues sorted by the corresponding eigenvectors component in the model space. It shows quite clearly that the correct eigenvalues are reproduced. It also show that the bound state has a significant contribution from high momenta states. As this is not relevant to this thesis, we will not discuss this further.

Table 5.3 shows the same results for the baryon coupled Λp ${}^1S_0$ channel, but here the bound state is excluded from the modelspace. We see that this bound state has a large contribution from the complement space and only around 19% from the model space. This is still a significant contribution, implying that the model space is too small. Increasing λ will include this state in the model space.

An interesting point to consider is whether it is appropriate to use different model space cutoffs for the nucleon channels and the hyperon channel. This will be left for future work.

However, we see that the correct eigenvalues are reproduced after the renormalization procedure and conclude that this is done correctly.

Eigenvalue	Reproduced eigenvalue	Component in modelspace
3.416601064070259e-02	3.41660106e-02	0.999999999992494
0.913903152314748	0.913903152	0.999999753686008
5.18630750445118	5.18630750	0.999974849789549
4.516991861981452e-02	4.51699186e-02	0.999958631124156
16.4629477952448	16.4629478	0.999758927293190
37.7695969033269	37.7695969	0.999108809044036
69.3632452801311	69.3632453	0.997653026322953
1.67204257152620	1.67204257	0.997582563473404
195.194304801923	195.194305	0.994676613576324
107.812877403660	107.812877	0.994535352050002
9.76155442131280	9.76155442	0.993109180996088
28.4094550039271	28.4094550	0.989205524579391
173.255007035140	173.255007	0.988438865803158
146.799723753018	146.799724	0.988372255392941
58.6794630832711	58.6794631	0.985687987040545
138.349146335824	138.349146	0.984494304532942
-2.19710759140494	-2.19710759	0.984457926217859
97.4994811462420	97.4994811	0.983668484782110
178.935732589212	178.935733	0.979172364286131
197.909623323750	197.909623	0.975195198852643
271.883233439369		7.102835286156098e-002
596.099991172277		4.854211803254944e-002
1492.54639803869		1.805410080982251e-002
247.718642318589		1.047373856972548e-002
3474.87162141087		6.363343100272794e-003
559.658434811425		5.088947011829431e-003
1564.65534547262		3.818912188671142e-003
6614.59502627771		1.185214809276037e-003
10401.2967031056		1.815889093970478e-004
7121.68371265130		7.510327242621864e-005
3735.20336991250		5.425637132750700e-005
13783.0881807160		3.469573388278385e-005
10970.2187921564		2.180550537412961e-005
20164.6648712114		1.052273828274194e-005
15931.5067321025		7.901703897141228e-006
14221.9419099527		5.285175464116271e-006
16166.6721877254		1.605607634895445e-006
21063.4672720451		1.326855932755532e-006
34727.7534440951		5.451528507677166e-007
35663.7515013417		1.675389847408723e-008

Table 5.2: Eigenvalues for pn in ${}^3S_1$–${}^3D_1$

Eigenvalue	Reproduced eigenvalue	Component in modelspace
3.268651738859335e-02	3.26865174e-02	0.999999698416531
2.923637269837017e-02	2.92363727e-02	0.999998156522386
2.367334246202904e-02	2.36733425	0.999985258477301
0.942226667923387	0.942226668	0.999981379442691
0.731771165862715	0.731771166	0.999909601583194
5.53410662992936	5.53410663	0.999815158762751
0.534697917194415	0.534697917	0.999579427757253
4.11705074370058	4.11705074	0.999451113944257
17.4391960392881	17.4391960	0.999113356242823
3.23935374704335	3.23935375	0.998525218000787
13.4268335487878	13.4268335	0.998459859370169
11.6039468196882	11.6039468	0.997160991467505
39.2031242675101	39.2031243	0.997097568295071
31.8611897324750	31.8611897	0.996914621528358
29.2031930562889	29.2031931	0.995349116099256
60.3037585720680	60.3037586	0.994658865274222
57.1011197117443	57.1011197	0.992685677360692
70.3989437522830	70.3989438	0.992253881071362
95.8942702495549	95.8942702	0.991344972135024
92.3621046354053	92.3621046	0.988635948332279
132.571143429431	132.571143	0.986442851286620
128.708303112619	128.708303	0.982663006480691
106.761328765209	106.761329	0.981597878473652
163.011430480446	163.011430	0.978891471256576
176.445063550510	176.445064	0.975582557725821
158.741971793430	158.741972	0.975515622588271
181.135332038870	181.135332	0.963863220556515
141.514315796328	141.514316	0.960809840060427
167.959290986953	167.959291	0.938036674767626
185.443052743800	185.443053	0.720107156802245
318.876718966845		0.191577989416503
-755.418950911826		0.187529153367256
780.831303883140		5.035996804574255e-002
616.229915880355		3.116679806819013e-002
244.825262568516		3.010173830215701e-002
247.094266213475		2.649039023687568e-002
1725.56053398083		2.301748522367679e-002
585.895549337287		1.912477185283220e-002
1587.30022947448		1.356064536574410e-002
3893.91398477343		7.959289370681364e-003
3589.69723185945		6.100199796512800e-003
1849.61252135105		3.935172656286296e-003
6985.70016626551		1.664626177825600e-003
⋮		⋮

Table 5.3: Eigenvalues for $\Lambda p\ ^1S_0$

5.2.2 The deuteron

The deuteron is a two-particle system consisting of a proton and a neutron. This is the only experimentally observed bound two-particle state and presents an ideal test case for the interaction models. The deuteron is a result of a bound state in the ${}^3S_1$–${}^3D_1$ tensor coupled channel with binding energy $E_B = 2.2246$ MeV (withour relativistic corrections. Our NN interaction models should be able to reproduce this bound state. Figure 5.9 shows the result of diagonalizing the Hamiltonian in the tensor coupled ${}^3S_1$–${}^3D_1$ channel for the interaction models NSC97A, NSC97F and N3LO.

The most striking observation is that the NSC97 models do not reproduce the energy of the deuteron bound state. This result is converged and does not change when the number of meshpoints are increased or when the cutoff k_{max} is increased. We will therefore not use the NSC97 models for the NN interaction, only the N3LO model [23]. The NSC97A and NSC97F models will be used only for the YN and YY interactions.

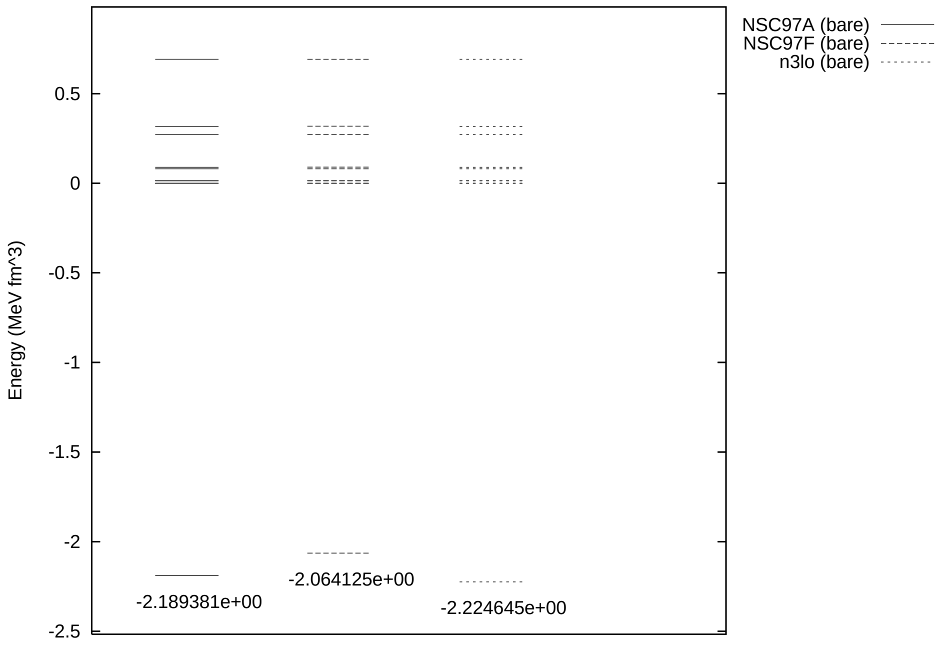


Figure 5.9: Energy eigenvalues for the deuteron in the tensor coupled ${}^3S_1$–${}^3D_1$ for NSC97A, NSC97F and N3LO.

Figures 5.10, 5.11 and 5.12 show the diagonal matrix elements of the ${}^3S_1$ block, the ${}^3D_1$ block and the ${}^3S_1$–${}^3D_1$ block respectively. The first point to notice, is that the N3LO model is only valid for low momenta. Above $k \approx 4\text{fm}^{-1}$, all elements are zero.

The NSC97 models show no such cutoff and are available for high momenta as well. Although the two models are fitted to the same scattering data, their treatment of the different transitions are very different. The N3LO model has more attraction in the $^3S_1$ and $^3D_1$ diagonal elements, with more repulsion in the off-diagonal $^3S_1$–$^3D_1$ elements.

These interaction models show completely different behaviours, also in the uncoupled channels. But since they are fitted to the same scattering data, the energy eigenvalues in these channels are still identical. For all other tensor coupled channels, the results are also the same, but in the $^3S_1$–$^3D_1$ channel, which is the most important for the deuteron, the NSC97 NN models are not accurate enough.

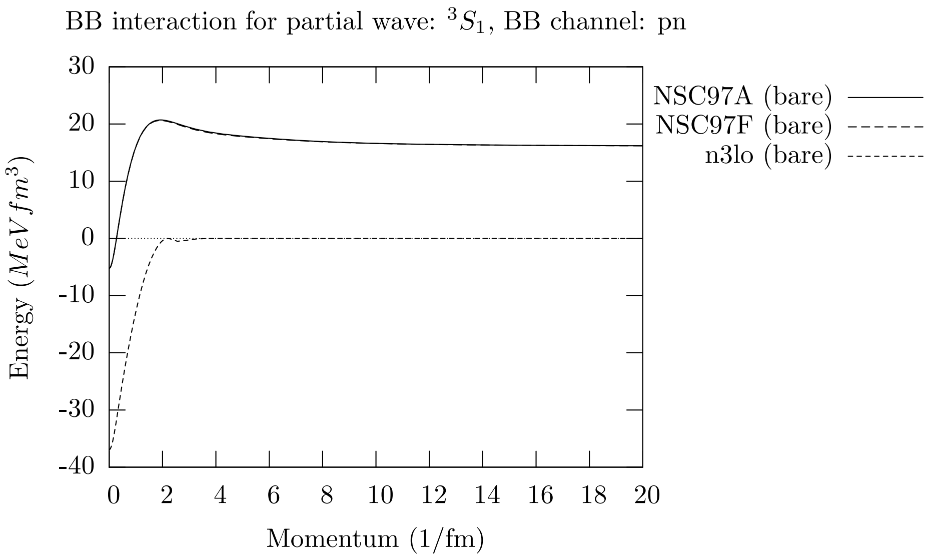


Figure 5.10: Diagonal elements of the $^3S_1$ interaction for the deuteron.

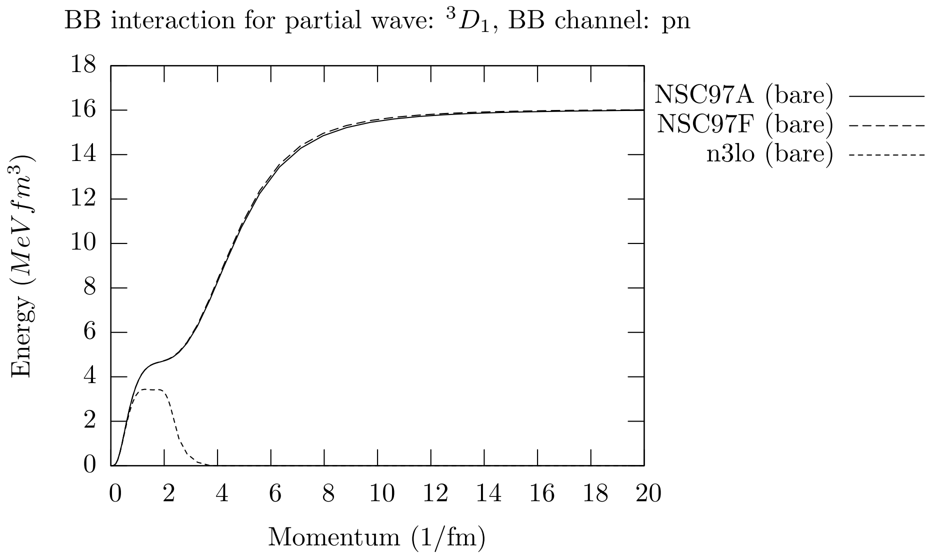


Figure 5.11: Diagonal elements of the $^3D_1$ interaction for the deuteron.

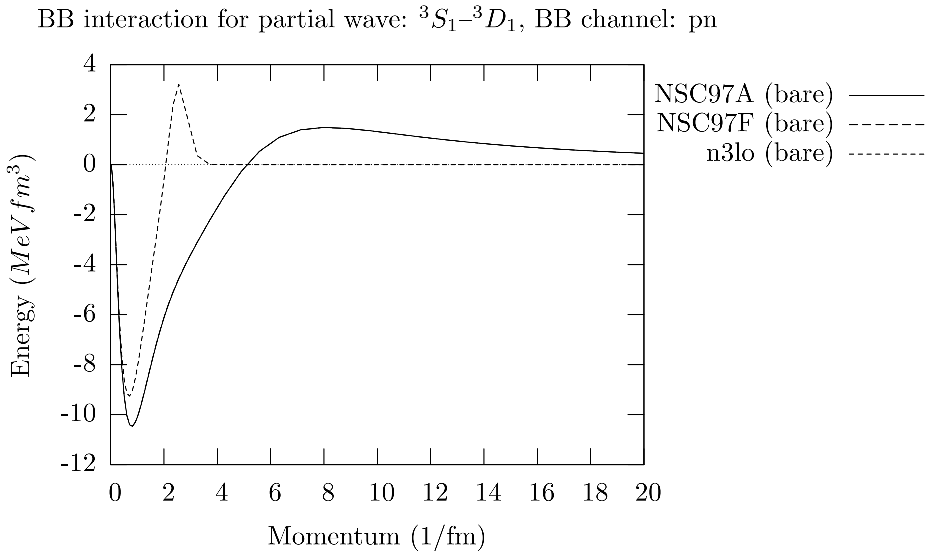


Figure 5.12: Diagonal elements of the $^3S_1$–$^3D_1$ interaction for the deuteron.

5.3 Nucleon shells

We investigate how the energy of the Λ $0s_{1/2}$ single particle orbital behave for different sized harmonic oscillator spaces for nucleons. The renormalized interaction is set up in a basis of relative momenta k with $k \in [0, \lambda]$. When the matrix elements are transformed to the relative harmonic oscillator basis and later to the laboratory frame, we need to define the size of the harmonic oscillator space we wish to use. This needs to be a compromise between calculational speed and the need for exact results. In theory, we would need to include harmonic oscillator shells to infinite order for the renomalized interaction to be equivalent in both the momentum space and the harmonic oscillator space. We will, however, only include nucleon harmonic oscillator orbitals where the relation $2n+l \leq N$ for $N \in 3, 5, 7, 9$. For hyperons, the cutoff is defined sligthly different. We will include all hyperon orbitals where $n = 0, 1, 2, 3$ and $l = 0$. Since we are only looking at the Λ in the $0s_{1/2}$ orbital, we will not need elements with higher orbital momentum.

For ${}^{17}_{\Lambda}$O we see in figure 5.13 that the minimum result is practically converged for all models, when we include nucleon orbits up to $N = 7$. Not much is gained by increasing the size of the modelspace to $N = 9$, which increases the runtime of each variation eightfold.

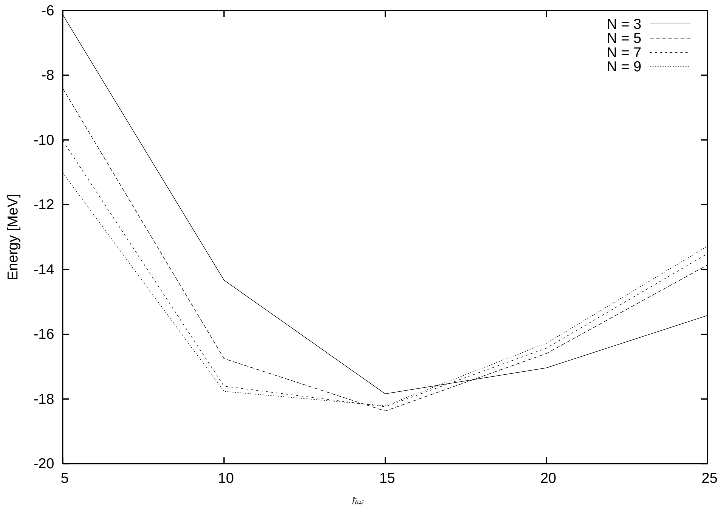
For the larger nuclei in figure 5.14 and 5.15, showing results for ${}^{41}_{\Lambda}$Ca and ${}^{91}_{\Lambda}$Zr respectively, we still see that at the minimum, the results converge for larger N. The J04 interaction, however, seems to converge more slowly than the NSC97 models. This is especially clear in 5.15 c). A possible explanation is that the relative partial waves with high total angular momentum contributes more to the renormalized matrix elements for the J04 model than for the NSC97 models. It can also mean that the matrix elements for transitions between higher orbits are larger for the J04 model than for the NSC97 models. As these elements are weighted with the factor $2j+1$ in the calculation of the single particle potential, small uncertainties in these elements will be magnified, making the final results very sensitive to these matrix elements.

As the space is increased, we see also that the minima are moved to lower values of the oscillator parameter β and that the results calculated with lower values of β depend more on the size of the model space than results calculated with higher values of β.

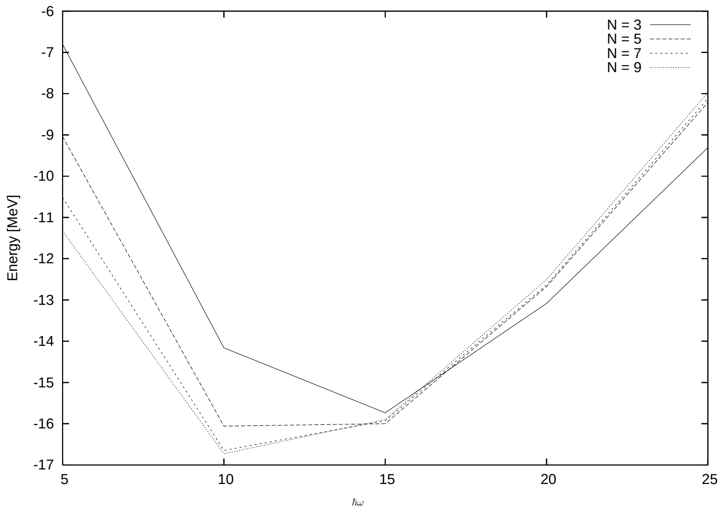
For nuclei larger than ${}^{41}_{\Lambda}$Ca, we need at least $N = 9$ in order to fined converged results.

The most important point, is again that the results are very different for the different interaction models. This will be made even clearer in the next section, when we discuss how the energy depends on the variational parameters.

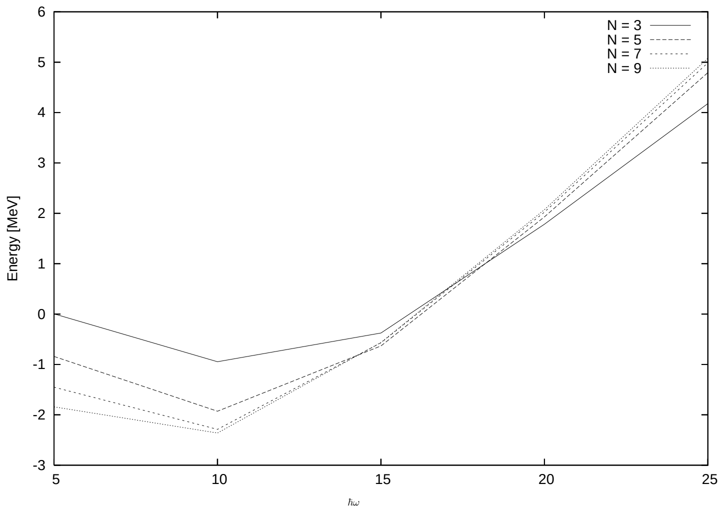
Table 5.4, 5.5 and 5.6 tabulate the energies used in the previously discussed plots.



(a) NSC97A

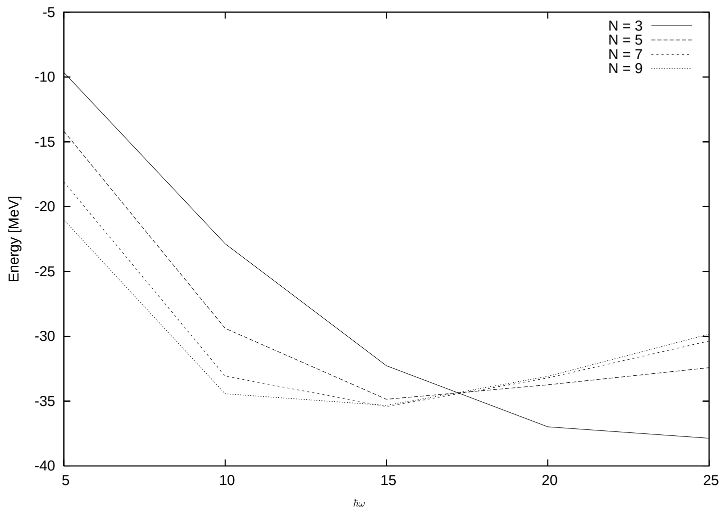


(b) NSC97F

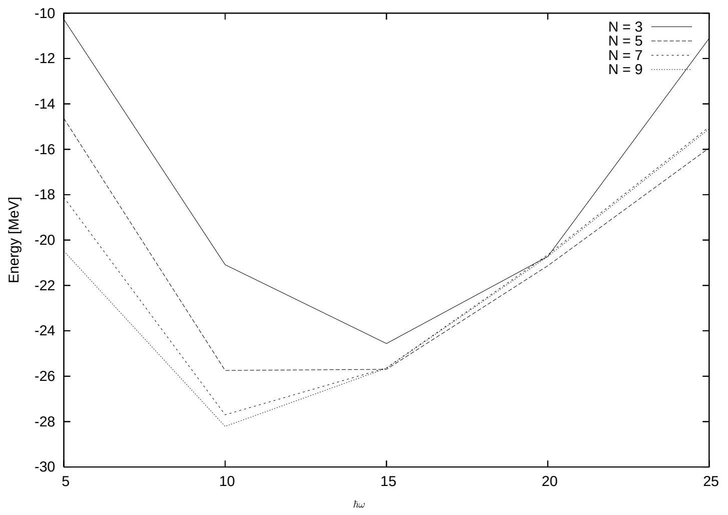


(c) J04

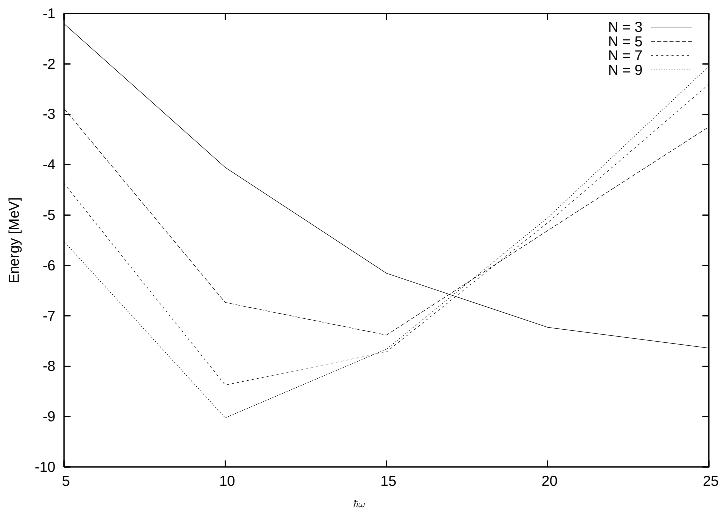
Figure 5.13: Single particle energies for Λ in the $0s_{1/2}$ orbital in ${}^{17}_{\Lambda}$O as a function of the oscillator parameter for different sized model space. N stands for the maximum $2n + l$ in the modelspace for the nucleons.



(a) NSC97A

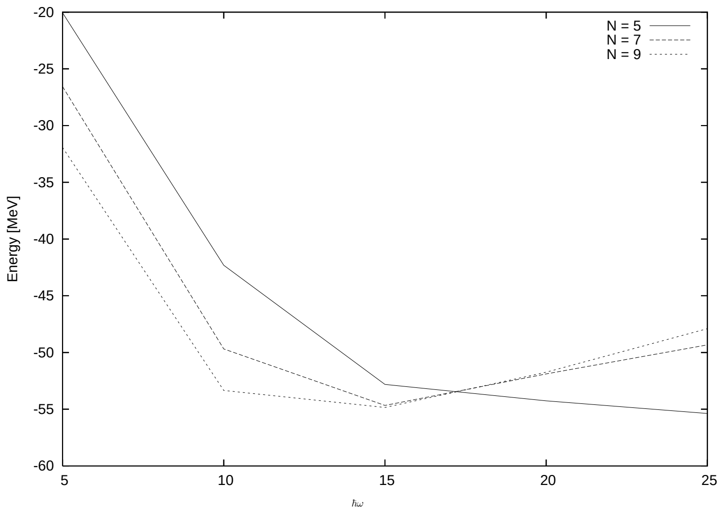


(b) NSC97F

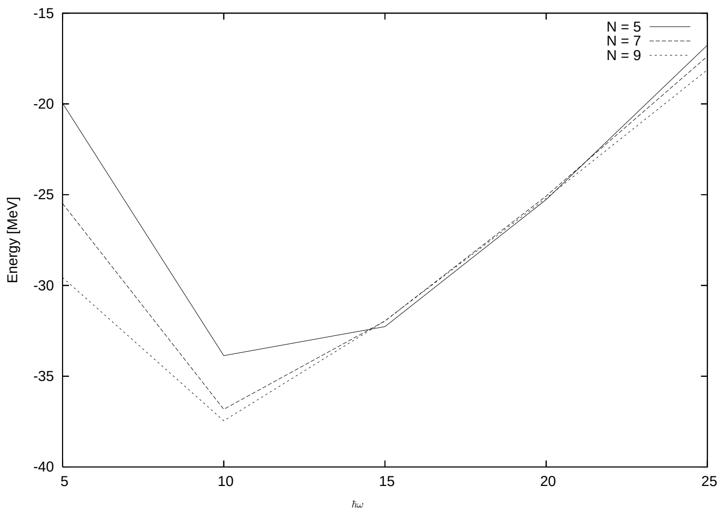


(c) J04

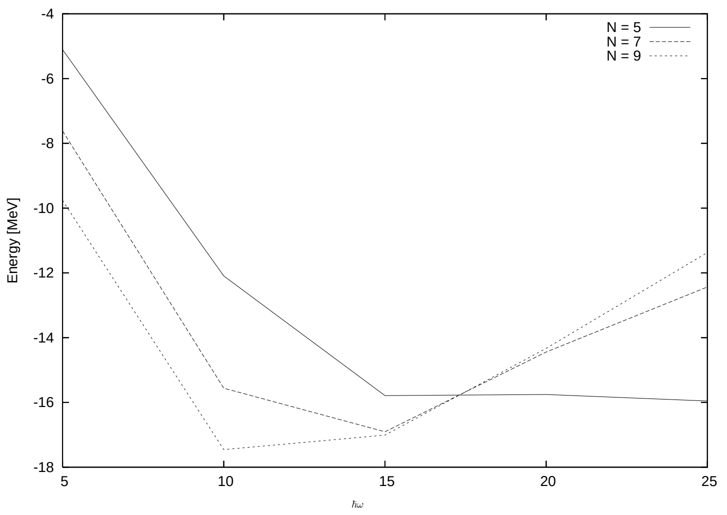
Figure 5.14: Single particle energies for Λ in the $0s_{1/2}$ orbital in ${}^{41}_{\Lambda}$Ca as a function of the oscillator parameter for different sized modelspace. N stands for the maximum $2n + l$ in the model space for the nucleons.



(a) NSC97A



(b) NSC97F



(c) J04

Figure 5.15: Single particle energies for Λ in the $0s_{1/2}$ orbital in ${}^{91}_{\Lambda}$Zr as a function of the oscillator parameter for different sized model spaces. N stands for the maximum $2n + l$ in the model space for the nucleons.

$\hbar\omega$	λ	NSC97A	NSC97F	J04
5.0	2.0	-.142130E+02	-.146005E+02	-.236463E+01
5.0	2.5	-.100220E+02	-.105153E+02	-.145238E+01
5.0	3.0	-.580315E+01	-.621929E+01	-.585954E+00
10.0	2.0	-.283548E+02	-.264695E+02	-.508280E+01
10.0	2.5	-.176048E+02	-.166459E+02	-.228835E+01
10.0	3.0	-.765527E+01	-.718411E+01	0.269105E+00
15.0	2.0	-.343005E+02	-.301819E+02	-.522945E+01
15.0	2.5	-.182380E+02	-.159281E+02	-.570465E+00
15.0	3.0	-.573403E+01	-.417381E+01	0.280576E+01
20.0	2.0	-.353245E+02	-.294330E+02	-.371252E+01
20.0	2.5	-.164269E+02	-.126355E+02	0.202292E+01
20.0	3.0	-.284581E+01	0.692835E-01	0.572580E+01
25.0	2.0	-.341553E+02	-.265827E+02	-.149092E+01
25.0	2.5	-.135103E+02	-.812080E+01	0.498695E+01
25.0	3.0	0.490273E+00	0.497532E+01	0.879451E+01

Table 5.4: Single particle energies for Λ in the $0s_{1/2}$ orbital outside a core of ^{16}O.

$\hbar\omega$	λ	NSC97A	NSC97F	J04
5.0	2.0	-.249132E+02	-.248547E+02	-.591739E+01
5.0	2.5	-.180783E+02	-.181185E+02	-.437293E+01
5.0	3.0	-.113244E+02	-.112121E+02	-.299530E+01
10.0	2.0	-.509021E+02	-.440472E+02	-.132682E+02
10.0	2.5	-.330701E+02	-.277015E+02	-.837402E+01
10.0	3.0	-.164776E+02	-.127420E+02	-.399717E+01
15.0	2.0	-.642515E+02	-.490546E+02	-.173640E+02
15.0	2.5	-.354149E+02	-.256347E+02	-.771858E+01
15.0	3.0	-.142093E+02	-.821421E+01	-.141636E+01
20.0	2.0	-.671886E+02	-.470663E+02	-.175821E+02
20.0	2.5	-.332148E+02	-.206469E+02	-.515134E+01
20.0	3.0	-.113397E+02	-.284464E+01	0.135490E+01
25.0	2.0	-.660513E+02	-.428834E+02	-.159834E+02
25.0	2.5	-.303610E+02	-.150148E+02	-.240329E+01
25.0	3.0	-.844995E+01	0.326465E+01	0.392458E+01

Table 5.5: Single particle energies for Λ in the $0s_{1/2}$ orbital outside a core of ^{40}Ca.

$\hbar\omega$	λ	NSC97A	NSC97F	J04
5.0	2.0	-.358840E+02	-.347509E+02	-.977846E+01
5.0	2.5	-.265584E+02	-.254830E+02	-.761521E+01
5.0	3.0	-.172536E+02	-.159492E+02	-.573762E+01
10.0	2.0	-.750713E+02	-.603279E+02	-.229602E+02
10.0	2.5	-.496889E+02	-.368208E+02	-.155582E+02
10.0	3.0	-.251061E+02	-.162715E+02	-.854547E+01
15.0	2.0	-.986818E+02	-.654056E+02	-.337210E+02
15.0	2.5	-.546621E+02	-.319591E+02	-.169061E+02
15.0	3.0	-.221431E+02	-.102903E+02	-.572883E+01
20.0	2.0	-.105513E+03	-.618633E+02	-.378006E+02
20.0	2.5	-.518825E+02	-.250758E+02	-.144423E+02
20.0	3.0	-.197019E+02	-.289759E+01	-.363980E+01
25.0	2.0	-.104345E+03	-.568018E+02	-.370464E+02
25.0	2.5	-.493258E+02	-.173729E+02	-.124339E+02
25.0	3.0	-.171029E+02	0.701692E+01	-.213177E+01

Table 5.6: Single particle energies for Λ in the $0s_{1/2}$ orbital outside a core of ^{90}Zr.

5.4 Baryon coupled channels

When we defined the channels in section 3.4, the conservation of charge limited the transitions between different baryon combinations. Those channels where these transitions were allowed, were called baryon coupled channels.

We will now investigate the effect of the baryon coupling on the single particle energy for Λ in the $0s_{1/2}$ orbital, outside a core of ^{16}O.

For most channels the baryon coupling has little effect, but for the tensor coupled channel $^3S_1$–$^3D_1$, this is not the case. The $^3S_1$–$^3D_1$ channel with charge $q = 0$, couples Λn, $\Sigma^0 n$ and $\Sigma^- p$, while the channel with $q = 1$ couples Λp, $\Sigma^0 p$ and $\Sigma^+ n$. We will investigate how the final results are affected when we turn off the baryon coupling in these two channels. By turning off, we will mean setting the matrix elements representing transitions between different particle combinations in the same channel, to zero.

Type	Coupled	Uncoupled	ΔE
NSC97A	-17.6048	14.2381	31.8429
NSC97F	-16.6459	16.9117	33.5576
Juelich	-2.28835	-.531068	1.757282

Table 5.7: Differences between the single particle energies for the Λ in the $0s_{1/2}$ orbital in $^{17}_{\Lambda}$O with and without the baryon coupling in the ΛN $^3S_1$–$^3D_1$ channel

Table 5.7 shows the single particle energy when the renormalized matrix elements have been calculated with and without baryon coupling. The difference ΔE between the coupled and uncoupled are also shown. It is obvious that this coupling has a significant effect, more so for the NSC97 models, than for J04. The difference for NSC97 is over 30 MeV, while for the J04 mode it is only about 2 MeV.

a	b	c	d	j	Coupled	Uncoupled	ΔE
$\Lambda\, 0s_{1/2}$	$p\, 0s_{1/2}$	$\Lambda\, 0s_{1/2}$	$p\, 0s_{1/2}$	0	-0.519790	-0.519790	0.000000
$\Lambda\, 0s_{1/2}$	$p\, 0s_{1/2}$	$\Lambda\, 0s_{1/2}$	$p\, 0s_{1/2}$	1	-3.157155	0.646005	3.803160
$\Lambda\, 0s_{1/2}$	$n\, 0s_{1/2}$	$\Lambda\, 0s_{1/2}$	$n\, 0s_{1/2}$	0	-0.556515	-0.556515	0.000000
$\Lambda\, 0s_{1/2}$	$n\, 0s_{1/2}$	$\Lambda\, 0s_{1/2}$	$n\, 0s_{1/2}$	1	-3.196860	0.709984	3.906844
$\Lambda\, 0s_{1/2}$	$p\, 0p_{3/2}$	$\Lambda\, 0s_{1/2}$	$p\, 0p_{3/2}$	1	-0.778716	-0.199606	0.579109
$\Lambda\, 0s_{1/2}$	$p\, 0p_{3/2}$	$\Lambda\, 0s_{1/2}$	$p\, 0p_{3/2}$	2	-2.623350	0.272197	2.895547
$\Lambda\, 0s_{1/2}$	$n\, 0p_{3/2}$	$\Lambda\, 0s_{1/2}$	$n\, 0p_{3/2}$	1	-0.814054	-0.218710	0.595344
$\Lambda\, 0s_{1/2}$	$n\, 0p_{3/2}$	$\Lambda\, 0s_{1/2}$	$n\, 0p_{3/2}$	2	-2.660325	0.316392	2.976717
$\Lambda\, 0s_{1/2}$	$p\, 0p_{1/2}$	$\Lambda\, 0s_{1/2}$	$p\, 0p_{1/2}$	0	-0.470052	0.109057	0.579109
$\Lambda\, 0s_{1/2}$	$p\, 0p_{1/2}$	$\Lambda\, 0s_{1/2}$	$p\, 0p_{1/2}$	1	-0.941512	0.216705	1.158217
$\Lambda\, 0s_{1/2}$	$n\, 0p_{1/2}$	$\Lambda\, 0s_{1/2}$	$n\, 0p_{1/2}$	0	-0.500150	0.095195	0.595345
$\Lambda\, 0s_{1/2}$	$n\, 0p_{1/2}$	$\Lambda\, 0s_{1/2}$	$n\, 0p_{1/2}$	1	-0.946399	0.244290	1.190689
Total					-17.164880	1.115205	18.280085

Table 5.8: Contributions from different matrix elements to the single-particle potential in equation 3.3.1 for the Λ in the $0s_{1/2}$ orbital in $^{17}_{\Lambda}$O using the NSC97A model.

a	b	c	d	j	Coupled	Uncoupled	ΔE
$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	0	-1.47394000	-1.47394000	0.00000000
$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	1	-2.55241500	1.39046850	3.94288350
$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	0	-1.51145500	-1.51145500	0.00000000
$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	1	-2.58103500	1.48748550	4.06852050
$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	1	-1.35887550	-0.75849150	0.60038400
$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	2	-1.97320250	1.02872250	3.00192500
$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	1	-1.39523100	-0.77524950	0.61998150
$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	2	-1.99994250	1.09995750	3.09990000
$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	0	-0.29276500	0.30762000	0.60038500
$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	1	-0.94644000	0.25432950	1.20076950
$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	0	-0.31987950	0.30010050	0.61998000
$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	1	-0.94965750	0.29030400	1.23996150
Total					-17.354838	1.639852	18.99469000

Table 5.9: Contributions from different matrix elements to the single-particle potential in equation 3.3.1 for the Λ in the $0s_{1/2}$ orbital in $^{17}_{\Lambda}$O using the NSC97F model.

a	b	c	d	j	Coupled	Uncoupled	ΔE
$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	0	-1.09650500	-1.09650500	0.00000000
$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0s_{1/2}$	1	-0.22785300	-0.01824540	0.20960760
$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	0	-1.09843000	-1.09843000	0.00000000
$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0s_{1/2}$	1	-0.26662500	-0.01828065	0.24834435
$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	1	-1.11270000	-1.08078300	0.03191700
$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{3/2}$	2	-0.17968350	-0.02009815	0.15958535
$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	1	-1.12126650	-1.08342300	0.03784350
$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{3/2}$	2	-0.20930450	-0.02008483	0.18921967
$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	0	-0.03469530	-0.00277825	0.03191705
$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$p\,0p_{1/2}$	1	-0.46604400	-0.40220850	0.06383550
$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	0	-0.04062960	-0.00278569	0.03784391
$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	$\Lambda\,0s_{1/2}$	$n\,0p_{1/2}$	1	-0.47968350	-0.40399500	0.07568850
Total					-6.333420	-5.247617	1.08580300

Table 5.10: Contributions from different matrix elements to the single-particle potential in equation 3.3.1 for the Λ in the $0s_{1/2}$ orbital in $^{17}_{\Lambda}$O using the J04 model.

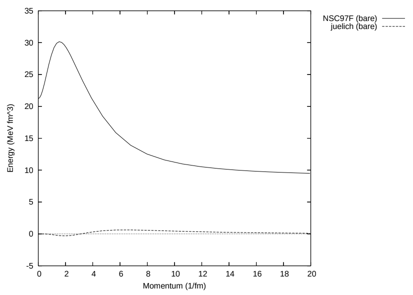
In tables 5.8, 5.9 and 5.10 we see how the different matrix elements contribute to this result. Matrix elements that sum up to the first iteration of the single particle potential are tabulated for NSC97A, NSC97F and J04 respectiviely. We see clearly that the matrix elements with the largest difference, are those that couple to a nonzero total angular momentum. This is true for all three models, but the results for the J04 model are an order of magnitude smaller than for the NSC97 models.

Again, as in previous sections, we find ourselves discussing the large differences between the various interaction models. We now touch upon one of the major problems when doing hypernuclear structure calculations using current interaction models. For NN models, direct scattering experiments have helped constrain the NN interaction to a point where structure calculations are possible. But partly because of the short lifetime of the hyperons, such scattering data are scarce and the quality is not so good for the YN interaction. Although they reproduce scattering data, they all produce different scattering lengths [25].

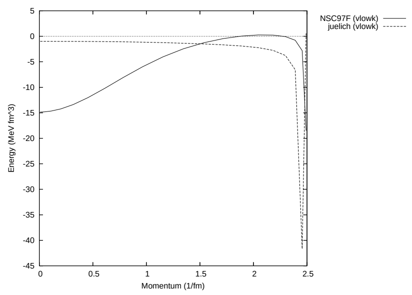
The authors of different interaction models have chosen different ways to parameterize the unknown parts of the interaction, resulting in widely different behaviours, especially for coupled channels. We see this very clearly in figure 5.16, where different results for the Λp $^3S_1$–$^3D_1$ channel are plotted (results for Λn are similar).

In figure 5.16 g) and f), we see that the energy eigenvalues of this channel are different. The NSC97F model has a loosely bound state in this channel that is not present in the J04 model. The difference, however, is only about 0.25 MeV between the two.

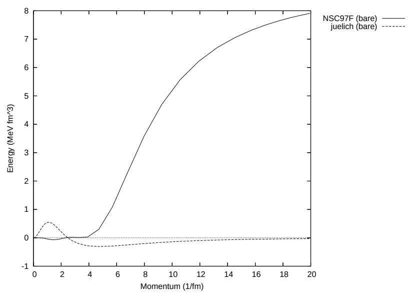
The largest discrepancy comes from the $^3S_1$ block matrix, where the J04 elements are very small compared to the NSC97 elements. The renormalized interaction for this block is plotted in 5.16 b), where it is evident that the NSC97F model generates almost 15 MeV more binding for low-momentum states than the J04 model in this channel.



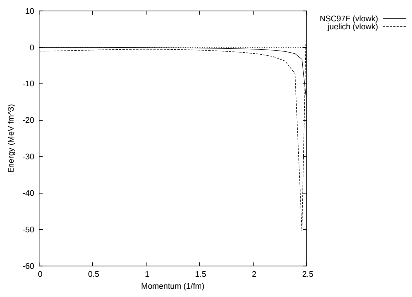
(a) Bare interaction in the $^3S_1$ block matrix.



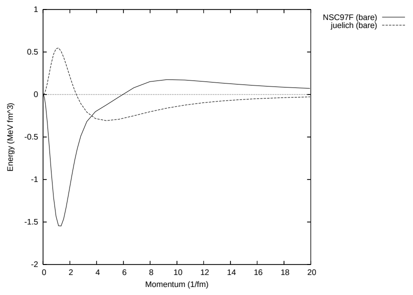
(b) Renormalized interaction in the $^3S_1$ block matrix.



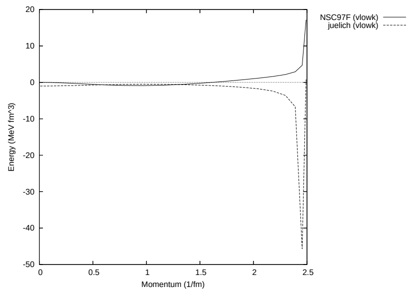
(c) Bare interaction in the $^3D_1$ block matrix.



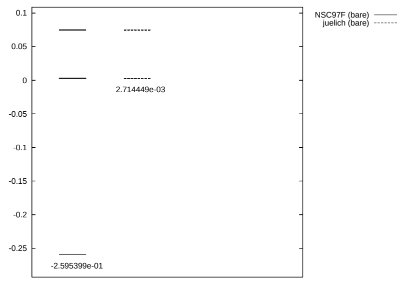
(d) Renormalized interaction in the $^3D_1$ block matrix.



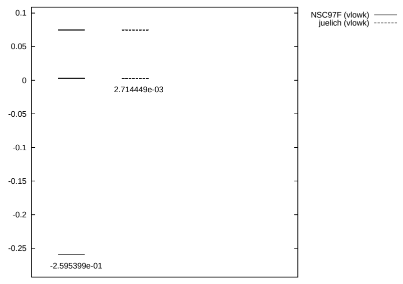
(e) Bare interaction in the $^3S_1$–$^3D_1$ block matrix.



(f) Renormalized interaction in the $^3S_1$–$^3D_1$ block matrix.



(g) Energy eigenvalues of the bare Hamiltonian.



(h) Energy eigenvalues of the renormalized Hamiltonian.

Figure 5.16: Diagonal elements and energy eigenvalues in the Λp $^3S_1$–$^3D_1$ baryon and tensor coupled channel.

5.5 Single particle energies

Our main results in this thesis is the calculation of self-consistent single particle orbitals defining a new basis for further calculation of an effective many-particle interaction using perturbative methods. We will present the single particle energies for the $0s_{1/2}$ Λ orbital in this new basis, which is really a linear combination of three-dimensional spherical harmonic oscillator wavefunctions. We have kept the convenient nomenclature, while redefining the orbitals, energies and matrix elements to compensate for these new basis states.

We will concentrate on five doubly magic nuclei, adding a Λ to these configurations. The selected nuclei are ${}^{17}_{\Lambda}$O, ${}^{41}_{\Lambda}$Ca, ${}^{91}_{\Lambda}$Zr, ${}^{133}_{\Lambda}$Sn and ${}^{209}_{\Lambda}$Pb.

Results are calculated using 60 meshpoints for an uncoupled channel, with 30 in the low momentum model space and 30 in the complement space. The integration limit used is 20.0 fm^{-1}, as justified in section 5.1. The harmonic oscillator space is limited to $N = 2n + l = 7$ for ${}^{17}_{\Lambda}$O, ${}^{41}_{\Lambda}$Ca and ${}^{91}_{\Lambda}$Zr, and to $N = 9$ for ${}^{133}_{\Lambda}$Sn and ${}^{209}_{\Lambda}$Pb. All results in this section are for Λ in the $0s_{1/2}$ orbital for the selected nucleus, although the application is capable of calculating the energies for different orbitals and other hyperons. This is done because we do no get any kind of converged results, so we concentrate on showing how this single particle energy behaves as we vary the the oscillator parameter β and momentum cutoff λ for the selected hypernuclei.

Figures 5.17, 5.18, 5.19, 5.20 and 5.21 each consist of eight plots where a)-e) plot the energy as a function of λ for different values of β, while f)-h) plot the energy as a function of β for different values of λ. The figures relate to ${}^{17}_{\Lambda}$O, ${}^{41}_{\Lambda}$Ca, ${}^{91}_{\Lambda}$Zr, ${}^{133}_{\Lambda}$Sn and ${}^{209}_{\Lambda}$Pb respectively.

The most striking feature, present for all five nuclei, is the almost linear behaviour of the energy as a function of λ for all β. That the amount of binding would increase for smaller model space, was expected. The renormalized two-particle matrix elements in a smaller model space would need to be more attractive in order to sum up to the same amount of binding as the summation in the full space. Although this would be correct for a two-particle problem, for a many-particle problem, we would need many-particle forces to compensate.

The linear behaviour was, however, not expected and needs to be studied further.

Another major feature, discussed also in previous sections, is the large difference between the results from the different interaction models. Results for all nuclei show this feature. This is the primary reason we cannot get converged results from these calculations. The interaction models are not properly constrained by scattering data, giving large uncertainties in the two-particle interaction. All is not lost, however, since there are several experiments on the horizon aimed at constraining the YN interaction and even the YY interaction, needed for perturbative methods beyond first order and hypernuclei consisting of more than one hyperon.

With the libraries developed for this thesis, we are ready to test new models as soon as they are made available and obtain many-particle results immediately using the new data.

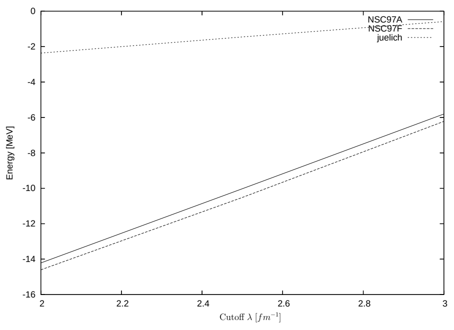
A third common feature of these plots, are the minima of plots f)-h), showing the single-particle energy as a function of β for different cutoffs λ. As the model space is increased with higher cutoff, the energy minima is found for smaller β, with considerable less energy. This tells us that the interaction is weaker for higher cutoffs, generating less binding. This is also consistent with results discussed earlier.

The major difference between the different nuclei, is how strongly the Λ is bound to the nucleus. For a larger nucleus, the Λ is more deeply bound, which is to be expected. In addition, the results for J04 and NSC97 seems to get closer for larger nuclei. It would be interesting to study this difference more closely.

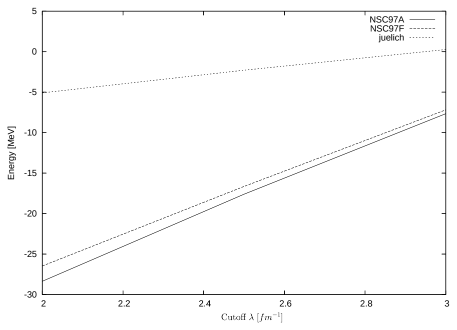
The energy range for a particular plot also increase as the nucleus grows. Differences of up to 100 MeV are registred for the ${}^{209}_{\Lambda}$Pb results, indicating that the results are less reliable for larger nuclei. Since the results are already plagued with large uncertainties, this point will not be stressed further.

It would be interesting to compare these results with the G-matrix method to establish if there is a difference in the results achieved using different renormalization techniques and if so, where these differences originates.

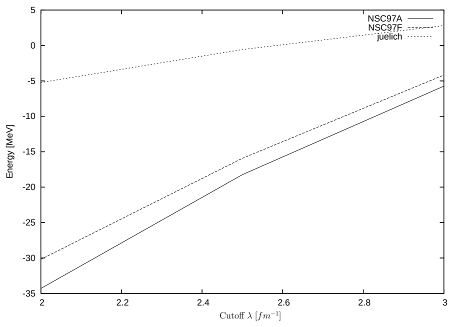
In figure 5.22 we have plotted the minima of the single-particle energy as a function of the size of the nuclei. We can see that the single-particle energies converge towards a limit for both the J04 model and the NSC97F model. For the NSC97A model, this is not so clear, but this may be because we have not hit the minima with our β samples. This limit is the situation where a Λ is in a medium of infinite nuclear matter with $k_F^{\Lambda} = 0$. It will be necessary to compare these results with calculations done for infinite nuclear matter, but this has not been done for this thesis, but is reserved for future work.



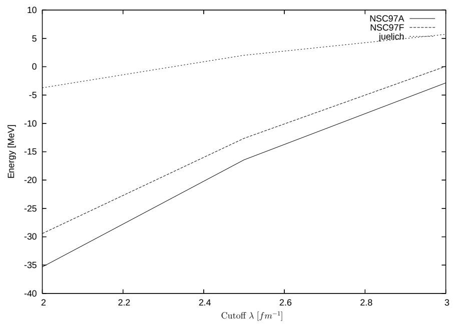
(a) $\hbar\omega = 5$



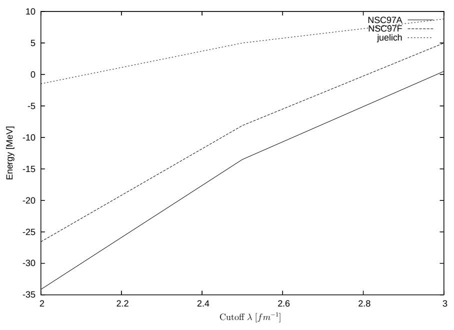
(b) $\hbar\omega = 10$



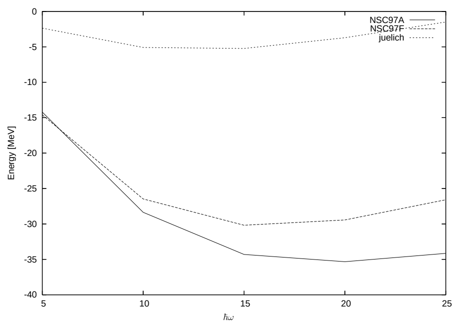
(c) $\hbar\omega = 15$



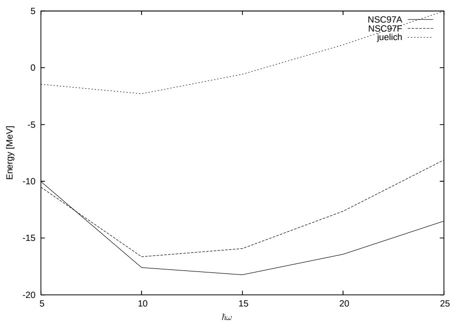
(d) $\hbar\omega = 20$



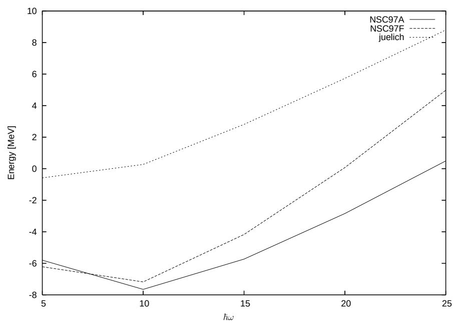
(e) $\hbar\omega = 25$



(f) $\lambda = 2.0$

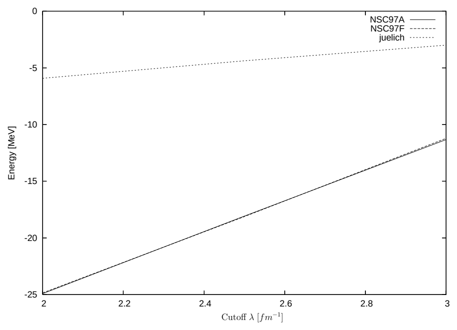


(g) $\lambda = 2.5$

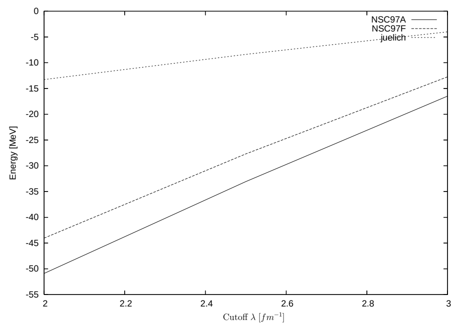


(h) $\lambda = 3.0$

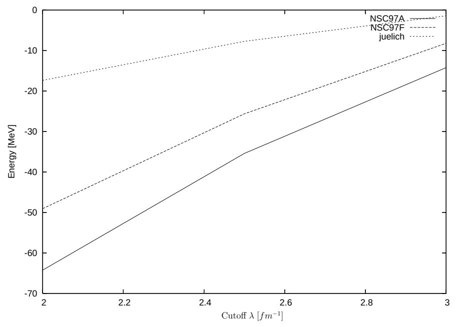
Figure 5.17: Single particle energies for a Λ in $0s_{1/2}$ for ${}^{17}_{\Lambda}$O.



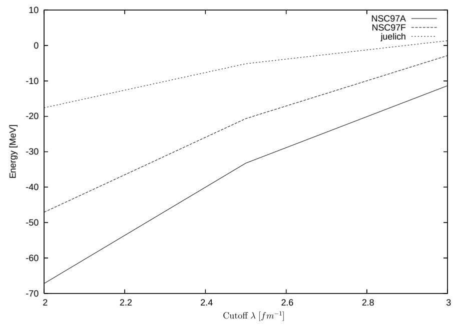
(a) $\hbar\omega = 5$



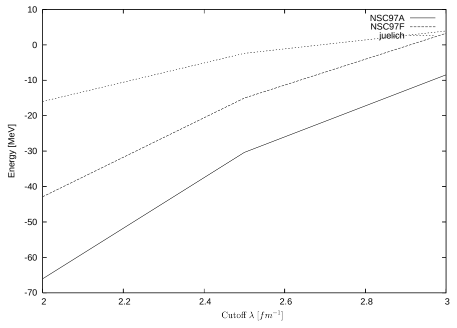
(b) $\hbar\omega = 10$



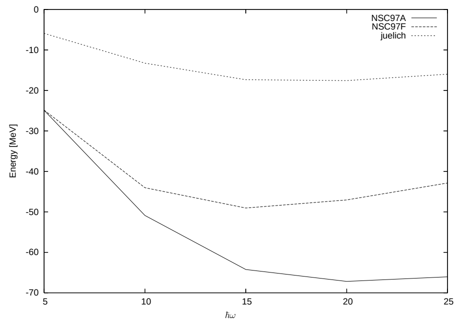
(c) $\hbar\omega = 15$



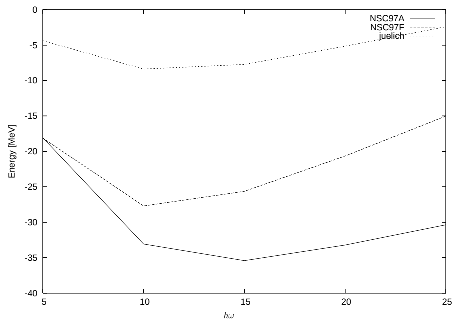
(d) $\hbar\omega = 20$



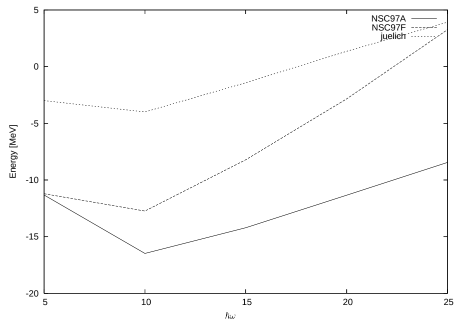
(e) $\hbar\omega = 25$



(f) $\lambda = 2.0$

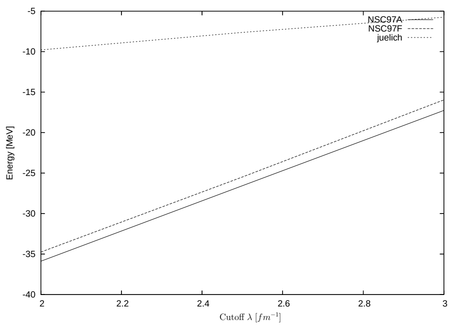


(g) $\lambda = 2.5$

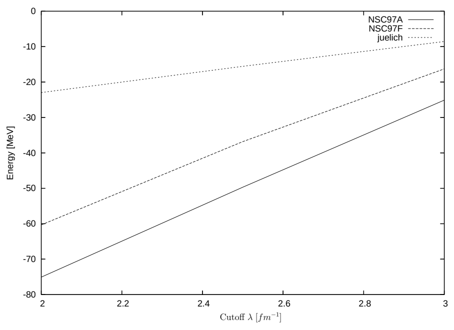


(h) $\lambda = 3.0$

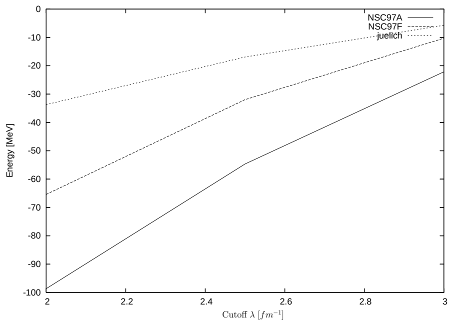
Figure 5.18: Single particle energies for a Λ in $0s_{1/2}$ for ${}^{41}_{\Lambda}$Ca.



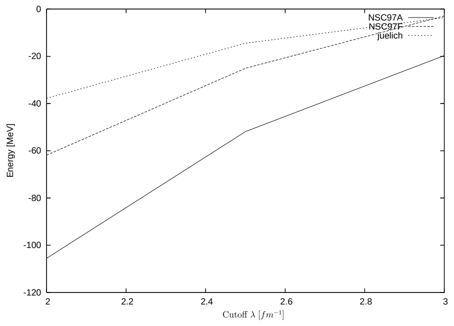
(a) $\hbar\omega = 5$



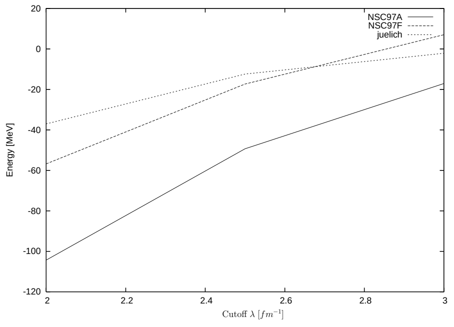
(b) $\hbar\omega = 10$



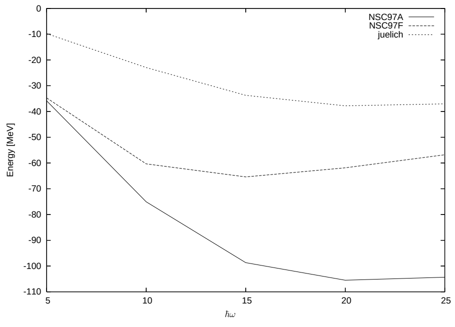
(c) $\hbar\omega = 15$



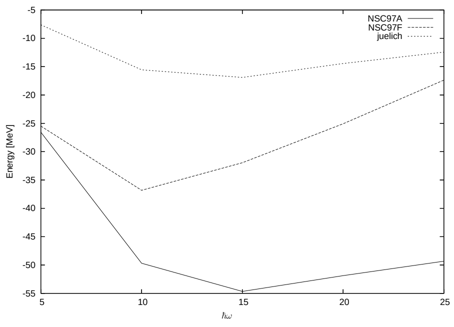
(d) $\hbar\omega = 20$



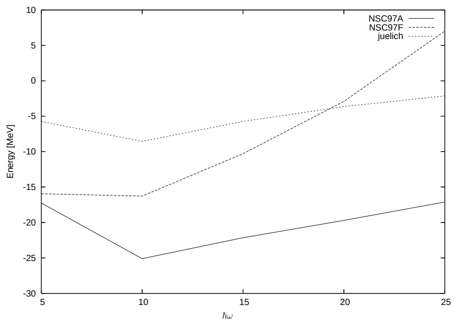
(e) $\hbar\omega = 25$



(f) $\lambda = 2.0$

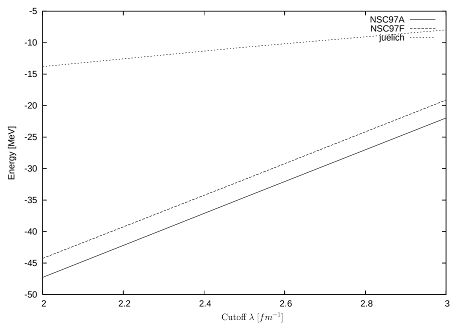


(g) $\lambda = 2.5$

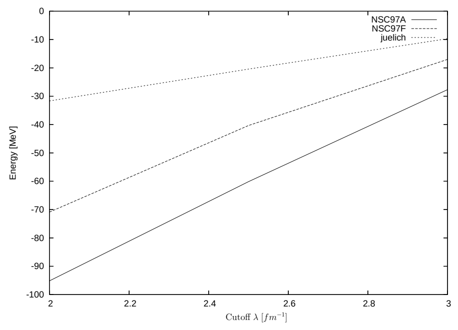


(h) $\lambda = 3.0$

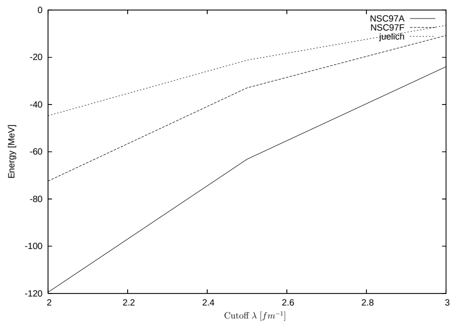
Figure 5.19: Single particle energies for a Λ in $0s_{1/2}$ for ${}^{91}_{\Lambda}$Zr.



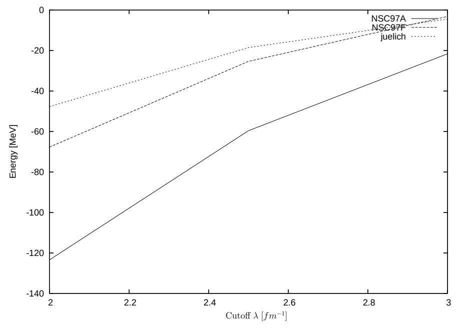
(a) $\hbar\omega = 5$



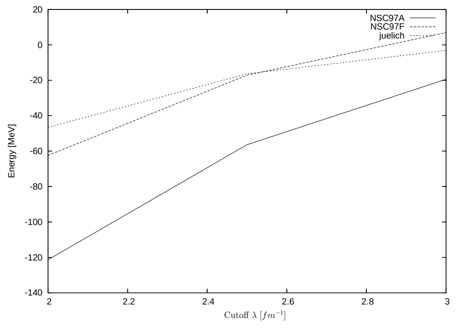
(b) $\hbar\omega = 10$



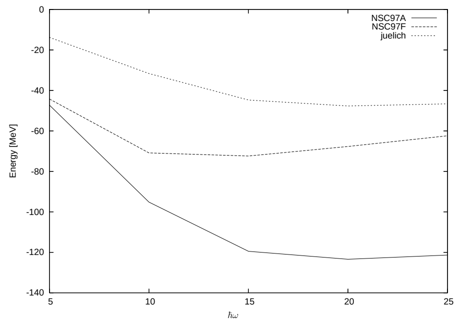
(c) $\hbar\omega = 15$



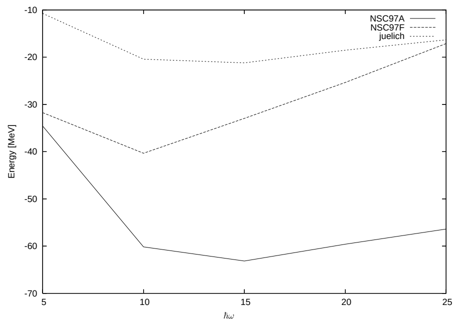
(d) $\hbar\omega = 20$



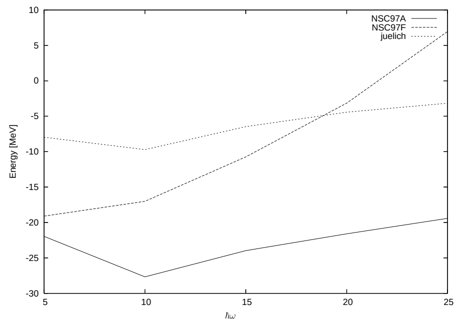
(e) $\hbar\omega = 25$



(f) $\lambda = 2.0$

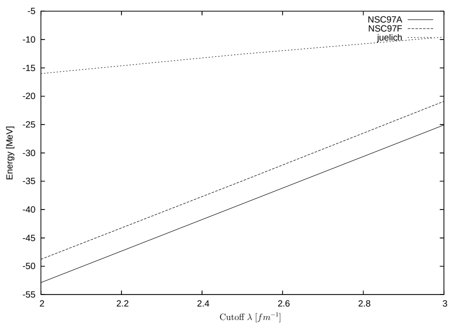


(g) $\lambda = 2.5$

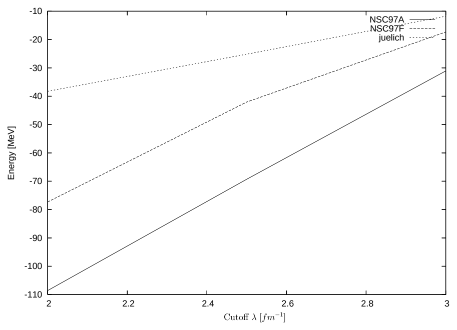


(h) $\lambda = 3.0$

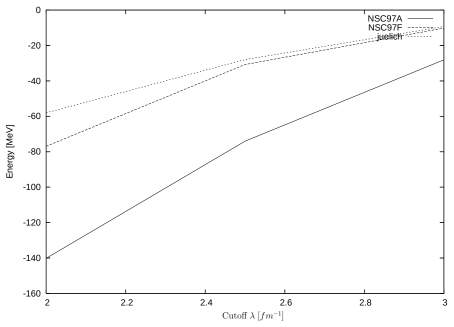
Figure 5.20: Single particle energies for a Λ in $0s_{1/2}$ for ${}^{133}_{\Lambda}$Sn.



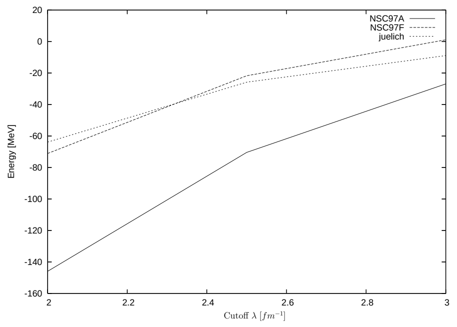
(a) $\hbar\omega = 5$



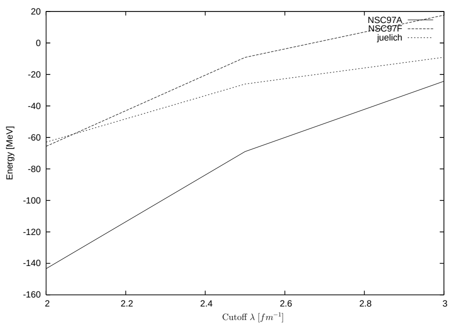
(b) $\hbar\omega = 10$



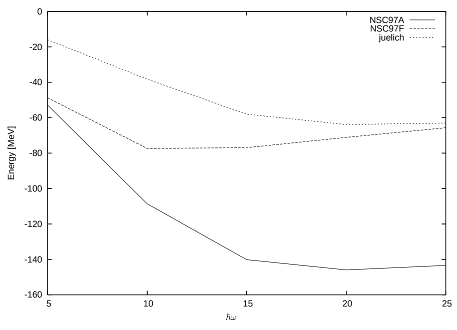
(c) $\hbar\omega = 15$



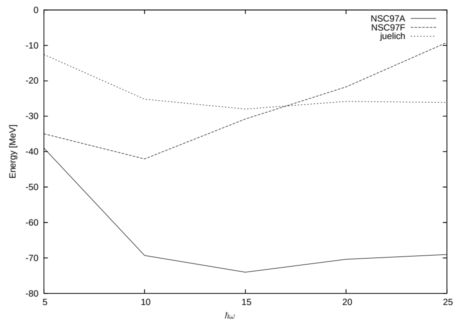
(d) $\hbar\omega = 20$



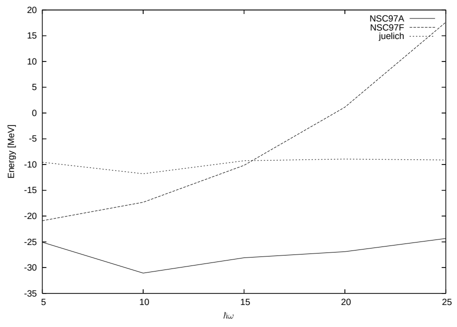
(e) $\hbar\omega = 25$



(f) $\lambda = 2.0$

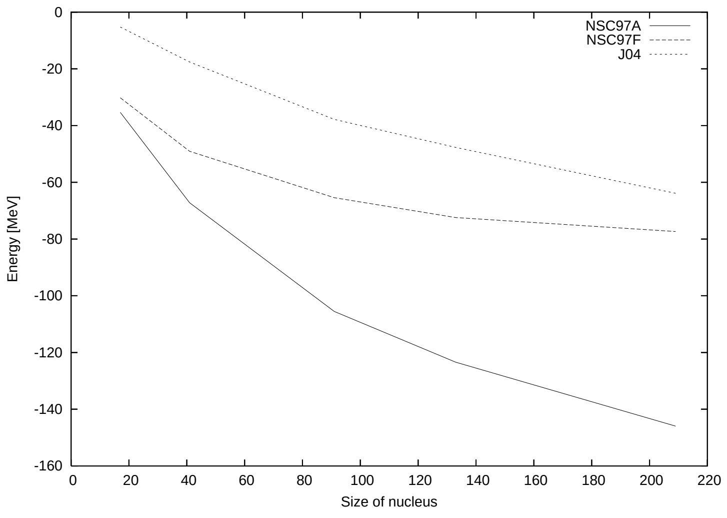


(g) $\lambda = 2.5$

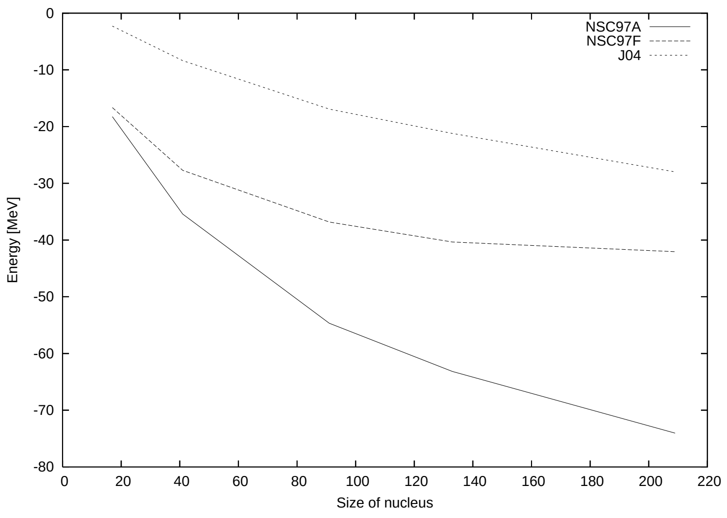


(h) $\lambda = 3.0$

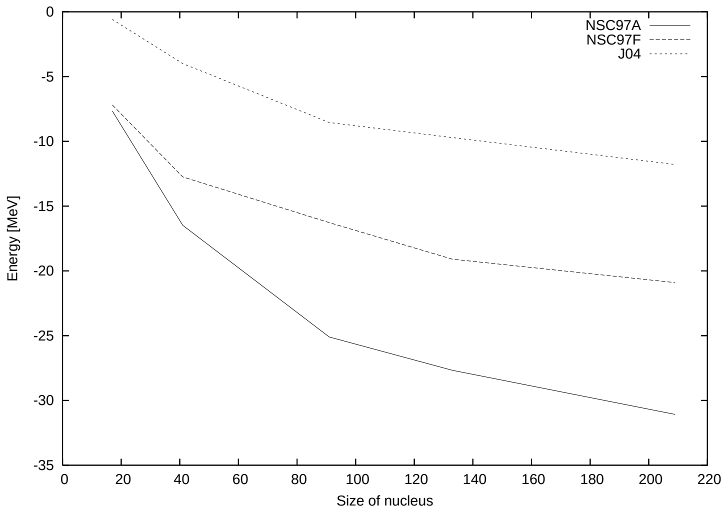
Figure 5.21: Single particle energies for a Λ in $0s_{1/2}$ for ${}^{209}_{\Lambda}$Pb.



(a)



(b)



(c)

Figure 5.22: Medium dependency of the Λ single-particle energy in the $0s_{1/2}$ orbital. Figure (a) shows the result for the model space defined by cutoff $\lambda = 2.0$ in units of fm^{-1}, while (b) and (c) show the results for $\lambda = 2.5$ and $\lambda = 3.0$, respectively.

6 Conclusion

The combination of Python with a low-level compiled language is powerful. We have created a library that extends the Python language with capabilities to do structure calculations and used this library to calculate effective two-particle interactions, using realistic NN interaction models and currently available YN interaction models.

Using these models in a many-particle calculation has,however, proven difficult, as the results using the different models deviate substantially and to the point where they cannot be used to give realistic results. We saw that the Λp and Λn ${}^3S_1$–${}^3D_1$ channels were especially important and that different treatment of these channels could explain the major differences between the interaction models. Additional analysis is needed to give a more quantitative description of these differences and if other channels have similar effects. Our findings agree with recent results from for example [50]. We will need to study the partial wave contributions to specific matrix elements in the laboratory frame, but due to limitations in the tools used in creating the library, this is currently not possible. However, promising new tools are now available to help to alleviate this situation.

We conclude that the currently available YN interaction models are not accurate enough to be used in many-particle calculations. High quality scattering data is needed to constrain the YN interaction, but because of the short lifetime of the hyperons, the data currently available is plagued with large uncertainties.

Several experiments are on the horizon to produce higher quality data for YN scattering and even for YY scattering, but there will be some time before this result in revised interaction models. We will use this time wisely to expand and improve our tools, in order to perform more comprehensive analyses as well as to gain additional capabilities for including hyperons in many-particle calculations.

As our results indicated, a realistic many-particle calculation will also need to incorporate three-body forces. A general framework for incorporating three-body forces is already included in the original application, but will need significant revisions to be able to include hyperons.

During the next decade, there is also the possibility that more general baryon-baryon interactions can be derived directly from the fundamental interaction between quarks and gluons, using Lattice QCD. This holds great promise for a more fundamental understanding of the strong interaction, hopefully bringing our knowledge of nuclear phenomena to a higher level.

A Special functions

A.1 Spherical Bessel functions

The spherical Bessel functions [51], are solutions of the radial differential equation

$$x^2\frac{d^2y}{dx^2}+2x\frac{dy}{dx}+\left(x^2-l(l+1)\right)y=0 \tag{A.1.1}$$

This equation has two linearly independent sets of solutions j_l and y_l called the spherical Bessel functions of the first and second kind respectively. The latter is also known as the spherical Neumann functions,

$$j_l(x)=\sqrt{\frac{\pi}{2x}}J_{l+\frac{1}{2}}(x), \tag{A.1.2}$$

$$y_l(x)=\sqrt{\frac{\pi}{2x}}Y_{l+\frac{1}{2}}(x)=(-1)^{l+1}\sqrt{\frac{\pi}{2x}}J_{-l-\frac{1}{2}}(x), \tag{A.1.3}$$

where J_n and Y_n are the ordinary Bessel functions of the first and second kind. The latter is also called the ordinary Neumann functions.

These are two independent solutions to a related differential equation

$$x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+\left(x^2-n^2\right)y=0, \tag{A.1.4}$$

for integer n. For non-integer n, two independent solutions are

$$J_n(x), \tag{A.1.5}$$

$$J_{-n}(x). \tag{A.1.6}$$

We will only need the spherical Bessel functions of the first kind. They can be expressed as

$$j_l(x)=(-x)^l\left(\frac{1}{x}\frac{d}{dx}\right)^l\frac{\sin x}{x}, \tag{A.1.7}$$

where the solutions for $l=0,1,2$ are

$$j_0(x)=\frac{\sin x}{x}, \tag{A.1.8}$$

$$j_1(x)=\frac{\sin x}{x^2}-\frac{\cos x}{x}, \tag{A.1.9}$$

$$j_2(x)=\left(\frac{3}{x^2}-1\right)\frac{\sin x}{x}-\frac{3\cos x}{x^2}. \tag{A.1.10}$$

A.2 Legendre polynomials

The Legendre polynomials or Legendre functions [51], are solutions of the Legendre differential equation

$$\frac{d}{dx}\left((1-x^2)\frac{d}{dx}P(x)\right)+l(l+1)P(x)=0. \tag{A.2.1}$$

The solutions to this equation are called Legendre polynomial of order l - $P_l(x)$ and are polynomias of order l. They may be expressed using Rodrigues' formula

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} \left(x^2 - 1\right)^l . \tag{A.2.2}$$

The polynomials have an orthogonality property on the interval $-1 \leq x \leq 1$.

$$\int_{-1}^{1} \mathrm{d}x P_m(x) P_n(x) = \frac{2}{2n+1} \delta_{mn}. \tag{A.2.3}$$

The Legendre polynomials for $l = 0, 1, 2$ are

$$P_0(x) = 1, \tag{A.2.4}$$
$$P_1(x) = x, \tag{A.2.5}$$
$$P_2(x) = \frac{1}{2} \left(3x^2 - 1\right). \tag{A.2.6}$$

A.3 Associated Laguerre polynomials

The associated or generalized Laguerre polynomials [51] $L_n^k(x)$ are related to the simple Laguerre polynomials $L_n(x)$ by the relation

$$L_n^k(x) = (-1)^k \frac{d^k}{dx^k} L_{n+k}(x). \tag{A.3.1}$$

The simple Laguerre polynomials are power series solutions of the differential equation

$$x \frac{d^2}{dx^2} L(x) + (1 - x) \frac{d}{dx} L(x) + n L(x) = 0, \tag{A.3.2}$$

and may be defined by the Rodrigues formula

$$L_n(x) = \frac{1}{n!} e^x \frac{d^n}{x^n} \left(x^n e^{-x}\right). \tag{A.3.3}$$

The associated Laguerre polynomials satifies the differential equation

$$x \frac{d^2}{dx^2} L(x) + (k + 1 - x) \frac{d}{dx} L(x) + n L(x) = 0, \tag{A.3.4}$$

and can be expressed on a form derived from the Rodrigues formula for the simple Laguerre polynomials

$$L_m^k(x) = \frac{x^{-k} e^x}{n!} \frac{d^n}{dx^n} \left(x^{n+k} e^{-x}\right). \tag{A.3.5}$$

The associated Laguerre polynomials satisfies an orthonormality relation on the interval $x \in [0, \infty]$

$$\int_0^\infty \mathrm{d}x \; x^k e^{-x} L_n^k(x) L_m^k(x) = \frac{(n+k)!}{n!} \delta_{mn}. \tag{A.3.6}$$

B Relativistic field theory

B.1 Notation

We define the gamma matrices [1]

$$
\begin{aligned}
\gamma^0 &= \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & -\mathbf{1} \end{bmatrix}, \\
\gamma^k &= \begin{bmatrix} \mathbf{0} & \sigma^k \\ -\sigma^k & \mathbf{0} \end{bmatrix}, \\
\gamma^5 = \gamma_5 = i\gamma^0\gamma^1\gamma^2\gamma`3 &= \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} \end{bmatrix},
\end{aligned}
$$

where σ^k are the usual Pauli matrices

$$
\begin{aligned}
\sigma^1 = \sigma_1 &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \\
\sigma^2 = \sigma_2 &= \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \\
\sigma^3 = \sigma_3 &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},
\end{aligned}
$$

and also the commutator and partial derivatives

$$
\begin{aligned}
\sigma^{\mu\nu} &= \frac{i}{2}[\gamma^\mu, \gamma^\nu], \\
i\partial_\mu &= i\frac{\partial}{\partial x^\mu}.
\end{aligned}
$$

We will use Einstein's summing conventions, where we sum over equal indices. Greek indices range from 0 to 3, while latin indices range from 1 to 3.

$\bar{\psi} = \psi^\dagger\gamma^0$ is the adjoint operator, m_B is the nucleon mass, while $\varphi^{(s)}$, $\varphi^{(ps)}$ and $\varphi^{(v)}_\mu$ are the scalar, pseudoscalar and vector meson fields respectively.

B.2 Dirac Spinor field

$$\psi = u(\mathbf{q})e^{-i\mathbf{p}\cdot\mathbf{x}}, \tag{B.2.1}$$

where

$$u(\mathbf{q}) = \sqrt{\frac{E+m_B}{2m_B}} \begin{bmatrix} \chi \\ \frac{\sigma\cdot\mathbf{p}}{E+m_B}\chi \end{bmatrix}, \tag{B.2.2}$$

$$\tag{B.2.3}$$

and χ are the Pauli Spinors

$$\chi_- = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad \chi_+ = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{B.2.4}$$

representing negative and positive spin respectively.

[1]The relativistic notatin used here follows [52].

B.3 Meson-Baryon vertices and meson propagators

The propagator and Lagrangian density depends on what kind of meson field is evaluated. For low-energy NN systems there are only three relevant meson fields [20]

- Scalar (s)
- Pseudoscalar (ps)
- vector (v)

The commonly used Lagrangians to couple these meson fields to the baryon fields are

$$\begin{aligned} \mathcal{L}_s &= g_s \bar{\psi}\psi\varphi^{(s)}, && \text{(B.3.1)} \\ \mathcal{L}_{ps} &= -g_{ps}\bar{\psi}i\gamma_5\psi\varphi^{(ps)}, && \text{(B.3.2)} \\ \mathcal{L}_v &= -g_v\bar{\psi}\gamma^\mu\psi\varphi^{(v)}_\mu - \frac{f_v}{4m_B}\bar{\psi}\sigma^{\mu\nu}\psi\left(\partial_\mu\varphi^{(v)}_\nu - \partial_\nu\varphi^{(v)}_\mu\right), && \text{(B.3.3)} \end{aligned}$$

where ψ represents the baryon field as defined in appendix B.2, $\bar{\psi} = \psi^\dagger\gamma^0$ is the adjoint operator, m_B is the nucleon mass, while $\varphi^{(s)}$, $\varphi^{(ps)}$ and $\varphi^{(v)}_\mu$ are the scalar, pseudoscalar and vector meson fields respectivly.

Instead of the pseudoscalar field, we can alternatively use a pseudovector(pv) field with Lagrangian

$$\mathcal{L}_{pv} = -\frac{f_{ps}}{m_{ps}}\bar{\psi}\gamma^5\gamma^\mu\psi\delta_\mu\varphi^{(pv)}, \qquad \text{(B.3.4)}$$

where $f_{ps} = \frac{m_ps}{2M}g_{ps}$.

For an isovector meson, the meson field φ^α must be exchanged with $\tau\cdot\varphi^\alpha$, where τ are the familiar Pauli matrices.

The Meson-Baryon vertices are derived from the Lagrangian density of the corresponding meson field and given by

$$\begin{aligned} \Gamma_s &= g_s\hat{1}, \\ \Gamma_{ps} &= -g_{ps}i\gamma^5, \\ \Gamma_{pv} &= -\frac{f_{ps}}{m_ps}\gamma^5\gamma^\mu i(p-p')_\nu, \\ \Gamma_v &= -g_v\gamma^\mu + \frac{f_v}{2m_B}\sigma^{\mu\nu}i(p-p')_\nu. \end{aligned}$$

The propagators for scalar and pseudo scalar exchanges are

$$\frac{i}{(q'-q)^2 - m_\alpha}, \qquad \text{(B.3.5)}$$

and for vector exchanges

$$i\frac{-g_{\mu\nu} + (q'-q)_\nu/m_v^2}{-(\mathbf{q}'-\mathbf{q})^2 - m_v^2}. \qquad \text{(B.3.6)}$$

B.4 Momentum space interaction for a scalar meson exchange

The contribution to the OBE interaction for a scalar meson exchange as given in [20] is

$$\begin{aligned}\langle \mathbf{q}'\lambda_1'\lambda_2'|V_s|\mathbf{q}\lambda_1\lambda_2\rangle &= -g_s^2\bar{u}(\mathbf{q}',\lambda_1')u(\mathbf{q},\lambda_1)\bar{u}(-\mathbf{q}',\lambda_2')u(-\mathbf{q},\lambda_2)\times \\ &\quad \left[(\mathbf{q}'-\mathbf{q})^2+m_s^2\right]^{-1,}\end{aligned}$$

where $\lambda_i(\lambda_i')$ represents the helicity of an incoming(outgoing) baryon, u($\bar{u}$) is the incoming(outgoing) dirac spinor field as defined in appendix B.2.

C Angular momentum

C.1 Definition

A general angular momentum operator $\hat{J}$, satisfies the following properties

$$\left[\hat{J}_i, \hat{J}_j\right] = i\epsilon_{ijk}\hbar\hat{J}_k, \tag{C.1.1}$$

$$\left[\hat{J}^2, \hat{J}_i\right] = 0, \tag{C.1.2}$$

$$\left[\hat{J}^2, \hat{J}_\pm\right] = 0, \tag{C.1.3}$$

$$\left[\hat{J}_z, \hat{J}_\pm\right] = \pm\hbar\hat{J}_\pm, \tag{C.1.4}$$

$$\left[\hat{J}_+, \hat{J}_-\right] = 2\hbar\hat{J}_z, \tag{C.1.5}$$

where we define

$$\hat{J}^2 = \hat{J}_x\hat{J}_x + \hat{J}_y\hat{J}_y + \hat{J}_z\hat{J}_z, \tag{C.1.6}$$

since in general the angular momentum operators in different directions does not commute. We have also defined the so-called ladder operators

$$\hat{J}_\pm = \hat{J}_x + i\hat{J}_y. \tag{C.1.7}$$

We define the eigenvectors of the angular momentum operators $|j;m\rangle$ and set up the eigenvalue equations

$$\hat{J}^2|j;m\rangle = j(j+1)\hbar^2|j;m\rangle, \tag{C.1.8}$$

$$\hat{J}_i|j;m\rangle = m\hbar|j;m\rangle, \tag{C.1.9}$$

$$\hat{J}_\pm|j;m\rangle = \hbar\sqrt{(j\mp m)(j\pm m+1)}|j;m\pm 1\rangle. \tag{C.1.10}$$

C.1.1 Coupling of angular momentum operators

When we couple two angular momentum, we can operate with two equivalent basis sets.

We define two angular momentum operators $\hat{J}_1$ and $\hat{J}_2$ satisfying all relations in appendix C.1. They operate in different subspaces, so they commute

$$\left[\hat{J}_{1i}, \hat{J}_{2j}\right] = 0, \tag{C.1.11}$$

where the subscripts i and j means the different directions of the operators.

These operator satisfies the eigenvalue equations

$$\hat{J}_1^2|j_1;m_1\rangle = j_1(j_1+1)\hbar^2|j_1;m_1\rangle, \tag{C.1.12}$$

$$\hat{J}_{1i}|j_1;m_1\rangle = m_1\hbar|j_1;m_1\rangle, \tag{C.1.13}$$

$$\hat{J}_2^2|j_2;m_2\rangle = j_2(j_2+1)\hbar^2|j_2;m_2\rangle, \tag{C.1.14}$$

$$\hat{J}_{2i}|j_2;m_2\rangle = m_2\hbar|j_2;m_2\rangle. \tag{C.1.15}$$

When we couple the subspaces spanned by the basis sets $|j_1; m_1\rangle$ and $|j_2; m_2\rangle$ we have the option to define a new operator

$$\hat{J} = \hat{J}_1 \otimes \mathbb{I}_2 + \hat{J}_2 \otimes \mathbb{I}_1, \tag{C.1.16}$$

where $\otimes$ is the operator for a tensor product, while $\mathbb{I}_i$ is the identity operator in the subspace where $\hat{J}_i$ operates. This is commonly written as

$$\hat{J} = \hat{J}_1 + \hat{J}_2, \tag{C.1.17}$$

which is also the notation we'll use here. The coupled basis set will in this notation reads

$$|j_1; m_1\rangle \otimes |j_2; m_2\rangle = |j_1, j_2; m_1, m_2\rangle, \tag{C.1.18}$$

The operator $\hat{J}$ satisfies angular momentum algebra and the eigenvalue equations

$$\hat{J}^2 |j_1, j_2; jm\rangle = j(j+1)\hbar^2 |j_1, j_2; jm\rangle, \tag{C.1.19}$$
$$\hat{J}_i |j_1, j_2; jm\rangle = m\hbar |j_1, j_2; jm\rangle, \tag{C.1.20}$$
$$\tag{C.1.21}$$

where

$$|j_1 - j_2| \leq j \leq j_1 + j_2, \tag{C.1.22}$$
$$-j \leq m \leq j. \tag{C.1.23}$$

The overlap between the two different basis sets is defined as

$$|j_1, j_2; jm\rangle = \sum_{m_1, m_2} |j_1, j_2; m_1, m_2\rangle\langle j_1, j_2; m_1, m_2 | j_1, j_2; jm\rangle, \tag{C.1.24}$$

where we have used the completeness property of the eigenstates

$$\sum_{m_1, m_2} |j_1, j_2; m_1, m_2\rangle\langle j_1, j_2; m_1, m_2| = \mathbb{I}, \tag{C.1.25}$$

and $\langle j_1, j_2; m_1, m_2 | j_1, j_2; jm\rangle$ are the so-called Clebsch-Gordan (CG) coefficients.

The CG coefficients are related to the Wigner 3-j symbols by the transformation [53]

$$\langle j_1, j_2; m_1, m_2 | j_1, j_2; jm\rangle = (-1)^{j_1 - j_2 + m} \sqrt{2j+1} \begin{pmatrix} j_1 & j_2 & j \\ m_1 & m_2 & -m \end{pmatrix} \tag{C.1.26}$$

Extensive sets of tables for the Wigner 3-j symbols can be found in [54], while expressions for calculating the symbols can be found in [55].

For couping three and four angular momenta, the Wigner 6-j and 9-j symbols are used, see for example [56].

C.2 Isospin

Isospin is an SU(2) summetry, and as such, the operators $\hat{T}_\alpha$ satisfies all the properties of an angular momentum operator defined in section C.1. We have

$$\left[\hat{T}_i, \hat{T}_j\right] = i\epsilon_{ijk}\hbar\hat{T}_k, \tag{C.2.1}$$

$$\left[\hat{T}^2, \hat{T}_i\right] = 0, \tag{C.2.2}$$

$$\left[\hat{T}^2, \hat{T}_\pm\right] = 0, \tag{C.2.3}$$

$$\left[\hat{T}_z, \hat{T}_\pm\right] = \pm\hbar\hat{T}_\pm, \tag{C.2.4}$$

$$\left[\hat{T}_+, \hat{T}_-\right] = 2\hbar\hat{T}_z, \tag{C.2.5}$$

where we define

$$\hat{T}^2 = \hat{T}_x\hat{T}_x + \hat{T}_y\hat{T}_y + \hat{T}_z\hat{T}_z, \tag{C.2.6}$$

and

$$\hat{T}_\pm = \hat{T}_x + i\hat{T}_y. \tag{C.2.7}$$

We define the eigenvectors of the isospin operators $|j;m\rangle$ and set up the eigenvalue equations

$$\hat{T}^2|t;m\rangle = t(t+1)\hbar^2|t;m\rangle, \tag{C.2.8}$$

$$\hat{T}_i|t;m\rangle = m\hbar|t;m\rangle, \tag{C.2.9}$$

$$\hat{T}_\pm|t;m\rangle = \hbar\sqrt{(t\mp m)(t\pm m+1)}|t;m\pm 1\rangle. \tag{C.2.10}$$

We will now see how the isospin part of the two-particle wavefunction behaves when we couple the two particles individual isospin, to a total isospin.

From section C.1.1 we have the relation

$$|t_1, t_2; m_1, m_2\rangle = \sum_{tm} |t_1, t_2; tm\rangle\langle t_1, t_2; tm|t_1, t_2; m_1, m_2\rangle, \tag{C.2.11}$$

where $\langle t_1, t_2; tm|t_1, t_2; m_1, m_2\rangle$ are the Clebch-Gordan coefficients defined in appendix C.1.1.

We will set up the Clebch-Gordan coefficients for the possible NN and YN combinations.

For a Nucleon-Nucleon (NN) wavefunction we have

$$t_1 = 1/2, \tag{C.2.12}$$

$$t_2 = 1/2, \tag{C.2.13}$$

$$t = 0, 1. \tag{C.2.14}$$

For a Lambda-Nucleon (ΛN) wavefunction we have

$$t_1 = 1/2, \tag{C.2.15}$$

$$t_2 = 0, \tag{C.2.16}$$

$$t = 1/2. \tag{C.2.17}$$

For a Sigma-Nucleon (Σ N) wavefunction we have

$$\begin{aligned} t_1 &= 1/2, && \text{(C.2.18)} \\ t_2 &= 1, && \text{(C.2.19)} \\ t &= 1/2, 3/2. && \text{(C.2.20)} \end{aligned}$$

C.2.1 Proton Proton case

For proton proton we have

$$\begin{aligned} m_1 &= 1/2, && \text{(C.2.21)} \\ m_2 &= 1/2, && \text{(C.2.22)} \\ m &= m_1 + m_2 = 1. && \text{(C.2.23)} \end{aligned}$$

The wavefunction becomes

$$\begin{aligned} |\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, \tfrac{1}{2}\rangle &= \sum_t |\tfrac{1}{2}, \tfrac{1}{2}; t, m = 1\rangle\langle \tfrac{1}{2}, \tfrac{1}{2}; t, m = 1|\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, \tfrac{1}{2}\rangle && \text{(C.2.24)} \\ &= |\tfrac{1}{2}, \tfrac{1}{2}; t = 0, m = 1\rangle\langle \tfrac{1}{2}, \tfrac{1}{2}; t = 0, m = 1|\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, \tfrac{1}{2}\rangle && \text{(C.2.25)} \\ &+ |\tfrac{1}{2}, \tfrac{1}{2}; t = 1, m = 1\rangle\langle \tfrac{1}{2}, \tfrac{1}{2}; t = 1, m = 1|\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, \tfrac{1}{2}\rangle. && \text{(C.2.26)} \end{aligned}$$

C.2.2 Proton neutron case

For proton neutron we have

$$\begin{aligned} m_1 &= 1/2, && \text{(C.2.27)} \\ m_2 &= -1/2, && \text{(C.2.28)} \\ m &= m_1 + m_2 = 0. && \text{(C.2.29)} \end{aligned}$$

The wavefunction becomes

$$\begin{aligned} |\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, -\tfrac{1}{2}\rangle &= \sum_t |\tfrac{1}{2}, \tfrac{1}{2}; t, m = 0\rangle\langle \tfrac{1}{2}, \tfrac{1}{2}; t, m = 0|\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, -\tfrac{1}{2}\rangle && \text{(C.2.30)} \\ &= |\tfrac{1}{2}, \tfrac{1}{2}; t = 0, m = 0\rangle\langle \tfrac{1}{2}, \tfrac{1}{2}; t = 0, m = 0|\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, -\tfrac{1}{2}\rangle && \text{(C.2.31)} \\ &+ |\tfrac{1}{2}, \tfrac{1}{2}; t = 1, m = 0\rangle\langle \tfrac{1}{2}, \tfrac{1}{2}; t = 1, m = 0|\tfrac{1}{2}, \tfrac{1}{2}; \tfrac{1}{2}, -\tfrac{1}{2}\rangle. && \text{(C.2.32)} \end{aligned}$$

C.2.3 Neutron Neutron case

For neutron neutron we have

$$\begin{aligned} m_1 &= -1/2, && \text{(C.2.33)} \\ m_2 &= -1/2, && \text{(C.2.34)} \\ m &= m_1 + m_2 = -1. && \text{(C.2.35)} \end{aligned}$$

The wavefunction becomes

$$|\tfrac{1}{2},\tfrac{1}{2};-\tfrac{1}{2},-\tfrac{1}{2}\rangle = \sum_t |\tfrac{1}{2},\tfrac{1}{2};t,m=-1\rangle\langle\tfrac{1}{2},\tfrac{1}{2};t,m=-1|\tfrac{1}{2},\tfrac{1}{2};-\tfrac{1}{2},-\tfrac{1}{2}\rangle \quad \text{(C.2.36)}$$

$$= |\tfrac{1}{2},\tfrac{1}{2};t=0,m=-1\rangle\langle\tfrac{1}{2},\tfrac{1}{2};t=0,m=-1|\tfrac{1}{2},\tfrac{1}{2};-\tfrac{1}{2},-\tfrac{1}{2}\rangle \quad \text{(C.2.37)}$$

$$+ |\tfrac{1}{2},\tfrac{1}{2};t=1,m=-1\rangle\langle\tfrac{1}{2},\tfrac{1}{2};t=1,m=-1|\tfrac{1}{2},\tfrac{1}{2};-\tfrac{1}{2},-\tfrac{1}{2}\rangle. \quad \text{(C.2.38)}$$

C.2.4 Proton Lambda case

For proton lambda we have

$$m_1 = 1/2, \quad \text{(C.2.39)}$$

$$m_2 = 0, \quad \text{(C.2.40)}$$

$$m = m_1 + m_2 = 1/2. \quad \text{(C.2.41)}$$

The wavefunction becomes

$$|\tfrac{1}{2},0;\tfrac{1}{2},0\rangle = |\tfrac{1}{2},0;t=\tfrac{1}{2},m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},0;t=\tfrac{1}{2},m=\tfrac{1}{2}|\tfrac{1}{2},0;\tfrac{1}{2},0\rangle \quad \text{(C.2.42)}$$

$$= |\tfrac{1}{2},0;t=\tfrac{1}{2},m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},0;t=\tfrac{1}{2},m=\tfrac{1}{2}|\tfrac{1}{2},0;\tfrac{1}{2},0\rangle. \quad \text{(C.2.43)}$$

$$\text{(C.2.44)}$$

C.2.5 Neutron Lambda case

For neutron lambda we have

$$m_1 = -1/2, \quad \text{(C.2.45)}$$

$$m_2 = 0, \quad \text{(C.2.46)}$$

$$m = m_1 + m_2 = -1/2. \quad \text{(C.2.47)}$$

The wavefunction becomes

$$|\tfrac{1}{2},0;-\tfrac{1}{2},0\rangle = |\tfrac{1}{2},0;t=\tfrac{1}{2},m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},0;t=\tfrac{1}{2},m=-\tfrac{1}{2}|\tfrac{1}{2},0;-\tfrac{1}{2},0\rangle \quad \text{(C.2.48)}$$

$$= |\tfrac{1}{2},0;t=\tfrac{1}{2},m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},0;t=\tfrac{1}{2},m=-\tfrac{1}{2}|\tfrac{1}{2},0;-\tfrac{1}{2},0\rangle. \quad \text{(C.2.49)}$$

$$\text{(C.2.50)}$$

C.2.6 Proton Σ^+ case

For proton $Sigma^+$ we have

$$m_1 = 1/2, \quad \text{(C.2.51)}$$

$$m_2 = 1, \quad \text{(C.2.52)}$$

$$m = m_1 + m_2 = 3/2, \quad \text{(C.2.53)}$$

The wavefunction becomes

$$|\tfrac{1}{2},1;\tfrac{1}{2},1\rangle = \sum_t |\tfrac{1}{2},1;t,m=\tfrac{3}{2}\rangle\langle\tfrac{1}{2},1;t,m=\tfrac{3}{2}|\tfrac{1}{2},1;\tfrac{1}{2},1\rangle \tag{C.2.54}$$
$$= |\tfrac{1}{2},1;t=\tfrac{1}{2},m=\tfrac{3}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{1}{2},m=\tfrac{3}{2}|\tfrac{1}{2},1;\tfrac{1}{2},1\rangle \tag{C.2.55}$$
$$+ |\tfrac{1}{2},1;t=\tfrac{3}{2},m=\tfrac{3}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{3}{2},m=\tfrac{3}{2}|\tfrac{1}{2},1;\tfrac{1}{2},1\rangle. \tag{C.2.56}$$

C.2.7 Neutron Σ^+ case

For neutron $Sigma^+$ we have

$$m_1 = -1/2, \tag{C.2.57}$$
$$m_2 = 1, \tag{C.2.58}$$
$$m = m_1 + m_2 = 1/2. \tag{C.2.59}$$

The wavefunction becomes

$$|\tfrac{1}{2},1;-\tfrac{1}{2},1\rangle = \sum_t |\tfrac{1}{2},1;t,m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t,m=\tfrac{1}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},1\rangle \tag{C.2.60}$$
$$= |\tfrac{1}{2},1;t=\tfrac{1}{2},m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{1}{2},m=\tfrac{1}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},1\rangle \tag{C.2.61}$$
$$+ |\tfrac{1}{2},1;t=\tfrac{3}{2},m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{3}{2},m=\tfrac{1}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},1\rangle. \tag{C.2.62}$$

C.2.8 Proton Σ^0 case

For proton Σ^0 we have

$$m_1 = 1/2, \tag{C.2.63}$$
$$m_2 = 0, \tag{C.2.64}$$
$$m = m_1 + m_2 = 1/2. \tag{C.2.65}$$

The wavefunction becomes

$$|\tfrac{1}{2},1;\tfrac{1}{2},0\rangle = \sum_t |\tfrac{1}{2},1;t,m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t,m=\tfrac{1}{2}|\tfrac{1}{2},1;\tfrac{1}{2},0\rangle \tag{C.2.66}$$
$$= |\tfrac{1}{2},1;t=\tfrac{1}{2},m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{1}{2},m=\tfrac{1}{2}|\tfrac{1}{2},1;\tfrac{1}{2},0\rangle \tag{C.2.67}$$
$$+ |\tfrac{1}{2},1;t=\tfrac{3}{2},m=\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{3}{2},m=\tfrac{1}{2}|\tfrac{1}{2},1;\tfrac{1}{2},0\rangle. \tag{C.2.68}$$

C.2.9 Neutron Σ^0 case

For neutron Σ^0 we have

$$m_1 = -1/2, \tag{C.2.69}$$
$$m_2 = 0, \tag{C.2.70}$$
$$m = m_1 + m_2 = -1/2. \tag{C.2.71}$$

The wavefunction becomes

$$|\tfrac{1}{2},1;-\tfrac{1}{2},0\rangle = \sum_t |\tfrac{1}{2},1;t,m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t,m=-\tfrac{1}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},0\rangle \tag{C.2.72}$$

$$= |\tfrac{1}{2},1;t=\tfrac{1}{2},m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{1}{2},m=-\tfrac{1}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},0\rangle \tag{C.2.73}$$

$$+ |\tfrac{1}{2},1;t=\tfrac{3}{2},m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{3}{2},m=-\tfrac{1}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},0\rangle. \tag{C.2.74}$$

C.2.10 Proton Σ^- case

For proton $Sigma^-$ we have

$$m_1 = 1/2, \tag{C.2.75}$$

$$m_2 = -1, \tag{C.2.76}$$

$$m = m_1 + m_2 = -1/2. \tag{C.2.77}$$

The wavefunction becomes

$$|\tfrac{1}{2},1;\tfrac{1}{2},-1\rangle = \sum_t |\tfrac{1}{2},1;t,m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t,m=-\tfrac{1}{2}|\tfrac{1}{2},1;\tfrac{1}{2},-1\rangle \tag{C.2.78}$$

$$= |\tfrac{1}{2},1;t=\tfrac{1}{2},m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{1}{2},m=-\tfrac{1}{2}|\tfrac{1}{2},1;\tfrac{1}{2},-1\rangle \tag{C.2.79}$$

$$+ |\tfrac{1}{2},1;t=\tfrac{3}{2},m=-\tfrac{1}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{3}{2},m=-\tfrac{1}{2}|\tfrac{1}{2},1;\tfrac{1}{2},-1\rangle. \tag{C.2.80}$$

C.2.11 Neutron Σ^- case

For neutron Σ^- we have

$$m_1 = -1/2, \tag{C.2.81}$$

$$m_2 = -1, \tag{C.2.82}$$

$$m = m_1 + m_2 = -3/2. \tag{C.2.83}$$

The wavefunction becomes

$$|\tfrac{1}{2},1;-\tfrac{1}{2},-1\rangle = \sum_t |\tfrac{1}{2},1;t,m=-\tfrac{3}{2}\rangle\langle\tfrac{1}{2},1;t,m=-\tfrac{3}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},-1\rangle \tag{C.2.84}$$

$$= |\tfrac{1}{2},1;t=\tfrac{1}{2},m=-\tfrac{3}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{1}{2},m=-\tfrac{3}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},-1\rangle \tag{C.2.85}$$

$$+ |\tfrac{1}{2},1;t=\tfrac{3}{2},m=-\tfrac{3}{2}\rangle\langle\tfrac{1}{2},1;t=\tfrac{3}{2},m=-\tfrac{3}{2}|\tfrac{1}{2},1;-\tfrac{1}{2},-1\rangle. \tag{C.2.86}$$

D Schrödinger equation

D.1 Two-body Schrödinger equation in the relative and center of mass(CoM) frame

We begin with the time-independent Schrödinger equation on operator form

$$\left(\frac{\hat{\mathbf{p}}_1^2}{2\mathrm{m}_1} + \frac{\hat{\mathbf{p}}_2^2}{2\mathrm{m}_2} + \hat{V}(\hat{\mathbf{r}}_1,\hat{\mathbf{r}}_2)\right)|\Psi\rangle \quad = \quad \mathrm{E}\,|\Psi\rangle, \tag{D.1.1}$$

where $\hat{\mathbf{p}}_i$ is the momentum operator for particle i, $\hat{\mathbf{r}}_i$ is the position operator and m_i is the mass. $\hat{V}(\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2)$ is the two-body interaction operator which is a function of the two position operators. $|\Psi\rangle$ is the two-body wavefunction and E is the energy of the two-body state. To transform this equation to relative and CoM coordinates, we will need the definition of the relative and CoM coordinates for both position and momentum operators expressed as functions of their single-particle analogues.

$$\begin{aligned}
\hat{\mathbf{P}} &= \hat{\mathbf{p}}_1 + \hat{\mathbf{p}}_2, \\
\hat{\mathbf{p}} &= \beta\hat{\mathbf{p}}_1 - \alpha\hat{\mathbf{p}}_2, \\
\hat{\mathbf{r}} &= \hat{\mathbf{r}}_1 - \hat{\mathbf{r}}_2, \\
\hat{\mathbf{R}} &= \alpha\hat{\mathbf{r}}_1 + \beta\hat{\mathbf{r}}_2,
\end{aligned}$$

where $\alpha = \frac{m_1}{M}$, $\beta = \frac{m_2}{M}$ and $\mathrm{M} = m_1 + m_2$ is the total mass. The single particle momentum operators can now be written as

$$\hat{\mathbf{p}}_1 = \alpha\hat{\mathbf{P}} + \hat{\mathbf{p}}, \tag{D.1.2}$$

$$\hat{\mathbf{p}}_2 = \beta\hat{\mathbf{P}} - \hat{\mathbf{p}}. \tag{D.1.3}$$

We will also need the definition of the reduced mass of the two particles

$$m = \frac{m_1 m_2}{M}.$$

If we assume a central symmetric potential, the interaction operator can be written

$$\hat{V}(\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2) = \hat{V}(\hat{r}) = \hat{V}(\hat{r}), \tag{D.1.4}$$

and the Schrödinger equation can be separated into two operator equations that can be solved separately. Inserting D.1.2, D.1.3 and D.1.4 into D.1.1, we get

$$\left(\frac{(\alpha\hat{\mathbf{P}} + \hat{\mathbf{p}})^2}{2m_1} + \frac{(\beta\hat{\mathbf{P}} - \hat{\mathbf{p}})^2}{2m_2} + \hat{V}(|\hat{\mathbf{r}}|)\right)|\Psi\rangle = E\,|\Psi\rangle.$$

The momentum part can be rewritten as

$$\frac{(\alpha\hat{\mathbf{P}} + \hat{\mathbf{p}})^2}{2m_1} + \frac{(\beta\hat{\mathbf{P}} - \hat{\mathbf{p}})^2}{2m_2} = \frac{\hat{\mathbf{P}}^2}{2M} + \frac{\hat{\mathbf{p}}^2}{2m},$$

which gives the Schrödinger equation on operator form in the relative and CoM frame

$$\left(\frac{\hat{\mathbf{P}}^2}{2M} + \frac{\hat{\mathbf{p}}^2}{2m} + \hat{V}(\hat{r})\right)|\Psi\rangle = \mathrm{E}\,|\Psi\rangle. \tag{D.1.5}$$

If we assume that the wavefunction is separable in relative and CoM coordinates so that

$$|\Psi\rangle = |\psi(\mathbf{r}); \phi(\mathbf{R})\rangle = |\psi(\mathbf{r})\rangle|\phi(\mathbf{R})\rangle,$$

we can rewrite the two-body Schrödinger equation D.1.5 as two separate equations using standard separation of variables tecniques

$$\frac{\hat{\mathbf{P}}^2}{2M}|\phi(\mathbf{R})\rangle = \epsilon_{\mathrm{R}}\,|\phi(\mathbf{R})\rangle, \tag{D.1.6}$$

$$\left(\frac{\hat{\mathbf{p}}^2}{2m} + \hat{V}(\hat{r})\right)|\psi(\mathbf{r})\rangle = \epsilon_{\mathrm{r}}\,|\psi(\mathbf{r})\rangle, \tag{D.1.7}$$

where $E = \epsilon_r + \epsilon_R$.

D.2 Two-body Schrödinger equation in CoM coordinates in momentum space

Using the Schrödinger equation on operator form in CoM coordinates (D.1.6) and projecting it onto momentum space by multiplying from the left with a plane wave state $\langle \mathbf{P}|$, we get the Schrödinger equation in CoM coordinates in momentum space

$$\begin{aligned}
\langle \mathbf{P}|\frac{\hat{\mathbf{P}}^2}{2M}|\phi(\mathbf{R})\rangle &= \epsilon_{\mathrm{R}}\ \langle \mathbf{P}|\phi(\mathbf{R})\rangle, \\
\frac{\mathbf{P}^2}{2M}\langle \mathbf{P}|\phi(\mathbf{R})\rangle &= \epsilon_{\mathrm{R}}\ \langle \mathbf{P}|\phi(\mathbf{R})\rangle, \\
\frac{\mathbf{P}^2}{2M}\phi(\mathbf{P}) &= \epsilon_{\mathrm{R}}\ \phi(\mathbf{P}),
\end{aligned} \tag{D.2.1}$$

where $\mathbf{P}$ is the solution to the eigenvalue equation

$$\hat{\mathbf{P}}|\mathbf{P}\rangle = \mathbf{P}|\mathbf{P}\rangle,$$

and we have introduced

$$\phi(\mathbf{P}) = \langle \mathbf{P}|\phi(\mathbf{R})\rangle.$$

D.3 Two-body Schrödinger equation in relative coordinates in momentum space

Using the Schrödinger equation on operator form in relative coordinates (D.1.7) and projecting it onto momentum space by multiplying from the left with a plane wave state $\langle \mathbf{p}|$, we get the Schrödinger equation in relative coordinates in momentum space

$$\begin{aligned}
\langle \mathbf{p}|\left(\frac{\hat{\mathbf{p}}^2}{2m} + \hat{V}(\hat{r})\right)|\psi(\mathbf{r})\rangle &= \epsilon_{\mathrm{r}}\ \langle \mathbf{p}|\psi(\mathbf{r})\rangle, \\
\frac{\mathbf{p}^2}{2m}\langle \mathbf{p}|\psi(\mathbf{r})\rangle + \langle \mathbf{p}|\hat{V}(\hat{r})|\psi(\mathbf{r})\rangle &= \epsilon_{\mathrm{r}}\ \langle \mathbf{p}|\psi(\mathbf{r})\rangle, \\
\frac{\mathbf{p}^2}{2m}\psi(\mathbf{p}) + \int \mathrm{d}^3\mathbf{p}'\langle \mathbf{p}|\hat{V}(\hat{r})|\mathbf{p}'\rangle\psi(\mathbf{p}') &= \epsilon_{\mathrm{r}}\ \psi(\mathbf{p}),
\end{aligned} \tag{D.3.1}$$

where $\mathbf{p}$ is the solution to the eigenvalue equation

$$\hat{\mathbf{p}}|\mathbf{p}\rangle = \mathbf{p}|\mathbf{p}\rangle,$$

and we have introduced the momentum-space wavefunction

$$\psi(\mathbf{p}) = \langle \mathbf{p}|\phi(\mathbf{r})\rangle.$$

We have also used the completness relation for the plane-wave states

$$\int \mathrm{d}^3\mathbf{k}\langle \mathbf{k}|\mathbf{k}\rangle = \mathbf{1}. \tag{D.3.2}$$

D.4 Partial wave expansion in momentum-space of the relative Schrödinger equation

We begin with the relative Schrödinger equation in operator form (D.1.7) and multiply from the left with a momentum eigenstate $\langle \mathbf{p}|$. As before $\hbar = c = 1$, $\hat{\mathbf{p}}$ is the relative momentum operator, m is the reduced mass of the two-body state and $\hat{V}(\hat{r})$ is the two-body interaction operator and a function of the relative position operator alone. The interaction operator could be a non-local operator, so we preserve the angular dependence of the position operator. We have

$$\begin{aligned}
\langle \mathbf{p}| \left(\frac{\hat{\mathbf{p}}^2}{2m} + \hat{V}(\hat{\mathbf{r}}) \right) |\psi\rangle &= \epsilon_{\mathrm{r}} \, \langle \mathbf{p}|\psi\rangle, \\
\frac{\mathbf{p}^2}{2m} \langle \mathbf{p}|\psi\rangle + \langle \mathbf{p}|\hat{V}(\hat{\mathbf{r}})|\psi\rangle &= \epsilon_{\mathrm{r}} \, \langle \mathbf{p}|\psi\rangle, \\
\frac{\mathbf{p}^2}{2m} \langle \mathbf{p}|\psi\rangle + \int \mathrm{d}^3\mathbf{k} \, \langle \mathbf{p}|\hat{V}(\hat{\mathbf{r}})|\mathbf{k}\rangle \langle \mathbf{k}|\psi\rangle &= \epsilon_{\mathrm{r}} \, \langle \mathbf{p}|\psi\rangle, \qquad \text{(D.4.1)}
\end{aligned}$$

where we have used the completeness relation for momentum eigenstates from (D.3.2).

First we find the interaction in the relative partial wave basis for a local potential

$$\begin{aligned}
\langle \mathbf{p}|\hat{V}(\hat{\mathbf{r}})|\mathbf{k}\rangle &= \int \mathrm{d}^3\mathbf{r} \int \mathrm{d}^3\mathbf{r}' \, \langle \mathbf{p}|\mathbf{r}\rangle \langle \mathbf{r}|\hat{\mathbf{V}}(\hat{\mathbf{r}})|\mathbf{r}'\rangle \langle \mathbf{r}'|\mathbf{k}\rangle \\
&= \frac{1}{(2\pi)^3} \int \mathrm{d}^3\mathbf{r} \, e^{-i\mathbf{p}\cdot\mathbf{r}} V(\mathbf{r}) e^{i\mathbf{k}\cdot\mathbf{r}}, \qquad \text{(D.4.2)}
\end{aligned}$$

where we have used the definitions of the transformation functions from the position to momentum representation and vice-versa

$$\begin{aligned}
\langle \mathbf{p}|\mathbf{r}\rangle &= \frac{1}{(2\pi)^{\frac{3}{2}}} e^{-i\mathbf{p}\cdot\mathbf{r}}, \qquad \text{(D.4.3)} \\
\langle \mathbf{r}|\mathbf{p}\rangle &= \frac{1}{(2\pi)^{\frac{3}{2}}} e^{i\mathbf{p}\cdot\mathbf{r}}. \qquad \text{(D.4.4)}
\end{aligned}$$

Further, we expand the exponential function in the Bauer series where $e^{i\mathbf{p}\cdot\mathbf{r}}$ kan be expressed as a sum of Legendre polynomials and spherical Bessel functions,

$$e^{i\mathbf{p}\cdot\mathbf{r}} = \sum_{l=0}^{\infty} (2l+1) i^l j_l(pr) P_l(\Omega_{p,r}), \qquad \text{(D.4.5)}$$

where the spherical Bessel functions $j_l(pr)$ depends on the radial part of the momentum and position vector and is defined in appendix A.1, while the Legendre polynomials depends on $\Omega_{p,r} = \frac{\mathbf{p}\cdot\mathbf{r}}{|\mathbf{p}||\mathbf{r}|}$, which is the cosine of the angle between $\mathbf{p}$ and $\mathbf{r}$. We have also introduced the orbital momentum l.

Inserting D.4.5 into D.4.2 we get

$$\begin{aligned}
&\langle\mathbf{p}|\hat{\mathbf{V}}(\hat{\mathbf{r}})|\mathbf{k}\rangle = \\
&\frac{1}{(2\pi)^3}\int \mathrm{d}^3\mathbf{r}\sum_{l=0}^{\infty}(2l+1)i^{-l}j_l(pr)P_l(\Omega_{p,r})V(\mathbf{r})\sum_{l'=0}^{\infty}(2l'+1)i^{-l'}j_{l'}(kr)P_{l'}(\Omega_{k,r}) \\
= \;& \frac{1}{(2\pi)^3}\sum_{l,l'}^{\infty}(2l'+1)(2l+1)i^{l'-l}\int_0^{2\pi}\mathrm{d}\phi\int_0^{\pi}\mathrm{d}\theta\sin\theta P_{l'}(\Omega_{k,r})P_l(\Omega_{p,r})\times \\
&\int_0^{\infty}\mathrm{d}r r^2 V(\mathbf{r})j_l(pr)j_{l'}(kr) \\
= \;& \frac{1}{(2\pi)^2}\sum_{l,l'}^{\infty}(2l'+1)(2l+1)i^{l'-l}\int_{-1}^{1}\mathrm{d}(\cos\theta)P_{l'}(\Omega_{k,r})P_l(\Omega_{p,r})\langle pl|\hat{V}|kl'\rangle, \quad \text{(D.4.6)}
\end{aligned}$$

where we have defined the partial wave decomposition of a two-body interaction operator

$$\langle pl|\hat{V}|kl'\rangle = \int_0^{\infty}\mathrm{d}r r^2 V(r)j_l(pr)j_{l'}(kr). \tag{D.4.7}$$

We have the Legendre polynomial relation

$$\int_{-1}^{1}\mathrm{d}(\cos\theta)P_{l'}(\Omega_{k,r})P_l(\Omega_{p,r}) = \frac{2}{2l+1}P_l(\Omega_{p,k})\delta_{l,l'}, \tag{D.4.8}$$

which, for a centrally symmetric potential, gives us

$$\langle\mathbf{p}|\hat{\mathbf{V}}(\hat{\mathbf{r}})|\mathbf{k}\rangle = \frac{2}{(2\pi)^2}\sum_{l=0}^{\infty}(2l+1)P_l(\Omega_{p,k})\langle pl|\hat{V}|kl\rangle. \tag{D.4.9}$$

Inserting this back into equation D.4.1 and transforming to spherical coordinates we get [57]

$$\frac{p^2}{2m}\langle pl|\psi\rangle + \frac{2}{\pi}\int_0^{\infty}\mathrm{d}k k^2\langle pl|\hat{V}|kl\rangle\langle kl|\psi\rangle \quad = \quad \epsilon_{\mathrm{r}}\;\langle pl|\psi\rangle, \tag{D.4.10}$$

where we have placed the coordinate system so that $\mathbf{p}$ coincides with the z-axis and the fact that $P_0(x) = 1$ which gives

$$\int_{-1}^{1}P_l(x)P_0(x) = \frac{2}{2l+1}\delta_{l,0} = 2.$$

For a tensor coupled channel, the equation becomes

$$\sum_l\left(\frac{\mathbf{p}^2}{2m}\langle pl|\psi\rangle + \sum_{l'}\frac{2}{\pi}\int_0^{\infty}\mathrm{d}k k^2\langle pl|\hat{V}|kl'\rangle\langle kl'|\psi\rangle\right) = \sum_l \epsilon_{\mathrm{r}}\;\langle pl|\psi\rangle.$$

For a baryon coupled channel, the equation becomes

$$\sum_B\left(\frac{\mathbf{p}^2}{2m_B}\langle(B)pl|\psi\rangle + \frac{2}{\pi}\sum_{B'}\int_0^{\infty}\mathrm{d}k k^2\langle(B)pl|\hat{V}|(B')kl\rangle\langle(B')kl|\psi\rangle\right) = \sum_B \epsilon_{\mathrm{r}}\;\langle(B)pl|\psi\rangle,$$

where the sum over B means a sum over the various baryon couplings we have for a specific partial wave with definite isospspin projection.

For a channel with both tensor and baryon coupling, the equation becomes

$$\sum_{l,B}\left(\frac{\mathbf{p}^2}{2m_B}\langle(B)pl|\psi\rangle+\frac{2}{\pi}\sum_{l',B'}\int_0^\infty \mathrm{d}kk^2\langle(B)pl|\hat{V}|(B')kl'\rangle\langle(B')kl'|\psi\rangle\right)=\sum_{l,B}\epsilon_\mathrm{r}\,\langle(B)pl|\psi\rangle.$$

Even though formally the sum over angular momenta goes to infinity - for the strong interaction the total spin J is a conserved quantum number. In actual calculations the sum is limited to $l=J-1,J+1$ for a specific value of J.

We can write these equations in block matrix notation, anticipating the discretization of the equations. First we define

$$T_i=\frac{\mathbf{p}^2}{2m_{B_i}}, \tag{D.4.11}$$

$$V_{i,j}^{k,l}=\frac{2}{\pi}\int_0^\infty \mathrm{d}kk^2\langle(B_i)pl_k|\hat{V}|(B_j)kl_l\rangle, \tag{D.4.12}$$

$$\psi_i^j=\langle(B_i)pl_j|\psi\rangle. \tag{D.4.13}$$

For an uncoupled channel we have just

$$\left[T_1+V_{1,1}^{1,1}\right]\left[\psi_1^1\right]=\epsilon_\mathrm{r}\left[\psi_1^1\right]. \tag{D.4.14}$$

For a tensor coupled channel we have

$$\begin{bmatrix}T_1+V_{1,1}^{1,1} & V_{1,1}^{1,2}\\ V_{1,1}^{2,1} & T_1+V_{1,1}^{2,2}\end{bmatrix}\begin{bmatrix}\psi_1^1\\ \psi_1^2\end{bmatrix}=\epsilon_\mathrm{r}\begin{bmatrix}\psi_1^1\\ \psi_1^2\end{bmatrix}, \tag{D.4.15}$$

and similary for the tensor coupled channel where $s=0$ and $s=1$ couples to the orbital momentum to give the same total angular momentum.

For a baryon coupled channel with m possible couplings we have

$$\begin{bmatrix}T_1+V_{1,1}^{1,1} & \dots & V_{1,m}^{1,1}\\ \vdots & \ddots & \vdots\\ V_{m,1}^{1,1} & \dots & T_m+V_{m,m}^{1,1}\end{bmatrix}\begin{bmatrix}\psi_1^1\\ \vdots\\ \psi_m^1\end{bmatrix}=\epsilon_\mathrm{r}\begin{bmatrix}\psi_1^1\\ \vdots\\ \psi_m^1\end{bmatrix}, \tag{D.4.16}$$

and finally for a tensor and baryon coupled channel we have

$$\begin{bmatrix}T_1+V_{1,1}^{1,1} & \dots & V_{1,m}^{1,1} & V_{1,1}^{1,2} & \dots & V_{1,m}^{1,2}\\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ V_{m,1}^{1,1} & \dots & T_m+V_{m,m}^{1,1} & V_{m,1}^{1,2} & \dots & V_{m,m}^{1,2}\\ T_1+V_{1,1}^{2,1} & \dots & V_{1,m}^{2,1} & T_1+V_{1,1}^{2,2} & \dots & V_{1,m}^{2,2}\\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots\\ V_{m,1}^{2,1} & \dots & V_{m,m}^{2,1} & V_{m,1}^{2,2} & \dots & T_m+V_{m,m}^{2,2}\end{bmatrix}\begin{bmatrix}\psi_1^1\\ \vdots\\ \psi_m^1\\ \psi_1^2\\ \vdots\\ \psi_m^2\end{bmatrix}=\epsilon_\mathrm{r}\begin{bmatrix}\psi_1^1\\ \vdots\\ \psi_m^1\\ \psi_1^2\\ \vdots\\ \psi_m^2\end{bmatrix}. \tag{D.4.17}$$

E The Harmonic oscillator

E.1 The three-dimensional isotropic harmonic oscillator

This problem is best represented in spherical coordinates, where the eigenkets of the Hamiltonian are labelled by the quantum numbers n, l and m.

The Schrödinger equation reads

$$\hat{H}_{ho}|nlm\rangle = E_{nl}|nlm\rangle, \tag{E.1.1}$$

with the Hamiltonian

$$\hat{H}_{ho} = \frac{\hat{\mathbf{p}}^2}{2\mu} + 1/2\mu\omega^2\hat{r}^2, \tag{E.1.2}$$

and energy

$$E_{nl} = \hbar\omega\left(2n + l + \frac{3}{2}\right). \tag{E.1.3}$$

Here μ is the mass of the oscillator, while ω is the oscillator parameter.

In coordinate space the eigenfunctions are given by [47]

$$\begin{aligned}
\langle\mathbf{r}|nlm\rangle &= \psi_{nlm}(r,\theta,\phi) = R_{nl}(r)Y_{lm}(\theta,\phi),\\
R_{nl}(r) &= N_{nl}r^l e^{-\nu r^2} L_n^{(l+\frac{1}{2})}(2\nu r^2), && \text{(E.1.4)}\\
N_{nl} &= \sqrt{\sqrt{\frac{2\nu^3}{\pi}}\frac{2^{n+2l+3}n!r^l}{(2n+2l+1)!!}}, && \text{(E.1.5)}\\
&\mathit{and}\\
\nu &= \frac{\mu\omega}{2\hbar}, && \text{(E.1.6)}
\end{aligned}$$

where $L_n^{(l+\frac{1}{2})}(2\nu r^2)$ are the generalized Laguerre polynomials from appendix A.3.

In momentum space, the eigenfunctions are given by [47]

$$\langle\mathbf{p}|nlm\rangle = P_{nl}(k)Y_{lm}(\theta,\phi) = i^{-l}(-1)^n R_{nl}(k)Y_{lm}(\theta,\phi). \tag{E.1.7}$$

E.2 Kinetic energy operator in a three-dimensional harmonic oscillator basis

To calculate the kinetic energy operator, we need the following matrix elements

$$\begin{aligned}
\langle nlm|\hat{\mathbf{p}}^2|n'l'm'\rangle &= \iint \mathrm{d}^3\mathbf{p}\,\mathrm{d}^3\mathbf{p}'\,\langle nlm|\mathbf{p}\rangle\langle\mathbf{p}|\hat{\mathbf{p}}^{\mathbf{2}}|\mathbf{p}'\rangle\langle\mathbf{p}'|n'l'm'\rangle\\
&= \iint \mathrm{d}^3\mathbf{p}\,\mathrm{d}^3\mathbf{p}'\,p^2\delta(\mathbf{p}-\mathbf{p}')\langle nlm|\mathbf{p}\rangle\langle\mathbf{p}'|n'l'm'\rangle\\
&= \int \mathrm{d}^3\mathbf{p}\,p^2\langle nlm|\mathbf{p}\rangle\langle\mathbf{p}|n'l'm'\rangle. && \text{(E.2.1)}
\end{aligned}$$

We introduce spherical coordinates and the harmonic oscillator wave-function in momentum space

$$\langle \mathbf{p}|nlm\rangle = P_{nl}(p)Y_{lm}(\theta,\phi), \tag{E.2.2}$$

where $P_nl(p)$ is the radial wavefunction for a three-dimensional harmonic oscillator in momentum space introduced in appendix E.1 and $Y_{lm}(\theta,\phi)$ are the spherical harmonics. We have

$$\begin{aligned}
\langle nlm|\hat{\mathbf{p}}^2|n'l'm'\rangle &= \iint_\Omega \sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi\, Y^*_{lm}(\theta,\phi)Y_{l'm'}(\theta,\phi)\int_0^\infty \mathrm{d}p\,p^4 P^*_{nl}(p)P_{n'l'}(p) \\
&= \delta_{ll'}\delta_{mm'}\int_0^\infty \mathrm{d}p\,p^4 P^*_{nl}(p)P_{n'l'}(p), \\
\langle nlm|\hat{\mathbf{p}}^2|n'l'm'\rangle &= \int_0^\infty \mathrm{d}p\,p^4 P^*_{nl}(p)P_{n'l}(p),
\end{aligned} \tag{E.2.3}$$

where we have used the normalisation property of the spherical harmonics

$$\iint_\Omega \sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi\, Y^*_{lm}(\theta,\phi)Y_{l'm'}(\theta,\phi) = \delta_{ll'}\delta_{mm'}. \tag{E.2.4}$$

The kinetic energy operator $\hat{T} = \frac{\hat{\mathbf{p}}^2}{2\mu}$ in a three-dimensional harmonic hoscillator basis is now given by its matrix elements

$$\langle nlm|\hat{\mathbf{T}}|n'lm\rangle = \frac{1}{2\mu}\int_0^\infty \mathrm{d}p\,p^4 P_{nl}(p)P_{n'l}(p). \tag{E.2.5}$$

E.3 Coulomb interaction operator in a three-dimensional harmonic oscillator basis

To calculate the Coulomb interaction operator, we need the following matrix elements

$$\begin{aligned}
\langle nlm|\hat{\mathbf{r}}^{-1}|n'l'm'\rangle &= \iint \mathrm{d}^3\mathbf{r}\,\mathrm{d}^3\mathbf{r}'\,\langle nlm|\mathbf{r}\rangle\langle\mathbf{r}|\hat{\mathbf{r}}^{-1}|\mathbf{r}'\rangle\langle\mathbf{r}'|n'l'm'\rangle \\
&= \iint \mathrm{d}^3\mathbf{r}\,\mathrm{d}^3\mathbf{r}'\,\langle nlm|\mathbf{r}\rangle\mathbf{r}^{-1}\delta(\mathbf{r}-\mathbf{r}')\langle\mathbf{r}'|n'l'm'\rangle \\
&= \int \mathrm{d}^3\mathbf{r}\,\mathbf{r}^{-1}\langle nlm|\mathbf{r}\rangle\langle\mathbf{r}|n'l'm'\rangle.
\end{aligned}$$

We introduce spherical coordinates and the harmonic oscillator wave-function in coordinate space

$$\langle \mathbf{r}|nlm\rangle = R_{nl}(r)Y_{lm}(\theta,\phi), \tag{E.3.1}$$

where $R_{nl}(r)$ is the radial wavefunction for a three-dimensional harmonic oscillator in coordinate space introduced in appendix E.1 and $Y_{lm}(\theta,\phi)$ are the sperical harmonics. We obtain

$$\begin{aligned}
\langle nlm|\hat{\mathbf{r}}^{-1}|n'l'm'\rangle &= \iint_\Omega \sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi\, Y^*_{lm}(\theta,\phi)Y_{l'm'}(\theta,\phi)\int_0^\infty \mathrm{d}r\,rR^*_{nl}(r)R_{n'l'}(r) \\
&= \delta_{ll'}\delta_{mm'}\int_0^\infty \mathrm{d}r\,rR^*_{nl}(r)R_{n'l'}(r), \\
\langle nlm|\hat{\mathbf{r}}^{-1}|n'l'm'\rangle &= \int_0^\infty \mathrm{d}r\,rR_{nl}(r)R_{n'l}(r),
\end{aligned} \tag{E.3.2}$$

where we have used the normalisation property of the spherical harmonics

$$\iint_{\Omega} \sin\theta \,\mathrm{d}\theta \,\mathrm{d}\phi \, Y^*_{lm}(\theta,\phi) Y_{l'm'}(\theta,\phi) = \delta_{ll'}\delta_{mm'}. \tag{E.3.3}$$

The Coulomb interaction operator $\hat{V}_C = \frac{q_1 q_2}{4\pi\epsilon_0 \hat{\mathbf{r}}}$ in a 3D harmonic oscillator basis is now given by it's matrix elements

$$\langle nlm|\hat{V}_C|n'lm\rangle = \frac{q_1 q_2}{4\pi\epsilon_0} \int_0^\infty \mathrm{d}r \, r R_{nl}(r) R_{n'l}(r), \tag{E.3.4}$$

where q_i is the charge of the i'th particle in units of e and ϵ_0 is the permitivity in free space.

F Perturbation theory

We start with the Schrödinger equation

$$\hat{H}|\Psi\rangle = E|\Psi\rangle, \tag{F.0.5}$$

where $\hat{H}$ is the many-body Hamiltonian or energy operator, E is the energy eigenvalue of the Hamiltonian and $|\Psi\rangle$ is the many-body wavefunction expressed in an arbitrary basis

$$|\Psi\rangle = \sum_{i=1}^{\infty} \alpha_i |\Phi_i\rangle. \tag{F.0.6}$$

We now choose a so-called model space, defined by the projection operators

$$\hat{P} = \sum_{i=1}^{d} |\Phi_i\rangle\langle\Phi_i|, \tag{F.0.7}$$

$$\hat{Q} = \sum_{i=d+1}^{\infty} |\Phi_i\rangle\langle\Phi_i|, \tag{F.0.8}$$

with the completeness property

$$\hat{P} + \hat{Q} = \hat{1}, \tag{F.0.9}$$

and the usual properties of projection operators

$$\hat{P}^2 = \hat{P}, \tag{F.0.10}$$

$$\hat{Q}^2 = \hat{Q}, \tag{F.0.11}$$

$$\hat{P}\hat{Q} = \hat{Q}\hat{P} = 0. \tag{F.0.12}$$

(F.0.13)

We can now write the wavefunction as

$$|\Psi\rangle = |\Psi_D\rangle + \hat{Q}|\Psi\rangle, \tag{F.0.14}$$

where

$$|\Psi_D\rangle = \hat{P}|\Psi\rangle. \tag{F.0.15}$$

We split the Hamiltonian into an unperturbed part $\hat{H}_0$, for which we have an exact solution, and a perturbed part $\hat{V}$, which we assume is small compared to the unperturbed part.

$$\hat{H} = \hat{H}_0 + \hat{V}. \tag{F.0.16}$$

We can derive a perturbation series by writing the Schrödinger equation as

$$\left(\omega - \hat{H}_0\right)|\Psi\rangle = \left(\omega - E + \hat{V}\right)|\Psi\rangle, \tag{F.0.17}$$

where ω is a scalar quantity. By assuming that the inverse exist

$$\left(\omega - \hat{H}_0\right)^{-1} \equiv \frac{1}{\omega - \hat{H}_0}, \tag{F.0.18}$$

we can write the Schrödinger equation as

$$|\Psi\rangle = |\Psi_D\rangle + \frac{\hat{Q}}{\omega - \hat{H}_0}\left(\omega - E + \hat{V}\right)|\Psi\rangle, \tag{F.0.19}$$

where we have used equation (F.0.15) and the properties of the projection operators. We obtain a perturbation expansion for the wavefunction

$$|\Psi\rangle = \sum_{n=0}^{\infty}\left(\frac{\hat{Q}}{\omega - \hat{H}_0}\left(\omega - E + \hat{V}\right)\right)^n |\Psi_D\rangle. \tag{F.0.20}$$

By choosing $\omega = E$ we get what is called a Brillouin-Wigner type perturbation series, where the expansion depends on the total energy of the system.

We can also choose $\omega = W$, where W is defined as the energy eigenstate of the unperturbed Hamiltonian on the model-space wavefunction,

$$\hat{H}_0|\Psi_D\rangle = W|\Psi_D\rangle. \tag{F.0.21}$$

We now get a Rayleigh-Schrödinger type perturbation series

$$|\Psi\rangle = \sum_{n=0}^{\infty}\left(\frac{\hat{Q}}{\omega - \hat{H}_0}\left(\hat{V} - \Delta E\right)\right)^n |\Psi_D\rangle, \tag{F.0.22}$$

where we have defined $\Delta E = E - W$ as the difference between the exact and unperturbed energy.

We obtain a perturbaton series for ΔE for the Brillouin-Wigner perturbation theory

$$\Delta E = \sum_{n=0}^{\infty}\langle\Psi_D|\hat{V}\left(\frac{\hat{Q}}{E - \hat{H}_0}\hat{V}\right)^n |\Psi_D\rangle, \tag{F.0.23}$$

while Rayleigh-Schrødinger perturbation theory gives

$$\Delta E = \sum_{n=0}^{\infty}\langle\Psi_D|\hat{V}\left(\frac{\hat{Q}}{W - \hat{H}_0}\left(\hat{V} - \Delta E\right)\right)^n |\Psi_D\rangle. \tag{F.0.24}$$

G Api documentation

G.1 Module common

G.1.1 Functions

check_file(*filename*)

Check if file is safe to read.

Parameters

`filename`: Filename to check.
(type=String)

Return Value

True if file exists and is a file, False if not

debug(*s1*, *s2*, *debug*=`False`)

Print debugging information if DEBUG is True.
If multiline string, the preamble is printed before every line.
Typical usage:

```
>>> debug('preamble', 'message', True|False)
preamble:: message
```

Parameters

`s1`: Name of the caller.
(type=string)

`s2`: Message to be printed.
(type=object that have a string representation)

`debug`: Should message be printed or not. If not supplied use this modules DEBUG variable.
(type=Boolean)

dirname_check(*dirname*)

Return the basename of a possible exact path.

generate_configs(*filename*)

Generate multiple configurations in the format: key=value from a file where multiple values are specified for each key: key=value1,value2, value3 where each configuration contains only one value pr. file. Used for running variations with different parameter sets. Returns a tuple containing a dictionary of all common variables, and a list of all variations of the remaining variables.

Parameters

`filename`:	Original file with multiple values *(type=String)*

Return Value

dictionary, list

generate_variations(*variations*, *key*, *items*)

Generate permutations of an existing set of configurations, with a new key and values for this key. Returns a list of variations that couples all the new values to all the old configurations.

Parameters

`variations`:	List of old configurations. *(type=List of dictionaries)*
`key`:	New key to be added to old configurations *(type=String)*
`items`:	list of values for this key to be coupled with all old configurations. *(type=list of strings)*

Return Value

A new list of configuration dictionaries

get_dirserial(*dirname*)

Iterates over the files in the same directory as the parameter and find a new serial for this filename. Returns the next available serial for this filename.

Parameters

`dirname`:	Path to a filename to generate a serial for. *(type=String)*

Return Value

The next available serial for this filename.

read_config(*filename*)

Reads a configuration file in the format: key=value populates a Python dictionary with the keys and corresponding values.

Parameters

`filename`: Filename of the config file.
(type=String)

Return Value

Dictionary with all elements in the configuration file.

stringify(*s*)

Convert a Fortran90 character array into a Python sring object.

Parameters

`s`: The fortran string to convert.
(type=Fortran character array)

Return Value

A python string object.

triag(*i*, *j*, *k*)

Return True if the to spin variables i and j can add up to k.

Parameters

`i`: spin variable
(type=int)

`j`: Spin variable
(type=int)

`k`: Sum of i and j
(type=int)

Return Value

True if valid combination. False otherwise.

G.1.2 Variables

Name	Description
cname	**Value:** `'common'` *(type=`str`)*
DEBUG	**Value:** `False` *(type=`bool`)*

G.2 Module debug

G.2.1 Functions

debug(*s1*, *s2*, *debug*=`False`)

Print debugging information if DEBUG is True.
If multiline string, the preamble is printed before every line.
Typical usage:

```
>>> debug('preamble', 'message')
preamble:: message
```

Parameters

`s1`: Name of the caller.
(type=string)

`s2`: Message to be printed.
(type=object that have a string representation)

G.2.2 Variables

Name	Description
DEBUG	**Value:** `False` *(type=`bool`)*

G.3 Module gui

This modules defines standard gui classes that are used often. The main components are the myFrame class which all new Frames inherits and the three Button classes - Quit, Plot and Print.

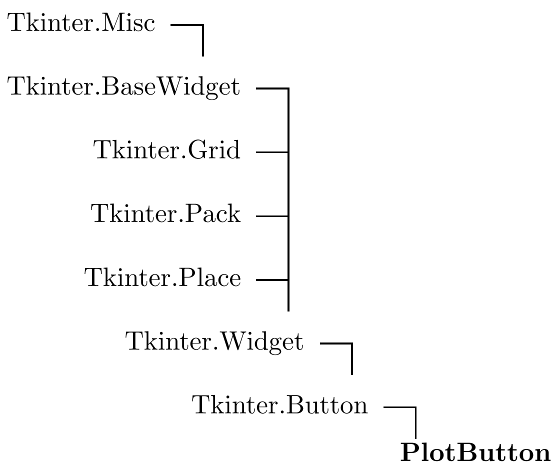
G.3.1 Classes

- **myFrame**: All Tkinter frames inherit this class.
(Section G.7, p. 121)
- **PlotButton**: Defines a Tkinter.Button with classname plotButton.
(Section G.4, p. 117)
- **PrintButton**: Defines a Tkinter.Button with classname printButton.
(Section G.5, p. 118)
- **QuitButton**: Defines a Tkinter.Button with classname quitButton.
(Section G.6, p. 120)

G.3.2 Variables

Name	Description
cname	**Value:** `'gui'` *(type=`str`)*
DEBUG	**Value:** `False` *(type=`bool`)*

G.4 Class gui.PlotButton



Defines a Tkinter.Button with classname plotButton. The appearence and text is controlled by a resource file. Entries in the resource file may be specified as:

*Button*activebackground: Grey *Button*background: Grey *Button*highlightcolor: Grey *Button*foreground: Black *Button*font: -Adobe-Helvetica-Bold-R-Normal–*-120-*-*-*-*-*-* *Button*width: 8 *Button*height: 1 *plotButton*text: Plot

G.4.1 Methods

__init__(*self*, *master*, ***kw*)
Overrides: Tkinter.Button.__init__

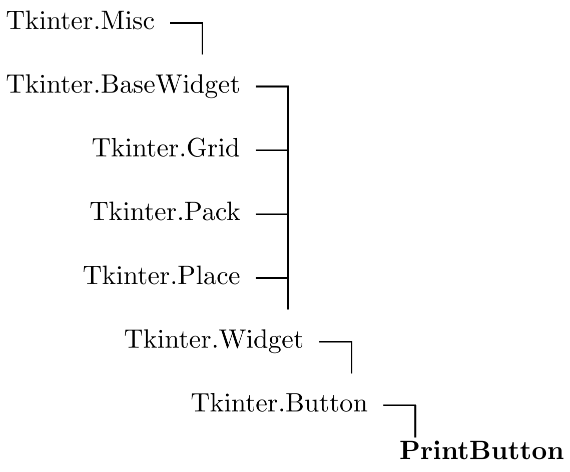
Inherited from BaseWidget: destroy
Inherited from Button: flash, invoke, tkButtonDown, tkButtonEnter, tkButtonInvoke, tkButtonLeave, tkButtonUp
Inherited from Grid: grid, grid_configure, grid_forget, grid_info, grid_remove
Inherited from Misc: __getitem__, __setitem__, __str__, _nametowidget, after, after_cancel, after_idle, bbox, bell, bind, bind_all, bind_class, bindtags, cget, clipboard_append, clipboard_clear, colormodel, columnconfigure, config, configure, deletecommand, event_add, event_delete, event_generate, event_info, focus, focus_displayof, focus_force, focus_get, focus_lastfor, focus_set, getboolean, getvar, grab_current, grab_release, grab_set, grab_set_global, grab_status, grid_bbox, grid_columnconfigure, grid_location, grid_propagate, grid_rowconfigure, grid_size, grid_slaves, image_names, image_types, keys, lift, location, lower, mainloop, nametowidget, option_add, option_clear, option_get, option_readfile, pack_propagate, pack_slaves, place_slaves, propagate, quit, rowconfigure, selection_clear, selection_get, selection_handle, selection_own, selection_own_get, send, setvar, size, slaves, tk_bisque, tk_focusFollowsMouse, tk_focusNext, tk_focusPrev, tk_menuBar, tk_setPalette, tk_strictMotif, tkraise, unbind, unbind_all, unbind_class, update, update_idletasks, wait_variable, wait_visibility, wait_window, waitvar, winfo_atom, winfo_atomname, winfo_cells,

winfo_children, winfo_class, winfo_colormapfull, winfo_containing, winfo_depth, winfo_exists, winfo_fpixels, winfo_geometry, winfo_height, winfo_id, winfo_interps, winfo_ismapped, winfo_manager, winfo_name, winfo_parent, winfo_pathname, winfo_pixels, winfo_pointerx, winfo_pointerxy, winfo_pointery, winfo_reqheight, winfo_reqwidth, winfo_rgb, winfo_rootx, winfo_rooty, winfo_screen, winfo_screencells, winfo_screendepth, winfo_screenheight, winfo_screenmmheight, winfo_screenmmwidth, winfo_screenvisual, winfo_screenwidth, winfo_server, winfo_toplevel, winfo_viewable, winfo_visual, winfo_visualid, winfo_visualsavailable, winfo_vrootheight, winfo_vrootwidth, winfo_vrootx, winfo_vrooty, winfo_width, winfo_x, winfo_y
Inherited from Pack: forget, info, pack, pack_configure, pack_forget, pack_info
Inherited from Place: place, place_configure, place_forget, place_info

G.4.2 Class Variables

Name	Description
Inherited from Misc: _noarg_ *()*	

G.5 Class gui.PrintButton



Defines a Tkinter.Button with classname printButton. The appearence and text is controlled by a resource file. Entries in the resource file may be specified as:

*Button*activebackground: Grey *Button*background: Grey *Button*highlightcolor: Grey *Button*foreground: Black *Button*font: -Adobe-Helvetica-Bold-R-Normal–*-120-*-*-*-*-*-* *Button*width: 8 *Button*height: 1 *printButton*text: Print

G.5.1 Methods

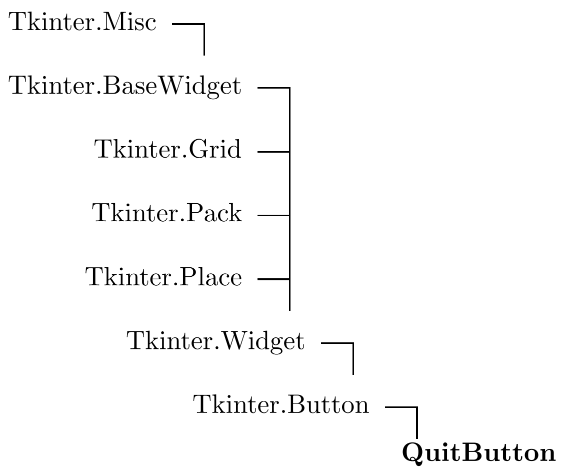
__init__(*self*, *master*, ***kw*) Overrides: Tkinter.Button.__init__

Inherited from BaseWidget: destroy
Inherited from Button: flash, invoke, tkButtonDown, tkButtonEnter, tkButtonInvoke, tkButtonLeave, tkButtonUp
Inherited from Grid: grid, grid_configure, grid_forget, grid_info, grid_remove
Inherited from Misc: __getitem__, __setitem__, __str__, _nametowidget, after, after_cancel, after_idle, bbox, bell, bind, bind_all, bind_class, bindtags, cget, clipboard_append, clipboard_clear, colormodel, columnconfigure, config, configure, deletecommand, event_add, event_delete, event_generate, event_info, focus, focus_displayof, focus_force, focus_get, focus_lastfor, focus_set, getboolean, getvar, grab_current, grab_release, grab_set, grab_set_global, grab_status, grid_bbox, grid_columnconfigure, grid_location, grid_propagate, grid_rowconfigure, grid_size, grid_slaves, image_names, image_types, keys, lift, location, lower, mainloop, nametowidget, option_add, option_clear, option_get, option_readfile, pack_propagate, pack_slaves, place_slaves, propagate, quit, rowconfigure, selection_clear, selection_get, selection_handle, selection_own, selection_own_get, send, setvar, size, slaves, tk_bisque, tk_focusFollowsMouse, tk_focusNext, tk_focusPrev, tk_menuBar, tk_setPalette, tk_strictMotif, tkraise, unbind, unbind_all, unbind_class, update, update_idletasks, wait_variable, wait_visibility, wait_window, waitvar, winfo_atom, winfo_atomname, winfo_cells, winfo_children, winfo_class, winfo_colormapfull, winfo_containing, winfo_depth, winfo_exists, winfo_fpixels, winfo_geometry, winfo_height, winfo_id, winfo_interps, winfo_ismapped, winfo_manager, winfo_name, winfo_parent, winfo_pathname, winfo_pixels, winfo_pointerx, winfo_pointerxy, winfo_pointery, winfo_reqheight, winfo_reqwidth, winfo_rgb, winfo_rootx, winfo_rooty, winfo_screen, winfo_screencells, winfo_screendepth, winfo_screenheight, winfo_screenmmheight, winfo_screenmmwidth, winfo_screenvisual, winfo_screenwidth, winfo_server, winfo_toplevel, winfo_viewable, winfo_visual, winfo_visualid, winfo_visualsavailable, winfo_vrootheight, winfo_vrootwidth, winfo_vrootx, winfo_vrooty, winfo_width, winfo_x, winfo_y
Inherited from Pack: forget, info, pack, pack_configure, pack_forget, pack_info
Inherited from Place: place, place_configure, place_forget, place_info

G.5.2 Class Variables

Name	Description
Inherited from Misc: _noarg_ *()*	

G.6 Class gui.QuitButton



Defines a Tkinter.Button with classname quitButton. The appearence and text is controlled by a resource file. Entries in the resource file may be specified as: *Button*activebackground: Grey *Button*background: Grey *Button*highlightcolor: Grey *Button*foreground: Black *Button*font: -Adobe-Helvetica-Bold-R-Normal–*-120-*-*-*-*-*-* *Button*width: 8 *Button*height: 1 *quitButton*text: Quit

G.6.1 Methods

__init__(*self*, *master*, ***kw*)

Constructor.

Parameters

`self:`	Reference to this instance of the class. *(type=Instance of QuitButton)*
`master:`	Parent frame *(type=Tkinter.Frame)*
`kw:`	Keyword arguments *(type=dictionary)*

Overrides: Tkinter.Button.__init__

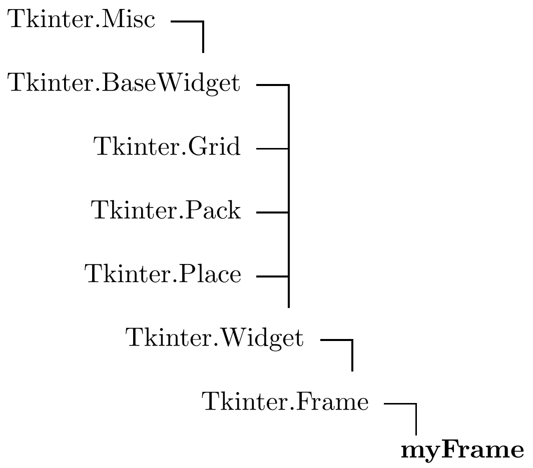
Inherited from BaseWidget: destroy
Inherited from Button: flash, invoke, tkButtonDown, tkButtonEnter, tkButtonInvoke, tkButtonLeave, tkButtonUp
Inherited from Grid: grid, grid_configure, grid_forget, grid_info, grid_remove
Inherited from Misc: __getitem__, __setitem__, __str__, _nametowidget, after, after_cancel, after_idle, bbox, bell, bind, bind_all, bind_class, bindtags, cget, clipboard_append, clipboard_clear, colormodel, columnconfigure, config, configure, deletecommand, event_add, event_delete, event_generate, event_info, focus, focus_displayof, focus_force, focus_get, focus_lastfor, focus_set, getboolean, getvar, grab_current, grab_release, grab_set, grab_set_global,

grab_status, grid_bbox, grid_columnconfigure, grid_location, grid_propagate, grid_rowconfigure, grid_size, grid_slaves, image_names, image_types, keys, lift, location, lower, mainloop, nametowidget, option_add, option_clear, option_get, option_readfile, pack_propagate, pack_slaves, place_slaves, propagate, quit, rowconfigure, selection_clear, selection_get, selection_handle, selection_own, selection_own_get, send, setvar, size, slaves, tk_bisque, tk_focusFollowsMouse, tk_focusNext, tk_focusPrev, tk_menuBar, tk_setPalette, tk_strictMotif, tkraise, unbind, unbind_all, unbind_class, update, update_idletasks, wait_variable, wait_visibility, wait_window, waitvar, winfo_atom, winfo_atomname, winfo_cells, winfo_children, winfo_class, winfo_colormapfull, winfo_containing, winfo_depth, winfo_exists, winfo_fpixels, winfo_geometry, winfo_height, winfo_id, winfo_interps, winfo_ismapped, winfo_manager, winfo_name, winfo_parent, winfo_pathname, winfo_pixels, winfo_pointerx, winfo_pointerxy, winfo_pointery, winfo_reqheight, winfo_reqwidth, winfo_rgb, winfo_rootx, winfo_rooty, winfo_screen, winfo_screencells, winfo_screendepth, winfo_screenheight, winfo_screenmmheight, winfo_screenmmwidth, winfo_screenvisual, winfo_screenwidth, winfo_server, winfo_toplevel, winfo_viewable, winfo_visual, winfo_visualid, winfo_visualsavailable, winfo_vrootheight, winfo_vrootwidth, winfo_vrootx, winfo_vrooty, winfo_width, winfo_x, winfo_y
Inherited from Pack: forget, info, pack, pack_configure, pack_forget, pack_info
Inherited from Place: place, place_configure, place_forget, place_info

G.6.2 Class Variables

Name	Description
Inherited from Misc: _noarg_ *()*	

G.7 Class gui.myFrame



All Tkinter frames inherit this class. This gives the new frame attributes like a control class, a master frame and a root frame. The control class handles all application control. The master frame is the parent of this frame, while the root frame is the application root.

G.7.1 Methods

__init__(*self*, *master*, **kw)
Overrides: Tkinter.Frame.__init__

Inherited from BaseWidget: destroy
Inherited from Grid: grid, grid_configure, grid_forget, grid_info, grid_remove
Inherited from Misc: __getitem__, __setitem__, __str__, _nametowidget, after, after_cancel, after_idle, bbox, bell, bind, bind_all, bind_class, bindtags, cget, clipboard_append, clipboard_clear, colormodel, columnconfigure, config, configure, deletecommand, event_add, event_delete, event_generate, event_info, focus, focus_displayof, focus_force, focus_get, focus_lastfor, focus_set, getboolean, getvar, grab_current, grab_release, grab_set, grab_set_global, grab_status, grid_bbox, grid_columnconfigure, grid_location, grid_propagate, grid_rowconfigure, grid_size, grid_slaves, image_names, image_types, keys, lift, location, lower, mainloop, nametowidget, option_add, option_clear, option_get, option_readfile, pack_propagate, pack_slaves, place_slaves, propagate, quit, rowconfigure, selection_clear, selection_get, selection_handle, selection_own, selection_own_get, send, setvar, size, slaves, tk_bisque, tk_focusFollowsMouse, tk_focusNext, tk_focusPrev, tk_menuBar, tk_setPalette, tk_strictMotif, tkraise, unbind, unbind_all, unbind_class, update, update_idletasks, wait_variable, wait_visibility, wait_window, waitvar, winfo_atom, winfo_atomname, winfo_cells, winfo_children, winfo_class, winfo_colormapfull, winfo_containing, winfo_depth, winfo_exists, winfo_fpixels, winfo_geometry, winfo_height, winfo_id, winfo_interps, winfo_ismapped, winfo_manager, winfo_name, winfo_parent, winfo_pathname, winfo_pixels, winfo_pointerx, winfo_pointerxy, winfo_pointery, winfo_reqheight, winfo_reqwidth, winfo_rgb, winfo_rootx, winfo_rooty, winfo_screen, winfo_screencells, winfo_screendepth, winfo_screenheight, winfo_screenmmheight, winfo_screenmmwidth, winfo_screenvisual, winfo_screenwidth, winfo_server, winfo_toplevel, winfo_viewable, winfo_visual, winfo_visualid, winfo_visualsavailable, winfo_vrootheight, winfo_vrootwidth, winfo_vrootx, winfo_vrooty, winfo_width, winfo_x, winfo_y
Inherited from Pack: forget, info, pack, pack_configure, pack_forget, pack_info
Inherited from Place: place, place_configure, place_forget, place_info

G.7.2 Instance Variables

Name	Description
control	Class controlling the application *(type=Instance of a userdefined class)*
master	Parent frame *(type=Tkinter.Frame)*
root	Application root *(type=Tkinter.Frame)*

G.7.3 Class Variables

Name	Description
Inherited from Misc: _noarg_ *()*	

G.8 Module myexceptions

This module contains all userdefined exceptions for this package.

G.8.1 Classes

- **ConfigError**: Exception class raised when errors are found in a configuration file.
  (Section G.9, p. 123)

G.9 Class myexceptions.ConfigError

```
exceptions.Exception ─┐
                      │
                  ConfigError
```

Exception class raised when errors are found in a configuration file.

G.9.1 Methods

__init__(*self*, *message*)

Constructor.

Parameters

`self:`	An instance of this class. *(type=An instance of ConfigError.)*
`message:`	Exception message. *(type=String)*

Overrides: exceptions.Exception.__init__

__str__(*self*)

Overrides builin method called when an instance of this object is printed.

Parameters

`self:`	An instance of this class. *(type=An instance of ConfigError.)*

Overrides: exceptions.Exception.__str__

Inherited from Exception: __getitem__

G.9.2 Instance Variables

Name	Description
message	Message to give when this exception is raised. *(type=String)*

G.10 Module configuration.particle

This module contains the class Particle, which holds all particle information. It contains a datafile of the properties of available baryons and contains a static method for reading this datafile.

Typical usage:

```
>>> from particle import *
>>> particles = read_particles(filename)
>>> for p in particles:
    # process Particle object
```

G.10.1 Classes

- **Particle**: Identifies a baryon
  (Section G.11, p. 124)

G.11 Class configuration.particle.Particle

Identifies a baryon

G.11.1 Methods

__init__(*self*, *name*, *short*, *id_*, *tz*, *s*, *m*, *q*)

Constructor.
Initializes all instance variables to the correct values.

Parameters

`self:`	Reference to this instance of the class. *(type=Particle)*
`name:`	Baryon name. *(type=string)*
`short:`	Baryon shortname. *(type=string)*
`id_:`	Baryon id *(type=int)*
`tz:`	Twice isospin projection. *(type=int)*
`s:`	strange number. *(type=int)*
`m:`	mass. *(type=float)*
`q:`	charge. *(type=int)*

__eq__(*self*, *p2*)

Compares to particles to see if they are equal.

Parameters

`self:`	Reference to this instance of the class. *(type=Particle)*
`p2:`	The other particle to compare with. *(type=Particle)*

Return Value

True if two particles are equal. False otherwise.
(type=Boolean)

__str__(*self*)

Parameters

`self:`	Reference to this instance of the class. *(type=Particle)*

Return Value

a string representation of this baryon.
(type=String)

get_id(*self*)

Get the numerical id given to this particle.

Parameters

`self`: Reference to this instance of the class.
(type=Particle)

Return Value

The numerical id given to this baryon.
(type=int)

G.11.2 Static Methods

read_particles(*filename*)

Return a list of particle objects whose string representations are stored in the file with name filename.

Parameters

`filename`: Filename where baryon properties are stored.
(type=Valid filename as a string)

Return Value

A list of Particle objects read from file.
(type=list)

Raises

`IOError` When filename is not a valid file.
`ValueError` When file doesn't have the correct format.

G.11.3 Instance Variables

Name	Description
id_	Particle id *(type=int)*
m	mass. *(type=float)*
name	Name. *(type=string)*
q	charge. *(type=int)*
s	strange number. *(type=int)*
short	Shortname. *(type=string)*
tz	Twice isospin projection. *(type=int)*

G.11.4 Class Variables

Name	Description
particles	Dictionary of all particles and their properties. **Value:** {'sigma-': <configuration.partic-le.Particle instance at 0x2b40abb39d40>-, 'sig... *(type=dict of Particles)*

G.12 Package configuration.relative

G.12.1 Modules

- **relative** *(Section G.13, p. 127)*

G.13 Module configuration.relative.relative

G.13.1 Classes

- **Channel**: Identifies a channel with specific total angular momentum (J), total isospin projection (Tz), max/min orbital momentum, spin and strangenumber.
  (Section G.14, p. 128)
- **Channel2**: Identifies a channel with specific total angular momentum (J), total isospin projection (Tz), max/min orbital momentum, spin and strangenumber.
  (Section G.14, p. 128)
- **SubChannel**: Identifies a subchannel (baryon-baryon configuration) with 2 baryons and their collective properties.
  (Section G.15, p. 134)

G.13.2 Variables

Name	Description
charge_to_iso	**Value:** `{0: {0: -2, 1: 0, 2: 2}, -2: {0: -0, 1: 2, 2: 4, -2: -4, -1: -2}, -1: {0:--1, ...` *(type=`dict`)*
charges	**Value:** `{0: 0, 1: 2, 2: 4, -2: -4, -1: -2}` *(type=`dict`)*
DEBUG	**Value:** `False` *(type=`bool`)*
hyp_max_j	**Value:** `7` *(type=`int`)*
partial_waves	**Value:** `['s', 'p', 'd', 'f', 'g', 'h', '-i', 'j']` *(type=`list`)*

G.14 Class configuration.relative.relative.Channel2

```
__builtin__.object ─┐
                    Channel2
```

Identifies a channel with specific total angular momentum (J), total isospin projection (Tz), max/min orbital momentum, spin and strangenumber. It also includes the number of subchannels and enumerates the specific subchannels in an array.

G.14.1 Methods

__init__(*self*, *J*, *Tz*, *L_min*, *L_max*, *S_min*, *S_max*, *strange*)

Constructor.
Initialize the channel instance variables.

Parameters

`J:`	Total angular momentum. *(type=int)*
`Tz:`	Total isospin projection (actually twice to get an integer). *(type=int)*
`L_min:`	Minimum total orbital momentum. *(type=int)*
`L_max:`	Maximum total orbital momentum. *(type=int)*
`S_min:`	Minimum intrinsic spin. *(type=int)*
`S_max:`	Maximum intrinsic spin. *(type=int)*
`strange:`	Total strange number. *(type=int)*

Overrides: __builtin__.object.__init__

__str__(*self*)

Return Value

a string representation of this channel instance.
(type=String)

Overrides: __builtin__.object.__str__

find_subs(*self*)

This method finds all unique subchannels for a given channel. If the subchannel contains identical particles, the anti-symmetry requirement is applied.
Updates the instance variables subs, nsub and charge.

Return Value

None

get_ids(*self*)

Return a single list of all particles in the subchannels.

Return Value

a list of particle id's (int)
(type=int)

get_label(*self*)

Creates a string label for this channel.

Return Value

Label representing this channel.
(type=String)

get_masses(*self*)

Return a list of reduced masses for the subchannels of this channel. If the channel is coupled in l, the list is duplicated so the masses appear twice.

Return Value

a list of reduced masses (floats) for this channel
(type=list)

get_number_sub(*self*)

Return Value

Number of baryon-baryon configurations in this channel.
(type=int)

get_reduced(*self*)

Return a list of reduced masses for the subchannels of this channel. If the channel is coupled in l, the list is duplicated so the masses appear twice.

Return Value

a list of reduced masses (floats) for this channel
(type=float)

is_coupled(*self*)

Return Value

True if this channel couples configurations with different orbital momentum.

(type=Boolean)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__

G.14.2 Static Methods

get_channel(*j*, *s*, *l*, *t*, *strange*)

Return a Channel instance with the quantum numbers specified in the parameters. If no such channel exists, return None.

Parameters

`j:`	Total angular momentum. *(type=int)*
`s:`	Total intrinsic spin projection. *(type=int)*
`l:`	Orbital momentum. *(type=int)*
`t:`	Twice total isospin. *(type=int)*
`strange:`	Strange quantum number. *(type=int)*

Return Value

A Channel instance if it exists. None Otherwise.

(type=Channel)

get_channel_by_charge(*j*, *s*, *l*, *q*, *strange*)

Return a Channel instance with the quantum numbers specified in the parameters. If no such channel exists, return None.

Parameters

`j:`	Total angular momentum. *(type=int)*
`s:`	Total intrinsic spin projection. *(type=int)*
`l:`	Orbital momentum. *(type=int)*
`q:`	Total charge. *(type=int)*
`strange:`	Strange quantum number. *(type=int)*

Return Value

A Channel instance if it exists. None Otherwise.
(type=Channel)

has_subs(*s*, *l*, *Tz*, *strange*)

Checks if a Channel identified with the supplied quantum numbers has subchannels. i.e, there are baryon-baryon configurations with these quantum numbers. If there is such a configuration - antisymmetry is also required if the baryons are identical.

Parameters

`s:`	Total spin. *(type=int)*
`l:`	Total orbital momentum. *(type=int)*
`Tz:`	Twice total isospin projection. *(type=int)*
`strange:`	Total strange number. *(type=int)*

Return Value

True if the channel has subchannels. False otherwise.
(type=boolean)

is_lcoupled(*l*, *j*, *strange*, *t*)

Check if the channel represented by these quantum numbers couples different orbital momenta.

Parameters

`l:`	Total orbital momentum. *(type=int)*
`j:`	Total angular momentum. *(type=int)*
`strange:`	Total strange number. *(type=int)*
`t:`	Twice total isospin projection. *(type=int)*

Return Value

True if these quantum numbers couples different orb. mom.
(type=boolean)

is_scoupled(*l*, *j*, *strange*, *t*)

Check if the channel represented by these quantum numbers couples different intrinsic spins.

Parameters

`l:`	Total orbital momentum. *(type=int)*
`j:`	Total angular momentum. *(type=int)*
`strange:`	Total strange number. *(type=int)*
`t:`	Twice total isospin projection. *(type=int)*

Return Value

True if these quantum numbers couples different spins.
(type=boolean)

setup_channels(*j_min*, *j_max*)

This method creates a list of all channels. The parameters specify for which total angular momenta to create channels.

Parameters

`j_min:`	Minimum total angular momentum. *(type=int)*
`j_max:`	Maximum total angular momentum. *(type=int)*

Return Value

List of all channel instances for the given range of total angular momentum.
(type=list)

valid_channel(*l*, *j*, *strange*, *t*)

Is the current configuration of quantum numbers a valid channel. Does it have correct relation between total angular momentum, total orbital momentum and total spin. If identical particles, the combination has to be anti- symmetric with regards to s, l and t.

Parameters

`l:`	Total orbital momentum. *(type=int)*
`j:`	Total angular momentum. *(type=int)*
`strange:`	Total strange number. *(type=int)*
`t:`	Twice total isospin projection. *(type=int)*

Return Value

A list of channel objects satisfying this set of quantum numbers.
(type=list)

G.14.3 Instance Variables

Name	Description
charge	Total charge for channel. *(type=int)*
J	Total angular momentum. *(type=int)*
l_coupled	Does this channel have a coupling between different orbital momenta? *(type=boolean)*
L_max	Maximum total orbital momentum. *(type=int)*
L_min	Minimum total orbital momentum. *(type=int)*
nsub	Number of subchannels in this channel. *(type=int)*
s_coupled	Does this channel have a coupling between different isospin projections? *(type=boolean)*
S_max	Maximum total isospin projection. *(type=int)*
S_min	Minimum total isospin projection. *(type=int)*
strange	Total strange number. *(type=int)*
subs	List of SubChannel objects for this channel. *(type=list of SubChannel)*

continued on next page

Name	Description
t_coupled	Does this channel have a coupling between different baryon baryon configuration. *(type=boolean)*
Tz	Total isospin projection (actually twice to get an integer). *(type=int)*

G.14.4 Class Variables

Name	Description
strange_min	Minimum value of strange quatum number to setup channels for. **Value:** `-1` *(type=`int`)*
t_max	**Value:** `4` *(type=`int`)*
valid_particles	A list of particles to include when setting up valid channels. For now: Exclude Cascade baryons and baryons with strange number less than -2. **Value:** `['proton', 'neutron', 'lambda', 'sigma+', 'sigma0', 'sigma-']` *(type=`list`)*

G.15 Class configuration.relative.relative.SubChannel

Identifies a subchannel (baryon-baryon configuration) with 2 baryons and their collective properties.

G.15.1 Methods

__init__(*self*, *p1*, *p2*)

Constructor.
Initializes all instance variables to the correct values.

Parameters

`p1`: Baryon 1 in this subchannel.
(type=Particle)

`p2`: Baryon 2 in this subchannel.
(type=Particle)

__eq__(*self*, *s2*)

Return Value

True if the two subchannels are equal. False otherwise.
(type=boolean)

__str__(*self*)

Return Value

a string representation of this subchannel.
(type=String)

get_ids(*self*)

Creates a list of numerical particle id's for the two particles in this subchannel.

Return Value

List of numerical particle id's.
(type=list)

G.15.2 Instance Variables

Name	Description
charge	Total charge of the two baryons. *(type=int)*
mass	Reduced mass of the baryons. *(type=float)*
name	Shortname to identify the subchannel. *(type=string)*
p1	Baryon 1 in this subchannel. *(type=Particle)*
p2	Baryon 2 in this subchannel. *(type=Particle)*
t_mass	Total mass of the baryons. *(type=float)*

G.16 Package interaction

This package contains the interface to all baryon-baryon interactions. Currently the only supported are the nijmegen NSC97 potential, the Idaho n3lo NN potential and the juelich YN potential.

typical usage:

```
>>> capabilities = get_capabilities()
>>> for c in capabilities:
>>>     model = selector(c)
>>>     # Process model
```

G.16.1 Modules

- **intmodel**: Toplevel class of all interaction models.
  (Section G.17, p. 136)

- **juelich** *(Section G.19, p. 138)*
  - **juelich**: This module implements the Juelich hyperon-nucleon potential.
    (Section G.20, p. 138)
- **n3lo** *(Section G.22, p. 140)*
  - **n3lo**: This module implements the Idaho n3lo nucleon-nucleon potential.
    (Section G.23, p. 141)
- **nijmegen** *(Section G.25, p. 142)*
  - **nijmegen**: This module implements the Nijmegen NSC97 potentials.
    (Section G.26, p. 143)

G.16.2 Functions

get_capabilities()

Create a list of the text representation of all interaction models currently implemented.

Return Value
List of interaction models supported.
(type=list)

selector(*pot*)

Returns an instance of the interaction model identified by the parameter.

Parameters
`pot`: The name of the potential you want.
(type=String)

Return Value
IntModel instance of correct type.
(type=IntModel)

G.17 Module interaction.intmodel

Toplevel class of all interaction models. Implements a null potential to for testing purposes.

G.17.1 Classes

- **IntModel**: Top level class for the different interaction models.
  (Section G.18, p. 137)

G.18 Class interaction.intmodel.IntModel

__builtin__.object ─┐
IntModel

Known Subclasses: Juelich, N3lo, NSC97

Top level class for the different interaction models.

G.18.1 Methods

__init__(*self*)

Constructor.

Overrides: __builtin__.object.__init__

__str__(*self*)

Creates a string representation of the object.

Return Value

String representation ob this object.
(type=String)

Overrides: __builtin__.object.__str__

get_potential(*self*, *channel*, *q*)

This method returns the matrix elements of this interaction model for the selected channels and meshpoints.

Parameters

`channel`:	The channel to calculate matrix elements. *(type=Channel)*
`q`:	Momenta to calculate matrix elements for. *(type=list of floats.)*

Return Value

A 2 dimensional array containing all marix elements for this channel.
(type=2-dim numpy array)

sort_subs(*self*, *channel*)

Default function for sorting the subchannels in a channel so as to match the order of the interaction model. No sorting necessary for null potential

Parameters

`channel`: The channel to sort the subchannels in.
(type=Channel)

Return Value

A channel object with sorted subchannels.
(type=Channel)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__

G.19 Package interaction.juelich

G.19.1 Modules

- **juelich**: This module implements the Juelich hyperon-nucleon potential. *(Section G.20, p. 138)*

G.19.2 Functions

get_capabilities()

G.19.3 Variables

Name	Description
orig	**Value:** `'/home/grj/work/uio/master/code/-packages/interaction'` *(type=`str`)*

G.20 Module interaction.juelich.juelich

This module implements the Juelich hyperon-nucleon potential. The matrix elements originally com in a partial wave isospin basis, but are translated to the physical basis on the fly.

The actual calculation of the matrix elements are done in the core module implemented in fortran.

The module consist of the Juelich class, which is the interface to to the potential. All calculations are done in the core module. The core module contain one callable subroutine juelich_front that collect the matrix elements for the specified momenta.

This module is selfcontained. All libraries needed are included in the shared object file.

G.20.1 Classes

- **Juelich**: Class for the Juelich interaction model.
  (Section G.21, p. 139)

G.20.2 Variables

Name	Description
DEBUG	**Value:** `True` *(type=`bool`)*
max_q	**Value:** `70` *(type=`int`)*

G.21 Class interaction.juelich.juelich.Juelich

__builtin__.object
interaction.intmodel.IntModel
Juelich

Class for the Juelich interaction model.

G.21.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.intmodel.IntModel.__init__

check_channel(*self*, *channel*, *q*)

Check that the channel doesn't overstep the possible properties of the model.

Parameters

`channel`: Channel object to calculate the matrix elements for.
(type=Channel)

Return Value

True if channel is valid, False otherwise.
(type=boolean)

create_subs(*self*)

Creates the subchannel structure for the potential code. Identifies where data for each of the subchannels are stored in the result array from the potential.

get_potential(*self*, *channel*, *q*)

This method returns the matrix elements of this interaction model for the selected channels and meshpoints.

Parameters

`channel:`	The channel to calculate matrix elements. *(type=Channel)*
`q:`	Momenta to calculate matrix elements for. *(type=list of floats.)*

Return Value

A 2 dimensional array containing all marix elements for this channel.
(type=2-dim numpy array)

Overrides: interaction.intmodel.IntModel.get_potential

translate_subs(*self*, *channel*)

Creates a translation list between the subchannels in the channel and the subchannels in the potential code.

Parameters

`channel:`	Channel object to calculate matrix elements for. *(type=Channel)*

Return Value

Translation array
(type=list)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__, sort_subs

G.21.2 Instance Variables

Name	Description
function	Function to call to calculate matrix elements.
max_q	Maximum number of momenta to calculate matrix elements for.
subs	Subchannel structure implemented in the potential code.

G.22 Package interaction.n3lo

G.22.1 Modules

- **n3lo**: This module implements the Idaho n3lo nucleon-nucleon potential. *(Section G.23, p. 141)*

G.22.2 Functions

get_capabilities()

G.23 Module interaction.n3lo.n3lo

This module implements the Idaho n3lo nucleon-nucleon potential. The matrix elements come in a partial wave physical basis, with options for Charge symmetry breaking and charge independence breaking. Bith are turned on by default.

The actual calculation of the matrix elements are done in the core module implemented in fortran.

The module consist of the N3lo class, which contains the interface to the potential. The core module is precompiled and contains the subroutine n3lo_front that collects all matrix elements for the specied momenta into a numpy array of rank 2. This module is selfcontained, so it does not need any other libraries installed. All lapack routines used in the potential code are included in the library.

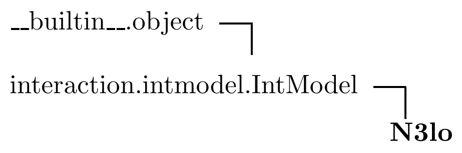
G.23.1 Classes

- **N3lo**: Class for the n3lo interaction model.
  (Section G.24, p. 141)

G.23.2 Variables

Name	Description
DEBUG	**Value:** `True` *(type=*`bool`*)*

G.24 Class interaction.n3lo.n3lo.N3lo



Class for the n3lo interaction model.

G.24.1 Methods

__init__(*self*)
Constructor.
Overrides: interaction.intmodel.IntModel.__init__

check_channel(*self*, *channel*)

Check that the channel doesn't overstep the possible properties of the model.

Parameters

`channel`: Channel object to calculate the matrix elements for.
(type=Channel)

Return Value

True if channel is valid, False otherwise.
(type=boolean)

get_potential(*self*, *channel*, *q*)

This method returns the matrix elements of this interaction model for the selected channels and meshpoints.

Parameters

`channel`: The channel to calculate matrix elements.
(type=Channel)

`q`: Momenta to calculate matrix elements for.
(type=list of floats.)

Return Value

A 2 dimensional array containing all marix elements for this channel.
(type=2-dim numpy array)

Overrides: interaction.intmodel.IntModel.get_potential

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__, sort_subs

G.24.2 Instance Variables

Name	Description
function	Interface function to call for the matrix elements.

G.25 Package interaction.nijmegen

G.25.1 Modules

- **nijmegen**: This module implements the Nijmegen NSC97 potentials. *(Section G.26, p. 143)*

G.25.2 Functions

get_capabilities()

G.26 Module interaction.nijmegen.nijmegen

This module implements the Nijmegen NSC97 potentials. There are 6 of them in total each support strangeness channels $S \geq -4$. Only nucleon-nucleon, hyperon-nucleon and hyperon-hyperon are implemented in this interface however. These strangeness channels require different treatment in the potential code, so a class hierarchy has been created to be able to handle these difference seamlessly.

The first structure is the NSC97BB class and it's children. There is one child pr. strangeness channel supported, since each have different baryon-baryon configurations, different interface functions and so on.

The secon structure is the NSC97 class and it's children. There are 6 children in total, one for each different model within the NSC97 group. The children defines different parameters needed for the different models to be calculated.

The actual calculations are performed in the potential code written in fortran. The core module contains 3 callable subroutines, one for each strangeness channel supported. These subroutines are responsible for extracting the correct matrix elements.

This module is comletely selcontained. All library functions are included in the shared object.

G.26.1 Classes

- **NSC97**: Parent class for all potential interfaces.
  (Section G.31, p. 147)
- **NSC97A**: Class for the NSC97a interaction model
  (Section G.32, p. 148)
- **NSC97B**: Class for the NSC97b interaction model
  (Section G.33, p. 149)
- **NSC97BB**: Parent class of classes implmenting the different strangeness channels.
  (Section G.27, p. 144)
- **NSC97C**: Class for the NSC97c interaction model
  (Section G.34, p. 150)
- **NSC97D**: Class for the NSC97d interaction model
  (Section G.35, p. 150)
- **NSC97E**: Class for the NSC97e interaction model
  (Section G.36, p. 151)
- **NSC97F**: Class for the NSC97f interaction model
  (Section G.37, p. 152)
- **NSC97NN**: Interface to the Nucleon-Nucleon potential.
  (Section G.28, p. 145)
- **NSC97YN**: Interface to the Nucleon-Hyperon potential from the stoks nijmegen code.
  (Section G.29, p. 145)
- **NSC97YY**: Interface to the Hyperon-Hyperon potential.
  (Section G.30, p. 146)

G.26.2 Variables

Name	Description
DEBUG	**Value:** `False` *(type=`bool`)*

G.27 Class interaction.nijmegen.nijmegen.NSC97BB

```
__builtin__.object ─┐
                    NSC97BB
```

Known Subclasses: NSC97NN, NSC97YN, NSC97YY

Parent class of classes implmenting the different strangeness channels. Contains methods and variables common to all children.

G.27.1 Methods

__init__(*self*)

Constructor for potential interface base class.
All instance variables that are common between the subclasses are set here.

Overrides: __builtin__.object.__init__

translate_subs(*self*, *channel*)

Creates a translation list between the subchannels in the channel and the subchannels in the potential code.

Parameters

`channel`: The channel to calculate matrix elements for.
(type=Channel)

Return Value

Translation array
(type=list)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

G.27.2 Instance Variables

Name	Description
core_function	Function to call when calculating the matrix elements.
subs	Structure containing the baryon-baryon configurations for the different strangeness channels.

G.28 Class interaction.nijmegen.nijmegen.NSC97NN

```
__builtin__.object ─┐
                    │
interaction.nijmegen.nijmegen.NSC97BB ─┐
                                       │
                                    NSC97NN
```

Interface to the Nucleon-Nucleon potential. Strangeness 0.

G.28.1 Methods

__init__(*self*)

Constructor for Nucleon-Nucleon potential interface sub class. Strange number = 0

Overrides: interaction.nijmegen.nijmegen.NSC97BB.__init__

create_subs(*self*)

Creates the subchannel structure for the potential code. Identifies where data for each of the subchannels are stored in the result array from the potential.

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from NSC97BB: translate_subs

G.28.2 Instance Variables

Name	Description
Inherited from NSC97BB: core_function *(p. 144)*, subs *(p. 144)*	

G.29 Class interaction.nijmegen.nijmegen.NSC97YN

```
__builtin__.object ─┐
                    │
interaction.nijmegen.nijmegen.NSC97BB ─┐
                                       │
                                    NSC97YN
```

Interface to the Nucleon-Hyperon potential from the stoks nijmegen code. Strangeness -1.

G.29.1 Methods

__init__(*self*)

Constructor for Nucleon-Hyperon potential interface sub class. Strange number = -1.

Overrides: interaction.nijmegen.nijmegen.NSC97BB.__init__

create_subs(*self*)

Creates the subchannel structure for the potential code. Identifies where data for each of the subchannels are stored in the result array from the potential.

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from NSC97BB: translate_subs

G.29.2 Instance Variables

Name	Description
Inherited from NSC97BB: core_function *(p. 144)*, subs *(p. 144)*	

G.30 Class interaction.nijmegen.nijmegen.NSC97YY

__builtin__.object

interaction.nijmegen.nijmegen.NSC97BB

NSC97YY

Interface to the Hyperon-Hyperon potential. Strangeness -2.

G.30.1 Methods

__init__(*self*)

Constructor for Hyperon-Hyperon potential interface sub class for the nijmegen potential. Strange number = -2

Overrides: interaction.nijmegen.nijmegen.NSC97BB.__init__

create_subs(*self*)

Creates the subchannel structure for the potential code. Identifies where data for each of the subchannels are stored in the result array from the potential.

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from NSC97BB: translate_subs

G.30.2 Instance Variables

Name	Description
Inherited from NSC97BB: core_function *(p. 144)*, subs *(p. 144)*	

G.31 Class interaction.nijmegen.nijmegen.NSC97

```
__builtin__.object ─┐
                    │
interaction.intmodel.IntModel ─┐
                               │
                             NSC97
```

Known Subclasses: NSC97A, NSC97B, NSC97C, NSC97D, NSC97E, NSC97F

Parent class for all potential interfaces. All methods and variables shared between the subclasses has been placed here.

G.31.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.intmodel.IntModel.__init__

get_potential(*self*, *channel*, *q*)

This method returns the matrix elements of this interaction model for the selected channels and meshpoints.

Parameters

`channel`:	The channel to calculate matrix elements. *(type=Channel)*
`q`:	Momenta to calculate matrix elements for. *(type=list of floats.)*

Return Value

A 2 dimensional array containing all marix elements for this channel.
(type=2-dim numpy array)

Overrides: interaction.intmodel.IntModel.get_potential

sort_subs(*self*, *channel*)

Sort the subchannels of the given channel in accordance with the subchannel structure in the potential code.

Parameters

`channel`: Channel to sort.
(type=Channel)

Return Value

Channel object with sorted subchannels.
(type=Channel)

Overrides: interaction.intmodel.IntModel.sort_subs

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__

G.31.2 Class Variables

Name	Description
nn_structure	Structure object for the NN interaction **Value:** `<interaction.nijmegen.nijmegen.N-SC97NN object at 0x2aae6199ebd0>` *(type=NSC97BB)*
ns	Identifies how much debugging information should be written in the potential code. **Value:** `0` *(type=int)*
nymod	Identifies which NSC97 potential should be used. *(type=int)*
yn_structure	Structure object for the YN interaction **Value:** `<interaction.nijmegen.nijmegen.N-SC97YN object at 0x2aae6199ec90>` *(type=NSC97BB)*
yy_structure	Structure object for the YY interaction **Value:** `<interaction.nijmegen.nijmegen.N-SC97YY object at 0x2aae6199ecd0>` *(type=NSC97BB*

G.32 Class interaction.nijmegen.nijmegen.NSC97A

```
__builtin__.object ─┐
                    │
interaction.intmodel.IntModel ─┐
                               │
interaction.nijmegen.nijmegen.NSC97 ─┐
                                     │
                                  NSC97A
```

Class for the NSC97a interaction model

G.32.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.nijmegen.nijmegen.NSC97.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__
Inherited from NSC97: get_potential, sort_subs

G.32.2 Class Variables

Name	Description
Inherited from NSC97: nn_structure *(p. 147)*, ns *(p. 147)*, nymod *(p. 147)*, yn_structure *(p. 147)*, yy_structure *(p. 147)*	

G.33 Class interaction.nijmegen.nijmegen.NSC97B

```
__builtin__.object ─┐
                    │
interaction.intmodel.IntModel ─┐
                               │
interaction.nijmegen.nijmegen.NSC97 ─┐
                                     │
                                   NSC97B
```

Class for the NSC97b interaction model

G.33.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.nijmegen.nijmegen.NSC97.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__
Inherited from NSC97: get_potential, sort_subs

G.33.2 Class Variables

Name	Description
Inherited from NSC97: nn_structure *(p. 147)*, ns *(p. 147)*, nymod *(p. 147)*, yn_structure *(p. 147)*, yy_structure *(p. 147)*	

G.34 Class interaction.nijmegen.nijmegen.NSC97C

```
__builtin__.object ─┐
interaction.intmodel.IntModel ─┐
interaction.nijmegen.nijmegen.NSC97 ─┐
                                     NSC97C
```

Class for the NSC97c interaction model

G.34.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.nijmegen.nijmegen.NSC97.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__
Inherited from NSC97: get_potential, sort_subs

G.34.2 Class Variables

Name	Description
Inherited from NSC97: nn_structure *(p. 147)*, ns *(p. 147)*, nymod *(p. 147)*, yn_structure *(p. 147)*, yy_structure *(p. 147)*	

G.35 Class interaction.nijmegen.nijmegen.NSC97D

```
__builtin__.object ─┐
interaction.intmodel.IntModel ─┐
interaction.nijmegen.nijmegen.NSC97 ─┐
                                     NSC97D
```

Class for the NSC97d interaction model

G.35.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.nijmegen.nijmegen.NSC97.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__
Inherited from NSC97: get_potential, sort_subs

G.35.2 Class Variables

Name	Description
Inherited from NSC97: nn_structure *(p. 147)*, ns *(p. 147)*, nymod *(p. 147)*, yn_structure *(p. 147)*, yy_structure *(p. 147)*	

G.36 Class interaction.nijmegen.nijmegen.NSC97E

```
__builtin__.object ─┐
interaction.intmodel.IntModel ─┐
interaction.nijmegen.nijmegen.NSC97 ─┐
                                     NSC97E
```

Class for the NSC97e interaction model

G.36.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.nijmegen.nijmegen.NSC97.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__
Inherited from NSC97: get_potential, sort_subs

G.36.2 Class Variables

Name	Description
Inherited from NSC97: nn_structure *(p. 147)*, ns *(p. 147)*, nymod *(p. 147)*, yn_structure *(p. 147)*, yy_structure *(p. 147)*	

G.37 Class interaction.nijmegen.nijmegen.NSC97F

__builtin__.object

interaction.intmodel.IntModel

interaction.nijmegen.nijmegen.NSC97

NSC97F

Class for the NSC97f interaction model

G.37.1 Methods

__init__(*self*)

Constructor.

Overrides: interaction.nijmegen.nijmegen.NSC97.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__
Inherited from IntModel: __str__
Inherited from NSC97: get_potential, sort_subs

G.37.2 Class Variables

Name	Description
Inherited from NSC97: nn_structure *(p. 147)*, ns *(p. 147)*, nymod *(p. 147)*, yn_structure *(p. 147)*, yy_structure *(p. 147)*	

G.38 Package renormalization

This package implements the different kinds of renormalization procedures used for creating an effective two-body baryon-baryon interaction. The end result can be presented in the relative/com frame with a partial wave momentum basis, or the elements can be presented in a 3-d spherical harmonic oscillator basis in the lab frame coupled to a specific total angular momentum.

G.38.1 Modules

- **renormalization**: This module contains the parent class of all renormalization classes.
  (Section G.39, p. 153)
- **vlowk** *(Section G.41, p. 161)*
  - **vlowk**: This module contains the class Vlowk, implementing the low momentum interaction described in section ef{sec:vlowk}.
    (Section G.42, p. 161)

G.38.2 Functions

selector(*filename*)

The function reads a configuration file and determines which type of calculation should be done. The correct class is called for this calculation.

Parameters

`filename`: Name of configuration file.
(type=String)

Return Value

Specific renormalization object for the selected configuration.
(type=Renromalization)

G.38.3 Variables

Name	Description
all	**Value:** `['renormalization', 'vlowk', 'selector']` *(type=`list`)*
config_log_file	**Value:** `'/home/grj/work/uio/master/code/packages/renormalization/data/config.log'` *(type=`str`)*
serial_file	**Value:** `'/home/grj/work/uio/master/code/packages/renormalization/data/last_serial.dat'` *(type=`str`)*

G.39 Module renormalization.renormalization

This module contains the parent class of all renormalization classes. Most of the functionality is placed in here, while only parameters and methods specific to a renormalization procedure is in the derived classes.

This module also contains methods to solve the \SE for a given channel with matrix elements from a specified interaction model.

Typical usage:

```
>>> r = Renormalization()
>>> # Get the bare interaction elements for a single channel
>>> q, v = r.get_exact_potential(channel, model)
>>> # Get the energy eigenvalues for a specific channel
>>> e = r.get_exact_eigenvalues(channel, model)
```

G.39.1 Classes

- **Renormalization**: Top level class for the renormalization types.

(Section G.40, p. 154)

G.39.2 Variables

Name	Description
DEBUG	**Value:** `True` *(type=`bool`)*

G.40 Class renormalization.renormalization.Renormalization

```
__builtin__.object ─┐
                    Renormalization
```

Known Subclasses: Vlowk

Top level class for the renormalization types.

G.40.1 Methods

__init__(*self*)

Constructor

Overrides: __builtin__.object.__init__

check_channels(*self*, *channels*)

Check that all channels about to be initialized satisfies the configuration and optionally any other constraint the user wish to impose on the channels selected.

Parameters

`channels`: List of all channels currently selected for initialization.
(type=list)

Return Value

List of all channels satisfying the specified constraints.
(type=list)

check_config(*self*)

Check if a configuration dict contains all required key,value pairs.

Return Value

True if the configuration is valid for this calculation.
(type=boolean)

get(*self*, *key*)

Return a configuration value with a specified key. None if the key does not exist.

Parameters

`key`: Key to find in the configuration array.
(type=String)

Return Value

Value of the specified key.
(type=String)

get_exact_eigenvalues(*self*, *channel*, *model*)

Return eigenvalues of the Hamiltonian in the selected channel where the interaction elements are created using the specified model.

Parameters

`channel`: The channel to diagonalize.
(type=Channel)

`model`: The interaction model to use.
(type=IntModel)

Return Value

The energy eigenavlues of the specified channel, where the matrix elements for the interaction has been created with the specified model.
(type=numpy array of rank 1)

get_exact_potential_data(*self*, *channel*, *model*)

Initialize the core module and then get the interaction matrix elements for the specified channel, using the specified model to generate them.

Parameters

`channel`: The channel to diagonalize.
(type=Channel)

`model`: The interaction model to use.
(type=IntModel)

Return Value

The meshpoints and the generated matrix elements for the interaction.
(type=tuple containing a rank1 and rank 2 numpy array)

get_interaction_data(*self*, *channel*, *model*)

Get the interaction matrix elements for the specified channel, using the specified model to generate them.

Parameters

`channel`: The channel to diagonalize.
(type=Channel)

`model`: The interaction model to use.
(type=IntModel)

Return Value

The meshpoints and the generated matrix elements for the interaction.
(type=tuple containing a rank1 and rank 2 numpy array)

get_nn_models(*self*)

Create a list of available nn interaction models.

Return Value

A list of available nn interaction models.
(type=list)

get_renorm_potential_data(*self*, *channel*, *model*)

Get the matrix elements after renormalization for a specific channel using a specified model. All classes derived from this should overload this one.

get_yn_models(*self*)

Create a list of available yn interaction models.

Return Value

A list of available yn interaction models.
(type=list)

get_yy_models(*self*)

Create a list of available yy interaction models.

Return Value

A list of available yy interaction models.
(type=list)

include_nn(*self*)

Should the current run generate nucleon-nucleon matrix elements.

Return Value

True if nn elemenst should be generated.
(type=boolean)

include_yn(*self*)

Should the current run generate hyperon-nucleon matrix elements.

Return Value

True if yn elemenst should be generated.
(type=boolean)

include_yy(*self*)

Should the current run generate hyperon-hyperon matrix elements.

Return Value

True if y elemenst should be generated.
(type=boolean)

init_config(*self*)

Initialize the compiled core module. The core module is written in fortran and compiled as a shared object. This initialization is needed so the datastructures are allocated correctly and necessary variables set.

Return Value

True if the initialization went ok. False otherwise.
(type=boolean)

init_post_channel(*self*)

Do all initialization that needs to be done after the core module has been initialized with channel information.

Return Value

None

list_channels(*self*)

Tell the core module to list all channels currently configured.

Return Value

None

list_config(*self*)

List current configuration to screen.

Return Value

None

prepare_single(*self*, *channel*, *model*)

Prepare to renormalize a single channel in momentum basis. All necessary configuration options are set and the specified channel is initialized in the core module.

Parameters

`channel`: Channel to generate renormalized matrix elements for.
(type=Channel)

`model`: Interaction model used to create the matrix elements.
(type=IntModel)

Return Value

None

Raises

`ValueError` If not properly initialized, or an invalid channel or model an exception is raised.

setup_all_channels(*self*)

Create all channels from the given configuration and initialize the core modules partial_wave structure with correct channel data for each channel. After the core module has been initialized, all remaining initializations requiring the partial_waves allready set, are run.

Return Value

None

setup_all_potentials(*self*)

Loop through all initialized channels and get the matrix elements for this channel. Any constraints of which model should be used for a specific channel is done in this function. The matrix elements are passed to an interface function for updating the core module.

Return Value

None

setup_channel(*self*, *i*, *c*)

Initialize the core module with a single channel with the specified index number.
For a single run, the index will allways be 1, while for a lab transformation, it will run through the number of channels needed.

Parameters

`i`: Index of the channel beeing initialized. This refers to the core module's internal storage of channels.
(type=int)

`c`: Channel to initialize.
(type=Channel)

Return Value

None

setup_one_channel(*self*, *j*, *s*, *q*, *strange*)

Create a channel object from the given parameters and initialize the core module with the correct partial_wave structure. Run remaining initializations after the core module has been set.
This method is typically called when the end result should be a renormalized interaction for a single channel, in the relative momentum basis.

Parameters

`j`:	Total angular momentum. *(type=int)*
`s`:	Intrinsic spin. *(type=int)*
`q`:	Total charge. *(type=int)*
`strange`:	Strangeness flavour. *(type=int)*

Return Value

None

setup_potential(*self*, *i*, *c*, *pot*, *q*)

Update the core module with the matrix elements for a specific channel. The core module renormalizes the selected interaction with the selected renormalization procedure and transforms them to a relative harmonic oscillator basis for use in the transformation to the lab system.

Parameters

`i`:	index referring to the channels storage in the core module. *(type=int)*
`c`:	Channel to renormalize. *(type=Channel)*
`pot`:	Name of the interaction model to use for this channel. *(type=String)*
`q`:	Meshpoints to calculate the matrix elements for. *(type=Numpy array of rank 1.)*

Return Value

None

transform_lab(*self*)

This method is the only method needed to call after the renormalization has been created. The different derived classes implement their own tecniques and the matrix elements are transformed to the lab frame in an harmonic oscillator basis. All derived classes should overload this method to create the proper renormalized matrix elements.

Return Value

None

Raises

`ValueError` This exceptions is raised if the current configuration is not valid for a lab transformation.

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

G.40.2 Static Methods

read_config(*filename*)

Reads a configuration file in the format: key=value populates a Python dictionary with the keys and corresponding values.

Parameters

`filename:` Filename of the config file.
(type=String)

Return Value

Dictionary with all elements in the configuration file.

G.40.3 Instance Variables

Name	Description
options	Dictionary of options passed to the constructor. **Value:** {} *(type=dict)*
singular	True if result should be presented in momentum relative partial wave basis. *(type=boolean)*

G.40.4 Class Variables

continued on next page

Name	Description

Name	Description

required	A list of option keys that are required for initialization.
	Value:
	`['type_of_renormv', 'coulomb_included', -`
	`'output_run', 'include_yn', 'include_...`
	(type=list)

G.41 Package renormalization.vlowk

G.41.1 Modules

- **vlowk**: This module contains the class Vlowk, implementing the low momentum interaction described in section ef{sec:vlowk}.
  (Section G.42, p. 161)

G.42 Module renormalization.vlowk.vlowk

This module contains the class Vlowk, implementing the low momentum interaction described in section ef{sec:vlowk}.

Typical usage:

```
>>> vlowk = Vlowk(options)
>>> # Get the renormalized matrix elements of a single channel
>>> q, v = vlowk.get_renorm_eigenvalues(channel, model)
>>> # Get the renormalized eigenvalues of a single channel
>>> e = vlowk.get_renorm_eigenvalues(channel, model)
>>> # Transform to the lab frame
>>> vlowk.transform_lab()
```

G.42.1 Classes

- **Vlowk**: Vlowk renormalization type.
  (Section G.43, p. 162)

G.42.2 Variables

Name	Description
DEBUG	**Value:** `True` *(type=`bool`)*

G.43 Class renormalization.vlowk.vlowk.Vlowk

```
__builtin__.object ─┐
                    │
renormalization.renormalization.Renormalization ─┐
                                                 │
                                               Vlowk
```

Vlowk renormalization type. core is the vlowk_front module of the fortran package

G.43.1 Methods

__init__(*self*, *options*=`None`)

Constructor.

Parameters

`options`: Optional dictionary of options to initialize.
(type=dict)

Overrides: renormalization.renormalization.Renormalization.__init__

get_renorm_eigenvalues(*self*, *channel*, *model*)

Return eigenvalues of the Hamiltonian in the selected channel where the interaction elements are created using the specified model, after the interaction has been renormalized using this procedure.

Parameters

`channel`: The channel to diagonalize.
(type=Channel)

`model`: The interaction model to use.
(type=IntModel)

Return Value

The energy eigenavlues of the specified channel, where the matrix elements for the interaction has been created with the specified model.
(type=numpy array of rank 1)

get_renorm_potential_data(*self*, *channel*, *model*)

Return the matrix elements of the renormalized interaction, where the interaction elements are created using the specified model, after the interaction has been renormalized using this procedure.

Parameters

`channel`:	The channel to generate elements for. *(type=Channel)*
`model`:	The interaction model to use. *(type=IntModel)*

Return Value

The meshpoints used and the renormalized matrix elements.
(type=numpy array of rank 1 and rank 2)

Overrides:
renormalization.renormalization.Renormalization.get_renorm_potential_data

init_config(*self*)

Initialize the core module with specific vlowk options. Basically the mesh.

Return Value

True if initialization was completed.
(type=boolean)

Overrides: renormalization.renormalization.Renormalization.init_config

init_post_channel(*self*)

Do all initialization that needs to be done after the core module has been initialized with channel information.

Return Value

None

Overrides:
renormalization.renormalization.Renormalization.init_post_channel

list_config(*self*)

List current configuration to screen.

Return Value

None

Overrides: renormalization.renormalization.Renormalization.list_config

set_mesh(*self*, *n_m*, *c_m*, *n_q*, *c_q*)

Update core module with new mesh for the renormalization. The core module creates the new meshpoints using a Gauss- Legendre polynomial.

Parameters

`n_m`: Number of mehpoints in the modelspace.
(type=int)

`c_m`: Cutoff in the modelspace.
(type=int)

`n_q`: Number of meshpoints in the q-space.
(type=int)

`c_q`: Cutoff in the q-space
(type=int)

Return Value

None

test_mesh(*self*)

Check if the meshpoints are set to valid values.

Return Value

True if the meshpoints are set, False otherwise.
(type=boolean)

transform_lab(*self*)

This method is the only method needed to call after the renormalization has been created. The different derived classes implement their own tecniques and the matrix elements are transformed to the lab frame in an harmonic oscillator basis. All derived classes should overload this method to create the proper renormalized matrix elements.

Return Value

None

Raises

`ValueError` This exceptions is raised if the current configuration is not valid for a lab transformation.

Overrides: renormalization.renormalization.Renormalization.transform_lab

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from Renormalization: check_channels, check_config, get, get_exact_eigenvalues, get_exact_potential_data, get_interaction_data, get_nn_models, get_yn_models, get_yy_models, include_nn, include_yn, include_yy, list_channels, prepare_single, setup_all_channels, setup_all_potentials, setup_channel, setup_one_channel, setup_potential

G.43.2 Static Methods

Inherited from Renormalization: read_config

G.43.3 Instance Variables

Name	Description
mesh	Meshpoints and cutoffs for the modelspace and q-space. **Value:** `[None, None, None, None]` *(type=list)*
options	Dictionary of configuration options. *(type=dict)*
Inherited from Renormalization: singular *(p. 154)*	

G.43.4 Class Variables

Name	Description
additional	Additional recuired key,value pairs in the configuration . **Value:** `['n_k1', 'n_k2', 'k_cutoff', 'k_max']` *(type=list)*
name	**Value:** `'vlowk'` *(type=str)*
Inherited from Renormalization: required *(p. 154)*	

G.44 Package effective

(section) Effectice manybody interaction

This module implements the creation of an effective many-body interaction, created from two-body interaction in a harmonic oscillator basis.

If the hartree fock option is set, a self consistent set of basis states are calculated in first order, redfining the states and matrix elements, so as to not operate with linear combinations of harmonic oscillator states. The new states and matrix elements are written to file.

G.44.1 Modules

- **bhf**: This module 'bhf' is auto-generated with f2py (version:2_3979).
  (Section G.45, p. 165)
- **effective**: This module contains the class Effective, which is the interface to the compiled module bhf.
  (Section G.46, p. 167)

G.45 Module effective.bhf

```
This module 'bhf' is auto-generated with f2py (version:2_3979).
Functions:
  set_outputfiles(outputfile)
  set_inputfiles(renormfile,spdatafile)
  set_interaction(interaction,order)
  set_hf_iterations(hf)
  set_hffiles(hf_renorm,hf_spdata)
  setup()
  set_energies(num,energy)
  set_hyperons(l,sp,s0,sm)
  do_hartree_fock()
  do_onebody()
Fortran 90/95 modules:
  constants --- nmax,cutoff,nlmax,n_sigmap,total_mass,n_cm_mom,
  n_sigmam,pauli_operator,oscli,hf_iterations,hb2ip,
  type_of_yn_pot,pi_2,pi_4,type_of_yy_pot,itzmax,pi,
  coulomb_channel,sp_mass,n_startenergy_g,n_body_int,
  n_sigma0,neutron_mass,n_total,itzmin,coulomb_included,dp,
  n_lambda,cib,lab_nmax,n_rel,lmax,dpc,sigma0_mass,lab_lmax,
  type_of_renormv,strange_min,proton_mass,sp_charge,
  keep_originalenergies,n_startenergy_veff,include_yn,
  number_homega_exct,sigmam_mass,e_start_g,jmin,include_yy,
  starting_energy,theta_rot,lambda_mass,order_of_interaction,
  csb,sigmap_mass,jmax,wcn,j_lab_min,p_mass,j_lab_max,
  strange_max,hbarc,square_calculation,oscl,nlmax_model,
  type_of_interaction,type_of_nn_pot,hbar_omega,mass_nucleus,lambda.
```

G.45.1 Variables

Name	Description
__version__	**Value:** `'$Revision: $'` *(type=`str`)*
constants	**Value:** `<fortran object at 0x2b67aade92b0>` *(type=`fortran`)*
do_hartree_fock	**Value:** `<fortran object at 0x2b67aade9260>` *(type=`fortran`)*
do_onebody	**Value:** `<fortran object at 0x2b67aade9288>` *(type=`fortran`)*
set_energies	**Value:** `<fortran object at 0x2b67aade9210>` *(type=`fortran`)*
set_hf_iterations	**Value:** `<fortran object at 0x2b67aade9198>` *(type=`fortran`)*
set_hffiles	**Value:** `<fortran object at 0x2b67aade91c0>` *(type=`fortran`)*
set_hyperons	**Value:** `<fortran object at 0x2b67aade9238>` *(type=`fortran`)*

continued on next page

Name	Description
set_inputfiles	**Value:** `<fortran object at 0x2b67aade9148>` *(type=`fortran`)*
set_interaction	**Value:** `<fortran object at 0x2b67aade9170>` *(type=`fortran`)*
set_outputfiles	**Value:** `<fortran object at 0x2b67aade9120>` *(type=`fortran`)*
setup	**Value:** `<fortran object at 0x2b67aade91e8>` *(type=`fortran`)*

G.46 Module effective.effective

(section) Effective

This module contains the class Effective, which is the interface to the compiled module bhf. This module does all the actual calculation of diagrams and diagonalization, while the interface lets the user choose what operation is wanted.

For now only the setup_hf_orbits is implemented, which creates a self- consistent single particle basis for the given problem defined in the single particle data file and the configuration file.

In the single particle data file you specify a set of hole states for the selected nuclei and which states are to be part pf the modelspace and not. You must also specify which hyperon states to be included in the calculation.

In the configuration file, you specify the mass of the system, by specifying how many of the different hyperons are included. The total mass is calculated from this number and what hole states are defined.

Typical usage:

```
>>> e = Effective(filename)
>>> e.init_config()
>>> e.setup_hf_orbits()
```

G.46.1 Classes

- **Effective**: This class is the interface to the many-body module bhf. *(Section G.47, p. 168)*

G.46.2 Variables

Name	Description
DEBUG	**Value:** `True` *(type=`bool`)*

G.47 Class effective.effective.Effective

```
__builtin__.object ─┐
                    Effective
```

This class is the interface to the many-body module bhf. Currently the only option is to do a self consistent calculation of a new basis set.

G.47.1 Methods

__init__(*self*, *filename*)

Constructor

Parameters

`filename`: Name of the configuration file.
(type=String)

Overrides: __builtin__.object.__init__

check_config(*self*)

Check if the given configuration is valid.

Return Value

True if the current config is valid. False otherwise.
(type=boolean)

init_config(*self*)

Initialize the compiled core module. The core module is written in fortran and compiled as a shared object. This initialization is needed so the datastructures are allocated correctly and necessary variables set.

Return Value

True if the initialization went ok. False otherwise.
(type=boolean)

list_config(*self*)

Print the current configuration to screen.

Return Value

None

read_config(*self*, *filename*)

Reads a configuration file in the format: key=value populates a Python dictionary with the keys and corresponding values.

Parameters

`filename`: Filename of the config file.
(type=String)

Return Value

Dictionary with all elements in the configuration file.

setup_hf_orbits(*self*)

Interface function to the subroutine that does the actual calculations.

Return Value

None

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

G.47.2 Instance Variables

Name	Description
options	Internal representation of the options set in the config- file

G.47.3 Class Variables

Name	Description
required	Required options for a run. *(type=list)*

G.48 Module visualization.plot

G.48.1 Classes

- **EigData**: Class containing an eigenvalue dataset to plot.
  (Section G.49, p. 174)
- **PlotData**: Class containing a dataset to plot.
  (Section G.50, p. 174)

G.48.2 Functions

find_closest(*q*, *k*)

Return the index of the element of q, closest to k

Parameters

`q`: List of floats to compare.
(type=List of floats.)

`k`: Value to compare.
(type=float)

Return Value

The index of the element in q, closest to k.

get_diagonal(*vkk*)

Return the diaogonal of a matrix.

Parameters

`vkk`: 2-dim dataset.
(type=numpy array of rank 2 containing floats.)

Return Value

A list of floats containg the diagonal elements of vkk.

get_offdiag(*q*, *data*, *k*)

Return row of data with index i matching the closest q[i] to k.

Parameters

`q`: vector of floats to match with k.
(type=List of floats.)

`data`: 2 dim dataset.
(type=numpy array of rank 2 containing floats.)

`k`: Matching point.
(type=float)

Return Value

List of floats containing a row of the dataset.

plot_diagonal(*q*, *data*, *options*=`None`)

Plot the diagonal elements of a 2-dim dataset against a set of meshpoints. Optional dictionary is passed along.

Parameters

`q:`	Meshpoints. *(type=List of floats.)*
`data:`	2-dim dataset. *(type=numpy array of rank 2 containing floats.)*
`options:`	options to passed along. *(type=dict of parameters.)*

plot_eig_multiple(*data*, *num*, *options*=`None`)

Plot multiple datasets as eigenvalue plots

Parameters

`data:` The dataset to be plotted.
(type=An instance of the EigData class)

`options:` Optional dict of key-value pairs. The function currently support the followin keys:

- title: The title of the plot.
- copydir: The directory where a copy of the plot should be placed. If not specified, no plot is stored to file.

Return Value

None

plot_multiple(*data*, *options*=`None`)

Plot multiple datasets as traditional x,y plots.

Parameters

`data`: The dataset to be plotted.
(type=An instance of the PlotData class)

`options`: Optional dict of key-value pairs. The function currently support the followin keys:

- offdiag: If an offdiagonal plot of the dataset is needed, this option specifies which row/coloumn to plot.
- title: The title of the plot.
- xbound: The range on the x axis that should be plotted. Specified as a tuple (xmin,xmax).
- ybound The range on the x axis that should be plotted. Specified as a tuple (ymin, ymax).
- xlabel: The label on the x-axis.
- ylabel: The label on the y-axis.
- copydir: The directory where a copy of the plot should be placed. If not specified, no plot is stored to file.

Return Value

None

plot_offdiag(*q*, *data*, *k*, *options*=`None`)

Selects the row of a 2 dimensional dataset closest to a given k plots this row againt the meshpoints in q. Optional options are passed along.

Parameters

`q`: Meshpoints.
(type=List of floats)

`data`: 2 dim dataset.
(type=numpy array of rank 2 containing floats.)

`options`: options to passed along.
(type=dict of parameters.)

plot_single(*q*, *f*, *options*=`None`)

This method plots a single dataset f, against a mesh q. An optional options dictionary can contain additional parameters to the plot. Supported options are: legend: A legend for the dataset title: Plot title xlabel: Label for the x axis ylabel: Label for the y axis xbound: Boundaries for the plot along the x axis ybound: Boundaries for the plot along the y axis Additional options are ignored.

Parameters

`q:`	Points along the x axis *(type=List if floats)*
`f:`	Points for the y-axis *(type=List of floats)*
`options:`	Dictionary of additional parameters. *(type=Dictionary of parameters.)*

G.48.3 Variables

Name	Description
cname	**Value:** `'plot'` *(type=`str`)*
DEBUG	**Value:** `True` *(type=`bool`)*

G.49 Class visualization.plot.EigData

Class containing an eigenvalue dataset to plot.

G.49.1 Methods

__init__(*self*, *data*, *legend*)

Constructor.

Parameters

`self`:	An instance of this class.
`data`:	Eigenvalues to plot. *(type=List of floats.)*
`legend`:	A legend for this dataset. *(type=String)*

cut_data(*self*, *n*)

Return only the n first elements of an array. If n is less than the length of the array, return the complete array.

G.49.2 Instance Variables

Name	Description
data	Eigenvalues to plot. *(type=List of floats.)*
legend	A legend for this dataset. *(type=String)*

G.50 Class visualization.plot.PlotData

Class containing a dataset to plot.

G.50.1 Methods

__init__(*self*, *x*, *data*, *legend*)

Constructor.

Parameters

`self:`	An instance of this class.
`x:`	Points along the x axis. *(type=List of floats.)*
`data:`	Values corresponding to the points in x. *(type=List of floats.)*
`legend:`	A legend for this dataset. *(type=String)*

G.50.2 Instance Variables

Name	Description
data	Values corresponding to the points in x. *(type=List of floats.)*
legend	A legend for this dataset. *(type=String)*
x	Points along the x axis. *(type=List of floats.)*

References

[1] Python Programming Language – Official Website. *http://www.python.org/*

[2] T. Engeland, M. Hjorth-Jensen and G.R. Jansen. In preparation, 2008.

[3] S.K. Bogner, T.T.S. Kuo and A. Schwenk. *Phys. Rep.*, 386(1):1, 2003.

[4] J. Schaffner-Bielich. *Nucl. Phys. A*, 804(1-4):309, 2008.

[5] C. Kiefer, B. Sandöefer. arXiv:gr-qc/0804.0672v2, 2008.

[6] Y. Okada. arXiv: hep-ph/0708.2016v1, 2007.

[7] Review of Particle Physics. *J. Phys. G*, 33, 2006.

[8] T. Nakano et al. *Phys. Rev. Lett.*, 91(1):012002, 2003.

[9] S. Weinberg. *The quantum theory of fields*, Paperback ed., Cambridge University Press, New York, 2005.

[10] Quark model. *http://en.wikipedia.org/wiki/Quark_model/*

[11] M. Gell-Mann. *Phys. Rev.*, 92:833, 1953.

[12] T. Nakano and K. Nishijima. *Prog. Theor. Phys.*, 10:581, 1953.

[13] A. Pais. *Phys. Rev.*, 86:663, 1952.

[14] I.J.R Aitchinson and A.J.G Hey. *Gauge theories in particle physics*, Third ed., Taylor and Francis Group, Abingdon, 2003.

[15] M. Hjorth-Jensen. *Lecture notes in computational physics*, unpublished, Oslo, 2007.

[16] N . Ishii, S. Aoki, and T. Hatsuda. arXiv:nucl-th/0611096, 2006.

[17] H. Yukawa. *Proc. Phys. Math. Soc. Jpn.*, 17:48, 1935.

[18] M. Taketani, S. Nakamura and M. Sasaki. Proc. Theor. Phys, 6:581, 1951.

[19] J.D. Bjorken and S.D. Drell. *Relativistic Quantum Mechanics*, Cust. ed., McGraw Hill, Dubuque, 1998.

[20] R. Machleidt. *Adv. Nucl. Phys.*, 19:189, 1989.

[21] R. Machleidt, K. Holinde and C. Elster. *Phys. Rep.*, 149(1):1, 1987.

[22] R. Machleidt. in *Relativistic Dynamics and Quark-Nuclear Physics*, edited by M. B. Johnson and A. Picklesimer, John Wiley & Sons, New York, 1986.

[23] D.R. Entem and R. Machleidt. *Phys. Lett. B*, 524(1-2):93, 2002.

[24] J. Haidenbauer and Ulf-G. Meißner. *Phys. Rev. C*, 72:044005, 2005.

[25] T.A. Rijken, V.G.J Stoks and Y. Yamamoto. *Phys. Rev. C*, 59(1):21, 1999.

[26] K Heyde. *Basic ideas and concepts in nuclear physics*, 3. ed., IOP

Publishing, Bristol, 2004.

[27] D. J. Dean and M. Hjorth-Jensen. *Phys. Rev. C*, 69:54320, 2004.

[28] B. S. Pudliner, V. R. Pandharipande, J. Carlson, Steven C. Pieper and R. B. Wiringa. *Phys. Rev. C*, 56(4):1720, 1997.

[29] S.E. Koonin, D.J. Dean and K. Langanke. *Phys. Rep.*, 278:1, 1997.

[30] D. M. Ceperley. *Rev. Mod. Phys.*, 67(2):279, 1995.

[31] Takahiro Mizusaki, Michio Honma and Takaharu Otsuka. *Phys. Rev. C*, 53(6):2786, 1996.

[32] Michio Honma, Takahiro Mizusaki and Takaharu Otsuka. *Phys. Rev. Letters*, 77(16):3315, 1996.

[33] Yutaka Utsuno, Takaharu Otsuka, Takahiro Mizusaki and Michio Honma. *Phys. Rev. C*, 60(5):054315, 1999.

[34] M. Hjorth-Jensen, T. T. S. Kuo and E. Osnes. *Phys. Rep.*, 261:125, 1995.

[35] D.J. Dean, T. Engeland, M. Hjorth-Jensen, M.P. Kartamyshev and E. Osnes. *Prog. Part. Nucl. Phys.*, 53:419, 2004.

[36] A. L. Fetter and J. D. Walecka. *Quantum theory of many-particle systems*, Dover publications, New York, 2003.

[37] G. Hagen, T. PapenBrock, D.J. Dean and M. Hjorth-Jensen. In preparation, and private communications, 2008.

[38] R. D. Lawson. *Theory of the Nuclear Shell Model*, Clarendon Press, New York, 1980.

[39] P. Navrátil, J. P. Vary and B. R. Barrett. *Phys. Rev. C*, 62:054311, 2000.

[40] K. Suzuki and S. Y. Lee. *Prog. Theor. Phys.*, 64:2091, 1980.

[41] K. Suzuki. *Progr.Theor.Phys.*, 68:246, 1982.

[42] K Suzuki and R. Okamoto. *Prog. Theor. Phys.*, 92:1045, 1994.

[43] K. Suzuki and R. Okamoto. *Progr.Theor.Phys.*, 93:905, 1995.

[44] G. Hagen, M. Hjorth-Jensen and N. Michel. *Phys. Rev. C*, 73:064307, 2006.

[45] T.T.S Kuo, J.Shurpin, K.C. Tan, E. Osnes and P.J. Ellis. *Ann. Phys.*, 132(2):237, 1981.

[46] G.H. Golub and C.F Van Loan. *Matrix Computations*, 3. ed., The Johns Hopkins University Press, Baltimore, 1996.

[47] M. Moshinsky and Y.F. Smirnov. *The harmonic oscillator in modern physics*, Harwood academic publishers, Amsterdam, 1996.

[48] SciPy *http://www.scipy.org/*

[49] Python Reference Manual. *http://docs.python.org/ref/ref.html*.

[50] H. Djapo, B. Schaefer and J. Wambach. arXiv:nucl-th/0802.2646v2, 2008.

[51] M.L. Boas. *Mathematical methods in the physical sciences*, 3. ed., John Wiley & Sons, 2006.

[52] F. Madl, G. Shaw. *Quantum Field Theory*, rev. ed., John Wiley & Sons, Chichester, 1993.

[53] J.J. Sakurai. *Modern Quantum Mechanics*, rev. ed., Addison Wesley, Reading, 1994.

[54] M. Rotenberg et al. *The 3-j and 6-j symbols*, Mit Pr., 1960.

[55] A. de Shalit and I. Talmi. *Nuclear Shell Theory*, Academic Press, 1963.

[56] B.A. Brown. *Lecture Notes in Nuclear Structure Physics*, http://www.nscl.msu.edu/ brown/Jina-workshop/BAB-lecture-notes.pdf, unpublished, Michigan, 2005.

[57] G.E. Brown and A.D. Jackson. *The nucleon-nucleon interaction*, North Holland publishing company, 1976.

[58] P.J. Ellis and E. Osnes. *Rev. Mod. Phys.*, 49(4):777, 1977.