Exploring Many-Body Force Induction in the Similarity Renormalization Group in 1 Dimension

UNDERGRADUATE RESEARCH THESIS

Presented in Partial Fulfillment of the Requirements for Graduation with Honors Research Distinction in Physics in the College of Arts and Sciences of The Ohio State University

By

Matthias Heinz

Undergraduate Program in Physics

The Ohio State University

2018

Thesis Committee:

Prof. Richard Furnstahl, Advisor

Prof. Robert Perry

Prof. P Sadayappan

Abstract

A major goal of nuclear theory is to model atomic nuclei and make theoretical predictions of nuclear observables starting from inter-nucleon forces. Approaches starting from these inter-nucleon forces, known as ab-initio methods, face significant computational challenges due to the complexity of the nuclear system and the nature of the forces. The similarity renormalization group (SRG) method is often used in modern calculations to soften these interactions, which simplifies the problem thereby allowing ab-initio methods to be extended to larger systems. SRG, when applied to an A-particle system, induces many-body forces that are not accounted for when the A-body results are used to compute results for larger systems. The errors from these omitted induced many-body forces currently limit the application of SRG to small and medium systems, as for large systems these errors become too large for calculations to yield useful results. The SRG method takes an input from the user, the flow operator $\hat{G}_s$, which is taken to be the kinetic energy in most modern SRG calculations. Results have suggested that alternative choices for the flow operator may produce smaller induced many-body forces, which would allow calculations with SRG to be extended to larger systems.

We return to the 1-dimensional system of bosons in which the nuclear theory group at OSU initially explored the application of SRG to nuclear problems. As the results from this simple system are generalizable to full 3-dimensional calculations, we seek to test alternative flow operators in this 1-dimensional system, where visualizing and interpreting results is substantially easier. We develop a Python library to handle the setup of the physical system

and the SRG evolution. We compare the results obtained using this library to the results from an analogous paper by Jurgenson and Furnstahl in 2008 to verify the correctness of our implementation. We then use this framework to test induced 3-body forces for several 2-body flow operator choices. We discuss our preliminary results and offer some options for further exploration.

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Dick Furnstahl, for his patient and steady guidance over the years. After giving me a starting point, he was consistently available to clarify concepts and give feedback on my work. As I worked my way up from not knowing anything about quantum mechanics, he helped guide my intuition and encouraged me to apply the technical skills I had acquired in my research. His knowledge, wisdom, productivity, and integrity inspire me daily to become a better scientist and a better member of society.

I am forever indebted to my parents, who have made everything I have accomplished possible. My father, Ulrich Heinz, helped to develop how I think about the world and solving problems in it. My mother, Christiane Heinz-Neidhart, has helped me develop discipline, work ethic, and a reasonable structure for my daily life. Their overwhelming support for my education and my ambitions is the reason I am where I am today.

I would also like to thank my brother, Michael Heinz, for inspiring and supporting me. His work ethic and brilliance consistently reminds me that there is more that I could do and better ways I could do it. I can count on him for a listening ear when I need it and honest, useful feedback and advice when I want it.

I want to thank the other members of my committee, Prof. P Sadayappan and Prof. Robert Perry, for their helpful input on my thesis. I want to thank the members of the low-energy nuclear theory group at OSU during the time I was here; their comments during group meetings, conferences, and isolated conversations have helped develop my knowledge of the field as a whole, which has also helped my own research.

Finally, I would like to thank all of my friends for their support. They have been understanding of my sometimes erratic availability and encouraging of my ambitions.

Table of Contents

Chapters

List of Figures

List of Tables

Chapter 1
INTRODUCTION

Nuclear theory is concerned with modeling atomic nuclei and nuclear matter. Nuclear matter is made of protons and neutrons, which are generally referred to as nucleons. Nucleons interact primarily through the strong interaction, which together with the weak interaction governs the stability of nuclear isotopes. Nuclear theory seeks to answer open questions in four areas: what are the properties of nuclei, especially exotic isotopes; what are the properties of nuclear matter in astrophysical systems such as neutron stars; what can be learned about beyond Standard Model (BSM) physics from the nucleus; and how can more accurate models of nuclear systems be leveraged in a variety of applications, ranging from national security and energy to medicine [3].

Nuclear theory faces two major challenges when trying to model systems of nucleons at low energies. The first is that this is a quantum many-body problem. Quantum many-body theory is relevant to many different fields, including quantum chemistry and condensed matter theory. Many-body problems quickly become intractable when approached naively due to the combinatorial growth of the size of the problem with respect to the number of particles. The second major challenge of nuclear theory is that while the forces between nuclei are given by quantum chromodynamics (QCD), the theory underlying the strong interaction, QCD cannot be solved in closed form at energies relevant to nuclei. The strong interaction is between quarks and gluons, collectively referred to as partons, which make up

nucleons, instead of between nucleons themselves. In the past, these challenges forced nuclear physics to rely on phenomenological models, both for the form of the strong interaction and the treatment of the many-body problem. These models were typically fit to experimental data for certain nuclei and used to predict the properties of nearby nuclei. However, their predictive ability did not extend far outside the domain in which they were fit, limiting their range of applicability. Furthermore, they did not provide theorists with error estimates or any ways to tell when they became invalid.

With the Facility for Rare Isotope Beams (FRIB) on the horizon [4], there is demand for more accurate calculations of nuclear observables with new methods as well as for theoretical predictions of properties of heavy, exotic nuclei that have not been measured yet. Moreover, accurate models of nuclear matter are essential to understanding dense stars, supernovae, and certain astrophysical events, such as the collisions of neutron stars, which are hypothesized to be a location for the synthesis of heavy elements, such as gold and silver [5]. Improved nuclear models are also essential in predicting the rate of certain nuclear decays, such as neutrino-less double-beta decay, which is currently being searched for by many experiments to answer whether or not neutrinos are Majorana particles, that is, they are their own anti-particle [6].

In recent years, the improvements in computational hardware and the development of new computational methods has brought about a rapid expansion (see Fig. 1.1) in the range of nuclear isotopes able to be modeled via ab-initio calculations, which are calculations starting from the forces between nucleons as determined by scattering experiments and few-body nuclei. The improvements in computational hardware have come through the development and proliferation of high-throughput devices (such as GPUs) and the assembly of highly parallel systems. Current trends, shown in Fig. 1.2, suggest that a machine with exa-FLOP (floating-point operations per second) throughput will exist by 2020 and such systems will be commonplace soon after [7]. Much work is being done to ensure that the processing,
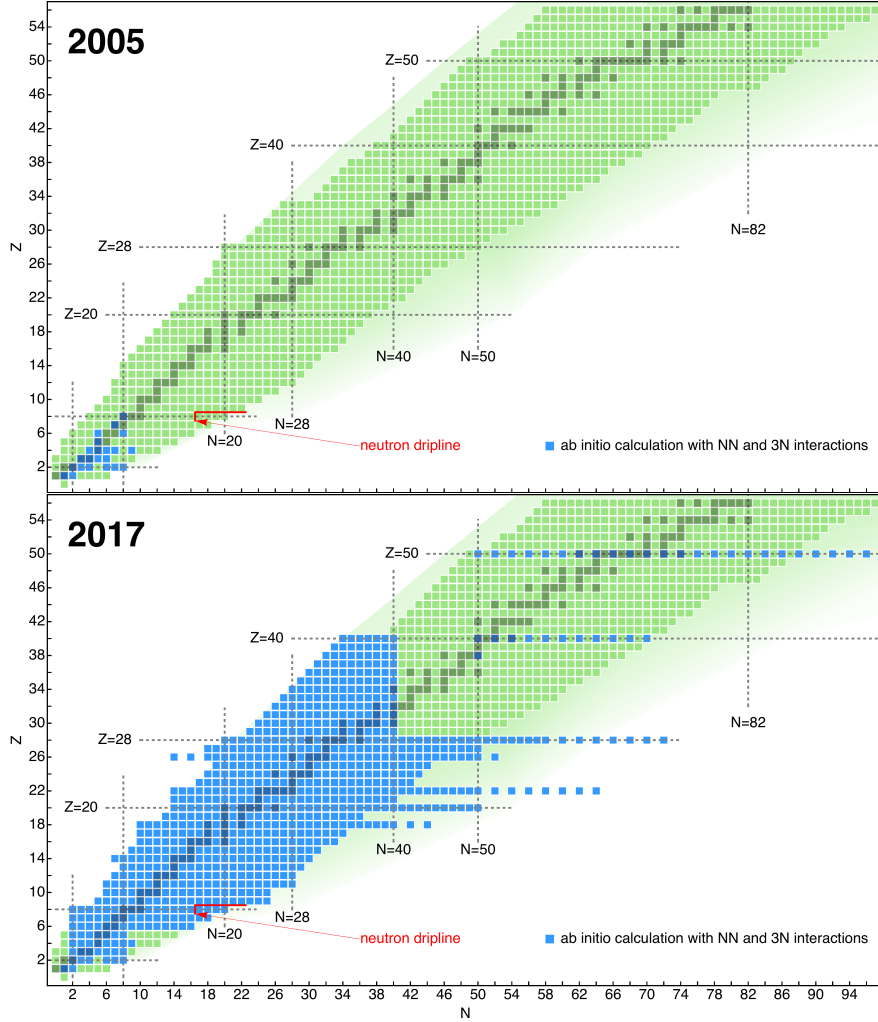
Figure 1.1: Nuclear charts for 2005 and 2017 demonstrating the range of nuclei modeled using ab-initio methods. Z and N are proton and neutron number of the isotope, respectively. Images courtesy of Heiko Hergert [1].

networking, and power-consumption of these systems will continue to scale as it has for the past two decades.

The computational methods used in modern low-energy nuclear theory come in three classes: many-body methods, effective field theory (EFT) methods, and renormalization group (RG) methods. Modern many-body methods, like the no-core shell model (NCSM) [8], seek to model nuclei from first principles and work in conjunction with RG and EFT

3
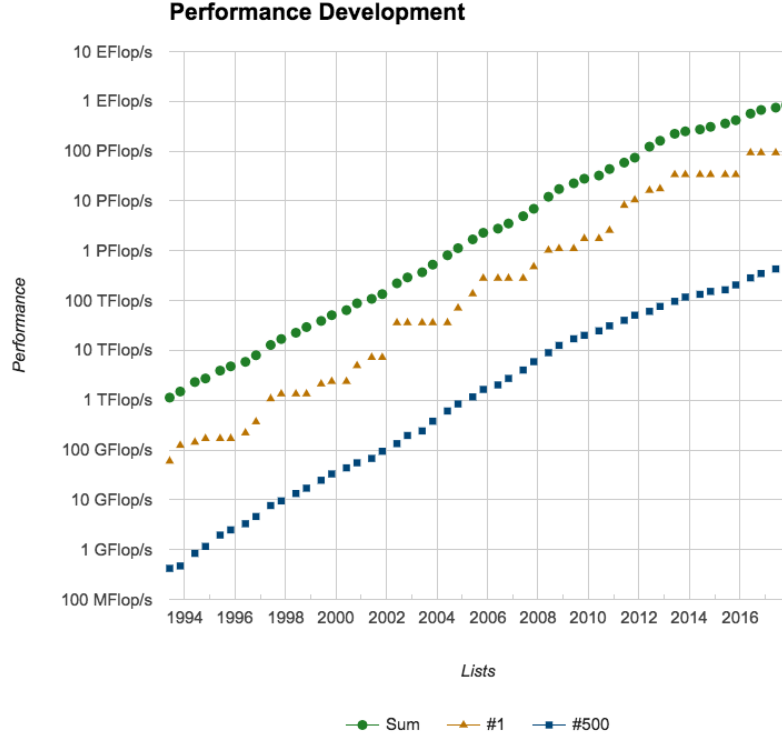
Figure 1.2: According to the Top500 list of supercomputers, there should be an exa-scale cluster in 2020.

methods to make these calculations feasible. EFT methods offer systematically improvable, model-independent expansions of nuclear interactions and other operators that allows for importance truncation and uncertainty quantification [9][10]. RG methods are applied to interactions given by EFT methods and modify these interactions to match the resolution relevant to the problem at hand [11]. These methods have already been used to provide both theoretical predictions that improve on previous calculations and theoretical predictions for nuclei that were unable to be modeled previously. The use of these methods in low-energy nuclear calculations is the state of the art. As a result, questions regarding their application are open problems and the focus of extensive research.

The similarity renormalization group method (SRG) is an interaction softening method

from the class of RG methods whose use in nuclear theory was first explored in a 1-dimensional setting at OSU in 2007 [12]. It is a class of continuous unitary transformations that decouple large off-diagonal matrix elements in the interaction Hamiltonian, softening the potential as a result. The evolution of the Hamiltonian to a decoupled form allows a truncated subspace of the original basis to be used in later calculations without affecting the observables such as the energy eigenvalues. This basis truncation offers a significant reduction in the size of the problem. SRG is frequently used in modern nuclear theory calculations to soften interactions and extend the range of certain calculations to larger systems. In our work, we return to a simple 1-dimensional setting to study features of SRG and seek to understand how to optimize its use in many-body calculations.

1.1 Contributions

Our main contributions are:

- We have developed an open-source Python library with easy-to-use abstractions for the SRG method for use by others to test SRG in a simple setting.

- We verify the results published in the 2008 Jurgenson paper for 2-body and 3-body systems, validating our implementation for those systems and implying its correctness for general A-body systems. We also provide a suite of tests for the library, verifying that it fails gracefully when misused and correctly calculates the results of simple analytically solvable cases when used correctly.

- We explore some alternative transformation generators and seek to quantify their performance relative to the standard generator that is currently used. We discuss the results of these calculations and offer some preliminary analysis.

1.2 Outline

The rest of the thesis is as follows:

- In chapter 2, we review the matrix representation of quantum operators. We then discuss the details of two different bases used, relative Jacobi momentum coordinates and 1-dimensional symmetrized harmonic oscillator states. We then discuss the details of the SRG method and explain the need for a careful revisit of SRG in a 1-dimensional setting.

- In chapter 3, we explain the design of the Python library. Much of the effort on this project went into making the library design logical and simple-to-use, so this chapter will explain the abstractions made and the reasoning behind those decisions.

- In chapter 4, we explain the approach taken to reproducing the 2008 Jurgenson results. We outline our approach to testing new transformation generators once our implementation has been validated.

- In chapter 5, we present our results. We discuss in detail the various different ways in which we verify that what we are doing is correct at each stage of the exploration. Finally, we show some alternative transformation generators that induce smaller 3-body forces.

- In chapter 6, we summarize our results, point to open questions and specific points of interest, and discuss what features of the `srg1d` library still need to be implemented.

Chapter 2

SIMILARITY RENORMALIZATION GROUP FORMALISM

In this chapter, we introduce the physics concepts and formalism necessary to understand the 1-dimensional problems to which SRG is applied. We then introduce SRG and discuss details and open questions regarding its use.

2.1 Quantum Mechanics Operators

Operators in quantum mechanics act linearly on state vectors, which represent specific states of the system. Each operator corresponds to some physical quantity; for example, the Hamiltonian, $\hat{H}$, which is the sum of kinetic and potential energy operators, $\hat{T}$ and $\hat{V}$ respectively, is connected with the possible energies of the system. A state, which in Dirac notation is denoted by an abstract ket, $|\Psi\rangle$, is said to be an eigenstate with eigenvalue ω of some operator, $\hat{\Omega}$, when the following equation holds:

$$\hat{\Omega} |\Psi\rangle = \omega |\Psi\rangle. \tag{2.1}$$

These eigenvalues are the *observables* (i.e. measurable quantities) of quantum mechanical systems.

Operators exist in a Hilbert space spanned by a basis of state vectors, $|\Psi_i\rangle$. An operator's
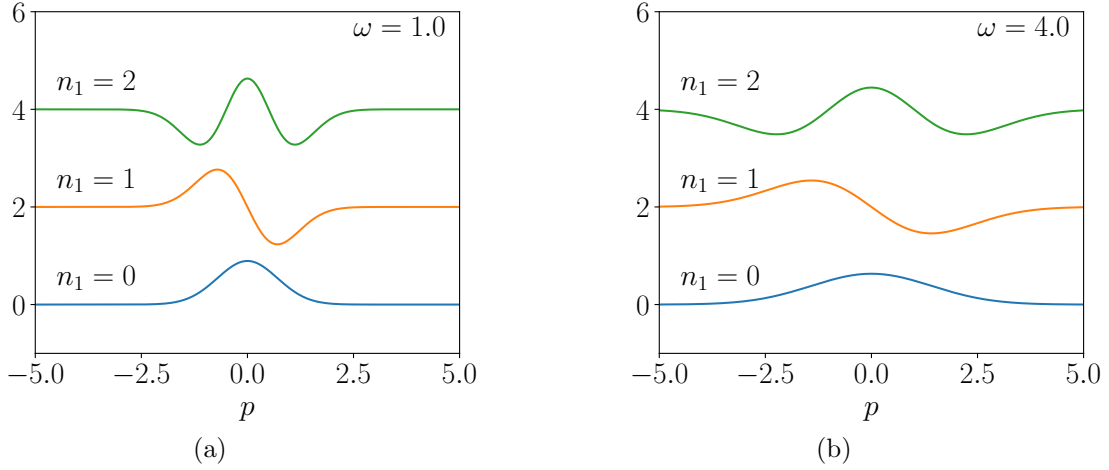
Figure 2.1: 1-dimensional harmonic oscillator wavefunctions for the three lowest energy states for (a) $\omega = 1$ and (b) $\omega = 4$.

representation with respect to a basis is given by its matrix elements,

$$\hat{\Omega}_{ij} = \langle \Psi_i | \, \Omega \, | \Psi_j \rangle . \tag{2.2}$$

An operator can be transformed to a different basis through a unitary transformation, which preserves its eigenvalues and thus its observables. Examples of 1-dimensional single-particle bases include particle position, x, particle momentum, p, and the 1-dimensional harmonic oscillator basis, which will be discussed next.

2.1.1   1-Dimensional Harmonic Oscillator

The 1-dimensional harmonic oscillator is a single particle in a purely quadratic potential, $\hat{V}(x) = m\omega^2 \hat{x}^2 / 2$. The Hamiltonian for the quantum harmonic oscillator is given by

$$\hat{H}_{HO} = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\omega^2\hat{x}^2, \tag{2.3}$$

which has eigenstates $|n\rangle$, which have corresponding wavefunctions

$$\Psi_n(x) = \frac{1}{\sqrt{2^n n!}} \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-m\omega x^2/2\hbar} H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right) \tag{2.4}$$

$$\Phi_n(p) = \frac{(-i)^n}{\sqrt{2^n n!}} \left(\frac{1}{\pi m\hbar\omega}\right)^{1/4} e^{-p^2/2m\hbar\omega} H_n\left(\frac{p}{\sqrt{m\omega\hbar}}\right) \tag{2.5}$$

where $H_n(x)$ is the n-th Hermite polynomial. The three lowest energy momentum-space wavefunctions, $\Phi_n(p)$, can be seen in Fig. 2.1. The harmonic oscillator Hamiltonian can also be rewritten as

$$\hat{H}_{HO} = \hbar\omega\left(\hat{a}^\dagger\hat{a} + \frac{1}{2}\right), \tag{2.6}$$

where $\hat{a}^\dagger$ and $\hat{a}$ are raising and lowering operators which act on the harmonic oscillator eigenstates like

$$\hat{a}^\dagger |n\rangle = \sqrt{n+1}\,|n+1\rangle \tag{2.7}$$

$$\hat{a} |n\rangle = \sqrt{n}\,|n-1\rangle. \tag{2.8}$$

For an operator, $\hat{V}$, represented with respect to a momentum basis, $V(p, p')$, a transformation to a harmonic oscillator basis can be achieved by the following calculation:

$$\langle n| \hat{V} |n'\rangle = \iint V(p, p')\Phi_n^*(p)\Phi_{n'}(p')\,dp\,dp'. \tag{2.9}$$

2.1.2 Many-particle states

For a system of A non-interacting particles, the energy eigenstates of the system can be written as a product of the eigenstates of the single particle Hamiltonians:

$$|\Psi\rangle = \prod_{i=1}^{A} |\Psi_k\rangle_i \tag{2.10}$$

where $|\Psi_k\rangle_i$ is the k-th eigenstate of the i-th particle's single particle Hamiltonian. For A non-interacting particles in a harmonic oscillator potential, these product states are denoted by:

$$|n_1 n_2 ... n_A\rangle = \prod_{i=1}^{A} |n_i\rangle_i. \qquad (2.11)$$

2.2 1-Dimensional System of Bosons

The Hamiltonian for a 1-dimensional system of A identical bosons with mass m that interact via a local two-body potential is as follows:

$$\hat{H} = \frac{1}{2m} \sum_{i=1}^{A} \hat{k}_i^2 + \sum_{i=1}^{A} \sum_{j=i+1}^{A} \hat{V}(x_i - x_j). \qquad (2.12)$$

In this equation, the $\hat{k}_i$ are the single-particle momenta and the x_i are the single-particle coordinates. A local potential is a function of the distance between two particles, as opposed to a function of both of their absolute coordinates.

2.2.1 Jacobi Coordinates

A factorization of the Hamiltonian into a center-of-mass component, which is unaffected by the local potential, and a component relative to the center of mass is possible. This is achieved by a transformation to relative momentum Jacobi coordinates [2]. These are defined to be:

$$p_i = \sqrt{\frac{i}{i+1}} \left(\frac{1}{i} \sum_{j=1}^{i} (k_j - k_{i+1}) \right), \qquad (2.13)$$

with k_i being the single-particle momenta as mentioned above. It is worth noting that for a system of A particles, we only need $A - 1$ Jacobi momentum coordinates. We can get $\hat{V}$ in terms of incoming Jacobi momentum p and outgoing momentum p' by taking the following

Fourier transform of $\hat{V}(x_1 - x_2)$

$$\hat{V}(p, p') = \int \hat{V}(\sqrt{2}l_1)e^{-i(p-p')l_1} \, dl_1, \qquad (2.14)$$

where l_1 is the coordinate conjugate of p_1, $(x_1 - x_2)/\sqrt{2}$. Removing the center-of-mass component of the Hamiltonian, we find the new Hamiltonian to be

$$\hat{H} = \frac{1}{2\mu}\sum_{i=1}^{A-1}\hat{p}_i^2 + \sum_{i=1}^{A-1}\hat{V}(p_i, p_i'), \qquad (2.15)$$

where μ is the reduced mass given by $\mu = m/A$ for particles of equal mass. For the purposes of this work, we will be starting with a potential defined with respect to the Jacobi momentum coordinates, avoiding the process of Fourier transforming a local coordinate-space potential.

2.2.2 Transformation To Harmonic Oscillator States

In momentum space, SRG evolutions require separate treatment of 2-body, 3-body, and higher-body potentials, to avoid delta functions caused by spectator particles [13]. A particle is a spectator particle when it is in a state where it does not interact with two particles that are interacting with each other. To avoid the cognitive load of handling all these potentials separately, we can make a transformation to a discrete basis. The discrete basis of choice for this project is the harmonic oscillator basis with respect to the Jacobi coordinates. This transformation can be achieved as shown in Eq. 2.9. From here on out, $|n_i\rangle$ will mean the n-th harmonic oscillator state with respect to the i-th Jacobi coordinate. So for a full treatment of an A-particle system, we will need product states for $A-1$ Jacobi coordinates,

$$|n_1 n_2 ... n_{A-1}\rangle = \prod_{i=1}^{A-1} |n_i\rangle. \qquad (2.16)$$

11

2.2.3 Symmetrizing the Harmonic Oscillator Basis

According to the spin-statistics theorem, the state of a system of bosons must be symmetric under any permutation of the particle coordinates [14]. We approach the problem of generating a basis of states that reflect this symmetry recursively, by first symmetrizing the 2-body system and then going from a symmetrized $(A-1)$-body basis to a symmetrized A-body basis. At each stage, we must diagonalize the symmetrizer, the form of which we will show for the 2-body and 3-body cases.

For the 2-particle system, we are only working with the first Jacobi coordinate and the harmonic oscillator states corresponding to it, $|n_1\rangle$. The symmetrizer is $\hat{S} = (1 + \hat{P}_{12})/2$, where $\hat{P}_{ij}$ is the operator that exchanges the coordinates between i-th and j-th particles. Because harmonic oscillator wavefunctions are either even for even n or odd for odd n, $\hat{P}_{12}|n_1\rangle = (-1)^{n_1}|n_1\rangle$ and the symmetrizer is already diagonal with eigenvalue 0 for odd n_1 and eigenvalue 1 for even n_1. We select the states with eigenvalue 1 to create our symmetric harmonic oscillator basis for the 2-particle system.

To generate the partially symmetrized basis for the 3-body system, we take the states $|n_1 n_2\rangle$ where the possible n_1 values come from the symmetrized 2-body basis. The general 3-body symmetrizer is governed by the permutation group S_3, generated by $\hat{P}_{12}$ and $\hat{P}_{23}$, and has the form

$$\hat{S} = \frac{1}{6}(1 + \hat{P}_{12} + \hat{P}_{23} + \hat{P}_{12}\hat{P}_{23} + \hat{P}_{23}\hat{P}_{12} + \hat{P}_{12}\hat{P}_{23}\hat{P}_{12}) \tag{2.17}$$

Since the states included in our to-be-symmetrized basis are already symmetric with respect to $\hat{P}_{12}$, the symmetrizer simplifies to $\hat{S} = (1 + 2\hat{P}_{23})/3$. The matrix elements of $\hat{P}_{23}$ in our partially symmetrized basis are

$$\langle n_1' n_2'| \hat{P}_{23} |n_1 n_2\rangle = \delta_{N',N} \langle n_1' n_2'|n_1 n_2\rangle_3 , \tag{2.18}$$

where $N' = n_1' + n_2'$, $N = n_1 + n_2$, and $\langle n_1' n_2'|n_1 n_2\rangle_3$ is the 1-dimensional harmonic oscillator

transformation bracket for particles with mass ratio 3. This transformation bracket value is calculated as

$$\langle n'_1 n'_2 | n_1 n_2 \rangle_3 = \frac{\sqrt{n_1! n_2!}}{\sqrt{n'_1! n'_2!}} \sum_{k=0}^{n'_1} \binom{n'_1}{k} \binom{n'_2}{n_2 - k} \left(\frac{1}{2}\right)^{n'_1 + n_2 - 2k} \left(\frac{\sqrt{3}}{2}\right)^{n'_2 - n_2 + 2k} (-1)^{n_2 - k}. \quad (2.19)$$

Diagonalizing $\hat{S}$ will give eigenvectors that are normalized linear combinations of product states with the same total harmonic oscillator number N. We again select the ones with eigenvalue 1 and keep those as our fully symmetrized 3-body basis. While the project leading up to this thesis only worked with 2-particle and 3-particle systems and thus only the treatment of symmetrizing those bases is relevant to this work, the procedure is generalizable to symmetrize up to any A-particle basis. The details of this procedure are explained in Ref. [2].

2.2.4 Matrix Elements in the 3-Body Harmonic Oscillator Space

From Eq. 2.9, we are able to transform both parts of our 2-body Hamiltonian into the symmetrized 2-body harmonic oscillator basis. In order to work in a symmetrized 3-body basis, we need to compute the kinetic energy for the 3-body system and embed the 2-body potential in the 3-body space. Both the A-body kinetic energy and the A-body embedded 2-body potential are defined with respect to their $(A-1)$-body counterparts [2], so we will set up the discussion to give those formulas. Let $|N_A i_A\rangle$ be a symmetrized A-body state with total harmonic oscillator number N_A and degeneracy label i_A. $|N_A i_A\rangle$ is defined as

$$|N_A i_A\rangle = \sum_{l=1}^{k} c_l |N_{A-1} i_{A-1}; n_{A-1}\rangle, \quad (2.20)$$

where k is the number of states in the linear combination, the c_l are the coefficients of the linear combination of product states that make up the symmetrized state, $|n_{A-1}\rangle$ is a harmonic oscillator state with respect to the $(A-1)$-th Jacobi coordinate, and $|N_{A-1} i_{A-1}\rangle$

is a symmetrized $(A-1)$-body state. The A-body kinetic energy is calculated as

$$
\begin{aligned}
\langle N'_A i'_A | \hat{T}_A | N_A i_A \rangle = \sum_{l=1}^{k} \sum_{l'=1}^{k'} c_l c'_{l'} (\langle N'_{A-1} i'_{A-1} | \hat{T}_{A-1} | N_{A-1} i_{A-1} \rangle \delta_{n'_{A-1}, n_{A-1}} \\
- \frac{\omega}{4} \delta_{N'_{A-1}, N_{A-1}} (\sqrt{(n_{A-1}+1)(n_{A-1}+2)} \delta_{n'_{A-1}, n_{A-1}+2} \\
+ \sqrt{n_{A-1}(n_{A-1}-1)} \delta_{n'_{A-1}, n_{A-1}-2} \\
- (2n_{A-1}+1) \delta_{n'_{A-1}, n_{A-1}})).
\end{aligned}
\tag{2.21}
$$

Similarly, the embedded 2-body potential in the A-body basis is calculated as

$$
\langle N'_A i'_A | \hat{V}_A | N_A i_A \rangle = \frac{A}{A-2} \sum_{l=1}^{k} \sum_{l'=1}^{k'} c_l c'_{l'} \langle N'_{A-1} i'_{A-1} | \hat{V}_{A-1} | N_{A-1} i_{A-1} \rangle \delta_{n'_{A-1}, n_{A-1}}.
\tag{2.22}
$$

2.3 Similarity Renormalization Group

The similarity renormalization group (SRG), whose use in low-energy nuclear physics was initially explored at OSU [12], is one method of reducing the computational complexity of low-energy nuclear calculations. The idea behind it is to continuously unitarily transform the operator of interest (for example, the Hamiltonian) into a simpler form. This simpler form is chosen to allow the large basis to be truncated without affecting the operator eigenvalues, which are essential to the truncated operator's utility in later calculations. The SRG transformation is given by the flow equation for the evolving Hamiltonian $\hat{H}_s$,

$$
\frac{d\hat{H}_s}{ds} = [\hat{\eta}_s, \hat{H}_s],
\tag{2.23}
$$

where $[A, B]$ is the matrix commutator and $\hat{\eta}_s$ is the generator of the transformation which is defined as

$$
\hat{\eta}_s = [\hat{G}_s, \hat{H}_s],
\tag{2.24}
$$

where $\hat{G}_s$ is known as the flow operator. The changes in the Hamiltonian over the course of the SRG evolution are absorbed into the potential, $\hat{V}_s$, leaving the kinetic energy, $\hat{T}$,

independent of s, the flow parameter. s is taken to be 0 for the un-evolved Hamiltonian. It is often convenient to use $\lambda = 1/s^{1/4}$ as an alternative flow parameter, so instead of going from $s = 0$ towards $s = \infty$ over the course of an SRG transformation, you are going from $\lambda = \infty$ towards $\lambda = 0$. For our purposes, $\lambda = 40$ (or $s = 3.9 \times 10^{-7}$) will be a good enough place to start with the initial Hamiltonian.

2.3.1 Induced Many-Body Forces and Flow Operators

The SRG evolution is fully unitary for an A-body operator when done in the A-body space. However, working in the full many-body space is only feasible for the smallest of systems, due to the combinatorial scaling of the basis size with respect to A. When evolving an A-body system operator in some smaller basis, the SRG evolution induces many-body forces which show up as an error when eventually computing operator observables further down the line. This error has limited the SRG's domain of applicability to calculations for small to medium-sized systems.

Certain results have suggested that alternative choices for the flow operator, $\hat{G}_s$, could reduce this many-body force induction and thereby reduce the error induced by applying SRG to calculations [15]. The standard choice for $\hat{G}_s$ is the kinetic energy, which is easy to calculate and represent in most problems. Alternative flow operator choices have been explored previously, but not thoroughly.

2.3.2 Value of the 1-Dimensional Laboratory

SRG was explored initially in the 1-dimensional setting, which made it easy to test and allowed those exploring it to learn a great deal, such as the more careful treatment necessary for A-body evolutions in momentum space. Moreover, the setup of the 1-dimensional system is analogous to that of more complex calculations, so the results from the 1-dimensional setting generalize to application of the SRG in 3-dimensional calculations. If there is something

to be learned about the connection between flow operator and many-body force induction, it is best comprehensively explored in a simple 1-dimensional problem and then generalized to expensive, difficult 3-dimensional calculations.

In chapter 3, we will explain the structure of the Python framework designed to make it easy to explore SRG in this 1-dimensional setting.

Chapter 3
SRG1D PYTHON LIBRARY

In this chapter, we discuss the design and features of the `srg1d` Python library. The development of this library was a significant portion of the work that led to this thesis. We will begin by discussing the practical importance of a careful, conscious approach to programming for physics, followed by a review of some programming concepts and terms to facilitate the explanation of the library. We will also speak briefly about the Python programming language and the reasons it was chosen for this project. Then, we will move into the design of the library overall as well as the specific parts. Finally, we will discuss details regarding the implementation and testing the library.

3.1 Philosophy of Programming for Physics

The three principles that guided the programming done for this work are modularization, validation, and abstraction. By writing modular code, we isolate logically independent portions of our project that offer some functionality that is useful on its own. These independent parts can then be applied only to those problems to which they are useful. We can also validate the implementation of these isolated functions, testing them through unit tests or by comparing their results to the results of other implementations. Unit tests are especially useful because they isolate specific use-cases. When changes are made to the codebase, running a comprehensive suite of unit tests can give developers a good idea of what features

these changes affected and where they can go to fix the new bugs. Finally, with properly modular code, it is possible to offer logical abstractions that allow the user to avoid thinking about low-level implementation details and work in terms of objects that are relevant to the problem at hand.

These principles are realized by most software engineering projects, where maintainability and collaboration are central concerns. However, we feel that they are equally valuable when developing codebases for physics research, for the same reasons. Projects that follow these principles will allow for easier collaboration and set a solid foundation for projects that are extensions of the current work, in addition to bug detection and identification benefits of the approach.

3.2 Review of Programming Concepts

This review assumes the reader has a basic understanding of programming and aims to explain certain programming concepts and communicate their practical importance. This will better equip readers to understand the rest of the chapter.

- **Classes** are a way for programmers to create complex objects for cases where the programming language basic types (integers, floating point numbers, strings, booleans, arrays) are not enough to meet their needs. A class has some internal data, as well as methods and properties which interact with that internal data. A user can create an object of a class by calling its constructor method. It can interact with the object through its interface.

- **Interfaces** define how a user can interact with an object, through methods and properties. Methods typically change the object in some way or do some significant evaluation of the object internals. Properties allow users to extract certain properties of the object, without changing them.

- **Collections** are classes that contain a bunch of similar objects and define how users can access the objects in the collection. Different variations of collections exist to enforce some conditions on the collection, such as no duplicates or limited ways to access objects in the collection.

- **Indexing** is an operation defined on some types of collections which allows a user to get any object in the collection by providing the collection the proper index. The simplest example of indexing is a standard array, which most people who have used a programming language have seen.

- **Iterating** on a collection is a way for a user to sequentially access every object in a collection. For a user, accessing objects in a collection via iteration requires the user to know as little as possible about the collection or how to actually access the objects in the collection. Some collections guarantee that the order of the objects accessed via iteration is the same every time as long as the collection is not modified.

- **Mutability** is the ability of an object to be modified by the user after it is created. Mutable objects (or objects that can change by something the user does) give the user more flexibility in terms of possible functionality, but open the door for potential misuse by the user when properties of the collection are saved by the user, the object is modified, and those saved properties are not updated to reflect the changes. Immutable objects make sense in cases where the objects should not change, like an array of the days of the week.

- **References** allow multiple objects to have access to the same thing. If the reference to an object is given to two different objects and the first modifies the referenced object, the changes will be reflected in the second object as well. This can lead to difficult-to-manage behavior. As a result, it is good to use references in conjunction with immutable objects, because, in that case, both objects can be sure that the object

their reference refers to will never change.

3.3 A Note on the Python Programming Language

Python is a dynamically-typed, interpreted, high-level programming language that is used for general-purpose programming [16]. Python is used by many, both inside and outside the sciences, and according to StackOverflow, it has "a solid claim to being the fastest growing major programming language" [17]. Python currently has two supported major versions, Python 2 and Python 3, with Python 2 being supported for developers unable to immediately transition to Python 3 [18].

Although the numerical libraries in Python are quite fast due to their low-level C bindings, Python is still not a high-performance language. Projects written in Python that are not satisfied with their performance can profile their code to identify bottlenecks and take advantage of Python bindings to low-level languages like C and Fortran to speed up work-intensive portions of their program. The Pareto principle applied to programming states that 80% of the work of a program will be done by 20% of the code, which we call the work-intensive part of the code. A profiling tool like `cProfile` can be used to help the developer identify these portions [19]. Python's C implementation, Cython, has native C bindings that allow developers to call C subroutines, which will allow them to achieve their desired performance [20].

However, despite the general computational challenge faced by nuclear theory in general, the simple 1-dimensional system explored here is numerically relatively simple. Thus, high throughput and efficient memory usage are not requirements for these calculations to be done in a reasonable amount of time on any modern personal computer. In addition, Python is extremely expressive with a large standard library and well-documented third party libraries, making developer productivity very high. This was a priority in this research, leading to the decision to work in Python.

The next section discusses the general design of the library and references the interfaces for various different classes. A feature of Python is that users with sufficient knowledge of the internal representation of data in Python classes can directly access those internals and use or modify them how they see fit. This means that published interfaces don't truly limit what a user can do with a library and things like immutability are difficult or impossible to truly enforce. However, Python developers have the philosophy that all Python users are "consenting adults" [21]. This means that the documentation of interfaces and conditions on the objects like immutability are communicated to users in an understanding that they should stick to these documented designs if they expect things to work as advertised.

3.4 Library Design

The class-based abstractions offered by the library offer consistent representations of different logical classes of objects present in any SRG calculation. By offering the user classes and functions that handle much of the complex, but frequently required functionality in any SRG calculation, the library allows the user to focus on exploration and research rather than reinventing the wheel. Behind the abstractions in the library are also several checks that ensure the physical correctness and consistency of what the user is doing. This further boosts user productivity, as the library provides transparent, clear errors as to what conditions were not met, allowing the user to quickly understand where the error in their program is.

The library is roughly split into two logical halves. The data representation half (`state`, `basis`, and `operator`) handles the representations of the matrices for the operators in the calculations. Its primary goal is to add the context to the numerical representation of matrices and as a result make transformations of those matrices easier to achieve. The evolution half of the library (`srg`) handles the SRG transformation of the data. While the data representation classes correspond to concrete objects, the `srg` class is more of a state machine that allows the user to put some data in, turn the crank, and get out intermediate

```
                    evolves
                      via
  ┌──────────────┐            ┌──────────────┐
  │   operator   │──────────▶│     srg      │
  └──────────────┘            └──────────────┘
         │
       holds
       data
       with
      respect
       to a
         ▼
  ┌──────────────┐
  │    basis     │
  └──────────────┘
         │
       is an
      array of
         ▼
  ┌──────────────┐
  │    state     │
  └──────────────┘
```
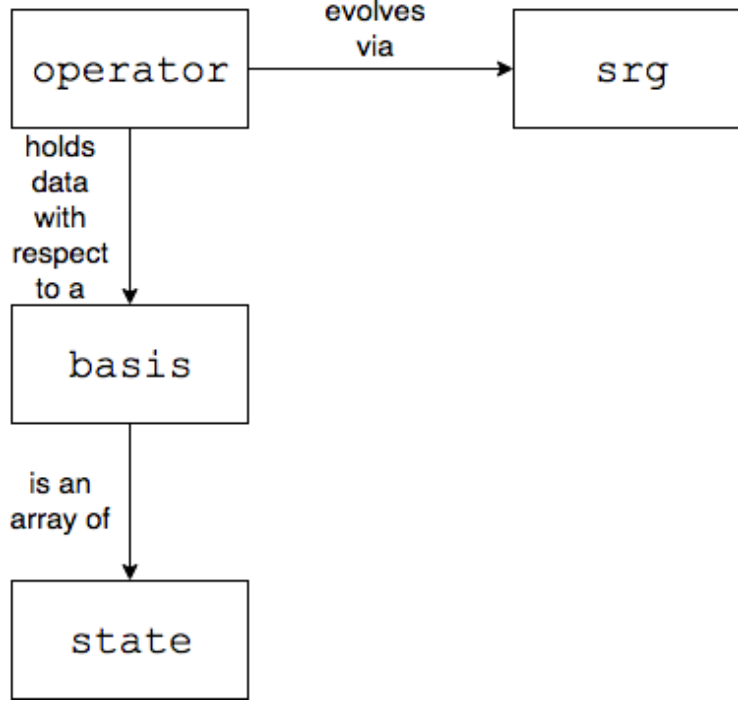
Figure 3.1: A general structure for the modules in the `srg1d` library.

and final results.

3.4.1 `state`

The `state` module provides the fundamental building blocks for the data representation side of the library. It contains two classes, `ho_state_2body` and `ho_state_nbody`, that implement the same simple interface. The interface for both classes simply defines a constructor and `val`, a method to evaluate the harmonic oscillator wavefunction that the state corresponds to at some momentum. These classes are intended to add context data (harmonic oscillator number, number of particles, Dirac notation) to the wavefunction that will simplify the evaluation of the wavefunction and allow for the construction of states with more complex structures due to additional physical conditions imposed on the states, like symmetry. Because these states are only intended to be used to get wavefunction values, their interface

defines them to be immutable.

`ho_state_2body` represents a 2-body harmonic oscillator state. `ho_state_nbody` is different from `ho_state_2body` in that it represents a general linear combination of n-body product states. The only conditions on this linear combination of product states are that it is normalized and that the total harmonic oscillator number for each product state in the linear combination is the same. `ho_state_2body` is the basic building block of these general n-body product states.

It is worth mentioning that `ho_state_nbody` does not enforce any symmetry on these states. The conditions imposed on the linear combination of product states are the minimum conditions for such a state to be physically useful. The burden of ensuring proper symmetry falls on the operations creating the states, which is handled elsewhere in the `srg1d` library. Thus, for the purposes of an SRG calculation, a user should never have to deal with the `state` module directly.

3.4.2 `basis`

The `basis` module provides a basic interface for a `basis` class and two classes, `p_basis` and `ho_basis`, that implement this interface. The `basis` class interface comprises only a constructor, but it also defines `__len__`, giving the size of the basis, and `__getitem__`, allowing the user to access a state by its index. These additionally defined methods make any basis indexable and iterable, with iterability being especially valuable for working with bases. The classes that implement the `basis` interface are also defined to be immutable, because after construction these bases have certain properties (symmetry, completeness, etc.) that would be disturbed by the removal, addition, or modification of a state in the basis.

`p_basis` is the representation of a momentum basis. Strictly speaking, a momentum basis is a continuous basis, that is operators need to be defined for any pair of real momenta. However, for numerical purposes, it makes sense to discretize the momentum space and

impose some upper and lower cutoffs on momenta included in the basis. The upper and lower cutoffs on a discretized momentum basis are left to the user, who can judge what the range of relevant momenta is. The discretization scheme for the points between those upper and lower bounds is given by a Gaussian quadrature, which provides points p_i and weights w_i at which to evaluate the function. This provides us with the momentum basis for a single Jacobi momentum. From the single momentum basis, we can generate the two Jacobi momentum basis by taking the Cartesian product between two single momentum bases, with the weight assigned to to each new state being the product of the two weights corresponding to the states in the ordered pair. This approach can be used to generate a basis for any number of Jacobi momenta. The data representation of the n-body momentum basis is simply a list of these states, which are represented as n-tuples with $n-1$ Jacobi coordinates and finally a weight for the state.

`ho_basis` is the representation of a harmonic oscillator basis. This is simply a list of harmonic oscillator states (2-body or n-body) up to some maximum total harmonic oscillator number, N_{max}. The standard constructor for `ho_basis` generates a basis of 2-body states with proper symmetry. `ho_basis` has a second constructor `ho_basis.from_basis` that takes any n-body `ho_basis` object and generates an appropriately symmetrized $n+1$-body basis. Additionally, each n-body basis has a reference to the $(n-1)$-body basis used to generate it, which is useful for functions which need to recurse down to the 2-body basis. Because generating bases with appropriate symmetry is an expensive operation, it is important to avoid recreating `ho_basis` objects. Thus, the `srg1d` library makes an effort to work with references to existing objects whenever it is possible.

3.4.3 `operator`

The `operator` module provides a simple `operator` class which couples a matrix with a basis that corresponds to its representation along with several additional useful methods that work

on operators. The packaging of a matrix and a basis into one object reduces the potential for bugs when calling methods that require both the data and the basis, which occurs frequently in the setup of an SRG calculation. It also ensures that methods requiring operators can be certain that the basis and matrix are consistent, so methods do not need to do redundant checks, such as ensuring the basis and data sizes match.

The methods provided by the `operator` module do one of three things: generate operators, transform operators, or embed operators. The operator generating methods, `create_ho_basis_kinetic_energy`, `create_p_basis_kinetic_energy`, and `create_kinetic_energy`, are methods that calculate the kinetic energy operator in a certain basis and return a corresponding `operator` object. While the general n-body kinetic energy can be calculated directly, the n-body kinetic energy is calculated recursively, with the 2-body kinetic energy being the base case. This is where the n-body `ho_basis` internal reference to the $(n-1)$-body `ho_basis` comes in handy.

The `transform_operator_to_p_basis`, `transform_operator_to_ho_basis`, and `transform_operator` methods all compute a unitary transformation for the operator data between a harmonic oscillator basis representation and a momentum basis representation. The harmonic oscillator basis has a parameter ω which relates to the strength of the oscillator; this value needs to be optimized for the transformation from momentum space to harmonic oscillator space. If no value is given for ω in the function call, it will be optimized internally and returned with the transformed operator. A transformation from harmonic oscillator space to momentum space requires the ω used originally to transform the operator to harmonic oscillator to be passed in by the user. A future implementation may add an internal member to the `operator` class keeping track of ω for operators in harmonic oscillator space to avoid placing the burden of keeping track of the value on the user.

Finally, `embed_operator` embeds an operator in harmonic oscillator space in a higher-

body harmonic oscillator basis provided by the user. The logic for this embedding is similar to the logic for generating the general n-body kinetic energy in harmonic oscillator space. Again, this algorithm takes advantage of the internal reference to the $(n-1)$-body basis kept by the n-body `ho_basis` to avoid recreating a new equivalent basis.

3.4.4 `srg`

The `srg` module only contains the `srg` class, which handles the evolution of some operator via the SRG flow equation. The three methods of this class are its constructor, `evolve`, and `set_generator`. The constructor takes in a potential and a kinetic energy (which are the Hamiltonian when added up), as well as a 2-tuple of matrices called the generator. These two generator matrices are weights assigned to matrix elements of the potential and kinetic energies that are used when computing the flow operator $\hat{G}_s$. For the standard $\hat{G}_s = \hat{T}$, a zero matrix for the generator corresponding to the potential energy and a matrix of ones for the generator corresponding to the kinetic energy is one way to compute the correct flow operator. Allowing the user to provide a function to compute the flow operator from the kinetic and potential energy was a design that was considered, but rejected in favor of the easier-to-check and less complex generator matrices. The `set_generator` method allows the user to alter the flow operator used between evolutions. While it is not typically useful to change flow operators part of the way through a full SRG calculation, this was added to allow us to easily switch between operators during calculations and investigate how composite SRG evolutions affect results.

The `evolve` method evolves the Hamiltonian provided in the constructor to a specified value of the flow parameter λ. The method uses `scipy.integrate.ode` to solve the system of coupled differential equations [22]. In addition to taking the target value of λ as a parameter, it also optionally takes parameters to specify an integrator and parameters for that integrator for users with specific needs who are also familiar with the integrators available through the

`scipy.integrate.ode` method. The default integrator used is `'dopri5'`, a fourth order Runge-Kutta method that works well for non-stiff systems [23]. However, for problems that are more stiff, it makes sense to switch to `'lsoda'`, a solver that dynamically switches between different algorithms for non-stiff and stiff problems [24]. The only downside to the `'lsoda'` integrator is that it is not re-entrant meaning that different `srg` objects cannot both use it at the same time and therefore would be unable to run in parallel. The `srg` class also has accessor functions to allow the user to quickly access the data of the evolved potential as well as the current value of the flow parameter λ.

3.5 Implementation

The first version of the library was written to be compatible only with Python 2.7. We made a decision fairly early on to focus development on Python 3, with an auxiliary goal of still being Python 2 compatible. Over time, the attention paid to keeping the code Python 2 compatible waned. As a result, it is currently not able to be used in Python 2. Fortunately, the recent Python Developers Survey 2017 published by Jetbrains shows that Python 3 is used by 75% of Python developers as their primary version of the language, a substantial shift from 2013, when this project began, and 2015, when the switch to Python 3 on the project was made [25]. Additionally, Python 2 will officially stop being supported in 2020 [18]. All of this leaves the `srg1d` library in a good place considering Python 3 adoption trends and the waning support for Python 2.

3.5.1 Libraries Used

The only non-standard libraries used in `srg1d` are NumPy [26] and SciPy [27]. NumPy arrays are convenient for representing vectors and matrices and can be used interchangeably with lists and lists of lists. Additionally, they allow for efficient numerical calculations and are the standard representation for data throughout the NumPy and SciPy libraries. NumPy also

27

offers matrix and vector operations. SciPy extends the utility of NumPy by offering more linear algebra functions such as eigenvalue decomposition and numerical solvers for systems of coupled differential equations. Both NumPy and SciPy are available in the Python Package Index (PyPI) and can be installed easily with the Python package management system, `pip` [28].

3.5.2 Handling Unexpected Usage

It was mentioned previously that Python relies on library implementers and library users being "consenting adults." As far as libraries are concerned, this means library implementers only have an obligation to document the interface, not actually enforce it. This is different from programming in languages like C++ or Java, where static typing forces users to adhere to an interface. In fact, Python users typically take the stance that if a library user is able to provide an object that doesn't strictly match the library interface but interacts with the library implementation in a way that works, that usage is fine. This is known as "duck typing," following the principle "if it walks like a duck and talks like a duck, then it must be a duck." In keeping in line with this philosophy, the `srg1d` library never enforces the type of objects using `isinstance`. Instead, it typically checks that an object implements the necessary interface, which is done by checking the existence of the required methods and properties.

However, in cases where data is taken in but not used, like in an object constructor call, it makes sense to ensure that errors that will definitely lead to problems later are immediately detected and pointed out to the user. Examples of this include checking that matrices that will be multiplied later have the correct dimensions and checking that corresponding bases and matrices have the same length. The implementation of the `srg1d` library strives to do comprehensive checks in this regard wherever it is reasonable, primarily in object constructors.

One notable exception to this rule is the `val` method from the `ho_state_2body` and `ho_state_nbody` class. This method is ideally never directly called by the user, but profiling done with `cProfile` showed that around 75% of time spent initializing the SRG calculation (calculating operators and transforming or embedding them) was spent inside of it. Removing the checks done on the parameters in `val` halved the time spent inside it. Outer loop optimizations also reduced the number of calls to the method from several million to several hundred thousand. As the library is designed to make it so the user never has to use this method and it is so frequently called, we believe removing error checking from it is worth the performance. In cases where the user does need to directly work with the `ho_state_2body` and `ho_state_nbody` objects, they still have the documentation of the interface to fall back on.

3.6 Testing

An effort was made to adopt a test-driven development (TDD) approach in the development of the `srg1d` library. Tests are written using the `unittest` standard Python library [29], and the documented way to run the tests involves using `nose` [30]. `nose` is an extension to `unittest` that can easily be installed via `pip`. The installation of `nose` comes with a script called `nosetests`, which can be run by the user and will automatically discover and run tests. While the project does not have complete code coverage, the existing tests focus on object constructors and specifically on graceful failure when these constructors are incorrectly called. The `state` module has good test coverage of the `val` method for both classes. One of the goals for the library before publication of the package is to extend the current incomplete test suite to have better coverage.

Chapter 4
Approach to Exploration

In this chapter, we discuss the approach taken to reproducing results from previous work and exploring how using alternative flow operators changes these results.

4.1 Objectives

4.1.1 Verify Many-Body Force Induction

The first major goal of the project was to use the new, more flexible framework provided by the `srg1d` library to reproduce 2-body and 3-body results published in Ref. [2]. While the 2-body results do not give any insight into many-body force induction, they offer an intermediate point for us to look at and be certain of the correctness of our work so far. The 3-body results allow us to look at many-body force induction and reproducing the Jurgenson 3-body results gives us a good point to compare to when testing other flow operators.

4.1.2 Test Alternative Flow Operators

The second major goal after reproducing previous results was to use alternative flow operators and observe how the results of the SRG evolution change. With the framework of the `srg1d` library, changing the flow operator for existing an SRG calculation is as simple as changing two lines of code in most cases. With the ability to easily switch flow operators, we are able to easily take calculations reproducing previous results and use them with new flow
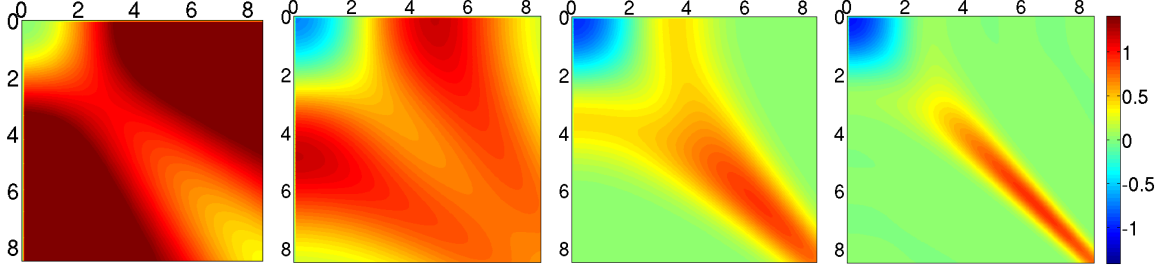
Figure 4.1: Evolution from $\lambda = \infty$ to $\lambda = 2$ of the even part of the $\hat{V}_\alpha$ potential in 2-body momentum space from Ref. [2]. Figure reproduced with permission of the author.

Name	V_1	σ_1	V_2	σ_2	E_2	E_3
$\hat{V}_\alpha$	12.0	0.2	-12.0	0.8	0.920	-2.567
$\hat{V}_\beta$	0.0	0.0	-2.0	0.8	-0.474	-1.708

Table 4.1: Parameters for two variants of the 2-body Negele potential from Eq. 4.1 and their corresponding 2-body and 3-body ground state energies, denoted by E_2 and E_3 respectively.

operators to directly see the change caused by changing the flow operator.

4.2 2-Particle Systems

We use the Negele 2-body potential adopted from Ref. [31], which was also used in Ref. [2]:

$$\hat{V}(p, p') = \frac{V_1}{2\pi\sqrt{2}}e^{-(p-p')^2\sigma_1^2/8} + \frac{V_2}{2\pi\sqrt{2}}e^{-(p-p')^2\sigma_2^2/8}. \tag{4.1}$$

We use the same sets of parameters as used previously, detailed in Table 4.1. The $\hat{V}_\alpha$ potential features a mid-range attraction and a strong short-range repulsion, similar in nature to real nucleon-nucleon potentials. The $\hat{V}_\beta$ potential is a purely attractive potential.

We compute the 2-body Hamiltonian with this potential in momentum space and verify that the ground state energy eigenvalues match. We then run the SRG method on the 2-body momentum-space Hamiltonian and verify that it is truly unitary and behaves in the
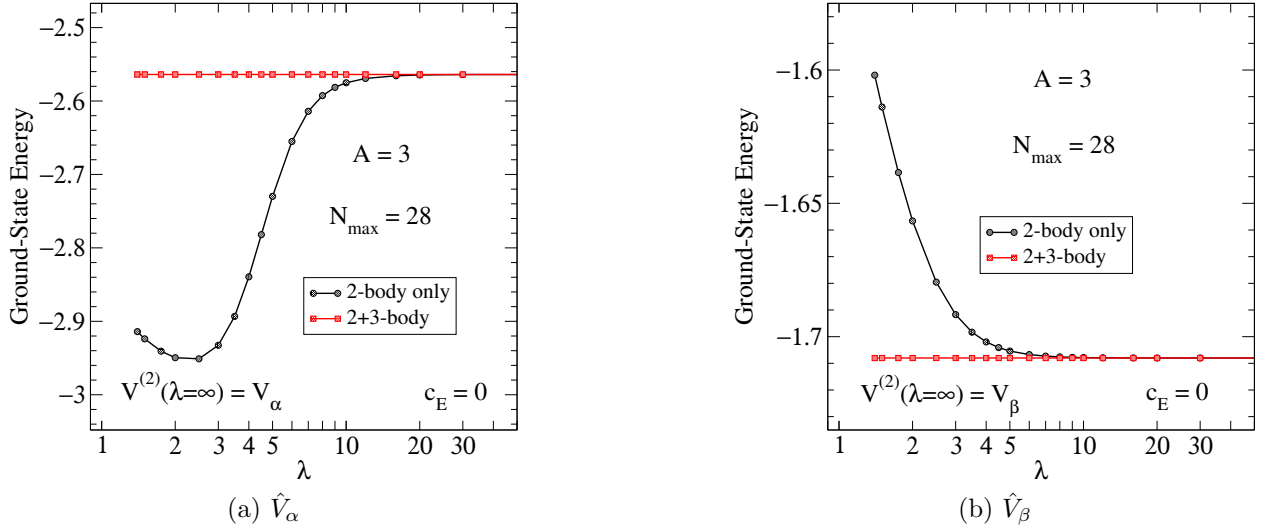
Figure 4.2: Demonstration of an induced 3-body force in (a) the 3-body $\hat{V}_\alpha$ ground state energy and (b) the 3-body $\hat{V}_\beta$ ground state energy from Ref. [2]. Figure reproduced with permission of the author.

same way as in Fig. 4.1. We then perform the transformation to a symmetrized 2-body harmonic oscillator basis, checking that the ground state energy is unchanged by the unitary transformation. We perform an additional check on the correctness of the transformation by ensuring that the transformed kinetic energy operator is equal to the 2-body kinetic energy as calculated directly from the harmonic oscillator basis.

We then use the 2-body Hamiltonian to experiment with different flow operators. While the 2-body system does not give any insight into the many-body forces induced by SRG when using these flow operators, it does give an excellent way to visualize how the action of the SRG transformation on the operator changes because of the alternative flow operators.

4.3 3-Particle Systems

Moving to a 3-body system allows us to get our first results for 3-body force induction for Hamiltonians evolved in a 2-body system. To observe this, we first generate the 3-body

symmetrized harmonic oscillator basis by generating all possible product states, computing and diagonalizing the symmetrizer, and keeping states with eigenvalue 1. With the symmetric 3-body basis, we are able to compute the 3-body kinetic energy and embed the 2-body potential from the 2-body basis in the 3-body basis. With these 3-body operators, we are able to compute the 3-body ground state energy and compare it to the previously published value for it given in Table 4.1.

After properly embedding the 2-body potential in the 3-body basis, we are able to see how the 2-body SRG evolution with the standard kinetic energy flow operator induces a 3-body force in the 3-body ground state energy. We verify that this behavior matches that shown in Fig. 4.2. We then use the same approach to see the strength of 3-body forces induced by 2-body SRG evolution with alternative flow operators.

Chapter 5
RESULTS

In this chapter, we discuss the results found through the approach outlined in Chapter 4. We see that our results match the results published in Ref. [2]. We also note the different behavior of the SRG method when used with alternate flow operators.

5.1 2-Body

For the 2-body momentum-space system, the key results for us to reproduce are the 2-body ground state energies in Table 4.1 and the behavior of the SRG using $\hat{G}_s = \hat{T}$ as shown in Fig. 4.1. After transforming to the symmetric 2-body harmonic oscillator basis, we can validate our implementation by showing that the 2-body ground state energies are unchanged and the transformed kinetic energy matches the kinetic energy computed directly from the harmonic oscillator basis.

5.1.1 Momentum Space SRG

In computing the 2-body Hamiltonian from the Negele potential, we find the 2-body ground state energies to be -0.920 for $\hat{V}_\alpha$ and -0.474 for $\hat{V}_\beta$, consistent with previous results. Furthermore, the SRG evolution with $\hat{G}_s = \hat{T}$ of $\hat{V}_\alpha$ shown in Fig. 5.1 matches Fig. 4.1. The ground state energy eigenvalues are also preserved up to the error in the numerical solver used, confirming that the SRG method for a 2-body operator in a 2-body basis is unitary. In
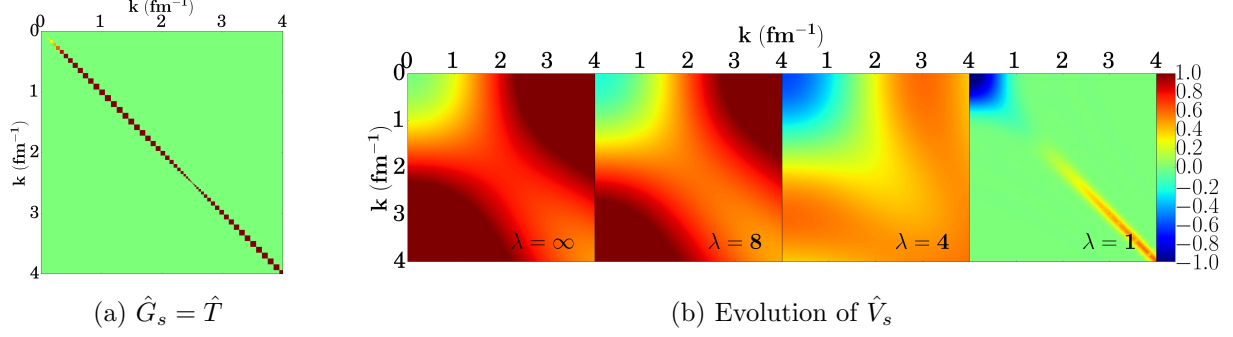
(a) $\hat{G}_s = \hat{T}$

(b) Evolution of $\hat{V}_s$

Figure 5.1: Evolution from $\lambda = \infty$ to $\lambda = 1$ of the even part of the $\hat{V}_\alpha$ potential in 2-body momentum space with $\hat{G}_s = \hat{T}$ as shown in (a).
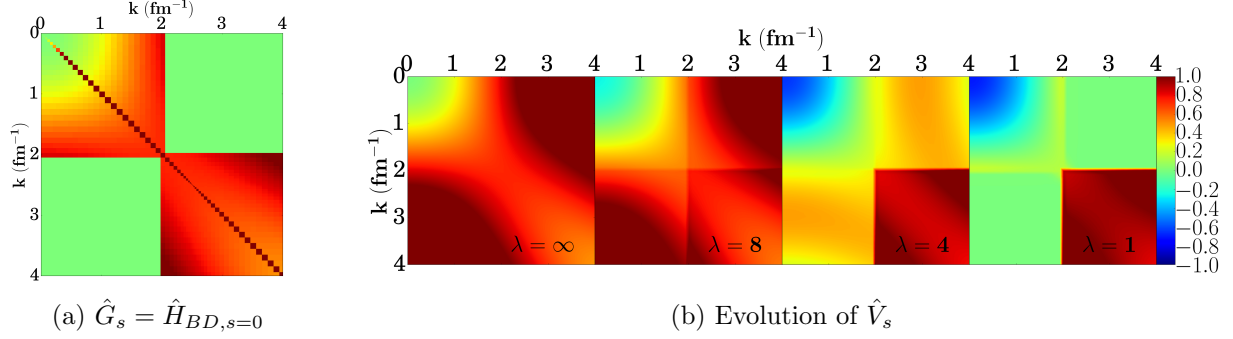

(a) $\hat{G}_s = \hat{H}_{BD,s=0}$

(b) Evolution of $\hat{V}_s$

Figure 5.2: Evolution from $\lambda = \infty$ to $\lambda = 1$ of the even part of the $\hat{V}_\alpha$ potential in 2-body momentum space with $\hat{G}_s = \hat{H}_{BD,s}$ as shown in (a).

Fig. 5.2, we show the SRG evolution using a different flow operator, a block diagonal matrix with two blocks defined as

$$\hat{G}_s = \hat{H}_{BD} = \hat{T} + \hat{V}_s\Theta(p - \Lambda)\Theta(p' - \Lambda) + \hat{V}_s\Theta(\Lambda - p)\Theta(\Lambda - p'), \qquad (5.1)$$

where Λ is the cutoff momentum that defines the two blocks. For our purposes, we used $\Lambda = 2fm^{-1}$. The value of $\hat{H}_{BD}$ at $\lambda = \infty$ is shown in Fig. 5.2a.

The results of the SRG evolution with this flow operator are shown in Fig. 5.2b. This offers a good way to visualize how alternative generators change the form to which the SRG
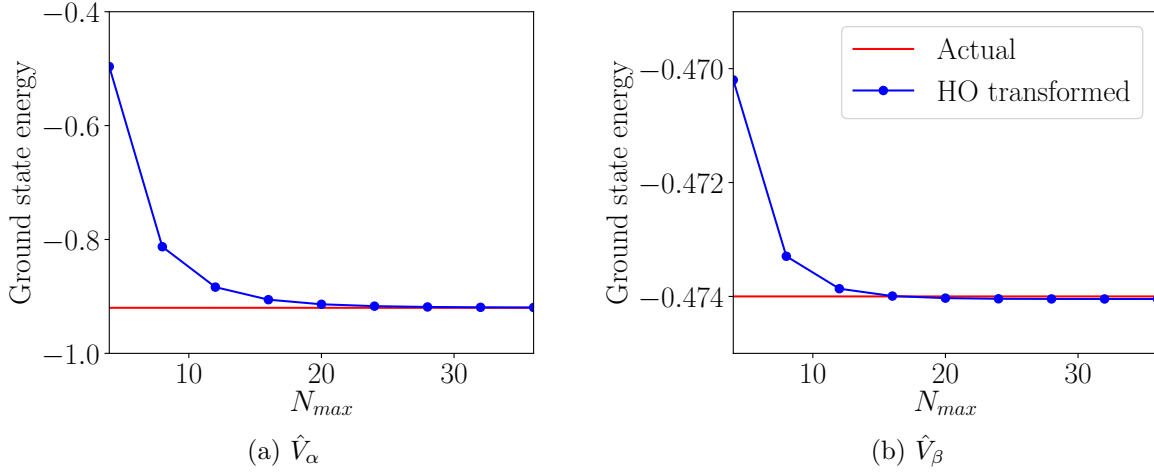
Figure 5.3: 2-body harmonic oscillator basis convergence patterns for (a) $\hat{V}_\alpha$ and (b) $\hat{V}_\beta$ with respect to N_{max}.

method evolves operators. The reason for applying SRG to an operator is to reduce the basis size required to accurately preserve low-energy eigenvalues. What this means is that we want the coupling between the low-energy sector we want to keep and the high-energy sector we want to discard to be zero. Applying SRG with $\hat{G}_s = \hat{T}$ does more than this, driving the entire operator toward the diagonal, even in the low-energy block. It is important to note that if all the low-energy physics is contained inside the low-energy block, we do not care if the matrix is diagonal or not in that block; we have already successfully reduced the size of the problem. Alternative flow operators can be chosen to better reflect these goals, and the intuition is that by avoiding doing unnecessary decoupling, SRG with these flow operators may induce smaller many-body forces. We want to put this intuition to the test when we reach the 3-body case.

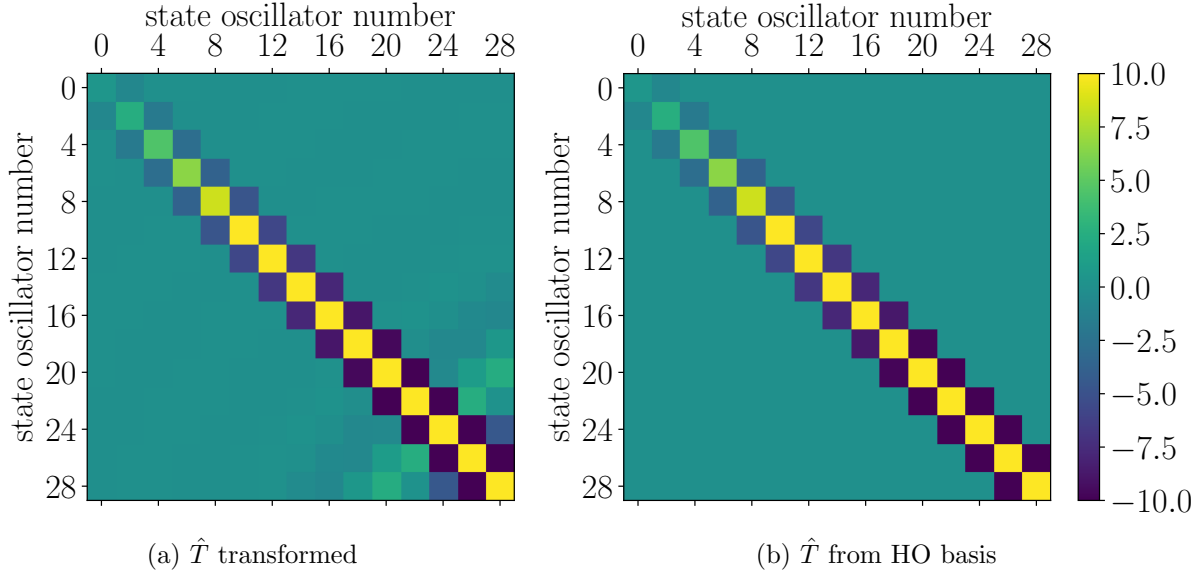(a) $\hat{T}$ transformed   (b) $\hat{T}$ from HO basis

Figure 5.4: The harmonic oscillator kinetic energy operator as computed (a) via a transformation from the momentum space operator and (b) directly from the harmonic oscillator basis.

5.1.2 Harmonic Oscillator Space Transformation

To lay the groundwork for exploring the 3-body system, we transform our 2-body operators into a symmetric harmonic oscillator basis. In practice, we truncate our harmonic oscillator basis at some maximum total harmonic oscillator number, N_{max}. We use $N_{max} = 28$, and we show in Fig. 5.3 that by this value the 2-body ground state eigenvalue is converged to the value calculated in momentum space to within 0.1%.

As an additional check, we compare the kinetic energy calculated in momentum space and transformed to harmonic oscillator space to the kinetic energy operator computed directly from the symmetric harmonic oscillator basis. We find in Fig. 5.4 that they are the same at low harmonic oscillator number, which corresponds to low energies. At high energies, we see the transform kinetic energy operator stray from the true tridiagonal form of the kinetic energy in harmonic oscillator space. This is caused by ringing due to the truncation of the
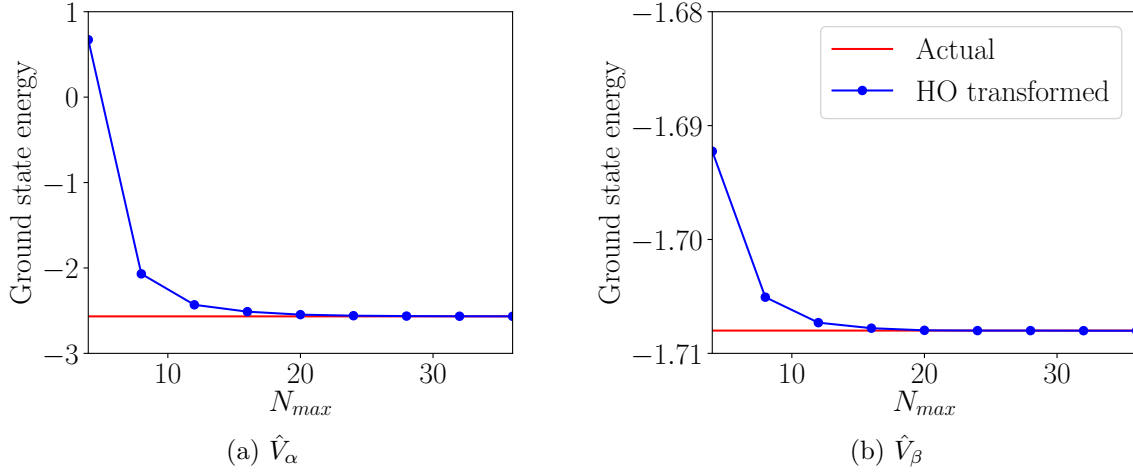
Figure 5.5: 3-body harmonic oscillator basis convergence patterns for (a) $\hat{V}_\alpha$ and (b) $\hat{V}_\beta$ with respect to N_{max}.

harmonic oscillator basis. However, this ringing does not affect the low-energy parts of the kinetic energy and thus has no effect on the lowest eigenvalues, the bound state energies we care about preserving. For the 3-body case we will be working with the transformed potential energy and the kinetic energy computed directly from the harmonic oscillator basis, as it is cleaner to calculate and does not have the ringing artifacts.

5.2 3-Body

For the 3-body system, the key results for us to reproduce are the 3-body ground state energies in Table 4.1 and the induced 3-body force in Fig. 4.2 for $\hat{G}_s = \hat{T}$.

5.2.1 3-Particle Ground State

For the 3-body symmetric harmonic oscillator basis, we stay with $N_{max} = 28$ in order to replicate the same results as Ref. [2]. There is also a performance consideration here, because, while the basis size for a 2-body system scales linearly in N_{max}, the basis size for a 3-body
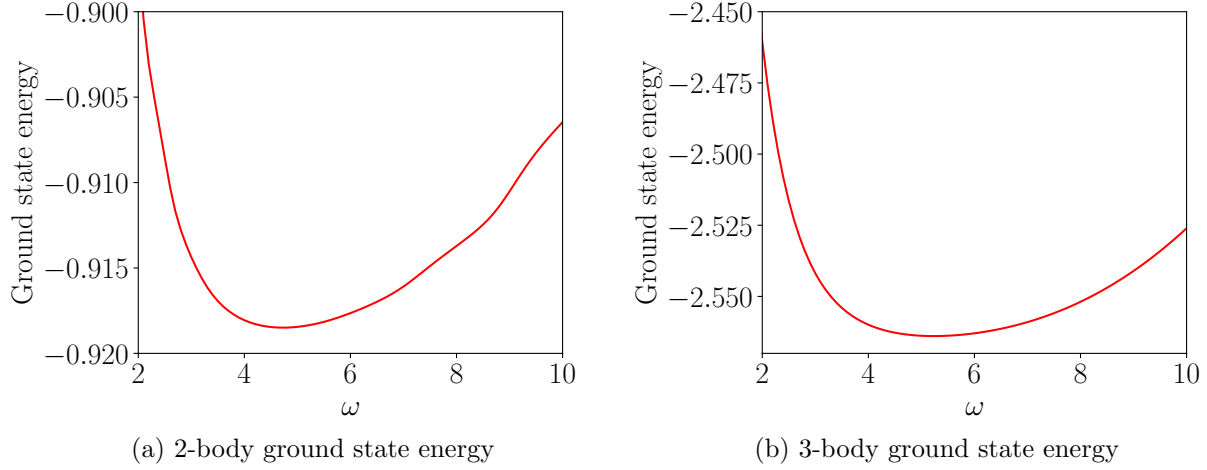
(a) 2-body ground state energy   (b) 3-body ground state energy

Figure 5.6: Ground state energy is a smooth function of ω and thus easy to minimize.

system scales quadratically in N_{max}. Since at $N_{max} = 28$ we are already converged to within 1% in the 3-body system, as shown in Fig. 5.5, by raising N_{max} we would simply be slowing down our ability to test alternative flow generators without any substantial numerical gain. With $N_{max} = 28$, we get the 3-body ground state energy to be -2.563 for $\hat{V}_\alpha$ and -1.708 for $\hat{V}_\beta$, matching the values listed in Table 4.1.

When going from momentum space to harmonic oscillator space, we must optimize ω, the parameter for the harmonic oscillator basis. The optimal ω will minimize the ground state energy, which will be the best we can do to reproduce the low-energy eigenvalues for a given N_{max}. We find that roughly the same value for ω optimizes both the 2-body ground state energy and the 3-body ground state energy, as shown in Fig. 5.6.

5.2.2 3-Body Force Induction for $\hat{G}_s = \hat{T}$

We then compare the 3-body ground state values for a Hamiltonian evolved in 3-body systems and a Hamiltonian evolved in the 2-body system and embedded in the 3-body system at each intermediate point during the evolution. We see in Fig. 5.7 the same trends as in Fig. 4.2.
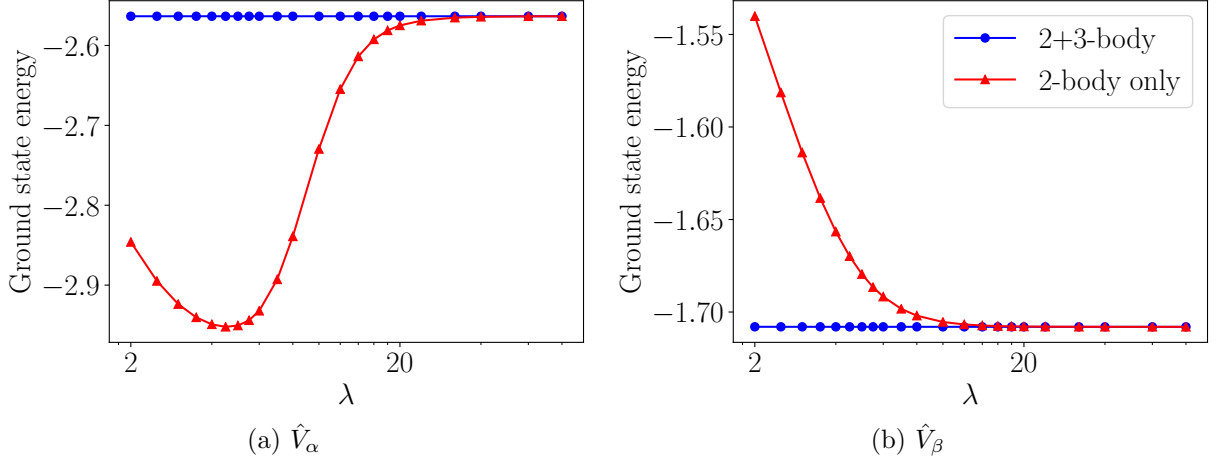
39

(a) $\hat{V}_\alpha$ (b) $\hat{V}_\beta$

Figure 5.7: 3-body force induction with $\hat{G}_s = \hat{T}$ leading to error in the computed 3-body ground state energy.

With this, we have verified the correctness of our framework up to the 3-body system and can now use it to test the 3-body force induction of SRG with alternative flow operators.

5.2.3 3-Body Force Induction for Alternative Flow Operators

We test two parametrizations for an alternative flow operator, $\hat{H}_{BDHO}$, that is block diagonal in harmonic oscillator space, defined as

$$\hat{G}_s = \hat{H}_{BDHO} = \hat{T} + \hat{V}\Theta(N - \Lambda)\Theta(N' - \Lambda), \tag{5.2}$$

where Λ is the block cutoff in harmonic oscillator space. We use $\Lambda = 6$ and $\Lambda = 10$ for our tests. We find that both parametrizations offer a substantial decrease in the induced 3-body force, as shown in Fig. 5.8. In particular, $\Lambda = 10$ induces nearly no 3-body force, keeping the 3-body ground state energy within less than 1% of its true value.

While these results show promise for $\hat{H}_{BDHO}$ as an improved flow operator, they are still preliminary. It may be the case case that $\hat{H}_{BDHO}$ and $\hat{T}$ decouple off-diagonal matrix
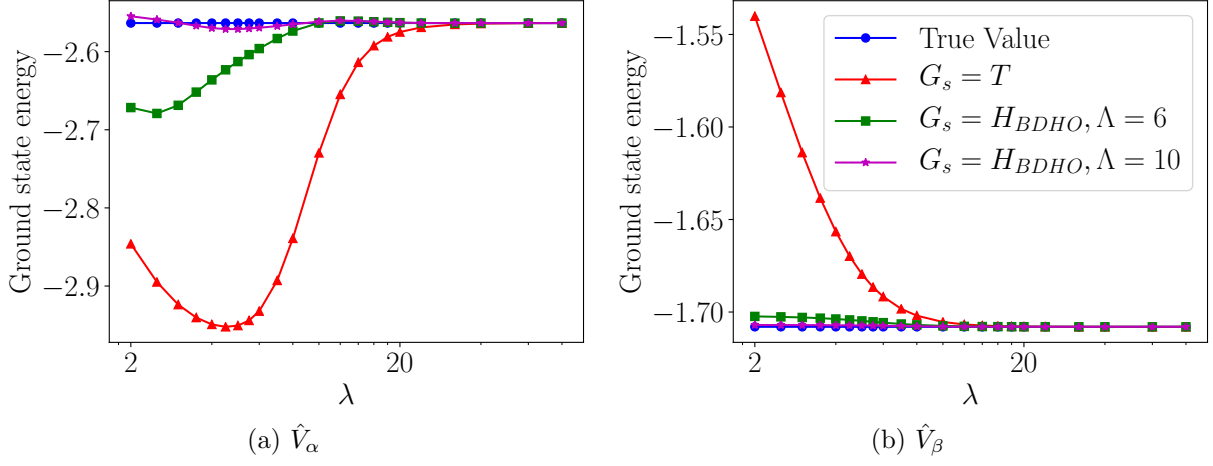
Figure 5.8: 3-body force induction for various different generators. We see $\hat{H}_{BDHO}$ leads to a generally smaller induced 3-body force.

elements at different rates with respect to λ, meaning that $\lambda = 1$ for $\hat{T}$ evolutions does not correspond to $\lambda = 1$ for $\hat{H}_{BDHO}$ evolutions. While some quick tests "by eye" suggest this is not the case, it must still be verified more robustly. This can be done by imposing some cutoff on the 2-body space and checking that the λ at which both evolutions reproduce the correct ground state energy is similar. If it is not similar, it will be necessary to figure out a mapping of corresponding values of λ for the two evolutions.

Additionally, tests must be run that look at 4-body and 5-body forces, in order to ensure induced many-body forces remain strictly smaller than for $\hat{G}_s = \hat{T}$. The `srg1d` framework also allows us to easily test more flow operators, which we must do to get a more comprehensive idea of what features of the flow operators reduce or enhance many-body force induction.

Chapter 6

CONCLUSION

In this chapter, we discuss what is still left in scope for development on the `srg1d` library. We recap the results of our exploration. Finally, we discuss open questions and suggest some points of interest that merit further exploration.

6.1 Future Development on `srg1d`

The biggest feature left to implement for the `srg1d` library is support for the generation of a general A-body basis. This feature comes mostly down to computing the matrix elements in the symmetrizer. While the equation to compute the matrix elements is listed in Ref. [2], it is something that leaves room for many potential bugs. As a result, the implementation must be handled with a lot of care and must be accompanied by extensive validation, most likely in the form of evaluation of the symmetrizer matrix elements for low N_{max} by hand.

There is also some room for additional quality-of-life improvements. First, the coefficients for the symmetrized basis can be memoized by storing them in several files and looking them up on basis calculations. These lookups can be used even when the user requests a larger N_{max} basis than has been stored because the symmetrization for each set of states with the same total harmonic oscillator happens independently. Symmetrization and the calculation of the symmetric basis states is expensive. Profiling of the current state of the code suggests that symmetrizing the 3-body basis accounts for over half of the setup (pre-SRG) time in

calculations.

Additionally, while the documentation for the library so far is fairly comprehensive, the pydocs could be fleshed out with concrete examples. Variable names could be adjusted to offer more clarity to readers. When interfaces are fully finalized, it will also make sense to push the entire library to version 1.0.0 and carefully follow the semantic versioning specification [32].

6.2 Recap of Results

We found that with the `srg1d` framework we are able to reproduce the 2-body and 3-body results from the 2008 Jurgenson paper [2]. We use these results to validate our implementation for 2-body and 3-body systems. As the key algorithms for working in higher-body spaces are recursive, this also implies that our implementation is correct in general. We were able to reproduce the key plots showing 3-body force induction with $\hat{G}_s = \hat{T}$.

We defined an alternative flow operator, $\hat{H}_{BDHO}$. When testing 3-body force induction for SRG with $\hat{H}_{BDHO}$, we found that the 3-body force induced was substantially smaller than for $\hat{T}$. While this result is promising, we cannot speak to how these flow operators perform in general until we are able to test them in 4-body and 5-body systems.

6.3 Open Questions and Outlook

There are a couple open questions directly related to this work:

- How do the 4-body and 5-body forces induced by SRG with $\hat{H}_{BDHO}$ compare to those induced by SRG with $\hat{T}$?

- What features of flow operators reduce or enhance the induction of many-body forces?

- What are the performance or practicality trade-offs that come as a result of smaller many-body force induction?

The `srg1d` framework is (or soon will be) in an excellent position to allow researchers to get answers to all of these questions. This thesis offers preliminary results that show this problem still has a lot of room for exploration. However, these preliminary results must be investigated more comprehensively before general conclusions can be drawn.

Bibliography

[1] H. Hergert, S. K. Bogner, T. D. Morris, A. Schwenk, and K. Tsukiyama. The In-Medium Similarity Renormalization Group: A Novel Ab Initio Method for Nuclei. *Phys. Rept.*, 621:165–222, 2016.

[2] E. D. Jurgenson and R. J. Furnstahl. Similarity Renormalization Group Evolution of Many-Body Forces in a One-Dimensional Model. *Nucl. Phys.*, A818:152–173, 2009.

[3] Ani Aprahamian, Suzanne Lapi, Paul Mantica, Renee Fatemi, Julia Velkovska, John Wilkerson, Bradley Filippone, Vincenzo Cirigliano, Jurgen Schukraft, Xiangdong Ji, et al. Reaching for the horizon: The 2015 long range plan for nuclear science. 2015.

[4] G Bollen. Fribfacility for rare isotope beams. In *AIP Conference Proceedings*, volume 1224, pages 432–441. AIP, 2010.

[5] Benjamin P Abbott, Rich Abbott, TD Abbott, Fausto Acernese, Kendall Ackley, Carl Adams, Thomas Adams, Paolo Addesso, RX Adhikari, VB Adya, et al. Gw170817: observation of gravitational waves from a binary neutron star inspiral. *Physical Review Letters*, 119(16):161101, 2017.

[6] Stefano Dell'Oro, Simone Marcocci, Matteo Viel, and Francesco Vissani. Neutrinoless double beta decay: 2015 review. *Adv. High Energy Phys.*, 2016:2162659, 2016.

[7] Top500 supercomputer sites. https://www.top500.org/. Accessed: 2018-04-03.

[8] Bruce R. Barrett, Petr Navratil, and James P. Vary. Ab initio no core shell model. *Prog. Part. Nucl. Phys.*, 69:131–181, 2013.

[9] R. Machleidt and D. R. Entem. Chiral effective field theory and nuclear forces. *Phys. Rept.*, 503:1–75, 2011.

[10] Evgeny Epelbaum, Hans-Werner Hammer, and Ulf-G. Meissner. Modern Theory of Nuclear Forces. *Rev. Mod. Phys.*, 81:1773–1825, 2009.

[11] S. K. Bogner, R. J. Furnstahl, and A. Schwenk. From low-momentum interactions to nuclear structure. *Prog. Part. Nucl. Phys.*, 65:94–147, 2010.

[12] E. D. Jurgenson, S. K. Bogner, R. J. Furnstahl, and R. J. Perry. Decoupling in the Similarity Renormalization Group for Nucleon-Nucleon Forces. *Phys. Rev.*, C78:014003, 2008.

[13] W. Glöckle. *The quantum mechanical few-body problem*. Texts and monographs in physics. Springer-Verlag, 1983.

[14] R.F. Streater and A.S. Wightman. *PCT, Spin and Statistics, and All that*. Landmarks in Physics. Princeton University Press, 2000.

[15] Nuiok M. Dicaire, Conor Omand, and Petr Navratil. Alternative similarity renormalization group generators in nuclear structure calculations. *Phys. Rev.*, C90(3):034302, 2014.

[16] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

[17] David Robinson. The incredible growth of Python. `https://stackoverflow.blog/2017/09/06/incredible-growth-python/`. Accessed: 2018-04-03.

[18] Benjamin Peterson. Pep 373 – Python 2.7 release schedule. https://www.python.org/dev/peps/pep-0373/. Accessed: 2018-04-03.

[19] 27.4 the Python profilers. https://docs.python.org/3/library/profile.html. Accessed: 2018-04-03.

[20] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[21] Karl Fast. [tutor] what is "pythonic"? https://mail.python.org/pipermail/tutor/2003-October/025932.html. Accessed: 2018-04-03.

[22] scipy.integrate.ode. https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.integrate.ode.html. Accessed: 2018-04-03.

[23] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2008.

[24] Alan C Hindmarsh. Odepack, a systematized collection of ode solvers. *Scientific computing*, pages 55–64, 1983.

[25] Python developers survey 2017 results. https://www.jetbrains.com/research/python-developers-survey-2017/. Accessed: 2018-04-03.

[26] T.E. Oliphant. *A Guide to NumPy*. Trelgol Publishing, 2006.

[27] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. Accessed: 2018-04-03.

[28] PyPI - the Python package index. https://pypi.org/. Accessed: 2018-04-03.

[29] 26.4. unittest - unit testing framework. https://docs.python.org/3/library/unittest.html. Accessed: 2018-04-03.

[30] Note to users - nose 1.3.7 documentation. http://nose.readthedocs.io/en/latest/. Accessed: 2018-04-03.

[31] C. Alexandrou, J. Myczkowski, and John W. Negele. Comparison of mean-field and exact Monte Carlo solutions of a one-dimensional nuclear model. *Phys. Rev.*, C39:1076–1087, 1989.

[32] Semantic versioning 2.0.0. https://semver.org/. Accessed: 2018-04-03.