PES2MP - Programmer's Guide

A python-based user-friendly program for generating NN augmented *ab initio* Potential Energy Surface (PES) and mapping it into analytic Radial Terms $v_\Lambda(R)$

Apoorv Kushwaha, Pooja Chahal, Habit Tatin, and Prof. T.J. Dhilip Kumar

May 19, 2025

बेवज़ह की बेबसी लाइलाज है।

Acknowledgements

The author would like to thank his doctoral guide, Prof. T.J. Dhilip Kumar, under whom this work has been published. The author also acknowledges Ms. Pooja Chahal, who contributed significantly to debugging codes, generating the logo, and developing the manual and interface for the program. The author also acknowledges Habit Tatin, who contributed in implementing TensorFlow for Neural Networks (NN) model generation as part of his master's thesis. Finally, the author wants to thank his family and friends for their continuous help and support.

The author thanks the open-source community for making programs and codes accessible in academics. The contents of this work are the result of continuous efforts made by hundreds of people who have provided open-source content and libraries in the Python programming language. Microsoft PowerPoint and Designer are duly acknowledged for creating graphical images.

The author acknowledges the use of open source libraries in Python programming language such as *Psi4* for PES Generation; Google's *TensorFlow* and *Colab Notebook* for NN model generation, validation and testing; *SciPy* and *Pyshtools* for calculating Legendre and Spherical Harmonics terms respectively; *lmfit* for curve fitting; and other libraries such as *NumPy*, *Pandas*, *tqdm* and *Matplotlib*. The author also duly acknowledges the PES-Learn library which was earlier used for benchmarking and building supervised machine learning models such as Gaussian process, and neural networks using PyTorch and GPy respectively (template available as Jupyter Notebook in PES2MP-Ipynb GitHub repository).

The author finally thanks his parents, Mrs. Meera Kushwaha and Mr. Virendra Kushwaha, for their support in every phase of life. This work would not have existed without their patience and encouragement.

Contents

List of Tables

List of Figures

Abstract

The PES2MP is a Python-based program that can generate radial terms to study the rotational dynamics of any rigid rotor (linear molecule) for its collision with another atom/rigid rotor. It has a graphical user interface (GUI) and consists of several modules (along with thorough comments) to maintain a general layout for advanced users and enable easy code modification for various applications. The PESGen module of the program can automate PES building for 1D/2D/4D collision of atom − atom, rigid rotor − atom, and two rigid rotors using Psi4, Molpro, and Gaussian. While 2D and 4D collisions can be used to study rotational dynamics, the 1D collision is for visualizing and benchmarking the *ab initio* methods and analytical expression (that can be used for fitting the radial data). The PESPlot module creates high-quality plots in the required formats as: 'eps', 'pdf', etc. The *ab initio* PES can then be augmented using the NNGen module that creates an ensemble of NN models to save valuable computational resources and augment the PES without the caveats of a singly trained model. The FnFit module has two components, one for fitting PES and the other for fitting radial terms. The PES component of the FnFit module fits the radial coordinates (for each angular coordinate) using the analytical expression (of choice). It also gives the option to do high-energy and long-range extrapolations using appropriate functions. Fitting the PES into the NN model and/or analytical expression will also provide missing data points that are needed for multipole expansion. This NN-augmented/Function-fitted PES is then expanded in terms of Legendre polynomials and Spherical Harmonics (for 2D and 4D multipole expansion, respectively) to get radial terms (v_Λ) using the MPExp module. The program uses least squares fit (achieved by taking the pseudo-inverse of Legendre/Spherical-Harmonics coefficients stored in a 2D matrix and multiplying with PES) to expand PES into radial terms (replacing angular coordinates) with increasing order of Λ. Finally, the radial terms are expressed as MOLSCAT readable functions (Slater and/or inverse power of R functions) using the second component of the FnFit module, which fits each radial term (v_Λ) individually. The resulting function with its optimized coefficients is printed into a MOLSCAT-readable file. This file can be directly used in the general-purpose version, i.e., MOLSCAT-BASIC's &POTL block to study the rotational dynamics of the rigid rotor. This program requires Anaconda software to provide a clean environment for installing required packages. The makefiles (also with a GUI interface) for installing the PES2MP environment and required libraries are provided with the program.

1 Program Summary

Program Title:	**PES2MP**
Application:	**Cold and Ultracold Chemistry**
GitHub link:	**GitHub.com/QuantumDynamicsLab/PES2MP**
Programming language:	**Python**
Contact:	Prof. T.J. Dhilip Kumar Apoorv Kushwaha
Licensing provisions:	**Apache-2.0** `PSF` : Matplotlib [1], `GPL-3.0` : Psi4 [2], DFT-D3/D4 [3–6], `MIT` : Spyder [7], PyDot [8], Graphviz [9], Tqdm [10], `BSD-3` : Pyshtools [11], Scikit-Learn [12], Pandas [13], `BSD-3` : Lmfit [14], NumPy [15], SciPy [16], Jupyter [17], `Apache-2.0` : TensorFlow [18], Keras-Tuner [19], PyArrow [20]

Keywords:

ML - *Machine Learning*, **TF** - *TensorFlow*, **NN** - *Neural Networks*, **GP** - *Gaussian Process*, **PES** - *Potential Energy Surface*, **2D/4D** - *2/4 Dimensional*, **COM** - *Center of Mass*, **RR** - *Rigid Rotor*, **MP** - *Multipole Expansion*, **BSIE** - *Basis Set Incompleteness Error*, **BSSE** - *Basis Set Superposition Error*, **SCE** - *Size Consistency Error*, **CBS** - *Complete Basis Set Limit*, **GUI** - *Graphical User Interface*, **CP** - *Counterpoise Correction*, **HF** - *Hartree Fock*, **CI** - *Configuration Interaction*, **CCSD(T)** - *Coupled Cluster Singles Doubles and Purturbative Triples*, **MRCC** - *Multi-Reference Coupled Cluster*, **AVQZ/Aug-cc-pVQZ** - *Augmented Correlated Consistent Polarized Valence Quadruple Zeta Basis*, **LTE** - *Local Thermodynamic Equilibrium*, $\mathbf{P}_\lambda(\theta)$ - *Legendre Polynomials*, $\mathbf{Y}_l^m(\theta,\phi)$/**SH** - *Spherical Harmonics*, **RMSE** - *Root Mean Squared Error*, **HPC** - *High Performance Computing Facility*, **WF** - *Wave Function*.

1.1 Scope

PES2MP is a Python-based program that can generate and map potential energy surface (PES) into multipole expansion series (first few radial terms V_Λ: V_λ using Legendre polynomials for 2D PES and $V_{\lambda_1\lambda_2\lambda}$ using coupled Spherical Harmonics for 4D PES) for studying rotational dynamics of linear species in cold and ultracold temperatures. The program is designed to handle 1D, 2D, and 4D PES, which results from an atom − atom, atom − rigid rotor, and rigid rotor − rigid rotor collision, respectively.

The program has various modules for handling different tasks. The summary for each is listed below. For detailed information, see Chapter 2. The basic usage of the program and input files are described in Chapter 3, while the use of GUI for quick calculations using Psi4 and Molpro is explained in Chapter 4. The Appendix A and B contain information that can be beneficial for new users.

1.2 Various Modules of PES2MP

- The **PESGen** module automatically generates PES for various collisions by calculating the center of mass (COM) of rigid rotors and generating required input files in XYZ coordinates. The module can also generate PES internally using Psi4 for rough estimation before proceeding to a high level of theory and a large basis set. For external calculations (on remote servers), input files using Psi4, Molpro, and Gaussian are generated for each coordinate, and scripts for running calculations and collecting results are provided for the same in *"input_files/aux_scripts"* folder.
  Currently, PESGen can generate input files for:

  - 1D collision: atom–atom collision
  - 2D collision: atom–rigid rotor collision
  - 4D collision: rigid rotor–rigid rotor collision

  Ab initio packages supported:

  - Psi4 (Internal): Automatically generates input coordinates, runs the Psi4 API, and gives PES.
  - Psi4 (External): Generates 1D/2D/4D input files for PES calculation using Psi4 application.
  - Molpro (External): Generates 1D/2D/4D input files for PES calculation using Molpro.
  - Gaussian (External): Generates 1D/2D/4D input files for PES calculation using Gaussian.

- The **PESPlot** module generates 1D (Energy *vs* R) and polar (Energy *vs* R, θ) plots for various collisional PES. The program is designed to automatically call PESPlot to generate PES plots after internal Psi4 calculations. For externally generated PES, the module can be manually called to get the plots. The plot (x/y) limits and step-size can be varied to identify anomalies in data points before proceeding with fitting or multipole expansion.
- The **NNGen** module creates Keras-based NN models for PES augmentation. It has been designed to learn properties (Energy, dipole, etc.) resulting from the collision of various molecules. The code can handle multidimensional input (provided as Jacobi coordinates e.g. R, r_i, θ_i, ϕ_i, etc.), and can map them to multiple outputs simultaneously e.g. Electronic (V), Free Energies (ΔG), Dipole moment (μ), etc. The code creates and trains multiple models with varying split ratios of input dataset (Training: Testing: Validation) and then creates a final ensemble model to mitigate errors occurring due to a single trained model. The module has a dedicated neural architecture to train collisional PES data (with constraints that force $E \to \infty$ at $R \to 0$ and $E \to 0$ at $R \to \infty$); however, there is also a generalized architecture that can handle other outputs, e.g. Excited state potential, dipole, etc.
- The **MPExp** module fits PES (2D and 4D) into radial terms by the methods of least squares fitting, taking pseudo- (Moore-Penrose) inverse of the matrix containing Legendre/Spherical Harmonics terms. This is a specialized module only needed by those who want to study the rotational transition of rigid rotor using MOLSCAT or other dynamical packages such as Hibridon (not supported). The module can also inverse fit the radial terms into PES (and print a residual plot) to assess the quality of the fit.
- The **FnFit** module has two parts: (a) PES and (b) V_Λ. The first part, the PES FnFit module, fits collisional PES data (1D, 2D, and 4D) into an analytical expression dependent on R, i.e., $f(R)$ of choice. The program is designed to fit each angular coordinate separately, allowing for simple Slater ($\alpha e^{-\beta R}$) functions to fit the PES. There is a dedicated example for fitting PES into a series of Slater functions with fixed β coefficients. *Why?* The same function can fit radial terms after multipole expansion without any error. Apart from Slater, the other MOLSCAT readable function is R^{-n}. In the second part (V_Λ FnFit module), radial terms (V_Λ) are fitted into the same/unique analytical expression of choice. Once the fitting is done, both parts of the module give a residual plot (to assess the quality of the fit). While the first part prints raw (1D/2D/4D PES) coefficients to a file (that can regenerate PES using an auxiliary code provided with PES2MP), the second part also prints radial terms into a MOLSCAT readable potential file (along with a raw coefficient file).

1.3 Highlights (Capabilities)

General Features

1. **Beginner Friendly:** Only basic knowledge of Python programming is needed for the intended use of PES2MP.
2. **Plots:** The plots and figures can be saved into publication ready *'pdf'* and *'eps'* format.
3. **Connected Pipelines:** The program is divided into modules, and output for one module serves as input for the next. The required input file templates are provided to run each section of the code.
4. **Analytical Fitting and Extrapolation:** Each angular term can be fitted into radial functions to get missing data points. Similarly, extrapolation of long and short-range regions can be done with functions of choice.
5. **Angular Augmentation:** The angular terms can be interpolated by using the NN model, which can augment the surface with spectroscopic accuracy.
6. **Curve fitting into MOLSCAT readable format:** The program can be used to fit radial coefficients into various analytic expressions of choice and prints the coefficients into a format that can directly be used into &POTL section of the MOLSCAT input file.

Machine Learning Features

1. **Ensemble ML models:** NNGen module creates multiple NN models with varying split ratios (of input dataset) and takes ensemble average (using dedicated ensemble NN architecture) to make predictions.
2. **Supports multiple output datasets:** NNGen utilizes the functional API of the Keras library that provides it the ability to learn/predict multiple outputs (sharing the same input) using a single NN model.
3. **Automatic hyper-parameter tuning:** NNGen utilizes the Bayesian interface of the Keras-Tuner library to search for optimum NN architecture of base models. It optimizes the best hyper-parameters (number of branches, number of layers, and number of nodes in each layer) before training a base NN model for a given dataset split.
4. **Custom Activation Function/R constraint:** For collisional PES, a dedicated NN architecture is provided that utilizes Gaussian and negative GELU activation functions for faster convergence in fewer cycles (epochs). The model also puts a constraint on R to ensure correct physical behavior when the same is extrapolated after fitting.
5. **Bayesian-Optimization:** The hyper-parameters are optimized using Bayesian Optimization, which relies on the posterior probability that is updated each time (in the light of new data). Tuning is done using the most popular method called Gaussian Process (GP).
6. **Boundary separation:** NNGen has a dedicated code to separate boundary elements (data points) from non-boundary elements. The non-boundary elements are distributed (split) into training, testing, and validation datasets. The boundary elements are explicitly added to training datasets to make sure the NN model does not miss the real boundary while training.

Optional Machine Learning Features

1. **Splitting Datasets (High Energy region):** NNGen can separate PES into two regions (or more if needed: requires minor modification of code). The first is the minima and asymptotic region, and the other is the high-energy region. Including high-energy points will result in a poor description of the minima region. Therefore, the program is structured to separate the high-energy region that can be (a) the final scattering energy, (b) the first bending frequency, or (c) 2-5 times |energy minimum|, depending on the case. The model also creates an ensemble model for the minima and asymptotic regions as required. The high-energy region is extrapolated as a Slater function to give one continuous output (PES data) during prediction.
2. **Binning for Data Distribution (Training, Testing and Validation in NNGen):** The distribution of data points into training, testing, and validation datasets can be done either by random distribution or by using binning (stratification). The objective can be achieved by specifying the number of bins for either the input or output dataset. Currently, one output is supported (implemented) for stratification, as two outputs can be very dissimilar. The default (and recommended choice) is to use random splitting, which works fine as the data boundaries are already separated and included in the training dataset.

2 Introduction and Theory

2.1 PREFACE

The current program PES2MP can study the collision between two atoms, an atom-rigid rotor, and two rigid rotor molecules. While the first collision (atom-atom) is an additional feature of the program (for benchmarking various analytical expressions), the other two collisions are used to study the changes in the rotational states of the rigid rotor under investigation. Such studies require utmost precision with very small errors in the description of potential energy surface (PES), i.e., spectroscopic precision, which assumes an error of order 1 cm^{-1}. If the PES has a high error (either in the description of well depth or shape), the radial fits can result in a poor description of the quantum phenomenon under inspection. It is essential to understand why these studies are done and what can constitute an error before starting such theoretical calculations. The 'Introduction and Theory' section for this guide covers *'Why are collisional studies done?'*, *'Why is rotational dynamics studied?'*, *'Why computational resource understanding/management is needed for such studies?'*, and *'How does this program make computations fast and efficient?'*.

2.2 Introduction: Molecules in Space

On Earth, the collision of molecules is a rapid, dynamic process. It results in the transfer of energy and matter across our whole planet. In a chemistry lab, these collisions (usually in a solution form) result in chemical reactions that cause rapid structural changes. These changes are the result of vibrational/electronic excitations resulting in bond breaking and formation due to heat, light, or electricity. But physically, it is again a transfer of either energy, matter, or both. Therefore, one can speculate that the collisions in space, which are massively diffused, should also result in the transfer of energy and matter, but at a different timescale [21, 22].

The majority of collisions in space result from Hydrogen (H: $\backsim$ 91% by number) and Helium (He: $\backsim$ 8.9%), followed by other abundant species [23]. These collisions create complex organic/inorganic molecules in space [24, 25], which can be both neutral and charged [26]. But not all collisions result in chemical reactions [27, 28]; some, as we know, merely transfer their kinetic (translational) energy [29]. These collisions can change the rotational, vibrational, and/or translational energy of the collider as well as its colliding partner and are termed non-reactive collisions.

* In astronomy, elements heavier than H and He are called *'metals'*, and their abundance is called *'metallicity'*.

Why bother with collisions that are not bringing any chemical change?

In space, molecules can be found with exotic valencies [30, 31]. This is due to the diffuse nature of clouds, where we usually study them. How these species constitute and affect the environment in dust clouds [32,

33] or around a protostar is usually studied using spectroscopy. This gives us insight into how our own solar system evolved [34] and how life began on Earth [35, 36]. Many of these components of life are even speculated to be in space; however, at present, their detections are still awaited. These detections are generally based on microwave (rotational) spectroscopy, where the signature spectra of the interstellar species are investigated. This spectrum (i.e., the distribution of rotational states) can be affected by the collision of hydrogen and helium [37], making non-reactive collisions important. The study of molecules in space is a vital component of astrochemistry, and the spectra we get from these species give us information about their abundance and environment (e.g., Temperature) around them [38, 39]. A few websites that keep a directory of interstellar species are astrochymist.org, Cologne database for molecular spectroscopy (CDMS) [40], and wikipedia. ∗ *ISM stands for Inter-Stellar Medium, which primarily includes gas clouds along with dust particles.*

This manual is a brief introduction to a vast topic, and the readers must also refer to reviews and books to get a deeper insight into these interstellar collisions. A few suggestions for the topic include *Molecular Collisions in the Interstellar Medium* by D. Flower [41], *Gas-Phase Chemistry in Space: From elementary particles to complex organic molecules* by F. Lique and A. Faure [42], and *Astrochemistry: The Physical Chemistry of the Universe* by A. M. Shaw [43].

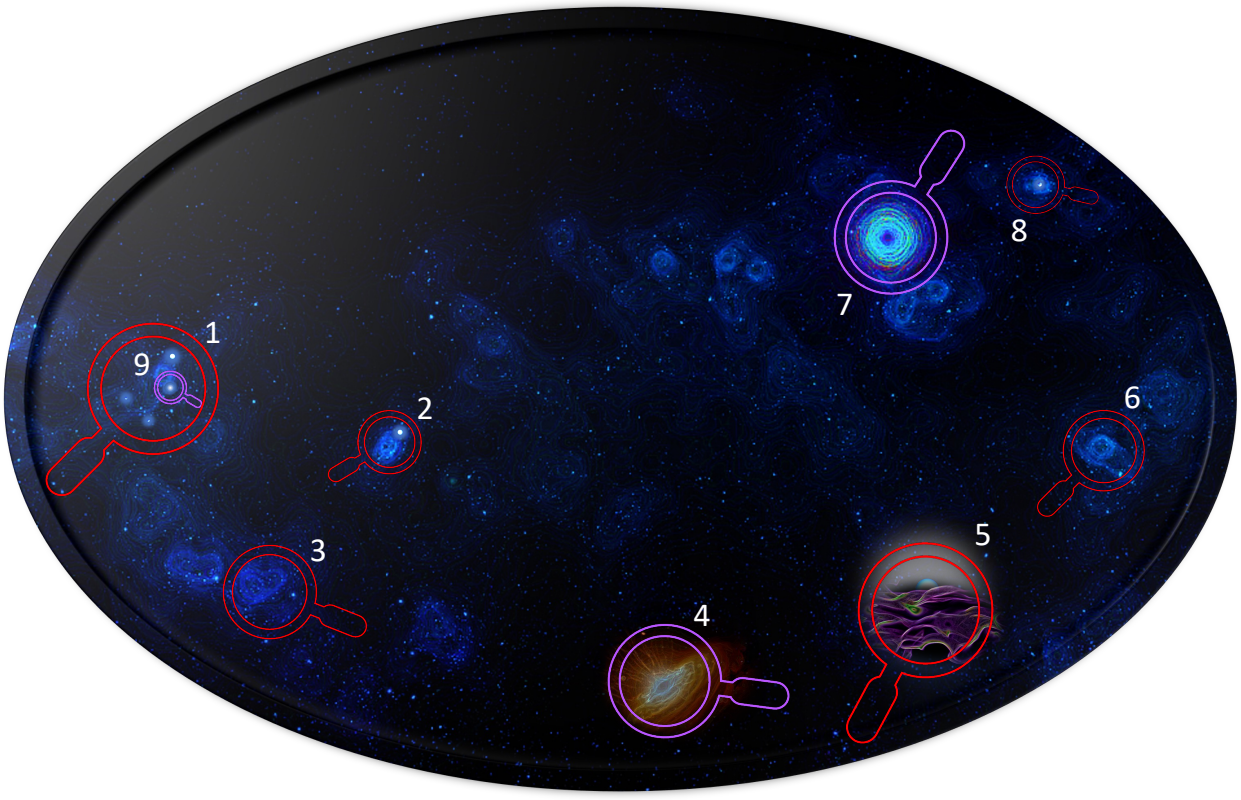


Figure 2.1. Imagine different scenarios that can give us electromagnetic signals from space. Here, an image represents (for illustration purposes only) galaxies/stellar (purple lens: UV-visible range) and gaseous (red lens: IR-microwave range) regions of space. The (small) bright bluish-white spots represent stellar regions **(9)**, and larger disk-shaped objects **(4,7)** represent galaxies. These regions will emit radiation in the UV-visible and higher energy spectra. The interstellar (between stars) medium consists mostly of dust and gases (a) near star-forming regions **(1,2,8)**, (b) cold regions **(3,6)**, and (c) regions lit up by background stars **(5)**. These regions will emit radiation in the IR-microwave range. The spectra of these clouds will vary based on these conditions. † *Image has been created and edited using Microsoft Designer and PowerPoint*

What kind of spectra do we get from space?

The spectra from space come in many different wavelengths, and humans have found ingenious ways of using them to unravel the mysteries of the universe [44–47].

Radio waves mostly carry atomic signatures such as those arising from hyperfine splitting (e.g., the 21 cm Hydrogen line). These signals have the lowest energy and therefore huge wavelengths ($E = hc/\lambda$). Such properties allow radio waves to travel vast distances through space (and reach Earth) with minimal interaction. *High energy radiations?* The higher the energy, the smaller the wavelength, and the greater its chances of being absorbed or scattered by obstacles (atoms/molecules/ions/dust) in its path.

The microwave spectrum is the most important spectroscopic tool for molecular detection [48, 49] as it also has a large wavelength and can penetrate deep into space compared to IR and vis-UV radiation. With microwave spectra, we get information about the rotational states of molecules and consequently, the abundance and temperature of their environment. However, for molecules that do not have permanent dipole moments, one relies on electronic or far-IR signals [50]. Nevertheless, these studies are getting popular, and in the last decade alone, we have detected more than 150 out of the 300+ known species in ISM.

The Infrared (IR) spectra correspond to the energy range of vibrational (and if resolved enough, ro-vibrational) transitions [51]. These contain information about functional groups of molecules. Though they have a longer range (less susceptible to scattering) than visible light, the IR detections outside our solar systems are usually assisted by nearby stellar bodies, illuminating the region.

Finally, **the visible-UV spectra** contain information about electronic transitions in atoms and molecules [52, 53]. Among them, the high-energy transitions (< 300 nm) are absorbed by the ozone layer and, therefore, cannot be observed from the surface of the Earth. The even higher energy photons in the X-ray and gamma-ray regions are associated with star formation and bursts of stellar activity. Such radiations are not discussed in detail here, since our focus mostly revolves around molecular collisions causing rotational transitions.

Fig. 2.1 loosely represents the space, and the magnifying glass represents the kind of spectra expected from its various regions. The purple lens represents a region around a stellar region that will observe spectra in the UV-visible region. The red lens represents dim clouds covering large areas of space that will mostly emit low-energy radiations (microwave-IR region) due to the rotation-vibration of molecules that have permanent dipole moments. The dominant molecule in these cold interstellar clouds is molecular hydrogen, accompanied by mostly neutral species. Such regions in space are called **`H I` regions**. On the contrary, the region next to a newly formed star can have exotic charged species such as H^+, He^+, etc., since they are constantly bombarded with high-energy radiation from the star. These ionized regions of the clouds are called the **`H II` region** and can last a few million years before the thermal and radiation pressure drives most of the gas away [54]. The scales at which such processes happen are hard to imagine. But a simple way to visualize particles in space would be to think of a particle moving in a very viscous medium. Except, in reality, space is nearly a vacuum; however, due to the weak nature of gravity and the vastness of space, large-scale equilibrium in space needs millions of years.

What do we know and what do we want to know about the universe?

Our knowledge of the bigger picture is always a bit stretched (in terms of evidence). And what we know today can change dramatically with time [55]. However, to make life easy for the readers, we shall take a quick recap of the existing theories and follow it up with new and interesting ones. Then, we shall see the established facts about the physical makeup of the observable universe [56] and how we study its contents using telescopes and computers.

The Beginning: The universe (*may have!*) began $\backsim$ 13.8 billion years ago. *How?* We found evidence

(cosmic microwave background, i.e. CMB) that suggests that the universe just blasted out one moment from the size of an atom and stretched to the size we see today [57, 58]. It's called the Big Bang. *How large is it today?* The observable universe is $\backsim 93$ billion light-years in diameter. *Observable?* Yes, beyond that, the space (and its contents) stretches away from us faster than the speed of light. So, we have no means (*yet!*) to speculate what's beyond it. It's called the cosmic horizon. *But is it all true [59]?* Let's say we can take it as a starting point, as anything before the Big Bang was probably a soup of protons, electrons, and just an absurd amount of energy. The Big Bang provides us with reference points in space to see how celestial bodies like stars and galaxies evolved. Also, we have an abundance of lighter elements, which indicates that the universe is still young. *But what existed in the void before the Big Bang, and will it happen again?* Till today, we cannot be sure if anything was there before the Big Bang. Some would argue that it has no meaning as gravity (and time) didn't exist before that point. Maybe this universe will one day start collapsing and do another big bang. Maybe it's a one-time thing and our universe will keep on stretching, or perhaps it happened much earlier, and what we see as CMB is just an echo [60–62]. There are still debates about the beginning and end of the universe. But here we are! It exists, and we exist. The story of our universe and the universe itself might appear to us as an Illusion, or outright absurd (like 'The Twilight Zone'); but the pieces are all there, and all we need to do is: collect them, and make sense of them.

Beyond The Beginning: Now, the story after the Big Bang is not as contentious. We anticipate that there was hydrogen, a little helium, and probably very little lithium. The expansion of the universe cooled it down, and gaseous clouds were dominant in the sky. The universe is evidently still expanding, pushing celestial bodies away from us. There is still matter/energy that is unaccounted for, but I will skip the story of anti-particles, dark energy, dark matter, and neutrinos. It is beyond (a) our topic and (b) my knowledge. So, the cold hydrogen in space began to clump, and once it was massive enough, nuclear fusion started (at the core of stars). The light, heat, and heavier nuclei like carbon [63] came into the picture [64]. Once a star reached the limits of 'sustaining the fusion', it would die either (a) a cold death (usually small stars) or (b) an explosion called a 'supernova' (large stars). It can also become a neutron star (a very dense stellar core that is made entirely of neutrons) or a black hole (a massive star collapses into a small radius, and its gravity is so strong that even light does not escape). Overall, they do what they must as dictated by the physical laws of the universe. The stars would continue this cycle of birth and death and now after (probably) 13.8 billion years, we have the Sun and Earth (estimated to have been formed $\backsim 4.5$ billion years ago) which have Carbon, Silicon, Iron, and even Uranium, all of which are formed from this cycle of birth and death of stars.

The Present: The molecular clouds we observe in space today have a varying abundance of elements, and the chemistry of each will therefore be slightly different. Their metallicity indicates their age, as large concentrations of metals need multiple stellar generations. Studying these regions can also give us information about the various stages of star formation. We can also study the temperature of these clouds to determine the average kinetic energy of the molecules within them. Sometimes, these molecules are in equilibrium with their surroundings, called local thermodynamic equilibrium (LTE), but mostly, they are not (non-LTE conditions) [65–68]. The cold chemistry of these clouds is fairly similar to what we observe on Earth, except that (a) their kinetic energy is much smaller and (b) the density of gas is much lower; therefore, attaining the thermal equilibrium is very difficult.

To summarize, this universe is dynamic, with a constant flow of energy and matter, where everything wants to reach a thermal equilibrium while increasing the entropy to a maximum. The space can be cold (i.e., very diffused molecules with very little kinetic energy), but that does not stop these collisions. It is a blessing in disguise that we can actually study these collisions, but the cost of such computations is high, and studying them can be complex, but not impossible. The universe still has an abundance of hydrogen

and helium, much like it's speculated to have been in the early stages of its formation. But unlike the early universe, we now have the chemistry of dozens of other elements and thousands of new species. The bulk of the physical characteristics (like rotational states and kinetic energy) for these species in space are defined through their collision with hydrogen and helium. For a low-temperature study, we can assume a linear molecule to be a rigid rotor and its collision with H/He/H^+/He^+/He^{2+} is a 2D PES, while collision with a hydrogen molecule (H_2) is a 4D PES. Now, if one can assume spectroscopic accuracy (< 1 cm^{-1}) [69] in the description of the collisional PES of any species, we can describe their scattering with H_2 and He (the two most abundant gases) that affect their rotational states in the ISM.

Now, one may wonder, how the multipole expansion of any PES is related to such studies!

Since the collisions in space happen over large timescales, we can assume that an infinite number of collisions have occurred from all orientations and various energies (of the collider) before we observe the molecule through our instruments. We can, in theory, do a time-dependent study of these collisions (through wave packets). However, it will be more useful to know how the rotational distribution of any molecule will be affected by equilibrium averages of collisions (of H_2 and He), since it is the data we get from microwave spectroscopy. To do so, the multipole expansion of the potential energy surface (PES) is utilized [70], which takes an average over every orientation using an appropriate rapidly converging orthogonal function, representing PES as a sum of radial terms with progressively finer angular features [71, 72]. These radial terms (V_Λ) are used to get the time-independent transition probability for rotational states of the rigid rotor.

So, the PES does describe the collision, but we must transform (fit) it into an orthogonal expansion series that are Legendre polynomials and Bispherical Harmonics for 2D and 4D PES, respectively. Once we can achieve this fitting, a widely used program, MOLSCAT [73], can give us information (cross-sections) about the probability of rotational transitions of any particular species. The mathematical details for the same will be discussed in section 2.6. Since, in a 2D PES, we have distance R and angle of collision (θ), the Legendre polynomials are used to fit it into radial terms. On the other hand, for a 4D collision, we need two Spherical Harmonics terms, one for each angle θ_1 and θ_2. For dihedral, we take $\phi_1 = \phi$ and $\phi_2 = 0$.

Since the multipole expansion of PES is not specific to He or H_2 as a collider, one can use it for any other atom or rigid rotor as a collider. Even though their application in interstellar chemistry is limited to collision with abundant species (H, H_2, and He), they can also be used for another widely popular field of chemistry, ultracold collisions. The cold and ultracold collisions [74] will be discussed in section 2.2.2.

2.2.1 Rotational Dynamics

The microwave spectrum is the go-to parameter when it comes to identifying molecules in space [75]. The unique mass and arrangement of atoms give a molecule its unique moment of inertia and rotational states. These rotational states emit/absorb wavelengths in the microwave region, and with the aid of experimental/theoretical predictions, more and more molecules/ions (and their isotopes) are detected in space. Diving deeper, one finds that these rotational states can split/vanish due to the electronic/nuclear properties of the molecules [76]. Since the focus of this work mainly deals with linear rigid rotor molecules, we shall see what kind of rotational transitions one can expect from these molecules when they collide with other species.

1. **Center of symmetry, singlet (zero electronic spin $S = 0$) and zero nuclear spin ($I = 0$).**
   For example, the C_2 molecule (both ^{12}C) has a center of symmetry. This makes its dipole moment zero, and the next term (quadruple moment) causes the molecule to exhibit only even transitions ($\Delta j = 2, 4, 6, 8$, etc.). However, the story does not end here. The molecule has two ^{12}C with nuclear

spin $I = 0$. This property causes the odd rotational states of the molecule ($j = 1, 3, 5, 7$, etc.) to vanish. The only transitions you can expect from C_2 are $j - j' = 0-2,\ 0-4,\ 2-4$, etc.
Result: $\Delta j = \pm 2n$ transitions to/from even rotational states.

2. **Center of symmetry, singlet (zero electronic spin $S = 0$) and non-zero nuclear spin ($I > 0$).**
The NCCN molecule (^{12}C and ^{14}N) has center of symmetry. Again, this forces the molecule to exhibit only even transitions. However, the two nitrogen atoms with nuclear spin $I = 1$ cause the odd rotational states of the molecule ($j = 1, 3, 5, 7$, etc.) to exist, and even transitions are seen from both odd and even rotational states of NCCN e.g. $j - j' = 0-2,\ 1-3,\ 0-4,\ 1-5,\ 2-4,\ 3-5$, etc. The presence of non-zero nuclear spin will also cause hyperfine splitting of rotational states. However, the corrections are usually small, and such studies become exponentially difficult to study theoretically.
Result: $\Delta j = \pm 2n$ transitions from both even and odd rotational states.
⋆ Expect (small) hyperfine splitting for atoms with $I > 0$.

3. **No center of symmetry, singlet (zero electronic spin $S = 0$).**
These linear molecules e.g. (HCl, CO, etc.) will show all transitions $j - j' = 0-1,\ 0-2,\ 0-3,\ 0-4,\ 1-2,\ 1-3$, etc. Remember, the CO molecule with (^{12}C and ^{16}O) has $I = 0$ and will not show hyperfine splitting but HCl (^{1}H with $I = 1/2$ and ^{35}Cl with $I = 3/2$) will.
Result: $\Delta j = \pm n$ (all possible) transitions.
⋆ Expect (small) hyperfine splitting for species with $I > 0$.

4. **No center of symmetry, non-singlet (non-zero electronic spin $S > 0$).**
In such cases, all transitions are possible, just like in the previous case. And the hyperfine splitting will apply if $I > 0$. However, non-singlet species show another kind of splitting, fine-structure splitting, arising due to the spin angular momentum of the electrons.
Result: $\Delta j = \pm n$ (all possible) transitions with fine structure splitting of rotational states
⋆ Expect further hyperfine splitting of (fine structure split) rotational states for species with $I > 0$.

5. **Special Case: Triplet species with center of symmetry**.
Molecules like $^{16}O_2$ and $^{12}C_4$ have $^{3}\Sigma_g^-$ symmetry. The center of symmetry with negative parity forces the even states to vanish and therefore we see results opposite to that observed for $^{12}C_2$ ($^{1}\Sigma_g^+$), i.e. molecules will still have even transitions (due to center of symmetry) but transitions will occur from odd rotational states e.g. $j - j' = 1-3,\ 1-5,\ 3-5$, etc.
Result: $\Delta j = \pm 2n$ transitions from odd rotational states with fine structure splitting.
⋆ Expect further hyperfine splitting of (fine structure split) rotational states for species with $I > 0$.

6. **Special Case: H_2 molecule**.
This simplest molecule is suspiciously two different molecules hiding in plain sight. The nuclear spin of ^{1}H ($I = 1/2$) can pair up to form composite boson with $I_{tot} = 0$ (para-H_2) and $I_{tot} = 1$ (ortho-H_2). This results in para-H_2 molecule to have only even rotational states ($j = 0, 2, 4, 6, ...$) and ortho-H_2 to have odd rotational states ($j = 1, 3, 5, 7, ...$) [77].

These examples should give an impression of the challenges faced while studying the rotational dynamics of a 1D chain (linear rigid rotor). A species with $S > 0$ and $I > 0$ could easily amass rotational states that can overwhelm even modern-day computing resources. The probability of rotational transitions (both excitation and de-excitation) can be predicted theoretically as a cross-section at any particular collisional energy.

These cross-sections for any rotational transition remain zero till the collisional energy (the kinetic energy of the incoming collider) is smaller than the rotational energy of any particular state ($J = n$). Suppose we are

interested in 5^{th} rotational state of any rigid rotor with rotational constant ($B = 1.5$ cm^{-1}). The energy of that particular state will be $B*J(J+1)$, i.e. $1.5*5*6 = 45$ cm^{-1}. Therefore, till the collisional energy becomes infinitesimally larger than 45 cm^{-1}, the rotational transitions to and from the channel ($J = 5$) remain closed. Each quantum state is referred to as a channel in MOLSCAT and can affect the convergence of cross-sections for nearby open channels. The closed channels are therefore included in calculations for collisional energies even below 45 cm^{-1} to ensure convergence, which further takes up computational resources. The details about the same are discussed in Sec. 2.7.

There is yet another constant for a vibrating (non-rigid rotor) molecule, that correlates the increase in moment of inertia to a decrease in rotational constant, i.e. centrifugal distortion. Unlike the rotational constant (B), the centrifugal distortion constant (D) is related to the changing bond length of a molecule. However, the constant is very small and does not affect the low-lying rotational states in the ground vibrational state of a rigid rotor. This is true only if one deals with collisions at cold and ultra-cold temperatures. Thus, for our work, we can ignore this distortion with caution.

2.2.2 Cold and Ultra-cold Collisions

The collisional dynamics can be studied theoretically in both cold and ultracold temperatures. In cold collisions, the temperature is expected to go as low as 1 mK, while for ultracold collisions, the scale moves to µK. The former is usually employed to study the collisions occurring in interstellar clouds, while the latter is a leap into the future where we hope to prepare molecules with specific energy and quantum states to enable controlled chemistry [78–81].

∗ *Kinetic Energy vs Temperature:* The collisional energy of an atom is the relative kinetic energy of that atom (collider) relative to another atom/molecule that it is colliding into. It depends on their mass (very small) and velocity and is therefore measured in cm^{-1} (which is $\backsim 1/350$ kcal/mol or $\backsim 1/8000$ eV). On the other hand, Temperature is a bulk phenomenon observed due to the collective property (kinetic energy) of matter. Therefore, at the atomistic scale, the temperature loses its meaning as opposed to the kinetic energy.

Cold Temperatures and Interstellar Medium

As discussed, the intensity of rotational spectra gives us the relative abundance of any species (w.r.t. H_2), and its rotational states can give us an idea about the temperature of the environment. However, the results can be misleading in cases where thermal equilibrium is not achieved, which is largely the scenario due to the diffuse nature of space. These non-LTE conditions (mostly the case in ISM) are where collisional dynamics are the most important. But we do not need to study each and every collision. The majority of collisions occur with H_2 and He. The He collision results in a 2D PES and H_2 in a 4D PES. Next, we must decide the number of rotational states to be included in the dynamical study.

The spacing between rotational states decreases rapidly as a species becomes massive. The more the mass, the smaller the rotational constant and therefore, the more the number of rotational states. The general formula for calculating the energy of rotational states is given by:

$$E_r = B \times J(J+1) - D \times J^2(J+1)^2 \tag{2.1}$$

where B is $h^2/(8\pi^2 I)$ and $I = \mu r^2$ i.e. moment of inertia. The second term describes centrifugal distortion and is neglected for the rigid rotor approximation. The centrifugal distortion constant is $D = 4B^3/\omega^2$, where ω is the vibration wavenumber.

A good starting point for choosing rotational states for any molecule and the kinetic energy of the collider would be to look at the regions of space in which it is detected (confirmed or tentative). These regions between stars are massive clouds of dust [82] and gas, hence, the Interstellar Medium (ISM) can be hot or cold depending upon the dynamics of the region. A cold region has dense clouds due to the low kinetic energy of gases, while a hot region becomes more diffuse. In such regions, molecules are mostly neutral. Molecules near stars can be charged due to stellar flares and can reach very high kinetic energies.

In cold neutral regions, H_2 is the most important collider, followed by Helium. Around stellar bodies, exotic species such as H^+, $He^{+/2+}$, etc. are also important colliders [83, 84]. Since the kinetic energies of molecules can be very large and molecules can be highly excited, such collisions require a ro-vibrational (non-rigid rotor) description of the PES. Such collisions are studied using time-dependent wave-packet dynamics for charge transfer and excited-state (electronic) dynamics. While a time-independent study for a vibrating rotor is also possible, the same is usually restricted to lighter molecules [85–87].

In gaseous regions of ISM, temperatures usually reside in the range of 1-100 K. Therefore, one can expect molecules to have small kinetic energy and low rotational quantum numbers, as previously discussed. The collisions can result in excitation as well as de-excitation of the species. These collisional cross-sections are then averaged over the Maxwell-Boltzmann distribution to get rate coefficients at various temperatures. With this data, one can model the environment around any species under non-LTE conditions.

A good starting point to have a deeper understanding of such methods would be to look at the cited review by Roueff and Lique [88]. The review covers collisional, radiative, and chemical excitations in the interstellar medium with examples of various species and the regions of space they are detected.

Ultra-cold Temperatures and Controlled Chemistry

In recent years, physicists have been able to recreate temperatures at these cold temperatures and even beyond, i.e., to the ultracold regimes [89, 90]. The intuitive understanding of temperature changes when we go into the quantum realm. In the macro-scale, the temperature is what we observe through the flow of heat. In the quantum world, this flow of heat is nothing but collisions transferring kinetic energy. Therefore, when we talk about ultracold temperatures, we talk about the average kinetic (i.e. translational) energy of a molecule. Using ultracold cooling, a gaseous molecule is restricted to near-zero kinetic energy while it remains in a particular rotational state.

Such new techniques give enormous control over the chemistry of molecules and enable us to look into future methods of chemical synthesis. Currently, such experiments are limited, but the ability to theoretically predict experimental results does give new insight into chemistry that is yet to be explored.

When the kinetic energy of colliders corresponds to the rotational gap of molecules, de-excitation transitions have a very large probability. This is explained by Wigner's threshold law [91] which states that relaxation cross-sections tend to infinity in the limit of zero kinetic energy of the incoming atom.

While the current breakthrough in experimental ultracold chemistry is currently limited to lighter molecules [92], theoretically, such collisions can be modeled similarly to cold collisions. While they have limited applicability today, future experiments will be aided by these theoretical studies. The current program can generate radial coefficients that can describe the collision between any required pair of atom/rigid rotors.

To know more, take a look at the comprehensive review of ultracold chemistry by Balakrishnan [78]. The review discusses various experimental breakthroughs in cooling gas to ultracold temperatures.

2.3 Computational Resources and Quantum Chemistry

One of the biggest hurdles in computational chemistry (apart from understanding the irritatingly large number of parameters for PES/dynamical calculations) is the judicious use of computational resources. It is useful to understand the way computers are built and how parallelized the codes are, to run any program more efficiently. For e.g., *what is the difference between using a personal computer (PC) vs a supercomputer for performing any calculation? Can it speed up your calculation? If yes, by how much?*

Understanding the limits of any standalone computer (PC/Workstation) or a cluster of them, i.e., high-performance computing (HPC) facility, can be beneficial before a job can be submitted on the same. In the current era, standalone PCs have also become powerful enough to perform chemical computations. Follow this tutorial and get a grasp of how a new computer can be built from scratch, or what its essential elements are. This is a piece of information any theoretician must know before starting any computation when chemical problems are involved.

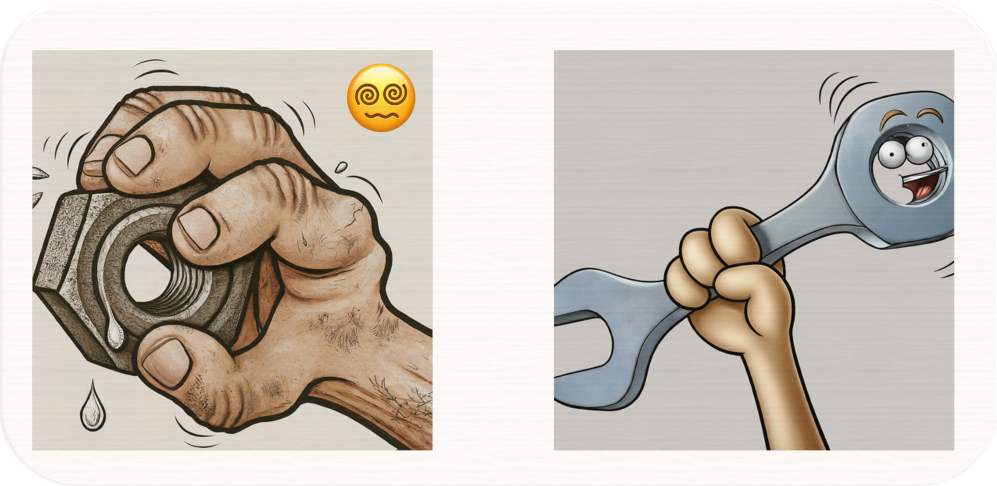


Figure 2.2. Brute-Force *vs* Smart-Force. † *Image has been created and edited using Microsoft Designer and PowerPoint*

2.3.1 Understanding Computer Hardware

Here is a list of a few hardware components essential for building a computer. The examples are relevant for the year 2025 but will change over the years. Nevertheless, the basics of a PC/cluster are the most important aspect of a computational chemist and one must update its components with time to make sure it runs with maximum efficiency.

1. **Motherboard:** The main board on which CPU, GPU, RAM, and storage (HDD/SSD) are installed. In an HPC (high-performance computing) facility/cluster, multiple boards are attached, and each board is called a node. The main node managing the jobs is called the master node, and it sends the jobs to the worker nodes. It is usually a centralized computing facility that is managed by professionals.

   A poorly designed motherboard will be slow even with high-end components. Always go for standard motherboards that can support future upgrades. It must have a dedicated NVMe port for fast SSD, multiple SATA connectors for HDD, support for 4 or more lanes of memory, and must be compatible with new high-end processors. This is the single most important component of a PC that cannot be

upgraded, and the CPU and memory are tied to the compatibility of a motherboard. In practice, look for a motherboard that is compatible with the CPU of your choice and then check if it supports the current generation of RAM.

2. **Processor (CPUs)**: Modern central processing units (CPUs) have multiple cores for running logical calculations. The processor/core count of a CPU tells you how many parallel jobs one can run on a single machine. Computer codes can also take advantage of running a single job on multiple cores. However, remember that there is a limit to such parallelization depending on the *ab initio* method chosen (discussed in the next section). In modern machines, each core can act as two individual cores, which doubles the number of jobs one can run. These broken-down cores are called threads. When a motherboard has more than one socket for the CPU, the cores/threads for each are said to be in different nodes (similar to HPC but without any master node).

   For quantum chemistry calculations on a single machine, buy high-end processors (> 3.0 GHz with 16 or more cores). Always refer to the power (base and max.) rating of the CPUs, as a 125W CPU will always outperform the same CPU under-clocked to 65W.

3. **RAM (Random-access memory):** RAM stores variables for jobs running in CPU. If a job is running in parallel across multiple threads, it will take up more and more memory; therefore, if you allocate more processor/core/threads for a job, make sure to increase its memory allocation. RAM stores data only temporarily till the jobs are running, but it is much faster than conventional storage (HDD/SSD).

   The current generation of RAM is DDR5 (fast and expensive with speeds $\backsim 4$ GHz), but the older generation DDR4 RAM (3200 MHz) is cost-effective and also has decent performance. RAM should be installed in the range of 32GB to 1TB, depending on use. A 32*4=128GB RAM is usually good enough for most quantum chemistry jobs.

4. **STORAGE (HDD/SSD):** The basic devices for storing data (read and write) are either hard disk (HDD) or solid-state (SSD) drives. They store large amounts of data that are permanently stored (data is not lost if the computer is switched off) and are slower than RAM. If your calculations require storage of large integrals that exceed the capacity of RAM, they can be stored in storage devices, however, this will slow down the calculations considerably.

   As a general rule, use a small 256/500 GB SSD for installing the operating system (OS) and software, and add multiple 1TB/2TB/4TB HDD/SSD for temporary files depending on the budget.

   SSDs are great for accelerating operating systems (OS) and jobs requiring temporary files in the range of 100 GB to > 1 TB (terabytes). Older SSDs are rated at 500MB/s (they use SATA connection) while modern Gen4 SSDs (use NVMe connector) are rated as fast as 5GB/s (i.e. 5000MB/s). Compare that to 50 MB/s for a 5600 rpm HDD. However, HDDs are still cost-effective, and if required, one must opt for 7200 rpm HDDs, which can give speeds up to 150 MB/s and lasts longer read-write cycles.

Auxiliary Information:

- **MBps vs Mbps** Use of M**b**ps instead of M**B**ps is a marketing term created to mislead consumers. The small **'b' is bits**, and the capital **'B' is bytes**. E.g., 100 Mbps speed is actually 100 mega**bits** per second, i.e., 100/8 = 12.5 MBps (mega**bytes** per second).

  The standard storage unit has always been MB/GB and not Mb/Gb, so if you ever see a small 'b' in speed (e.g. 9 Gbps), just divide the number by 8, and that is your actual speed.

- **Compatibility:** The CPU, RAM, and motherboards always have some restrictions on the generation of hardware they can support. Even modern-day SSDs come with such restrictions, but they are usually backward-compatible (however, their speed will be reduced if the connector is of a previous generation). So, always check if the parts are inter-compatible (there will always be mistakes, and it's fine; just make sure it's returnable).

- **GPUs:** Graphics processing units (GPU) can accelerate some specialized calculations, but the current application of the same is limited. See the Gaussian 16 manual for GPU acceleration. GPUs are also essential if you want to run ML calculations on them. In PCs, they are primarily needed to output a display by attaching them to a monitor (using HDMI/DVI/VGA/etc. connectors).

  GPUs are expensive, so if not required, one can opt for a very basic GPU to run a display. Also, many modern PC processors (such as Intel i3 14100) have inbuilt (integrated) GPUs, so an external (dedicated) GPU is not needed to run the display (*Warning: Similarly named i3 14100F processor does not have an integrated GPU*). If you just want to attach a display to the previous CPU or other server-grade CPUs such as Intel Xeon and AMD EPYC, use the cheapest branded GPU on the market.

- **Power supply:** This is one of the most important and overlooked components of a computer. Always make sure that the power supply is of good quality and rated higher (in Watts) than the CPU and other components combined. Always prefer a branded modular (cables can be detached and attached as needed) power supply with decent power efficiency certifications. The current standard for a PC with a decent power supply is $\backsim 250-500$ W without a dedicated GPU and $\backsim 750-1000$ W with a dedicated GPU.

- **Thermal paste:** The CPUs are always attached to a metal heat sink (with a dedicated fan) since their performance is susceptible to thermal throttling. The contact between the CPU and heat sink is filled with highly conducting thermal paste, which can dry in a few years. If the CPU often reaches 100 °C (even on idle/moderate load), it's time to reapply the thermal paste.

Alternative to Hardware Details

Buy a MAC! Hardware-wise, they are well-configured and efficient. Software-wise, MacOS is based on Linux, is stable, and has most of the capabilities of a Linux distribution. It has a great UI (user interface) that most non-coding communities can work with. Downside: $$$. Saying desktop-grade MACs are expensive is an understatement, but they do outperform most desktop builds in terms of efficiency and life expectancy.

2.3.2 Understanding Computer Software

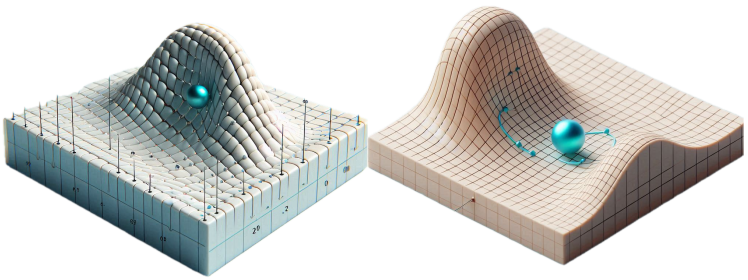
- **OS:** Operating System! For a computational scientist, go with Linux. They are not perfect and can make you spend 10 days trying to debug the code. A code that broke without any reason, to seemingly start working another day without giving any clues as to what actually went wrong. It requires you to spend hours building libraries or installing software that installs with 2 clicks on a paid OS. But Linux makes you a complete programmer. It makes you familiar with how a computer works. The pain needed to debug a code/script will eventually go away, leaving lifelong experience to your aid. And that is what you want! Also, LINUX is currently the 'undisputed king' for servers. On the other hand, the paid OS will protect you from breaking the system, will come with built-in features and precompiled libraries, and will cost you a lot of money. The journey, to say, is not the same.

- **Coding:** Python and Fortran. Python is easy! It has thousands of open-source libraries, is constantly being updated, and most importantly, is beginner-friendly. Python is a high-level language (slow and memory demanding), and therefore, Python programs generally use low-level C/Fortran libraries (at the core) to make them faster. F77, on the other hand, is painfully complex for beginners. But like Linux, it will teach you the workings of a code. It requires you to declare variables, and memory pointers, and follow a strict pattern (column spaces are reserved with special meaning) and length (72 characters) for coding. But in reality, F77 is so outdated, that there is a chance you *may* need to install a 32-bit OS in order to find/run a compiler. The new F90 and Fortran 2023 are worth looking at if you want to. Personally, it will give you a lot of appreciation for Python and help you write efficient codes. In the list of paid programming languages, MATLAB is the go-to choice for academia.

2.3.3 Understanding *ab initio* Calculations

Coming to computations, we do *ab initio* calculations to get the potential map of what an incoming atom/molecule should face during a collision, as shown in Fig. 2.3 (b) and (c). This picture stems from the fact that molecular dynamics for any colliding system is dictated by these electronic states. The change in rotational and vibrational states for these collisions can therefore be modeled by (a) wave packet study and (b) close-coupling calculations. The wave packet studies are time-dependent and can provide a chemical picture of the time evolution on the femtosecond scale [93, 94]. On the other hand, close-coupling calculations are based on the time-independent Schrödinger equation [95], imposing boundary conditions that assume the scattering process has occurred over asymptotically long time scales, resulting in well-defined initial and final states. The collisions occurring in molecular clouds of the interstellar medium follow such time scales, and therefore, we shall focus on this time-independent view, which is beneficial for modeling the distribution of rotational populations of interstellar molecules.

(a) Collision between two atoms



(b) Potential Example 1 (c) Potential Example 2

Figure 2.3. Images illustrating (a) particle picture for collision between two atoms, while (b) and (c) represent particle moving in potential. For non-reactive collisions, the potential energy surface dictates the motion of the incoming collider and therefore describes probabilities of ro-vibrational transitions and bound states. † *Images have been created and edited using Microsoft Designer and PowerPoint.*

Before we proceed to the core of *ab initio* calculations, let's get introduced to two important concepts: method and basis set. Some methods require a basis set, and some don't. Some basis sets are available for heavy atoms, while some require you to get pseudo-potential basis sets from websites like Basis Set Exchange. But before diving into the infinite space of computational misery, let us begin *ab initio* (from the beginning).

Whenever in despair, redefine the problem with just the basic conditions. Therefore, we will begin with the energy equation [96]. In the quantum world, the energy operator is called the Hamiltonian, so let's take a look at the same.

Molecular Hamiltonian

Suppose we have set up a Hamiltonian for ground state energy, our equation would look like this:

$$\hat{H}\Psi(x) = E_{GS}\Psi(x) \tag{2.2}$$

where $\hat{H}$ is the Hamiltonian describing energy operator and Ψ is the wavefunction containing information about our molecular system. The eigenvalue E_{GS} is our desired result [97–99].

If we break this Hamiltonian into its constituent terms (considering a single nucleus and single electron), we get:

$$\hat{H} = -\frac{\hbar}{2m_{nu}}(\nabla^2_{R_{nu}}) - \frac{\hbar}{2m_{el}}(\nabla^2_{R_{el}}) - \frac{1}{4\pi\epsilon_0}\frac{Ze^2}{|R_{el} - R_{nu}|} \tag{2.3}$$

The equation seems fairly simple, but it grows exponentially with system size. Moreover, the electron-electron interaction makes the problem even worse due to our inability to accurately describe what happens when two electrons come close together. The generalized equation for our simple Hamiltonian looks like:

$$\hat{H} = \hat{K}_N + \hat{K}_e + \hat{V}_{NN} + \hat{V}_{eN} + \hat{V}_{ec} \tag{2.4}$$

Each component (operator) of the equation is a summation function (Σ) describing :

- (a) kinetic energy of
  - nucleus : $\hat{K}_N = -\sum_i^{\text{nuclei}} \frac{\hbar^2}{2M_i}\nabla^2_{\mathbf{R}_i}$,
  - electrons : $\hat{K}_e = -\sum_i^{\text{electrons}} \frac{\hbar^2}{2\ m_e}\nabla^2_{\mathbf{r}_i}$,
- (b) potential energy of
  - nucleus-nucleus repulsion : $\hat{V}_{NN} = \sum_i \sum_{j>i} \frac{Z_i Z_j e^2}{4\pi\varepsilon_0 |\mathbf{R}_i - \mathbf{R}_j|}$,
  - nucleus-electron attraction : $\hat{V}_{eN} = -\sum_i \sum_j \frac{Z_e e^2}{4\pi\varepsilon_0 |\mathbf{R}_i - \mathbf{r}_j|}$,
  - electron-electron correlation $\hat{V}_{ec} = \sum_i \sum_{i<j} \frac{e^2}{4\pi\varepsilon_0 |\mathbf{r}_i - \mathbf{r}_j|}$

Nevertheless, we can still make a Born–Oppenheimer (BO) approximation to simplify this equation. Remember that electrons are lighter (faster), and the nucleus is heavier (slower). This gives us an approximation like the Franck–Condon principle, where we consider electronic transitions to be vertical (the distance r does not change) due to the slower vibration of molecules. Similarly, BO approximation can be applied here, exploiting the fact that electrons move much faster, and their Hamiltonian in the above equation can be separated from the nuclear motion.

What does BO approximation do? It makes the molecular Schrödinger equation simpler and easier (faster) to solve on computers. Remember the wavefunction Ψ, it can now be separated into Ψ_{nu} and Ψ_{el}.

Thus, we can separately solve the nuclear Hamiltonian and the electronic Hamiltonian, which were earlier coupled. The expectation value for the same can be obtained via:

$$\langle E \rangle = \frac{\int\int..\int \Psi_{el}^{*}(R_1, R_2..R_N)\hat{H}_{el}\Psi_{el}(R_1, R_2..R_N)dR_1, dR_2..dR_N}{\int\int..\int \Psi_{el}^{*}(R_1, R_2..R_N)\Psi_{el}(R_1, R_2..R_N)dR_1, dR_2..dR_N} \tag{2.5}$$

This approximation does fail when the curve crossing occurs, since the nuclear motion gets coupled with the electronic motion, leading to numerical errors. Then, we have no choice but to solve the coupled equation (but only in the region where the BO fails) to study a very interesting field called non-adiabatic (non-single state) chemistry.

And how do we approach to solve this ???? The simplest method to solve the molecular Schrödinger equation *ab initio* is Hartree-Fock (HF) [100–102]. It uses mean field theory or self-consistent field (SCF) theory (therefore HF can also be referred to as SCF), where we consider that each electron moves in the collective field of (all) other electrons. It is the first step towards any quantum mechanical solution, and therefore, its results serve as the reference for further approximate (with lower error) methods called post-HF methods.

** Read about first and second quantization for deeper insights.*

So what are methods and what are basis sets?

For a chemist, in an elementary language, the basis sets are the set of functions that represent the wavefunction containing information about the system. On the other hand, a method represents the computational theory that describes the approach used to approximate the wavefunction for solving our multi-electron system. Since there is no current way to know this wavefunction exactly, we build the basis set carefully as a linear combination of atomic orbitals (inspired by the solutions of the simplest two-body system, H atom). The results of the H atom are Slater functions (radial part), and they are computationally expensive (no analytic solution) when used as basis sets. Therefore, we currently use multiple Gaussian functions to mimic any Slater function. So, most of the basis sets for quantum chemistry consist of Gaussian-type orbitals (GTOs) [103, 104]. The plane wave/numerical basis used in solid-state chemistry has its own advantages in the reciprocal (momentum) space [105]. For quantum chemistry of species in the gas/solution phase, we keep the plane 'real' and the basis 'normal'.

Basis Sets

Ideally, infinite basis functions [106] give exact results for any specific theory (basis-set limit). In practice, we'll always have a limited number of basis functions and we shall use them smartly.

Minimal Basis Set STO-*N*G : The minimal basis mimics each Slater function [107] with N Gaussian functions. The problem is that Gaussian functions decay faster than Slater functions, causing incorrect descriptions at long ranges. It has no extra function to account for the polarization of valence orbitals or the long-range correlation. This basis set is rarely used for dynamical calculations but is useful for benchmarking and method development.

Dunning's aug-cc-pV*N*Z : The augmented basis set includes a diffuse Gaussian function that decays more slowly, to account for long-range correlation in anionic/Van der Waals complexes [108]. The 'p' stands for polarization, meaning that s-orbitals will have an additional p-orbital, and the p-orbitals will have an additional d-orbital to describe the polarization of the electron in respective orbitals. 'VTZ' stands for valence-only orbitals with triple zeta basis, i.e., each valence orbital is broken into a set of three different

orbitals, allowing more functions to mimic the 'real' orbital and give better results. The 'cc' means correlated consistent, meaning that DZ, TZ, QZ, and so on, will converge to the complete basis-set limit (CBS). This way, one can approximate the CBS limit for any theory (method) with few (3-4) basis sets. However, these calculations are expensive, and there is another method to correct errors for a finite basis set for bi-molecular collisions called counterpoise (CP) correction. The idea involves estimating the amount of artificial stabilization of one fragment on the other by the extra basis functions and correcting for the same. The same will be discussed in detail later.

Pople basis set $X-YZ+G^{**}$: Similar to the previous example, these basis sets use split valence orbitals [109] and introduce polarization and diffuse functions. While they are not correlated-consistent (cannot be extrapolated to CBS limit), these basis sets are less computationally expensive and are therefore widely used with DFT methods for studying organic molecules, e.g., 6-311+G(d,p).

The final solution for any combination of method and basis set is obtained by making an initial guess (trial wavefunction) and reiterating with results till the required accuracy is achieved (variational method).

Methods

The solution to a many-electron system starts with the Hartree-Fock method (single-electron Slater determinant). The electron correlation is then added by including contributions from various electronic configurations. When we use a complete set where all possible configurations are included (as a linear combination), the result is exact and is called "Full CI". This method scales exponentially with system size, and therefore, we presently rely on truncated methods that have viable scaling with system size.

Understanding scaling!

The scaling αN^x depends on the method, and N loosely represents the number of basis functions in our basis set (system size). Therefore, a system with a larger basis set (AV5Z) or with too many electrons (Fe, Pt, etc.) can quickly become large enough to overwhelm any modern computer.

HF method: The simplest theory, Hartree-Fock (HF) [101] scales with αN^4 in naive form. However, Schwarz screening [110] and density fitting [111] can bring the scaling down to $\alpha N^{<3}$. It has errors in the eV scale since it lacks electron correlation. The Hartree operator only represents the classic Coulomb repulsion between an electron and all other electrons in the system. The Fock operator, however, combines the core Hamiltonian (kinetic and nuclear-electron attraction), the Hartree term (electron-electron repulsion), and the exchange term (anti-symmetry of the wavefunction, accounting for quantum exchange interactions). Its wavefunction is a single Slater determinant, which is the anti-symmetrized product of one-electron molecular orbitals. As stated earlier, HF introduces an effective mean-field approximation for electron-electron interactions. While the true $\hat{H}$ involves explicit electron-electron interaction, HF approximates this interaction by replacing it with a mean-field potential created by all other electrons.

DFT: The density functional theory (DFT) methods [112] retain the same scaling $\mathcal{O}(N^3)$ with a larger prefactor (α), bringing the error down to the kcal/mol scale (chemical accuracy). Here, we rewrite the problem in terms of the electron density ($\rho(r)$) instead of the wavefunction. In hybrid DFT methods, the exchange-correlation energy functional is modified by including a portion of the Hartree-Fock exact exchange ($\alpha E_{Ex}^{HF} + (1-\alpha)E_{Ex}^{DFT} + E_{Corr}^{DFT}$). Though it is revolutionary for studying chemical bond formation/breaking, it does not provide enough accuracy for spectroscopic studies that require error in cm^{-1} scale. HF and DFT methods can currently be used to study systems with $\backsim 100$ atoms or more.

MPx: Another method based on the Møller–Plesset (MP) perturbation theory introduces an increasing order of correction in the Hamiltonian [113]. The new Hamiltonian consists of ($\hat{H} = \hat{H_0} + \hat{H'}$) where $\hat{H_0}$ is

the Fock term and $\hat{H}'$ is the perturbation term. These methods quickly become very expensive as system size increases (MP2 scales αN^5 for 2^{nd} order correction) and are rarely used since more and more specialized DFT methods(αN^3) have become available. However, current density-fitted (DF)-MP2 methods are fast and reliable for benchmarking purposes.

CI methods: Configuration interaction methods like CISD [114] scale as αN^6. The excitation recovers correlation from single and double excitations with HF as reference energy. These methods suffer from size consistency errors (incorrect dissociation limit) and size-extensive errors (incorrect scaling with system size) inherited from HF methods. CISDTQ scales as $\mathcal{O}(N^{10})$ and are not yet practical (with current computing power) for application in computational chemistry.

FCI: Full CI methods [115] are size-extensive and consistent. They converge to the exact solution, and are therefore great reference points for benchmarking small systems against any new development. But the method scales exponentially ($(\mathcal{O}(\binom{N}{n}))^3 \approx 2^{3N}$) and not polynomially ($\alpha N^x$), thereby limiting its use beyond minimal basis sets. Here, N is the number of orbitals, and n is the number of electrons.

CC methods: Coupled cluster (CC) methods [116–119] unlike CI are size-consistent and extensive due to the exponential nature ($e^{\hat{T}}$) of the cluster ($\hat{T} = \hat{S} + \hat{D} + \hat{T}...$) operator resulting in excitations with pseudo-higher order terms. The CC formulation is solved *via* projection [120] and can describe the ground state of any molecule with the same scaling as CI. The CCSD(T) method is considered the gold standard for molecular systems, where (T) indicates that triple excitations are included perturbatively. This method can describe weakly correlated systems (like van der Waals) with spectroscopic accuracy (< 1 cm^{-1}) and αN^7 scaling. However, this method can suffer from numerical instability at asymptotic regions [121] while describing the dissociation of very strong bonds [122] (e.g., triple-bonded N_2).

*** Strongly correlated systems**: A point to remember is that all these methods use a single HF as reference wavefunction and then use further approximate methods to account for missing electron correlation. Suppose HF does not recover (99%) of the exact energy (which is the case for strongly correlated systems), post-HF methods (especially with perturbative corrections) will perform poorly. Thankfully, the He and H_2 collision (van der Waals chemistry) lies in the region of weakly correlated systems. Systems involving metals and strong bond dissociation generally need multi-reference methods such as MRCI/MRCC (*also check out DMRG*).

Multi-reference methods: Multi-reference methods [123], such as MRCI/MRCC, have been proposed to deal with strongly correlated systems. MRCI methods, by design, can also be used to study the potential energy surface of excited states. In multi-reference methods, multiple Slater determinants ($|\Psi_{MR}\rangle = \sum_i c_i |\Phi_i\rangle$)

Table 2.1. Computational time for single point energy of CNCN (at experimental equilibrium geometry) is compared for various methods and basis sets using Psi4.

Method	Basis-Set	Time (s) *
HF	AVDZ	0.64
MP2	AVDZ	0.86
MN15 (DFT)	AVDZ	1.57
MP4	AVDZ	5.43
CCSD	AVDZ	7.13
CISD	AVDZ	8.53
Mk-MRCCSD(T)	AVDZ	17.05
CCSD(T)	AVDZ	7.94
CCSD(T)	AVTZ	74.31 ($\backsim$ 1 min)
CCSD(T)	AVQZ	839.81 ($\backsim$ 14 min)

* The calculations were performed on single thread of **Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz**

are used to capture important configurations for systems where a single determinant ($|\Phi_0\rangle$) is inadequate.

Table 2.1 illustrates the difference in the energy calculation time (CNCN molecule) for the available methods (in the Psi4 software package). At first, the methods are compared with a relatively moderate basis (AVDZ), showing how quickly the CC/CI methods become expensive. However, when the basis set is increased, the time required almost jumps to an order higher. The CCSD(T)/AVQZ takes $\backsim 14$ minutes for one single point. For 10,000 data points, it's almost 3 months of computation time. This is the reason why building PES can be challenging. A poorly chosen method/basis set (or even a simple mistake in the input file) can ruin months of computation time.

What options do we currently have to build collisional PES with appropriate accuracy?

The biggest problem of quantum chemistry is trying to solve chemical systems *ab initio* within spectroscopic accuracy (< 1 cm^{-1}) [69]. The chemically accurate calculations (< 1 kcal/mol) are achieved using DFT calculations and save us a huge amount of computational time for studying the reaction dynamics of large systems (mostly (bio-)organic compounds). But DFT methods are heavily parameterized to describe changes in chemical bonds and can have errors ($219474.6/627.5 =) \backsim 350$ times worse than spectroscopic accuracy. Therefore, quantum dynamics problems involving rotational states require solutions from far more computationally accurate (and currently expensive) theories described in Table 2.2.

Table 2.2. A summary of various methods in quantum chemistry with their application and limitations.

Theory (scaling $\mathcal{O}$)	Application	Limitation
Ground State		
HF (αN^3)	ground state reference energy	error $\backsim$eV $= \backsim 8000$cm^{-1}
DFT (αN^3)	bond breaking/formation	error $\backsim$kcal/mol $= \backsim 350$cm^{-1}
CCSD(T) (αN^7)	ground state spectroscopic PES	Fails for strongly correlated systems
Excited State		
MRCISD (αN^6)	Excited state PES	Size consistency error (SCE)
F12 methods	reduced basis set and time	Requires benchmarking & SCE

The coupled cluster (CC) method can describe the ground state PES with spectroscopic accuracy, but it fails miserably for strongly correlated systems (e.g. systems with multiple bonds) when stretched far from equilibrium. For weakly correlated systems, i.e. collision of two species with van der Waals interaction, CCSD(T) (scaling $\mathcal{O}(N^7)$) is a gold standard. For molecules where the CC method fails, multi-reference methods can come to our rescue. Additionally, density-fitted (DF) and F12 methods [124] can significantly reduce computational time. The role of each will be discussed later in this section.

There are also methods such as CASSCF (complete active space), RASSCF (restricted active space), CASPT2, RS2, etc., for advanced users who can filter out the energy states by freezing non-essential orbitals. This can be tricky for new users, and improper filtering of states can result in a disaster. Thus, always take time and benchmark PES with multiple methods. Even HF and hybrid DFT methods (such as B3LYP and M06) with dispersion correction (D2, D3, D3BJ, D4, etc.) are fine for a quick comparison, but take these results with a pinch of salt. At present, these methods should only be used for initial screening and grid search before starting calculations with more accurate methods. Remember to look for trends and outliers. Not all algorithms are equivalent and the idea is to look for methods (and their algorithms) that are reliable (benchmarked with experimental results such as CC/CI), robust (converge to expected results without numerical errors), and accurate (close to FCI/CBS as per your benchmark results).

Seeing is believing!

Before we begin with approximations (in methods), let us see how collisional PES behaves with the method and basis set. A quick literature survey will suggest that CCSD(T) with AVQZ (aug-cc-pVQZ) basis set and counterpoise (CP) correction is the standard for such calculations. But why? We shall see the same in the C-He collision PES plots provided in Fig. 2.4. To learn more about corrections and other approximations in quantum chemistry, please refer to the notes by The Sherrill Group.

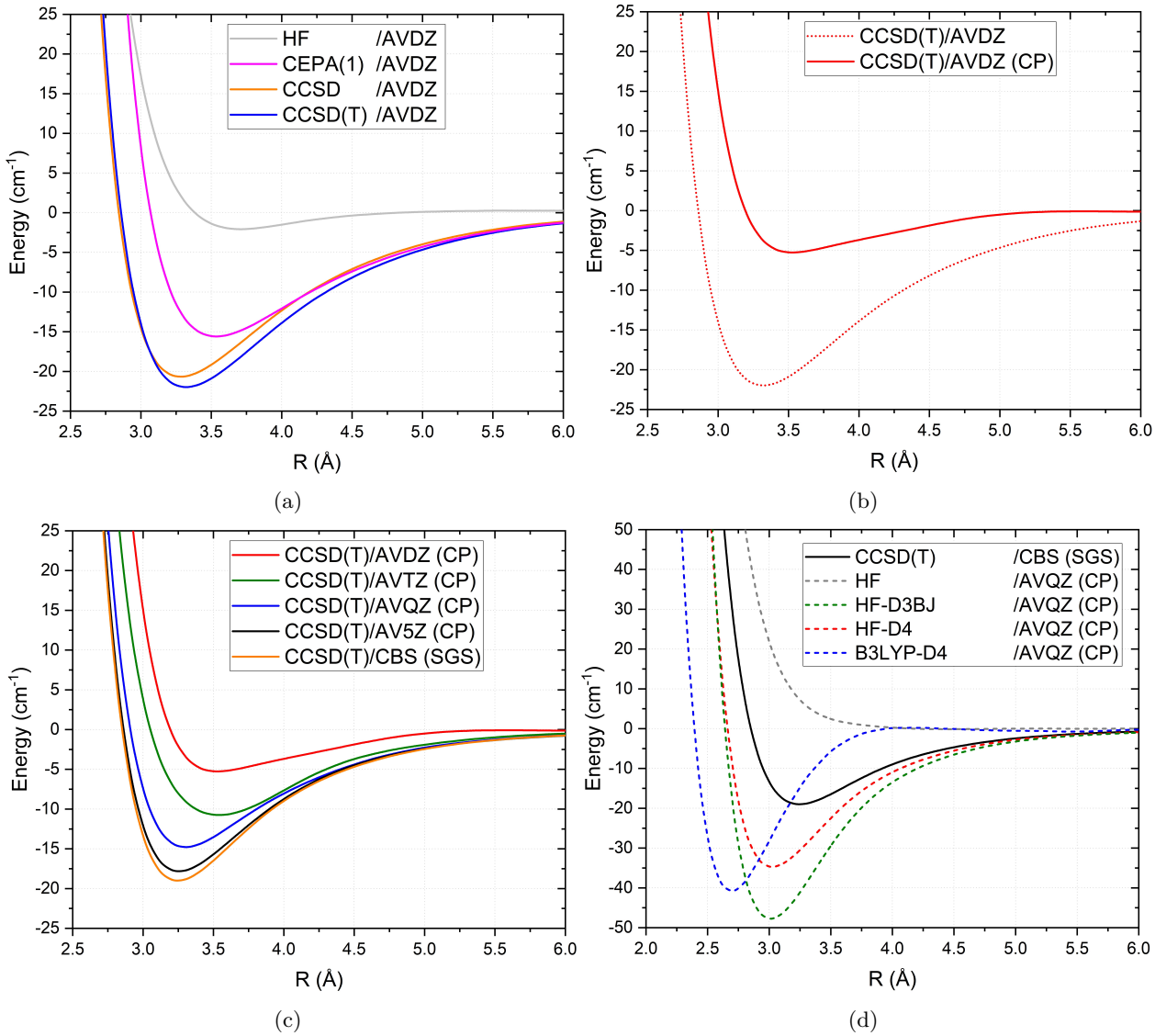


Figure 2.4. C-He collision PES showing (a) correlation energy recovered post-HF methods such as CCSD(T), (b) the effect of basis set with counterpoise (CP) correction, (c) CBS extrapolation vs CP corrected results and (d) performance of DFT methods (and dispersion correction) compared to CCSD(T)/CBS. † *Plotted using Origin.*
* All data have been calculated using Psi4 API. ! SGS stands for Sherrill's Gold Standard

Role of post-HF methods in recovering electron correlation!

Fig. 2.4 (a) shows that the simple Hartree-Fock calculation does not capture full electron correlation as the interaction energy is much smaller than other post-HF methods. Methods like CEPA(1) and CCSD add electron correlation, increasing the well depth. The CCSD(T) method produces a well depth $\backsim 5$ times

larger than HF for C-He van der Waals interaction. CCSDT(Q) will surely add some more correlation, but those calculations are way too slow (as well as memory-intensive) to be practical, and their results should also be close to CCSD(T), which can recover more than 99% of the correlation energy if the system is weakly interacting.

Effect of including counterpoise correction!

Going further to Fig. 2.4 (b), we see the role of the basis set superposition error (BSSE). *What is BSSE?* When a monomer (A) approaches another monomer (B), the dimer can be artificially stabilized as each monomer uses the extra basis functions from the other to describe its electron distribution. This extra artificial stabilization is what results in BSSE. We use the counterpoise (CP) correction method to remove the BSSE. The CP method results in the removal of extra stabilization energy, making the minima shallower compared to the non-CP PES. However, we shall see that the DZ basis is not enough to capture the correct picture of PES in the next plot.

Effect of using a complete basis set extrapolation!

Fig. 2.4 (c): even after correcting BSSE, we still have an error in our interaction energies, which is called "basis set incompleteness error" (BSIE). This error arises from the fact that the basis sets are incomplete, therefore, we must either use a large QZ/5Z basis with CP correction or use the complete basis set extrapolation, where we approximate infinite basis set energies. Here, we use the 'Sherrill's Gold Standard' (SGS) method for CBS extrapolation, which is directly implemented in Psi4.

So CBS methods should not have BSIE? Not really, since there are no practical 'infinite' basis sets. We use finite basis sets to approximate the complete basis set (CBS) limit. So, the error in theory should be zero, but in practice, it will be finite but close to zero.

Effect of using dispersion corrected methods (HF/DFT)!

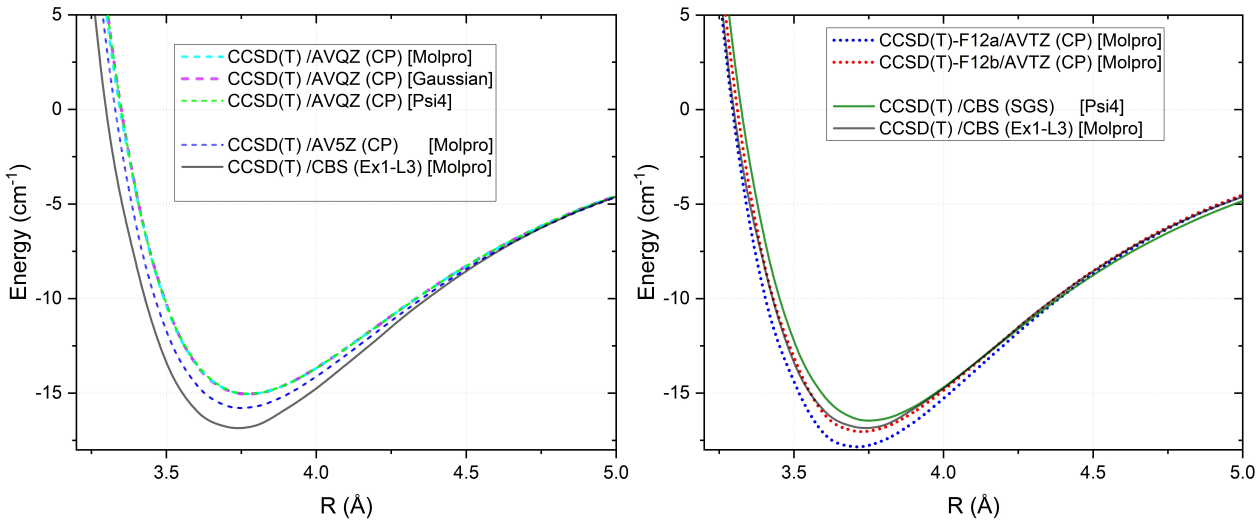
Fig 2.4 (d): DFT methods perform poorly for spectroscopic PES calculations due to parametrization and can only offer chemical accuracy. However, they can be beneficial to evaluate the rough PES landscape before beginning a fully accurate PES. The shape, location/depth of minima, and RMSE are important factors in determining an approximate method and basis set. In the plot, the CCSD(T)/CBS result is shown for reference. As observed, the B3LYP method with D4 dispersion correction performs poorly in estimating both the well depth and location of minima. The same was observed for other DFT methods (M06, MN15, HSE06, wB97X, PBE0, LDA0, etc.) which produced a well depth > 100 cm^{-1}, while simple HF barely registers any minima. The better option is to use HF with dispersion correction (D4 or D3BJ) as the results retain the proper shape of PES and location of minima. The error is still large, but the approximate PES can help in choosing the proper grid size for computationally expensive CC/CI calculations.

Benchmarking any PES!

Overall, at the current stage, CC and CI methods (single HF reference or multi-reference) with SD or SD(T) excitations are the choice of methods we have for spectroscopic studies. While higher excitation theories (methods such as CCSDTQ) may give better correlation energies, they are also 'overkill' for weakly bound systems. In the future, the computation power/efficiency and density fitting (DF) approximations could probably make higher excitation (say CCSDTQ5) calculations more efficient, but the current algorithms are not so 'user-friendly' for PES calculations with large basis sets.

However, instead of relying on CBS methods, one can opt for approximations that can produce CBS energies (in theory) with a smaller basis set. One such approximation for CI and CC methods is F12, which initially requires benchmarking with CCSD(T) methods using CBS or a counterpoise-corrected large-basis

set. Currently, these F12 methods are implemented in Molpro. An example for benchmarking C_2-He collision at $\theta = 90°$ is provided in Fig. 2.5.



(a) (b)

Figure 2.5. Benchmarking F12 approximate methods with CBS extrapolated methods. The PES is generated for C_2-He collision at $\theta = 90°$. Subfigure (a) shows equivalence in CCSD(T)/AVQZ(CP) results for Psi4, Molpro, and Gaussian 16 while comparing them to larger AV5Z and CBS basis. Subfigure (b) compares the performance of F12 approximations with AVTZ(CP) basis with CBS results. † *Plotted using Origin.*

Fig. 2.5 (a) illustrates that different *ab initio* packages yield the same results for CCSD(T)/AVQZ (CP) and are therefore reliable. The AV5Z(CP) and CBS basis improves the results marginally (1-2 cm^{-1}) and can be avoided for large systems as the time taken by the last two methods is 2-10× the AVQZ(CP) method. In Fig. 2.5 (b), two different F12 approximations for CCSD(T), i.e., F12a and F12b, have been used with a reduced basis AVTZ(CP). The results for the CCSD(T)-F12b method are found to nearly overlap with CBS results from Molpro. This is crucial for studying 4D collisions, as such approximations can further reduce computational time by 2-5× depending on the system. Another piece of information that we get from the plot is that not all CBS methods are equivalent [106]. The SGS extrapolation in Psi4 produces a slightly shifted PES compared to EX1-L3 CBS extrapolation in Molpro that uses AV[TQ5]Z for extrapolating energies. The details of the same are provided below.

Details about various approximate and correction methods!

BSSE: To remove the basis set superposition error (BSSE), we subtract the extra artificial stabilization by calculating BSSE contributions from each fragment. This contribution is calculated by subtracting the energy of the first fragment from the energy of the same fragment in the presence of the second (ghosted) fragment [125]. The mathematical formula of the same is provided below and the same is also implemented in the counterpoise corrected Molpro template.

$$\begin{aligned} E_{\mathrm{AB}}^{\mathrm{CP}} &= E_{\mathrm{AB}} - E_{\mathrm{BSSE}}^{\mathrm{Total}} \\ &= E_{\mathrm{AB}} - \left[E_{\mathrm{BSSE}}^{\mathrm{A}} + E_{\mathrm{BSSE}}^{\mathrm{B}}\right] \\ &= E_{\mathrm{AB}} - \left[\left(E_{\mathrm{AB}}^{\mathrm{A}} - E_{\infty}^{\mathrm{A}}\right) + \left(E_{\mathrm{AB}}^{\mathrm{B}} - E_{\infty}^{\mathrm{B}}\right)\right] \end{aligned} \tag{2.6}$$

Always use CP correction if any CBS extrapolation is not used. The choice of basis set can depend on

the system, but stick to AVQZ/AV5Z for benchmarking small 1D/2D PES, and AVTZ with F12 methods for 4D PES generation.

SCE: There is one more error that is inherent to methods that are not size consistent, such as truncated CI and F12 methods [126, 127], i.e., size consistency error (SCE). This means that the total energy of a system having two non-interacting fragments (at infinite separation) does not equal the sum of the individual fragment energies.

$$E_{\mathrm{AB}}(r=\infty) \neq E_{\mathrm{A}} + E_{\mathrm{B}} \tag{2.7}$$

For our application, this means that the PES at infinite separation may vary depending on the angle for our 2D/4D PES. Here, we can do SCE correction by subtracting energies of each angle at large separation, say 100/200 Å, respectively.

$$E_{\mathrm{AB}}^{Corr}(r,\theta) = E_{\mathrm{AB}}(r,\theta) - E_{\mathrm{AB}}(r=\infty,\theta) \tag{2.8}$$

Usually, the residuals are small but can amplify for strongly bonded systems. Therefore, it is a good practice to remove SCE, especially for CI and F12 methods.

BSIE: The error arising from the finite size of basis functions is called basis set incompleteness error (BSIE), which in theory can be removed using complete basis set (CBS) extrapolation. The extrapolation results of any two CBS methods can differ based on the functionals. A smart way to approach such calculations is to first generate a part of the PES using AVQZ/AV5Z with CP correction. If the CP and CBS energies are similar, but the shapes are not, it is better to stick to CP methods, as CBS extrapolation functionals can introduce artifacts due to functional parameterization.

For example, in Molpro, there are a few different functionals that can extrapolate the reference and correlation energies separately. Based on my personal experience, the difference in the results of various functionals is rather small (except for the LX functional) if one opts to extrapolate till AV5Z. For a smaller basis (till AVQZ), the results do vary, but there is one combination that outperforms others in terms of simplicity and robustness, i.e., $EX1-L3$.

$$\begin{aligned} E_{\mathrm{EX1}} &= E_{\mathrm{CBS}} + A\exp(-Cn) \\ E_{\mathrm{L3}} &= E_{\mathrm{CBS}} + An^{-3} \end{aligned} \tag{2.9}$$

Here, n is the cardinal number of the basis set (e.g., 3 for TZ), and A/C are the fitting coefficients optimized using the least squares method. The recommended example in Molpro uses $EX1$ functional with three basis sets (AV[TQ5]Z) for reference (HF) energy extrapolation. In contrast, it uses another keyword $npc = 2$, signifying that only the last two basis sets (AV[Q5]Z) will be used for extrapolating correlation energies and recommends the $L3$ functional for the same. This is a relatively robust method that can be a good reference point for benchmarking any approximate method.

In Psi4, the Sherrill-Gold-Standard (SGS) is a fast and robust CBS method for computing PES.

$$E_{\mathrm{CBS}} = E_{\text{total, SCF}}^{\text{aug-cc-VQZ}} + E_{\text{corl, MP2}}^{\text{aug-cc-pV[TQ]Z}} + \delta_{\mathrm{MP2}}^{\mathrm{CCSD(T)}}\Big|_{\text{aug-cc-pVTZ}} \tag{2.10}$$

Here, HF is computed using the aug-cc-pVQZ basis set, and MP2 correlation energy is extrapolated using aug-cc-pVTZ and aug-cc-pVQZ. The final term corrects the MP2 correlation energy by adding the difference between MP2 and CCSD(T) at the aug-cc-pVTZ level. This makes the CBS computations faster compared to traditional CBS methods. The two-point extrapolation of MP2 energies is carried out using:

$$E_{corl}^{X} = E_{corl}^{\infty} + \beta X^{-\alpha}. \tag{2.11}$$

F12: The F12 approximation methods are designed to converge to the CBS limit with a smaller basis set [124]. The upside is that we can get spectroscopic accuracy ($\backsim 1\text{cm}^{-1}$) for PES generation at a much lower computational cost. The only careful consideration for F12 methods is that they must be benchmarked with a known method and a large (CP) or CBS basis set like the example in Fig. 2.5 (b) where F12b method performed better than F12a method with AVTZ basis set.

Since the PES of a rigid rotor − rigid rotor 4D PES can take months to calculate, therefore, dissociation curves for a few orientations must be benchmarked with the two methods mentioned below, before starting a full 4D PES calculation. The final choice of CCSD/CCSD(T) | F12a/F12b | AVTZ(CP)/VTZ(CP)/VDZ, etc., can depend on the system size. If the system is too large, use the less expensive method with a smaller basis. However, if possible, try to also calculate the PES minimum by calculating the potential at R_{eq} and R_{∞} using a larger basis to evaluate the error at minima.

1. CCSD(T)/CBS(TQ5) or CCSD(T)/CBS(DTQ).
2. CCSD(T)/AV5Z(CP) or AVQZ(CP).

The first option (a) corresponds to the basis set limit with 2-3 large basis functions (preferably augmented). The second method (b) ensures that the CBS expansion method (there are many), is appropriate and does not significantly wander off from the expected results. The AV5Z method may not always be ideal, as it may be too computationally expensive. Therefore, one can also use the AVQZ(CP) method since it has been the standard basis for many PES calculations for the previous decade. If CBS results are poor or too expensive, the ideal choice for the benchmark should be CCSD(T)/AVQZ(CP). The F12 methods can slightly over/undershoot the standard CCSD(T) method with a larger basis. Still, if the results have RMSE of $\backsim 1\text{cm}^{-1}$, F12 methods can save a significantly large computational time for the same PES since we require a smaller basis set. Once the F12 method/basis has been benchmarked against the standard method/basis, one must proceed with the same for the calculation of the remaining PES. For charged systems (especially anionic), long-range correlation becomes important, therefore, an augmented basis must necessarily be used. The reference sheet for the choice of method/basis is listed in Table 2.3.

Table 2.3. Reference for choice of basis set and method. 2D collision: atom−rigid rotor and 4D collision: rigid rotor−rigid rotor. VXZ: use D/T zeta basis depending on computational resources.

State	Method
Ground State	Bond: Weak - CCSD(T) | Strong[#] - MRCCSD
Excited State	MRCISD / MRCCSD
Condition	Basis Set & Approximations[*]
4D PES	(A)VTZ basis | Molpro: use F12 approximation
2-8 light atoms	2D PES: AVQZ(CP) or CBS | 4D PES: F12/AVTZ(CP)
8+ heavy atoms	F12/VXZ basis set
anionic Van der Waals complex	Use augmented basis set
cationic/neutral complex	Use augmented unless too expensive for calculation

[#] Use multi-reference methods and avoid perturbative corrections for strongly correlated systems.
[*] Always benchmark approximate methods with CCSD(T)/CCSD and AVQZ/AV5Z/CBS whichever is feasible.

Overall, the standard choice of method and basis set combination for ground state PES is CCSD(T) with AVQZ(CP), and excited state PESs is MRCISD or MRCCSD with AVQZ. Counterpoise correction can be omitted for the excited state. However, for MRCI, Davidson correction must be used to improve energies. Users can also opt for the MRCI-F12 method with a smaller basis after appropriate benchmarking.

2.3.4 PES Generation

A popular Fortran based program AUTOSURF [128] can generate generalized collisional PES, and can be explored by users. The PES2MP uses a Python-based open-source *ab initio* package called Psi4. The package is implemented to generate PES internally as well as generate input files (at all required coordinates) for carrying out calculations later. The internal calculations with Psi4 API (i.e. conda installed Psi4 used for internal calculations) are there for quick PES generation, and after satisfactory results, the external calculations can be set up by generating input files for Molpro [129], Gaussian [130, 131], or Psi4 [2] (need separate installation).

The internal calculation must be used for reference only, i.e., PES must be generated using HF(D4)/DF-MP2 or other less expensive methods with a smaller basis. This is suggested as multi-threading does not work properly, and sometimes numerical errors can stop the code, which cannot be restarted (even with the try-except block).

The Psi4 software (installed using the dedicated installer and not conda) must be used to calculate the full PES, as it will offer the use of multiprocessing. The failed calculations in this case will just be skipped, and therefore, will not affect any subsequent calculations. In the manual, the conda-installed Psi4 package will be referred to as 'Psi4 API', and Psi4 installed using a dedicated installer will be referred to as 'Psi4 software' or 'Psithon'. The Psi4 API can create 1D/2D/4D **reference PES** automatically by calculating the center of mass (COMs) for rigid rotors. The only data needed for PES generation is:

1. atom(s) for the first atom/rigid rotor,
2. atoms(s) for the second atom/rigid rotor,
3. equilibrium bond lengths (experimental/theoretical) for each rigid rotor,
4. charge and multiplicity of both colliders and the overall system,
5. method/basis set.

The generated (rough) PES should give a better idea about regions requiring a denser grid size for the high-level PES calculation. Once this step is achieved, use the program to generate external PES input files for Gaussian, Molpro, or Psi4. The three software are compared in Table 2.4.

Table 2.4. A review of *ab initio* packages for PES generation.

	Gaussian	Molpro	Psi4
License	Paid	Paid	Open-source **(Free)**
Programming/Plugins	× / !	✓ / !	✓ / ✓
CCSD(T)	✓	✓	✓
Approximations	!	F12 methods	Density-Fitting
Excited states	CASSCF/**EOM**	CASSCF/**MRCI**/EOM	CASSCF/**MRCC**/EOM
Ease of use	Easy (GUI)	Intermediate to Hard	Intermediate

! stands for limited or paid availability.

Chemists use Gaussian 09/16 to model chemical reactions due to its visualizer and easy-to-use interface. Almost no aspect of Gaussian requires the user to be familiar with UNIX/codes (except when installing the same on LINUX machines), and its visualizer can handle calculations/results. While the CCSD(T) and EOM-CCSD implementations of Gaussian are more memory demanding, their implementation (especially for the **unrestricted basis**) is very **stable**. This can be useful for studying non-singlet ($S > 0$) species.

On the other hand, Molpro has a more programmer-friendly interface and allows users to create variables, tables, etc., inside the input file. The software has a fast and memory-efficient CCSD(T)/MRCI implemen-

tation for symmetric systems (if molecules are linear, planar, etc.) along with F12 approximations, which make it very competitive. Currently, Molpro is the **widely used** package for building collisional PES.

Psi4 software combines the use of high-level packages (slow but easy to code) of **Python** and fast implementation of low-level packages. The software is constantly under development with new features being added. Currently, it is still undergoing development, however, its biggest advantage is its free license and integration with Python. It is developer-friendly and a must-know software for computational chemists, given its **availability as an API** and potential for building codes around the environment. The use of density-fitted (DF) methods (default for HF/MP2 and CCSD) is recommended to save time. The template for building PES in PES2MP program are provided in Table 2.5:

Table 2.5. A review of templates provided in PES2MP for *ab initio* PES generation.

PES2MP Input File Templates	Gaussian	Molpro	Psithon (Psi4 exec.)	Psi4 (API)
CP corrected PES	✓	✓	✓	✓
CBS extrapolated PES	*o*	✓	✓	✓
Excited States	*o*	✓ (MRCI)	✓ (MRCC)	*o*
SAPT	*o*	✓	✓	*o*
Other custom calculations	*o*	✓	✓	*o*

✓ : Template provided | *o* : Template not provided.

The comprehensive details about each input file are provided in the next section of the manual.
Some additional (axillary) scripts are also provided in the "input_files/aux_scripts" folder:

- `batch_scripts`: Scripts to run Gaussian, Molpro and Psi4 external calculations.
- `PES_extract`: Scripts to extract PES data from individual files in $R, \theta_n .., E$ format.
- `PES_to_cm`: Scripts to convert Energy from Hartree to cm^{-1} by subtracting E at $R_\infty(\theta)$ for each angular term. This is important to remove the size consistency error in F12 methods.

2.4 Analytic Curve Fitting

Analytic representation is the use of a function with coefficients (weights) and variables (say R), where plugging in the variable (with values) gives you the original data back. The simplest example will be to fit data based on Beer's Law $A = \varepsilon bc$. As chemists, we all know that for dilute solutions, absorbance and concentration are linearly related. Instead of saving data with 10,000 data points, one can simply fit the data into the $y = mx + c$ format. This will save disk space while also allowing the user to get molar absorptivity from the slope of the linear fit.

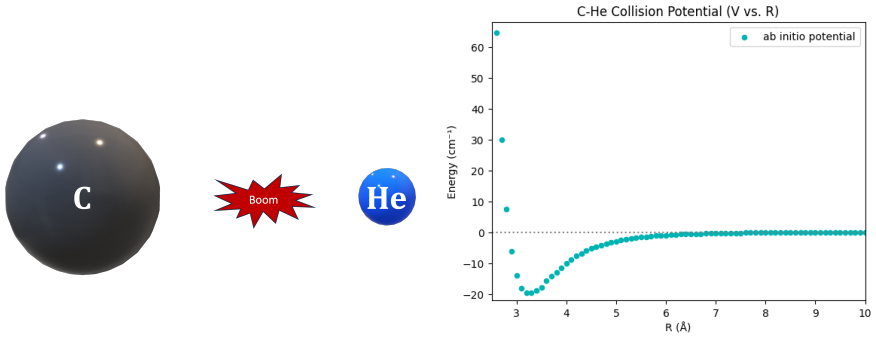
Analytic representation of any numerical data is a popular method to save gigabytes/terabytes of data. So, analytic representations are nice, but they come with a catch. Errors! For a PES, there are several regions of varying slope and to analytically describe the PES using a function can result in a large number of parameterized functions with conditions and switching [132] parameters.

A 1D PES curve depending on R (distance) can be described by a few well-known functions [133, 134]. However, the problem that many face in computational chemistry/chemical physics is the dimensionality (dependent features) of data. Suppose one needs to augment (spline/interpolate) a dataset that took months or even years to generate *ab initio*. If someone can carefully express it in an analytic form (considering errors in an acceptable range), with all features, they can save a huge amount of computational resources. In analytic form, any number of data points (potentials) can be created by just plugging in the variables

(XYZ coordinates). We shall now examine how 1D and 2D data can be represented analytically and why multi-dimensional fitting is a painstaking process.

2.4.1 1D Fitting

Consider the dissociation curve of the C-He molecule. It is a one-dimensional (1D) potential energy surface (PES). Why 1D? Because the final result, i.e., the potential (V or E), depends on only one parameter, R, where R is the distance between the Carbon and Helium atoms. Therefore, we can also represent the final potential with its dependent variable as V_R. To visualize the same, Fig. 2.6 (a) gives the static image of a collision between two atoms and (b) gives the V (cm^{-1}) vs R (Å) plot for the same.



(a) (b)

Figure 2.6. (a) 1D collision between C-He and, (b) the *ab initio* potential (CCSD(T)/AVQZ). † *Image (a) is created using PowerPoint and (b) using Matplotlib.*

A simple Jupyter-Notebook code to fit the 1D PES ("*1D_fit_aux.ipynb*") is provided in the Git-Hub repository *apoorv-kushwaha/PES2MP-IPython* inside '*curve_fit folder*'. The code fits the data given in "*scan_tot_energy.dat*" (C-He collision PES) into functions given by Eq. 2.12 by making use of the *SciPy's curve_fit* function. It uses a standard least squares fit to find the best possible coefficients for the function.

$$V = a_1 * e^{-1.5*x} + a_2 * e^{-3*x} + a_3 * e^{-4*x}, \tag{2.12a}$$

$$V = b_1 * e^{-2*x} + b_2 * e^{-3*x} \tag{2.12b}$$

The users are encouraged to play around with the simple function (by making it more complex) or by changing constants in the exponential. A good fit is not easy to find, but with a little tuning, a good initial guess, and restrictions on lower and upper bounds, the job can be done.

```
def exp_fit3(x, a,b,c):      # Function 1
    return  a*np.exp(-1.5*x)+b*np.exp(-3*x)+c*np.exp(-4*x)
def exp_fit2(x, d,e):        # Function 2
    return  d*np.exp(-2*x)+e*np.exp(-3*x)

# Output:
" Fitting coefficients for exp_fit function
[a b c] :  [-4606.3870857   139821.89094076 3297345.42536237]
[d e]   :  [-41463.48481245 717756.5884436 ]

Double exponential RMSE1 =  0.207488977651915
Double exponential RMSE2 =  0.595123836146574 "
```

Listing 2.1. Python code and output for 2 and 3 exponential function fitting

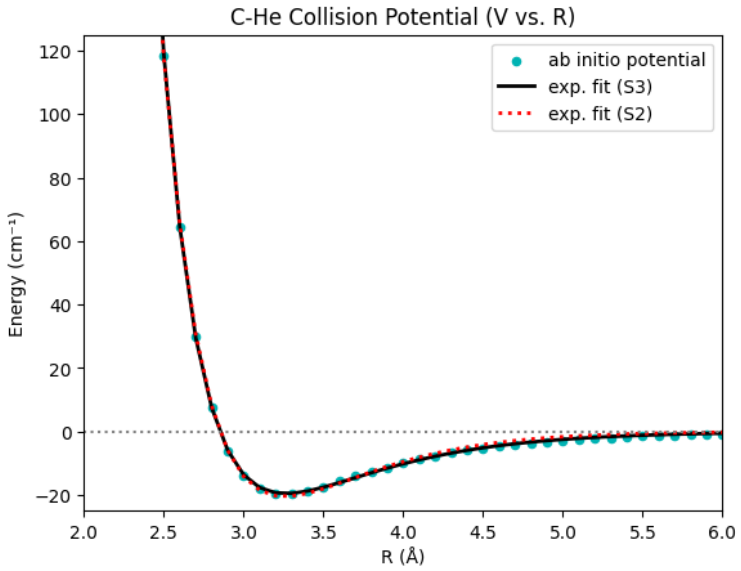


Figure 2.7. The *ab initio* potential of C-He at CCSD(T)/AVQZ fitted using two and three Slater functions.
† *Plotted using Matplotlib.*

We can see that the three exponential functions in Eq. 2.12 (a) give a slightly lower error of 0.2 cm^{-1} compared to the second function in Eq. 2.12 (b) with RMSE of 0.6 cm^{-1}. To bring the error further down, we can generalize the function even more to find more suitable parameters.

To test the same, the function in Eq. 2.12 (b) is changed to 2.13 given below,

$$V = b_1 * e^{-c_1 * x} + b_2 * e^{-c_2 * x} \tag{2.13}$$

When the PES data is fitted into the above function using the SciPy library, without using constraints, it gives the output:

```
"/usr/local/lib/python3.10/dist-packages/scipy/optimize/_minpack_py.py:906: OptimizeWarning:
    Covariance of the parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated'."
```

Listing 2.2. Output for generalised 2 exponential function fitting

It is observed that the same code is now completely overwhelmed by the number of possible ways the curve fitting can be achieved and fails to do the task. A way around this is to put constraints on the values the exponential coefficients (c_i) can take. This is something important to remember. Both Eq. 2.12 (b) and Eq. 2.13 are the same function if we set $c_1 = 2$ and $c_2 = 3$, but the code cannot find optimum parameters for the Eq. 2.13 due to the large search space ($-\infty$ to $+\infty$). Therefore, for a large number of parameters, the curve fitting will only work if the search space is reduced to some finite window.

One may, in theory, think that having more functions must always be better. But apart from the program failing to optimize the coefficients (due to too many combinations), there is an underlying problem of overfitting. Our potential data (for C-He collision) requires the function to behave like a Lennard-Jones potential (a function almost all chemists are familiar with) i.e. it should tend towards infinity at small R, 0 at large R, and a minimum at some finite R. Having too many functions will allow for a lot of variance outside the known range of R. An example of the same is shown Fig. 2.8 for Eq. 2.14.

$$V = d_1 * e^{-1.5 * x} + d_2 * e^{-3 * x} + d_3 * e^{-6 * x} + d_4 * e^{-7 * x} \tag{2.14}$$

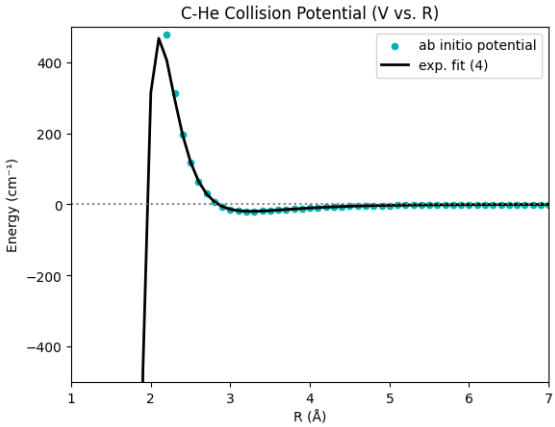


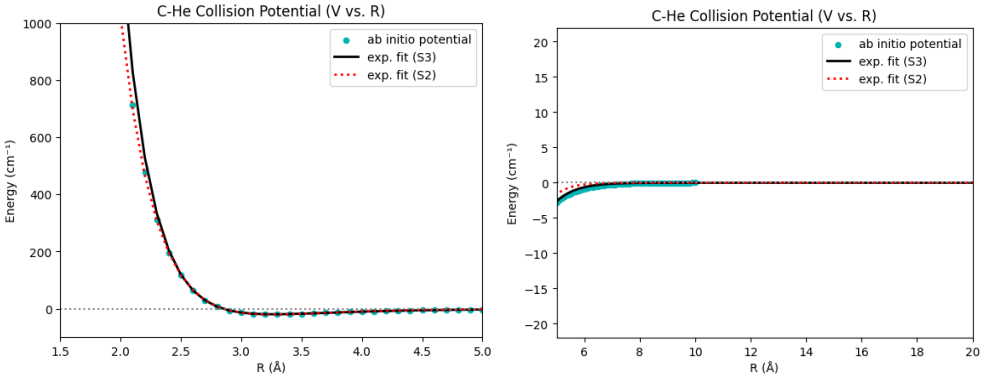
Figure 2.8. Figure showing incorrect behavior of four Slaters fitting at $R \to 0$. † *Plotted using Matplotlib.*

Fitting into the above function gives a lower error of 0.18 in the fitting range (i.e., 2.5 Å to 10 Å). But this also results in a poor description of the high-energy region where the curve suddenly drops to $-\infty$.

Sometimes the problems are even worse, where the curve may show artificial maxima and minima at places it should just connect smoothly (suppose you have a long break between two points towards infinity and a hump appears between those two points out of nowhere). So, in practice, remember:

1. The data must not under-fit (high error).
2. The data must not over-fit (low error for fitted coordinates but high error and artificial features for unseen coordinates).
3. The data must fit into smooth, well-behaved (as expected/required) functions.

If we look at the two (S2) and three (S3) Slater fits more closely in Fig. 2.9, we see that while the S2 curve has a slightly higher error in the asymptotic region, it has a better fit in the high-energy region. This is an important thing to address before proceeding. A poor asymptotic behavior can gravely affect the dynamical calculations (especially in charged species). The accuracy for high energy fits can be considered up to a certain threshold, such as (a) the energy corresponding to the first bending frequency of the rigid rotor or (b) the collisional energy till which dynamical calculations are performed.



(a) (b)

Figure 2.9. The (a) high energy and (b) asymptotic region obtained by curve fitting with 2 and 3 Slater functions. The PES data is a 1D collision between C-He (CCSD(T)/AVQZ). † *Plotted using Matplotlib.*

Overall, with a bit of wit and intuition, we can find a good function (with a range for the parameters) that is best suited for our application.

Why Slater Functions!

The reason we looked at the example of Slater functions is that they are implemented directly into the MOLSCAT software. When we fit PES into $\sum_i \alpha_i e^{-\beta_i R}$ functions with fixed β_i coefficients, the same function can be used to fit radial terms $V_\Lambda(R)$ (obtained after multipole expansion) without any error for the full range of R (the details of the same are available in the next chapter). However, if the same is not required, one can use any function of choice to fit the radial PES.

The PES2MP package can fit radial PES ($V(R;\theta_i)$) for each combination of angular terms separately in 2D/4D PES. This is beneficial when the rigid rotor has many atoms (making it longer) and causes minima to shift significantly for perpendicular (90°) and linear (0°) collisions. At (0°), the high energy at small R can be extrapolated easily if we can fit PES into simple Slater functions.

Use Slater functions but prevent curve inversion! The PES2MP package uses lmfit, which is a high-level library based on SciPy's curve fit. Its implementation is faster, and it can handle more variables before the optimization starts failing. If a large number of Slater functions are used to fit the PES, it is obvious that at some angles, the PES curve will show inversion (at small R). While the same can be prevented by varying the high energy cutoff or upper and lower bounds of the coefficients, there is another option, i.e., using a series of paired Slater [135] functions:

$$\begin{aligned} V(R) &= \sum_i \alpha_i (e^{-b_i R}) && \text{Simple Slater Functions} \\ &= \sum_i \alpha_i (e^{-b_i R} - e^{-c_i R}) && \text{Paired Slater Functions} \end{aligned} \tag{2.15}$$

Here, we shall use user-defined b_i (and c_i) coefficients (constants), and use PES2MP to optimize α_i coefficients. Currently, the optimum b_i (and c_i) coefficients are found using the hit-and-trial method. These functions can be generated using the 'Fn_generate_P.py' file. The details of the same are available in the next chapter. However, the theoretical details of the same are provided here. The large number of paired Slater functions can sometimes fit the PES without curve inversion. Another simpler method available in PES2MP is to use custom functions for high-energy (also available for long-range) regions.

The current version of MOLSCAT is restricted to $\alpha R^{-\beta}$ or $\alpha e^{-\beta R}$ functions. Given the versatility of Slater functions, they can be used to mimic any R^x function and have a lower error. Therefore, it has been implemented directly in PES2MP in two formats, 'simple' and 'paired'.

Coupled functions!

Since a single Slater function does not have the shape of the PES (anharmonic oscillator), we have to use multiple functions in series to fit the PES. There are, in theory, some functions that can mimic the shape of the radial PES:

$$V(R) = -\alpha \times (e^{-\beta R}) \times \ln(\gamma R) \qquad \text{Decay function with minima} \tag{2.16}$$

where α, β and γ are restricted to non-negative values. The coupled Slater-log Function has properties of PES i.e. V approaches ∞ as $R \to 0$ and vice versa (V approaches 0 as $R \to \infty$). Overall, there is not much literature on this function, and finding the optimum values with boundary conditions is slightly tricky. Nevertheless, the above function is optionally implemented in the NN model to put constraints on R and is discussed in the next section. The presence of minima makes it similar to PES collision among neutral species. For the collision of a cationic species, there is usually a small maximum just after the minima. That can be added by modifying the above function to include another log function (E1L2) as listed below:

$$
\begin{aligned}
V(R) &= -a_1 \times (e^{-a_2 R}) \times \ln(a_3 R) && \text{Decay function with minima : E1L1} \\
&= \ \ a_1 \times (e^{-a_2 R}) \times \ln(a_3 R) \times \ln\big(a_4 R\big) && \text{Minima with maxima : E1L2} \\
&= \ \ a_1 \times (e^{-a_2 R}) \times \ln(a_3 R) \times \ln\big(a_4 R^{-a_5}\big) && \text{Minima with maxima (flexible) : E1Lx} \\
&= -a_1 \times (e^{-a_2 R}) \times \ln(a_3 R) \times e^{(-a_4 R)^{a_5}} && \text{Allows only quantized values of } a_5 \text{ : E2L1} \\
&= -a_1 \times (a_3 R)^{-a_2} \times a_2 \ln(a_3 R) && \text{Decay function with flexible minima : EL1L1} \\
&= -a_1 \times a_2 (R - a_3) \times e^{a_2 (R - a_3)} && \text{Shifted decay function with flexible minima : E1x}
\end{aligned}
\tag{2.17}
$$

where coefficients a_i $(i \in Z)$ are restricted to positive values and e is the exponential constant. While these coupled functions retain the shape of radial PES, they are quite rigid and have larger errors compared to a series of Slater functions.

2.4.2 2D Fitting

Let us now increase the dimensions to 2D. Consider the dissociation curve of C_2-He molecules. It has a total of three dimensions (R, r_{cc}, and θ), but if we consider a C_2 molecule to be a rigid rotor, it reduces to 2D (R and θ). Therefore, our final potential can be represented as $V_{R,\theta}$ as shown in Fig. 2.10.

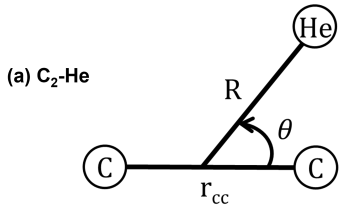


Figure 2.10. (a) 2D Jacobi coordinates for C_2-He collision.

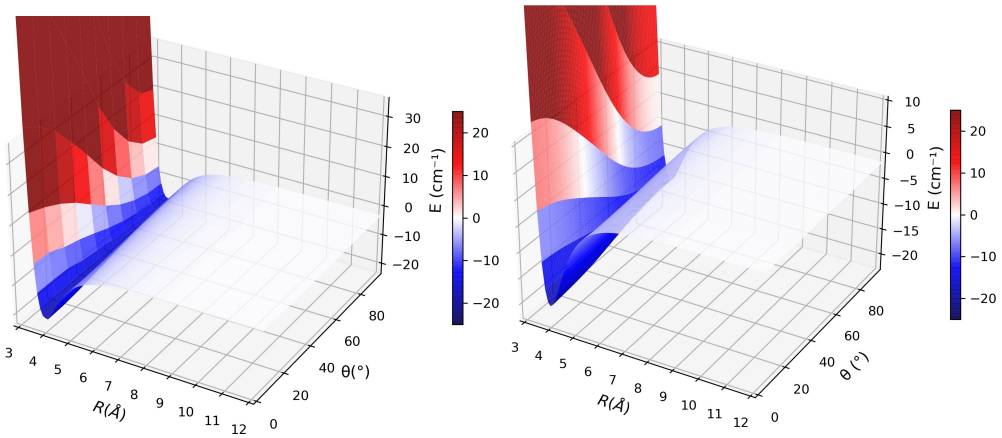
Now, finding a coupled 2D function is a bit tricky, and one can do the exercise to figure it out. A clever way out is to consider the 2D surface as a bunch of 1D curves and use the previous equation (or modified ones) to fit each 1D curve (e.g.$V_{R,\theta=0°}$, $V_{R,\theta=15°}$, etc) by using a loop. This way, one can spline data points across R and θ one by one, but not both at once. Similar strategies can be used for higher dimensions. Overall, the task is tedious and requires attention to any unwanted features or poor fits.

A still better way is to use multivariate interpolation/splining (implemented in SciPy as *SciPy. interpolate. RectBivariateSpline*) [136]. However, such tricks fail too as the dimensionality of data increases and the number of data points is very low [137]. A simple 2D splined dataset is shown in Fig. 2.11. The original data 2.11 (a) has θ coordinates at a step size of $\Delta\theta = 10°$ and looks very coarse (the data boundaries are not smooth and almost look patchy). The splined dataset ($\Delta\theta = 1°$) shown in Fig. 2.11 (b) is much smoother compared to the original (*ab initio*) dataset.

While the 2D spline works for a potential with small anisotropy, the splining method can become overwhelmingly difficult to implement if we are dealing with non-rigid rotor molecules or molecule-molecule collisions. The one-fits-all functions and parameters are hard to find and are often case-to-case specific [139].

Can't we just calculate the data points using a finer grid in the first place?

Well, the number of data points in the splined dataset (Fig. 2.11 (b)) increased by a factor of 10. The *ab initio* calculations for the same can now take months compared to days. And time will be a bottleneck for *ab initio* calculations where spectroscopic accuracy is required (i.e. error < 1 cm^{-1}). Currently, that means



(a) Coarse (b) Splined

Figure 2.11. PES for C_2-He collision (a) without splining and (b) with splining. [138]

using coupled cluster (CCSD/MRCCSD/CCSD(T)) or Multi-Reference configuration Interaction methods (MRCI), all of which scale at $\mathcal{O}(\alpha N^6)$ or worse. As discussed, α is the pre-factor, and N is the relative system size (roughly the number of orbitals).

2.4.3 N-Dimensional Fitting

To understand the problem of dimensionality from the eyes of a computational chemist, follow these steps:

- **Consider an atom**. Now think! How will you define its position in space?
- *Simple: Cartesian coordinates* $(\hat{i}, \hat{j}, \hat{k})$.

- **Now consider two atoms**. How will you define their positions in space?
- *For two atoms, 6 positional coordinates* $(\hat{i_1}, \hat{j_1}, \hat{k_1}$ *and* $\hat{i_2}, \hat{j_2}, \hat{k_2})$.

- But what if both atoms move, but their relative distance (say R) remains the same?
- **Then you need a direction vector** $\vec{R}$.
- *Does it mean anything? Where did this come from? Where did the* $\hat{i}, \hat{j}, \hat{k}$ *coordinates go?*

This may seem very rudimentary. But visualizing vectors and their projection on the three axes (Euclidean space) is a fundamental aspect of mathematics, closely tied to linear algebra, matrices, and ultimately quantum mechanics.

The vector $\vec{R}$ is the distance between the two coordinates $|\vec{r}_1 - \vec{r}_2| = \sqrt{(\hat{i_1} - \hat{i_2})^2 + (\hat{j_1} - \hat{j_2})^2 + (\hat{k_1} - \hat{k_2})^2}$. Look at the three Spherical coordinates (r, ϕ, θ) in Fig. 2.12. Each atom defined using Spherical coordinates can also be resolved in the Cartesian axis (x, y, z). To convert Spherical coordinate ϕ into Cartesian coordinates $(\hat{i}, \hat{j})$, we simply resolve it two times. Take a look:

- First we project vector $\vec{r}$ on z axis and $x - y$ plane.
- The angle θ originates from z axis. Therefore, $\vec{r}$ will be projected as $r \times cos(\theta)$ on the z axis.
- The angle θ will project $\vec{r}$ in $x - y$ plane as $r \times sin(\theta)$ (we will call this $\vec{P}_{xy}$).
- The projected $\vec{r}$ in $x - y$ plane i.e. $\vec{P}_{xy}$ makes angle ϕ with the y axis.
- Therefore, projecting the same on y axis will make $\vec{r}$ length $r \times sin(\theta) \times cos(\phi)$
- The projection of $\vec{r}$ on the last remaining axis (x) will therefore be $r \times sin(\theta) \times sin(\phi)$

If this is clear, let us move to our problem of dimensions for two atoms.

Atom/ vector	X	Y	Z
$\vec{C}$	$r * \sin(\theta) * \sin(\phi)$	$r * \sin(\theta) * \cos(\phi)$	$r * \cos(\theta)$

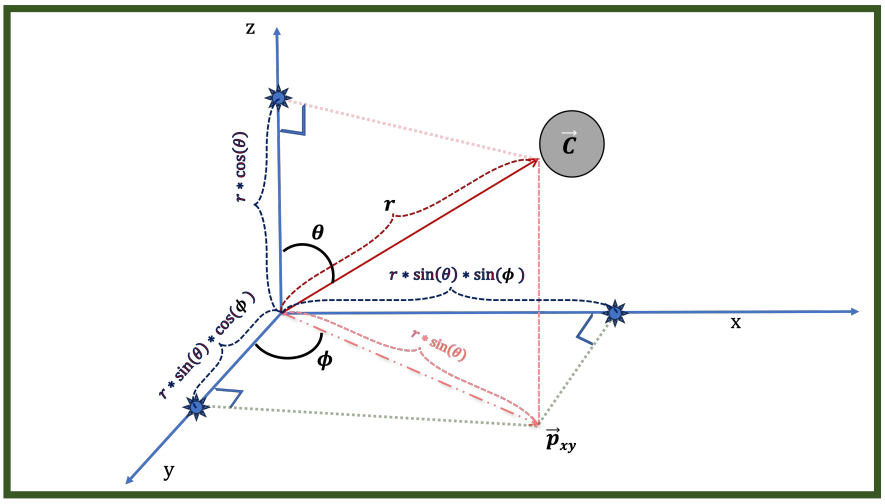


Figure 2.12. Image illustrating the projection of a carbon atom on XYZ axes, which can be considered a vector of length r creating angle θ and ϕ with z and y axes, respectively. † *Image generated using PowerPoint.*

If first atom (A1) is moved to coordinate $(0\hat{i}, 0\hat{j}, 0\hat{k})$ and second atom (B1) is placed at distance r on z axis i.e. $0\hat{i}, 0\hat{j}, r\hat{k}$. Doesn't matter how much the two atoms shift (equally) on the z axis. The distance between them will always be $r_{A1}\hat{k} - r_{B1}\hat{k}$ i.e. the $\vec{r}$ vector. This will hold even when the atoms are moved in x, y axes. But an important thing about the constraint (keeping the atoms on the z axis) is that the angle and dihedral become zero and we are left with a 1D system.

For three atoms, we will have a diatom (A2) and an atom (B1). Move A2's COM to $0\hat{i}, 0\hat{j}, r\hat{k}$ and force it to rotate in the $y-z$ plane. The incoming B1 will approach from the z-axis. Therefore, we only have two coordinates: R - distance between B1 and COM of A2; and θ - rotation of A2 across its COM in $y-z$ plane. The angle theta will not go from $0°-360°$. The rotation of A2 across the plane has reflection (mirror) symmetry, restricting θ to $180°$. If A2 is symmetric, θ will be restricted to just $90°$.

Later, we shall also see the symmetry in a 4-atom system (two rigid rotor collision) where the difference between two dihedral angles is constant, thereby reducing the two coordinates $(\phi_1 - \phi_2 = \phi)$ into one.

Conclusion?

Spherical coordinates enable us to determine the vector coordinates (and their symmetries) needed to describe any system. With more coordinates, more computations will be needed (to vary each coordinate) to build a PES. These coordinates can also be equivalent for certain orientations, depending on the symmetry of the system. For them, no extra calculations will be needed.

Jacobi Coordinates further simplifies spatial coordinates by describing the relative positions of the bodies rather than their absolute positions. Here, we also move the molecule of interest to its COM to allow for further simplification and use of spatial symmetries.

Cartesian coordinates are fixed points in space, and are therefore great for representing atoms on computers. Using Jacobi coordinates can cause issues (rare but annoying) like the occurrence of singularities (infinity) when dihedral angles approach $0°/180°$. PES2MP generates input files in xyz format by converting Jacobi coordinates into Cartesian coordinates.

Going from 3 atoms to 4 atoms!

In a crude manner, we can say that the maximum number of dimensions for a 4-atom system is $4*3=12$. However, using molecular properties like spatial symmetry and bond rigidity, some dimensions can be combined or neglected altogether, thereby reducing the number of dimensions and their span.

For example, if we consider a collision between a linear triatom (A3) and an atom (B1), we end up having a 2D collision (taking the triatom as a rigid rotor). The dimensions of the same will be R (distance between COM of A3 and B1) and θ (angle between A3's COM and B1). Considering full dimensionality (vibration), A3 has $3N-5=3*3-5=4$ vibrational modes (bending: horizontal and vertical; stretching: symmetric and asymmetric), therefore giving $4+2=6$ dimensional surface [140]. For low-energy (low-temperature) collisions, the stretching modes (having large frequencies) are usually rigid. While the same can be true for bending modes, sometimes they are just incredibly weak; for example, the C_3 molecule has a bending frequency of just 63 cm^{-1}. In these scenarios, bending mode (γ) and projection dihedral (ϕ) are usually considered along with R and θ, giving 4D surface. Such relaxed collisions are called rigid bender collisions.

To summarize, a rigid rotor (no matter how long the chain is) and an atom collision will always have a 2D PES; while a three-atom rigid bender will have 4D PES. For a linear molecule with more than 3 atoms, usually, the lowest frequency is considered for **bending** (given that other frequencies are also not as small). If the idea of converting vibrational bending modes into angular terms seems complicated, it's OK. It is complicated and one needs to spend some time to properly visualize the same. Finally, a relaxed surface (linear molecule with N atoms) will have $(2+(N-1))$D PES with additional **stretching** of bond distances (r_1, r_2..., and so on). These stretching frequencies are usually large enough to ignore.

How do I visualize Jacobi coordinates?

To be fair, the best way to visualize molecular N-body coordinates (Jacobi coordinates) is to draw them on a visualization tool like GaussView (or similar tools) and rotate them along multiple axes. But the only way to dominate them is to use pen and paper. Draw the system on paper. Determine its Jacobi coordinates. Resolve them into the xyz axis. Redraw the system again with different orientations and repeat the two steps. Some orientations will be easier to resolve than others. Some will give general solutions. Code these general solutions (resolved xyz analytical expressions and generate input files and visualize (GaussView, etc) to see if the result (xyz coordinates) matches the Jacobi coordinates.

Analytical expression for xyz coordinates for two diatom collision.

Now, we will take a detailed look at another example where 4 atoms are involved. However, the same are arranged in a diatom-diatom configuration and have 4 Dimensions even under rigid rotor approximation. For a simple diatom-diatom collision, we can have a total of 6 Dimensions. An example of the same is illustrated below in Fig. 2.13 for C_2-H_2 collision.

Let's work out the radial coordinates first. The distance between the COM of the two systems is R. The equilibrium bond distance between $C-C$ is r_{CC}. Since the molecule is symmetric, its COM will lie at the halfway point of r_{CC}, which we will refer to as r_C. Similarly, the half distance of the $H-H$ bond will be called r_H. The vibration of each diatom can be modeled by varying r_C and r_H.

The angular coordinates are θ and ϕ. The rotation of C_2 in the $y-z$ plane will be called θ_1, and the same for H_2 will be called θ_2. Similarly, their individual rotation in the $x-y$ plane (dihedral angle) can be called ϕ_1 and ϕ_2. ***But we took just one dihedral angle ϕ?***

Imagine if we rotate both θ_1 and θ_2 to 90° and keep both dihedral ϕ_1 and ϕ_2 to 0°. We will have orientation defined by Fig. 2.14 (a) as shown below. Then, in Fig. 2.14 (b), we will rotate both dihedrals

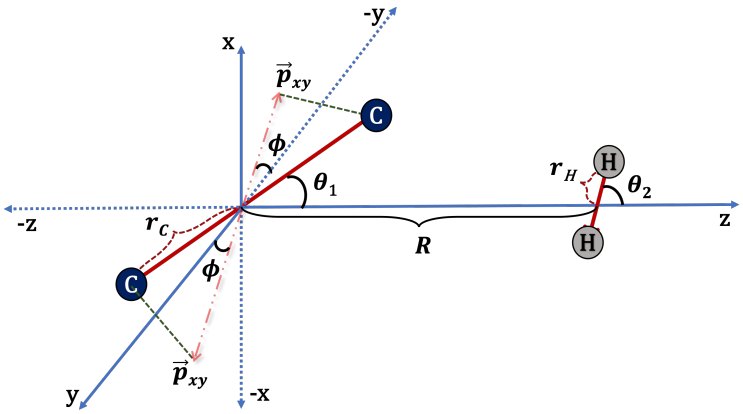


Figure 2.13. The six Jacobi coordinates for collision of C_2 and H_2 molecules. † *Image generated using PowerPoint.*

to 90°. The result is that both representations are equivalent and will have the same energy. Repeating these rotations will give us a symmetry, i.e., for two rigid rotors, the dihedral depends on effective rotation ($\phi_1 - \phi_2 = \phi$). So, we can simply keep one of the two rigid rotors fixed at $y-z$ axis ($\phi_2 = 0°$) and just rotate the other in $x-y$ axis ($\phi_1 = \phi$).

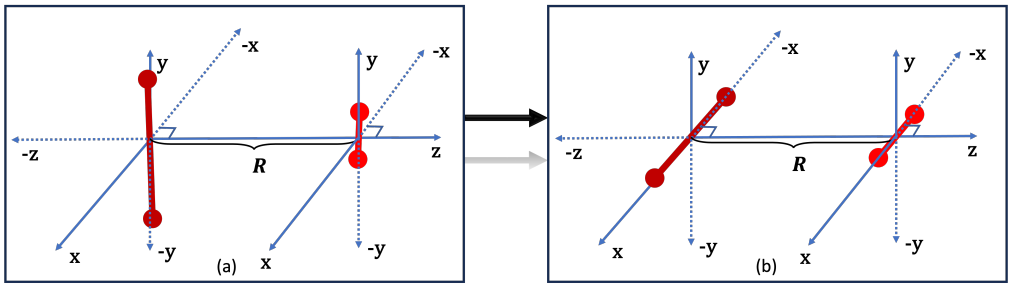


Figure 2.14. Image showing symmetric equivalence for two orientations with (a) $\phi_1 = \phi_2 = 0°$ and (b) $\phi_1 = \phi_2 = 90°$. The difference $\phi_1 - \phi_2 = 0°$ is the same in both representations. † *Image generated using PowerPoint.*

Now that we have defined all 6 Jacobi coordinates, the next step is to resolve them into xyz coordinates. The same can be described using a model in Fig. 2.15 showing the projection of C in all 3 axes (has θ_1 rotation in $y-z$ plane and ϕ rotation in $x-y$ plane) and both H as vectors at R distance from origin $(0,0,0)$.

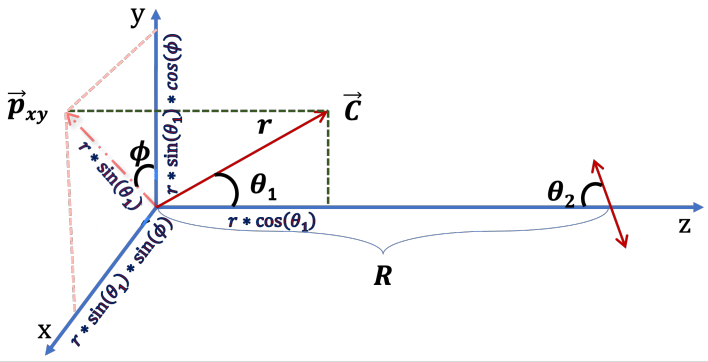
The C_2 molecule's COM is placed at the origin $(0,0,0)$. Therefore, the distance r_C of one of the carbon atoms will be in the +ive axes while the other will be pointing at the −ive axis. The rotation of C_2 will also follow this formula because of symmetry. Therefore, the projection of both carbons will only differ by a negative sign.

For the H_2 molecule, its y axis projection will be either projected in $+y$ or $-y$, thus resulting in a negative sign in the y-axis. Since we have kept its dihedral angle zero, there is no rotation in the $x-y$ plane, and its x component is zero. The final z axis will have a similar projection as the y-axis; however, since it is located at R distance in the z axis, the same will be added to both projections.

Regression Failure?

These six coordinates may look naive. But remember, how a simple 1D collision (an atom colliding with another atom) could be fitted into a suitable analytical expression, while finding an analytical expression

$$6D\ (R, \theta_1, \theta_2, \phi, r_C, r_H)$$



Atom/ vector	X	Y	Z
$\vec{C}$	$r_C * \sin(\theta_1) * \sin(\phi)$	$r_C * \sin(\theta_1) * \cos(\phi)$	$r_C * \cos(\theta_1)$
$\vec{C}$	$-r_C * \sin(\theta_1) * \sin(\phi)$	$-r_C * \sin(\theta_1) * \cos(\phi)$	$-r_C * \cos(\theta_1)$
$\vec{H}$	0.00	$r_H \sin(\theta_2)$	$R + r_H \cos(\theta_2)$
$\vec{H}$	0.00	$-r_H \sin(\theta_2)$	$R - r_H \cos(\theta_2)$

Figure 2.15. The XYZ projection of all six coordinates when two diatoms (C_2 and H_2) collide. † *Image generated using PowerPoint.*

becomes exponentially difficult thereafter (diatom-atom collision). If we consider rigid rotor approximation, both bonds (r_C and r_H) will be fixed, and we can work with $6-2=4$D PES. Under this approximation, any two linear molecules will only have a 4D PES. For a 4D (two rigid rotor) collision, the efforts to find a generalized function may be nearly futile. Though one may attempt to spline/analytically fit the function, the overall results can often have large errors unless equally complex and specialized functions are used.

This is the curse of dimensionality! And to find a solution, we must be ready for out-of-the-box thinking. The best way to explain why dimensionality is a problem is to visualize gravity. We have an intuition that exists for a three-dimensional world. And if we try to add (and visualize) just one extra dimension to the same, our intuition breaks down. The forces, distances, and interactions we understand in three dimensions no longer behave in a familiar way.

How can an extra dimension of space (the one related to time) be visualized?

The 4th dimension (i.e., time in spacetime coordinates) is the coordinate accounting for the sequence and duration of events. The 3 spatial dimensions (XYZ) are ideally **perpendicular** to each other in **Euclidean space**, but in spacetime, they are interconnected and can be curved by mass and energy. This curvature defines the path that nearby celestial bodies follow, a phenomenon we associate with gravity. Without **spacetime curvature**, there will be no change in the position of these bodies, rendering time meaningless. Therefore, spacetime has curvature; this curvature forces objects to move (giving them momentum), which we observe as gravity, and this act of motion is what we perceive as time.

The idea for this analogy is to make you aware that dimensions can be tricky. The simple way to visualize any system with a large number of dimensions is to take multiple 2D/3D snapshots of it and then spend time visualizing the same. Adding any dimension (such as r, taking into account molecular vibration) can be beneficial only if it affects the dynamics of our system.

However, visualization is not the only problem; we now have to solve equations that have become even more coupled and are, therefore, computationally expensive to solve. Sometimes, your system can have hundreds or thousands of variables (Dimensions), but in a desperate attempt to get some results, you may *(and I have)* run the calculations anyway, and now your computer has run out of memory or disk space, crashing altogether.

So what can be done?

Modern problems require modern solutions!

For any unsolved problem in science, the solution follows the trial and error method, and most importantly, *Failure!*. I will broadly talk about two scenarios that are common in computational chemistry.

(a) The molecule is small/medium size (or just doable). But I have to compute thousands/millions of data points (due to the large range/number of dimensions). It will take months. Suddenly, there is no space in the cluster (HPC), and now it will take Years. **Use supervised regression based NN model**. Just keep *ab initio* PES sparse with more data points where the potential gradient is large. The NN model will augment the surface and save months of computation time. In the PES2MP program, a PES-specific NN module (NNGen) is implemented to allow for *ab initio* PES augmentation.

(b) The system is too large. Each *ab initio* calculation will take days or will run out of memory for the CC method. In these cases, be innovative. Use HF-D4/TZ or CCSD-F12/DZ results. Test on a small model/fragments (where CCSD(T)/CBS results can be calculated) and then increase to medium-size fragments. **Train model with input coordinates and reference energy as input feature and CCSD(T)/CBS as output feature**. After training with smaller fragments, use the model to predict the results for medium-size fragments (should also have CCSD(T)/CBS results for getting residuals). Make sure that the model will work when scaled to your original system. Such an NN model will probably be tailored for a specific system (since a general-purpose program for the same can be hard to accomplish) but the idea is to generate the PES within spectroscopic accuracy while saving valuable computational time. If required, a general-purpose NN model is available in NNGen for testing the same. Be sure to trim high-energy regions (there is a keyword for the same) to improve accuracy.

Modern Methods

- Modern Supervised Learning: Gaussian Process (GP)
- Modern Supervised Learning: Neural Networks (NN)
- Classical Method: Using *for loop* taking a section of data (1D radial PES) over angular (can be multidimensional) terms.

The supervised machine learning models [141, 142] are indisputably the current champions in data interpolation and curve fitting. The NN methods [143] are designed to find trends in data with millions/billions of data points (with any number of dimensions). The GP method [144] is based on a probabilistic approach (having a Gaussian distribution) to predict the output (which is seen as an infinite-dimensional generalization of multivariate normal distributions). Both methods are discussed in detail in the upcoming sections.

The other method (using a for loop over each radial term) can be seen as an example of applying brute force to get optimum fits over a large sample of data. Though this may be inappropriate for most cases, it has its own advantage in fitting PES. In this approach, the radial term of the PES (R) is taken to be the fixed input parameter. For each angular combination (can vary from 10 for 2D to 25,000 for 4D collision), the output potential (V) is fitted into a series of Slater functions ($\Sigma_i \alpha_i e^{-\beta_i R}$). The α_i coefficients are varied

for each angular term, while the β_i coefficients are kept constant (after being optimized to give correct well depth and shape at minima and long-range regions). The accuracy of this result can be verified from a residual plot. This method has a limitation: it cannot augment angles. However, it has another significant advantage. After multipole expansion, the radial terms are still a function of R (also suggested by the name). This makes it possible to use the same Slater function with the same β_i coefficients to fit the radial terms without any error. The same has been implemented into the code (see *2_FnFit* method in *GUI_examples* Section 4.2.2) as the resulting radial terms must be analytically expressed in order to be used in MOLSCAT.

2.5 Machine Learning

Machine Learning (ML) is not new. It's an old [143, 144] technique with even older foundations that date back by decades or even centuries. But it has received its popularity through applications in various areas of scientific research and is making its way into quantum chemistry [145–158]. Today, we know its application as artificial intelligence (AI) chat-bots. Some years earlier, it also revolutionized cameras through its powerful image-processing capabilities.

What computers do efficiently is that they can repeat a set of instructions at speeds that can surpass any human capacity. What they lack is the ability to learn from mistakes or adapt while they are doing such calculations. The general design by which a machine can learn/adapt is implemented into Artificial Neural Networks (taking inspiration from biological neurons) and Bayesian Learning (Worked out independently first by Bayes and then by Laplace).

The latter requires making a guess first and adapting later in the light of new data. This reinforces the machine to make adjustments required to incorporate the new information. Computationally, this method is quite expensive. Its most popular implementation is known as the Gaussian Process, and we shall use it to tune our hyper-parameters (which number in the dozens) rather than using it to learn the PES, which can have data points in the millions.

What is Machine Learning going to do for me, a Computational/Theoretical Chemist?

In chemistry, solving the Schrödinger's equation numerically takes enormous memory and time. The most popular method, DFT, scales with $\mathcal{O}(n^3)$, but the error it bears is in order of kcal/mol (chemical accuracy). This is good enough for studying organic reactions, but not cold and ultracold chemistry, where the error must be of order cm^{-1} (spectroscopic accuracy) [69]. This is $\backsim 350$ times higher accuracy. And methods required to solve these (coupled cluster (CC) or configuration interaction (CI)), at best, have a scaling of $\mathcal{O}(n^6)$. What follows is that a calculation that may seem trivial with DFT can become a year-long computation with CC and CI.

However, there is also significant progress in reducing the computational time, where explicitly correlated methods called F12 are developed. A CCSD calculation with F12 approximation (which requires benchmarking with different basis sets) can acquire similar accuracy with a smaller basis set. It can reduce computation time significantly, given that aug-cc-pV5Z is exponentially more expensive than aug-cc-pVTZ and can even be deemed impractical for large/heavy molecules. Such approximations, along with the fact that personal computers nowadays can have workstation-grade processors and ECC (error correction code) memory, have made it possible to compute dense surfaces with ease. But as the dimensionality of the problem increases, one can still end up in the same conundrum.

So, before applying ML to create models to make data denser, we shall look into detail, how dimensionality increases and how error scales. A well-motivated study requires an understanding of the error it can bear

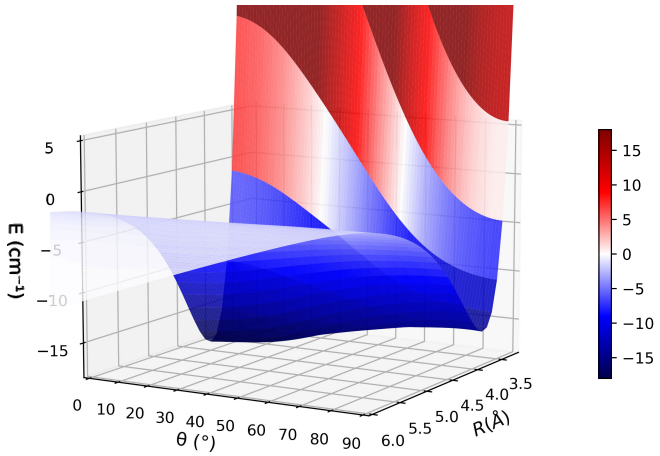


Figure 2.16. PES of He collision with C_2. An NN model is used to augment the angle from $\Delta\theta = 15°$ to $\Delta\theta = 1°$, giving dense PES. † *Plotted using Origin.*

before the results turn to garbage. ML methods are no different. But before understanding the ML part, let us take a brief overview to go from a 2D to a 4D problem.

2.5.1 NN in Fitting PES

First, let us properly visualize the molecule-atom collision PES, one we saw in the 2D spline. In our view of 3D space, we can have translations, rotations, and vibrations. Usually, chemistry deals with temperatures at which molecules are in their lowest vibrational state. This is true even at room temperature. So, at cold and ultracold temperatures, we can assume our molecules to be rigid rotors. Remember, the ground vibrational state does not have zero energy, so the molecules do vibrate in reality, but we shall discuss this once we go to 6 dimensions. So, our first approximation is to eliminate vibrations, which will enable the study of pure rotational transitions originating from this collision.

For this, all possible orientations (in our $360°$ of space) must be considered. What comes to our rescue first is the spatial symmetry of molecules. Due to symmetry, several features of the PES will be mirror images across a certain point or plane. Remember, in 2D fitting, the calculations were done till $90°$. The PES from $90°-180°$ will be a mirror image of PES from $0°-90°$, and PES from $180°-360°$ will be a mirror image of $0°-180°$. In the end, we are left with two coordinates, R (distance between the center of mass (COM) of two colliders) and θ (the angle at which He is approaching the COM of the rigid rotor). The spatial symmetry will reduce the number of points to a $1/4^{\text{th}}$ value. So, a surface that takes a month can be generated in a week. We can use ML similar to how we splined the angles and get an augmented surface that is smooth as well. An example of a smooth augmented PES is shown in Fig. 2.16.

Let us see the PES for colliding a linear rigid rotor, NCCN, with H_2. For H_2 collision, we have a 4-dimensional surface, i.e. the PES has a dependence on 4 Jacobi coordinates (R, θ_1, θ_2, and ϕ) similar to Fig. 2.13. The R, θ_1 are analogues to what 2D coordinates represent, and the θ_2 describes the rotation of H_2 rather than NCCN. The spatial symmetry allows us to eliminate one of the dihedrals by keeping that rotor fixed in one of the planes. The other molecule is free to rotate in that plane, and that will be referred to as ϕ.

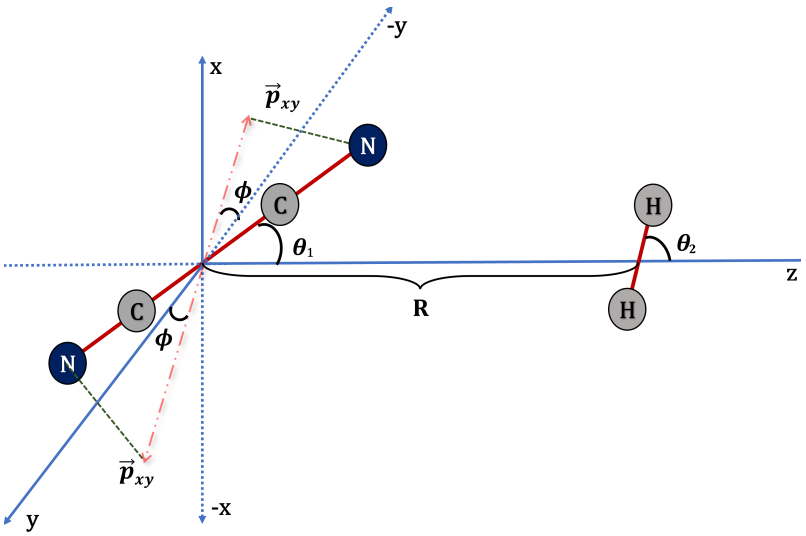


Figure 2.17. The four Jacobi coordinates describe the collision of two rigid rotors (NCCN and H_2) [159].

Table 2.6. The number of points in each coordinate to get a medium quality (sparse) vs high quality (dense) 4D PES.

		Sparse PES		Dense PES	
Parameter	**Range**	**Step-size**	**Points**	**Step-size**	**Points**
ϕ	0° to 90°	30°	4	5°	19
θ_2	0° to 90°	15°	7	5°	19
θ_1	0° to 180°	15°	13	5°	37
R	2.5 Å to 20 Å	variable	65	0.1 Å	176

Table 2.7. Table comparing NN method to *ab initio* computation time.

Ab initio calculations	**Stats. [Molpro CCSD(T)/AVQZ (CP)]**
Angular Terms	= 13 * 7 * 4 = 364
Total points	= 364 * 65 = ∽20,000 points
Total time	∽3200 days (with 1 processor; taking 1 pt. = 4 hrs.)
Time [40 Procs.] @ $4(T) \times 10(N)$ #	∽3 months
Machine Learning Model	**Neural Networks**
Angular Terms after augmenting	= 19 * 19 * 37 = 13357
Total points after augmenting	∽2,300,000
Equivalent *ab initio* time	∽1000 yrs. [1 Proc.] | ∽25 yrs. [40 Procs.]
NN training time (80% data points)	4-12 hrs.

T = Thread (or processor) count for each calculation.
CCSD(T) calculations don't scale properly for more than 4 threads.
N = Number of simultaneous calculations on different nodes in a cluster.

Now that we know spatial symmetry reduces our computational effort, we will see NCCN-H_2 collision as an example of how dimensionality becomes a bottleneck. In Table 2.6, a medium-quality surface has an angular step size of $15° - 30°$ and a variable radial step size with more data points in the minima region, resulting in $\backsim 20k - 25k$ *ab initio* data points. Taking a small step size of 5° and 0.1Å throughout the range of Jacobi coordinates can result in more than a million data points, which is computationally infeasible.

For creating a medium-quality surface, $\backsim 3200$ days of computational resources are needed, assuming a relatively new 3-4 GHz processor is used with SSD storage for temporary files and DDR4 memory. $*$ *For*

someone unfamiliar with these terms, a quick recap can be obtained from Sec. 2.3.1.

Since CC methods do not scale well above 4 processors, running 10 jobs with 4 processors each can bring down the time to $\backsim$ 90 days (ideally). However, running multiple jobs on the same unit (node of HPC) directly affects the time as they may share the same disk and memory. Remember that large integrals stored in disks are often the most time-consuming aspect of a CC calculation. Overheating is yet another issue, and a more realistic time frame for the same calculation will be $1.5-2\times$, i.e. $4-6$ months. *But there is still hope!* Using the F12 approximation can bring the time down to 0.5-1.5 hrs, as the basis set reduces to AVTZ. Now the problem seems more manageable as one can utilize a more realistic 40 processors to get the same surface in $\backsim$ 30 days.

But look at what happens if we decrease the step size and go for a dense, high-quality grid (at 5° and 0.1 Å interval). We'll have $19*19*37*176 =\backsim 2.3$ million data points. These calculations are impossible *ab initio*, and therefore NN model can be utilized to produce such large data. A sparse to medium-sized grid can be enough to augment a 4D surface, and by design, it can handle millions of data points without compromising much on time or quality. The only thing one must remember is to ensure the data quality is good (input data must not have non-converged poor data points that can cause problems in learning the surface), and a compatible model (neither too deep nor too shallow) must be used to create the model.

2.5.2 Artificial Neural Networks

A neural network begins as a single unit called a perceptron [143]. It draws inspiration from biological neurons and is therefore termed Artificial Neural Networks (ANNs) or simply Neural Networks (NNs). When Several units combine into a layer, and that layer is connected to an input and output, it becomes something called a hidden layer. There can be many hidden layers, and every unit or neuron in a layer is connected to every other unit in the system, just like a biological neuron. When we have a single hidden layer, it is usually (but not necessarily) known as a shallow NN, and when we use multiple hidden layers, it becomes a deep NN (DNN).

Neural Networks (NN)

Similar to biological neurons, the NN can tackle both classification and regression problems. This is achieved by giving each layer an activation function. This function controls weights in each cell/unit to best mimic the results during training. Some of the functions that we'll come across are given below with their plots to show why some activation functions will perform better than others.

The NN model is tuned by doing forward and backward propagation [160]. In the forward propagation, feature values are multiplied by weights (and bias is added if any) to each neuron, as shown below.

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \tag{2.18}$$

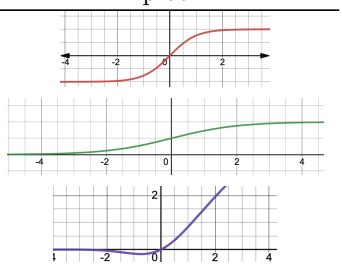
where β_0 = bias, β_i = weights, and x_i = input features. This weighted sum (z) is then passed through an activation function (a) to produce the output. A list of relevant activation functions is given in Table 2.8.

Another crucial step is back-propagation, where optimal parameters for the model are determined using the Chain rule of Calculus.

$$\frac{\partial L}{\partial w} = \frac{\partial z}{\partial w} \times \frac{\partial a}{\partial z} \times \frac{\partial L}{\partial a} \tag{2.19}$$

where L - loss function, w - weights, z - linear regression, and a - activation function used, e.g., tanh.

Table 2.8. Various activation functions important for regression NN and their plots

Activation function	Formula	plot
$\tanh(x)$	$\frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	
Sigmoid	$\frac{1}{(1+e^{-x})}$	
GELU	$0.5x\left(1+\tanh\left[\sqrt{2/\pi}\left(x+0.044715x^3\right)\right]\right)$	

This very efficient methodology gives NN the power to fit data with millions of data points, even on old computers with limited memory. When implemented on GPUs, which may not be as versatile or general-purpose as CPUs, the specialized cores can train the models with enormous efficiency. Since Keras (and its base library TensorFlow) already has a GPU implementation, end users who are familiar with Python and scripting can benefit from it.

Getting good hyper-parameters, as we discussed, is very important to prevent under-fitting or over-fitting of data. This is where Bayesian Learning (specifically the Gaussian Process) comes into the picture, which we shall discuss in detail in the upcoming subsection.

2.5.3 Bayes' Theorem

Bayes' theorem begins with the story of Bayesian inference, which provides a solution to inverse probability, whereby one systematically updates a hypothesis in the light of new evidence. It was published by Richard Price after the death of Thomas Bayes in the mid 18$^{\text{th}}$ century. However, there was another popular mathematician, Pierre-Simon Laplace, who gave us the current formalism and extended the work to what we currently know as the Bayesian interpretation of probability.

In simple words, if you know certain conditions to be true (a few X/Y data), and you make a guess based on them, its probability distribution may be very large, making the results look bad. But repeat it with more and more (new) data, and over time, your guess is as good as it can be. This forms the base for many statistical studies we do today, and it has found its implementation in the field of machine learning as well.

Gaussian Process (GP)

A Gaussian Process (GP)[154] is an ML method based on Bayesian inference whereby we start with a guess (for multiple variables) having a joint Gaussian distribution [144].

The mean and covariance of a Gaussian distribution are vectors and matrices, respectively:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

where, a = height of the curve's peak, b = the center position of the peak, c = the standard deviation, and e = Euler's number.

The difference between the Gaussian distribution and the Gaussian process is that the former is over vectors while the latter is defined over functions and is given by:

$$f \backsim GP(m,k) \tag{2.20}$$

where the function f has a GP distribution with a mean function m and a covariance function k.

For the Bayesian approach [161], this GP will be utilized as a prior. The prior does not rely on the training data, but it does specify some of the functions' attributes. This prior is updated in light of the training data [162] to make predictions for unseen test cases as given below:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma' \\ \Sigma'^{\top} & \Sigma'' \end{bmatrix}\right) \tag{2.21}$$

where $\mu = m(x_i),\ i = 1, ..., n$ stands for the training means and analogously μ_* stands for the test means. Σ stands for training set covariances, Σ' for the training-test set covariances, and Σ'' for test set covariances.

The NNGen module uses the Keras-Tuner library, with a custom implementation of the 'BayesianOptimization' tuning with the Gaussian process. The custom implementation allows the program to use the 'tqdm' library for visualizing the optimization process and allowing custom parameters for selecting the best model architecture for any given dataset.

2.5.4 Limitations

GP Model: Directly using GP to learn PES can be very expensive. A good implementation of the same using the GPy library can be found in PESLearn [147]. The method starts to become expensive around 1000 data points and as the training size increases to more than 5000, the training time becomes computationally impractical.

NN Model: At the end of the day, NN models are heavily parameterized interpolation methods, and are therefore not good for extrapolating. The data beyond the boundary of the input dataset will not be reliable and hence must be fitted analytically if needed. The general-purpose ensemble NN model provided in PES2MP is only meant to interpolate data points. The radial extrapolation implemented in the PES-specific NN model is described later in the chapter. Also, very deep NN models can easily overfit data, and therefore hyperparameters must be optimized judiciously.

2.5.5 Application and Deep Learning

Before we begin to dive deep into codes and algorithms, one must remember the practical and theoretical limits of any program/library one is using. For NN, we can have very large data, and it will be hard to overwhelm the fast implementation of TensorFlow. But what can affect our model is data quality and hyperparameters. Making sure these two things are in order will help create models that can even fit very sparse datasets.

In the paper by Kushwaha *et al.* [138], $\backsim$ 125 data points were chosen from the 2D PESs of C_2-He (9100 data points at 1° interval) and NCCN-He ($\backsim$ 1000 data points at 15° interval) collision. While the C_2-He PES had a mild anisotropy (slope of the PES), the NCCN showed strong anisotropy due to a larger potential well and longer chain length. The data points were kept sparse to the absolute limit before the interpolation failed. The original data points were recreated using cubic spline [163] and two ML methods (NN and GP models) using the PESLearn package. The interpolated and exact datasets were then used for calculating rotational dynamics and the errors for the same were evaluated. The results found large deviations (even 100% deviation) in the splined dataset compared to the exact results. While the GP model was substantially better than the splined data, it was slower and was outperformed by the NN model.

Another paper by Biswas *et al.* [164] showed that two NN models with different error profiles can be used to study such collisional dynamics. There are qualitative differences in the results, however, the deviations in the NN models do not change the order of the rate coefficients. Also, the rates/cross-sections for certain transitions are found to be indistinguishable for the two NN models.

Another paper by Roman Krems [165] uses GP as a Bayesian neural network model (NNGP) for learning the high-dimensional PES of polyatomic molecules. The authors demonstrate that NNGPs combine the fast scaling of NN with the uncertainty quantification of Gaussian processes to significantly reduce the computational cost compared to traditional GP kernels while maintaining high accuracy.

To conclude, the NN models can be used in ingenious ways to tackle chemical problems involving large dimensionality and data analysis. Tuning hyper-parameters can be tricky, and random choices can be poor. In PES2MP, the GP method has been implemented for hyperparameter optimization that searches for a suitable NN architecture (neither too deep nor too shallow) that can be free from under/over-fitting.

2.5.6 Data Quality and Transformation

Suppose PES data points are missing near the asymptotic limit, where we know the potential has a gradual slope. This will not affect our model as adversely as missing data points near the minima region (given that we control overfitting). So, make sure to include a higher concentration of data points where the potential slope is steep. Also, remember that wrong data points are worse than missing data.

How?

If there are non-converged data points, they may pose a threat that there are kinks and grooves where data must be smooth. This is something that will either force the model to under-fit to avoid these points or over-fit to incorporate these points (along with minima) in the trained model. Overall, ***a poorly sampled data is as good as no data.***

Therefore, determine that the data is free from kinks (non-converged potentials), has a finer grid near the energy minimum than the asymptotic region, and has high energy points trimmed. A feature of the NNGen module is the separation of high-energy points from minima and asymptotic energy data. Trying to learn both together in one model will result in missing features at the minimum region.

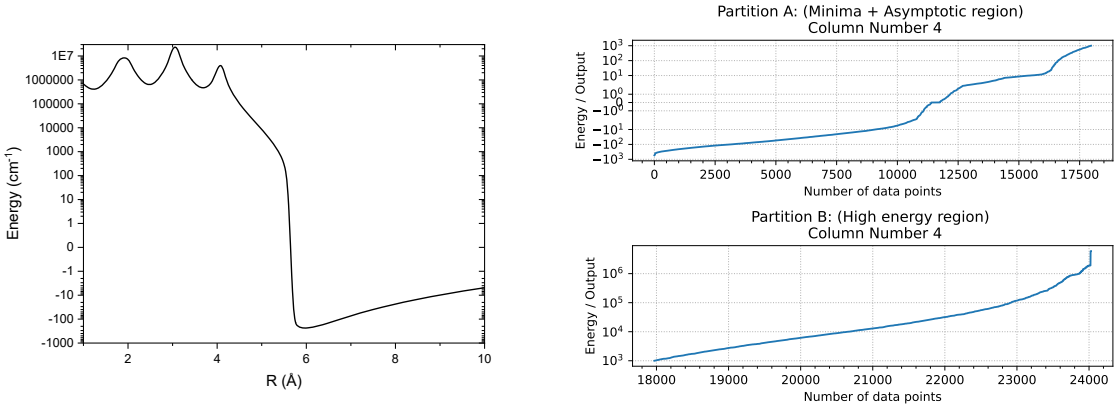
High energy trimming:

The trimming of the high energy region in PES2MP's FnFit module is done by identifying the global minimum at each angle (or angular combination for 4D PES) and then scanning backward for the high energy trim value (say 10^3 cm^{-1}). In the NNGen module, the high-energy region is trimmed for the whole dataset by identifying data points that have energy higher than the trim value.

For example, in Fig. 2.18 (a), the PES has a global minimum around 6 Å, while other high-energy minima arise when the collider enters inside the rigid rotor to reach COM ($R = 0$ Å). This problem is common when the rigid rotor is very long and the collision is head-on. The program removes such oscillations when it trims the high-energy region, however, the users are encouraged to check for *NaN* and $-\infty$ values at high-energy regions. Fig. 2.18 (b) shows the separation of these regions in the two subplots, with the y axis showing the potential energy and x axis showing the position of data points (in ascending order of energy).

Why check the minima of the partition plot?

Suppose a point fails to converge at a short R distance (< 5 Å in Fig. 2.18 (a)) for the head-on collision but still gives some garbage value in Hartree. If this value is $+\infty$ when converted to cm^{-1}, it is removed by



(a) Symmetric-log plot for head-on collision of long rigid rotor

(b) Partitioning high-energy region

Figure 2.18. Plot (a) represents PES for head-on collision for a collider (e.g. $\theta = 0°$ for He) with a very long RR. Since the collider is reaching the COM of the RR, it encounters several minima when the collider travels between the atoms. * Ignore the skewed shape of the PES at minima, which is due to a symmetric-log plot of energies. Plot (b) shows NNGen separated high-energy (lower) and minima region (upper).

the program when the high-energy region is trimmed. However, if this value becomes $-\infty$ (for e.g. -10^8 cm^{-1}), the program can treat the same as the actual minima and include it into the minima region. Such non-converged negative values can be identified using Fig. 2.18 (b). ***How?*** The global minimum of the PES, i.e., -10^3 cm^{-1} at 0^{th} position will be replaced by -10^8 cm^{-1}, which is too large for a Van der Waals collision.

How to interpret the partition plot to identify the above anomaly?

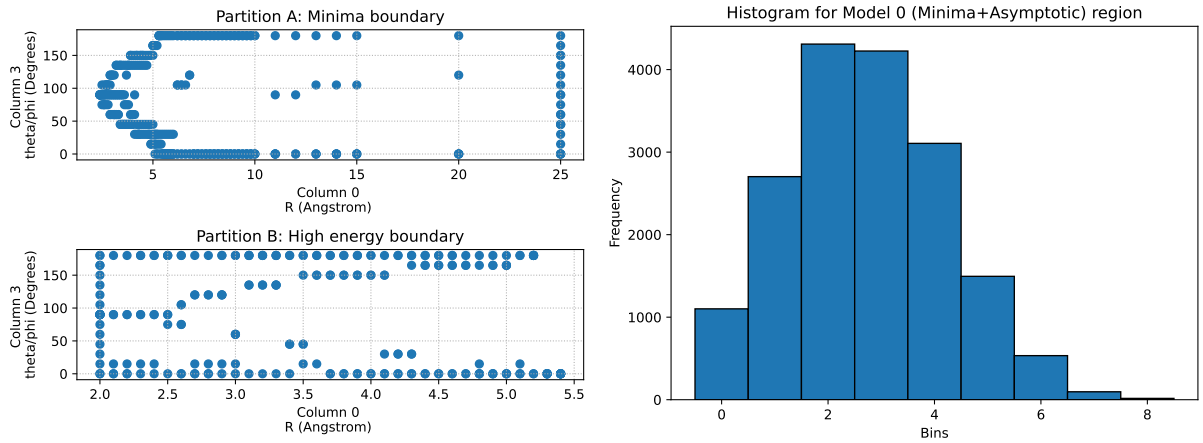
The upper panel of 2.18 (b) represents the sorted energies in the minima region, while the lower panel represents the high-energy region. The dataset energies are sorted in an increasing order, and the x axis represents the number of data points. The plot begins at minima (which is between -10^2 and -10^3), and since the high energy cutoff is provided by the user, the plot ends at 10^3, and the number of data points in the minima region is slightly more than $17,500$.

Similarly, the lower plot is the high-energy region (that is discarded). It begins at the high energy cutoff, i.e., 10^3, and ends at whatever is the maximum in the PES dataset (which is $\backsim 10^6$ in the plot).

I would recommend the users to always spend some time plotting multi-dimensional PES (which has been converted to cm^{-1}) to find any anomalies resulting from non-converged data and remove them manually before proceeding with NN fit or any other dynamical calculation. With ab initio methods, there is a high chance that some data points will not converge or converge to the wrong values.

Binning and Boundary Separation (Distributing training, testing, and validation datasets):

The NNGen module can also separate boundary elements from non-boundary elements as shown in Fig. 2.19 (a). Boundary elements are data points where input coordinates $(R, \theta(s)$ represent the physical boundary of data (e.g., initial/final θ/R values such as $\theta = 0°$, $R = 100$ Å, etc.). Missing boundaries can make the model predict poorly for the edges, as it has no idea what lies ahead. The separated boundary data points are always kept in the training dataset, which preserves these boundary features in the trained model. For the boundary plot, PES2MP keeps the first column (here R) fixed and varies the rest of the input coordinates $(\theta_n, \phi_n, r_n$, etc) for plots resulting in $N-1$ plots for boundaries where N is the number of input coordinates. Therefore, the 2D PES has one boundary plot, while the 4D PES has three.



(a) Partitioning high-energy region

(b) Binning input coordinates

Figure 2.19. Plot (a) represents the partitioning of boundary elements (R and ϕ) for separated minima (upper panel) and high-energy (lower panel) regions. Plot (b) shows the number of data points separated into various bins based on similar coordinates to allow for a better dataset split (into training, validation, and testing datasets). Both plots represent the output for a 4D PES.

To make sure that the data distribution is even based on our input/output, binning (stratification) is implemented. Users can choose whether to bin the data using input or output coordinates (e.g. Energies, Dipole, etc). This process separates data into training, testing, and validation datasets by grouping (binning) continuous data into discrete intervals. Overall, this allows for proper distributions of data (from all regions: attractive, repulsive, and asymptotic) into the three datasets. This will prevent situations where some shallow minima region has no points in the training dataset, or when no asymptotic data points are present in validation datasets, etc. An example plot showing the stratification of 4D PES into various bins is shown in Fig. 2.19 (b). The discretization of continuous energies or coordinates results in some bins having more data points than others. The program ensures that data in these bins (which represent regions with fewer or more points) gets distributed evenly across the three datasets.

* The program implements binning (or simple random distribution) for only dividing the non-boundary elements of PES into training, testing, and validation datasets. The program then combines the boundary elements into the above-generated training dataset. *Therefore, the boundary elements are always present in the training dataset.*

Finally, data standardization and normalization are necessary transformations for the input dataset. Use output data in an acceptable limit (always use cm^{-1} for spectroscopic accuracy and kcal/mol for chemical accuracy). In PES2MP, the data scaling to unit variance is true, but data centering with mean is set to false. This prevents features like R from having negative values and also allows potential V to have meaning to its negative and positive values. While the centering feature improves convergence, it also disturbs the distribution of positive and negative values, creating problems for the custom function that we use with R for PES-specific models. *User can enable data centering by editing 'pes2mp.py' for using the generic NN model.*

The NNGen module encourages users to convert data if needed, choose bins, and preview the data distribution by boundary and bins. This will ensure a good fit, and along with data standardization/normalization, the ensemble NN model will take care of the rest.

2.5.7 Hyper-parameter Tuning

The final important thing to remember is the optimal values of hyperparameters. Providing bad choices for NN architecture or training epochs can make the NN model perform worse than the cubic spline. We shall discuss how each hyperparameter in the NN model is used and then define the sub-space for the GP method to optimize.

- **Train-test-validation split ratio**: Before the NN model can be built, the original dataset must be split into training and testing datasets. The training dataset is the one the NN model uses for its training. Once it has trained, it will check the errors w.r.t. test dataset (it has not used these points for training the model). If the model has been under-fitted, it will show large errors for both the training dataset and also the test dataset. However, if the data has been overfitted, it will show a very small error for the training dataset (the one it sees while training) but a very high error for the test dataset (the unknown coordinates). This can be avoided by using a validation dataset. The NN model will still be trained using the training dataset, but after each epoch (iteration), the model will evaluate the performance of the NN model based on the validation dataset. Over time, the NN model will become biased towards learning the validation dataset (while the model tries to bring the validation errors down). Therefore, at the end of the learning phase, the NN model will likely have moderate or comparable errors for the test dataset. This is not a guarantee, but validation datasets can help with overfitting. The presence of this dataset is also beneficial while the hyperparameters are tuned using libraries like keras-tuner. The ideal split for the train-test-validation dataset split depends on the availability of data:
  - 20:40:40 split: If the input dataset is very very large (dense coordinates) [not our case].
  - 50:40:10 split: If the input dataset is neither too large (dense) nor too small (sparse).
  - 80:10:10 split: If the input dataset is small (sparse coordinates).
  - 90:05:05 split: If the input dataset is very small (very sparse coordinates).
- **Optimizer**: Usually, Adam (Adaptive Moment Estimation), Nadam (Adam with Nesterov momentum), or SGD (Stochastic Gradient Descent) optimizers are well suited for our purpose. The information about other optimizers can be found here at geeksforgeeks.org.
- **Activation function**: If the shape of the activation function is similar to (mimics) the shape of the output feature (linear/non-linear) we wish to learn, the training time can be reduced with lower errors. The *tanh(x)*, *softsign*, *gelu*, and *mish* are a few activation functions worth exploring here at keras.io.
- **Number of hidden layers in the NN model and Number of activation units (nodes) in each layer**: A large number of layers will make the model deep and capable of learning complex features. However, this can also cause the model to overfit and produce artificial features for unseen coordinates. A shallow model with a large number of nodes can fix this problem. Therefore, a balance between the two is important for a balanced model. The keras-tuner library is utilized for finding this balance.
- **Batch size and Number of iterations in training NN model**: The user can send training data in an NN model's forward/backward pass (epoch) in full or small user-specified batches (mini-batch). This makes NN training memory efficient and faster. However, a very small batch size can result in large fluctuations. The number of iterations for each batch size should be large enough to make sure that the error reduces to the desired limit. Once the NN model goes through the entire dataset (batch), an epoch is completed. Since PES datasets are usually sparse, PES2MP uses the full training dataset as a batch (no mini-batches).

There may be a dozen more hyperparameters that are more specific to the NN model being built. The general idea is that the search for optimal weights and biases depends entirely on these hyperparameters and your original data. So, it may be beneficial to spend some time learning about the same.

2.5.8 PES2MP NN Architecture

The PES2MP offers two choices for the NN model.

- Generic model: Simple **Nodes** and **Layers** (along with **Branches** that later merge)
- N-Dimensional PES model: Generic model + **R Constraint**

The architecture of the N-Dimensional PES model remains the same as the generic model; however, the penultimate layer (concatenate) is multiplied by a custom trainable function acting on R (to ensure the correct shape of the PES), and the bias is turned off (to ensure 0 value at infinite R). Both models will generate multiple NN models based on input split information and create an ensemble model from the same to remove errors that can occur from a singly trained model. The NN architectures of the two models are shown in Fig. 2.20. The model has 3 tunable hyperparameters that are searched using the keras-tuner's 'BayesianOptimization' function. The search space consists of: The deep_l hyperparameter controls the number of units in each hidden layer. The hidd_l controls the hidden layers, while the bran_l controls the number of branches. For example, the generic model has two sets of branches that use the 'GELU' activation function and are concatenated (except the last layer). Similarly, in the ND_PES model, alternating 'Gaussian' and 'NGELU' activation functions are used. Remember: Due to the architectural design, the branches must be provided in multiples of two. The search space can be expanded by including new values in the above variables. The number of trial searches can also be increased using the 'max_trials' parameter in the NNGen input file.

The architecture's search space can be provided in the NNGen input file for two NN functions ('create generic model' and 'create ND model' in the 'PES2MP_driver.py'). It would be a good idea to allow the model to train on several architectures using an 80:05 (Training: Testing) dataset split. Then, the best architecture could be entered into the input file as shown below to maximize the ensemble model's accuracy.

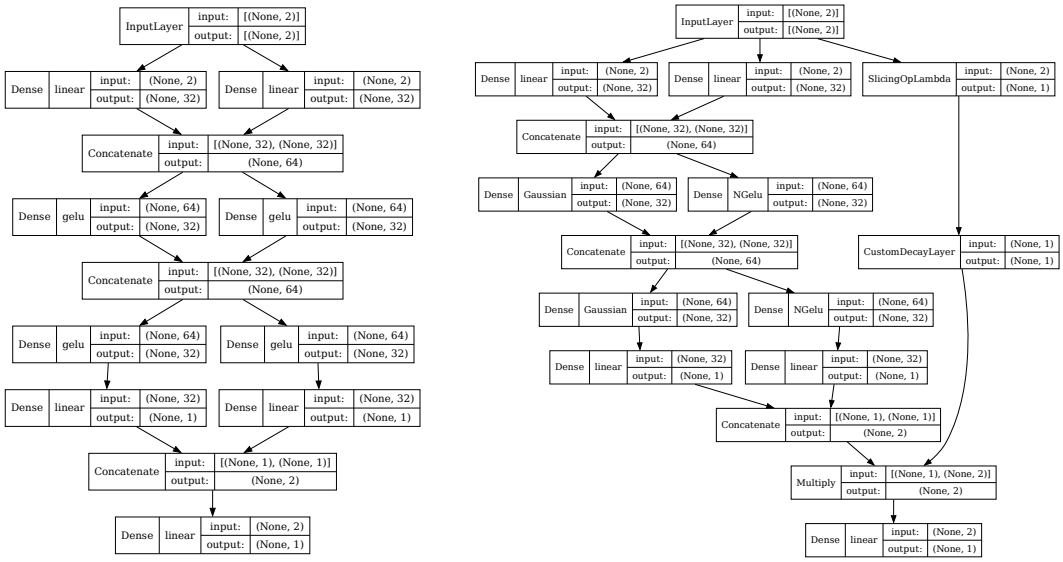
```
NN_hyperpara = {
    'Max_trial'   : 10,          # number of trials for architecture search
    'NN_nodes'    : [64],        # search space for NN nodes per layer
    'NN_layers'   : [4],         # search space for NN layers
    'NN_branches' : [4],         # search space for NN Branches: even only
    ...
```

Listing 2.3. Using custom architecture for NN model in PES2MP

Both models can handle multiple inputs and outputs simultaneously. The generic model does not have extrapolation capabilities, resulting in poor description at a very short range, but is useful for non-collisional PES data. The Generic model also has the bias enabled for learning adiabatic PES and other properties, such as dipole, etc. In the N-Dimensional PES model, the bias is turned off to let the model reach '0' at

Table 2.9. Comparing the two NN models available in PES2MP

	Generic model	**N-Dimensional PES model**
Activation Functions	GELU	Custom (Gaussian and NGELU)
Activation Bias	On	Off
R Constraint	None	Modified Slater Decay



(a) Generic (b) ND_PES

Figure 2.20. NN architectures for the (a) generic and (b) N-Dimensional PES model available in PES2MP. The example considers a 2-dimensional input (R, θ) and a 32, 2, and 2 configuration for units, hidden layers, and branches, respectively. The 'CustomDecayLayer' is the function of R (obtained by using 'SlicingOpLambda') that forces the model to have correct extrapolation values at small R.

infinite R. Enabling bias results in small deviations at the asymptotic region. The two custom activation functions are also chosen to allow for smooth angular augmentation and faster convergence.

The constraint on R is a modified Slayer function, as stated in Table 2.9. The function decays from infinity at short x to zero at asymptotic x, i.e., (e^{-x}/x). Optionally, it can have a trainable minimum when $e^{-x}\log(x)$ is used, but it can also easily result in overfitting (to enable and test, see $def\ create\_CustomDecayLayer()$), which is explained below.

The function has non-negative constraints on a, b, and c, which forces the function to retain the shape (and not flip), which is not possible when using summation functions. The coupled function, therefore, ensures smooth convergence without unwanted features (for small R extrapolation) compared to the sum of Slater $(\sum_i a_i e^{-b_i x})$ and/or inverse x functions $(\sum_i a_i x^{-b})$. The N-Dimensional PES model uses two custom activation functions of the form:

- Gaussian : $\beta e^{-\alpha(\mathbf{X}+\gamma)^2}$, and
- NGELU : $\beta \times \frac{-1(\mathbf{X}+\gamma)}{2} \times [1 + \frac{\mathrm{erf}(-1(\mathbf{X}+\gamma))}{(\alpha \times \sqrt{2})}]$

where α, β,and γ are coefficients and $\mathbf{X}$ denote the input coordinates (R, θ_n). The functions can be visualized online using Desmos. The plots for custom activation and R function are provided in Fig. 2.21.

Function for R constraint

The default R constraint uses Slater Decay over inverse function $= \frac{\beta e^{-\alpha(\mathbf{x}-\gamma)}}{\mathbf{x}}$. The function can be visualized from 2.22. The modified function follows the boundary conditions of an ideal PES, i.e.:

- $y \to \infty$ for $x \to 0$ [high energy region],
- $y \to 0$ for $x \to \infty$ [asymptotic region].

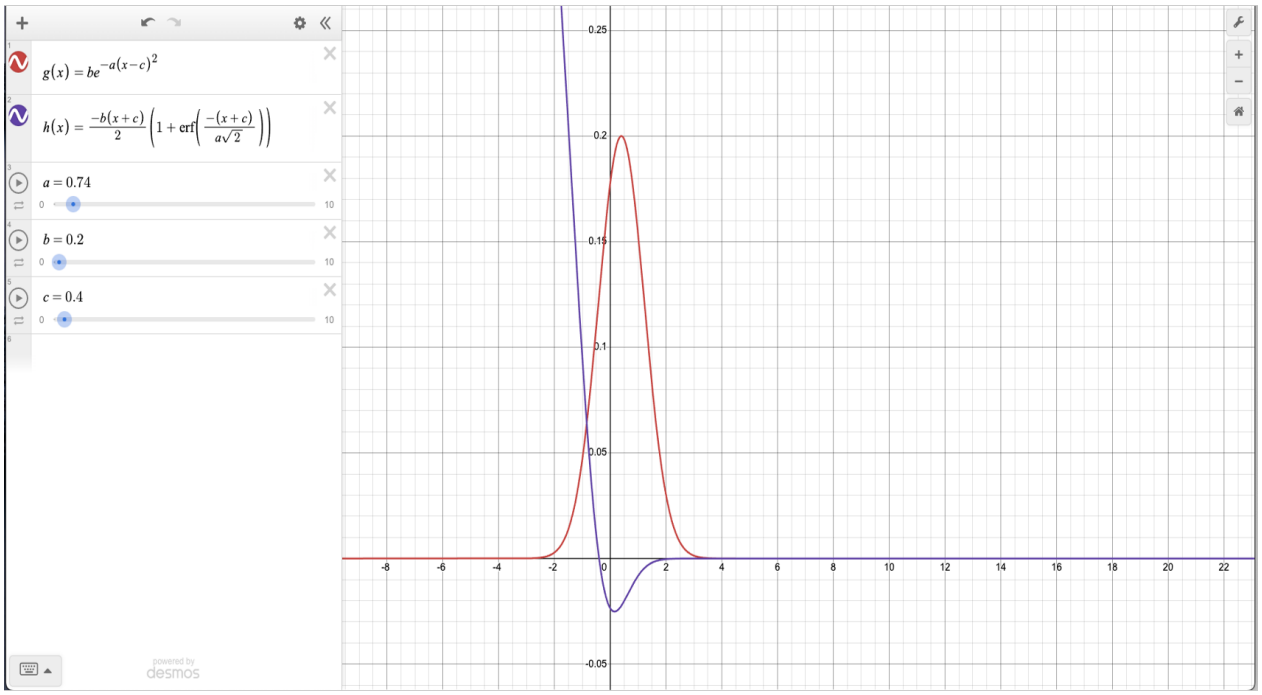


Figure 2.21. Image showing custom activation functions created for the PES_ND model i.e. Gaussian and NGELU are shown in red ($g(x)$) and purple ($h(x)$) respectively. † *Snapshot of functions plotted using desmos.com.*

The minima and other PES features are learned by the NN model, and since R is also passed through the branched NN hidden layers, the final results are smooth, and no further fitting is needed for extrapolating the radial features. However, due to the constraint, the model will need more epochs to converge and the error will be slightly higher than the generic model.

Another function $-a * e^{-b*x} \times \ln(c * x)$ is also available in the program (*commented out*), which has lower error but results in overfitting for long rigid rotors. The function $-e^{-x}\ln(x)$ appears as integral ($\gamma = -\int_0^\infty e^{-x}\ln(x)$) giving Euler's constant ($\gamma = 0.577...$), and inherently follows the generic shape of the PES:

- $y \to \infty$ for $x \to 0$ [high energy region],
- $y \to 0$ for $x \to 1$ [function enters negative region] ($y > 0$ for $x \to 1^-$ and $y < 0$ for $x \to 1^+$),
- point of inflection (curvature changes sign) [minima appears],
- $y \to 0$ for $x \to \infty$ [asymptotic region].

This function can be modified, i.e., divided by x to make the high energy region increase more rapidly ($x \to 0$) and also force the function to remain zero as $x \to \infty$. The constants (α, β, γ) can be added for flexibility to allow the function to take the shape of the PES and the well depth of the minimum.

If required, the original decay function can be modified in the 'PES2MP_driver.py' file using:

```
def create_CustomDecayLayer():
    ...
    def call(self, inputs):
        x = inputs
        # Comment original function:
        # return ( self.a * tf.exp( -self.b * (x - self.c) ) ) / (x)
        # And replace with this
        return ...
return CustomDecayLayer
```

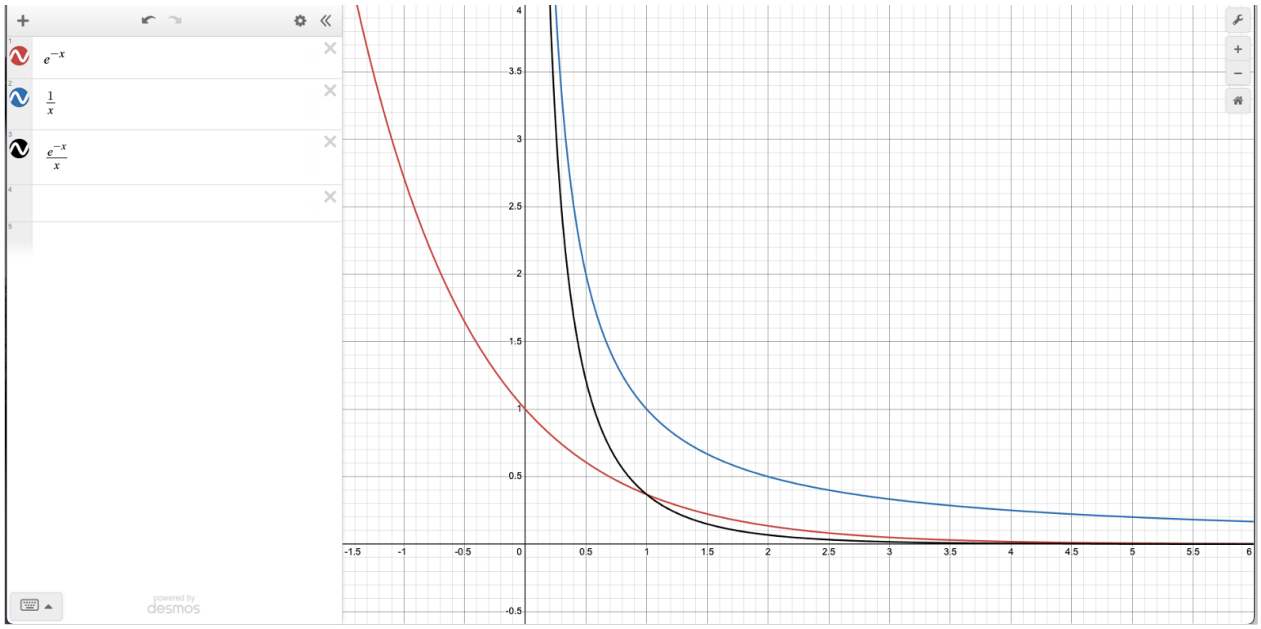


Figure 2.22. Image showing the function implemented to constrain R in the PES_ND model. The modified function has properties of both: the inverse function (i.e. $y \to \infty$ as $R \to 0$) and decay function (i.e. $y \to 0$ as $R \to \infty$). † *Snapshot of functions plotted using desmos.com.*

2.5.9 Ensemble Model

The final part of the PES2MP NN models is the ensemble model, which can be generated using N base models specified in the input file as shown below:

```
#-----------------------NN model Parameters-----------------------------------#
# NN model parameters (Program uses remaining data points for validation dataset)

# Example 1
train_dataset   = [90, 80]*2       # % of data points used for training model
testing_dataset = [5,   5]*2       # % of data points for testing model

# Result (Trn:Tst:Val): Model 1 = 90:05:05, Model 2: 80:05:15
# Model 3 = same as model 1, and Model 4 = same as model 2.
```

The training dataset $90\%, 80\%$ generates two models, and $*2$ depicts that two further models will be generated with the same split (a total of 4 models). Only the training and testing dataset split is provided in the input, and the validation dataset is the remaining data points ($100\% - \%$ Training data $- \%$ Testing data). The above models will have 5% $(100-90-5)$ and 15% $(100-80-5)$ data points in the validation dataset, respectively.

```
# Example 2 (creating ensemble model using eight NN models)

train_dataset   = [80]*8       # 8 Models all with 80% training dataset
testing_dataset = [ 5]*8       # 8 Models all with  5% testing dataset
```

Similarly, in the second example, 8 models will be created, all with the same dataset split of $80\% : 5\% : 15\%$ for training, testing, and validation, respectively.

Fig. 2.23 shows the architecture of the ensemble NN model. Here the four base models are represented as Functional which maps the input data to the base models, and their results are averaged and concatenated, giving a total output shape of (None, 5) representing the output for (a) 4 layers of base models and (b) 1

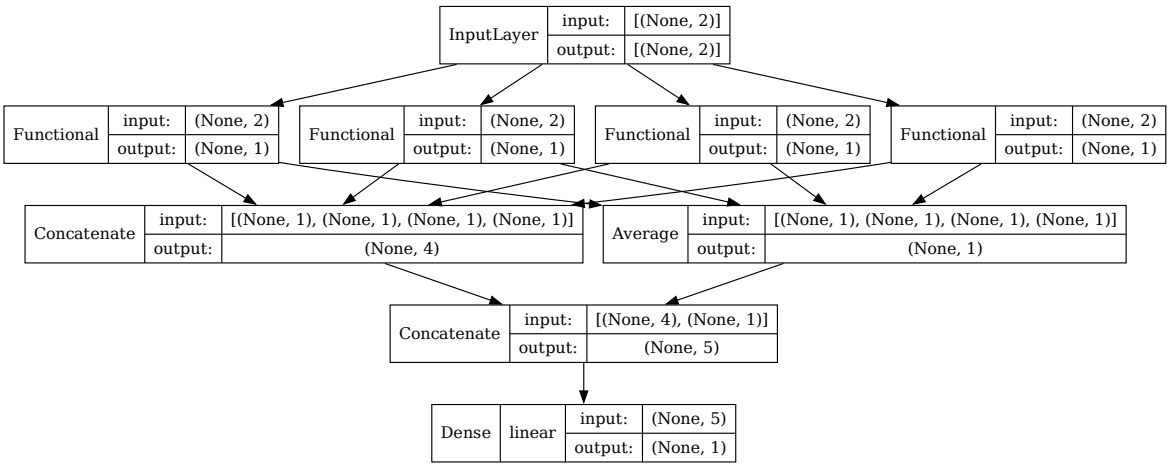


Figure 2.23. Example of ensemble model (generated and plotted by PES2MP) using four base models (represented as 'Functional'). The model creates an average layer ('Average') and concatenates the base results ('Concatenate'). Then it again concatenates 'Average' and 'Concatenate' layers to produce results that are free from the caveats of a singly trained model.

layer of the average layer. The final **linear** layer ensures that the results have the correct output shape to that of the number of outputs without introducing any new features into the results.

There is yet another important hyperparameter that users can set in the NNGen input file, i.e., the number of iterations (epochs):

```
NN_hyperpara = {
    ....
    'maxit_trial' : 250,          # max iterations during trial (100-500)
    'maxit_base'  : 1000,         # max iterations: base model (500-5000)
    'maxit_ensemble' : 10000      # max iterations: ensemble model (5-10K)
}
Ensemble_Model_Train = True
```

The trial iterations are used for finding optimum architecture from the provided options (uses Gaussian Process). The optimum values for the same are anywhere from 250 to 1500. Once the program finds the optimum architecture, it trains the base models and then finally proceeds to train the ensemble model.

If the *Ensemble_Model_Train* is set to **false**, the program will **not allow base models to be trained** when ensemble model training is being done. The ensemble model **uses the first dataset split ratio** for training and finding the best weights for each base model. In this case, users must set the base model training iterations to large values ($10,000+$). Here, the results of individual base models can be used for augmenting PES.

If the **ensemble model training** is **true**, the above set parameters (fewer training epochs for base models and more epochs for ensemble model) ensure that the **base models are trained together with other models during ensemble training**. The fewer epochs for the base models ensure that the base model is trained partially to allow more flexibility during ensemble training. However, the results of individual base models cannot be used for augmenting PES in this case. *Note: This method can be computationally expensive on older machines.*

The early stopping is also implemented in the program that stops the training if the validation error starts to plateau, i.e., no improvement in validation accuracy even after M cycles. The number of M cycles is denoted by patience, and the check for early stopping starts after a specific cycles, which can also be provided

by the user, as shown below:

```
early_stop_para =  {
    'start_after_cycle_base' :  20,   # 20-50% of max iterations: base
    'patience_step_base'     :   5,   # 5-10% of max iterations: base
    'start_after_cycle_en'   :  50,   # 50% of max iterations: ensemble
    'patience_step_en'       :  10   # 10% of max iterations: ensemble
}
```

Values are provided as % of training iterations. Usually, the base model needs 5000+ iterations (epochs) for proper training. In the initial training phase, the validation error can oscillate rapidly, therefore, users must set when the early stopping kicks in by adjusting the value of 'start after cycle base/en' accordingly. For e.g., if the max epochs for the ensemble is 10,000 and you want the ensemble model to train completely, set the value to 50, denoting 50% of $10,000 = 5,000$ epochs, i.e., the early stopping will not interfere with training till 5000 epochs.

The patience, on the other hand, denotes the number of epochs that have passed (after 5000 epochs in the above case) without any improvement in validation error. For the above case, 10% of ensemble epochs = 1000 epochs. If the validation error between, say, 6001 and 7000 does not improve, the program will stop training and will use the epochs that had the lowest error, i.e. 6000.

Accessing performance!

Training with base model tuning off!

The residual plots (both from the testing and the full dataset) and the training error plot (for the loss function (Huber loss) and MAE (mean absolute error)) are plotted for both base NN model and the final ensemble model. The same are provided below for reference.

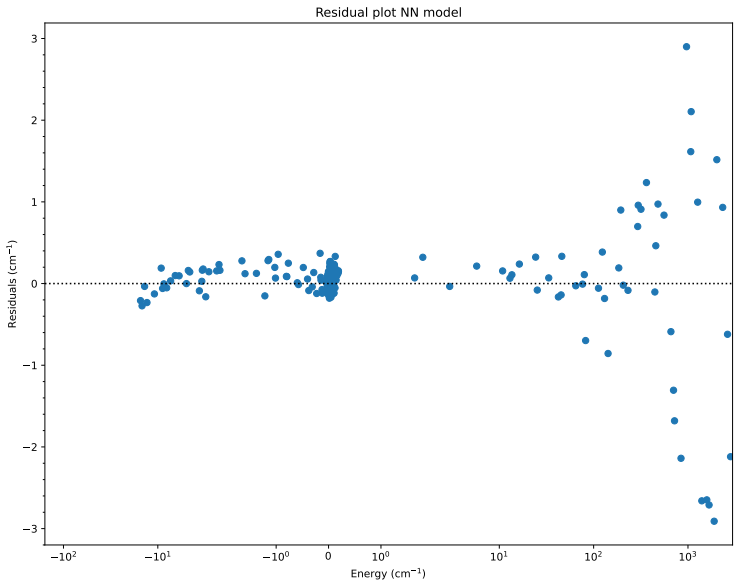
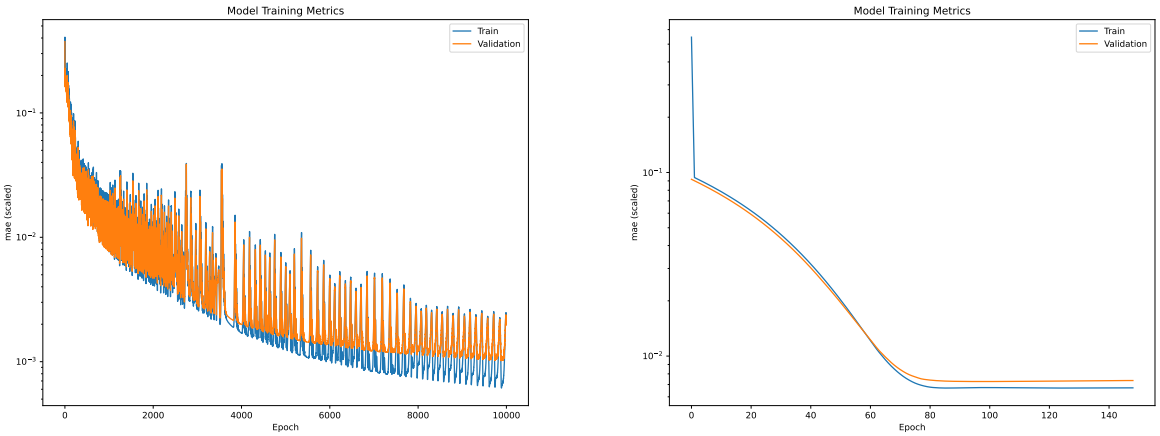


Figure 2.24. Full dataset residual plot for Ensemble NN model (ND_PES model) for a 2D PES with an energy cutoff of 3000 cm^{-1}. The residuals at the minima region show spectroscopic accuracy. The error > 1 cm^{-1} is observed for energies > 100 cm^{-1} with maximum error of ± 3 cm^{-1} for energies > 100 cm^{-1}.

The residual plot is the standard way to check if the Model output is performing within a reasonable accuracy and precision for a regression job. The above plot in Fig. 2.24 shows the deviation (both +ive and

-ive) of the predicted data when compared to the training data. The error in the minima and asymptotic region is < 1 cm^{-1} as desired. The high energy error increases to $\backsim 3$ cm^{-1} for energies > 1000 cm^{-1} which is still remarkable as it is just 0.3% deviation. The ND_PES model prioritizes the minima region as it controls the 'true' behavior when dynamics is performed. The very high-energy region within 1% deviation is good enough. The usual kinetic energy of a collider (i.e., range of collisional energies) is $1,000$ cm^{-1}, which can go up to $5,000-10,000$ cm^{-1} for small/light molecules with a very large rotational constant.

Apart from the full residual plot that gives the performance of the complete dataset, a test residual plot is also generated by NNGen, which only contains the test dataset. This is useful as it assesses the performance of the NN model on completely new data that the model never saw during training or validation.



(a) Base Model training MAE (b) Ensemble Model training MAE

Figure 2.25. Training MAE (mean absolute error) plots for (a) base model and (b) ensemble NN model generated by PES2MP with base model training turned off. The mae (scaled) represents normalized error, and epochs represent training iterations. In the base model (a), the validation error steadily decreases with training error as the epochs progress. The validation error stagnates around 8000 epochs, showing little improvement. In the ensemble model, the training stops early at 150 epochs since the training error does not improve even after 50 cycles.

The plots in Fig. 2.25 represent the training error in MAE (mean absolute error). The error is low since the input and output data are normalized (scaled) to keep NN coefficients small and make it easier for the model to find optimal parameters. So remember, the MAE error in these plots does not represent the true error in cm^{-1}. We shall get that data from the residual plots generated alongside these plots (in the same folder).

As visible, the ensemble training does not go to 10,000 epochs as the early stopping is implemented (which stops training if the validation error does not improve after 500 cycles). This is the case where the base model training during ensemble tuning is kept False. Here, the base models are the ones that do most of the 'heavyweight training'. If these models are poor, the ensemble model cannot improve more than $5\%-10\%$. Only the models are scaled to find the best balance, i.e., lowest overall error in prediction.

If users are working on a supercomputer or standalone workstation with a GPU/NPU and faster DDR5 RAM, they can set $Ensemble\_Model\_Train = True$ to enable fine-tuning of base model weights. The users must optimize the early stopping and training epochs to prevent under/overfitting.

The error for any supervised NN model for augmenting PES must be close to 1 cm^{-1}. But that must not be the end goal. Suppose the error for an NN model is around $2-3$ cm^{-1}. *Is it acceptable?* The users must

try certain things before giving up on NN training.

- Try flexible $e^{-x}\log(x)$ function for the 'R constraint' (modify pes2mp_driver.py).
- Try the generic NN model (without the 'R constraint'), keeping the bias zero (modify pes2mp_driver.py).
- Identify if a certain dataset split has less error and use the same for 4-8 ensemble models.
- Increase the number of training epochs ($10,000+$) for both base and ensemble models.
- Try a different (or custom) activation function that you think can better mimic the shape of PES.
- **IMPORTANT:** Use small data for training ($20\%-30\%$) and more for validation ($50\%-60\%$) to speed up training and access the performance.
- Use external NN packages like PESLearn (The template for the same is available in PES2MP_ipynb).

Suppose the error does not reduce, or it reduces, but new unwanted anomalies exist in the NN generated PES. Analyze the error to see what the error is around the extremes (minima/maxima) and near '0' where the asymptotic data point lies. If the error is slightly > 1 cm^{-1} but is justifiable, for example, the RMSE is $\backsim 1-5$ cm^{-1} but % relative error is $< 1\%$ compared to *ab initio* PES (in case of very large potential well: rigid rotor (anion/cation) and H_2 collision), proceed with the data.

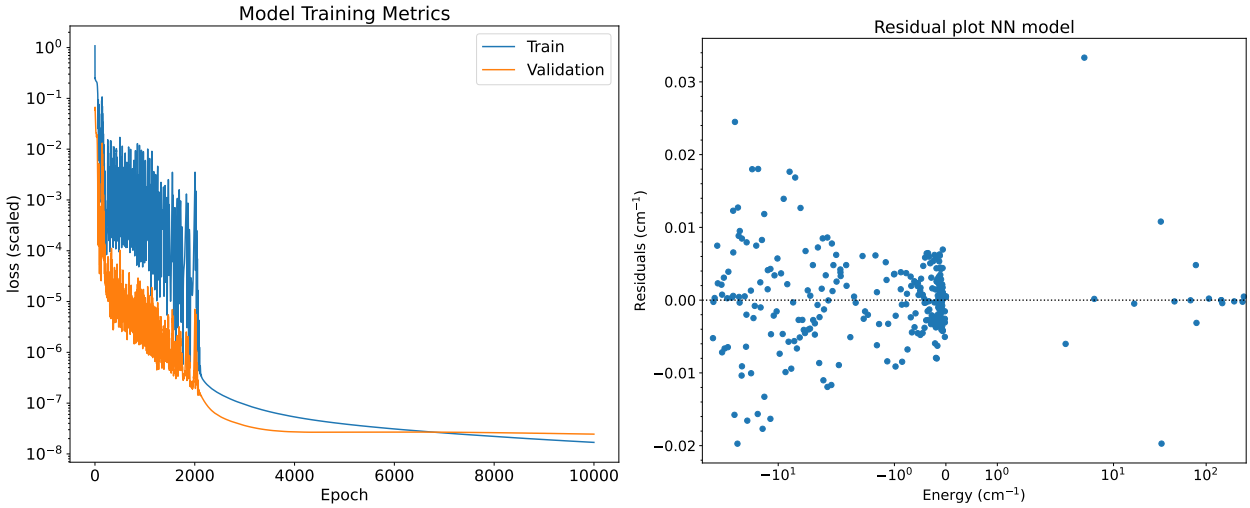
Training with base model tuning on!

While the NN results in Fig. 2.24 and Fig. 2.25 are trained on 4 base models (80:05:15 split ratio) with training epochs of 10,000. The ensemble model has base tuning off and is trained for a few 100 cycles. On the other hand, if the base model is trained with the same parameters, but the cutoff is lowered to 300 cm^{-1} and the ensemble model is trained for 10,000 epochs with base model tuning turned on, the residual error decreases 10-fold. The parameters for training a 2D PES (C_2-He) with base tuning enabled is provided below:

```
train_dataset   = [80]*4      # % of data points used for training model
testing_dataset = [5]*4       # % of data ponts for testing model

NN_hyperpara = {
    'Max_trial'   :       1,           # number of trials for architecture search
    'NN_nodes'    :    [64],      # search space for NN nodes per layer
    'NN_layers'   :     [4],      # search space for NN layers
    'NN_branches' :     [4],        # search space for NN Branches: even only
    'maxit_trial' :      10,          # max iterations during trial (100-500)
    'maxit_base'  :   10000,          # max iterations: base model (500-5000)
    'maxit_ensemble' : 10000      # max iterations: ensemble model (5-10K)
}
Ensemble_Model_Train = True
# NN model early stopping hyper-parameters
early_stop_para =  {
    'start_after_cycle_base' :  50,   # 20-50% of max iterations: base
    'patience_step_base'     :  10,   # 5-10% of max iterations: base
    'start_after_cycle_en'   :  50,   # 50% of max iterations: ensemble
    'patience_step_en'       :  10   # 10% of max iterations: ensemble
}

High_E_cutoff = 300        # E in cm-1 ---> Model Energies lower than this cutoff
```

The training loss and residual plot of the same is shown in Fig. 2.26. The training loss for the ensemble model in Fig. 2.26 (a) shows oscillations due to fine-tuning of the base models till 2000 epochs (this would not occur if the base model tuning is off), after which, the training loss steadily decreases. At epoch $\backsim 7000$, the validation loss becomes larger than the training loss, showing that model training is nearly complete and

continued training would have overfitted the model. The residual plot in Fig. 2.26 (b) shows that the error for the NN model has decreased 10-fold from the previous case with residuals in the range ±0.03 cm^{-1}.



(a) Ensemble Model Training Loss

(b) Ensemble Model Residuals

Figure 2.26. (a) Training loss (scaled) ensemble NN model generated by PES2MP with base model training turned on. The loss (scaled) represents total residual error, and epochs represent training iterations. (b) Residual plot showing error in NN-generated PES compared to *ab initio* dataset (i.e., including training, validation, and test datasets).

These results are obviously an overkill for the system, and the model needs much more training time (10-100x) compared to the previous case, where the base model training is off. While the 2D systems are doable, the 4D PES can overwhelm a PC (or old cluster) due to a larger dataset. However, the total time would still be a couple of days, which is much better than *ab initio* calculations. Therefore, if the computational resource permits, the ideal choice for PES training should be the above parameters, which are also provided in the */input_files/2_NN_Opt* as *NN_2D.py* and *NN_4D.py*. If not, the users can opt for a smaller architecture with frozen base coefficients, however, the validity of spectroscopic accuracy must be ensured for dynamical calculations [69].

2.6 Multipole Expansion

The multipole expansion of PES refers to expressing the PES into a series of orthogonal functions (with a dependent scattering angle θ and an increasing order of angular momentum Λ). The higher-order terms (moments) become smaller and, therefore, can be terminated after a certain accuracy is achieved. This is a very important part of studying collisional dynamics, as these multipole moments affect the probabilities (cross-sections) for rotational or ro-vibrational transitions for collisions at various kinetic energies.

2.6.1 Physical Interpretation

To understand a mathematical transformation, it is always beneficial to look at the underlying idea behind such a transformation [166, 167]. In the case of time-independent collisional studies, we take a PES which is dependent on the distance (R) and orientation ($\theta(s)$) of the incoming collider. For simplicity, take the rigid rotor to be a fan's blade rotating either clockwise or anticlockwise. Now, consider the incoming

collider (atom) as a ball approaches this rotating blade. The potential it will see will depend on both R and angle. However, over a long period of time (which we can expect in ISM), the colliders will approach the rigid rotor from all directions, and therefore the angle dependence can be averaged based on the potential minima/barrier that the incoming collider will feel. In quantum mechanical systems, these rotational states are quantized; therefore, the angle dependence is expanded using spherical harmonics, and cross-sections are computed via partial wave methods.

The multipole expansion generates radial terms in an increasing order of angular anisotropy, where the first term $\Lambda = 0$ is the isotropic average (monopole), $\Lambda = 1$ is the dipole anisotropy, $\Lambda = 2$ is the quadrupole anisotropy, and so on. These terms describe the probability of change in the rotational state of the rotor depending on the energy of the collider. For example, $\Lambda = 0$ describes probability of elastic scattering, i.e. $\Delta j = 0$, $\Lambda = 1$ describes $\Delta j = \pm 1$,etc. Therefore, in the time-independent study, we transform the PES into a series of radial terms that depend on the increasing order of monopole, dipole, etc.

A general expression for a 2D/4D rigid rotor PES expansion:

$$V(R, \theta_i) = \sum_{\Lambda} V_{\Lambda}(R) \vartheta_{\Lambda}(\theta_i) \tag{2.22}$$

where $V(R, \theta_i)$ is the 2D/4D PES, $V_{\Lambda}(R)$ are the radial terms (radial potential coefficients) that are obtained after the PES is expanded into a sum of Λ terms. Since the angular terms are converted into orthogonal $\vartheta_{\Lambda}(\theta_i)$ terms (which can be Legendre (2D) or coupled Spherical Harmonics (4D)), the final radial terms are independent of angles and represent the averaged effect of the collision with increasing order of angular dependence, i.e. isotropic (monopole) and anisotropic (dipole, quadrupole, etc.).

These terms give us the cross-section, which is the measure of the likelihood of an interaction. Classically, the cross-section represents the effective target area within which a collision can occur. In rotational dynamics, the cross-section quantifies the probability that a molecule undergoes a specific transition per unit flux of incoming colliders. Since the unit for our distance is Å, therefore, the final cross-sections are represented in $Å^2$. The cross-section of an elastic (no change of rotational state) and inelastic (rotational (de-)excitations) collision of a rigid rotor by any collider at a specific kinetic (translational/collisional) energy can be calculated using these terms. However, let us first take a look at how these Λ terms are important in imposing the selection rules on rotational states:

$$\left\langle j_f \middle| \vartheta_{\Lambda} \middle| j_i \right\rangle \tag{2.23}$$

Expanding the contributions from different terms, we get:

$$\left\langle j_f \middle| \vartheta_{\Lambda_0} + \vartheta_{\Lambda_1} + \vartheta_{\Lambda_2} \ldots \middle| j_i \right\rangle \tag{2.24}$$

$$\left\langle j_f \middle| \vartheta_{\Lambda_0} \middle| j_i \right\rangle + \left\langle j_f \middle| \vartheta_{\Lambda_1} \middle| j_i \right\rangle + \left\langle j_f \middle| \vartheta_{\Lambda_2} \middle| j_i \right\rangle + \ldots \tag{2.25}$$

Leading to selection rules:

$$\delta_{j_f j_i} + \delta_{j_f j_{i\pm1}} + \delta_{j_{jf\, i\pm2}} \ldots \tag{2.26}$$

How does one physically interpret monopole, dipole, and quadrupole terms? Remember a rotating fan and a colliding ball. Many physical analogies can be drawn from this example. For example, what if the blades are rotating slowly vs fast? Imagine a very fast rotating fan blade, the ball will only see an even field on all sides. This is the isotropic (angle-independent) behavior described by the monopole. When slow, the

ball will see uneven potential, e.g., there is a chance it will be deflected by the blade, or it can reach its COM without any barrier. This unevenness is what is expressed as first-order anisotropy. Similarly, when these blades are uneven or bent, we can expect more complex anisotropic behavior, which is described by the quadrupole and higher-order terms.

Suppose j_i and j_f represent the initial and final state of the rotational state of a rigid rotor molecule. The Λ is the order of the multipole expansion. Monopole describes only elastic collision, i.e., no change of state. Similarly, dipole and quadrupole moments allow $\Delta j = \pm 1$ and $\Delta j = \pm 2$ transitions, respectively. The final transition probability will depend on the interaction potential, selection rules based on symmetry, and the energy of the incoming collider.

2.6.2 Analytical Fit

2D: MP expansion of the simplest system is a rigid rotor (RR) − atom collision shown below:

$$V(R,\theta) = \sum_{\lambda} V_\lambda(R) P_\lambda(\cos\theta) \tag{2.27}$$

where, P_λ 's are the Legendre polynomial functions and $V_\lambda(R)$ are the radial terms we want to calculate. The Legendre polynomial terms are orthogonal and provide an independent basis for expanding the PES. Once fitted, only the first few terms are needed to fit the PES, as the series quickly converges and the contribution of higher terms becomes negligible. *Note: Interactions between ion-neutral species have long-range interactions that need more terms than neutral-neutral systems.*

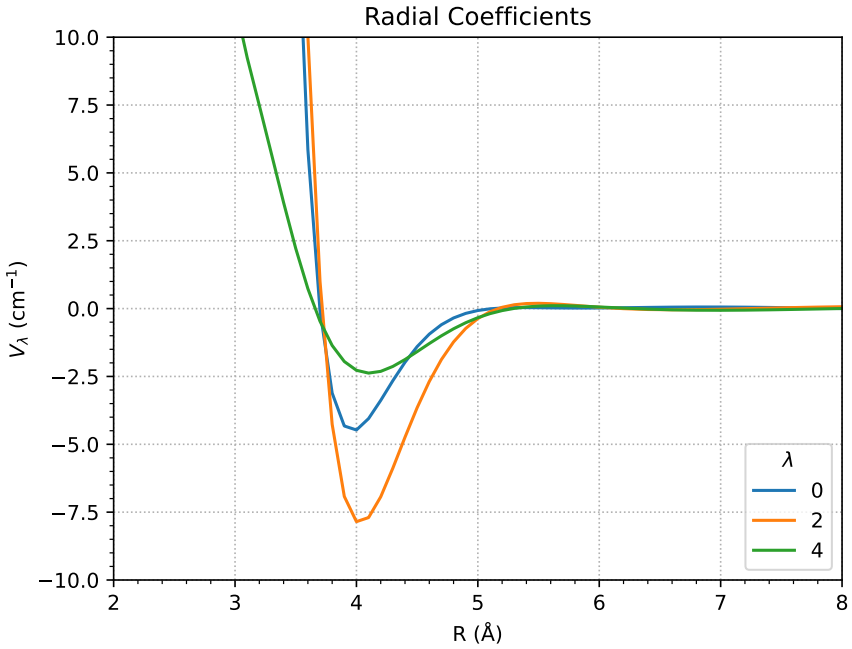


Figure 2.27. Example of MP expansion of C_2—He collision system (DF-MP2/AVDZ) for first three radial terms generated and plotted by PES2MP.

4D: For the 4D collision, the PES has three angular terms. A way to expand such potential is to use the coupled Spherical Harmonics term [168] that has been expanded and shown below:

$$V(R,\theta_1,\theta_2,\phi) = \sum_{\lambda_1\lambda_2\lambda} V_{\lambda_1\lambda_2\lambda}(R) \sum_m \langle \lambda_1 m \lambda_2 - m | \lambda 0 \rangle \, Y_{\lambda_1}^m(\theta_1, 0) Y_{\lambda_2}^{-m}(\theta_2, \phi) \tag{2.28}$$

where we use the two Y_l^m function and take the first ϕ to be zero. In the two RR collision case, $\phi_1 - \phi_2$ is a constant that is taken to be ϕ. Simplifying even further, the Y_l^m function is expanded into associated Legendre polynomials P_l^m for the case $m = 0$ and $m > 0$. The same is provided below:

$$\begin{aligned} I_{l_1 l_2} =& [(2l+1)/4\pi]^{1/2} \Big[\langle l_1 0 l_2 0 \mid l 0 \rangle \, P_{l_1}^0(\theta_1) P_{l_2}^0(\theta_2) \\ &+ \sum_{m=1}^{l_m} (-1)^m 2 \langle l_1 m l_2 - m \mid l 0 \rangle \, P_{l_1}^m(\theta_1) P_{l_2}^m(\theta_2) \\ &\times \cos[m(\phi_1 - \phi_2)] \Big], \quad l_m \leqslant l_1, l_2. \end{aligned} \tag{2.29}$$

where P_l^m are unnormalized associated Legendre functions including the Condon-Shortley phase i.e. $P_l^{-m}(\theta) = (-1)^m P_l^{-m}(\theta)$. The $\langle l_1 m_1 l_2 m_2 | l m \rangle$ are the Clebsch-Gordan coefficients such that $m_1 + m_2 = m$ and $l_1 + l_2 + l = \text{even}$ This is called the space-fixed expansion and is preferred over another uncoupled expansion (called body-fixed expansion) due to orthogonal nature and faster convergence of coefficients.

These analytic fits are not as difficult as *ab initio* calculations, and can be regarded as trivial by physicists/theoreticians. However, since these expansions are not as popular as DFT calculations, it can be difficult to find a user-friendly code for the same. The present expansion codes have been verified by comparing the cross-sections and rate coefficients generated by them against the published data for the known systems. The final codes are written as a general-purpose function in Python and can be modified further to include vibrating diatoms, rigid benders, asymmetric rotors, etc.

A detailed technical review of such expansions can be found in the 1985 paper by Sathyamurthy. [169] The paper is a recommended read and covers fitting procedures for both reactive and non-reactive scattering, i.e., rigid rotors, vibrating diatoms, etc.

The mathematical background of this expansion is directly linked with cross-sections for the rotational transition of the rigid rotor under study. One can either treat this as a black box to get cross-sections and rate coefficients (whenever necessary) or take a dive into the mathematical formulation of 3D rotation and projection from this Wikipedia page and Edmonds' Angular Momentum in Quantum Mechanics [170].

2.6.3 Radial Terms (V_Λ)

The fitting of PES into finite order of Legendre and Spherical Harmonics (SH) functions results in radial terms which are referred to as V_Λ where Λ is the general term for the order of angular expansion. As discussed, these functions converge rapidly in most cases (except for strong anisotropic interactions) and therefore only a handful of terms provide a good approximation of the original PES.

An important detail to remember for multipole expansion is that the PES data is transformed into V_Λ using a set of known functions **numerically**. This is achieved by taking the pseudo-inverse of the matrix containing Legendre/SH coefficients and multiplying with PES (one R point at a time). The resulting terms will therefore follow certain behaviors as listed below:

- **Symmetric molecules have even Λ values.** For a symmetric linear molecule and an atom (2D) collision, the angular term (θ) is converted to λ. Since the PES after 90°, i.e., from $90° - 180°$ is a mirror image of PES from $0° - 90°$, the even terms disappear (see Legendre polynomials plot for each term). For 4D collision, the three angular terms become $\lambda_1, \lambda_2, \lambda$. The λ_1 and λ_2 values depend on the symmetries of

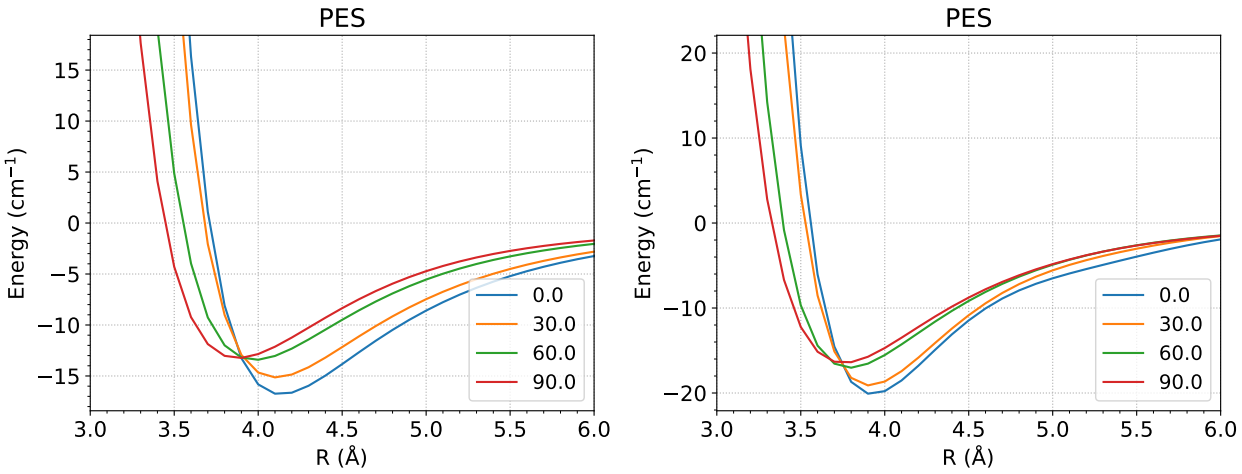
respective rigid rotors, i.e., if molecules are symmetric, then values are even. The vector sum λ will be even valued only if both λ_1 and λ_2 are even valued (i.e., both molecules are symmetric). This signifies that only even rotational transitions will occur in symmetric molecules.

- **The number of radial terms (V_Λ) cannot exceed number of angular terms.** However, fewer radial terms can be obtained from more angles. For example, if a 2D collision system has 19 angles, we can get a maximum of 19 λ terms (0-18). Since the code transforms the PES into V_Λ by taking the pseudo-inverse (least squares fit) of the matrix, the code will, in theory, give fewer or more V_Λ terms if needed. However, a warning will appear as the V_Λ terms after $\lambda = 19$ will be incorrect. Why? Solving 19 equations can only give 19 unknown variables, not 20 (remember Algebra!). Suppose you generated 13 radial terms (from 19 angles), i.e. $\lambda = 0-12$, it will be fine as a problem with 13 unknown variables and 19 equations will converge to the correct solution. However, if you wish to recreate PES from these 13 V_λ terms, the PES can only have 13 angles or less, not 19 as in the original dataset. The detailed example of the same is provided in Section 4.2.2.

- **No data point must be missing.** If any data point is missing, the matrix containing PES will be distributed poorly, resulting in errors or wrong data transformation/fitting. To get missing data points, the NN model and 1D analytical fitting can be used.

2.6.4 Quality of PES and V_Λ

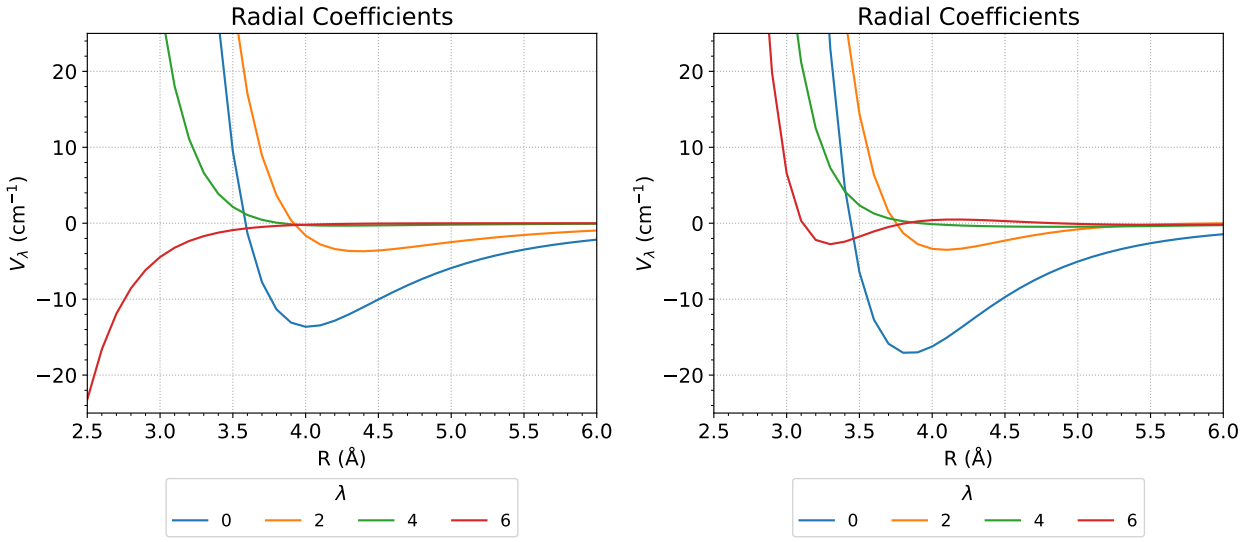
Let us take the example of C_2-He. For the 2D systems, the PES is generated using HF-D4/AVQZ (CP) and Sherrill's Gold Standard (SGS) method. Adding dispersion correction to HF is shown to preserve the location of minima and shape of PES compared to DFT methods as shown in Fig. 2.4(d).



(a) HF-D4 (b) SGS

Figure 2.28. 1D potential curves at four different angles for (a) HF-D4/AVQZ (CP) vs (b) Sherrill's Gold Standard (CBS) method.

In the above figure (Fig. 2.28), the 1D potential curves for D4 correct HF PES have slightly lower minima than the CBS method. After multipole expansion, the difference between the first four radial terms is shown below in Fig. 2.29, where we find that the radial terms have nearly identical shapes for the two PESs. The only difference is the well depth of the isotropic term and the curve shape for $\lambda = 6$.



(a) HF-D4 (b) SGS

Figure 2.29. Radial terms V_λ for PES obtained using (a) HF-D4/AVQZ (CP) vs (b) Sherrill's Gold Standard (CBS) method.

While the difference between the two interaction potentials may be small, the cross-section calculations are very sensitive to the shape and location of minima. The resulting cross-sections obtained after using the above radial terms in MOLSCAT are shown in Fig. 2.30. The figure is plotted using Origin.

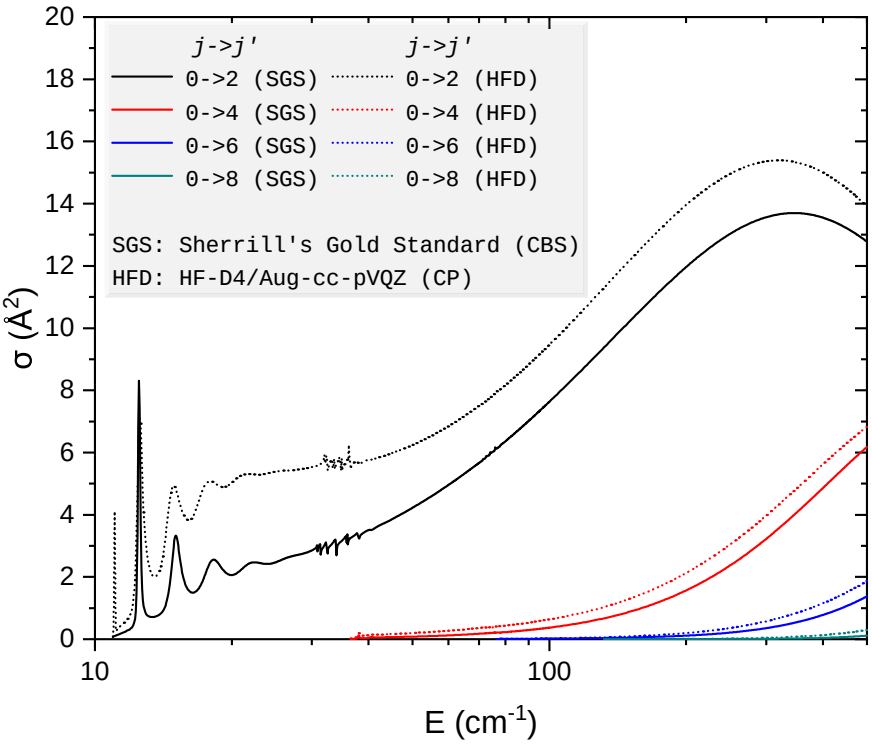


Figure 2.30. Cross-sections for rotational excitation of the first few rotational states of C_2 by He. The solid line represents cross-sections obtained from the interaction potential obtained using the SGS method, while the dotted line represents the same obtained using the HF-D4 method. † *Plotted using Origin.*

Another example of studying the quality of the PES directly affecting the quality of the radial terms and

cross-sections has been demonstrated in the paper by Kushwaha *et al.* [138], where a dense PES is generated and using a small subset ($\backsim$ 125 data points) of sparse PES. The PES was augmented using traditional cubic-spline, NN, and GP methods. Table 3 and Fig. 4 in the paper summarize the error in regenerating this PES. The four sets of PES (Exact, NN, GP, and spline augmented) were used to get radial terms, which are shown in Fig. 7 of the paper. The spline-generated surface had the highest error for both systems and had deviated substantially from the minimum location of the exact surface.

To understand the details and results of MOLSCAT calculations, follow the next Section.

2.7 MOLSCAT's &POTL Block

2.7.1 Introduction

MOLSCAT [73] is a software package written in Fortran that solves scattering cross-sections [171, 172] for collisions between various molecular systems [173–175]. The two most common collisional dynamics calculated using MOLSCAT are rigid rotor-atom and two rigid rotors. Here we shall see how we can utilize MOLSCAT for studying these collisions at cold and ultracold temperatures [91].

Remember the selection rules, where monopole describes the elastic collision, i.e., no change of state, while the anisotropic terms (dipole, quadrupole, etc. moments) describe the change in rotational states by $\Delta j = \pm 1$, $\Delta j = \pm 2$, etc. These selection rules are R-independent, while $V_\Lambda(R)$ terms are not and describe how the interaction strength of any term varies with distance. To understand how these radial terms give us cross-sections can be difficult at first. We shall briefly discuss the same for a 2D system:

$$\langle j_i | V(R,\theta) | j_i \rangle \tag{2.30}$$

$$\sum_\lambda V_\lambda(R) \langle j_i | P_\lambda(\cos\theta) | j_i \rangle \tag{2.31}$$

This yields the coupling terms, $W_{ij}(R)$, where R independent terms are evaluated once and V_Λ terms are provided separately. While the angular terms, $P_\lambda(\cos\theta)$ is responsible for imposing selection rules, the radial terms $V_\lambda(R)$ provide the corresponding radial strengths. These coupling terms are stored as $\mathbf{W}(R)$ and used in evaluating:

$$\frac{d^2\boldsymbol{\psi}}{dR^2} = [\mathbf{W}(R) - \mathscr{E}\mathbf{I}]\boldsymbol{\psi}(R) \tag{2.32}$$

where $\mathscr{E} = 2\mu E/\hbar^2$ and E is total energy. After solving for $\psi(R)$, the S-matrix is extracted using the boundary conditions described in the MOLSCAT manual. This S-matrix contains information about the probability amplitudes and phases for the various possible outcomes of the collision. Finally, using the S-matrix elements, the cross-sections for specific $j_i \to j_f$ are computed using the formula:

$$\sigma_{j_i \to j_f} = \frac{\pi}{g_{n_i} k_{n_i}^2} \sum_{\substack{J_{\text{tot}} \\ M}} (2J_{\text{tot}} + 1) \sum_{\substack{i \in n_i \\ f \in n_f}} \left| \delta_{if} - S_{if}^{J_{\text{tot}},M} \right|^2 \tag{2.33}$$

In MOLSCAT, one may find different terminology, e.g., rotational states are referred to as channels, and their numbering starts from 1 as opposed to 0. This is because MOLSCAT can handle many different kinds of problems where vibration and splitting of states are involved. Nevertheless, remember to look at the energy of the (rotational) state (denoted by f and i) from the reference sheet just above the cross-section table.

Consider an example where the maximum rotational states chosen for any particular study are 10. Suppose that the collisional energy provided by the user is smaller than the 5^{th} rotational state of the rigid rotor based on the rotational constant ($B \times j \times (j+1)$). Since the total energy provided includes both the rotational energy of the rigid rotor and the relative kinetic energy of the colliding partner, it will not be sufficient to reach the 5^{th} rotational state. Therefore, you will not see any transitions to/from ($j \geq 5$). These rotational states (channels) are deemed *closed*. While rotational states/channels ($j < 5$) are termed *open* and the cross-sections for the same can be calculated at that particular energy. In quantum scattering, closed channels can still contribute virtually through coupling effects, influencing the dynamics even if they are not physically accessible. Therefore, (sufficient) closed channels must be included in dynamical calculations to verify if the cross-sections are converging.

If you need to study the dynamics of a molecule at say $j = 5$ (6^{th} rotational state as j starts from 0), the collisional (total) energy that you must enter in the input file must be $E = E_{rot(j=5)} + E_k$, where $E_{rot(j=5)}$ is the energy of the 5^{th} rotational state of the rigid rotor ($E_{rot(j=5)} = B \times 5 \times 6$) and E_k is the additional kinetic energy. This is useful in ultracold collisional studies.

Apart from cross-sections, the above radial terms can also be used to find bound states of the pair at any particular kinetic energy (using BOUND), which are useful in the ultracold study of molecules. The third package, FIELD, can theoretically predict the effect of the external magnetic field on the bound states and Feshbach resonances at any particular kinetic energy of the collider.

2.7.2 Cross-Section Calculations

The cross-section calculations can be performed in PES2MP by using the *MOLSCAT_pot.txt* file that contains the radial terms in MOLSCAT readable (Slater function) format. The number of rotational states of the rigid rotor that must be included in the rotational basis must be specified using *l1max*. Remember that with increasing energy more and more channels will be open and calculations will also become more computationally demanding. Therefore, the general approach towards such calculations is to target a manageable number of states in the rotational basis (say 20-30). Then, cross-sections are benchmarked by including more and more rotational states to make sure that the same are converging.

When the collisional energy is small, and only the first few channels are open, the users can choose fewer closed channels in the rotational basis. For neutral RR-He molecules with a large rotational constant (e.g., C_2), one additional closed channel in the rotational basis is enough. However, it is still recommended to verify the same. If the rotational constant of the system is very small, the number of open channels will increase very fast, and the calculations will become very slow. Since the code is not parallel, the code provided with PES2MP generates separate input files for each collisional energy, and multiple calculations can be set up in parallel to save time.

Even then, calculations at higher collisional energies > 1000 cm^{-1} can be very slow and can be approximated by using the centrifugal sudden (CS) approximation method as opposed to the exact close coupling (CC) method. At such higher energies, the resonances usually diminish, and the CS approximation can save a lot of computational time. Again, the difference between CC and CS energies must be benchmarked to verify if CS cross-sections smoothly extend to the CS cross-sections.

Once the calculations are done, users can collect the results using auxiliary codes available in the *input_files/aux_scripts/rate_codes_MOLSCAT* folder. The same folder also contains code to calculate rate coefficients in a temperature range from collisional energies and cross-sections.

Kinetic Energy *vs* Temperature

The concept of kinetic energy and temperature must be clear before proceeding to rate coefficients. The kinetic energy is the fundamental concept valid at atomic scales, similar to our natural experience. A simple atom will not have temperature. It is a bulk phenomenon, a statistical property that emerges from the collective motion of molecules. Therefore, when we touch hot water (bulk), with molecules having a lot more kinetic/rotational/vibrational energy, the same is transferred to our skin, which we perceive as temperature.

The kinetic energy of any particle is defined by:

$$E_k = \frac{1}{2}mv^2. \tag{2.34}$$

However, for a classical ideal gas in three dimensions, the **average kinetic energy** of many molecules relates to temperature via the equipartition theorem:

$$\langle E_k \rangle = \frac{3}{2}k_B T \tag{2.35}$$

Putting the value of the constants and room temperature (300 K), the average kinetic energy per molecule is about 0.039 eV or 0.893 kcal/mol. Based on the Maxwellian distribution, most molecules will have < 1 kcal/mol energy; thereafter, the number of molecules with more energy in the distribution will decrease exponentially.

Suppose an iron bat swinging at some speed will have kinetic energy, but does it not correspond to its temperature? The temperature of the ball will be approximately the same as its surroundings due to thermal equilibrium, as the kinetic energy of an iron bat in motion is associated with its macroscopic, organized movement. **But,** if the bat repeatedly hits another object, its temperature will increase due to the transfer of kinetic energy into internal energy, increasing random vibrational and rotational motion of its atomic lattice.

At the atomic scale, the collision of atoms also produces such changes. The only catch is, single atoms do not have any rotational or vibrational motion. They only have kinetic energy (translational motion) and electronic excitations (which need energy in the visible spectrum or higher). Therefore, Helium plays a crucial role as a collider in ISM [176, 177], since it only forms weak bonds with other molecules, does not have internal ro-vibrational states, but can efficiently transfer its kinetic energy due to inelastic scattering.

2.7.3 Rate Coefficients

The interstellar gaseous clouds can have different temperatures in different regions due to the dilute nature of space. Due to inelastic collisions, the molecules start to move towards local thermodynamic equilibrium, but at a very slow rate. The rate coefficients give us the efficiency of rotational transitions due to these collisions, and using the rotational or ro-vibrational spectra, we can have a better approximation about the molecular abundance and temperature in that region (see Section 4.5.1).

The rate coefficients are obtained by averaging the energy-dependent cross-section over the Maxwell-Boltzmann distribution [178] of collision energies. The formula to calculate rate coefficients from cross-sections is provided below:

$$k_{j\to j'}(T) = \sqrt{\frac{8}{\pi\mu}}\beta^{\frac{3}{2}}\int_0^\infty \sigma(E)Ee^{-\beta E}dE \tag{2.36}$$

where $\beta = \frac{1}{k_B T}$, k_B is the Boltzmann constant, μ is the reduced mass of the system and T is the temperature. Because the system needs to provide enough energy to overcome the gap between the initial and final

rotational state, the excitation probability depends on how much total energy is available, and E corresponds to the total energy of the system. For de-excitation transitions, the excess energy from the transition is converted into kinetic energy of the colliding partners, and therefore, E will correspond to the kinetic energy (collisional/total energy - rotational energy) of the colliding partners.

Threshold Behavior, Temperature Dependence & Reverse Rate

Below the threshold energy (i.e., if collisional energy is less than rotational energy gap), rotational excitation will not occur, and therefore excitation rate coefficients will be negligible at low temperatures. However, de-excitation rate coefficients will be nonzero at all temperatures because molecules in higher rotational/vibrational states can always lose energy. In fact, the cross-sections (and therefore the rates) become larger if the kinetic energy of the system becomes infinitesimally larger than the energy of that particular rotational state.

For de-excitation cross-sections at very small kinetic energy, the cross-section often follows a Wigner Threshold Law [91], i.e., the cross-section tends to infinity in the limit of zero kinetic energy [179] of the incoming atom.

Finally, from the equilibrium relation between excitation and de-excitation:

$$\frac{k_{j\to j'}(T)}{k_{j'\to j}(T)} = \frac{g_{j'}}{g_j}\exp\left(-\frac{\Delta E}{k_B T}\right) \tag{2.37}$$

we can obtain excitation rate coefficients from de-excitation rate coefficients and vice versa without integrating again. The g_j stands for the degeneracy of the rotational state, ΔE is the rotational energy gap, and k_B is the Boltzmann constant. The same has been discussed in detail at: M. H. Alexander and P. J. Dagdigian, 'Cross sections, rate constants, microscopic reversibility, detailed balance, and the master equation in inelastic and reactive kinetics', University of Maryland Lecture Notes.

The input files for generating MOLSCAT input files, collecting results, and calculating rate coefficients are explained in detail in the next Chapter (see Section 4.5.1 for more detail on critical density [180, 181]).

3 Program Usage and Examples

The program is written to be easy, efficient, and robust for basic users. However, to make this program developer-friendly (to debug, add functionality, etc.), all program sections are modular. The program checks for job type, executes the module that is set to true, and skips the ones that are set to false.

3.1 Installation

The PES2MP has been created to work on Linux machines, though the program has currently been tested on Mac (both Intel and Apple-Silicon) and Ubuntu. There are two ways to install it. The first method uses a graphical user interface (GUI), while the other is the classic old school character user interface (CUI), where users can run the required script individually to install/run the program.

3.1.1 Preview for Installation

There are two options for PES2MP makefiles:

1. Quick install :: **Filename:** *install_pes2mp_quick.sh* || **Conda environment:** *pes2mp_q*)
2. Conda install :: **Filename:** *install_pes2mp.sh* || **Conda environment:** *pes2mp*)

The quick version uses **python -m pip** command (faster but does not check the compatibility among packages) while the normal version uses **conda** command to install required packages. While the conda installation is recommended, it can sometimes be painfully slow to solve environment conflicts among packages, and therefore, it is suggested that users install both makefiles in separate terminals. If the conda installation doesn't finish in 24 hours (i.e., conda is stuck in a loop and keeps solving the environment), just cancel the installation and use the quick version. If both install normally, users can use either of the two environments to run the program.

3.1.2 Prerequisite

Anaconda

Install Anaconda by visiting anaconda webpage i.e. `https://www.anaconda.com/download`.
It can create environments with specific Python versions, and two environments will not be affected by any change in the base (root) environment. For our work, we shall create **pes2mp** environment utilizing conda and **pes2mp_q** environment utilizing pip for installation of various packages (**makefile is provided**). The important packages are: Psi4, pyshtools, TensorFlow, SciPy, Matplotlib, tqdm, and many more (for the full list, see makefiles).

3.1.3 Installing PES2MP using CUI

The following commands are needed to install PES2MP on your machine. The same are also available in *'Quick Guide'* for the PES2MP manual. Detailed information about the Anaconda software package and managing environments is provided in Appendix A, along with various aspects of libraries installed in the PES2MP software. Here we shall discuss in brief how to install and start using the PES2MP program.

1. Download zipped file from GitHub (QuantumDynamicsLab).
2. Goto **"makefile"** folder.
3. Open **two** terminal windows (one for quick install and the other for conda install) and type the following commands: Users can install either the quick version or the normal version. The instructions are provided for both versions (read above). The quick version installation does not interfere with the conda installation. The quick version installs packages without rigorously checking for inter-compatibility and is therefore faster to install. However, conda install ensures that all packages are compatible and is therefore slow to install.

```
# Quick Install                          ||     # Conda Install
$ chmod +x install_pes2mp_quick.sh       ||     $ chmod +x install_pes2mp.sh
$ ./install_pes2mp_quick.sh              ||     $ ./install_pes2mp.sh
```

Listing 3.1. Two options for PES2MP installation.

4. Adding pes2mp command to bashrc/bash_profile (Optional):
   After installation, the command to run the program can be shortened by adding a few lines in bash_profile (Mac OS) or bashrc (Ubuntu). This is not essential, but it makes the program more user-friendly.

```
# Mac OS                                 ||     # Ubuntu (Linux):
$ open -t ~/.bash_profile                ||     $ gedit ~/.bashrc
```

Listing 3.2. Commend to open bashrc file for editing.

   Now the GUI text editor will open the bashrc/bash_profile. Paste the following lines at the bottom of the text file, save, and exit.

```
pes2mp ()
{
          python3 pes2mp.py $1
}
```

Listing 3.3. Script that can be added to bashrc for shortening PES2MP command.

The directions to run the program using the command line interface are provided in the following subsection.

3.1.4 Running PES2MP using CUI

1. **To run the program on Terminal**:
   (a) **Copy** the **pes2mp.py** and **pes2mp_driver.py** files into a folder of your choice.
   (b) Now **copy the input file(s)** that you want to execute (e.g. pesgen1D.py) in that folder.
   (c) Open the **terminal** at the folder (make sure the conda **base** environment is showing).
   (d) **Activate** conda **environment** (pes2mp) using either command depending on installation

```
$ conda activate pes2mp        ||     $ conda activate pes2mp_q
```

   (e) Now type the command in the terminal (e.g. Input file = pesgen1D.py):

```
# Step 4 completed              ||     # Step 4 incomplete
$ pes2mp pesgen1D               ||     $ python3 pes2mp.py pesgen1D
```

   Listing 3.4. Two options for running PES2MP code.

   The first command works only if the bashrc/bash_profile/zshrc is edited. If somehow, the bashrc could not be found/edited, the users can still run the program using the longer script:
   Important: The input file must have the **".py"** extension e.g. plot2D.py. However, while executing (running) the program, simply type **pes2mp plot2D** without the **".py"** extension.
2. Users can keep multiple input files (within the same folder) with the same 'Project_name' variable (the variable is set inside the input files) to execute them in sequence such as:
   (a) PES Generation (Internal Psi4 or External Psi4, Molpro or Gaussian)
   (b) Optional: NN Augmentation and PES plotting
   (c) Optional: Fitting PES into a Function (for good fit, use the same function in the final step)
   (d) Multipole Expansion of PES, and
   (e) Fitting Radial Terms into a function (generates the &POTL file to be used in MOLSCAT).

   After PES generation (in cm^{-1} scale), the subsequent PES2MP input calculations can be carried out in sequence. For external PES/radial terms, users must ensure that the file name and data separator are correct. The variable names for PES/radial terms (and other options) are set inside the respective input files.

The above task can be automated using the GUI interface. Several templates for each collision type (1D/2D/4D) are provided with the PES2MP program. The input file (templates) examples for GUI execution are provided inside the **GUI_examples** folder. The users can simply reuse the folder depending on the task and rename it. The environment name, folder location, and Project name are set in the GUI interface. For more details on using the GUI, see Section 3.2.

3.1.5 PES2MP Input Files, Standalone Python Scripts & Auxiliary Codes

This subsection provides instructions to run auxiliary scripts (Python and bash) for extracting and transforming PES and cross-section data. The instructions to enable GPU acceleration on MacOS are also covered.

PES2MP Input Files:

As discussed earlier, the input files are in Python format and are read by the PES2MP program. To run the PES2MP program, keep the 3 Python files, i.e.:

```
(1) pes2mp.py, (2) pes2mp_driver.py and (3) $inputfile.py
```

in the same folder. After which follow Step 1 of the previous subsection to activate the conda environment for pes2mp and run the program.

Auxiliary Codes:

For calculating the PES using external programs like Molpro, Psi4, and Gaussian, the batch scripts are provided in **/input_files/aux_scripts/** folder. Similarly, once the calculations are finished, a Python script to collect the results (in the required format) is provided as well.
To run auxiliary scripts, use the following commands:

```
Python scripts : $ conda activate pes2mp
                 $ python3 script.py

Bash scripts   : $ chmod +x script.sh
                 $ ./script.sh
```

- Apart from PES calculations and results, there are similar scripts to generate **MOLSCAT** input files, run them, collect results (cross-sections), and calculate **rate coefficients** for excitation and de-excitation transitions.
- There is also a Python script to convert energy from Hartree to cm^{-1} (which takes the asymptotic energy of each angle to **remove size-consistency error** as faced by CI and F12 methods).

GPU acceleration for NN model training (on MacOS only): The script to install required libraries is provided in the quick version of the PES2MP install. Uncomment the same and the program will automatically use the available Apple Silicon GPU for training the NN model with $2-10\times$ speed gain (no separate command or modification of code is needed).

```
# To use GPU acceleration in MacOS, uncomment the lines below
python -m pip uninstall tensorflow      # removes standard TF
python -m pip install tensorflow-macos  # installs macOS-specific TF
python -m pip install tensorflow-metal  # installs TF support for Apple Silicon GPU
```

3.1.6 Installing Psi4 Software (Standalone)

The PES2MP code has implemented the Psi4 as **API (library)** using the conda installation method. The Psi4 library is relatively stable, except for rare occurrences, where failed calculations (and other numerical errors originating from the same) caused the program to crash and not continue with the loop (even with a try-except block), while generating PES. The other shortcoming of the API is the problem with CPU parallelization, where the current Psi4 API (internal) calculations do not seem to parallelize well.

The users can, therefore, install the independent **Psi4 software** (to run expensive high-level calculations externally), which does not suffer from these bugs and has an efficient parallelization of *ab initio* methods. Alongside this, the Psi4 software still comes with the ability to use Pythonic scripts (*Psithon*) and inbuilt Python libraries. Its many advantages also include the fact that it is open source (free), developer-friendly, and receives continuous updates.

1. Goto Psi4 website.
2. Click on **"Downloads"** and select OS. Use the default installer (named 'installer'), Python version (currently 3.10), and release (stable). Use the on-screen commands:

```
# Comment: Open the terminal and enter the command after `$' in sequence.

# Comment: Downloading the installer file (check website for newer version)
 1. $ curl "http://vergil.chemistry.gatech.edu/psicode-download/Psi4conda-1.9.1-py310-Linux-x86_64.sh" -o
     Psi4conda-1.9.1-py310-Linux-x86_64.sh --keepalive-time 2
# Comment: Installation
 2. $ bash Psi4conda-1.9.1-py310-Linux-x86_64.sh -b -p $HOME/psi4conda
# Comment: Addi Psi4 to bashrc (and/or other terminal env. depending on the OS, e.g., new MacOS uses zshrc)
 3. $ echo $'. $HOME/psi4conda/etc/profile.d/conda.sh\nconda activate' >> ~/.bashrc
# Comment: Adding Psi4 to tcshrc
 4. $ echo "source $HOME/psi4conda/etc/profile.d/conda.csh\nconda activate" >> ~/.tcshrc
# Comment: Open another terminal so conda and psi4's paths are loaded. Run the command given below to check if
     psi4 is working.
 5. $ psi4 --test

```

Listing 3.5. Installing Psi4 Software

The **Psi4conda-1.9.1** is the current stable Psi4 version (1.9.1) and **py310** signifies that it is based on Python version 3.10. For newer versions and installation changes, refer to the website.

3.2 Installation and Usage using GUI

After downloading and extracting the zip file for PES2MP from GitHub, run the following command to open the PES2MP installer.

```
$ python3 installer_gui.py
```

This will open the following window:

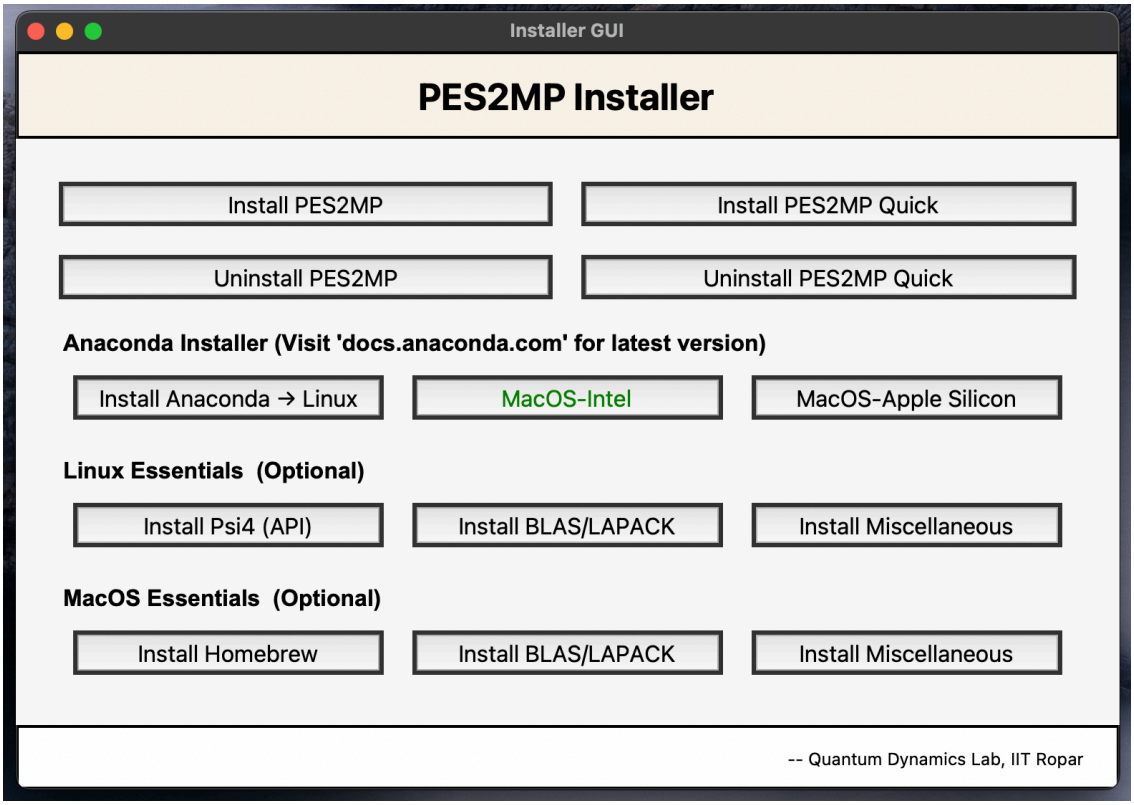


Figure 3.1. GUI Installer

- Select the 'Install Anaconda −>' option to begin the installation process. Choose either Linux, MacOS (Intel), or MacOS (Apple Silicon) based on your hardware and OS. Once clicked, a new terminal will open and install Anaconda. Follow the on-screen instructions and select the appropriate options to finish the installation. Once installation is done, close the terminal.
  # The scripts for anaconda installs are present in *'/makefiles/anaconda_install'* folder. Users can also update the script for installing any newer version of Anaconda if the one present in the script is no longer maintained.
- Once Anaconda is installed, click on either the 'Install PES2MP' or 'Install PES2MP Quick' option. A new terminal will open and the scripts automatically create environments and install the required Python packages needed for the PES2MP program. Follow on-screen commands, and once the installation is finished, close the terminal. *To learn more about the difference between the two installation options, see CUI installation.*

`(OPTIONAL INSTALLS)`

- Linux users can also install Psi4 (Standalone application for external calculations) directly from here. For MacOS visit Psi4 website.

- The BLAS and LAPACK libraries (along with gfortran for Linux and gcc for MacOS) can be installed using the on-screen command. They are needed for installing the MOLSCAT program. *(* **For MacOS users only** :: Homebrew an essential package manager for installing any library in MacOS. Therefore MacOS users must install Homebrew before installing BLAS/LAPACK or Miscellaneous!)*
- The miscellaneous packages contain several well-known packages like htop, cmake, etc. Users can open the *'/makefiles/macos(linux)_essentials'* folder to check the packages in *install_etc.sh* file.

Once PES2MP is installed, close the GUI installer. Type the following script to open the PES2MP GUI program.

```
$ python3 pes2mp_gui.py
```

This will open the following window:

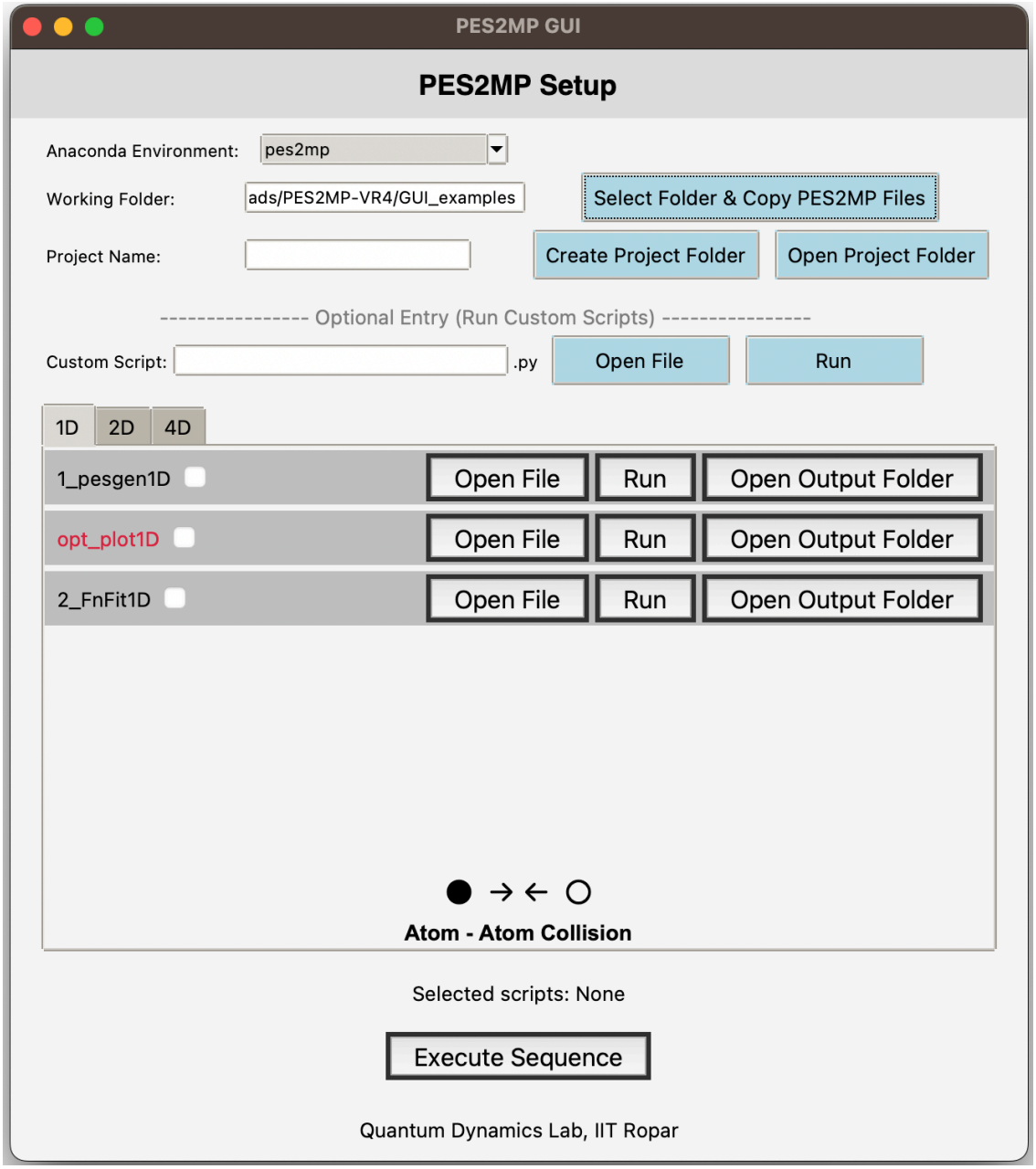
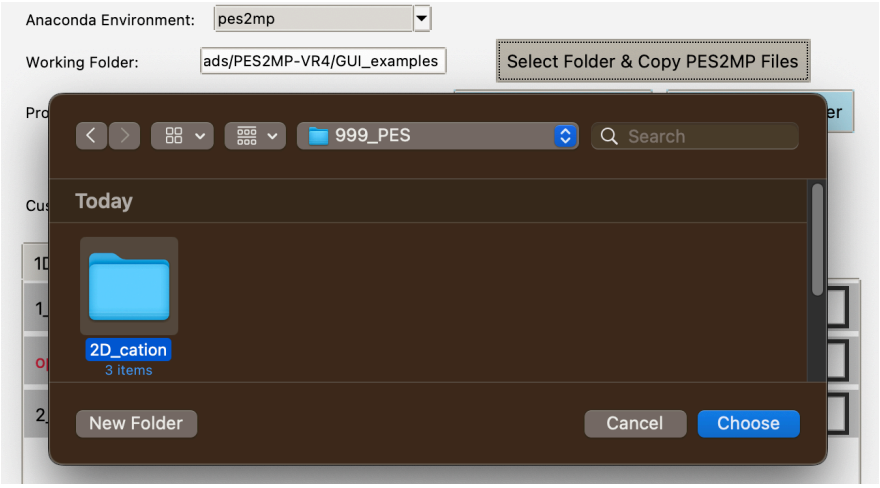


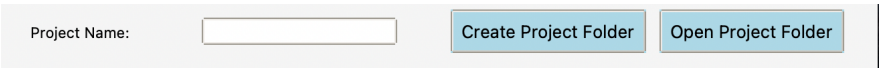
Figure 3.2. GUI Interface for PES2MP

- Select the Anaconda Environment −> pes2mp or pes2mp_q (for quick install version).

- Select the folder where input files for each module (1_PESGen1D.py, 2_FnFit1D.py, etc.) are located. ** The default directory is /GUI_examples/.*



- The GUI interface automatically copies the *pes2mp.py* and *pes2mp_driver.py* files in the selected folder. If files already exist, the user is asked if they want to overwrite the files.
- Enter a Project Name such as '*testrun1*'. The separate project name ensures that output files from different runs are not overwritten. The output files are located inside *$Working_Folder/Projects/$Project_Name*. The Project folder can be created and opened using the following commands:

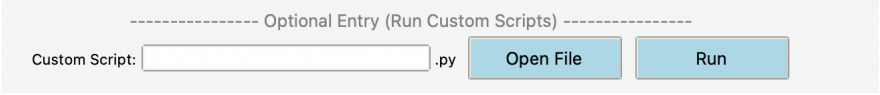


- Users can open the input file by clicking on '*Open File*' to make changes in the system or associated parameters. To see output files, users can click on '*Open Output Folder*'

  How to use input files provided in */input_files/* folder with GUI

  The input file names are tied to the GUI program, i.e., they cannot be changed (as the GUI uses these names to distinguish between different input files). The example files for each 1D, 2D, and 4D collision with their numbered (1_PESGen1D.py) and optional files (opt_plot2D.py) are provided inside */GUI_examples/* folder. To run custom scripts with non-standard names, use the 'Optional Entry' option explained below:

- If there are non-standard scripts, i.e., Python input files not shown in 1D/2D/4D Tabs (e.g., different scripts for creating PES files), they can be run using the optional entry shown below:

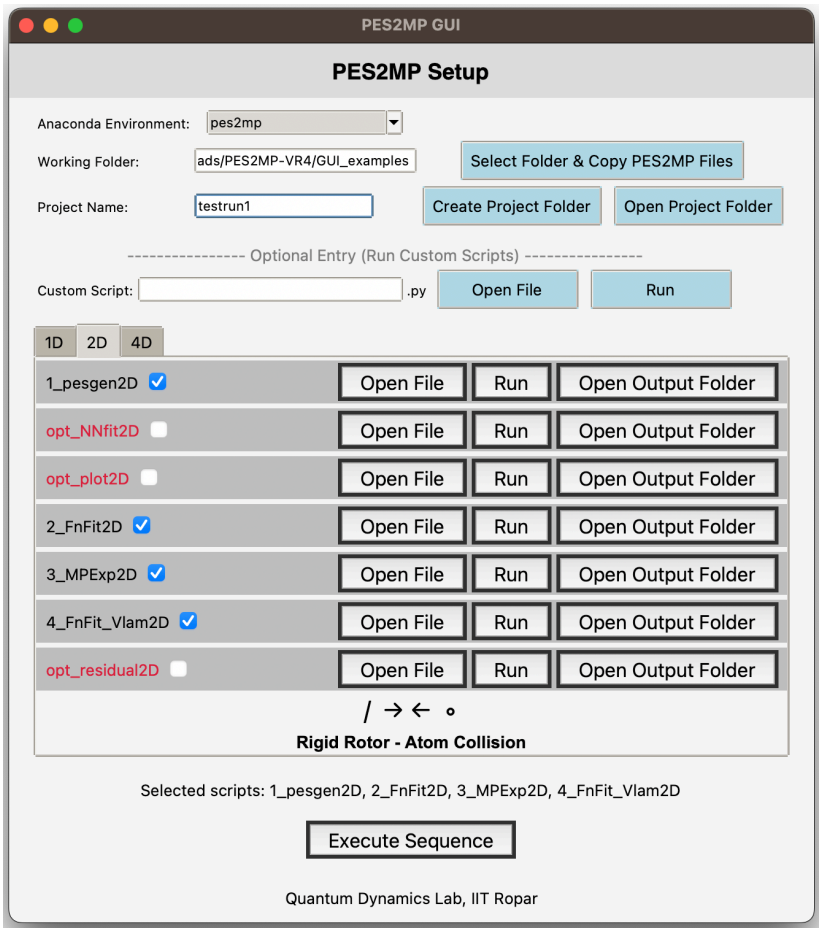


- Also, running the program using CUI takes the project name from the input file, whereas the GUI on the other hand, takes the project name in its interface. Therefore, users who want to use input files provided inside the '*/input_files/*' folder with the GUI must keep in mind the following changes:

  1. Users can either rename the required input files or run the same using the custom script option.
  2. The only change inside the GUI input file is the project name:

```
# CUI Project name CUI simply takes the project name in the input file
Proj_name  =  'testrun1_C2_He'  # This line has to be replaced!):
#-------------------------------------------------------------------------------#
# Replace the above code with these two lines (Project name is read from GUI interface):
import os
Proj_name  =  os.getenv("Proj_name", "default_project_name")
#-------------------------------------------------------------------------------#
```

- Users can run the individual module using the 'Run' command or by selecting the tick option and running multiple modules in sequence. The sequence is shown above the 'Execute Sequence' button. If nothing is selected, the 'Selected script:' shows 'None'.



- Clicking on different tabs selects different collision types, e.g., 2D and 4D collisions. The files for different collisions (1D/2D) can be placed in the same folder, as different dimension files use different extensions, i.e., 1D, 2D, and 4D.

Summary:

With the **GUI** interface, users can **automate**/perform following **actions**:

- Select anaconda environment: *pes2mp* or *pes2mp_q*. The *pes2mp_q* environment can be modified to use GPUs for training the NN model.
- Select working directory (folder): the PES2MP executables, i.e., main python script (*pes2mp.py*) and the driver script (*pes2mp_driver.py*) are automatically copied into that directory.
- Enter the project name (the same will be used for all input files); create and open the project directory to check results, add external files, etc.
- Run custom scripts (input files): E.g., input file for generating external PES.
- Open input files; run PES2MP calculation with that input file; and open the folder to check results.
- Select/deselect scripts to run input files in sequence.

Additional Information:

- The 1D/2D/4D tabs keep the GUI interface organized.
- The GUI program can run the individual modules of PES2MP like PESGen, FnFit, etc. either individually or in sequence as desired.
- If a module in sequence run fails (due to erroneous input), the previous module outputs are preserved. Therefore, the program can be restarted from the failed module in the next run.
- Since the input files are written in Python, users can use its format (e.g. # for comment) and packages (like os, NumPy, etc.) as needed.
- The Psi4 package used internally is installed with PES2MP (no separate installation is needed).
- The GUI generates a log file inside */Projects/Proj_name* folder for each run.

Creating clickable apps for installer and PES2MP:

MacOS: Any script can be automated and saved as an app using the Automator app in MacOS. This section is added to increase the productivity of the users and to allow portability for teaching and demonstrations. Users can create clickable apps using the following steps:

- Open Automator app on Mac. Select new->application
- Double click on `'Run Shell Script'` and copy the provided commands:

```
app_path=$(mdfind "kMDItemKind == 'Application' && kMDItemDisplayName == 'pes2mp.app'")
cd $app_path/../
source ~/.zshrc
python3 pes2mp_gui.py
```

- Save in pes2mp-main folder alongside pes2mp.py files
- For installer: create a new app using the same steps above but:
  - replace `'pes2mp.app'` with `'installer.app'`, and
  - replace `python3 pes2mp_gui.py` with `python3 installer_gui.py`

Ubuntu:

- Search `which python` in terminal (preferably after installing anaconda).
- Copy the output location to top of the `pes2mp_gui.py` and `installer_gui.py` files as:

```
#!/home/hell/anaconda3/bin/python
```

- Open terminal and type `'chmod +x pes2mp_gui.py'` and `'chmod +x installer_gui.py'`
- Simply right click on file and use `'Run as Program'`

3.3 GUI Examples

The following calculations (with input and output files) are provided in the GUI_examples folder. The same is provided for users to run the scripts and get familiar with the input files of PES2MP. Here, we shall briefly discuss the input and output files for each case. For full details, see the next section of the manual.

3.3.1 Practice Input Files

1. 0_ExtPES: Generating 1D, 2D, and 4D PES files for external calculations (on a server).
2. 1_NoFitPES: Generating 1D, 2D, and 4D PES files internally using Psi4:
   (a) 1D: Input files for handling (i) an anion, (ii) a cation, and (iii) a non-singlet (UHF) species. It also has examples for generating (iv) BSSE corrected and (v) CBS extrapolated PES.
   (b) 2D: Example for generating a (i) rough PES of C_2-He collision, (ii) expansion into radial terms, and (iii) fitting into mathematical expression (also generating MOLSCAT's potential (&POTL) file).
   (c) 4D: Example for generating a (i) rough PES of C_2-H_2 collision, (ii) expansion into radial terms, and (iii) fitting into mathematical expression (also generating MOLSCAT's &POTL file).
3. 2_FnFitPES: Generating 1D, 2D, and 4D PES files internally using Psi4 and fitting them into a mathematical expression (series of Slater functions) to interpolate or extrapolate missing data points. This example also has results for MOLSCAT output with rate coefficients.
   *The Slater functions ($\alpha e^{-\beta R}$) used for fitting PES and radial terms have the same β values:
   (a) 1D: Input files for handling (i) C-He collision, (ii) re-plotting PES with different ranges, texts, plot parameters, or file name extensions, and (iii) fitting the PES into four simple mathematical expressions.
   (b) 2D: Example for generating (i) a rough and a high-level PES of C_2-He collision, (ii) fitting into a mathematical expression, (iii) expansion into radial terms, and (iv) using the same mathematical expression (also generating MOLSCAT's &POTL file).
   (c) 4D: Example for generating a (i) rough PES of C_2-H_2 collision, (ii) fitting into a mathematical expression, (iii) expansion into radial terms, and (iv) using the same mathematical expression to fit radial terms (also generating MOLSCAT's &POTL file).
4. 3_NNFitPES: Generating sparse (angular terms) 2D and 4D PES files internally using Psi4 and then fitting into a PES-specific ensemble NN model.
   *The NN model has custom activation functions (negative GELU and Gaussian), and a constraint on R (decay function) to ensure desirable physical behavior at very-short and long-range regions:
   (a) 2D: Example for generating (i) a sparse PES for C_2-He collision, (ii) fitting into the NN model, (iii) fitting NN augmented PES into analytical expression and expansion into radial terms, and (iv) using the same expression to fit radial terms (also generating MOLSCAT's &POTL file).
   (b) 4D: Example for generating (i) a sparse PES for C_2-H_2 collision, (ii) fitting into the NN model, (iii) fitting NN augmented PES into analytical expression and expansion into radial terms, and (iv) using the same expression to fit radial terms (generating MOLSCAT's &POTL file).

Usually, fitting the PES into a function can have several advantages, such as saving disk space (coefficients along with the code take much less space than a full PES), and application in quantum dynamics calculations. The time and effort required to find a suitable function that fits a PES can improve the dynamical results of underlying rotational and vibrational states. Therefore, we shall now see a few test cases to understand how a suitable PES can be generated/fitted for our application. In the previous examples, the users saw how a rough PES can save them a lot of time before starting a high-level calculation. So, we shall now proceed to show a few test cases considering the same with 1D files:

3.3.2 999_PES cases:

Here we shall see why generating a physically correct PES can be challenging and how to generate a PES from scratch using Psi4 with PES2MP that can be used for further applications.

Fitting 1D PES into mathematical expression $f(R)$

There are three cases: 1D_anion, 1D_cation, and 1D_neutral. In each case, users can see three different folders inside */Projects/*, which are (a) *rough*, (b) *cbs*, and (c) *fit*.

- In *rough/* folder, a rough PES is calculated using HF-D4/cc-pvdz.
- In *cbs/* folder, PES is generated using a CBS method in Psi4: 'sherrill_gold_standard' (SGS).
- In *fit/* folder, the *ab initio* PES file (in cm^{-1}) is copied from the *cbs/* folder and fitted into (a) an analytical expression that fits the full range of minima and asymptotic region, and (b) custom functions that fit high-energy ($\alpha e^{-\beta R}$) and long-range ($\gamma R^{-\kappa}$) regions for correct physical behavior.

Fig. 3.3 shows the ab initio PES, the function-fitted (also referred to as custom function fitted in the manual) PES, and function-fitted PES with constraints on high-energy (HE) and long-range (LR) regions (HELR PES).

The function (custom-function) used to fit the full range of PES (for anion, cation, and neutral collision) is:

$$a_1e^{-5x} + a_2e^{-3.75x} + a_3e^{-3x} + a_4e^{-2.75x} + a_5e^{-2x} + a_6e^{-1.75x} + a_7e^{-1x} + a_8e^{-0.75x} \tag{3.1}$$

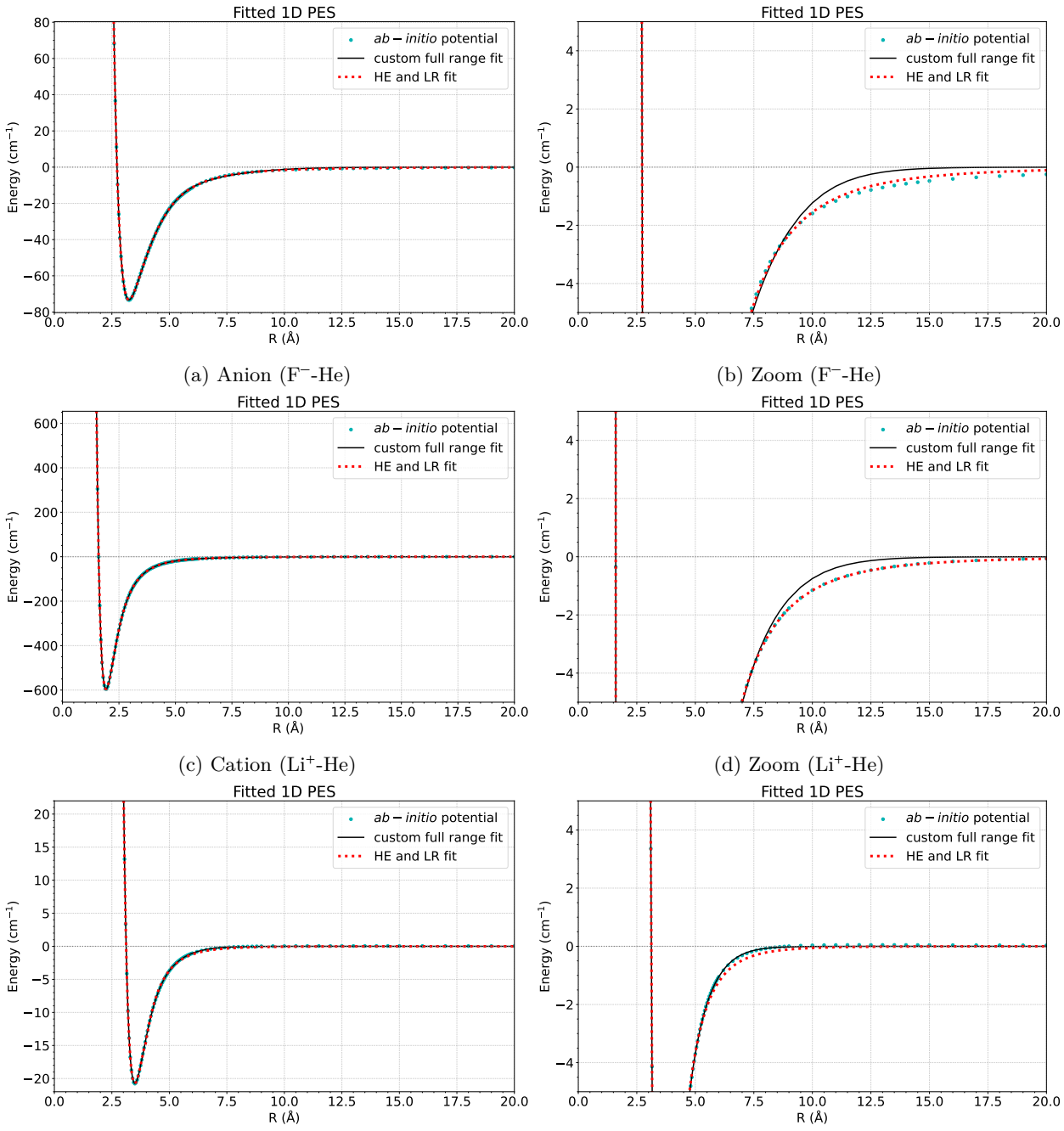
which is provided in the input file as:

```
cutoff = 300        # Energy cutoff in cm-1. PES features are preserved till the cutoff

def fnfit_custom(x, a1,a2,a3,a4,a5,a6,a7,a8):
    import numpy as np
    return  a1*np.exp(-5*x)+a2*np.exp(-3.75*x)+a3*np.exp(-3*x)+a4*np.exp(-2.75*x) + \
            a5*np.exp(-2*x)+a6*np.exp(-1.75*x)+a7*np.exp(-1*x)+a8*np.exp(-0.75*x)
initial_val  = [1e4]*8                    # Enter initial guess
```

where the mathematical expression is passed as a function and its cutoff (trims very high energy points) and initial guess are passed separately. (*An educated initial guess ensures convergence!)

As we can see in Fig. 3.3, the function provides a good overall fit but may deviate slightly when zoomed in to the spectroscopic scale. For this, users can use HELR fit as shown below.

Apart from fitting the full PES range using Slater functions, the high-energy (HE) and long-range (LR) regions can be further constrained to follow a certain shape. For this, physically correct functions are used for a certain range of *ab initio* data. The fitted outputs of HE-LR regions are then merged with the previously fitted data (in the minima region, as there may be missing data points) to give the final PES. An example function for each (HE and LR) is provided below.



(a) Anion (F^--He)

(b) Zoom (F^--He)

(c) Cation (Li^+-He)

(d) Zoom (Li^+-He)

(e) Neutral (Ar-He)

(f) Zoom (Ar-He)

Figure 3.3. Fitting results of fitting a 1D collisional PES both in full range and zoomed in. The blue dots are *ab initio* data points (SGS), the black line represents the custom function (Eq. 3.1) that fits the full range of the PES till cutoff, and the red dotted line represents the high-energy and long-range fitted PES (HE-LR) where the minima region is fitted using the custom function (Eq. 3.1) while the HE and LR regions are fitted using Slater ($\alpha e^{\beta R}$) and inverse power ($\alpha R^{-\beta}$) functions, respectively.

```
#-------------------------------------------------------------------------#
he_fit = True           # To fit high Energy region (Set True)
he_cutoff = 10000       # end value: fit data points till he_cutoff (cm-1)
he_min_offset = 4       # start value: fit from 4 positions before minima

def fnfit_he(x, he1,he2):             # Function for fitting high energy Region
    import numpy as np
    return  he1*np.exp(-he2*x)
he_initial_val  =  [1e4,1]            # Enter initial guess
#-------------------------------------------------------------------------#

#-------------------------------------------------------------------------#
lr_fit = True           # To fit Long Range Region (Set True)
lr_min_offset = 40      # fit from 80 positions before end value

def fnfit_lr(x, lr1,lr2):              # Function for fitting Long Range Region
    import numpy as np
    return  lr1*np.power(x,-lr2)
lr_initial_val  =  [1e4,4]             # Enter initial guess
#-------------------------------------------------------------------------#
```

The high-energy region is approximated using a single Slater function where both coefficients are optimized. Since we have two cases, a charge–quadrupole interaction (follows R^{-4} function) and London dispersion (van der Waals' follows R^{-6} function) at long range, we can either set them to exact values as shown below:

```

def fnfit_lr(x, lr1):              # Function for fitting Long Range Region
    import numpy as np
    return  lr1*np.power(x,-4)
lr_initial_val  =  [1e4]             # Enter initial guess
```

or allow them to relax with 4/6 as the initial guess (as in the previous case). The previous case is important if several long-range interactions average out (happens for H_2 collisions) to give an effective decay function.

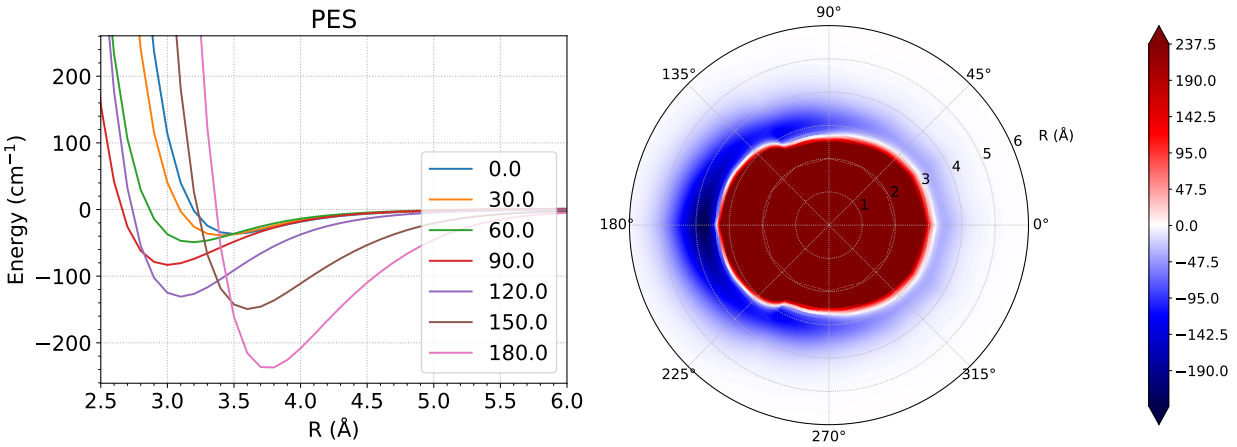
The high energy cutoff (*he_cutoff* value) can be used to set an upper limit for HE fit (in cm^{-1}). The program will fit the PES from the offset, i.e. N^{th} point before minima (for each angle) till the *he_cutoff*.

For long-range, only the offset value is used, which indicates that fitting will begin from N^{th} value before the final (R) data point (for each angle in case of 2D/4D). For example, if the offset is 40 and there are 140 *ab initio* data points in an angle, the fitting will start from (140-40) 100^{th} data point till the last point.

Fitting non-symmetric 2D PES (HCO^+-He) into mathematical expression $f(R)$

In this example, the 2D PES generation and fitting the same into a mathematical expression are discussed. In an atom-atom collision, it is rare to find a repulsive potential after the minimum, however, the same is not true for a cationic molecule collision with He/H_2. In such cases, it is recommended to select LR extrapolation very carefully or not to use the same at all. Instead, the slow decay can be approximated in the full range itself by using an additional Slater function with a small β $(0.2-0.01)$ coefficient.

Before proceeding with full PES calculation, a rough PES is generated in the intended range of R, θ and plotted to see if any additional data points will be needed.



(a) (b)

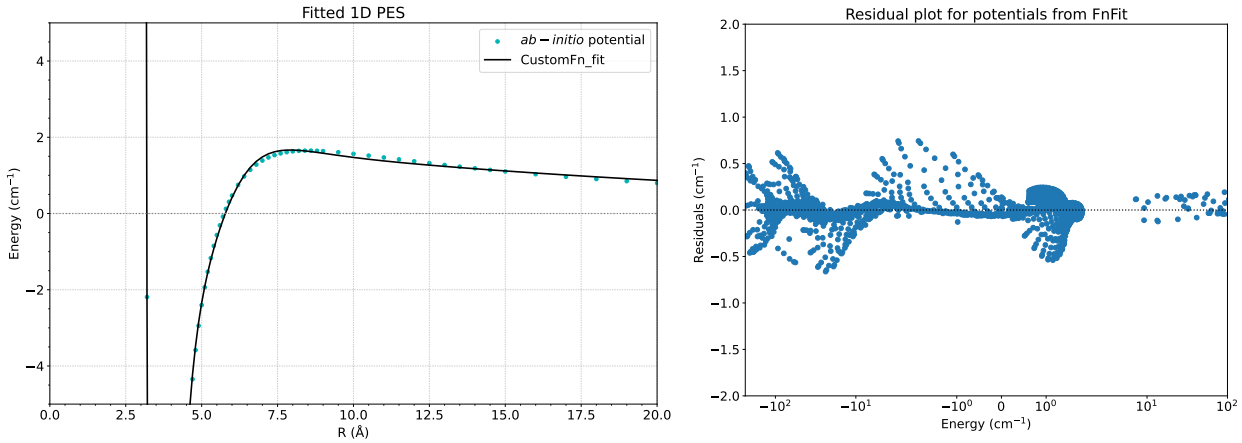
Figure 3.4. Rough PES (a) *R vs E* plot at various angles θ (°) and (b) polar plot for HCO^+-He collision calculated using Psi4 (HF-D4/cc-pvdz).

As seen in Fig. 3.4, the global minimum is at 180°. While the range is good for 0° and 180°, the potential at 90° and 2.5 Å is very close to 0 cm^{-1} which registers a maximum of $\backsim$ 150 cm^{-1}. Therefore, in the high-level *ab initio* PES, the range must be changed from 2.5 Å to 2.0 Å.

As discussed, there is a small repulsive region before He reaches the minima due to electrostatic repulsion. This can cause problems in long-range fitting. Since the *ab initio* range spans from 2.0 Å to 100Å, the long-range region of PES does not need extrapolation, and using an incorrect long-range fit with R^{-x} can change the shape of this region. Therefore, to preserve this feature, only high-energy fitting is done, keeping LR fitting 'False'.

Why is high-energy fitting needed? Due to a large number of Slater functions in the fitting function, the custom-fitted PES may go to $-\infty$ at $R \to 0$, which is corrected by HE fitting. Both custom-fitted and HE-fitted PES are located inside *Projects/Proj_name/PESFnFit/* alongside the original PES and residual data (*Fnfit_PES.dat, HELRfit_PES.dat, Original_PES_sort_cm.dat, residuals_Fnfit_PES.dat*).

The custom function is similar to one in Eq. 3.1 where the last β coefficient is changed from -0.75 to -0.05 to capture the slow decaying tail of this repulsive PES. The PES is then corrected for the high-energy regions, and residuals are calculated to obtain fitting error. The plot shown in Fig. 3.5(b) can be obtained from the *Projects/Proj_name/PESFnFit/Plots* folder.



(a) (b)

Figure 3.5. (a) PES at $\theta = 0°$ showing repulsive behavior for HCO^+-He collision. (b) The residual plot (excluding very high energy regions) shows the fitting error (into Slater functions) for the PES at various energy regions.

*__*There is no separate residual plot for custom and HE-fitted PES?*__ Since HE-Fit uses the custom-fitted data after the cutoff, the residual plot in the minima region remains the same for both. If needed, users can modify the existing code and plot the same.

Overall, the users have several options to generate and clean the PES. If a PES has no missing data points, users can directly proceed with MP expansion and fit them into a suitable function. Otherwise, users can fit the PES into functions of R and get the required missing data. Here, they can choose to either use (a) a custom-fitted function or (b) a custom-fitted function with special constraints on high-energy (HE) and long-range (LR) regions. Finally, for heavier molecules, users may opt for NN augmentation (of angles and R) using a sparse *ab initio* PES and then proceed as required.

The input file for generating an ensemble NN model to augment PES (both radial and angular parts simultaneously) is discussed in the upcoming subsections.

3.4 Input Files Guide

To understand the workings of the PES2MP program, we shall first see the example input files provided with the program in the folder '*/input_files/*'. The input files for PES2MP are written to enable users to invoke/run several modules in sequence or in parts as needed. We shall first discuss how to run each module individually. Ultimately, we shall see how a combined input file can automate several processes.

*** Only the variables in the input files are discussed here. The output of each module/input file is discussed in the upcoming Section 4, i.e., Results and Discussion.**

3.4.1 General Commands

The most important part of the input file is 'Proj_name'. All the input, output, and temporary files for any specific calculation are stored in *Projects/'Proj_name'*. The type of calculation is mentioned using the unique 'job type' keyword, such as 'Create_PES_input' to invoke the PESGen module.

```
############################  Global Options        ############################
Create_PES_input = True       # Set JOB Type(s) (True/False)
#-------------------------------------------------------------------------------#
Proj_name        = 'pesgen1D' # Set a folder (Project) name
#-------------------------------------------------------------------------------#
```

Listing 3.6. Setting global options in the input file.

For GUI run, the *Proj_name* variable is set inside the GUI interface, where the input files will look like:

```
#-------------------------------------------------------------------------------#
import os     # Project name is read from the GUI interface
Proj_name  =  os.getenv("Proj_name", "default_project_name")
#-------------------------------------------------------------------------------#
```

The idea is that all the calculations (e.g. PES Generation, NN augmentation, and MP expansion) for a specific system (e.g. C_2-He) can be done by placing different input files in the same folder, along with the *PES2MP.py* and *PES2MP_driver.py* files.

All of the input files must have the same '*$Proj_name*' as the program automatically places the required output data files in *Projects/$Proj_name/* for the subsequent calculations as shown in Fig. 3.6.

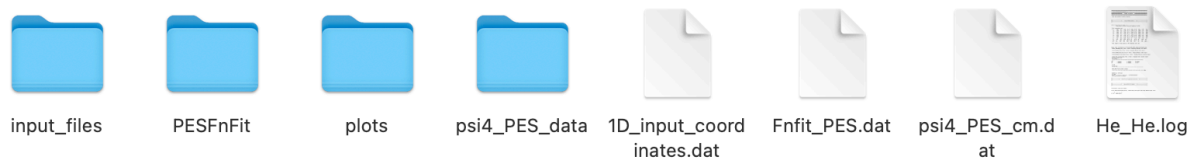


Figure 3.6. Screenshot of the project folder with name '**He_He**' containing folders for: PES generation (internal: '*psi4_PES_data*' and external: '*input_files*'), PES analytical fitting: '*PESFnFit*', and PES plots: '*plots*'. The folder also contains data files for coordinates, *ab initio* PES, and function fitted PES, along with the log file.

For example, the '*Create_PES_input*' with internal Psi4 calculation will create a folder for plot and temporary files inside *Projects/$Proj_name/input_files*. The final generated PES file will be copied outside, i.e., at *Projects/$Proj_name/* folder, where it can be read by the next calculation file, e.g., '*Create_NN_model*', '*FnFit*', etc. Similarly, the plot and other output files for PES-fitting calculations will be placed inside *Projects/$Proj_name/PESFnFit* folder.

3.4.2 *PESGen*: Generating collisional PES

Atom/Rigid Rotor Information

The program, as discussed in the introduction, can generate 1D, 2D & 4D PES using the PESGen module. The basic input for entering colliding atoms/molecules is provided below:

```
############################  Global Options        ############################
Create_PES_input = True       # Set JOB Type(s) (True/False)
#-------------------------------------------------------------------------------#
Proj_name        = 'H-H' # Set a folder (Project) name
#-------------------------------------------------------------------------------#
# Enter parameter for the first atom
RR1_atoms        = ['H']      # Enter atom 1
# Enter parameter for the second (collider) atom
RR2_atoms        = ['H']      # Enter atom 2
#-------------------------------------------------------------------------------#
# Enter charge & multiplicity (Do NOT change order)
Charge           = [0, 0, 0]  #       charge of [atom 1, atom 2, whole system]
Multiplicity     = [2, 2, 1]  # multiplicity of [atom 1, atom 2, whole system]
#-------------------------------------------------------------------------------#
```

Listing 3.7. 1D PES colliders

The program is structured to seek what job is set and skip jobs that are set as 'False' or not provided at all. As discussed earlier, the *'Create_PES_input'* directs the program to look for parameters and generate PES. The *'Proj_name'* variable must be kept unique to separate each project, as the program creates the folder of the same name inside the *'Projects/'* folder.

The atom/molecule pair is described via *'RR1_atoms'* and *'RR2_atoms'* lists (string format). The charge and multiplicity of each fragment and whole system are defined by '*Charge*' and '*Multiplicity*' lists (integer).

For a 2D/4D collision, the users must provide experimental or theoretical bond lengths (optimized) *via* *'RR1_bond_len'* and *'RR2_bond_len'* lists (float format). Make sure that the lengths are provided from left to right (i.e., same ordering as atoms), e.g., HCN molecule with H-C bond: 1.064 and C-N bond: 1.156. For a 2D collision, the *'RR2_bond_len'* list is not needed as the collider is just an atom.

```
# Enter parameter for RR1 : Enter experimental/theoretical bond length(s)
RR1_atoms        = ['H','C','N']   # Enter rigid rotor (RR1) atoms from one end
RR1_bond_len     = [1.064,1.156]   # Enter bond lengths from the same end

# Enter parameter for the second (collider) atom
RR2_atoms        = ['He']  # Enter atom(s) for RR2
```

Listing 3.8. 2D PES colliders

```
# Enter parameter for RR1 : Enter experimental/theoretical bond length(s)
RR1_atoms        = ['H','C','N']  # Enter rigid rotor (RR1) atoms from one end
RR1_bond_len     = [1.064,1.156]   # Enter bond lengths from the same end

# Enter parameter for second rigid rotor (RR2)
RR2_atoms        = ['H','H']  # Enter atom(s) for RR2
RR2_bond_len     = [0.7667]   # Enter bond lengths from the same end for RR2
```

Listing 3.9. 4D PES colliders

Note: If vibrationally averaged bond lengths (r_o) are available (e.g., 0.7667 for H_2), the same is recommended as it should generate a PES which would produce dynamical results closer to experimental rates (for rotational transitions). In the absence of the same experimental equilibrium bond lengths (r_e) or theoretically optimized bond lengths (ideally using high-level *ab initio* methods: coupled cluster with triple+ zeta basis) should be used.

Isotopes (Optional parameter)

The use of isotopes would change the position of the center of mass (COM) for the rigid rotor. For light molecules, H_2 *vs* HD/HT, this will significantly shift the PES. However, for heavy molecules, the shift in the PES is minimal. The effect of isotopes is widely studied for rotational transitions by changing the rotational constant of the isotope in the MOLSCAT software.

Since the PES2MP software automatically calculates the COM of the rigid rotor for computing the Jacobi coordinates, using isotopes will automatically incorporate the effect of change in COM.

```
#--------------------  Use Isotope (Optional)  -------------------------------#
#Use_isotope = True # only for calculating isotopic dependent COM (PES is same)!
#RR1_isotope_mass = [0, 13.00335483507]     # Enter isotope mass  (0 = default)
#RR2_isotope_mass = [0,  3.0160492779 ]     # Enter isotope mass  (0 = default)
```

Jacobi Coordinates

The Jacobi coordinates for 1D/2D and 4D must be specified using the commands below. The Radial coordinates are needed for all three collisions and are provided as:

```
# Radial coordinates (enter as many ranges and step sizes as desired)
R_i   = [0.1,  2.0,  6.0,  9.0, 15.0,  50.0] # initial R in Ang. (eg. 0.1 Ang.)
R_f   = [2.0,  6.0,  9.0, 15.0, 25.1, 100.1] # final R - R_stp (eg 1.95 Ang.)
R_stp = [0.05, 0.1,  0.2,  0.5,  1.0,  50.0] # step size for R_i-R_f
# Loops run from initial to penultimate value (eg, loop 2-6 goes till 5.9 Ang.)
```

Listing 3.10. 1D/2D/4D radial collisional coordinates

In the above files, the length of all three lists (float) must be the same. The R_i denotes the starting value of R, the R_f denotes the final value, and R_{stp} is the step size. Since the Python loops run from the initial value to the penultimate value, the R values will be generated in multiples of R_{stp} for values smaller than R_f. For example, in the above set of data, the R-value will start from 0.1, will go to 1.95, and stop, then the second column will start creating R values and will start from 2.0 to 5.9. Subsequent columns will also generate files with varying R values as described. In the final column, R will go from 50.0 to 100.0 since the final value is 100.1 (which is slightly larger than 100) and the step size is 50.

The method is followed to allow for variable step size in R values since the PES slope is much greater towards minima, which decreases towards infinity. The R values provided are a good starting point for a rough PES calculation (using HF/DFT). Once the same has been analyzed, the R lists can be modified to get denser/sparser coordinates as needed.

Similarly, in 2D collision, the R coordinates are described as stated above, and the angle of collision is described by the *theta* list (integer) with 3 values (initial, final, step size).

```
# 2D Angular template [theta] (to include the final value, i.e., 90, use 91)
theta  = [0, 91, 30]    # Python loop will run from 0 to 90 with a step size of 30
```

Listing 3.11. 2D angular collisional coordinates

The program simply generates PES by varying theta from 0° (first value) to 90° (<second value), with a step size of 30° (third value). The final value is kept at 91 to include 90° in PES, else the loop will run to the penultimate value and will have $0°, 30°, 60°$ only.

Note: For symmetric systems, i.e., C_2, NCCN, etc., generate PES till 90° only. For non-symmetric systems, e.g., CO, CNCN, etc., change the above list to $theta = [0, 181, 30]$, to generate PES till 180°. For symmetric RR, PES for $\theta > 90°$ are reflections of the generated PES ($\theta < 90°$), due to the spatial symmetry of the system.

For a 4D PES, the R coordinates are again similar to 1D collision, but the angles of collision are now described by three lists, one for each angular term. Each list contains 3 values denoting starting value, end value (+1), and step size:

```
# 4D angular [phi, theta2, theta1]  coordinates
phi    = [0,  91, 30]    # phi    : 0 to  90 with step size of 30
theta2 = [0,  91, 30]    # theta2 : 0 to  90 with step size of 30
theta1 = [0, 181, 60]    # theta1 : 0 to 180 with step size of 30
```

Listing 3.12. 4D angular collisional coordinates

where:

1. θ_1 : rotation of RR1 in $y-z$ plane.
2. θ_2 : rotation of RR2 in $y-z$ plane.
3. ϕ : dihedral (angle between planes of RR1 and RR2) change in $x-y$ plane. The same is described by rotation of RR1 in $x-y$ plane as $\phi_2 - \phi_1 = \phi$ (constant) for two rigid rotors (symmetries!)

The coordinate ranges in the above example will be used if both RR1 and RR2 are symmetric. If either of the two is non-symmetric, θ_2's range will double to $0° - 180°$. If both RR are non-symmetric, all three angular terms have the range $0° - 180°$.

Internal Calculation using Psi4

To run an internal rough calculation using Psi4, set $Run\_psi4 = True$, else set $False$. The parameters required for calculating PES are provided below:

```
#--------------------------------------------------------------------------#
Run_psi4 = True     # run internal rough calculation?                       #
#--------------------------------------------------------------------------#
#--------------------------------------------------------------------------#
#---------------           If Run_psi4 == True              ---------------#
#--------------------------------------------------------------------------#
# Use for rough estimation: default parameters (SCF (hf) and cc-pvdz basis)
# The program will run calculations and plot PES(s) in cm-1.
psi4_mem           = '4 GB'             # total memory
psi4_proc          = 1                  # number of processors(doesn't scale)
psi4_reference     = 'rhf'              # Reference WF: rhf/uhf/rohf
psi4_method_basis  = 'scf/cc-pvdz'      # Method/Basis Set
psi4_Frozen_Core   = True               # frozen core: True/False
psi4_bsse          = None               # counterpoise: 'cp', else: None
```

Listing 3.13. Basic parameters for internal Psi4 calculations.

The Psi4 API does not have a good parallelization (at present), and therefore it is recommended to keep '$psi4\_proc = 1$'. The reference can be 'rhf/uhf' and the method/basis can be selected from supported keywords (refer to Psi4 manual). To calculate counterpoise corrected PES, set $psi4\_bsse$ = 'cp', else $None$.

Apart from parameters to calculate PES, other parameters are required to calculate energy at infinite (asymptotic) R ((*R_inf*)), and convert PES to cm^{-1} for plotting. The file name for output PES can also be edited (if required). An example of the plotting parameters for simple 1D PES is provided below:

```
#-------------------------------------------------------------------------------#
R_lim = [2,8]      # Enter R limit [start, end] in Angstroms(Rough PES!)
fmt   = 'pdf'      # Enter plot(s) format: pdf, eps, png, jpeg, etc.
#-------------------------------------------------------------------------------#
R_inf              = 200               # R = 200 Angstrom
PES_filename       = "psi4_PES.dat"    # PES output file (in Hartree)
PES_filename_cm    = "psi4_PES_cm.dat" # PES output file (in cm-1)
#-------------------------------------------------------------------------------#
```

Listing 3.14. Paremeters for generating 1D PES plot.

The plot limit (*R_lim*) and format *fmt* are set to produce high-quality plots. Any additional changes (if required) can be made directly to the plotting function in *PES2MP_driver.py* file. For 2D PES, additional information is needed for the reference angle that will be used for subtracting energy, as provided below:

```
R_inf              = 200               # R @ infinity = 200 Angstrom
theta_2D_inf       = 90                # Angular Coordinate for E_infinity
thetax             = [0,30,60,90]      # Angles to be plotted (R vs E plot)
```

Listing 3.15. Paremeters for generating 2D PES plots (polar and *R vs E*).

While all the available angles are used to generate the PES polar (2D) plot, the *thetax* parameter sets the angle that will be plotted in 1D *R* vs *E* plots. * *Selecting too many angles can make the plot appear cluttered.*

A detailed discussion on the same (along with input parameters for 4D PES) is provided in Section 3.4.3, where a dedicated input file (*plot_ND.py*) is used to plot the PES generated using an external software.

External Calculations using Psi4, Gaussian and Molpro

For setting up calculations on external servers, the following variables can be set to True/False to generate dedicated PES input files (with inbuilt templates). Set the parameter to True and provide the required input for generating the PES using the templates. Usually, Molpro is used for generating high-level PES due to the availability of F12 approximate methods. Therefore, three templates: CP, CBS, and custom input are provided for Molpro. Gaussian is restricted to CP calculations. For Psi4, only a custom template is provided, as it can easily be modified.

PES2MP allows users to generate multiple PES files using a single input file. Each command will create separate PES input files in different folders (*Projects/$Proj_name/input_files/*). PES is created in XYZ format by projecting $R/\theta(s)$ into the three axes.

```
# create input files for external calculation of PES
Create_GAUSSIAN_input_files      = True      # option 1  = Gaussian(CP)
Create_MOLPRO_CP_input_files     = True      # option 2  = MOLPRO(CP)
Create_MOLPRO_CBS_input_files    = True      # option 3  = MOLPRO(CBS)
Create_MOLPRO_custom_input_files = True      # option 4  = Custom MOLPRO input
Create_Psi4_custom_input_files   = True      # option 5  = Custom Psi4 input
```

Listing 3.16. Options for generating PES input files for external calculations.

While the first three options (*CP/CBS*) require minimal editing, the last two options (*custom*) are very important if one needs to perform special calculations or wishes to use variables/tables in the calculations.

While it is possible to use Psi4 internal and for external calculation from the same file, remember that they both use the same '*psi4_bsse*' option. So, if the internal calculation uses CP correction, the external Psi4 files will also be generated for CP-corrected PES. The various templates are discussed in detail below:

Keywords for external calculations

The PES2MP package has the following built-in templates. The final PES for each can be extracted (from output files) using scripts provided in the '*input_files/aux_scripts/PES_extract*' folder.

1. Gaussian CP corrected PES:

```
#-------------------------------------------------------------------------#
#-------------   If Create_GAUSSIAN_input_files == True)   ---------------#
#-------------------------------------------------------------------------#
proc_g = 4            # number of processors (% nprocshared)
mem_g  = '32GB'       # total memory (% mem)
chk_g  = Proj_name    # Checkpoint file (% chk)
                      # Using project name.
                      # Replace with any string '' if needed!
# COMMAND LINE: CCSD(T) / AVQZ with counterpoise correction
cmd_g  = '# CCSD=(T,SaveAmplitudes,ReadAmplitudes)/aug-cc-pVQZ Counterpoise=2'
```

   - *proc_g*: Set processor: CC calculations do not scale well beyond 4-8 processors.
   - *mem_g*: total memory: Set memory based on system availability. Multiprocessor jobs need more memory than single-processor jobs. E.g., suppose 4 GB memory is enough for 1 processor (defined by '*proc_g*'), then use 16 GB for 4 processors and 32 GB for 8 processors.
   - *chk_g* : checkpoint file name: All input files will use $*Proj_name* for checkpoint file. Since the '*chk*' file stores the converged wavefunction, it will speed up the subsequent calculations.
   - *cmd_g*: This is the main command line for Gaussian calculation. The method/basis set and any other parameters for calculations are defined here. The PES2MP automatically assigns the atoms to respective fragments (in the Gaussian input files) for counterpoise calculations.

2. Molpro (General option for the three templates)

```
#-------------------------------------------------------------------------------------------#
# GENERAL OPTIONS MOLPRO (Use double brackets if a single bracket is needed in MOLPRO)
#-------------------------------------------------------------------------------------------#
mem_m      = '1 g'                    # memory per thread (1g = 1 giga words)
run_method = '''{{rhf;rccsd(t)}}'''   # molpro function to declare method and reference WF
```

   Molpro has two fixed templates: (a) CP corrected PES and (b) CBS extrapolated PES. Both use the above command to set the memory and method for calculation. Unlike Gaussian, the number of processors in MOLPRO will be defined in a batch (jobscript) file provided in the */batch_sripts/* folder.

   - *mem_m*: memory per processor: Unlike Gaussian, Molpro memory definition can be tricky as the memory is defined in words (which is system dependent) and not bytes. The memory can be approximated by the formula: $memory \times 8 \times num\_proc$, i.e., $1g \times 8 \times 4 = 32$ GB (for 4 processors). **Real memory will slightly vary depending on the system.*
   - *run_method*: This section contains information about method 'ccsd(t)' and reference 'rhf' wavefunction for Molpro calculations. The triple quotes mean that the input can be multi-line. It is kept this way to allow for wavefunction (WF) cards and other information to be added when

necessary. The double curly brackets (very important) direct Python to put one curly bracket around the command in the molpro input file.

(a) CP corrected PES: simply set a large basis set

```
# MOLPRO_CP (counterpoise corrected)
#-------------------------------------------------------------------------#
basis_cp   = 'AVQZ'                  # basis set for cp calculation
```

basis_cp: Basis set for counterpoise corrected PES. Use AV5Z or AVQZ for benchmarking any specific angle (usually, the angle with the global minimum is always benchmarked). Then for full PES, use AVTZ basis with F12 methods, giving the lowest error w.r.t. CBS or AV5Z(CP) basis.

(b) CBS extrapolated PES: Uses EX1-L3 extrapolation scheme

```
# MOLPRO_CBS (complete basis set limit)
#-------------------------------------------------------------------------#
basis_ref = 'avdz'  # reference basis - NOT included in CBS extrapolation
basis_cbs  = ['avtz','avqz','av5z']  # basis set for CBS extpol. (only 3)
                                     # Uses EX1 and L3 (npc=2) extpol.
```

- *basis_ref*: Basis set reference. A smaller basis set improves convergence for the larger basis.
- *basis_cbs*: The program implements EX1 and L3 functionals extrapolation of reference (scf) WF and correlation (ccsd(t)) energies, respectively. The npc=2 indicates that only correlation energies of the last two basis sets shall be used for extrapolation. By far, this is one of the most stable and suitable functionals for CBS extrapolation. For any other CBS extrapolation method, the custom calculation can be set up using the instructions below, or the function can be edited in the *molpro_driver.py* file.

Custom Calculations! Since Molpro and Psi4 allow variables to be declared in input files, custom templates can be used for the same. The program generates the input file with XYZ coordinates (generated from Jacobi coordinates), defines Jacobi coordinates as variables, and appends the files with the text provided in the custom template. Since these files only contain basic information about the position of atoms, the user needs to enter all other information, such as reference WF, basis set, method, variables, and table. It allows users to generate excited states PES and/or calculate other specific parameters that change with the geometry of the system. Templates for custom input are provided for the following calculations:

1. Molpro Custom Templates:

```
molpro_ext = """
basis = AVQZ
{{RHF}}
E_HF = energy
{{CCSD-F12a}}
E_CC = energy

table,R,$E_HF,$E_CC     ! 1D PES Table
digits,2,10,10
"""
```

The multi-line string in Python is defined by triple quotes. Also, remember to use double curly brackets where single curly brackets are needed. The example shows a calculation for 1D PES. For 2D/4D, the table must be changed as shown below:

```
""" ...
table,R,Theta,$E_HF,$E_CC     ! 2D PES Table
digits,2,2,10,10
"""
```

```
""" ...
table,R,Phi,Theta2,Theta1,$E_HF,$E_CC     ! 4D PES Table
digits,2,2,2,2,10,10
"""
```

In the Molpro's custom input file, Jacobi coordinates are defined by the variable names as shown in the table above. Additional templates like:

(a) MRCI - Excited states
(b) SAPT
(c) Shielding, etc.

are provided in '*input_files/1_PESGen/only_ext/Molpro_custom/templates_molpro.py*'.

2. Psi4 Custom Templates: (completely customizable) First, set the BSSE option to indicate what kind of input file is required:

```
psi4_bsse           = 'cp'     # counterpoise: 'cp' or None (update same below)
```

The Psi4 custom input provides two options for coordinate generation, *psi4_bsse* ='*cp*' and $psi4\_bsse = None$, leading to the following outputs.

```
molecule { 0 1
C          0.000000           0.000000        0.000000
He         0.000000           0.000000        2.000000

 no_com
 no_reorient
 }
R = 2.0000
```

Listing 3.17. Output for psi4_bsse = None

$psi4\_bsse = None$: If CP correction is not needed and the whole system needs to be treated as one unit, the variable $psi4\_bsse$ must be set to $None$. Here {0 1} denotes the spin and multiplicity of the complex.

```
molecule { 0 1
--
 0 1
C          0.000000           0.000000        0.000000
--
 0 1
He         0.000000           0.000000        2.000000

 no_com
 no_reorient
 }
 R = 1.0000
 Phi = 0.0000
 Theta2 = 0.0000
 Theta1 = 30.0000
```

Listing 3.18. Output for psi4_bsse = 'cp'

For *psi4_bsse* = 'cp', the input files will have separate fragments (for CP calculations) with respective charges and multiplicities. The overall charge and multiplicity are indicated at the top, while the same for fragments are provided just after '−−'.

The *no_com* and *no_reorient* commands were added to suppress the rotation of molecules (makes it easier to debug code in 2D/4D PES). This should cause no change in final energies. The program also prints coordinates with variable names, which makes it easier to create (print) tables.

The required command can be entered via multi-line (triple quotes bound) input in *psi4_ext*.

```
memory 8 GB
set_num_threads(8)

set reference rhf

E_TZ_CP = energy('ccsd(t)/aug-cc-pvtz', bsse_type='cp',return_total_data=True)

# Table will be printed in terminal | See Manual
print('%.4f\t%.12f'%(R, E_TZ_CP))
```

The simple command line allocates 8GB of memory (RAM) and 8 processors (threads: see Section 2.3.1) to the Psi4 calculation. The reference wavefunction is RHF and method/basis is CCSD(T)/AVTZ with BSSE.

The final line prints the table (R and Energy) in the terminal, which can be easily extracted. The Psi4 PES batch file (to run calculation) automatically saves the printed PES data into a 'PES.dat' file.

The file '*input_files/1_PESGen/only_ext/Psi4_custom/templates_psi4.py*' contains templates for CP-corrected/CBS-extrapolated PES, Excited states, MRCC, and other External calculations are provided for Psi4 calculations for 1D/2D/4D collisions.

Batch files for running external calculations and Python files for collecting results.

After generating input files for the required quantum chemistry package:

1. (a) Run calculations using batch scripts provided in '*input_files/aux_scripts/batch_scripts*',
2. (b) Extract PES using python scripts provided in '*input_files/aux_scripts/PES_extract*',
3. (c) Convert PES (1D/2D/4D) from Hartree to cm^{-1} using '*input_files/aux_scripts/PES_to_cminv*'. This codes automatically subtract the asymptotic (R) energy at each angular coordinate and therefore takes care of the size consistency error that exists for methods such as F12 and CI.

Place the batch scripts outside the folder containing the PES input file. The name of the folder (containing PES files) and the number of files can be specified in the bash and Python scripts, respectively. The auxiliary files are covered in detail in Section 3.5.

3.4.3 *PESPlot*: R *vs* E and Polar Plots

The PES plot module is invoked automatically if PES is generated using internal Psi4 calculations. However, if any change is required in the plotted figures or if the PES is generated externally using Psi4, Molpro, or Gaussian, the same can be plotted using the PESPlot module.

After the external PES file is saved/extracted in dataframe format (R, $\theta(s)$, E), the same is loaded and plotted using PESPlot in two formats (1D R vs E plot and 2D polar plot):

- R vs E plot, where different angles will be denoted by different colors:
  - 1D PES: Single plot for R (Å) against E i.e. Potential Energy (cm^{-1}),
  - 2D/4D PES: Single plot where different colors indicate different (4D: combination of) angles.
- Plot where R and θ are represented on the polar axis and E is represented by colors:
  - 2D PES: Single plot with R/θ in polar axis and E as contours,
  - 4D PES: Multiple plots based on the angle combination stated in the input file.

The PES plot module is invoked in the input file using $Plot\_PES = True$, and the $Proj\_name$ should be the same as the one used in the PESGen input file. The PES file name must be specified in $PES\_filename\_cm$ and should be kept inside $Projects/\$Proj\_name/$ folder.

```
Plot_PES = True
Proj_name = 'C_He'  # Project name (Path:/current-folder/Projects/Proj_name)
PES_filename_cm  = "PES.dat"     # Enter Filename (Energies in cm-1)
```

Very important: The separation in the input file must be mentioned using the '*sep*' keyword:

```
sep    = '\s+'   # data separation: ',' comma, '\t' tab, '\s+' multiple spaces
```

If the data is separated by space/multiple spaces/tab, then using '\s+' is sufficient. If the separation is a comma, use `','`. Make sure the separation is correct and quotes are used, as this is usually the source for most of the errors.

Plotting parameters: The format for the output file is provided using the 'fmt' keyword, and the range for R is provided using 'R_lim' in Å. The lower limit for polar plots is 0.

```
fmt    = 'pdf'   # format for created plots, options = pdf, eps, png, etc.
R_lim  = [2,8]   # R limit for plots in Angstroms (lower limit for the 1D plot only)
```

For 2D systems, an additional keyword '$thetax$' is needed that will indicate the angles to be included in the R vs E plot.

```
# 2D PES angles to be plotted! (Uncomment the next line to use! )
thetax  = [0,30,60,90]       # 2D (theta)
```

For 4D systems, '$thetax$' is a 2D array to indicate each combination of ϕ, θ_2, θ_1 to be included in the R vs E plot. Additionally, '$phix, theta1x, theta2x$' are also needed to indicate the combination of the same that will be used to generate the polar plots. The same is not required in 2D systems, as there is only one angle and therefore only a single plot is possible.

```
# 4D PES angle combinations to be plotted! (# R vs E plot)
#        [ [phi,th2,th1] , [phi,th2,th1] ]
thetax = [[0,0,0],[0,90,90],[0,90,0],[90,90,90]]

# Angle combinations for polar Plots
phix               = [0,30,90]          # phi angles
theta2x            = [0,30,90]          # theta2 angles
theta1x            = [0,60,180]         # theta1 angles
```

Additionally, several optional commands (commented by default) are available in the module.

```
########################### Optional Commands ######################################
E_inf = -40.632795690775      # define E_infinity (Asymptotic Energy R@Inf)
#############################################################################################
################ Limited Availability (X axis = all plots) ##########################
plot_name  = 'xyz'                          # 1D/2D only
plt_title  = 'Potential Energy Surface'     # 1D/2D only
plt_x_axis = r'R $\mathrm{(\AA)}$'          # X label (in latex $...$ format)
plt_y_axis = r'Energy $(\mathrm{cm}^{-1})$'  # Y label (in latex $...$ format)
E_lim = [-200,100]    # Fix upper/lower energy limit for plots
#############################################################################################
# By default, the pes-data file is searched inside the project name. For any other ###
# location, uncomment the next line and enter the location below within quotes ''. ###
#############################################################################################
Plot_folder = "/Volumes/xyz/Projects/H2_H2/"  # Enter external folder location
```

These optional commands can be uncommented to customize or override the default plot settings. The optional variables available in the PESPlot module are :

- *E_inf* (float): Enter Energy at Asymptotic R to convert energy from Hartree to cm^{-1}. If commented (default), the input PES is assumed to be in cm^{-1} scale and plotted as it is.
- Plot Options: Use quotes (`''`) if the input values are strings.
  - *plot_name* (strings): Change Plot name. Default: `'R_plot_ND'`.
  - *plt_title* (strings): Change Plot title. Default: `'PES'`.
  - *plt_x_axis* (strings): Label for X axis; Enter as `r'$...$'` to use latex symbols. Default: 'R (Å)'
  - *plt_y_axis* (strings): Label for the Y axis (similar latex format). Default: `'Energy (cm$^{-1}$)'`
  - *E_lim* (array): Set upper/lower energy limit for plots [lower limit, upper limit] (Both *R* vs *E* and polar plot). Default: the global minimum is used as lower bound. Upper bound is kept symmetric.
  - *E_stp* (Polar Plot Only: float): Step size for energy to change contouring (default 0.1). Adjust if the color shades cannot be distinguished visually.
- *Plot_folder*: User entered location for *PES_filename_cm* (if different from the default). Default: PES file is searched inside */Projects/$Proj_name/*.

3.4.4 *NNGen*: Augmenting PES

The input file has similar options with keywords indicating job type (*Create_NN_model*), and the project name (indicated by *Proj_name*), where the PES file (name indicated by *file_name* keyword) is present with sparse (and possibly also missing) coordinates.

```
############################## Global Options ######################################
Create_NN_model = True     # jobtype
Proj_name = 'C2-He'           # Folder name inside "Projects" folder
file_name = 'PES.dat'   # input file name for PES file
```

After the general/global parameters are declared, the NN-specific data must be entered.

```
#-----------------------Input File Parameters-------------------------------#
# Data Separation: '\t' for tab, '\s+' for multiple spaces, ',' for comma, etc.
# for the input file "file_name" provided above
sep = ','                # Data Separation: | '\t' | '\s+' | ',' |  etc..
num_X = 4                  # the number of  input columns (descriptors)
num_Y = 1                  # the number of output columns (descriptors)
```

The data separation is very important, as stated earlier, since the same can cause errors in the internal working of the program (always check how the data is separated), i.e. commas, spaces, etc.

The NN module in PES2MP is created using TensorFlow's functional API and can support multidimensional input/output data for supervised augmentation. In simple words, the model can learn and augment more than one output using a single model. Therefore, it is important to indicate the number of input and output features in your input data. The example here shows a 4D PES for 2 RR collision where we have 4-dimensional input i.e. $R, \phi, \theta_2, \theta_1$ (*num_X*) and single output V (*num_Y*).

```
#-----------------------Partition commands----------------------------------#

# The minima region will be learned separately from the High Energy (HE) region!!
partition_req = True       # partition data -> High Energy(HE) and minima region

High_E_cutoff = 300        # E in cm-1 ---> Energies lower than this cutoff
                           # will be used for the primary NN model. Higher energies
                           # in the repulsive region will be separated!

reference_output = 4       # Column number for sorting and separating minima/HE
                           # Column numbering starts from 0
```

The data partition is an instrumental part of the NN model where the code automatically separates the high-energy region (at short R distance) that is not useful for dynamical calculations and can be approximated using the PES-specific NN model (it uses a modified decay function for R coordinate). The presence of very high energy increases the error for the minima and asymptotic regions. The cutoff can be the kinetic energy up to which dynamical calculations are performed, which is usually the first bending frequency for the ground state of the rigid rotor.

Since there can be more than one output (as stated earlier), the output feature (column) used to trim high-energy coordinates is called the *reference_output*. Therefore, for most cases (with single output), the reference output is the potential (V), and its values are used for separating high-energy regions. An example of multiple outputs is discussed below.

Suppose, *num_X* is 4 ($R, \phi, \theta_2, \theta_1$) and *num_Y* is 2. The two outputs are CCSD(T) energies (V_{CC}) and HF PES (V_{HF}). Since the **column number starts from 0**, the values go as follows $R(0)$, $\phi(1)$, $\theta_2(2)$, $\theta_1(3)$,

$V_{CC}(4)$, $V_{HF}(5)$. Suppose the user requests the high energy cutoff to be V_{CC}, the $reference\_output$ will be 4; otherwise, to use V_{HF} for setting the cutoff energy, the $reference\_output$ will be 5.

The extra output feature can be anything like dipole moment, or other properties that change with Jacobi coordinates. Overall, if the same is not needed, the $partition\_req$ can be set to false, and no data will be separated/trimmed.

If the partition is requested, the PES2MP code will automatically separate the high-energy region and will use only minima and asymptotic regions to create NN model (which will be collectively referred to as minima-asymptotic region). The boundary elements for both regions (high-energy and minima-asymptotic) will be extracted and plotted, and if required, can be used by the user for other purposes. The code is written to retain all the regions (high-energy boundary and non-boundary) for any future application. The boundary elements are always included in the training dataset in order to remove boundary errors. The non-boundary elements are separated into the training, testing, and validation datasets as stated below:

```
#------------------------Train/Test/Validation Split---------------------------#
# Enter your choice for dataset Split (Refer to FAQs).
# 1 = Randomized (Default) OR Stratified : 2 = Input (Recommended) 3 = Output
Split_type = 1              # Randomized/Stratified (Input or Output): See above

#------------------------NN model Parameters-----------------------------------#
# NN model parameters (Program uses remaining data points for validation dataset)
train_dataset   = [90]*2       # % of data points used for training model
testing_dataset = [5]*2       # % of data points for testing model
```

Once the code has already separated high-energy (by default, not used in model training) and boundary (explicitly included in training dataset) elements, it needs to split the **non-boundary elements** of **minima-asymptotic region** into training, testing, and validation datasets. The users can use random splitting ($Split\_type = 1$) or binning (2/3). The binning stratification converts continuous numerical data into categorical data to make sure the data is split evenly across the input/output regions. Suppose the user feels that the data is more sensitive to Euclidean distances, then input stratification can be used ($Split\_type = 2$). If the data is more dense (or has varying slope) in some coordinates than others, it is better to go for output stratification ($Split\_type = 3$).

Generally, small improvements are seen for stratification; however, for a new user, random splitting is typically sufficient as it provides robust performance, and boundary errors are naturally handled through boundary separation. The seed (for $Split\_type = 1$) and number of bins ($Split\_type = 2/3$) are prompted in the terminal during the program execution.

NN model Parameters: Hyperparameters are the most important aspect of the NN model. The dataset splitting and model architecture must be carefully chosen to prevent underfitting/overfitting of data. A simple way to figure out the balance between the two is to create multiple models with different training epochs and observe the model error in residual plots.

The PES2MP code creates several models in a single run and then creates a final ensemble model from them to generate the final (interpolated) output. The value(s) of $train\_dataset$ and $testing\_dataset$ in the above lines describe the split ratio of train, test, and validation datasets, and their length defines the number of models. For example:

1. Simple two base NN models with 80:10:10 split (Training: Testing: Validation)
    - `train_dataset = [80]*2` | equivalent to `[80,80]`
    - `testing_dataset = [10]*2` | equivalent to `[10,10]`
    - For each model, the remaining points are used for the validation dataset, i.e., `[10,10]`

2. Four base NN model with following splits: [90:5:5], [80:5:15], [50:5:45], and [20:5:75]
    - `train_dataset = [90,80,50,20]` | training dataset has different split ratios
    - `testing_dataset = [5]*4` | all 4 models have 5% data in test dataset `[5,5,5,5]`
    - The remaining points in the validation dataset are `[5,15,45,75]`
3. Six NN model with following splits: [90:05:05] (2 models), [80:10:10] (4 models)
    - `train_dataset = [90]*2+[80]*4` | i.e. training model dataset [90,90,80,80,80,80]
    - `testing_dataset = [5]*2+[10]*4` | i.e. testing model dataset [5,5,10,10,10,10]
    - The remaining points in the validation dataset are `[5,5,10,10,10,10]`

The architecture of the ensemble model can be visualized from the theory section. The ensemble model uses the first split ratio to train the ensemble model, so make sure it has sufficient data points in the training dataset. The general rule for the split ratios is:

- if the data is sparse, use an 80:20 split ratio: (80% training and 20% for testing and validation)
- if the dataset is relatively dense, use a 50:50 split ratio: (50% training, 50% for testing, and validation)
- if the dataset is very dense, use a 20:80 split ratio: (20% training and 80% for testing and validation)

Since in our case, we have a sparse PES, sticking to a 90:10 and/or 80:20 split ratio is recommended.

```
NN_hyperpara = {
    'Max_trial'      : 10,           # number of trials for architecture search
    'NN_nodes'       : [16,32,64],   # search space for NN nodes per layer
    'NN_layers'      : [2,3,4],      # search space for NN layers
    'NN_branches'    : [2,4],        # search space for NN Branches: even only
    'maxit_trial'    : 250,          # max iterations during trial (100-500)
    'maxit_base'     : 1000,         # max iterations: base model (500-5000)
    'maxit_ensemble' : 10000         # max iterations: ensemble model (5-10K)
}
Ensemble_Model_Train = True

# NN model early stopping hyper-parameters
early_stop_para =  {
    'start_after_cycle_base' :  20,   # 20-50% of max iterations: base
    'patience_step_base'     :   5,   # 5-10% of max iterations: base
    'start_after_cycle_en'   :  50,   # 50% of max iterations: ensemble
    'patience_step_en'       :  10    # 10% of max iterations: ensemble
}
```

NN architecture search and number of training epochs

The search space for a suitable NN architecture depends on the complexity of the data. If the data is multidimensional, it will need a deeper model. PES2MP uses Bayesian search (in Keras-Tuner's library) to find the best model based on a small number of trial iterations (epochs) specified by *maxit_trial*. Users can play with different parameters to see how their final predictions vary with different hyperparameters.

- Max_trial: The number of trials for architecture search
- NN_layers: Search space for NN layers, e.g [2,3,4],
- NN_nodes: Search space for number of NN nodes per layer, e.g. [16,32,64],
- $NN\_branches$: Search space for NN Branches: even only, e.g. [2,4]

The Max_trial indicates the number of trials the program will run with various combinations of node, layers, and branches specified above. part from these, there are certain layers which are built into the model

as a part of the architecture, such as input, output, and concatenate. The nodes and layers are the most basic parameters of any NN model, while the branches have been introduced in this model for handling multidimensional data without making it too deep.

What are these? In simple language, the nodes are the number of basic units that have coefficients (based on the activation function), which are optimized during training. The layers here refer to the number of hidden rows, which defines how deep the model will be. The branches are columns that run parallel to the rows and are merged after each layer. Adding branches makes the training of multidimensional PES, such as 4D collisions, more efficient.

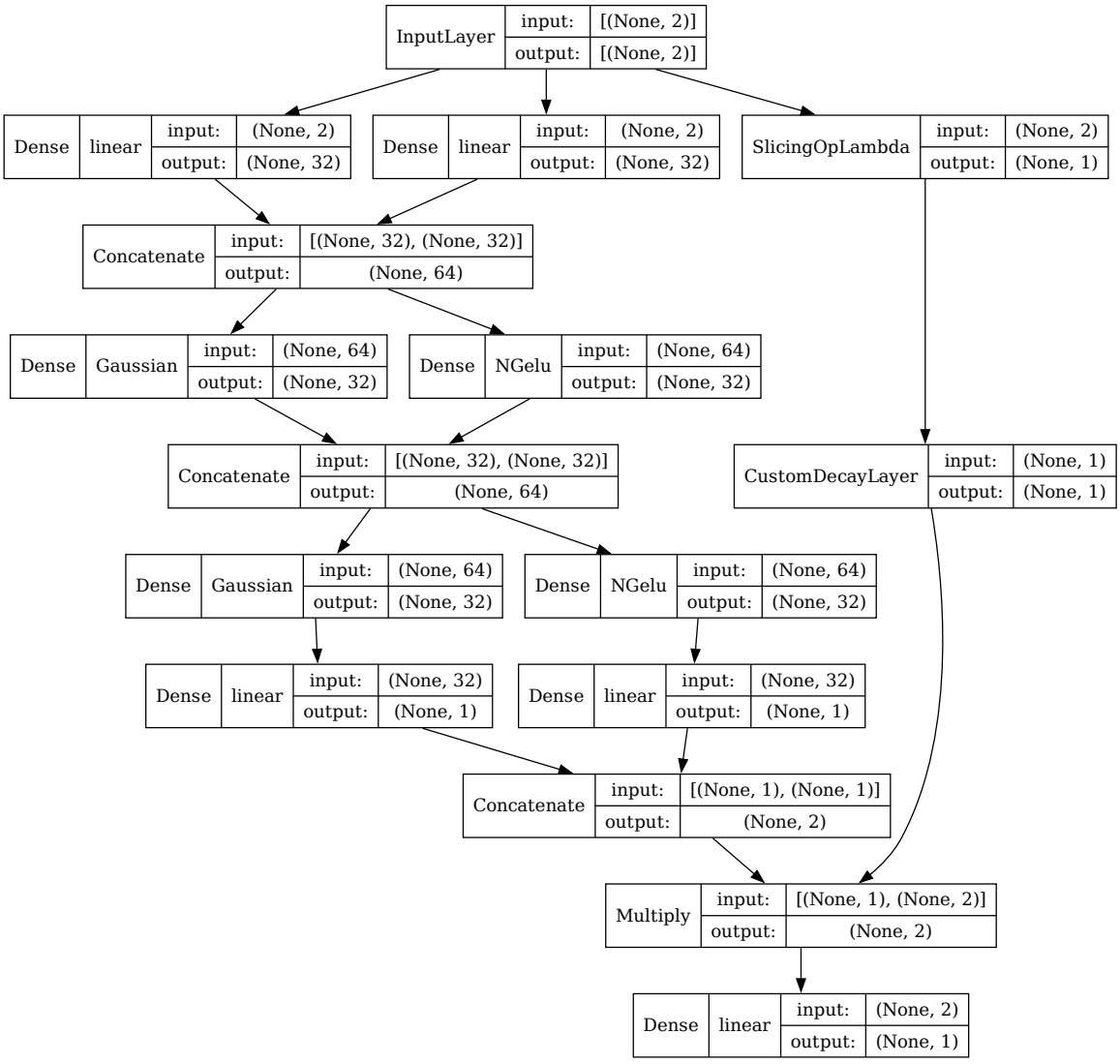


Figure 3.7. A simple example of PES-specific model (having a custom decay function as constraint on R) with 32 nodes, 2 layers (Gaussian and NGelu), and 2 branches.

In the model represented in Fig. 3.7, the *SlicingOpLambda* function passes only the R feature through a decay function specified by *CustomDecayLayer* on the right. This is the constraint that the generic model does not have. On the left, there are two columns of linear dense layers. Below the linear layers, there are

two columns of alternating *Gaussian* and *NGelu* activation layers. Here, the number of nodes (*NN_nodes*) is 32, and since there are two rows of *Gaussian/NGelu* layers, there are two layers (*NN_layers*). Finally, two columns of alternate *Gaussian* and *NGelu* activation layers represent two branches, which are concatenated after each layer. Due to the alternating layers (*Gaussian* and *NGelu*), the number of branches must always be even.

The program keeps the first and last layers as linear activation layers and concatenates each branch (having an alternative of *Gaussian* and *NGelu* activation layers) after each layer (row). In the generic model, the activation function used is *Gelu* for all branches. The linear, input, output, and multiply layers are part of the NN architecture, which are built in (not user-configurable).

Passing a single option in the architecture search space will make the model run a single trial iteration and use that model for training. The best results have been observed with:

- *NN_nodes:* [64],
- *NN_layers:* [4] (2D) / [8] (4D),
- *NN_branches:* [4].

Since the model architecture is fixed in this case, users can keep trial iterations (epochs) *maxit_trial*: 1.

- *maxit_trial*: 250, max iterations during trial (100-1500)
- *maxit_base*: 5000, max iterations: base model (5k-25K)
- *maxit_ensemble*: 10000 max iterations: ensemble model (5k-25K)

Here, the users choose the maximum number of iterations (epochs) while the model hyperparameters are searched, i.e., during trial iterations (*maxit_trial*: 250). The trial architecture will be trained for 250 epochs (kept small to save time), and the model with the lowest error will be chosen for full training.

Once the best architecture is found through trial iterations, that architecture is called the base model, and it is trained till the error flattens out (training is complete), e.g., *maxit_base: 5000*.

Since the program can create several base models based on the splitting of the input dataset, each base model will be trained for a certain number of cycles and then passed together with all other models into the ensemble model. Here, the weight of each model is tuned in *maxit_ensemble*: 10000 iterations (epochs).

Overall, these values depend on the approach for training. For architecture search (trial models), 250-500 epochs are good enough. For the base model and ensemble model, the training time depends on:

Ensemble_Model_Train = True/False

If the above parameter is True, the coefficients of base models are retrained with other models. If False, the coefficients are frozen and only the overall weight of the model is tuned (based on its performance). **Warning!** For old machines, re-optimizing coefficients, i.e., keeping *Ensemble_Model_Train* = True, can be very slow.

- **OPTION 1:** *Ensemble_Model_Train* = False: If users want to train the base models and freeze their coefficients in ensemble training, they must ensure that the **epochs for base models** (*maxit_base*) must be **large**, like 10,000, and **ensemble epochs** (*maxit_ensemble*) can be **small**, like 500-2000.
- **OPTION 2:** *Ensemble_Model_Train* = True: If users want to train the base models for a short while and then re-optimize its coefficients during ensemble training, they must ensure that the **epochs for base models** (*maxit_base*) must be **small**, like 500-2000, and **ensemble epochs** (*maxit_ensemble*) must be **large**, like 10,000.
- **OPTION 3:** *Ensemble_Model_Train* = True: If the resources permit, both **epochs for base models** (*maxit_base*) and **ensemble epochs** (*maxit_ensemble*) can be kept **large**, like 10,000+. This is sometimes necessary for 4D+ PES.

Early stopping to prevent overfitting

- $start\_after\_cycle\_base$: 50 % of $maxit\_base$
- $patience\_step\_base$: 10 % of $maxit\_base$
- $start\_after\_cycle\_en$: 50 % of $maxit\_ensemble$
- $patience\_step\_en$: 10 % of $maxit\_ensemble$

Early stopping is a great way to stop training when the model starts to overfit. Suppose during training, a model's validation error does not improve after a certain cycles, it is better to stop its training as it may start to overfit. But before this early stopping kicks in, it is a good practice to allow certain epochs to complete, which allows the model to train freely in the beginning.

Suppose your *maxit_base* is 1000 and *maxit_ensemble* is 10,000. Using *start_after_cycle_base* and *start_after_cycle_en* with a value of 50 allows the model to train for at least 500 and 5000 epochs, respectively. After that, base models will be checked after 100 epochs (*patience_step_base*: 10% of 1000) if they are showing improvement in validation error or not. Similarly, a value of 10 for *patience_step_en* indicates that the error improvement will be checked after 1000 epochs (10% of 10000). If there are no improvements, the training will stop.

```
# Augmented dimensions for NN prediction
ini_values = [1,     0,   0,   0]      # Initial values for each input dimension
fin_values = [50.1, 91,  91, 181]      # Final values for each input dimension
step_sizes = [0.1,  15,  15,  15]      # Step sizes for each input dimension
```

Once a model training is completed, the PES2MP package automatically generates augmented PES using the coordinates entered above. The augmented PES is predicted for all base models and the ensemble model. The initial and final values are entered in ini_values and fin_values, and the step sizes are entered in $step\_sizes$ (the ordering remains the same as to input features). Here is an example:

Suppose the NN model trains on 4 coordinates $(R, \phi, \theta_2, \theta_1)$. If the collider and RR of interest are symmetric, the ranges for angular coordinates are: $\phi = 0° - 90°$, $\theta_2 = 0° - 90°$, $\theta_1 = 0° - 180°$. Since Python loops run from $initial$ to $final - 1$, the data above results in:

- R goes from 1 Å to 50 Å with step size of 0.1 Å
- ϕ goes from 0° to 90° with step size of 15°
- θ_2 goes from 0° to 90° with step size of 15°
- θ_1 goes from 0° to 180° with step size of 15°

Usually, this step size is good enough for dynamical calculations. For special cases (e.g., where the target RR is anionic and radial terms did not converge), a finer grid may be required. However, a 5° step-size in any one coordinate will increase the file size 12-fold ($15/5 = 3$-fold in all four dimensions). Therefore, the file size can increase dramatically. On the other hand, a 2D surface (RR-atom collision) can even be augmented at 1° intervals for some time-dependent simulations (highly anisotropic) or for simulating high-resolution laser or field-controlled experiments.

∗ Do not use NN model for doable systems (small 2D PES), as it may introduce additional errors. Heavy rotors and 4D collisions are the conditions where the PES generation can be very slow. Here, users can selectively choose coordinates that can be calculated (filter using HF-D4 results) from the full dataset of coordinates. Include more high-level *ab initio* data points in anisotropic regions and keep the generated PES sparse. The missing points can then be obtained using the NN model.

Optional Commands

We shall now look into various optional commands, some of which may overlap with the commands that we discussed above.

```
##########################  Optional Commands ###################################
# Commands to rearrange/scale and remove columns in the dataset (Uncomment to Use)
# Default values 0: Uncomment and set variables to 1: follow on-screen commands

rearrange_columns = 1     # Rearrange the position of columns in the dataframe
drop_columns = 1          # drop any non-required column
scale_columns = 1         # scale R, theta, and energy into different units
```

The NNGen module has been provided with the ability to rearrange columns, drop (non-essential) columns, and scale columns (user-defined) as it may be used independently for augmenting PES. The users can **follow on-screen commands to perform the same tasks**. The scaling targets radial, angular, and energy values, and the most common scales are built into the program. Remember that the 'Augmented dimensions for NN prediction' part (augmented output) will now require the initial/final/step-size values in the new scaled/rearranged format.

Therefore, the users are requested to check the new (scaled/rearranged) data (saved in the NN output folder) and check if the data follows the intended range. In the next run, the new (scaled/rearranged) data can be used directly (without scaling/rearranging) to avoid confusion.

```
##########################  Advanced commands ###################################
#-----------------------------  Plot options  ---------------------------------#
yscale = f'cm$\mathrm(^{-1})$' # yscale is in cm-1
fmt = 'pdf'                    # output format for plots

#--------------------------     Plot Scale   ---------------------------------#
# for +ive logarithmic scale: "log" | for both +ive and -ive log scale: "symlog"
# Default values are provided as given below

# Sorted Partitioned/Unpartitioned plot
plt_srt_yscale = 'symlog'          # 'linear' | 'log' | 'symlog'| etc...
plt_srt_xscale = 'linear'          # 'linear' | 'log' | 'symlog'| etc...

# Boundary Partitioned/Unpartitioned plot
plt_bnd_yscale = 'linear'          # 'linear' | 'log' | 'symlog'| etc...
plt_bnd_xscale = 'linear'          # 'linear' | 'log' | 'symlog'| etc...
```

The plot option is simply the name of the y scale in the residual plot (currently set to cm^{-1}). If the data is in eV, the same can be changed to yscale = f'eV'. The *fmt* refers to the format of the residual plot (default is `'pdf'`). For very large datasets, use compressed file formats (e.g., `'jpeg'`).

The other commands control the x/y axis of the plot, i.e., linear scale, log (only +ive values), symlog (both +ive and -ive log scales), etc. Leave them as it is unless necessary.

```
#--------------------------  Partition  ------------------------------------#
partition_req   = 0      # partition the data into high energy and minima region
                           # Default: 1
auto_cutoff     = False  # if False, follow on-screen commands,
                           # if True (default), give High_E_cutoff below
High_E_cutoff   = 300    # E in cm-1 ---> Energies lower than this cutoff
                           # will be used for the primary NN model. Higher energies
                           # in the repulsive region will be separated!
reference_output = 2     # reference column for sorting, partitioning, and plotting
                           # default: first output column for multiple output data
```

Here, the module requests the user to first select if the partition is required or not (*partition_req*).. If not, then the other three options are not needed. If yes (1/True: both are acceptable in Python), which is the default, the users must mention if they want the module to automatically separate energies above a certain threshold (*auto_cutoff*) for the dataset or manually select the data points (hit and trial).

If *auto_cutoff* is set to 1/True, the other two commands are used where the user must enter the energy cutoff (e.g., 300 cm^{-1} here) and mention the column number that contains the energies (the numbering starts from 0, and the default is the first output column). The *reference_output* = 2 starts that there are two input features/columns (column number 0 and 1 in Python), and the energies are in the third column (column number 2 in Python).

If *auto_cutoff* is set to 0/False (and *partition_req* is set to 1/True), the users must enter the guess value (for the number of points to be included in the final dataset) after running the program (a prompt screen will appear). The dataset is sorted by energy (1st output column) and generates an *E vs row-number* plot. The guess value is the value between 1 and the final row count. The row numbers after the user input (guess) will be separated and not used in training. The user can give multiple guesses till they are satisfied with the row number. This is only useful if the user intends to visualize the data (minima/maxima), which may contain properties other than energy, before selecting the final cutoff.

```
#-------------------------   NN model  ------------------------------------#
generic_model = True  # Use generic model instead of default model (R/theta PES)
                       # The generic model does not constrain R

HE_train = False      # Combine HE data in the NN model (provide HE_epochs below)
HE_epochs = 100       # (50-200) WARNING: Large epochs will degrade minima
##############################################################################
```

In the final part, the NN-specific parameters are needed from the user. By default, the NNGen module uses the ND-PES model, i.e., an N-dimensional PES model, where high-energy and asymptotic regions are extrapolated using a modified decay function on *R*. The model also has bias turned off, and it may not be good for general datasets.

Therefore, the *generic_model* option can be used, which uses the same NN architecture as ND-PES without the decay function. The bias is also set to default (True), and an inbuilt (`gelu`) activation function is used rather than custom (`NGelu` and `Gaussian`) functions.

If the user intends to still use the high-energy data in the final NN model (using either the generic or ND-PES model), the same can be accomplished by setting *HE_train* = *True* and providing the number of epochs till which the model will be trained. Once the HE region is trained, the minima-asymptotic region training will begin. In my experience, the very high data points make errors much larger, and this forces the model to train over more epochs and therefore overfit. The better choice is to use the ND-PES model (default model), which puts a decaying constraint on *R*. The *HE_train* and *HE_epochs* options can be beneficial for special applications (e.g., dipole moment, NACME, etc) where the user wants special attention on small values (full training) while also training the model to learn larger values (but only for a few epochs).

* *The users are encouraged to edit the NN-related subroutines (`PES2MP_driver.py` file) and modify the parameters to suit their specific needs.*

3.4.5 *FnFit PES*: Fitting PES into analytical (R) function

This is another optional step that can help users get back missing data points in the PES. Compared to NN, this method loops over each angular term and only fits the radial PES. While it can also introduce additional errors into the PES, using Slater functions with fixed β values will prevent any additional errors when the PES is expanded into radial terms, as these radial terms must be expressed into analytic expressions that can be read by the MOLSCAT software suite. The idea is:

- For each angular term the PES is only E *vs* R.
- Fit each angular term (using a for loop) into $\sum_i \alpha_i \times e^{-\beta R}$ functions with fixed β values.
- The same function with fixed β valued can be used to fit the radial terms V_Λ.
- In essence, we only optimize α for both PES and V_Λ.
- In practice, the numerical error in fitting radial terms (obtained using PES fitted into Slater functions) is usually $\backsim 10^{-12}$. Even in the worst case, the error is $< 10^{-7}$, showing a near-zero error.
- The final coefficients, i.e., optimized α and fixed β value for PES and V_Λ, are saved into a file, which can regenerate the same later, if needed.

Remember that no data point must be missing in PES for MP expansion. If so, either these points are recalculated *ab initio* or using the ND-PES NN model. Using PES FnFit, the missing data points can be easily interpolated/extrapolated. Once the PES data is ready, the *ab initio*/NN-augmented/Function-Fitted PES can be expressed into radial terms using the MP expansion module explained in the next subsection. In this section, we shall discuss how PES can be fitted into a series of Slater functions for the reasons stated above. Users are encouraged to do MP expansion on PES with/without this fit and see how different the radial terms are. If enough Slater functions are used, the error is very small. However, for more Slater functions, there are more unknown parameters, and therefore, there is a chance that the fitting might fail.

The main advantage of using coefficients is that PES data can be saved in a much smaller file with the function and its coefficients. The auxiliary file used to regenerate PES from the coefficient file is explained in subsection 3.5.8.

∗ Fitting will fail when the number of unknown parameters > the number of data points used for fitting.

∗ *If NN augmentation is used!* Keep the step size for R in NN augmentation small (e.g. $\Delta R = \backsim 0.1$ Å) to allow for sufficient radial data points and prevent fitting failure.

```
##############################################################################
###################### Analytical radial fitting of PES ######################
##############################################################################
FnFit = True          # Job type
Proj_name = 'C2-He'  # Folder name inside "Projects" folder
Fnfit_type = 'PES'    # PES in cm-1 are located inside Proj_name folder
#-----------------------------------------------------------------------------#
filename  = "abinitio_cm.dat"     # Enter Filename @ /Proj_name/
sep    = ','      # data separation: ',' comma, '\t' tab, '\s+' multiple spaces
#-----------------------------------------------------------------------------#
```

- Similar to previous modules, the job name for fitting data into an analytical expression is simply *FnFit*.
- The project name is where the PES *'filename'* is present and data separation *'sep'* is either a comma, semicolon, or space(s) as discussed earlier.
- An additional keyword here is *Fnfit_type*, which denotes if the data to be fitted is: (a) PES or (b) radial terms obtained after MP expansion of PES.

```
#-------------------------------------------------------------------------#
# If PES must be in R, A, E format
PES_typ = '4D'      # Set '1D', '2D' or '4D' (important for reading file)
cutoff = 1500        # Energy cutoff in cm-1. Features are preserved till the cutoff
#-------------------------------------------------------------------------#
# plot parameters
fmt     = 'pdf'     # format for created plots, options = pdf, eps, png, etc.
scale_x = 'symlog'  # 'symlog' or 'linear'
scale_y = 'symlog'  # 'symlog' or 'linear'
Y_lim   = [-1,1]    # E (residual) limit for zoomed-in plot
R_lim   = [0,20]    # R limit for 1D PES plot
#-------------------------------------------------------------------------#
```

The *PES_typ* refers to 1D, 2D, or 4D PES resulting from various collisions. The other collisions, i.e., vibrational excitations, where molecules are not treated as rigid rotors, are not implemented in PES2MP. However, the 2D/4D code can be modified easily by the user to accomplish the same (the various *PES_typ* are implemented using if-else blocks).

The *cutoff* refers to the high energy cutoff in cm^{-1} where energies larger than this value (e.g., 1500 cm^{-1}) will not be included in PES fitting. This is done to ensure that the minima and long-range regions are fitted with less error. The high-energy region can then be extrapolated using the final analytical expression that is used to fit the PES.

Plot parameters for 1D/2D/4D collision (residual plot only): The residual plot shows the accuracy of the fit. Both axes (X and Y) use 'symlog' scaling for the residual plot with *Y_lim* (residual error in Y axis) ranging from +1 to -1. For an initial fit, users can increase the Y axis to ±5 or ±10 to better visualize residual errors.

Only 1D collision (*R vs E* plot): The PES2MP FnFit module plots the 1D *R vs E* plot for ab initio and function-fitted PES (along with a separate residual plot that is also generated for 2D/4D PES). It has been implemented for atom-atom collision since users will be more interested in seeing a direct ab initio vs fitted PES rather than the residual plot for a 1D curve. The minimum (and therefore the range for the Y axis) for the PES is determined automatically. Users can enter the range for the X-axis (*R_lim*) for this 1D PES plot to visualize the quality of the fit.

```
#------------------ Fitting PES each angular coordinate -----------------------------------#
# Use the below code to define a custom analytical expression for radial fitting
# Use templates and fn_generator.py for more functions
#-----------------------------------------------------------------------------------------#

def fnfit_custom(x, a1,a2,a3,a4):
    import numpy as np
    return a1*np.exp(-0.5*x) +  a2*np.exp(-1.75*x) +  a3*np.exp(-3.5*x) +  a4*np.exp(-5.0*x)

#-----------------------------------------------------------------------------------------#
initial_val  = [1e4]*4                # initial guess
#-----------------------------------------------------------------------------------------#
```

! `initial_val = [1e4]*4` is equivalent to `initial_val = [10000,10000,10000,1000]`.

The above example shows that four Slater functions ($\alpha e^{-\beta R}$) are being used to fit the radial part of the PES (for each angle). Such fitting seems odd as the beta coefficients are not relaxed. However, as previously discussed in the theory, the same function can be used to fit the radial terms (obtained after multipole expansion) without any error. Also, when both α and β are relaxed simultaneously, the fitting fails due to a large number of possible parameter combinations.

Important steps for choosing an analytical expression:

- Leave the name of the function (*def*) as it is ('*fnfit_custom*').
- The x should be the first input variable in the function.
- After 'x', enter the variables and other options as explained below:
- The numpy import is important to use the multivariate exponential function.
- Enter the α coefficients (e.g. a1,a2,a3,a4) to be optimized after x.
- Enter the Slater functions to be optimized after *return*
- **(Optional)** This whole section has been automated using '*Fn_generate.py*' python scripts available in *input_files*/3_*Fit_PES*/*Extra* folder. The same is explained in the auxiliary scripts subsection.
- Enter initial guess for each variable in *initial_val*. Make sure it has the same length as the number of variables.

Fitting the PES in this manner also has the advantage that the short and long-range of each angular term can be extrapolated using suitable functions. This feature is optional but is important for charged systems (very long tail), strongly correlated systems (perturbative methods break down at long-range regions), and very long molecules (numerical convergence at very short-range due to overlapping atoms).

OPTIONAL COMMANDS

```
#----------------------------- OPTIONAL ---------------------------------------#
E_inf = -186.545562935966      # define E_infinity (Asymptotic Energy R@Inf)
scale_Energy = 1               # multiplies cm-1 energy to scale_Energy
scale_R      = 1               # multiplies Ang. R diatance to scale_R
#------------------------------------------------------------------------------#
#------------------------ Optional Constraints --------------------------------#
lower_bounds = [-1e12]*4       # Lower bound
upper_bounds =  [1e12]*4       # Upper bound
```

The above commands can be optionally used for:

- *E_inf*: Converting Energies from Hartree to cm^{-1}. *
- *scale_Energy*: Optional scaling of energies for improving convergence.
- *scale_R*: Optional scaling of 'R' for improving convergence.
- *lower_bounds*: Constraining lower limit for coefficients (a1,a2,a3,a4)
- *upper_bounds*: Constraining upper limit for coefficients (a1,a2,a3,a4)

Some additional comments on the above-mentioned optional commands:

- The optimization of β coefficients is relatively hard, and it may require the use of upper and lower bound constraints along with a very good initial guess.
- Constraining limit to large value like ($\pm 10^{12}$) can improve convergence time as the default is $(-\infty, \infty)$.
- The scaling of R and energies (V) can be performed internally in the fitting function by including:

```
    def fnfit_custom(x, a1,a2):
        import numpy as np
        u = 1.8897259886 * x
        return 21974.63*(a1*np.exp(-0.5*u) +  a2*np.exp(-1.75*u))
```

  where R i.e. passed as x is scaled from Å to Bohr and saved as u. The whole equation is also scaled by multiplying 219474.63, which is equivalent to scaling energies from cm^{-1} to Hartree.
- For non size-consistent methods, use auxiliary script *PES_to_cminv.py* to convert Hartree to cm^{-1}. Use *E_inf* only if the asymptotic R energies are the same for all angles.

Optional High Energy and Long-Range fit

The long-range (LR) and short-range, i.e., high-energy (HE) regions of the PES can be extrapolated by fitting these regions into physically correct custom functions. The same can be invoked by setting either or both options (he_fit and lr_fit) to 1 or $True$. The FnFit module is designed to first fit the HE region, then the LR region, and then finally combine the results with the output of the custom function ($fnfit\_custom$) that fits the minima-asymptotic region.

```
#-------------------------- High Energy fit ---------------------------------#
he_fit        = True                # To fit high Energy region (Set True)
he_cutoff     = 10000               # end value: fit data points till he_cutoff
he_min_offset = 4                   # start value: fit from 4 positions before minima

def fnfit_he(x, he1,he2):           # Function for fitting high energy Region
    import numpy as np
    return  he1*np.exp(-he2*x)
he_initial_val  =  [1e4,1]          # Enter initial guess
#-----------------------------------------------------------------------------#
```

- The $he\_fit = 1$ or $True$ fits the short-range region that may not converge in long rotors.
- $he\_min\_offset = 4$: The 4^{th} data point before minima will be the starting point for the high-energy fit. Recommended Range: (4-6) is good enough for a 0.1 Å step size PES.
- he_cutoff: The repulsive energy points from he_min_offset till he_cutoff (cm^{-1}) will be used for fitting the function.

The function (def) is similar to the one described above, but has a unique name ($fnfit\_he$) that must not be changed. Here we use a single Slater function ($\alpha e^{-\beta R}$) to extrapolate the high-energy region, and both α and β coefficients ($he1$ and $he2$ respectively) are optimized. The initial guess for α and β is set to 10,000 and 1, respectively.

```
#-------------------------- Long Range fit ----------------------------------#
lr_fit            = True            # To fit Long Range Region (Set True)
lr_min_offset     = 40              # fit from 10 positions after minima till end

def fnfit_lr(x, lr1,lr2):           # Function for fitting Long Range Region
    import numpy as np
    return  lr1*np.power(x,-lr2)
lr_initial_val  =  [1e4,6]          # Enter initial guess
#-----------------------------------------------------------------------------#
```

The $lr\_fit = True$ fits the attractive region's data into the $fnfit\_lr$ function (inverse power of R). Since the behavior of neutral, cationic, and anionic van der Waals systems (He/H_2 collision) is known, we can take advantage of this knowledge to fit the long-range regions of the PES that may not converge for ab initio calculations due to numerical errors. ! *The users can even use multiple inverse R functions, Slater decay, or any other function they wish here. However, the reason for using any extrapolation function must be justified, as it can change the behavior of the dynamics.*

- The $lr\_fit = 1$ or $True$ requests that the long-range region must be fitted.
- $lr\_min\_offset = 40$: The 40^{th} data point after minima will be the starting point for the long-range fit. All the points thereafter will be included in the fit.

The function (def) accepts two variables $lr1$ and $lr2$. Here, we use an inverse power function ($\alpha R^{-\beta} = \alpha / R^{\beta}$) to extrapolate the long-range region. The initial guess for α and β is set to 10,000 and 6, respectively.

The use cases of LR and HE fit:

- Short range:
  - RR is too long: Energies don't converge at a small R region due to atom overlap.
  - Preventing artificial minima at very small R due to incorrect convergence.
- Long range:
  - The long-range region fails to converge due to numerical errors.
  - The molecule is strongly correlated: CCSD(T) will breakdown at long R.
  - Long-range behavior is known, e.g., He collision with:
    * Neutral RR: αR^{-6}
    * Cationic RR: αR^{-4}

* Fixing β, i.e., $lr2$, may result in a poor fit for the long-range region. Rather, giving an initial guess close to the expected value (e.g., 4) and allowing the value to relax (to 3.97 or 4.03) provides a better fit. These deviations from ideal behavior may arise from the angular dependence of radial terms.

While long-range fitting is necessary in some cases, it can be included in the custom function (that fits the minima-asymptotic region) by using a Slater function with a very small (if possible, also flexible) β coefficient. Nevertheless, the users must see the literature on rotational dynamics for a better understanding of the reasoning for any fitting.

More optional commands:

```
#------------------------------ More Options ------------------------------#
# The below command supersedes cutoff (Use with caution only if fitting fails)
cutoff_pos = 4  # Fitting will start 4 (cutoff_pos) positions before minima.
#--------------------------------------------------------------------------#
# New R range (Inter/Extrapolate R) for fitted PES
New_R = np.arange(0,50.1,0.1) # New R range from 0 to 50 Ang | Step size = 0.1
#--------------------------------------------------------------------------#
```

In simple $cutoff$, the user enters the cutoff energy (e.g., 300 cm^{-1}) for fitting the energies into a function of R. This signifies that energy values larger than 300 cm^{-1} will not be used to fit the function to maintain accuracy at minima. However, in some special cases, $cutoff\_pos = n$, i.e., cutoff position (n^{th} place) before minima can be used instead. Here, the fitting will only use energies from (let's say) 4^{th} place before minima to the last R.

Caution: In case both $cutoff$ and $cutoff\_pos$ are defined, the program will prioritize $cutoff\_pos$.

New_R: The PES will be predicted in this new R range. By default, the PES is predicted for the unique R values present in the input. Providing New_R can be used for interpolating or extrapolating the PES (or both) as intended. The program will predict the energies using the $fnfit\_custom$ function and plot a residual plot for visualizing fitting errors.

3.4.6 *MPExp*: Multipole Expansion and Radial Terms

The multipole (MP) expansion module of PES2MP is implemented for 2D (RR - atom) and 4D (RR - RR) collisions, where linear molecules are treated as rigid rotors. Users can modify the existing code for (a) 3D (vibrating rotor - atom) or (b) 5D (vibrating rotor - rigid rotor) cases. The input file for 2D/4D cases are described simultaneously below:

```
##############################################################################
############################ MP Expansion of PES ##############################
##############################################################################
MPExp = True              # Set job-type
Proj_name = 'test2D'      # Folder name inside "Projects" folder
#-----------------------------------------------------------------------------#
```

Like previous input files, the project name and job type are entered.

- $MPExp = True$ denotes that the job type is multipole expansion
- $Proj\_name$ can be kept the same (used for PESGen, NNGen, FnFit) as the program will automatically find the required PES in the project folder.

In case PES is generated externally (not using PES2MP): Create a $Projects/\$Proj\_name/$ folder (automated using GUI) and put the PES file inside the folder, i.e., for Proj_name = `'test2D'`, the PES file must be inside $Projects/test2D/$ folder.

```
#------------------------------ For 2D PES use -------------------------------#
Expansion_typ = '2D'      # 2D i.e. RR-atom collision PES expansion
#-----------------------------------------------------------------------------#

#------------------------------ For 4D PES use -------------------------------#
Expansion_typ = '4D'      # 4D i.e. RR-RR collision PES expansion
#-----------------------------------------------------------------------------#
```

Enter the expansion type (`'2D'/'4D'`) based on the collision type. The 2D expansion is carried out using Legendre polynomials, while the 4D expansion is done using bispherical harmonics.

```
# keep PES file inside Projects/Proj_name folder and provide data separation
# Do not use header and make sure that any R/theta coordinate is not missing
#-----------------------------------------------------------------------------#
PES_filename_cm  = "Fnfit_PES.dat"     # Enter Filename (Energies in cm-1)
sep = ','    # data separation: ',' comma, '\t' tab, '\s+' multiple spaces
#-----------------------------------------------------------------------------#
```

Enter the name of the PES file that can be *ab initio*, NN fitted, or analytically fitted (into Slater functions). Ensure that the PES has no missing coordinates, as the expansion uses pseudo-inverse (equivalent to least squares fit) and will error out if there are missing data points.

Since *ab initio* calculations usually result in missing data points, use NN or analytical fitting (i.e., $FnFit = True$ with $Fnfit\_type = 'PES'$) to get the missing data points.

Enter the data separation. If PES is separated by a semicolon (;), comma (,), etc., use the same. For single spaces, multiple spaces, or tab(s), using $\backslash s+$ will suffice. Also, check out the $pandas.readcsv$ option in pandas to check keywords for various separators.

The number of Λ terms in the 2D/4D expansion is specified using different variables, and the same is described below. The two important parameters are the number of expansion terms and the center of symmetry (RR).

```
#------------------------------ For 2D PES use -------------------------------#
lam_max = 7          # Maximum expansion terms for lambda in V_lambda
symmetric = True     # Verify if rigid rotor is symmetric (else put False)
#-----------------------------------------------------------------------------#

#------------------------------ For 4D PES use -------------------------------#
L1max = 4                           # max order for first radial term (L1)
L2max = 2                           # max order for second radial term (L2)
Symm_1 = True                       # True if RR1 is symmetric, else False
Symm_2 = True                       # True if RR2 is symmetric, else False
#-----------------------------------------------------------------------------#
```

The value of the maximum expansion term(s) must be entered in the input file as shown below:

- For 2D collision
  - lam_max: Number of λ terms.
  - $symmetric$: If rigid rotor is symmetric enter True/1. Else False/0.
- For 4D collision
  - $L1max$: Enter maximum value of λ_1. $L1max = 4$ means λ_1 expansion is done from 0 to 4.
  - $L2max$: Enter maximum value of λ_2. $L2max = 2$ means λ_2 expansion is done from 0 to 2.
  - $Symm\_1$: If rigid rotor (1) i.e. Rigid rotor with angle θ_1 is symmetric enter True/1. Else False/0.
  - $Symm\_2$: If rigid rotor (2) i.e. Rigid rotor with angle θ_2 is symmetric enter True/1. Else False/0.

The Table 3.1 defines how input PES and Λ terms must be provided for MP expansion. For 2D collision with symmetric RR, only provide PES till $\theta = 90°$, and PES2MP will take care of the rest. For symmetric RR, λ will not have odd values, denoting that the dipole moment is zero, and only even transitions (of rotational states) will occur in that RR.

Table 3.1. Reference sheet for performing MP expansion.

Collision Type	Symmetric	PES (Angular coordinates)	Λ
2D	RR: Yes	$\theta = 0° - 90°$	$\lambda = 0, 2, 4, 6...$
	RR: No	$\theta = 0° - 180°$	$\lambda = 0, 1, 2, 3...$
4D	RR1: Yes | RR2: Yes	$\theta_1 = 0° - 180°$ | θ_2, $\phi = 0° - 90°$	$\lambda_1 = 0, 2, 4, 6...$ | $\lambda_2 = 0, 2, 4..$
	RR1: No | RR2: Yes	$\theta_1, \theta_2 = 0° - 180°$ | $\phi = 0° - 90°$	$\lambda_1 = 0, 1, 2, 3...$ | $\lambda_2 = 0, 2, 4..$
	RR1: No | RR2: No	θ_1, θ_2, $\phi = 0° - 180°$	$\lambda_1 = 0, 1, 2, 3...$ | $\lambda_2 = 0, 1, 2..$

* For 4D, vector sum λ ranges from $\lambda_1 - \lambda_2$ to $\lambda_1 + \lambda_2$, and will have step size of 2 only if both rotors are symmetric (else 1).

The lam_max variable sets the maximum number of radial terms in 2D collision. E.g., $lam\_max = 7$ denotes seven terms and λ ranges from 0 to 6 (with a step size of 1) if RR is not symmetric. If symmetric, λ ranges from 0 to 12 with a step size of 2 (still seven terms).

On the other hand, for 4D PES, the $L1max$ and $L2max$ denote the maximum range of λ_1 (0 to $L1max$) and λ_2 (0 to $L1max$) terms, respectively. The program will automatically calculate the set of Λ from the provided values and symmetry of RRs. The number of radial terms will be printed on screen and saved in *Lambda_ref.dat* file. Ensure that it is less than the number of angular terms and proceed.

Remember that the number of expansion terms must ideally be equal to (or less than) the number of angles. The matrix solution (solving unknown Λ from known θ) is equivalent to the solution of linear equations, where the number of unknown terms cannot be more than the number of equations. E.g., in a 2D

system, if there are 37 angles ($0°-180°$ with a step size of $5°$), the number of λ terms in the solution will be 37 (i.e. $\lambda = 0-36$).

Given that the multipole expansion using the Legendre/SH series converges quickly, one can stop expansion once the expansion terms stop showing signs of minima/maxima. For example, the V_λ terms for a neutral RR colliding with He converge quickly, and therefore the first 12-19 λ terms are enough for the series to converge. After which the $V_\lambda \approx 0$. Users can also try and get 19 λ terms from 19, 37, or 181 angles and use them to calculate cross-sections. There would be a negligible difference between the computed cross-sections as the first 19 terms will be the same for all.

For charged systems, as many as $\backsim 37$ terms may be needed due to increased well-depth and increased long-range correlation of the PES. Overall, these numbers are indicative of anisotropic behavior induced by the He/H_2 collision. Since similar-looking chemical systems can be vastly different from each other, it is a good practice to always compute V_Λ and then check if the final terms are converging. If not, the number of angles can be increased using NN augmentation, and more V_Λ terms can be generated.

```
#------------------------------ For 2D PES use -------------------------------#
read_Legendre = False  # set read_Legendre to True to read existing file.
#-----------------------------------------------------------------------------#

#------------------------------ For 4D PES use -------------------------------#
read_SH = False  # set read_SH to True to read existing file.
#-----------------------------------------------------------------------------#
```

Here, the MPExp module asks if the user wants to read the existing Legendre (2D) or SH (4D) coefficients file. If Legendre/SH coefficients are available from a previous run (for the same angles and Λ combination), use the below command to read the existing NumPy file (*.npy*), else the code will generate the same from scratch, which will take a longer time.

This is beneficial if the user has already generated the Legendre/SH coefficients or when the PES is back generated from V_Λ using inverse transform to get residual plot for fitted surface (See section 3.4.8). Since in both cases the combination of Λ and $\theta(s)$ is the same, the same coefficients are reused. If you are unsure if an MP coefficients file exists (or has the same combination of θ and Λ, just keep *read_Legendre* (2D) or *read_SH* (4D) value *False*.

```
#-----------------------------------------------------------------------------#
# Enter a rough Estimate. Updated plots by --> read_Legendre = True --> new R/E
Ind_plot = False  # Each radial term is plotted individually (Full range/zoomed)
R_lim  = [0,10]   # R limit for R vs V_lambda plots in Angstroms
E_lim  = [-15,15]       # E limit for combined plot
fmt    = 'pdf'    # format for created plots, options = pdf, eps, png, etc.

############################ Optional Commands ################################
## If Energies are in Hartree, Enter E_inf below for conversion. (Optional)  ##
## Comment if Energies are in cm-1 (will result in wrong output)             ##
#E_inf = -188.31099452      # define E_infinity (Asymptotic Energy)
###############################################################################
```

The following options are for plotting the resulting radial coefficients. Since 4D PES can have upwards of 100 radial terms, the *Ind_plot* is kept *False* by default. To visualize each term separately, set this value to *True* (can take a long time depending on the number of plots). Similarly, other options like *R_lim* sets the limit for the x axis and *E_lim* for the y axis. The y axis (Energy) limit for individual plots is calculated individually for each term to give an overall, zoomed-out, and zoomed-in plot. The final optional command *E_inf* can be used to convert input PES from Hartree to cm^{-1} by entering E at $R = \infty$.

3.4.7 *FnFit Vlam*: MOLSCAT's &POTL block

This is the step where the radial terms calculated above can be fitted into analytical functions. If a user already has a modified MOLSCAT code that can read the raw data for radial terms (i.e., R *vs* V_Λ format) and carry out fitting internally, this step is not needed. However, if you're using the 2020 version of the code (compile `molscat-basic` and get the executable), these radial terms can be directly provided as a sum of Slater and power functions.

This input file will accept the data file containing V_Λ, and fit the same into an analytical expression of choice. The output will contain residual plot and MOLSCAT readable `&potl` file. If you have already fitted your PES (angle-wise) into Slater functions ($f(R)$) using the *FnFit* module for `'PES'`, use the same function (with the same fixed β coefficients) to fit your radial terms without any error.

```
####################### Analytical radial fitting of V_lam ####################
###############################################################################
FnFit = True                  # Job type
Proj_name  = 'C2_He'         # Folder name inside "Projects" folder
Fnfit_type = 'Vlam'  # Radial coefficients are located inside MP_files folder

# Keep header for V_lam-default file (V_lam must be in Matrix format)
filename  = "2D_Vlam.dat"     # Enter Filename (inside Proj_name/MP_files/)
sep       = ','   # data separation: ',' comma, '\t' tab, '\s+' multiple spaces
```

Similar to the PES input file, the *job* type is *FnFit*, *Proj_name* is the folder name where MP expansion was carried out and *filename* contains the file name of radial terms i.e. either '*2D_Vlam.dat*' for 2D PES or '*4D_Vlam.dat*' for 4D PES. The separation is by the default comma (,) unless the file has been obtained from an external source.

The PES and V_Λ both use the same *FnFit* module to fit the data into an analytical expression. However, the *Fnfit_type* is `'Vlam'` for V_Λ as shown above, as opposed to `'PES'` that is used for 1D/2D/4D PES.

```
#------------------------------ For 2D PES use ------------------------------#
# If coll_typ = '2D' , uncomment 'symmetric =' and comment 'Lmat ='
symmetric = True        # Verify if rigid rotor is symmetric (else put False)

#------------------------------ For 4D PES use ------------------------------#
# If coll_typ = '4D' , uncomment 'Lmat =' and comment 'symmetric ='
Lmat = 'Lambda_ref.dat' # for 4D enter file containing lambda terms
#----------------------------------------------------------------------------#
```

Then, for 2D collisions, users must specify whether the system is symmetric or not. For 4D collision, use the other command *Lmat* which will read the files having combinations of λ_1, λ_2 & λ.

```
#---------- If PES is already fitted into analytical expression -------------#

# To force 0 for all (recommended if PES is fitted using FnFit_PES)
# Change values if convergence fails, [0,1,2,4...]
start_pos = [0]*1000 # Set starting position for fitting radial term

#----------- If PES is NOT fitted into analytical expression ----------------#

cutoff    = 3000    # Energy cutoff in cm-1.
          # --> For attractive potential code automatically switches to -cutoff
```

Fitting V_Λ terms into a mathematical expression (here: series of Slater functions) is one of the most important steps. Users can either choose the same function they use for fitting the PES (with fixed β values)

or try various functions till a good fit is achieved (can be verified with residual plot).

For the first case, users can comment the cutoff variable and keep *start_pos* = [0]*1000, where [0] designates the starting point and 1000 is the maximum number of Lambda terms.

```
    [0]*5 is equivalent to [0,0,0,0,0], and,

    [-1]*2 + [4] is equivalent to [-1, -1, 4]
```

If someday the Lambda terms go beyond 1000, users can increase the value. In the first case (PES fitted into fixed β Slaters), the fitting never fails and results have error in the order of $< 10^{-10}$.

However, if *ab initio* or NN augmented PES is used, where PES has not been fitted into radial functions, the fitting can be done by:

- Using *cutoff*, where the program recognizes the repulsive and attractive nature of radial terms and uses either +cutoff or -cutoff value for determining the starting position of fit. This, however, is **not recommended** as radial terms can have multiple minima/maxima.
- Using *start_pos* and allowing β coefficients to relax. In this procedure, the starting position can be changed for each radial term (trial and error) to obtain a good radial fit. E.g., consider that there are 4 radial terms obtained after expanding *ab initio* PES. Suppose the first three radial terms fit perfectly, while the fourth fails. Here, users can try changing values of the starting position by giving values like: [0,0,0,1] or [0,0,0,2], or [0,0,0,3] to see if starting from N^{th} position (rather than the 0^{th} position) can fit the radial term. As is obvious from the above instructions, the procedure requires trial and error with constraints on upper and lower bounds for β. An example of the same is provided in the GUI example (`1_NoFitPES`).

```
1
2  #-------------------------------------------------------------------------#
3  # Use below code to define custom analytical function for radial fitting
4  # Use same templates as FnFit_PES just use -inf for lower_bounds
5
6  def fnfit_custom(x, a1,a2,a3,a4,a5,a6,a7,a8):
7      import numpy as np
8      return  a1*np.exp(-5*x)+a2*np.exp(-3.75*x)+a3*np.exp(-3*x)+a4*np.exp(-2.75*x) + \
9              a5*np.exp(-2*x)+a6*np.exp(-1.75*x)+a7*np.exp(-1*x)+a8*np.exp(-0.75*x)
10 initial_val  = [1e4]*8                    # Enter initial guess
11 #---------------------------- OPTIONAL ----------------------------------#
12 #scale_Energy = 1       # multiplies cm-1 energy to scale_Energy
13 #scale_R      = 1       # multiplies Ang. R diatance to scale_R
14 #-------------------------------------------------------------------------#
15 # Enter data about custom function for printing MOLSCAT &POTL File
16 import numpy as np
17
18 Exp_fns = 8                        # number of Exponential functions
19 N_Opt   = False                    # Are N values optimised (A.e^(-Nx))
20 N_Vals  = [-5,-3.75,-3,-2.75,2,-1.75,-1,0.75]      # If N_Opt=False, enter N coeffs (in order)
21 A_sign  = [+1]*Exp_fns             # Enter signs of coeffs A (in order)
22 Pair_fns = False        # pair func = a1*(np.exp(-3.1*x)+np.exp(-3.0*x))...
23
24 Print_raw = True                 # Raw coefficients for later formatting
```

In the final step, the analytical expression (mathematical function) used to fit radial terms is provided, similar to *FnFit PES*. The program generates the MOLSCAT potential file after fitting the V_Λ terms, which require some parameters to be set. The same are provided below:

1. Exp_fns: The number of exponential functions.
2. N_Opt: If β values are optimised, enter True. If β values are fixed like above, set False.
3. N_Vals: If N_Opt is True, comment this line. If N_Opt is False, like the above case, provide the fixed β values in order.
4. A_sign: While users can keep all α coefficients positive like above, there are cases where users may constrain α values to positive and use expressions like $\alpha_1 e^{-\beta_1 x} - \alpha_2 e^{-\beta_2 x}$. Since the function now contains a negative sign, its $A\_sign = [+1, -1]$. For simplicity, keep all functions connected by +ive sign and leave this variable as it is.
5. $Pair\_fns$: If users use function like $a_1(e^{-3.5*x} - e^{-3.0*x}) + ...$, they can set Pair_fns = True, else leave False. In practice, such functions are found to usually underfit the data.
6. $Print\_raw$ = True: This variable prints the coefficients in a raw text format alongside MOLSCAT readable format. Leave True.

3.4.8 *Residuals*: Inverse MPExp and Residual Plots

While the above step may seem final to users, there is one more thing that must be verified before the radial terms (expressed in mathematical function) can be used for dynamical calculations in MOLSCAT: the inverse fitting of V_Λ terms into PES.

Since the number of generated radial terms (V_λ) can be less than the original number of angles, users can expect deviations at high energies. However, at minima regions, the deviations are minimal (See Fig. 4.13).

While the residual plot is generated after each fitting step (i.e., PES FnFit and NNGen), users can also inverse fit V_Λ terms and regenerate PES and do a last sanity check by comparing the same with *ab initio* PES. This will give the overall error or all transformations and must be within spectroscopic accuracy for good dynamical results. The input file to accomplish the same is provided below:

```
################################################################################
############################## MP Expansion of PES ##############################
################################################################################
# Gives residual plot by recreating PES from V_lambda. Must use after FnFit.
# The raw V_lam will not give any error, but analytically fitted V_lams will!
# This provides an estimate for the quality of fit for the PES.

# The PES must be located inside Projects/Proj_name
# The V_lam file must be inside Projects/Proj_name/MP_files
#-------------------------------------------------------------------------------#
MPExp = True              # Set job-type
Proj_name = 'test'      # Folder name inside "Projects" folder
Residuals = True
#-------------------------------------------------------------------------------#
# Reference PES (No Header)
PES_filename_cm  = "ab_initio_PES.dat"     # Enter Filename (Energies in cm-1)
sep = ','    # data separation: ',' comma, '\t' tab, '\s+' multiple spaces
#-------------------------------------------------------------------------------#
Expansion_typ = '2D'      # 2D i.e. RR-atom collision PES expansion

# Read Legendre Coefficients
read_Legendre = True  # set read_Legendre/read_SH to True (2D/4D)

symmetric = True     # Verify if the rigid rotor is symmetric (else put False)
#-------------------------------------------------------------------------------#
```

The users must provide the project name and keep the rest of the parameters the same. They must enter the PES file (*PES_filename_cm*) and separator (',', tab, etc.) with which they would like to compare the recreated PES.

The expansion type must be mentioned (2D/4D) in *Expansion_typ*. For 2D collision use *read_Legendre* = True and set if the rigid rotor is symmetric or not. For 4D collision, just set *read_SH* to True.

```
# Enter format for residual plot
fmt     = 'pdf'     # format for created plots, options = pdf, eps, png, etc.
scale_x = 'symlog'  # 'symlog' or 'linear'
scale_y = 'symlog'  # 'symlog' or 'linear'
Y_lim   = [-1,1]    # E (residual) Y limit for zoomed in plot
cutoff  = 100       # E (residual) X limit for zoomed in plot
########################### Optional Commands ##################################
## If Energies are in Hartree, Enter E_inf below for conversion. (Optional)  ##
#E_inf = -188.31099452      # define E_infinity (Asymptotic Energy)
################################################################################
```

The above parameters are used to change residual plots, where plots are provided in 'pdf', x and y scales are in symmetric-log, and y limit (residual error) is from -1 to 1 (cm^{-1}). The cutoff denotes the high-energy cutoff (x-axis) in the zoomed-in plot. A full range plot is also generated alongside the zoom plot. Optionally, users can set *E_inf* to convert the provided PES from Hartree to cm^{-1}.

3.5 Auxiliary files

The auxiliary files (not part of the automated program) are discussed in this subsection. These scripts are in either written in bash or Python. Some Python codes may require external libraries, which have been installed in the PES2MP environments. Therefore, if the Python program shows `missing package error`, the users must activate either `pes2mp` or `pes2mp_q` environment before executing the Python script.

3.5.1 Running PES calculations using batch file (*batch_scripts*)

The PES input files are numbered from 0 to N, so the batch script can be used to run the same as discussed below. Suppose PES input files (for Psi4, Gaussian or Molpro) are inside the *input_file/PES_CP/* folder. Keep the batch file outside the PES folder (i.e., in *input_file/*) and enter */PES_CP/* folder location in the batch script as shown below. This has been done since the */PES_CP/* folder may be too crowded (4D PES), and it may take a long time to load all the files (Linux GUI problems). The batch file will enter the folder, run the calculation, and exit. In the next subsection, the Python files for extracting Molpro and Gaussian PES are discussed.

A simple generic loop is explained below:

```
for k in {0..7237}
do
psi4 ${k}.inp
done
```

that can also be entered as a one-line command in the terminal (not using the batch script):

```
for k in {0..7237}; do; psi4 ${k}.inp; done
```

where the loop runs from 0 to 7237 with a step size of 1 and assigns this value to k. The script will execute the command `psi4 $k.inp` for each value of k. In bash, the loop runs from initial value to final value, as opposed to Python, where the loop runs from initial to final-1 (penultimate) value. If the calculations have to be performed on servers with a dedicated job manager (that uses a jobscript file), just make the scripts executable (`chmod +x run.sh`) and add the line `./run.sh` at the end of the jobscript file.

Gaussian calculations: Users must load the Gaussian directory as shown below. It sources either `g16.login` or `g16.profile` script in `...g16/bsd/` directory. Users can also use keywords to assign a temporary/scratch folder in the script using `export GAUSS_SCRDIR= ${PWD}/scratch/`.

```
. /home/dummy_path/g16_lin/gaussian_vars_16.sh # Load Gaussian script with scratch dir information

cd Gaussian_CP                                 # Change directory to the location of input files
for k in {0..7237}                             # Loop to run Gaussian Calculations
do
g16 $k.gjf
done
```

Molpro: As stated in the Gaussian calculations, a scratch folder is created and assigned for temporary files in the script. The `$PWD` command refers to the `present working directory`, hence `mkdir $PWD/molpro_tmp` creates a `molpro_tmp` folder in the current directory. The `export` command redirects large temporary files (integrals and checkpoint files) into this folder. ***Choose non-OS HDD/SSD drive*** *for scratch directory (so no Desktop, Documents, Downloads, etc. folders), else your PC will start to hang. If your PC does not have an extra HDD/SSD installed, get one!*

In the loop, users must replace the dummy location `/usr/local/molpro/dummy_location/bin/molpro` with the actual location of the Molpro executable in their system. The `-n 16` refers to the use of 16 processors.

```
mkdir ${PWD}/molpro_tmp                  # Make a folder for molpro temporary files (Optional)
export TMPDIR="/${PWD}/molpro_tmp"       # Use custom scratch folder (Optional) for TMP  files
export TMPDIR4="/${PWD}/molpro_tmp"      # Use custom scratch folder (Optional) for TMP4 files

cd Molpro_CP                             # Change directory to the location of input files
for k in {0..7237}                       # Loop to run Molpro Calculations
do
/usr/local/molpro/dummy_location/bin/molpro -n 16 $k.inp
done
```

For Psi4, a scratch (temporary) directory is created (`mkdir`: make directory) inside the current folder `${PWD}/psi4_tmp` (again, for large calculations, make sure that the files are on a non-OS drive, i.e., an additional HDD/SSD). Once the calculations are done, the PES is automatically stored in the *PES.dat* file (`psi4 $k.inp >> ../PES.dat`) in the same directory where this batch file is kept.

```
mkdir ${PWD}/psi4_tmp                  # Make a folder for molpro temporary files (Optional)
export PSI_SCRATCH=/${PWD}/psi4_tmp    # Use custom scratch folder for temp files

cd psi4_custom                         # Change directory to the location of input files
for k in {0..118}                      # Loop to run Psi4 Calculations
do
psi4 $k.inp >> ../PES.dat
done
```

Remember: Do not use the OS disk for the temporary files. It will slow down your PC/workstation. Use a secondary hard disk or, as a last resort, use an external (Warning: USB-connected disks generally have higher latency and slower speeds) HDD/SSD.

3.5.2 PES: Extract V (Hartree) and Jacobi coordinates (*PES_extract*)

Psi4 script automatically saves PES (with Jacobi coordinates and energies in Hartree) into the *PES.dat* file. For Molpro and Gaussian, Python scripts are provided to extract the PES data along with the Jacobi coordinates. **Extracting each data point manually can take days (2D PES) to months (4D PES).** Keep the file outside the folder containing input files (next to the batch file) and enter the parameters in the Python script as required.

The only parameters needed are: (a) the folder name where output files are located (`PES_folder`), (b) the number of output files (`num_files`) having PES data for individual Jacobi coordinate, and (c) the filename for the final output file (`f1`) that will contain the extracted PES. The variable `f2` stores data for error files and can be left as it is.

```
path = os.getcwd()                       # Current path ($Proj_name/input_files/)

###############################################################################
PES_folder = path + '/Molpro_CP/'        # or '/Molpro_CBS/' or '/Gaussian_CP/'

f1 = open(os.path.join(path, "PES.dat"), "w+")
f2 = open(os.path.join(path, "PES_err.dat"), "w+")

num_files = 6371
###############################################################################
```

3.5.3 PES: Hartree to cm^{-1} (*PES_to_cminv*)

Once the PES is extracted, users can proceed with it by using the asymptotic (R) energy (as reference energy) for one of the angles in the PESPlot module. It will convert the PES from Hartree to cm^{-1}, save it into a new file, and plot the data.

However, if the PES is generated using non-size-consistent methods such as CI or F12 methods, this can lead to incorrect asymptotic values. Here, users can use the Python script (*PES_to_cminv.py*), which takes the asymptotic energy (R_∞) for each angle and subtracts the same from energies at other R points. The program automatically detects the final R data available for each angle in the input file.

Suppose, all angles are calculated till 100 Å but one of the angles has maximum R up to 50 Å (due to convergence failure or is manually removed due to incorrect value), the program will automatically use 100 Å for all other angles and 50 Å for that particular angle. The only input required for this script is the filename *PES.dat* and its separator (*sep*), i.e., spaces, tab, comma, etc.

```
data = pd.read_csv('PES.dat', sep='\s+' , header=None)
```

There are three Python files for 1D, 2D, and 4D collision. Just make sure that the input file name is correct and the output file name is unique. The column header is handled by the program; however, if there is any error, remove the header and check for the separator (the default separator is multiple spaces \s+).

3.5.4 MOLSCAT: Generating input files (1_Gen_molscat_files.py)

This Python script also requires the `MOLSCAT_POT.txt` generated by PES2MP (after V_Λ terms are fitted into an analytical expression). Just keep this file next to the Python script and run. The script generates separate files for different collisional energies to counter the problem of lost time if any power failure occurs (also, the MOLSCAT program is not parallelized, i.e., it runs on a single thread).

To generate MOLSCAT input files, with different step sizes for different ranges of collisional energies, users can manually add multiple loops as shown below:

```
################################ Input Parameters ################################
# Add loop by increasing counter (j) as j, j1, j2 ... Keep the last counter as jF

# Use fractional values only when step size < 1
j1 = loop (0.05, 30.0,  0.05, j)     # initial / final value / step size / counter
j2 = loop (30.1, 100.0, 0.1,  j1)    # initial / final value / step size / counter

# When step size > 1, do not use fractional values
j3 = loop (101,  200,   1,    j2)    # initial / final value / step size / counter
j4 = loop (210,  500,   25,   j3)    # initial / final value / step size / counter
j5 = loop (600,  900,   100,  j4)    # initial / final value / step size / counter
jF = loop (1000, 2500,  500,  j5)    # initial / final value / step size / counter
```

Contrary to other input files, the input parameters for the above file are located at the end of the file. The input calls the *loop* function multiple times in sequence that has an initial value, a final value, a step size, and a counter (that is `j` for the first loop and then continues as `j1, j2`...) The `j1, j2,...` are in sequence so, users can add a new `j` as:

```
....
j6 = loop (1000,  2500,   500,  j5)    # initial / final value / step size / counter
jF = loop (3000, 10000,  1000,  j6)    # initial / final value / step size / counter
```

where `j6` is added to the loop with `j5` as the counter and the counter for `jF` now becomes `j6`.

There is only one constraint. The initial and final values must be given as decimals if the step size is <1. If the step size >1, use integers for all three values.

The potential file `MOLSCAT_POT.txt` (that was generated by the PES2MP program) is loaded in variable `potf` and printed in MOLSCAT's *&potl* block. The users must ensure that parameters associated with potential are correct, such as: scaling of R (`rm=1.0` denotes no scaling), scaling of potential V_Λ (`epsil=1.0` denotes no scaling), and number of V_Λ terms (`mxlam=174`).

```
potf = open("MOLSCAT_POT.txt", "r+") # POT: reads potential from 'MOLSCAT_POT.txt' file

f1.write('  &input ured = 1.94051, nnrg=1, energy=%.4f\n' %(i* step))

if (i*steps < 10):
    f1.write('   intflg=8, steps=100, rmin=1.5, rmax=50.0, BCYOMN=10000, \n')
elif ( (i*steps > 10) and (i*steps < 30) ):
    f1.write('   intflg=8, steps=50, rmin=1.5, rmax=50.0, BCYOMN=10000, \n')
else:
    f1.write('   intflg=8, steps=20, rmin=1.5, rmax=50.0, BCYOMN=10000, \n')

f1.write("   label='H2-cncn system', jtotu = -1, \n")
f1.write('   prntlv=1, isigpr=1, LASTIN = 1,\n')
f1.write('/ \n')

f1.write(' &basis itype=3,  j1max=37, j2min=0, j2max=2, j2step=2 \n')
f1.write('    be= 0.1726,60.853,\n')
f1.write('/ \n')
f1.write(' &potl rm=1.0, epsil=1.0, mxlam=174, \n')

f1.write(potf.read())  # print potential file
```

While some of the required parameters are explained here, this is a dummy file for a 4D RR-RR collision. Refer to the MOLSCAT manual and published papers for selecting parameters such as `intflg, steps`, maximum rotational basis (`j1max`), etc. These can change from case to case depending on the rotational constant, potential well, and anisotropy of the system.

For example, simple neutral systems can be terminated at `rmax = 20` Å; however, charged anionic systems can show interaction till 50-200 Å. The systems with large well depths need a larger rotational basis for the convergence of cross-sections. The STEPS parameter must be large (smaller step size and tighter integration) for small collisional energies, but as the collisional energies become larger, this value can be reduced. Therefore, the `if else` command with `i*steps` represents the collisional energy, and for different collisional energies, different step size is used. Users can introduce the same `if else` block for changing `j1max`, which typically increases with collisional energies as more and more closed channels become accessible.

These are some of the examples. One must benchmark the system to see if the cross-sections are converging with the range of integration, step size, and rotational basis for various regions of collisional energies. Once satisfied, only then proceed with full CC (close coupling) calculations. *Why?* These calculations are very expensive, and it all comes down to the desired accuracy of the user for the rotational transitions under inspection. For very large collisional energies (where resonances completely diminish), the CS (centrifugal sudden) approximation can be used using `itype=21` (2D case) and `itype=23` (4D case). These calculations are faster, but must be benchmarked with CC to check the error associated with the approximation with increasing collisional energy.

3.5.5 MOLSCAT: Extracting $\sigma(i,j)$ ($2\_result\_extract\_molscat.py$)

Once the MOLSCAT calculations are done, the cross-sections (σ) are collected into a single *sigma.dat* file using this script. Like PES, keep this script outside the folder containing MOLSCAT input and output files. The folder name is specified in `sig_folder`, and the two output files have cross-sections and pair energies (quantized energy levels among which cross-sections are calculated).

```
################################################################################
#################### Input parameters (make changes here) ######################
################################################################################

# Enter folder name (keep this file outside the folder containing molscat files)
sig_folder = 'P0'            # Folder name containing MOLSCAT input/output files

filename = 'sigma.dat'       # output file name  (cross sec)
filename_PE = 'pair_E.dat'   # output file name (pair Energy)

ini   = 1                    # initial file number
final = 1000                 # final file number

# The final file number will be used for extracting the Pair Energies
```

3.5.6 Calculating Rate coefficients ($3\_molrates.py$)

For rate coefficients, users must enter the maximum temperature `tmax` (default: 1K to `tmax` with 1K step size), reduced mass of the colliding pair (`redm`), the sigma file name (`sigma_file`) extracted using above code, and names for the output files (containing rate coefficients).

Here, the rate coefficients are calculated using both the summation (output file: `fr = f"k_py.dat"`) and the integration (output file: `fr_int = f"intk_py.dat"`) method. This serves as a sanity check since results with both should be the same (<0.01% difference), as they are numerically equivalent when the input data (collisional energies) are dense. If the rates vary substantially, include additional collisional energies to make the dataset dense.

```
################################################################################################
#################### Input parameters (make changes here) ######################################
################################################################################################
#----------------------------------------------------------------------------------------------#
tmax = 200    # maximum temperature in kelvin
redm = 1.9354 # reduced mass in amu
#----------------------------------------------------------------------------------------------#
# File name for input (extract using result_extract_molscat.py)
sigma_file = "sigma.dat"   # name of the (extracted) file containing cross-sections
#----------------------------------------------------------------------------------------------#
# File names for output (will contain rate coefficients till tmax)
fr = f"k_ex.dat"                  # output rate file using summation method
fr_int = f"intk_ex.dat"           # output rate file using scipy integration
sig_out = "sig_ex.csv"            # output cross-section file for selected transitions

#----------------------------------------------------------------------------------------------#
# transitions required! Set all_tr to false and select initial and final j
all_tr = False    # calculate all transitions (not recommended: keep false)
#----------------------------------------------------------------------------------------------#
```

Users can calculate multiple transition data points using the same input file. They can also use `all_tr` to calculate rate coefficients for all available transitions in the `sigma.dat` file. This is not recommended as some transitions may have very few data points, and consequently, their rates may have substantial deviations. However, users can verify this from the `sig_out` file that extracts the cross-sections of specific transitions and provides them as columns. Therefore, transitions with very few data points (usually happen for large j) can be discarded manually from the final output file.

For final results, the Riemann summation (`fr`) is the simplest first-order method (easy and fast), and is more tolerant to noisy fluctuations. The Simpson integration (`fr_int`) method is a 4th-order approximation method and should be more accurate; however, it is more suited to smooth data and therefore can misbehave if enough data is not present. Overall, in the limit of a dense grid, both methods are accurate. If there are substantial differences between the two, try Trapezoid (2nd-order) method that offers a midpoint between the two methods (more accurate than Riemann, but still stable under noise) for sparse and noisy data. To implement the same, replace `I = scipy.integrate.simpson(res)` with `I = scipy.integrate.trapezoid(res)`.

The other integration method (available for discrete datasets) in scipy package are:

```
I = scipy.integrate.romb(res)
I = scipy.integrate.cumulative_simpson(res)
I = scipy.integrate.cumulative_trapezoid(res)
I = scipy.integrate.trapezoid(res)
I = scipy.integrate.simpson(res)
```

and can be found in `/aux_scripts/rate_codes_MOLSCAT/Extra/molrate.py`.

The transitions must be labeled using the MOLSCAT's notation (N), i.e., it starts from 1 and takes into account only the `PAIR LEVEL` (not rotational states) specified by `j1min/j1max/j1step` and `j2min/j2max/j2step`. Therefore, it is recommended to check out the output file and verify the levels 'N' to be entered below.

Users can either use NumPy to enter the transitions or, if they are not familiar with the same, they can manually enter each transition as a list provided in the final example.

```
# molscat labels start from 1 (don't give ji/jf values with 0) [see templates below]
ji = np.arange(2,5,1)     # 2 to 4      [2,3,4]
jf = np.arange(1,4,1)     # 1 to 3      [1,2,3]
#-------------------------------------------------------------------------------------#
# Example template for de-excitation delta j=1 (first 20 transitions)
#ji = np.arange(2,21,1)    # 2 to 20       [2,3,4,.....,19,20]
#jf = np.arange(1,20,1)    # 1 to 19       [1,2,3,.....,18,19]

# template for excitation delta j=1
#ji = np.arange(1,20,1)    # 1 to 19      [1,2,3,.....,18,19]
#jf = np.arange(2,21,1)    # 2 to 20      [2,3,4,.....,19,20]

# template for de-excitation delta j=2
#ji = np.arange(3,21,1)    # 3 to 20      [3,4,5,.....,19,20]
#jf = np.arange(1,19,1)    # 1 to 19      [1,2,3,.....,17,18]

# template for excitation delta j=2
#ji = np.arange(1,19,1)    # 1 to 19      [1,2,3,.....,17,18]
#jf = np.arange(3,21,1)    # 3 to 20      [3,4,5,.....,19,20]

# template for specific transitions (transitions provided as list)
#ji = [1,1,1,2,3]         # example transitions like 1-1, 1-2, 1-3, 2-1, and 3-1
#jf = [1,2,3,1,1]
```

Now, to convert the label `N` into rotational states `j` directly within the program, users can specify three parameters. For He (2D) collisions set `subtract_1`. This will subtract 1 from each label to give `j`. However, if the molecule only has even rotational states such as C_2, set `even_j1`, which will subtract -1 from `N` and double to give `j`. Finally, for 4D collision, if the molecule also has two rotational states, e.g., $j_2 = 0, 2$ for `p`-H_2 collision, setting *two_j2* to true will subtract 1 and halve the `N` to give `j`.

If this confuses you, simply keep all of them false and enter the transitions needed above. The program will give the rate for those transitions and will not change any labels. The labeling can be handled later by the user.

```
# Remember molscat starts j from 1. Change labels to rotational states as shown:
# To keep labels as it is, set all of them False.

# If RR only has even values like C2, C3, etc.            set      even_j1 = True
# If RR only has both even and odd values like CO, NCCN.  set   subtract_1 = True
# If two J2 cases: PO2 i.e. (J2=0,2)                      set       two_j2 = True
#-------------------------------------------------------------------------------#
# change label (x-1) to convert 1 to 0 and so on. (He and 1 rotational state of H2)
subtract_1 = False       # 1-->0, 2-->1 , 3-->2, etc...
#-------------------------------------------------------------------------------#
# Special Cases
#-------------------------------------------------------------------------------#
# For cases when only even states are present for cases with I=0 like C2, C3 etc.
# Subtracting 1 from States and doubling to give J (Keep subtract_1 = False)
even_j1    = True        # 1-->0, 2-->2 , 3-->4, etc...
#-------------------------------------------------------------------------------#
# change label (x-1)/2) (case where 2 rotational state of H2 are included in basis)
# Subtracting 1 from States and halving to give J (Keep subtract_1 = False)
two_j2     = False       # 1-->0, 2-->2 , 3-->4, etc...
#-------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------#
# subtract energy: take relative energy for (de-excitation transitions)
sub_E = False   # If true, rotational energy (from pair_E file) is subtracted
pair_E = np.loadtxt("pair_E.dat")  # name for the file containing rotational energy
#################################################################################
```

For de-excitation transitions, the reference rotational energy of the initial state must be subtracted from the total energy to give kinetic energy. This is done by setting `sub_E` to True. The `pair_E.dat` file generated with *sigma.dat* has these reference energies and can be automatically used. For special cases like *o*-H_2 excitation collision, this file must be edited manually. An example of the same is available in `pair_E_o.dat` located at `GUI_examples/2_FnFitPES/4D/MOLSCAT/o1-H2` where the `PAIR LEVEL` is kept same while the `PAIR ENERGIES` are changed to the same value (rotational energy of *o*-H_2). This subtracts the same from the collisional energies (the sum total of the rotational energy of both colliders and the kinetic energy). For de-excitation, the pair energy generated by the previous code (`pair_E.dat`) can be used directly (See Section 2.7.3) since both rotational energy of *o*-H_2 and rotational energy of RR1 must be subtracted to give kinetic energy.

3.5.7 Generating Series of Slater functions (*Fn_generate.py*)

There are two Python scripts in `input_files/3_Fit_PES/Extra`: *Fn_generate_S.py* generates simple Slater functions in sequence, and `Fn_generate_P.py` generates a sequence of paired Slater functions in

sequence as shown below in two examples:

$$a_1 e^{-3x} + a_2 e^{-2x} + a_3 e^{-1x} + a_4 e^{-0.25x} \tag{3.2}$$

$$a_1(e^{-3x} - e^{-2x}) + a_2(e^{-1x} - a_4 e^{-0.25x}) \tag{3.3}$$

Both are valid inputs for MOLSCAT's &POTL function. However, we shall focus on the first function. Suppose Someone needs to use 8-10 Slater functions to fit the data within the β range 0.05 to 6. They can directly enter the value of `N_exp` below to 8 and change `beta_c` to `np.linspace(0.05, 6, N_exp)`, which will generate eight equally spaced values between 0.05 and 6. Alternatively, users can provide the list themselves as shown in the commented example `beta_c = [3,2,1,0.25]`.

```
N_exp  = 4                                # number of exponential functions
beta_c = np.linspace(0.06, 6, N_exp)      # beta coeff range (automatic generation)
                                          # initial value, final value, no. of points

# beta_c = [3,2,1,0.25]              # beta coeff range (manual input)
```

For the above input, `N_exp = 4` and `beta_c = [-3,-2,-1,-0.25]`, the following output will be created:

```
def fnfit_custom(x ,a1,a2,a3,a4):
    import numpy as np
    return a1*np.exp(-0.3*x) +  a2*np.exp(-2*x) +  a3*np.exp(-1*x) +  a4*np.exp(-0.25*x)

initial_val  = [1e4]*4
```

For `N_exp=16` and `beta_c = np.linspace(0.05, 6.05, N_exp)`, the following output will be created:

```
def fnfit_custom(x ,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16):
    import numpy as np
    return a1*np.exp(-0.0500*x) +  a2*np.exp(-0.4500*x) +  a3*np.exp(-0.8500*x) +  a4*np.exp(-1.2500*x) +  \
           a5*np.exp(-1.6500*x) +  a6*np.exp(-2.0500*x) +  a7*np.exp(-2.4500*x) +  a8*np.exp(-2.8500*x) +  \
           a9*np.exp(-3.2500*x) +  a10*np.exp(-3.650*x) +  a11*np.exp(-4.050*x) +  a12*np.exp(-4.450*x) +  \
           a13*np.exp(-4.850*x) +  a14*np.exp(-5.250*x) +  a15*np.exp(-5.650*x) +  a16*np.exp(-6.050*x)

initial_val  = [1e4]*16
```

The script automatically adds a '\' (denotes continuity between codes spread over multiple lines) and switches to a new line after four terms to prevent cluttering.

The second code `Fn_generate_P` generates these Slater functions in pairs. For example, the code provided below defines the β values of positive and paired (negative) Slater functions using the two lists:

```
N_exp  = 4                                # number of pair exponential functions
beta_c = np.linspace(5, 0.2, N_exp)       # beta coeff range (automatic generation)
                                          # initial value, fin value, no. of points

diff_c = [0.5, 0.2, 0.1, 0.05]            # Difference between pair coefficients
```

where the paired Slater (one with the negative signs) will have a smaller β value as provided in `diff_c`. The paired functions do not go to $-\infty$ at $R \to 0$. Apart from that, they do not offer much improvement over simple Slater functions and can be used for exploring functional flexibility. The output for the above script is:

```
def fnfit_custom(x ,a1,a2,a3,a4):
    import numpy as np
    return a1*(np.exp(-5.0000*x) - np.exp(-4.5000*x)) +  a2*(np.exp(-3.4000*x) - np.exp(-3.2000*x)) +  \
           a3*(np.exp(-1.8000*x) - np.exp(-1.7000*x)) +  a4*(np.exp(-0.2000*x) - np.exp(-0.1500*x))

initial_val  = [1e4]*4
```

3.5.8 Generating PES from Fitted Coefficient ($fn2pes.py$)

When a 1D/2D/4D PES file is fitted into any analytical expression, its coefficients are saved in an appropriate format in `/PESFnFit/` folder as `pes_fn.txt` file. Using the same function and the coefficient file name, the PES will be regenerated (by `fn2pes.py`) over the range of R provided by `R_range`.

```
############################## INPUT PARAMETERS ##################################
#-----------------------------------------------------------------------------#
def fnfit_custom(x, a1,a2,a3,a4):     # Custom function
    import numpy as np
    return  a1*np.exp(-4*x) + a2*np.exp(-3*x) + a3*np.exp(-2*x) + a4*np.exp(-1*x)
#-----------------------------------------------------------------------------#
Num_coeff = 4                         # number of coefficients
R_range   = np.arange(1,20.1,0.1)     # range of R
#-----------------------------------------------------------------------------#
# Define input & output files
input_filename = "pes_fn.txt"         # filename having pes coefficients
output_dataframe = "pes_fngen.dat"    # filename for PES in dataframe format
output_matrix = "pesmat_fngen.dat"    # filename for PES in matrix format (2D/4D)
#-----------------------------------------------------------------------------#
```

In the above Python file, users must enter:

- `fnfit_custom`: custom function in which the PES was fitted,
- `Num_coeff`: Number of coefficients in the custom function,
- `R_range`: Range of R in 1D array format.

`np.arange(1,20.1,0.1)` generates values from 1 to 20 (penultimate value: Python Loops are designed this way for a reason! See Appendix A.2.2). Apart from these, users must enter the names of input and output files as described below:

- `input_filename`: input file having pes coefficients,
- `output_dataframe`: output file for saving PES in dataframe format (R, theta, E),
- `output_matrix`: output file for saving PES in matrix format (2D).

Each coefficient file contains a header (1D/2D/4D) to indicate the type of collision. An example of the coefficient files for 1D/2D/4D PES are provided below:

```
1D
1659166.398014146,2.77899841066747,6.618344948614832
```

Listing 3.19. 1D PES represented analytically as a function of R.

The 1D PES has 3 coefficients with values as provided above.

```
2D
0
83306731.13576201,1942613.99812364,-61341.30952697418,-834.4042920590489,-2.579572848619542
30
58481834.30238403,2060635.05343777,-60838.80197625618,-671.0945766358003,-2.517358547879758
....
```

Listing 3.20. 2D PES represented analytically as a function of R.

The 2D PES is fitted into an analytical function with 5 coefficients. The value above the coefficients refers to the angle $\theta(°)$.

```
4D
0,0,0
-138172924.5375377,19953212.75080189,-334517.8068582663,-534.7028719858957
30,0,0
-138172924.5375377,19953212.75080189,-334517.8068582663,-534.7028719858957
60,0,0
-138172924.5375377,19953212.75080189,-334517.8068582663,-534.7028719858957
....
```

Listing 3.21. 4D PES represented analytically as a function of R.

Similar to 2D PES, the first line in 4D PES indicates the dimension. The following alternating lines correspond to the angle combination in the order ϕ, θ_2, θ_1, and fitting coefficients. Here, the PES is fitted into an analytical function with 4 coefficients. The optimized coefficients are stored in the order they were provided in the fitting function `fnfit_custom(x, a1,a2,a3,a4)`, i.e., a1, a2, and so on.

The output is provided in both dataframe and matrix formats. The advantage of saving PES into a dataframe format (R, theta, E) is that each row clearly represents one data point and is good for filtering and analysis. The data in this format can be directly used for ML applications. There can be missing data points in this format for any specific coordinate.

In matrix format, missing data can result in erroneous assignments. Therefore, missing values must be manually filled with placeholders like `np.nan` or special flag values. However, for plotting purposes (if no data is missing), it is better to use the matrix format (2D), where rows are R, the columns are θ or $\theta(s)$, and entries are energies (E). Such data is suitable for plotting both 3D PES (R, θ vs E) and simple (R vs E) PES, where individual angles are represented by different colors. But the biggest advantage of the latter is file size. Since the coordinates are not repeated, this kind of file will be $3-5$ times smaller in size than the dataframe.

4 Results and Discussion

In this section, we shall discuss the results of the practice input files provided in the `GUI_examples/` folder for various cases introduced in the previous chapter. The following sections will discuss four different cases: `0_ExtPES`, `1_NoFitPES`, `2_FnFitPES`, and `3_NNFitPES`. The `999_PES` example is already covered in the previous chapter at Section 3.3.2.

4.1 0_ExtPES: Generating PES Input Files for External Calculations

Generating 1D, 2D, and 4D PES files for external calculations (on a server or workstation).

The three input files `1_pesgen1D.py`, `1_pesgen2D.py`, and `1_pesgen4D.py` are kept in the same folder and are run from the GUI interface. First, the folder location is set to `../GUI_examples/0_ExtPES/` and PES2MP python executable files are copied to the folder by clicking the `'Copy PES2MP Files'` icon next to the `'Browse'` icon. The conda environment can be `'pes2mp'` or `'pes2mp_q'` depending on installation, and the **project name** is kept `'1D'`. Now, in the 1D tab, the *run* command is used to run the Python input file.

Similarly, the other input files are run by changing the project name (project name for each input is chosen as follows: `1D`, `2D`, and `4D`). This ensures that the output files for each input file are written inside different folders, i.e., `Projects/1D/`, `Projects/2D/`, and `Projects/4D/`.

Fig. 4.1 shows the output folder for the 1D calculation.

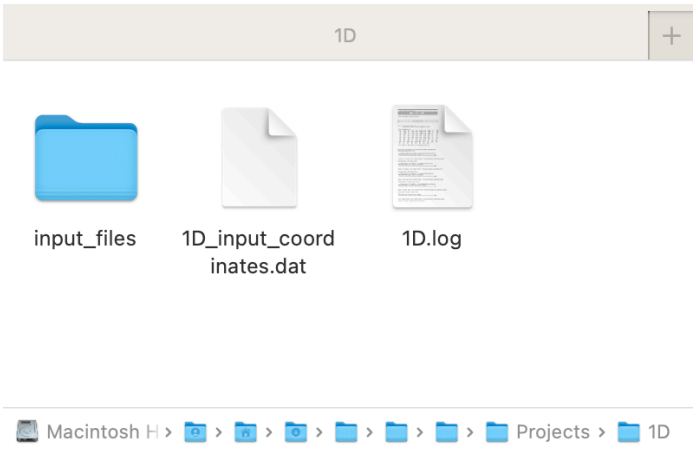


Figure 4.1. Output folder for `0_ExtPES` GUI example (atom-atom collision) with project name `1D`.

The PES files for Molpro, Gaussian, and Psi4 are located inside the `input_files` folder. A preview of the same is shown in Fig. 4.2. Apart from the folder, there are two files: the `1D_input_coordinates.dat`

file contains the input coordinates of the PES, while the log file (`1D.log`) contains the output log for the PES2MP run and has the same name as the project name.

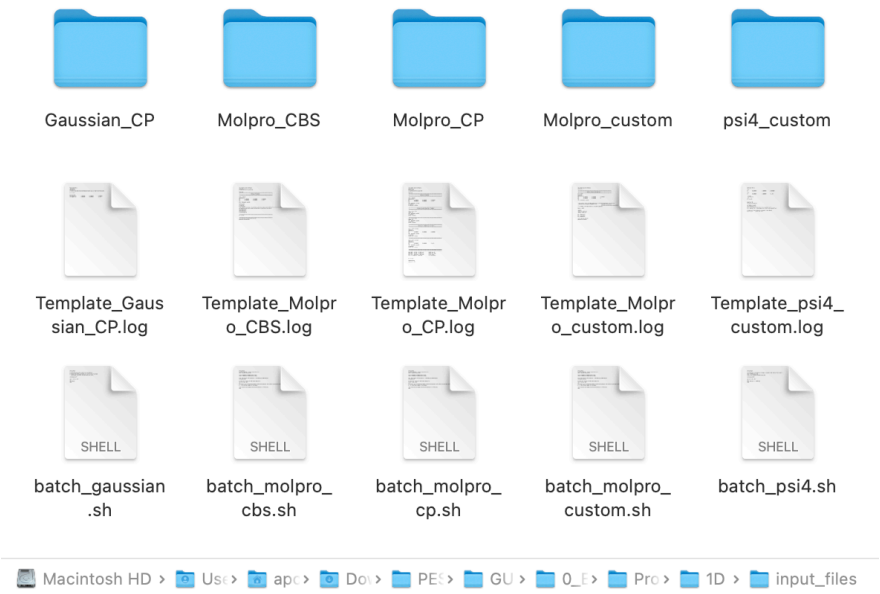


Figure 4.2. Files located inside `input_file/` folder for `0_ExtPES` calculation.

Fig. 4.2 depicts all possible templates that can be generated using PES2MP for external calculations. The folders have the input files (for calculations on any external server), and the template files contain the structure of the input file (for users to see what kind of calculation is set up). The reference files do not have coordinates but rather use symbol $\{:.6f\}$ showing the number of digits (precision up to 6 digits) for that coordinate. If users want more precision, this can be changed by editing the `PES2MP.py` and `PES2MP_driver.py` files.

An example of the Gaussian (CP) template is provided below:

```
%nprocshared=4
%mem=32GB
%chk=1D.chk
# CCSD=(T,SaveAmplitudes,ReadAmplitudes)/aug-cc-pVQZ Counterpoise=2

R = {:.4f}

0,1 0,2 0,2
H(Fragment=1)          0.000000          0.000000          0.000000
H(Fragment=2)          0.000000          0.000000          {:.6f}
```

The batch files shown in Fig. 4.2 are copied from the `input_files/aux_scripts` folder for running the input files. Users who are not familiar with batch scripts can refer to the previous Chapter, where it has been explained in greater detail. The variables in the PES input file are also briefly discussed in the next section.

The batch files for running calculations and Python files for extracting results must be placed outside the PES folders, as shown in Fig. 4.2. The folder names `Gaussian_CP`, `Molpro_CBS`, etc., must be entered in the batch/Python file. The same is implemented to prevent overcrowding and to keep the files organized.

∗Users can create multiple batch files having different ranges (of input file names such as 0-1000, 1000-2000, etc.), to run the same on different nodes/clusters in parallel. ***Why?*** *High-level ab initio methods do not scale well beyond 4-8 processors.*

4.2 1_NoFitPES: Internal Psi4 Calculations

Generating 1D, 2D, and 4D PES files internally using Psi4: In this example, we shall generate the PES internally using Psi4 and then proceed with the multipole expansion of 2D/4D PES.

4.2.1 PES and Multipole Expansion

The various examples in each 1D, 2D and 4D collisions are stated below:

1. 1D: Input files for handling (i) an anion, (ii) a cation, and (iii) a non-singlet (UHF) species. It also has examples for generating (iv) BSSE corrected and (v) CBS extrapolated PES.
2. 2D: Example for generating a (i) rough PES for C_2-He collision, (ii) expansion into radial terms, and (iii) fitting into mathematical expression (also generating MOLSCAT's potential (&POTL) file).
3. 4D: Example for generating a (i) rough PES for C_2-H_2 collision, (ii) expansion into radial terms, and (iii) fitting into mathematical expression (also generating MOLSCAT's &POTL file).

1D case:

In the 1D case, the PES input file uses the internal Psi4 API interface to generate PES and plot the same internally. The PES is generated for Helium's collision with (i) an anion, (ii) a cation, and (iii) a non-singlet (UHF) species. Instructions for generating (iv) BSSE corrected and (v) CBS extrapolated PES are also covered. This is to make users familiar with various features of Psi4 and how they can accelerate PES building. We shall use different project names (separate folders) for the five cases mentioned above.

The input files for an anionic collision (F^- and He) will have a negative charge on Fluorine, which is mentioned in the input file in the `Charge` variable. The first position of the list will have -1 to show charge on Fluorine, 0 to show no charge on Helium, and again a -1 showing overall charge on the whole molecule. Similarly, the multiplicity of the two fragments and whole system is a singlet, which is denoted by [1,1,1] in the list `Multiplicity`.

```
#--------------------------------------------------------------------------#
# Enter parameter for first atom -------------------------------------------#
RR1_atoms        = ['F']         # Enter atom 1
# Enter parameter for second (collider) atom -------------------------------#
RR2_atoms        = ['He']        # Enter atom 2
#--------------------------------------------------------------------------#
# Enter charge & multiplicity (Do NOT change order) ------------------------#
Charge           = [-1,  0, -1]  # charge       of [atom 1, atom 2, whole system]
Multiplicity     = [ 1,  1,  1]  # multiplicity of [atom 1, atom 2, whole system]
#--------------------------------------------------------------------------#
```

Similarly, cationic collision (Li^+-He) will have a positive charge on Lithium, which is mentioned in the input file in the `Charge` variable.

```
#--------------------------------------------------------------------------#
# Enter parameter for first atom -------------------------------------------#
RR1_atoms        = ['Li']      # Enter atom 1
# Enter parameter for second (collider) atom -------------------------------#
```

```
RR2_atoms        = ['He']      # Enter atom 2
# Enter charge & multiplicity (Do NOT change order) ---------------------------#
Charge           = [1, 0, 1]   # charge       of [atom 1, atom 2, whole system]
Multiplicity     = [1, 1, 1]   # multiplicity of [atom 1, atom 2, whole system]
#-------------------------------------------------------------------------------#
```

The UHF case has a carbon atom with two unpaired electrons, making its charge 0 but multiplicity of 3. Apart from the usual changes, the `psi4_reference` has to be changed from `'rhf'` to `'uhf'` or `'rohf'`.

```
#-------------------------------------------------------------------------------#
# Enter parameter for first atom ----------------------------------------------#
RR1_atoms        = ['C']      # Enter atom 1
# Enter parameter for second (collider) atom ----------------------------------#
RR2_atoms        = ['He']     # Enter atom 2
# Enter charge & multiplicity (Do NOT change order) ---------------------------#
Charge           = [0, 0, 0]  # charge       of [atom 1, atom 2, whole system]
Multiplicity     = [3, 1, 3]  # multiplicity of [atom 1, atom 2, whole system]
#-------------------------------------------------------------------------------#
psi4_reference     = 'uhf'    # Reference WF: rhf/uhf/rohf
#-------------------------------------------------------------------------------#
```

The fragment charge and multiplicity allow for BSSE (CP) corrected PES calculations. To demonstrate the same, an example of Ar-He collision is considered, where charges are 0 and multiplicities are singlet for all fragments. Instead of using `None` (no quotes) for `psi4_bsse`, the keyword changes to `'cp'` (use quotes as the input is a string, while None is an inbuilt data type in Python) to indicate counterpoise correction.

```
#-------------------------------------------------------------------------------#
# Enter parameter for first atom ----------------------------------------------#
RR1_atoms        = ['Ar']      # Enter atom 1
# Enter parameter for second (collider) atom ----------------------------------#
RR2_atoms        = ['He']      # Enter atom 2
# Enter charge & multiplicity (Do NOT change order) ---------------------------#
Charge           = [0, 0, 0]  # charge of [atom 1, atom 2, whole system]
Multiplicity     = [1, 1, 1]  # multiplicity of [atom 1, atom 2, whole system]
#-------------------------------------------------------------------------------#
psi4_bsse          = 'cp'               # counterpoise : 'cp'/None
#-------------------------------------------------------------------------------#
```

For CBS extrapolation, we use the same example as above. However, we will use `None` in `psi4_bsse` and change the basis set to `aug-cc-pv[dt]z` in `psi4_method_basis` to indicate CBS extrapolation:

```
#-------------------------------------------------------------------------------#
psi4_method_basis  = 'ccsd/aug-cc-pv[dt]z'      # Method/Basis Set
psi4_bsse          = None                       # counterpoise : 'cp'/None
#-------------------------------------------------------------------------------#
```

The above input does two-point extrapolation of CCSD correlation energies using aug-cc-pvdz and aug-cc-pvtz while using the reference energy of the largest basic set, i.e., aug-cc-pvtz. Users can also use in-built extrapolation schemes called `'sherrill_gold_standard'` and `'allen_focal_point'` directly in the `psi4_method_basis` variable. The details of the same are provided in in Psi4 manual and Section 2.3.3.

The project name for each can be kept as `anion, cation, neutral, cbs`, and `bsse`. The output folder for one of the calculations (anion-He collision: project name `anion`) is shown in Fig. 4.3. While the Psi4 results are stored in the `psi4_PES_data` folder, the final output PES in cm^{-1} along with the coordinate file is present outside in the project folder `Projects/$Proj_name/`. The `input_files` folder is empty since no external PES is generated.

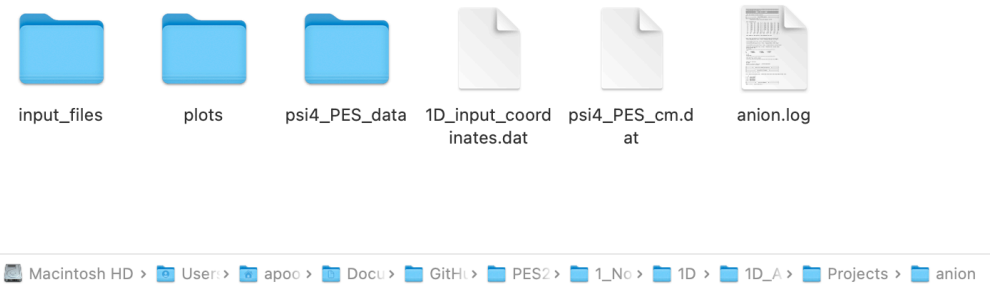


Figure 4.3. Output for 1D_Anion run.

The `1D_input_coordinates.dat` file contains the input coordinates of the PES without the energies, while the `psi4_PES_cm.dat` file contains the PES in cm^{-1}. The log file (`anion.log`) contains the output log for the PES2MP run and has the same name as the project.

A preview of the output log (`anion.log`) file for generating and plotting F^--He collisional PES using internal Psi4 calculations is attached below. The log file is appended if multiple calculations are performed using the same project name.

```
########################################################################
--- Date 2025-04-11 14:55:45 ---
########################################################################
#####################    PES2MP Initiated    ###########################
########################################################################

 /data and /project folders created.

#----------------------------------------------------------------------#
#-------------------#    Using PESGen module    #-------------------#
#----------------------------------------------------------------------#

Identifying collision type !!!:
   atom-atom collision: Using 1D template for PES

 The R coordinates are:
[  1.     1.1    1.2    1.3    1.4    1.5    1.6    1.7    1.8    1.9
   2.     2.05   2.1    2.15   2.2    2.25   2.3    2.35   2.4    2.45
   2.5    2.55   2.6    2.65   2.7    2.75   2.8    2.85   2.9    2.95
   3.     3.05   3.1    3.15   3.2    3.25   3.3    3.35   3.4    3.45
   3.5    3.55   3.6    3.65   3.7    3.75   3.8    3.85   3.9    3.95
   4.     4.05   4.1    4.15   4.2    4.25   4.3    4.35   4.4    4.45
   4.5    4.55   4.6    4.65   4.7    4.75   4.8    4.85   4.9    4.95
   5.     5.05   5.1    5.15   5.2    5.25   5.3    5.35   5.4    5.45
   5.5    5.55   5.6    5.65   5.7    5.75   5.8    5.85   5.9    5.95
   6.     6.2    6.4    6.6    6.8    7.     7.2    7.4    7.6    7.8
   8.     8.2    8.4    8.6    8.8    9.     9.5    10.    10.5   11.
   11.5   12.    12.5   13.    13.5   14.    14.5   15.    16.    17.
   18.    19.    20.    21.    22.    23.    24.    25.    50.    100.  ]

 Total number of data points in R coordinates are: 130

 Saving input coordinates in 'Projects/anion/1D_input_coordinates.dat'


 Create_GAUSSIAN_input_files      = False : Skipping Gaussian (CP) Input!
 Create_MOLPRO_input_files        = False : Skipping Molpro (CP) Input!
 Create_MOLPRO_CBS_input_files    = False : Skipping Molpro (CBS) Input!
 Create_MOLPRO_custom_input_files = False : Skipping Molpro (custom) Input!
 Create_Psi4_custom_input_files   = False : Skipping Psi4 (custom) Input!


```

```
 Generating Psi4 PES!
 Psi4 Input :
 -x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x
 -1 1
F          0.000000           0.000000        0.000000
He         0.000000           0.000000        {:.6f}

 no_com
 no_reorient
 -x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x

 TQDM LOOP: Psi4 1D PES created:
 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100% |

 Geometry at infinity:
 R = 200
 Energy at infinity (Hartree): -102.55223886182694
 Psi4 calculations finished!

#-------------------------------------------------------------------#
#--------------#    PES Files Created Successfully    #-------------#
#-------------------------------------------------------------------#

#-------------------------------------------------------------------#
#------------------#    Using PES Plot module    #------------------#
#-------------------------------------------------------------------#

Plotting PES from Psi4 output.

If you have unconverged points, remove them, and replot PES using PESPlot input file!

     R             E
0  1.0  77703.299699
1  1.1  56909.816403
2  1.2  40425.993984
3  1.3  28193.007101
4  1.4  19476.183654
#-------------------------------------------------------------------#
#--------------------#    PES Plots Updated    #--------------------#
#-------------------------------------------------------------------#

 NNGen = False : Skipping NN Model!

 MPExp = False : Skipping Multipole Expansion!

 FnFit = False : Skipping Analytical Fitting of Radial Coefficients!

#####################################################################
#####################    PES2MP Finished     ########################
#####################################################################
Run time: 0 hours, 0 minutes, 41.66 seconds
#####################################################################
```

The preview of `psi4_PES_cm.dat` file is also included below:

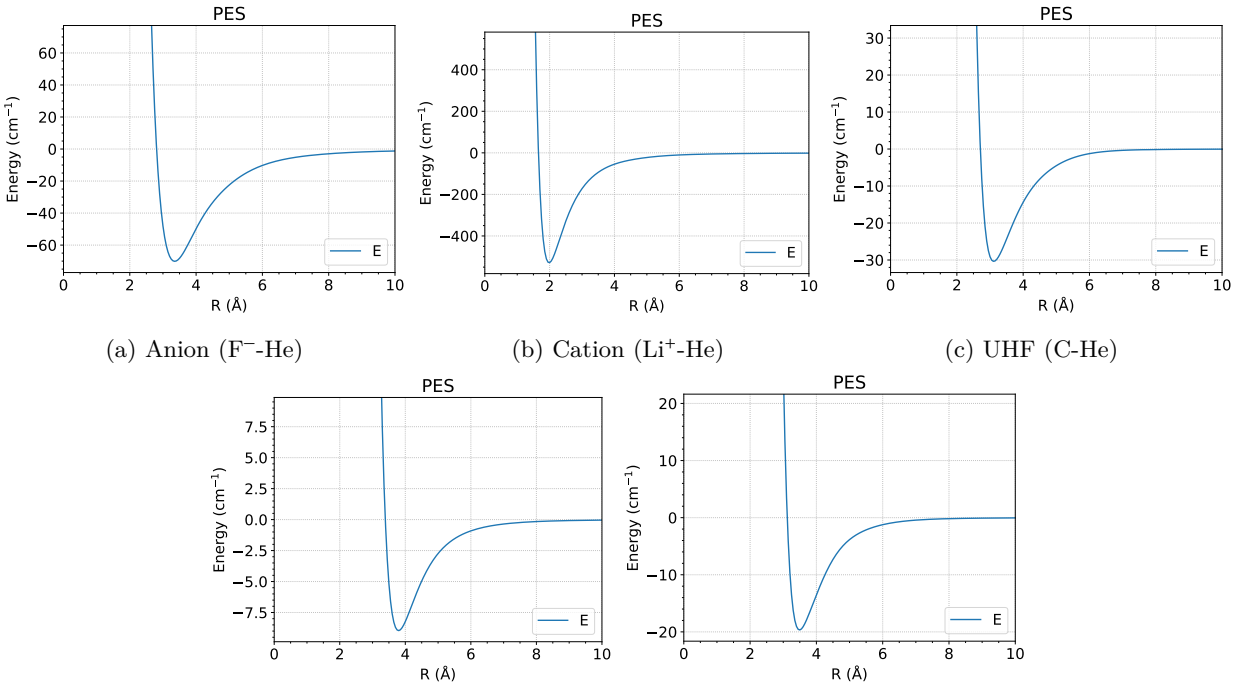
```
R     E
1.00  77703.299699
1.10  56909.816403
1.20  40425.993984
...
3.30  -69.779992
3.35  -70.125709
3.40  -69.920737
...
25.00 -0.030102
50.00 -0.00187
100.00  -0.00011
```

Inside the `psi4_PES_data/` folder, the `data/` folder contains the output files for the Psi4 run.

Figure 4.4. Output for 1D_Anion run inside `psi4_PES_data/` folder.

The `psi4_PES.dat` contains PES data in Hartree, and `psi4_failed_coordinates` contains the coordinates that have failed to converge (or encountered any other error).

The plot outputs for various 1D collisions are available in the `$Proj_name/plots/` folder. The range of the y axis is automatically set by the PES2MP program (10% more than the minima) in both (+ive and -ive axes). The generated plots for all five 1D runs are shown in Fig. 4.5.



(a) Anion (F^--He) (b) Cation (Li^+-He) (c) UHF (C-He)

(d) BSSE (Ar-He) (e) CBS (Ar-He)

Figure 4.5. PES plots generated by PES2MP after internal 1D calculations using Psi4. The sub-caption refers to various molecular collisions used in the example files.

The range of the x and y axes, along with the labels and header, can be changed (as discussed in the previous chapter) using `opt_plot.py` input files. The 2D/4D case will also follow the same folder pattern; however, there is an additional PES file in matrix format `psi4_PES_cm_matrix.dat`. For the 4D case, the polar plots contain only a specific combination of $\theta(s)/\phi$; therefore, there are multiple plots, and the data files for the same are stored in `plots_data_2D` (See Fig. 4.6).

2D/4D case: In the 2D case, the PES is generated internally, and since no data point is missing in the selected range, a direct multipole expansion is carried out, and the same is then fitted into a series of Slater functions. We shall discuss the 2D/4D collision cases simultaneously.

The PES calculation is carried out for 2D/4D PES using the input files:

```
#------------------------------------------------------------------------------#
# Enter parameter for RR1 : Enter experimental/theoretical bond length(s) -----#
RR1_atoms        = ['C','C'] # Enter rigid rotor (RR1) atoms from one end
RR1_bond_len     = [1.2425]  # Enter bond lengths from the same end (see manual)
# Enter parameter for second (collider) atom ----------------------------------#
RR2_atoms        = ['He']    # Enter collider atom
#------------------------------------------------------------------------------#
# Enter charge & multiplicity (Do NOT change order) --------------------------#
Charge           = [0, 0, 0] # charge       of [atom 1, atom 2, whole system]
Multiplicity     = [1, 1, 1] # multiplicity of [atom 1, atom 2, whole system]
#------------------------------------------------------------------------------#
```

Listing 4.1. 2D PES inout file

```
#------------------------------------------------------------------------------#
# Enter parameter for RR1 : Enter experimental/theoretical bond length(s)------#
RR1_atoms        = ['C','C'] # Enter rigid rotor (RR1) atoms from one end
RR1_bond_len     = [1.2425]  # Enter bond lengths from the same end (see manual)
# Enter parameter for second (collider) atom ----------------------------------#
RR2_atoms        = ['H','H'] # Enter atom(s) for RR2
RR2_bond_len     = [0.7667]  # Enter bond lengths from the same end for RR2
#------------------------------------------------------------------------------#
# Enter charge & multiplicity (Do NOT change order) --------------------------#
Charge           = [0, 0, 0] # charge       of [atom 1, atom 2, whole system]
Multiplicity     = [1, 1, 1] # multiplicity of [atom 1, atom 2, whole system]
#------------------------------------------------------------------------------#
```

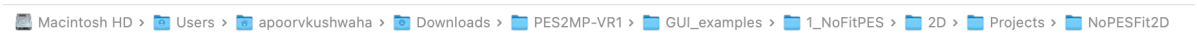
Listing 4.2. 4D PES inout file

where the `RR1_atoms` parameter contains two Carbon atoms and `RR1_bond_len` contains the experimental bond between C_2 molecule. In the 4D PES, the `RR2_atoms` contains two Hydrogen atoms, and the `RR2_bond_len` contains the vibrationally averaged bond length (r_0) for the ground state of H_2 molecule. The charge and multiplicities are 0 and 1 for all fragments, respectively.

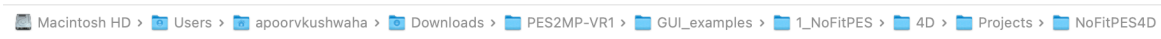
To keep the test calculations manageable, simple HF-D4/VDZ calculations have been carried out without BSSE correction. The D4 correction is made available via `dftd4` package and is installed alongside Psi4 in both `pes2mp` and `pes2mp_q` environments.

```
#------------------------------------------------------------------------------#
psi4_method_basis  = 'hf-d4/cc-pvdz'    # Method/Basis Set
psi4_Frozen_Core   = True               # frozen core : True/False
psi4_bsse          = None               # counterpoise : 'cp' | None for cbs/sgs
#------------------------------------------------------------------------------#
```

The overall results of the 2D/4D (internal Psi4) PES run are shown in Fig. 4.6 and are explained below:



(a) 2D Output



(b) 4D Output

Figure 4.6. Output folders of 2D and 4D collisional plots and radial terms.

There are three extra files in the main folder compared to the 1D example, i.e., `PES_psi4_cm_matrix.dat`, `MOLSCAT_POT.txt`, and `RAW_POT_custom.txt`. The `PES_psi4_cm.dat` contains PES in the usual dataframe R, θ, E format, while the `PES_psi4_cm_matrix.dat` file contains PES in matrix format for plotting purposes, and is shown below:

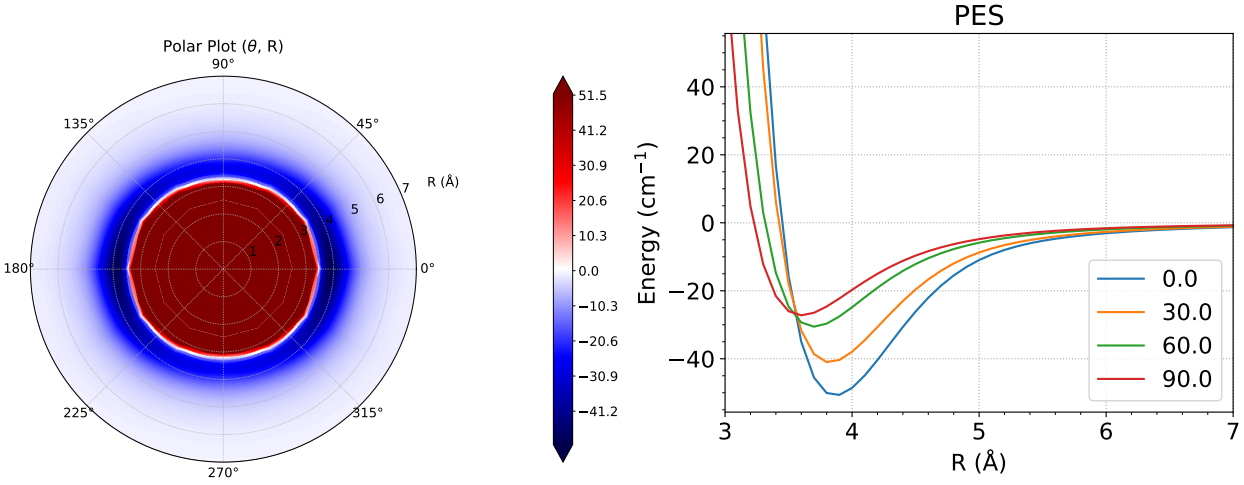
```
R            0.0         30.0         60.0         90.0
2.5     2808.663615 2837.451887 1444.357009  933.990799
2.6     2056.121904 1918.967779  970.168797  617.064710
... ... ...
3.6      -34.987927  -31.728689  -29.318138  -27.220558
3.7      -45.523756  -38.630803  -30.554998  -26.475278
3.8      -50.077642  -40.964383  -29.663996  -24.630241
3.9      -50.630053  -40.354504  -27.550546  -22.257532
... ... ...
50.0      -0.019913   -0.019916   -0.019929   -0.019932
100.0     -0.0         -4e-06      -2e-06       0.0
```

The other two files contain the coefficients for Slater functions used to fit the final radial terms V_Λ. The same will be discussed later in this subsection. The preview of the MOLSCAT readable potential file (`MOLSCAT_POT.txt`) is provided below:

```
LAMBDA = 0,2,4,6,8,10,12,
NTERM  = 3,3,3,3,3,3,3,
NPOWER = 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
A =
2.6178810348352877,2.6051153166382903,0.3504065530658476,
2.890479153353675,2.017672656247661,1.4761943532055537,
...

E =
106630848.41732779,106630848.41732779,-102143095.71321636,
106630848.41732779,-102143095.71321636,-10.60521049029151,
...
```

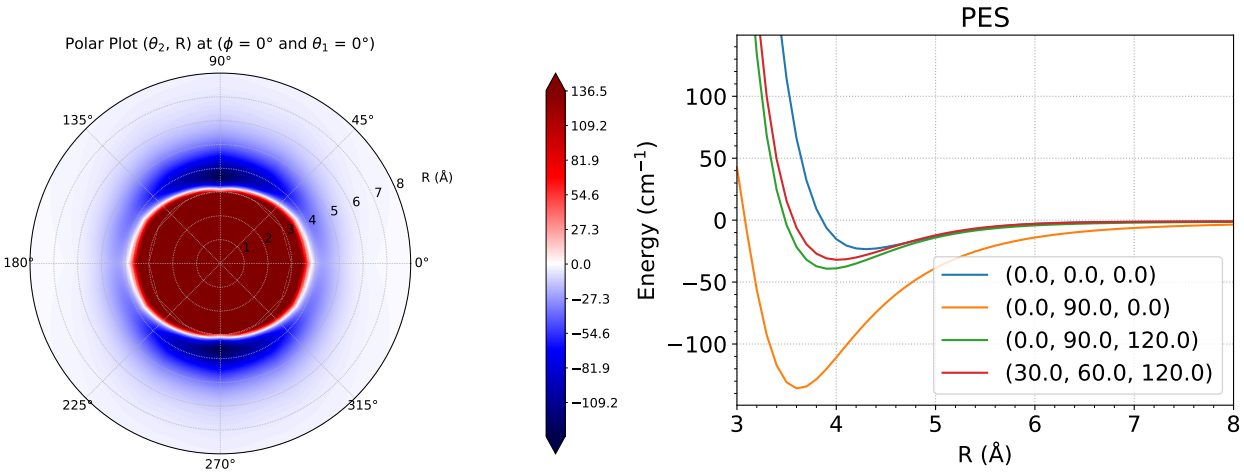
The `input_files` folder, as explained earlier, is empty since no external PES files are generated. The `plots` folder has *R vs E* and polar PES plot (single plot for 2D and multiple plots for 4D PES based on input coordinates). Since there are multiple polar plots with two angular terms frozen, their data is stored in `plots_data_2D/` and the files have the same name as the individual polar plots. The `MP_files` have radial terms and other outputs related to the multipole expansion of PES.

The results of rough PES (`'hf-d4/cc-pvdz'`) plots generated by PES2MP are shown in Fig. 4.7. The contour and labels for the same can be adjusted (if needed) using `opt_plot.py` input file.



(a) 2D: Polar Plot

(b) 2D: *R vs E*



(c) 4D: Polar Plot

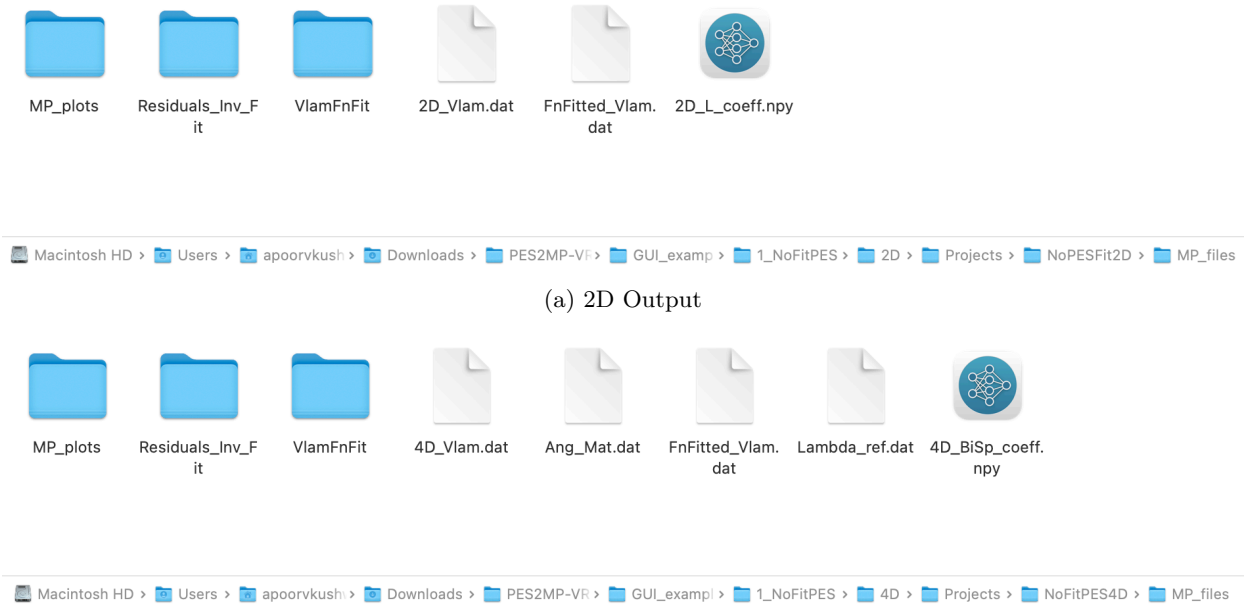
(d) 4D: *R vs E*

Figure 4.7. Output plots for 2D and 4D collisional PES. The 2D PES polar is shown in subplot (a) while the individual curves are shown in subplot (b). The minimum of the PES is at $\theta = 0°$. The 4D collision can have multiple polar plots; therefore, PES at global minimum ($\phi = 0°$, $\theta_2 = 90°$, $\theta_1 = 0°$) is shown for (θ_2, R) plot at fixed $\phi = 0°$, $\theta_1 = 0°$ in the subplot (c). The individual curves at fixed angular terms are shown in subplot (d).

The same PES is expanded into radial terms V_λ and $V_{\lambda_1,\lambda_2,\lambda}$ for 2D and 4D PES using `3_MPExp.py` files. The `2_FnFit.py` input file (that fits the PES into an analytical function) is not used for this example. The same is discussed in the next case: `2_FnFitPES` in the upcoming Section 4.3.

For the present examples, the combination of method/basis and the range of Jacobi coordinates is chosen such that all data points converge successfully in the final PES. Therefore, for demonstration purpose, we

shall directly fit this PES into radial terms (since no data point is missing). The obtained radial terms are then fitted into an analytical expression (MOLSCAT-readable) using `4_FnFit_Vlam.py`. Finally, the PES is regenerated from this function fitted V_Λ terms (inverse MP expansion) to assess the quality of analytical fitting, using the `opt_residual4D.py`. The files generated by these three input files are shown in Fig. 4.8.



(a) 2D Output

(b) 4D Output

Figure 4.8. Output files and folders of 2D and 4D multipole expansion. The same are located inside `$Proj_name/MP_Files/`.

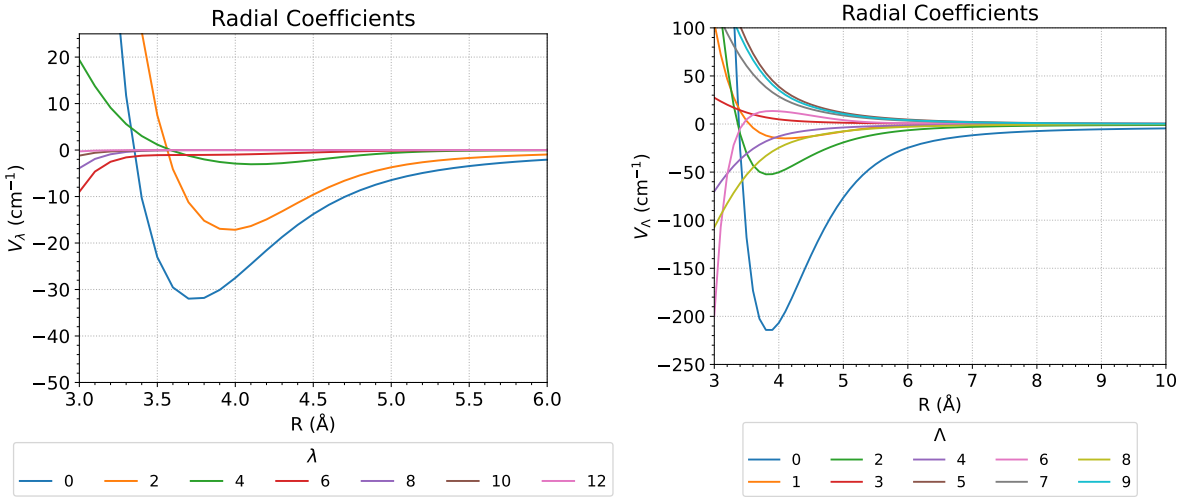
The radial terms obtained from PES are saved as `2D_Vlam.dat` and `4D_Vlam.dat` for 2D and 4D PES, respectively. The Legendre coefficients for 2D PES are saved in `2D_L_coeff.npy` (NumPy) file, while the SH coefficients for 4D PES is stored as `4D_BiSp_coeff.npy` (BiSpherical Harmonics coefficients). Apart from these, two more files are present for the 4D collision: where `Ang_Mat.dat` file stores the combination of angular terms in the PES, while `Lambda_ref.dat` stores the combination of radial terms as shown below:

```
# The radial terms in the Lambda_ref.dat file

Lambda   lambda_1  lambda_2  lambda
0        0         0         0
1        0         2         2
2        2         0         2
3        2         2         0
4        2         2         2
5        2         2         4
6        4         0         4
7        4         2         2
8        4         2         4
9        4         2         6
```

When the V_Λ terms are fitted into Slater functions (using `4_FnFit_Vlam.py`), the fitted radial terms are saved as `FnFitted_Vlam.dat` and the individual plots (showing fitting error) are saved inside the folder `VlamFnFit`. The same input file also generates potential files for MOLSCAT `&potl` block as previously shown in Fig. 4.6. The residual plot generated by `opt_residual4D.py` is stored in `Residuals_Inv_Fit`.

The contents of the folders are discussed below in detail.

The plot for radial terms present inside `/MP_files/MP_plots/` are shown in Fig. 4.9 for 2D and 4D PES.



(a) 2D MP Expansion (b) 4D MP Expansion

Figure 4.9. Radial terms obtained after 2D and 4D multipole expansion stored inside *MP_plots/*.The 2D radial terms are numbered accordingly, while the 4D radial terms are numbered from 0 to N. Users must refer to the `Lambda_ref.dat` file to identify the combination of $\lambda_1, \lambda_2, \& \lambda$ corresponding to the value of Λ.

The radial terms for 2D collision show the value of λ. However, for 4D collisions, the plot Fig. 4.9 (b) uses Λ which refers to $\lambda_1, \lambda_2, \lambda$ for 4D collision. The radial terms corresponding to Λ are provided in the `Lambda_ref.dat` file.

These radial terms can have errors when fitted into Slater functions. The `VlamFnFit` folder contains plots for individually fitted radial terms to inspect if the desired fitting is obtained. If the fitting is bad, users must include more Slater terms or find better coefficients that fit the data. An example of the same is shown in Fig. 4.10 where only three Slater functions are used to fit the data. The three subplots in the figure show a scaled, zoomed-out, and zoomed-in plot. The overall fit from the scaled plot is found to be good, while the zoomed-in plot shows a slight deviation that can be improved with more functions or a better initial guess:

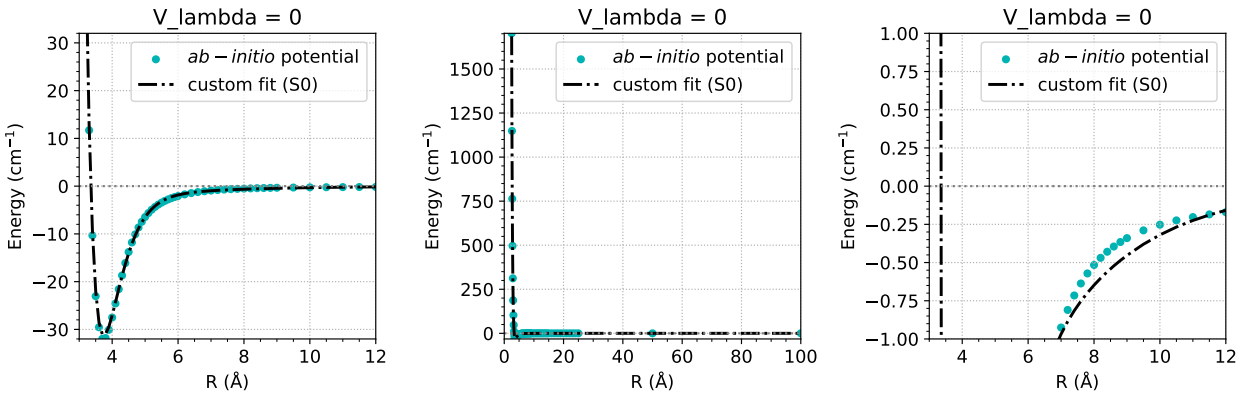


Figure 4.10. Fitting radial terms obtained after multipole expansion into an analytical expression.

The range of energy (y-axis) for the zoomed-in plot can be changed using the `def plot_Vlam` function in the `pes2mp_driver.py` file. The default for the same is -1 to +1 cm^{-1}.

The mathematical expression that is used to fit the 2D radial terms, along with upper and lower bound constraints and an initial guess, is shown below:

```
# Function used for fitting 2D collision radial terms
def fnfit_custom(x, a1,a2,a3,a4,a5,a6):
    import numpy as np
    return  a1*np.exp(-a4*x) + a2*np.exp(-a5*x) + a3*np.exp(-a6*x)

initial_val  =  [1e8, -1e7, 1e8, 4, 2, 1]                  # Enter initial guess
#----------------------- Optional Constraints --------------------------------#
lower_bounds = [-1e12]*3 +  [0]*3                     # Lower bound
upper_bounds = [ 1e12]*3 + [10]*3                     # Upper bound
#-----------------------------------------------------------------------------#
start_pos = [5,2,4,3,8,0,2]                           # fitting position
#-----------------------------------------------------------------------------#
```

The constraints require time and effort, as current numerical methods can fail to converge when both β and α coefficients are optimized simultaneously due to the inherent property of these functions (too many combinations with a lot of local minima and divergent solutions). The presence of very large values at small R can also result in a large fitting error. The error for the fit is printed, both on the screen and in the log file. There are two methods for trimming these large energies (+ive and -ive) at small R: (a) using the starting position, and (b) using a cutoff potential. In the 2D example, we shall make use of `start_pos`. Notice that only $\lambda = 5$ fits from the 0^{th} position, while for other V_λ terms, the starting position for fitting the curve is found using the trial and error method. The fitting output for the same is provided below:

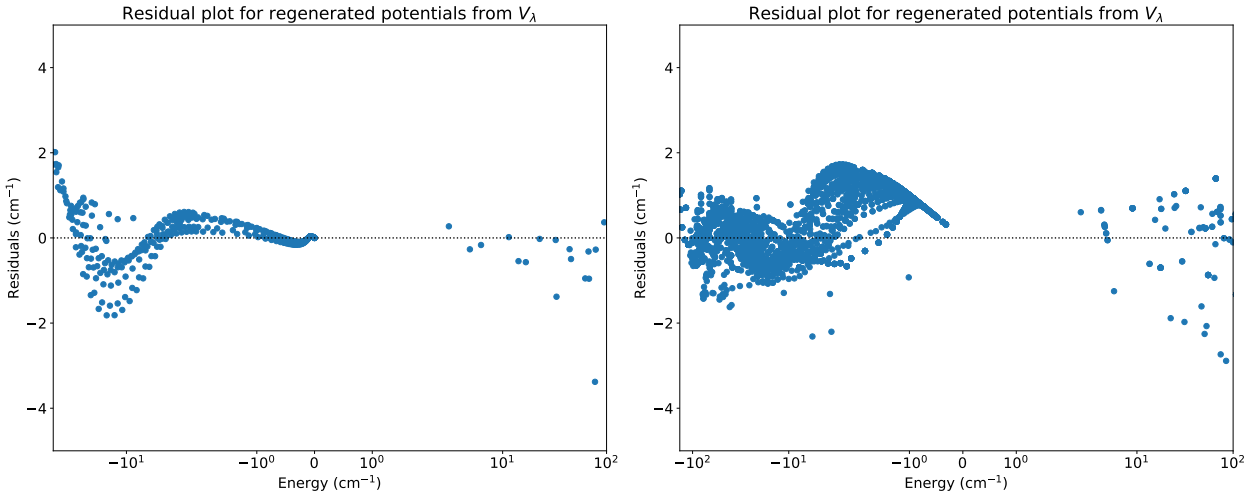
```
#-----------------------------------------------------------------#
Fitting Radial coefficients obtained after multipole expansion!

Minima Fit:0 | Start position: 5 | Value (cm-1) 187.08275754193951
RMSE 0 | 0.6556144177049593

Minima Fit:1 | Start position: 2 | Value (cm-1) 780.7690684344338
RMSE 1 | 0.4590711633767787

...
...

Minima Fit:6 | Start position: 2 | Value (cm-1) -4.753620586898968
RMSE 6 | 0.006930927090511125

Start position for fitting each radial term :
 start_pos = [5. 2. 4. 3. 8. 0. 2.]

 Mean RMS Error | 0.24120405491602195
#-------------------------------------------------------------------------#
```

In the 4D case, we use functions with fixed β values as shown below, along with the `cutoff` potential. These functions offer a much simpler optimization problem and would rarely fail to converge. Once an initial fit has been obtained, users can tweak the β values and `cutoff` for a better fit.

```
# Function used for fitting 4D collision radial terms
def fnfit_custom(x, a1,a2,a3,a4,a5,a6):
    import numpy as np
    return  a1*np.exp(-6*x) + a2*np.exp(-5*x) + a3*np.exp(-4*x) + a4*np.exp(-3*x) + \
            a5*np.exp(-2*x) + a6*np.exp(-1*x)

initial_val  =  [1e8, -1e7]*3                   # Enter initial guess
#-------------------------------------------------------------------------#
cutoff   = 300   # Energy cutoff in cm-1. (Not used: Using start pos)
           # --> For attractive potential, code automatically switches to -cutoff
#-------------------------------------------------------------------------#
```

While we can see the fitting error in V_Λ terms in the log file, we must also evaluate the error in the PES regenerated by these **fitted** radial terms (by comparing the same to *ab initio* PES).

Therefore, in the final step, the 2D/4D radial terms expressed as the Slater functions are reverse-fitted (inverse multipole expansion) to give back the PES using `opt_residuals.py` input file. The regenerated PES is stored inside `Residuals_Inv_Fit` folder. The folder also contains the residuals (both data and plot) comparing *ab initio* PES to regenerated PES. The residual plot is shown below in Fig. 4.11, which can be improved (as stated earlier) by using more Slater functions or a better initial guess.



(a) 2D collision (b) 4D collision

Figure 4.11. Residual plot showing error at various potential regions after the PES has been recreated from the analytically fitted radial terms for (a) 2D collision and (b) 4D collision.

In the next example `2_FnFitPES`, we shall see that fitting PES into Slater functions with fixed β values can be beneficial for not only interpolating/extrapolating missing PES data points, but the same functions can be used to fit the resulting radial terms. In that case, the inverse residual fitting error would be $\backsim 0$ (when compared with the Fitted PES). Before that, let us take a look at what happens when we try to generate more angles than the number of Λ terms.

4.2.2 Case Study: Inverse Multipole Expansion & Residuals

The radial terms (V_λ) obtained after multipole expansion of 2D PES must be less than or equal to the number of angles (θ). ***Why?***

Because N variables cannot be solved with $< N$ equations (linear algebra!).

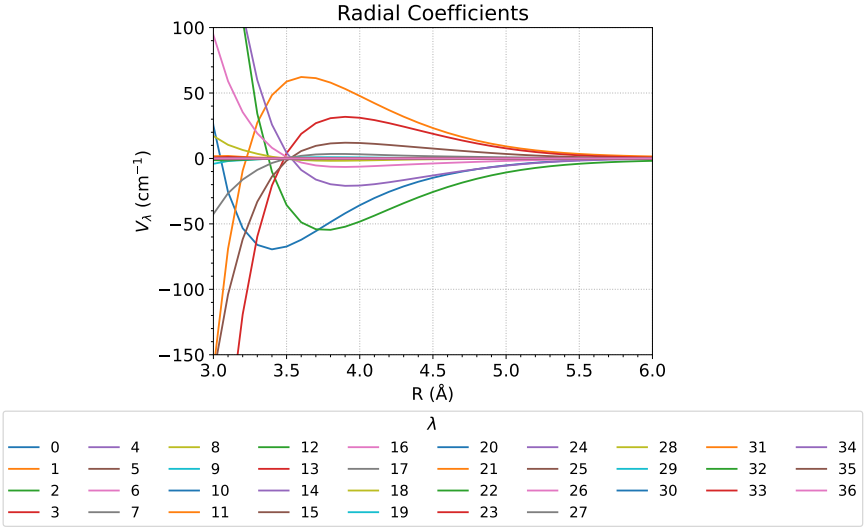
So, what happens when we solve for $< N$ variables with N equations, i.e.:

What if we solve for 19 λ terms when we have 37 θ?

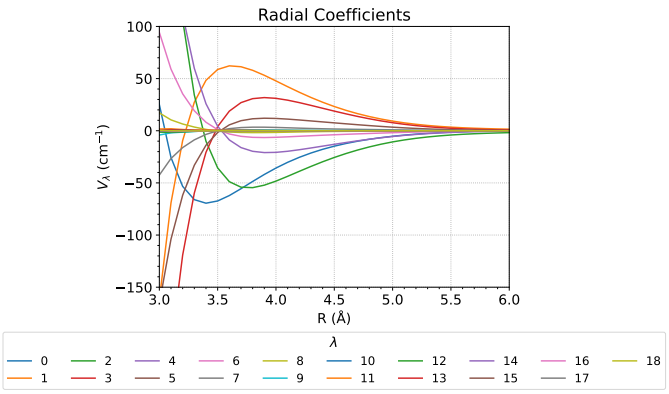
And what happens if we try to reverse fit these 19 V_λ terms back into 37 angles?

Here, we take HCO^+-He collision PES (as discussed in Section 3.4) with θ ranging from $0° - 180°$ with a step size of $5°$. So, a total of 37 angles are present, and the system is non-symmetric, so λ has both even and odd values. Since no PES data points are missing and the only objective is to validate reversibility of the multipole expansion, we shall not fit either the PES or V_λ.

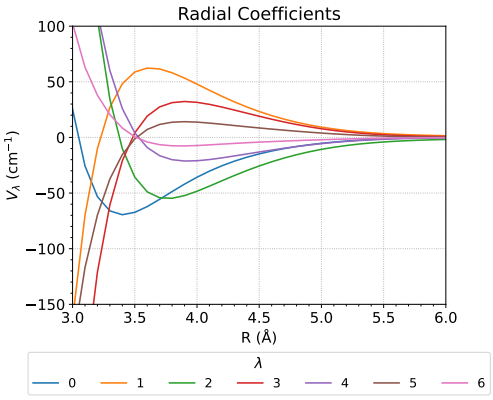
Fig. 4.12 shows the three plots for where expansion has been restricted to $\lambda = 37, 19,$ and 7. The plot shows that the current methodology (taking the pseudo-inverse of the matrix with Legendre coefficients) generates the correct radial terms when terminated to $\lambda \leq \text{number of } \theta$. The first seven radial terms in all three sub-figures are nearly identical. Quantitatively, the isotropic term of $\lambda_{max} = 19$ and 7 have average deviation $\backsim 0.002\%$ and $\backsim 0.134\%$ when compared to $\lambda_{max} = 37$.



(a) $\lambda = 37$



(b) $\lambda = 19$

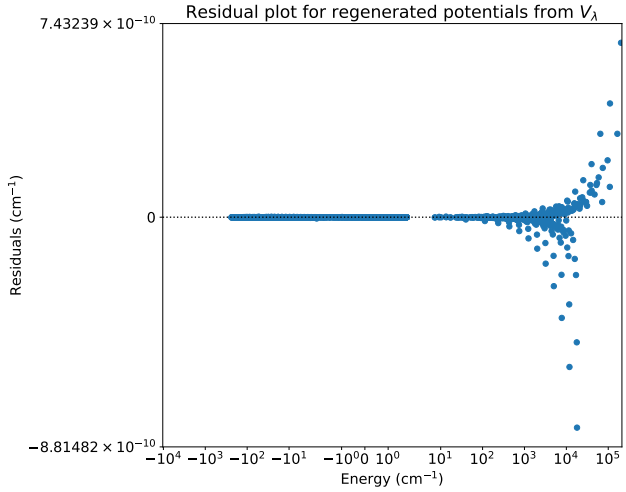


(c) $\lambda = 7$

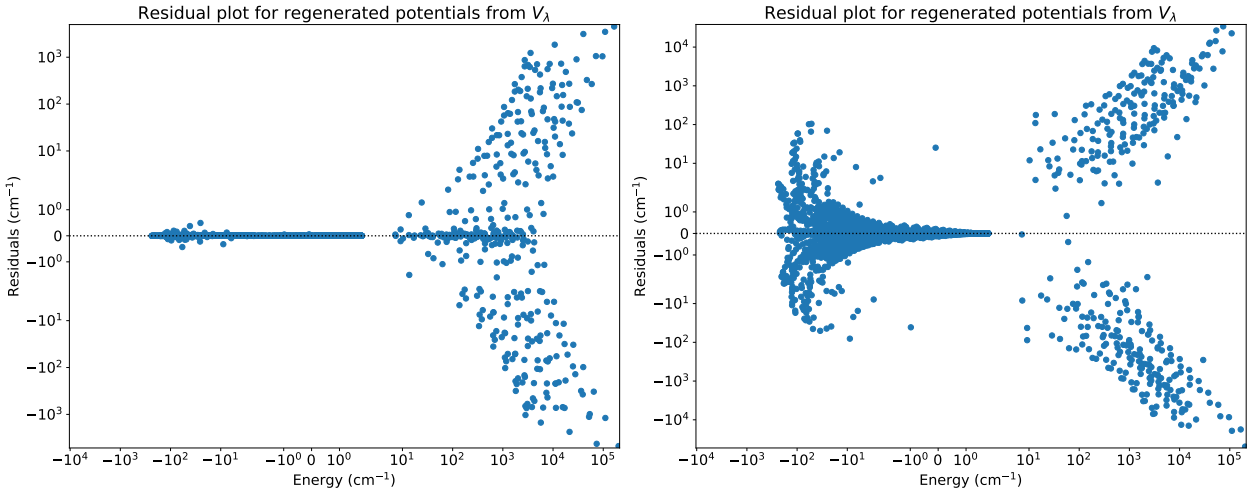
Figure 4.12. Radial terms obtained after multipole expansion of HCO^+-He collision PES.

Now we shall reverse fit these radial terms to give back 37 angles, i.e. θ ranging from $0° - 180°$ with a step size of $5°$. The residual plot for each is given in Fig. 4.13.

The results are what we expect from algebraic mathematics. Inverse fitting 37 λ terms to get 37 angles (at $5°$ interval) is a one-to-one mapping (has a unique solution) and the residual plot (a) shows the error in range $\pm 10^{-9}$, i.e., negligible. In sub-figure (b), 37 unknown variables (angles) are solved from 19 λ terms, which in theory should only give back 19 angles (at $10°$ interval). Therefore, we see that the solutions begin to diverge (mostly in the high energy region), while the residuals at the minima region still show error < 1 cm^{-1}. In the last example, 7 λ terms generate 37 angles and the results diverge both in the high energy and the minima region since 7 radial terms can only generate 7 angles at a $30°$ interval, which is 6 times less than $5°$. This shows that 7 radial terms are not enough to capture the full anisotropic behavior of the HCO^+-He PES in the minima region, but 19 terms are.



(a) $\lambda = 37$



(b) $\lambda = 19$ (c) $\lambda = 7$

Figure 4.13. Residual plot of regenerated PES, obtained after inverse multipole expansion of HCO^+-He V_λ terms, when compared to *ab initio* PES (full range: includes all high energy points).

Takeaway! Generate sufficient PES angles and do multipole expansion. For the 2D case, simply keep λ equal to the number of angles. Now check radial terms for convergence. If the first few radial terms converge to 0 (show no sign of minima or maxima), the higher order terms can be ignored/truncated. If the case where more angles are used to generate fewer radial terms, the results are nearly identical. However, regenerating PES from these fewer radial terms will show deviations. The fewer the number of radial terms, the higher the error in regenerating the PES.

In the 4D PES case, the value of λ depends on the value of λ_1 and λ_2, which are user input. The combination of radial terms (λ_1, λ_2) is usually (6,4),(12,6), etc., depending on the number of angles (which in turn depends on the anisotropy of PES). Therefore, we usually end up generating more angles than radial terms. For example, a PES for two symmetric rigid rotors with 196 angular terms ($N_{\theta_1} = 7, N_{\theta_2} = 7, N_\phi = 4$) can be used to generate first 92 radial terms ($\lambda_1 = 12$, $\lambda_2 = 6$). The inverse fit of this PES should show similar results as Fig. 4.13(b), where the minima region should show small error, but the higher energies will show greater deviation. Therefore, for 4D PES, users must monitor the fitting error in the PES/V_λ, and expect deviation in high energies during inverse fit.

4.3 2_FnFitPES: Fitting PES using Slater functions

Generating 1D, 2D, and 4D PES files internally using Psi4 and fitting them into a mathematical expression (series of Slater functions) to interpolate or extrapolate missing data points. This example also has results for MOLSCAT output with rate coefficients.

*The Slater functions ($\alpha e^{-\beta R}$) used for fitting 2D/4D PES and radial terms have the same β values:

1. 1D: Input files for handling (i) C-He collision, (ii) re-plotting PES with different ranges, texts, plot parameters, or file name extensions, and (iii) fitting the PES into five different mathematical expressions.
2. 2D: Example for generating (i) a rough and a high-level PES of C_2-He collision, (ii) fitting into a mathematical expression, (iii) expansion into radial terms, and (iv) using the same mathematical expression (also generating MOLSCAT's &POTL file).
3. 4D: Example for generating a (i) rough PES of C_2-H_2 collision, (ii) fitting into a mathematical expression, (iii) expansion into radial terms, and (iv) using the same mathematical expression to fit radial terms (also generating MOLSCAT's &POTL file).

Function fitting with 1D PES data.

In the 1D PES data, we shall see an example PES: C-He collision, calculated using HF-D4/3-21G. This PES is then fitted using five different functions as shown below:

```
# Fn1 (Under-Fits the PES)
def fnfit_custom(x, a1,a2):
    import numpy as np
    return  a1*np.exp(-3*x)+a2*np.exp(-2*x)

initial_val  = [1e4]*2                    # Enter initial guess
#-------------------------------------------------------------------------------#
# Fn2 (Over-Fits the PES)
def fnfit_custom(x, a1,a2,a3,a4,a5,a6,a7,a8):
    import numpy as np
    return  a1*np.exp(-5*x)+a2*np.exp(-3.75*x)+a3*np.exp(-3*x)+a4*np.exp(-2.75*x) + \
            a5*np.exp(-2*x)+a6*np.exp(-1.75*x)+a7*np.exp(-1*x)+a8*np.exp(-0.75*x)

initial_val  = [1e4]*8                    # Enter initial guess
#-------------------------------------------------------------------------------#
# Fn3 (Good Balanced Fit)
def fnfit_custom(x, a1,a2,a3,a4):
    import numpy as np
    return  a1*np.exp(-3*x)+a2*np.exp(-2.25*x)+a3*np.exp(-2*x)+a4*np.exp(-1.25*x)

initial_val  = [1e4]*4                    # Enter initial guess
#-------------------------------------------------------------------------------#
# Fn4 (Custom Exp-Log Function)
def fnfit_custom(x, a1,a2,a3):
    import numpy as np
    u = a2*x
    return  a1*np.exp(-u)*np.log(a3/u)

initial_val  = [1e6,1,1]                  # Enter initial guess
#-------------------------------------------------------------------------------#
# Fn5 (Custom ExpLog-Log Function)
def fnfit_custom(x, a1,a2,a3):
    import numpy as np
    u = a2*x*x
    Z = (-a3) * np.log(u)

    return a1 * Z * np.exp(Z)
initial_val  = [300, 0.1, 4]                  # Enter initial guess
```

In the first case, we shall keep the function 1 (Fn1) and comment rest of the functions (and constraints). For the second case, we will uncomment the second function and comment the rest, and so on. For functions 4 and 5 (Slater-log Functions), we will use constraints as shown below:

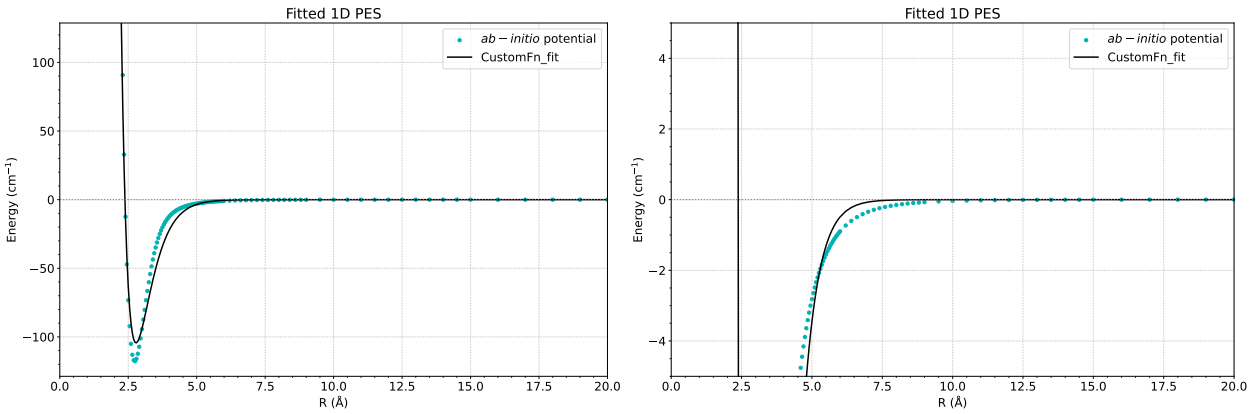
```
#------------------------ Optional Constraints -------------------------------#
# Fn(1,2,3) (Off:Commented) - Fn4/Fn5 (On: Uncomment)
#-----------------------------------------------------------------------------#
# Fn4 (Custom Exp-Log Function)
lower_bounds = [1,     0,   0]                  # Lower bound
upper_bounds = [1e12, 10,  10]                  # Upper bound
#-----------------------------------------------------------------------------#
# Fn5 (Custom ExpLog-Log Function)
lower_bounds = [0,     0, 0]                    # Lower bound
upper_bounds = [1000,  9, 9]                    # Upper bound
```

It is very important to understand how a function may behave with different coefficients. To build new functions or visualize the existing ones, visit Desmos (website to plot functions). For example, the $V(\text{cm}^{-1}) = a_1 e^{-a_2 R} + a_3 e^{-a_4 R}$ function (Fn1) has potential in cm^{-1}, therefore the coefficients a_1 and a_3 must have the same dimensionality/units (cm^{-1}). Since the exponential (and also logarithmic) functions are dimensionless (no units), the a_2 and a_4 functions will have inverse dimensionality to R, i.e., Å^{-1}.

Let us see the results for various fits. The cutoff for all functions is kept the same (100 cm^{-1}) to trim the high-energy potentials. The result for the first function fit (Eq. 4.1) is shown in Fig. 4.14, which fits the C-He collision data into two Slater functions with fixed β coefficients (3 and 2), and α coefficients that are optimized using the lmfit (SciPy) library. The function used for fitting the PES has the expression:

$$a_1 e^{-3x} + a_2 e^{-2x} \tag{4.1}$$

The residual plots: (a) full-range and (b) trimmed to cutoff; along with fitted PES: (c) full-range (includes high energy region), (d) overall fit (symmetric range w.r.t minima), and (e) zoomed-in (range provided in input file) plots, are saved in `$Proj_name/PESFnFit/Plot/` folder. The overall and zoomed-in plots are shown here to visualize the quality of the fit.



(a) Overall Fit (b) Zoomed-In

Figure 4.14. Fitting PES into Function Eq. 4.1

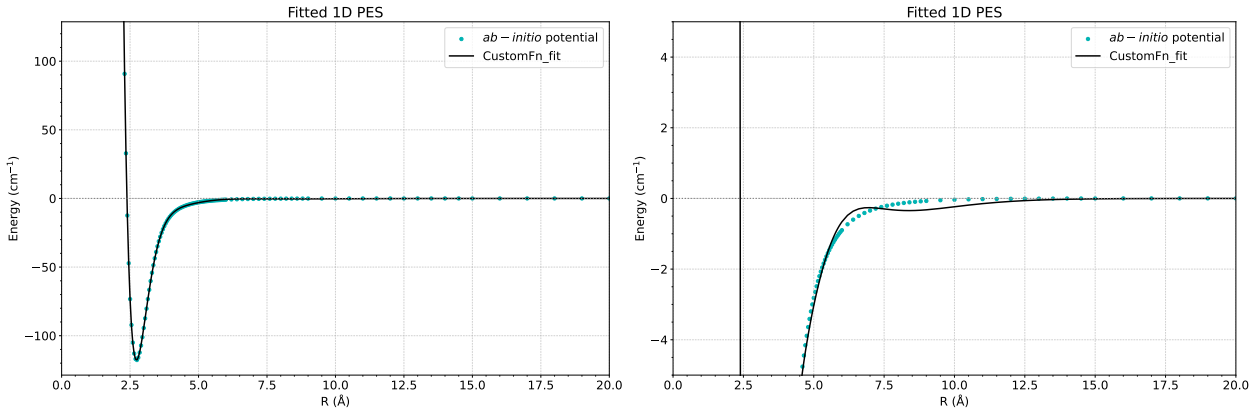
As expected, the simple equation under-fits the data and results in a poor overall fit. However, it has the correct shape of the PES, and with more functions, the error can be reduced.

∗ *Users must also try to modify the above function to* $a_1 e^{-a_2 R} + a_3 e^{-a_4 R}$ *with appropriate lower and upper bounds. This will allow the* β *coefficients to relax and provide a better fit.*

In the next function (Fn2), we use an expression with 8 Slater functions and fixed β coefficients. We do not use constraints (similar to the previous example) and use trial and error to find good β coefficients. Let's see if a better fit is obtained:

$$a_1 e^{-5x} + a_2 e^{-3.75x} + a_3 e^{-3x} + a_4 e^{-2.75x} + a_5 e^{-2x} + a_6 e^{-1.75x} + a_7 e^{-1x} + a_8 e^{-0.75x} \tag{4.2}$$

The result of this fit is shown in Fig. 4.15 with very good results when we see the overall fitting error. However, when zoomed in, this function shows deviation when it reaches the asymptotic region. This is the result of overfitting, where a large number of functions results in unwanted minima and maxima, which are not physically correct.

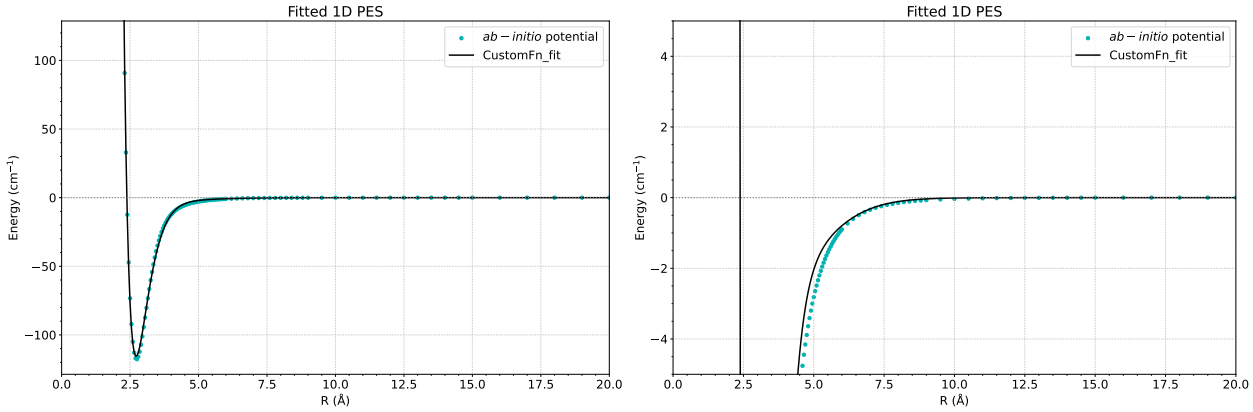


(a) Overall Fit (b) Zoomed-In

Figure 4.15. Fitting PES into Function Eq. 4.2

In Function 3, we have taken fewer (4) Slater functions, and the result does not under/overfit the data as seen above. So, blindly adding more Slaters into the function does not always yield good results. It takes patience and several trials to find a balanced function that stays physically correct and has spectroscopic accuracy (error $< 1\text{cm}^{-1}$).

$$a_1 e^{-3x} + a_2 e^{-2.25x} + a_3 e^{-2x} + a_4 e^{-1.25x} \tag{4.3}$$



(a) Overall Fit (b) Zoomed-In

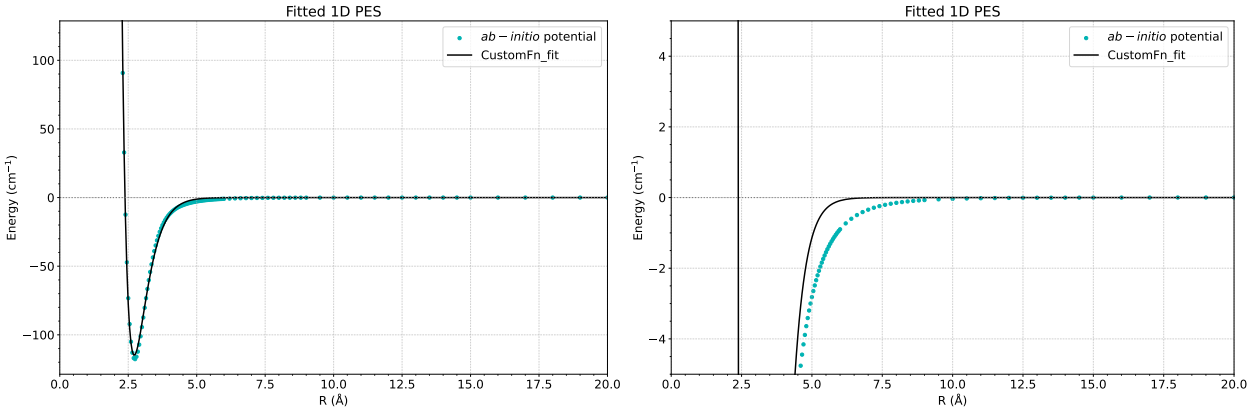
Figure 4.16. Fitting PES into Function Eq. 4.3

In the fourth example, a non-standard Slater-Log function is taken, i.e., $e^x \ln(x)$. Such functions have physically correct properties at extreme R regions (0 cm^{-1} at infinite R and ∞ at $R = 0$) with a tunable minima. Though such functions cannot be used in MOLSCAT or fit the PES any better than Slater functions, they are taken to show that the program (PES2MP) is not restricted to any specific function. Users can play around with different functions, which are a function of R, to benchmark or maybe find analytic solutions for any two-atom collisions.

The function in the expanded form is shown below:

$$a_1 e^{-a_2 x} \ln(a_3/(a_2 x)) \tag{4.4}$$

where we have declared $a_2 x = u$ and reduced the equation to $a_1 e^{-u} \ln(a_3/u)$. In this function, a_1 controls the depth of the function, while a_2 scales R and, along with a_3, controls the width and location of the well for the attractive region and repulsive region, respectively. The plots obtained for the above function fitting are shown in Fig. 4.17:



(a) Overall Fit (b) Zoomed-In

Figure 4.17. Fitting PES into Function Eq. 4.4

The Slater-log function shows an overall decent fit; however, in the zoomed-in plot, we see that the long-range behavior is not correct, which can be corrected by using more of such coupled (Slater-log) functions with a smaller decay coefficient.

The coefficients for the fit are saved in `pes_fn.txt` inside `$Proj_namr/PESFnFit/` folder. The same is shown below:

```
1D
1659166.096255528,2.77899834705676,6.6183447975715
```

where the coefficients a_1, a_2 and a_3 are optimized to $\approx 1659166, 2.778998$, and 6.618344, respectively. The terminal output (also saved in the log file) is:

```
Start position: 16 | Value (cm-1) 90.784263

No High Energy fitting!
No Long Range fitting!

 Using Custom Function for full range fitting!
Potential at initial R value:  89409.40773370655

{'a1': np.float64(1659166.096255528), 'a2': np.float64(2.77899834705676), 'a3': np.float64(6.6183447975715)}
```

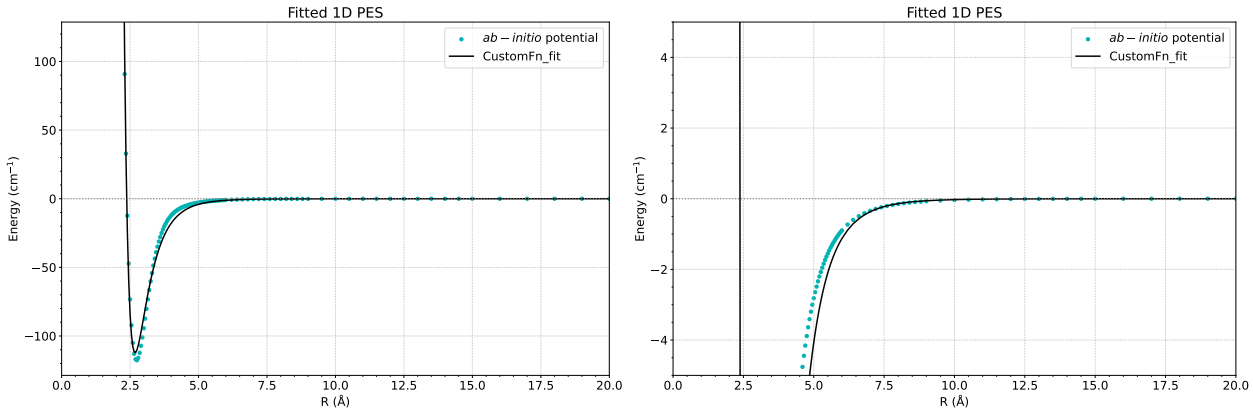
Some examples of Slater-log functions are also provided in 2.4.1. The users can also try to modify x, i.e., R, by modifying its power (e.g., x^2). For MOLSCAT applications, stick to simple Slater functions with α and β coefficients, as these functions are not inherently implemented in the same.

In the final function, the PES is fitted into a function of x^2:

$$-a_1 * a_3 * e^{-a_3 ln(a_2 x^2)} \ln(a_2 x^2)) \tag{4.5}$$

which can be simplified into $a_1 * (a_2 x^2)^{-a_3} * \ln\left(a_2 x^2\right)^{-a_3}$

In the input file, we have declared $a_2 x^2 = u$ (x scaling) and $Z = (-a_3) * \ln(u)$, which reduced the equation to $a_1 * Z * e^Z$. In this function, a_1 controls the depth of the function, while a_2 and a_3 control the width and location of the well for the attractive region and repulsive regions.



(a) Overall Fit (b) Zoomed-In

Figure 4.18. Fitting PES into Function Eq. 4.5

This function performs better than the previous function in the long-range region. However, similar to the previous example, this is provided to demonstrate the importance of finding a suitable analytical expression of PES. The same must not be used for fitting radial coefficients (in the 2D/4D case).

The last two functions only have positive coefficients (and therefore the constraints only have positive limits). The two functions also maintain the shape of the PES, and V never goes to $-\infty$ at $R \to 0$, unlike the first three (series of Slater) functions. Overall, such functions can be good candidates for fitting PES when modified to allow for better flexibility.

The output folder for the PES function fit (1D example) is shown below:

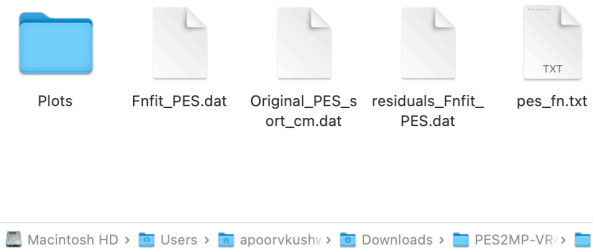


Figure 4.19. Contents of output folder (`/PESFnFit/`) for 1D PES function fit.

which contains PES data (both *ab initio* and fitted), residual data file, coefficients file, and a `Plots/` folder with plots of residuals and 1D PES (including those shown above).

Fitting 2D and 4D PES data using Slater functions.

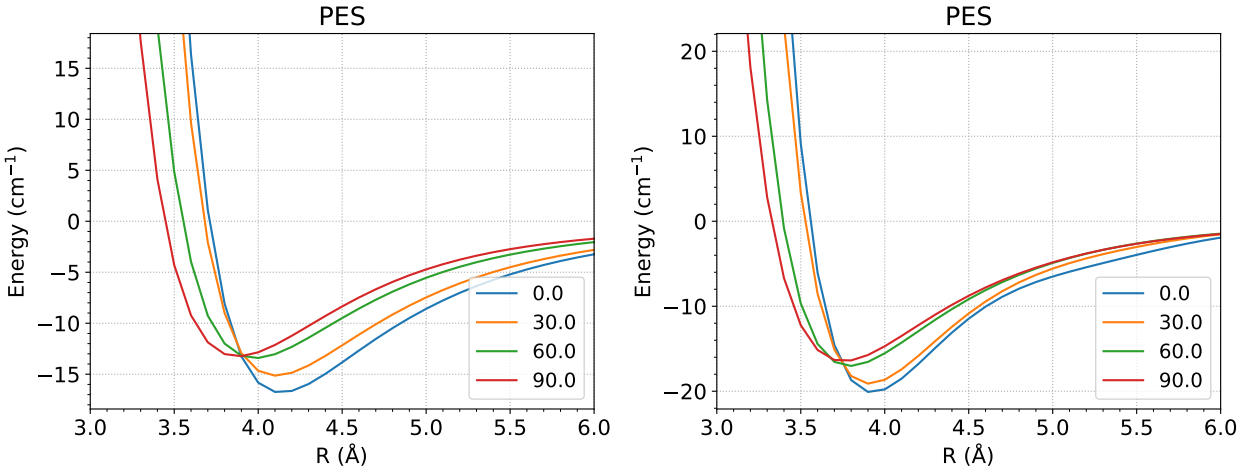
In the 2D/4D PES, we shall generate the *ab initio* PES internally using Psi4 and then fit the individual angles (*E vs R* data) using Slater functions with fixed β coefficients. After multipole expansion, the PES is fitted back into the same function. ***Why?*** When we calculate the residuals of the regenerated PES w.r.t. the function fitted PES, the residuals are negligible ($\backsim 10^{-10}$). The idea is to use functions that can interpolate/extrapolate any missing data point, while also minimizing the fitting errors before final MOLSCAT calculations.

2D PES

In the 2D PES example, we shall generate the two **ab initio** PES for C_2-He collision, one using HF-D4/aug-cc-pVQZ (CP), which will be referred to as ***HFD***, and another inbuilt CBS method called sherrill gold standard, which will be referred to as ***SGS***.

In the Introduction section, we saw that PES generated using dispersion corrected Hartree Fock methods performed better than DFT methods when we want the shape of the PES to be as precise as possible. Therefore, in this example, we shall not only calculate the radial terms but also calculate cross-sections and rate coefficients for a comparative study. The input files used for MOLSCAT calculations are available in `/input_files/aux_scripts`.

The PESs for the two methods are shown in Fig. 4.20. The ab initio PES is generated at 30° intervals from 0°−90°, since the system is symmetric and neutral, and the anisotropy is small. These properties make C_2 an ideal case for benchmarking purposes. However, it is recommended to take these results for preliminary evaluation only.



(a) HFD (b) SGS

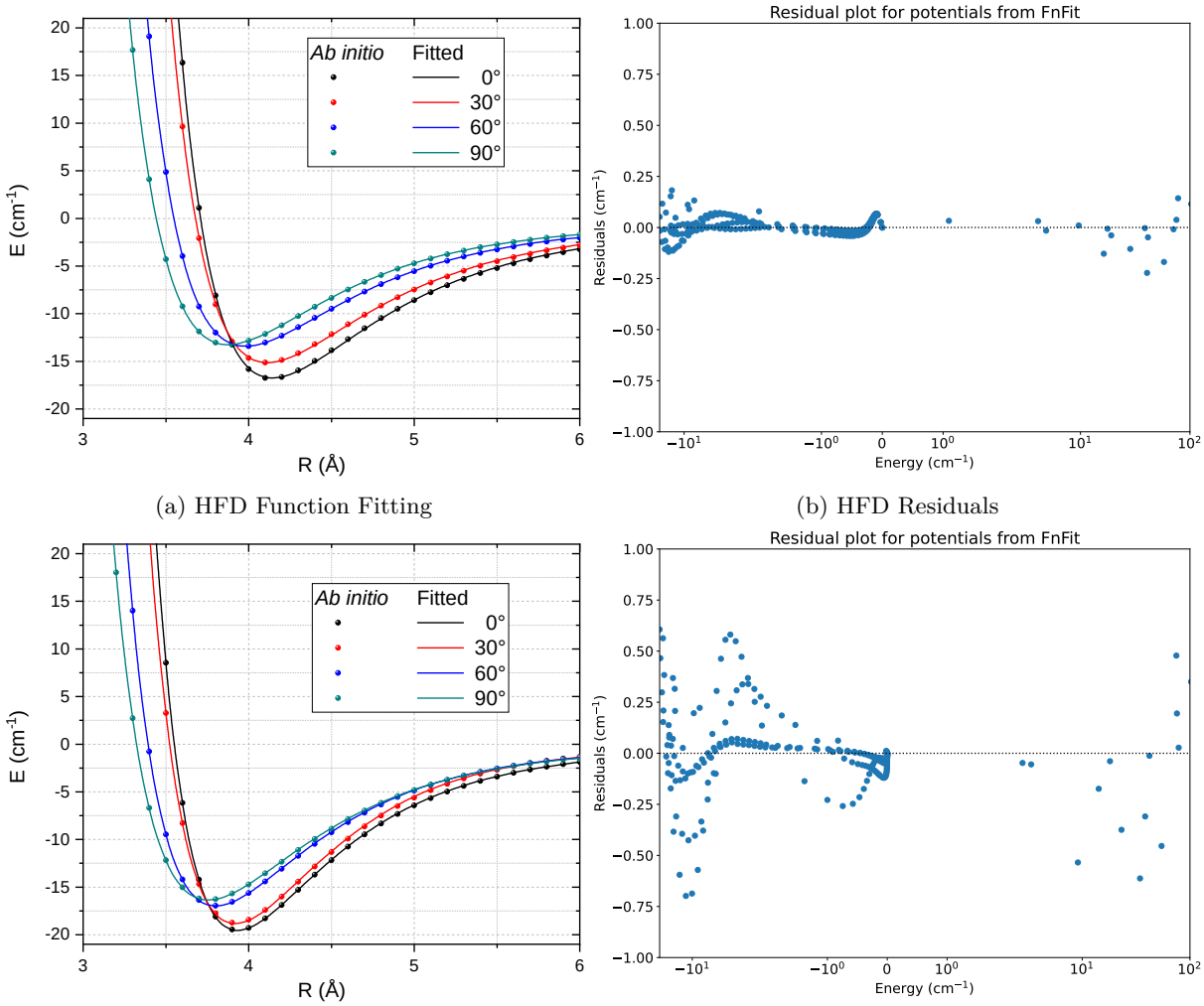
Figure 4.20. *Ab initio* PES for C_2-He collision (a) HF-D4/aug-cc-pVQZ with counterpoise correction (***HFD***), and (b) Sherrill gold standard, (***SGS***).

While the shapes of the curves in the two methods are nearly the same, the major difference can be observed in the well-depth of the minima. Now these curves are fitted into the given Slater function:

$$a_1 e^{-4x} + a_2 e^{-3x} + a_3 e^{-2x} + a_4 e^{-1x} + a_5 e^{-0.2x}. \tag{4.6}$$

The small β coefficient in the final Slater $a_5 e^{-0.2x}$ is added to allow the function to have a slow decaying tail, which can capture the long-range (R^{-6}) behavior of the PES. Being a neutral system, the long-range corrections are small, and therefore, the HELR (High-Energy/Long-Range) fit is kept False.

Fig. 4.21 shows the fitted potentials and corresponding residual plots. The figures (a,c) have been plotted using Origin.



(a) HFD Function Fitting

(b) HFD Residuals

(c) SGS Function Fitting

(d) SGS Residuals

Figure 4.21. *Ab initio* and function fitted PES for C_2-He collision obtained using (a) HF-D4/aug-cc-pVQZ with counterpoise correction (***HFD***), and (c) sherrill gold standard, (***SGS***) along with the residuals (b), and (d) for the respective fitting. † *Subplots (a,c) have been plotted using Origin.*

The fitted PES maintains the required shape at the high-energy region and the long-range regions, and the residual plots show spectroscopic accuracy in the fit. The fitted PES coefficients are saved in `PESFnFit/PES_fn.txt` and is shown below:

```
2D
0
83306731.13576201,1942613.998123643,-61341.30952697418,-834.4042920590489,-2.579572848619542
30
58481834.30238403,2060635.05343777,-60838.80197625618,-671.0945766358003,-2.517358547879758
60
21786406.21316013,1856307.205187443,-55635.61915819067,-415.1379427673821,-2.258742951967583
90
10422666.9410387,1605374.140161324,-50898.56017106478,-321.6136495860255,-2.069535591558693
```

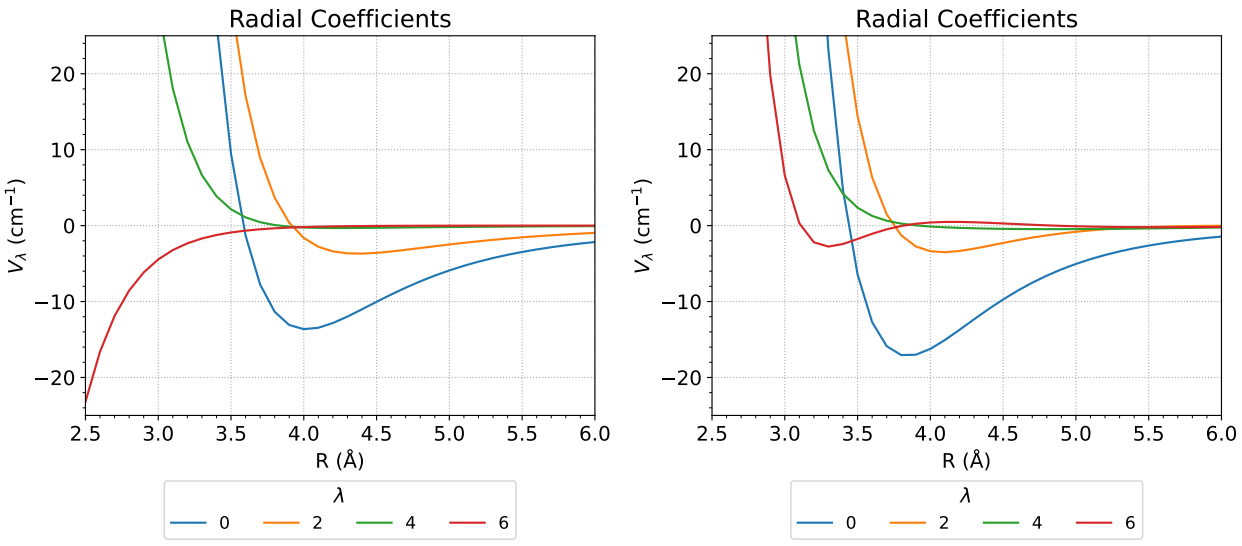
The same can be used to regenerate the PES using `fn2pes.py` Python code in `input_files/aux_scripts/` folder. Just set the parameters for C_2-He PES fit as shown below:

```
def fnfit_custom(x, a1,a2,a3,a4,a5):
    import numpy as np
    return  a1*np.exp(-4*x)+a2*np.exp(-3*x)+a3*np.exp(-2*x)+a4*np.exp(-1*x)+a5*np.exp(-0.2*x)

Num_coeff = 5                           # number of coefficients
R1        = np.arange(1,20,0.1)         # range of R1  1.0 Ang to 19.9 Ang step size 0.1 Ang
R2        = np.arange(20,51,5)          # range of R2 20.0 Ang to 50.0 Ang step size  5  Ang
R_range   = np.concatenate((R1, R2))    # combined R1 and R2 range

input_filename = "pes_fn.txt"           # filename having pes coefficients
```

Once a satisfactory fit for the PES has been obtained, the two PESs (HFD and SGS) are then expanded into radial terms (V_λ) up to four terms ($\lambda = 0, 2, 4, 6$), since we only have four angles in the PES. For molecules with large anisotropies in the PES, more radial terms are needed for the convergence of the radial terms. The input parameters for setting the same in `3_MPExp2D.py` are shown below:

```
lam_max   = 4           # Maximum expansion terms for lambda in V_lambda
symmetric = True        # Verify if the rigid rotor is symmetric (else put False)
```

The radial terms obtained after the multipole expansion of the function-fitted PES are shown in Fig. 4.22.

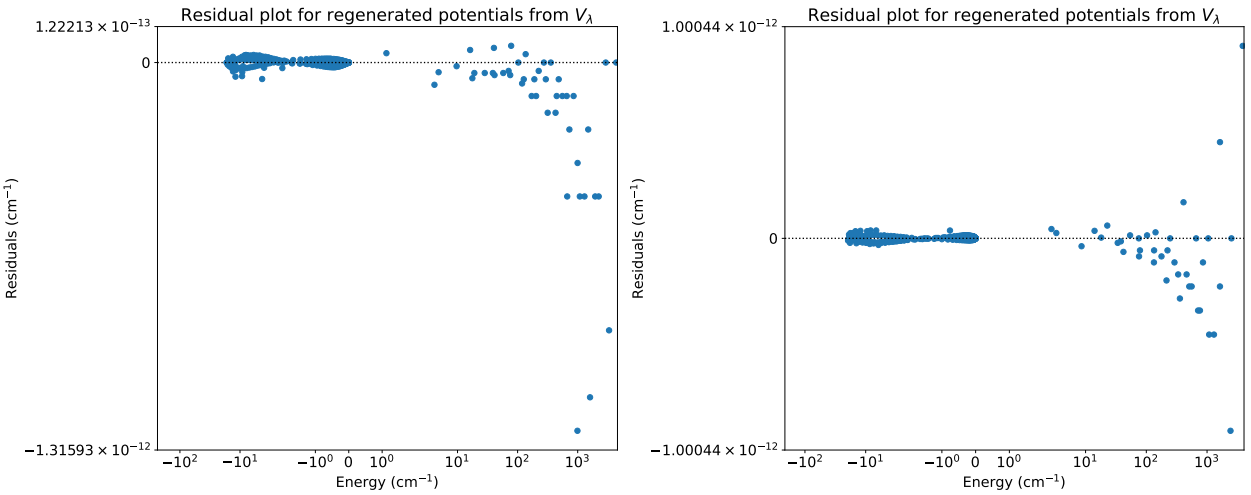


(a) HFD Radial Terms (V_λ) (b) SGS Radial Terms (V_λ)

Figure 4.22. Radial terms obtained after multipole expansion of function fitted HFD an SGS PES.

The radial terms obtained from the two methods are slightly different in depth and shape, with the third anisotropic term ($\lambda = 6$) showing an inverse behavior at small distance (R).

Fitting these radial terms into the Slater function mentioned above results in a fit that has RMSE of 10^{-14} and their residual plots (Regenerated PES is compared to function fitted PES) shown in Fig. 4.23 have a mean absolute error of 10^{-12} which shows that the fitting is only limited by machine precision. This exercise is done to show that fitting PES into radial functions can be beneficial in the final step, where we do not have to fit the radial terms again into Slater functions.



(a) HFD inverse residuals (b) SGS inverse residuals

Figure 4.23. Residual plot comparing the regenerated PES to the function fitted (a) HFD and (b) SGS PES. The inverse expansion of the radial terms is done to regenerate the PES.

Remember that the regenerated PES is compared to the function fitted HFD and SGS PES. If the same is compared to *ab initio* PES, the residual plot will be the same as shown in Fig. 4.21 (b) and (d).

The MOLSCAT coefficients file, `MOLSCAT_POT.txt` (obtained after fitting V_λ into the same analytical expression) is copied into `/GUI_examples/2_FnFitPES/2D/MOLSCAT/C2_He_SGS/sigma` folder to generate MOLSCAT input files (using `1_Gen_molscat_files.py`). The required Python files for extracting cross-sections and calculating rate coefficients are available inside the `../2D/MOLSCAT/C2_He_*/` folder.

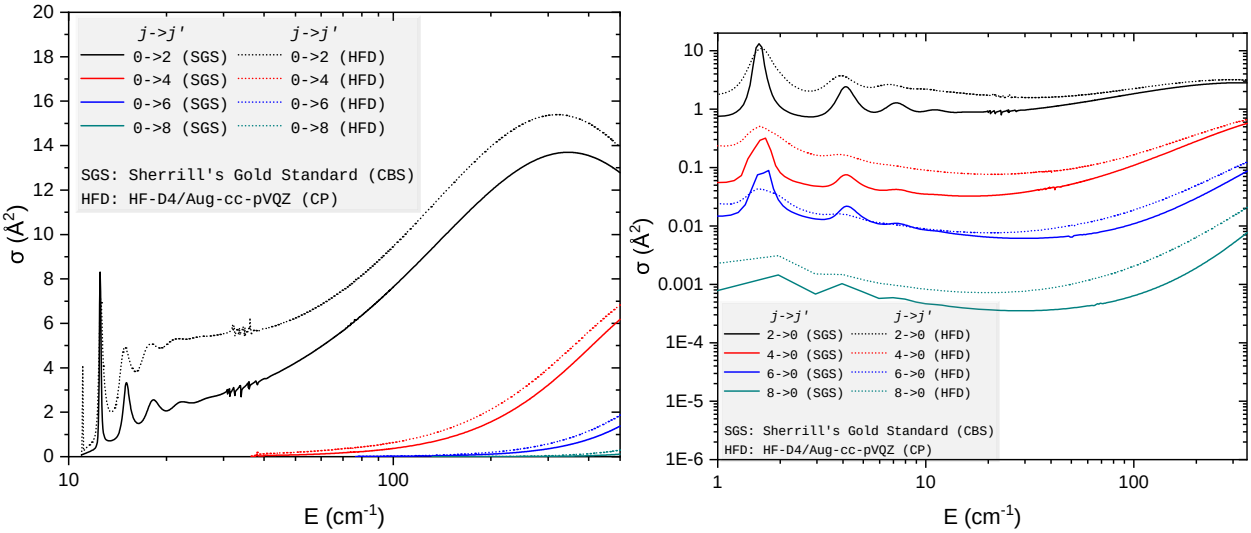
The MOLSCAT calculations (to calculate cross-section data) are carried out at collisional (total) energies ranging from 0.05 to 500 cm^{-1}. The step size for the collisional energies is kept variable: 0.05 cm^{-1} till 30 cm^{-1}, 0.1 cm^{-1} from 30.1 cm^{-1} to 100 cm^{-1}, and 1 cm^{-1} from 100 cm^{-1} to 500 cm^{-1}. The idea is to see how much the dynamical results change when two different levels of theory are used. Although their results only vary by a few cm^{-1}, HFD calculations are done in minutes, while the SGS method takes hours/days to compute the PES.

For molecules that have very small rotational constants, the convergence of cross-sections will be affected by the presence of energetically inaccessible low-lying rotational states (closed channels), and the resonances are therefore expected to show even larger deviations.

Since we had chosen a molecule (C_2) with a fairly large rotational constant with $I = 0$ and center of symmetry, the rotational states are well separated and only even rotational states with $\Delta j = \pm 2$ transitions are present. For de-excitation transitions, the energy of the current rotational state is subtracted from the collisional (total) energy to give kinetic energy. The overall results for cross-sections and rate coefficients are shown in Fig. 4.24 and Fig. 4.25, respectively. Both results are plotted externally using Origin.

The cross-section results show subtle differences in the intensity and position of Feshbach and shape resonances. These differences can be traced back to the shape of radial terms corresponding to the transition, for example, the isotropic term is mainly responsible for elastic cross-sections $\Delta j = 0$, the first anisotropic term for $\Delta j = 2$, and so on.

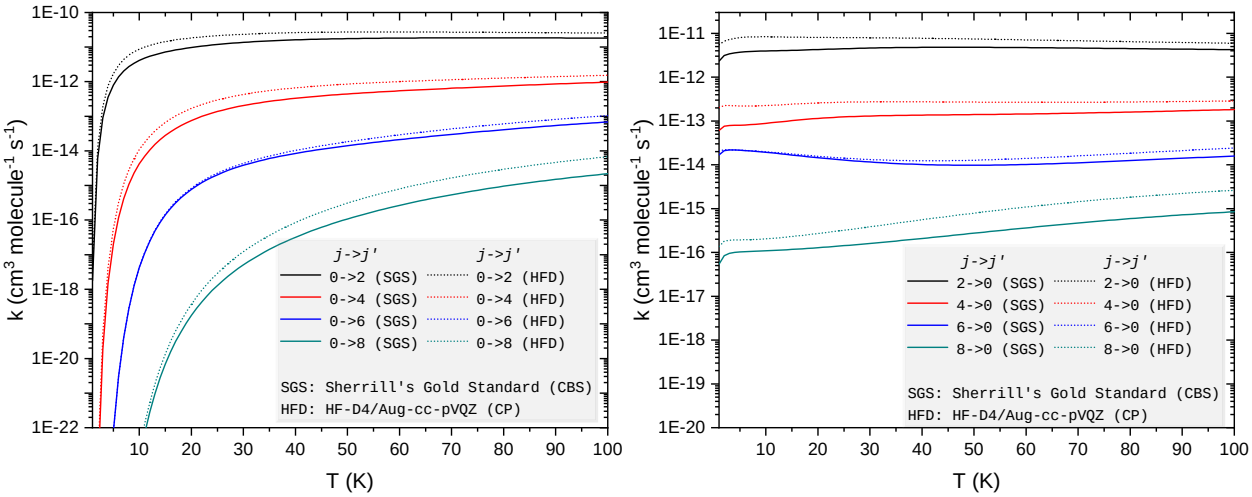
Since C_2 has an ideal anisotropy (low) and rotational constant (high) for benchmarking, we find that the overall characteristics of the results remain pretty similar, which is also reflected by the near-similar rate coefficients. However, if we look at the transition $0 \rightarrow 2$ in Fig. 4.24, there is an extra peak at the beginning for HFD PES, which is absent for SGS PES.



(a) Excitation ($0 \rightarrow j'$) (b) De-excitation ($j \rightarrow 0$)

Figure 4.24. Cross-sections (σ) for rotational transitions ($\Delta j = 2N$) of C_2 (a) from $j = 0$ till collisional energies ($E_c = 500$ cm^{-1}), and (b) to $j = 0$ till kinetic energies ($E_k = 350$ cm^{-1}). † *Plotted using Origin.*

Regarding the rate coefficients, the overall trend for rates is similar with the mean deviation of 65%, 100%, 30%, and 168% for $\Delta j = 2, 4, 6,$ and 8 transitions, respectively. Overall, the rate coefficients for HFD calculations are of the same order (as SGS) and therefore provide a quick reference point for the rough estimation of rate coefficients. However, these deviations in rate coefficients affect the estimation of molecular abundance in the ISM, due to the incorrect critical densities (from incorrect rates).



(a) Excitation ($0 \rightarrow j'$) (b) De-excitation ($j \rightarrow 0$)

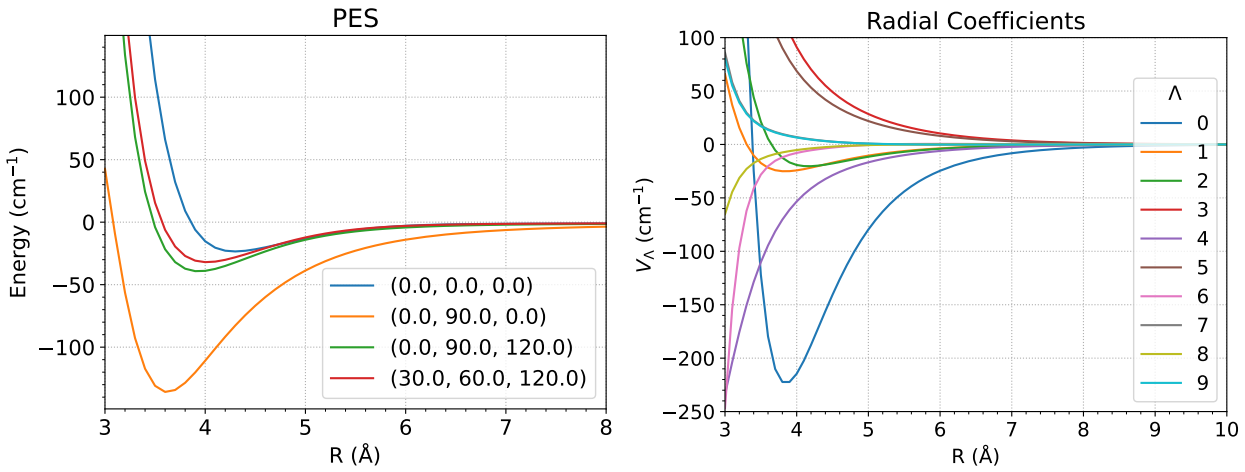
Figure 4.25. Rate coefficients (k) for rotational transitions ($\Delta j = 2N$) of C_2 to/from $j = 0$ till T = 50 K). † *Plotted using Origin.*

Therefore, for all practical purposes, such as ultracold experiments or astrochemical applications, the spectroscopic accuracy and shape of PES are of utmost importance. Users can repeat these calculations for various (singlet) molecules such as CO, HCN, NCCN, etc., at different levels of theories, such as CISD, HF-D3/D4, and F12 methods, and see how different the rate coefficients get. The practice will give new users physical insights into these calculations.

4D PES

For 4D, the input files for (a) calculation of C_2-H_2 4D PES, (b) function fitting, and (c) multipole expansion are provided. The procedure for calculating coefficients of radial terms is similar to the 2D case. However, MOLSCAT calculations for H_2 collision are different from He. Therefore, the same will be discussed below in detail.

The PES (HF-D4/cc-pVDZ) and radial terms for C_2-H_2 4D PES is shown below:



(a) PES (b) V_Λ

Figure 4.26. PES (HF-D4/cc-pVDZ) and radial terms for C_2-H_2 4D PES

Here $\Lambda = 0$ denotes $\lambda_1, \lambda_1, \lambda = 0, 0, 0$. The reference file for Λ is `Lambda_ref.dat` file inside the `MP_files/` folder. The input file `3_MPExp4D.py` has following parameters to indicate the maximum value of λ_1 and λ_2:

```
L1max = 4                    # max order for first radial term (L1)
L2max = 2                    # max order for second radial term (L2)
Symm_1 = True                # True if RR1 is symmetric, else False
Symm_2 = True                # True if RR2 is symmetric, else False
```

which results in ten radial terms as shown in Fig. 4.26 (b), which are calculated from 112 angular terms available in `Ang_Mat.dat` file (ϕ, θ_2, θ_1 = 0°, 0°, 0° to 90°, 90°, 180° with step size of 30° for all).

The molscat calculation files are provided inside the `MOLSCAT/` folder and contain three folders `p0-H2`, `p0,2-H2`, and `o1-H2`. ***Why three different calculations?***

The H_2 molecule has two protons that have nuclear spin $I = 1/2$. The molecule can possess two different spins, $I = 0$ if the spins pair up and $I = 1$ if the spins are parallel. This results in an interesting phenomenon, where para-hydrogen (p-H_2) always possesses an even rotational quantum number (j) while ortho-hydrogen (o-H_2) has odd j values. This makes them two completely different species when it comes to rotational dynamics.

Why `p0,2-H2` and not `o1,3-H2`?

The hydrogen molecule is very light, and therefore, the energy separation between levels (j_2 notation will be used for the collider rotational states) is very large. The $j_2 = 2$ energy levels are 0.000 cm^{-1} ($j_2 = 0$), 121.706 cm^{-1} ($j_2 = 1$), 365.118 cm^{-1} ($j_2 = 2$), and 730.236 cm^{-1} ($j_2 = 3$). Since the energy of $j_2 = 2$ (rotationally excited p-H_2: 365.118 cm^{-1}) level is usually accessible at low temperature (and therefore low kinetic energies) conditions of ISM, it is usually included in the theoretical studies to verify if the collisional rates are converging or not.

The $j_2 = 3$ (rotationally excited o-H_2: 730.236 cm^{-1}) level levels only become important for high energy regions usually illuminated by nearby stars. However, including an additional rotational basis for o-H_2 makes the number of channels too large to handle. Therefore, the same has not been included in the example files. However, if required, the same can be generated using $p0,2-H2$ files as reference. Specific to o-H_2 collision, the ground state energy of the collider is 121.706, and therefore, this energy is added into the `1_Gen_molscat_files.py` as shown below:

```
        f1.write('  &input ured = 1.8596, nnrg=1, energy=%.4f\n' %(121.7060+(i*step)) )
```

While calculating rate coefficients, it is important to indicate that there are two rotational basis in the collider (in `3_molrates.py` files) to get the correct labeling of rotational states for the rigid rotor (j_1):

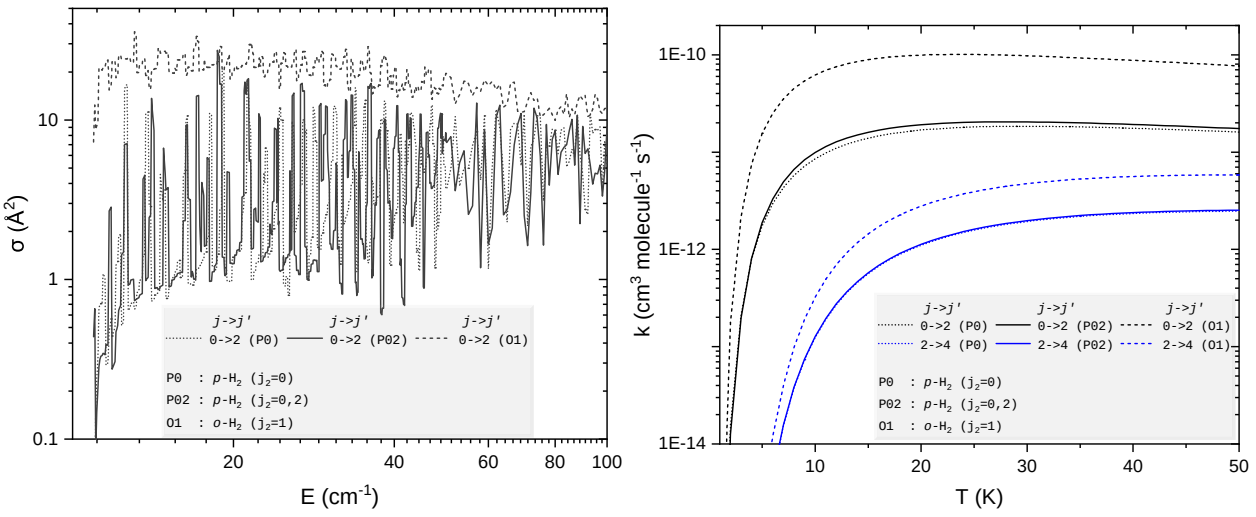
```
# Subtracting -1 from States and halving to give J (Keep subtract_1 = False)
two_j2      = True        # 1-->0, 2-->2 , 3-->4, etc...
```

For calculation of rate coefficients, the additional rotational energy of $j_2 = 1$ rotational state can be subtracted from the total collisional energy to get the kinetic energy in the *3_molrates.py* files:

```
# subtract energy: take relative energy
sub_E   = True    # If true, rotational energy (from pair_E file) is subtracted
pair_E = np.loadtxt("pair_E.dat")  # name for the file containing rotational energy
```

The `pair_E.dat` file is automatically generated based on pair energies (corresponding to the last MOLSCAT output file), however, the users can edit the same or create a new file for any special case (o-H_2 excitation) or some test calculations. The example files for the same have been made available with the program.

The results for cross-section and rate calculations of C_2 ($j \rightarrow j'$) for collision with p-H_2 ($j_2 = 0$), p-H_2 ($j_2 = 0, 2$), and o-H_2 ($j_2 = 1$) are shown in Fig. 4.27.



(a) Cross Sections (b) Rate Coefficients

Figure 4.27. Cross-section and rate coefficients for rotational excitation of C_2 by H_2. † *Plotted using Origin.*

There are a few takeaways from these results, like the cross-sections and rate coefficients for o-H_2 are found to be higher than p-H_2 (this is not always true: there are cases when the latter is larger than o-H_2 rates). Also, the correction for additional basis of p-H_2 ($j_2 = 0, 2$) is within $\backsim$ 5%, indicating that including an additional basis can be omitted as the $j_2 = 2$ state is relatively less populated at low temperatures (below 100 K). While the current results are limited to 100 cm−1 and 50 K. The same can be calculated till 1000+ cm−1 and 100+ K. However, compared to the He calculation (ITYPE=1), there are many more states in the rotational basis and therefore the cross-section calculations are more expensive for H_2 collisions (ITYPE=3).

4.4 3_NNFitPES: Augmenting PES using Ensemble NN Model

Generating sparse PES files and then fitting into a **PES-specific** supervised ensemble NN model.

**The NN model has custom activation functions (negative GELU and Gaussian functions), a constraint on R (decay function) to ensure desirable physical behavior at very-short and long-range regions:*

In the previous example, the PES was generated at an interval of 30° ($\Delta\theta$), giving $\lambda_{max} = 6$. However, for molecules, with large anisotropies $\Delta\theta = 15°$ or 10° may be needed to get $\lambda_{max} = 12$ and 18, respectively. Therefore, the current example discusses the steps (with input files) to augment a sparse 2D/4D PES generated using Psi4.

1. 2D: Example for generating (i) a sparse PES for C_2-He collision, (ii) fitting into the NN model, (iii) fitting NN augmented PES into analytical expression and expansion into radial terms, and (iv) using the same mathematical expression to fit radial terms (also generating MOLSCAT's &POTL file).

2. 4D: Example for generating (i) a sparse PES for C_2-H_2 collision, (ii) fitting into the NN model, (iii) fitting NN augmented PES into analytical expression and expansion into radial terms, and (iv) using the same mathematical expression to fit radial terms (generating MOLSCAT's &POTL file).

In the previous examples, we saw how to extrapolate/interpolate missing data points using analytical fitting of PES into a function of R. However, in the final example, we shall see how NN models can be utilized to augment the PES in full dimensionality (R, $\theta(s)$). The PES examples are the same for 2D and 4D PES, i.e. C_2-He and C_2-H_2 collision. The only difference being that, right after PES generation, the resulting surface is augmented by NN model and then the PESPlot module is used to plot the surface. The rest of the calculation (to get analytically expressed radial coefficients) remains the same. Users can either (a) directly proceed for MP expansion of NN generated PES and fit the radial terms (by finding the best suitable function), or (b) fit the NN generated surface and then proceed for MP expansion. In the second example, the same function can be used for fitting the radial terms.

NNGen Results

The 2D and 4D PES is augmented using default settings, however, users can tweak the same (or use templates settings provided in `/input_files/3_NN_Opt/` folder) for a better fit. See Section 2.5.9 for more details.

∗ If some default parameters like number of epochs or number of trial models differ from the provided examples, the same have been tweaked later in the code to improve the performance of the NN models.

Fig. 4.31 shows the main folder preview generated by the NNGen program, where three main folders exist, `NN_plots/`, `NN_trial_models/`, and `NN_final_models/` that contain NN plot data, trial models for architecture search , and final models (base and ensemble), respectively.

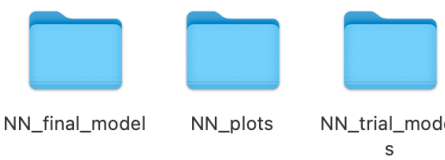


Figure 4.28. Preview of `NN_files` folder generated by NNGen module in the `$Proj_name` folder.

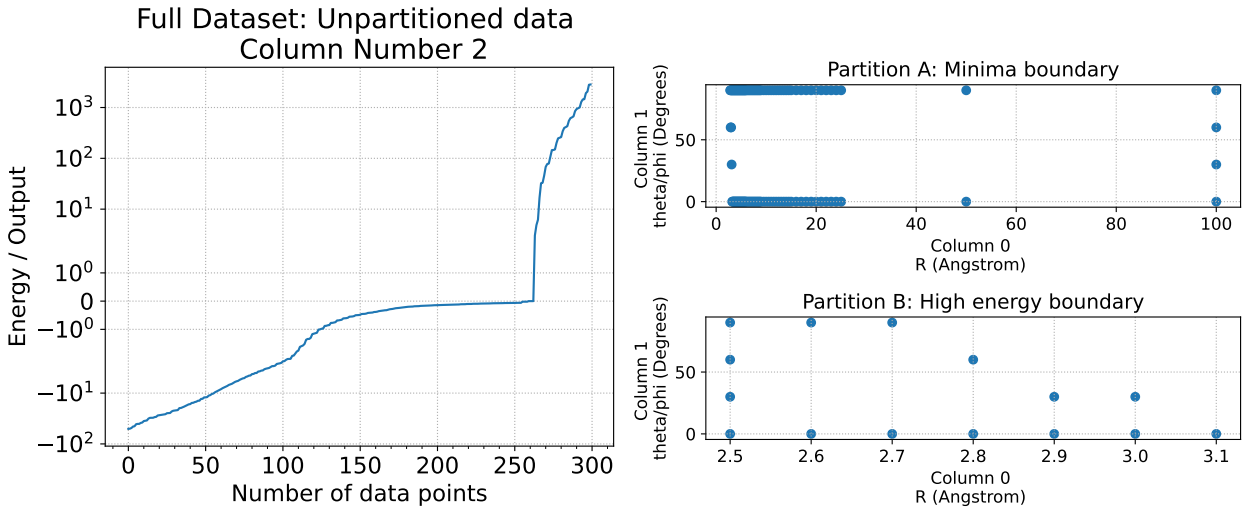
We shall discuss the output files in each folder in detail.

The preview of the plot folder is shown in Fig. 4.29, where the boundary separated and minima+asymptotic region's data are saved in `boundary_data` and `partition_data` folders, respectively. The plots for the same are available in the main folder as visible form the folder preview.

Figure 4.29. Preview of `NN_plots` folder inside `NN_files` folder.

Two of the plots in `NN_plots` folder are also provided in Fig. 4.30. The Fig. 4.30(a) shows the plot of potentials (sorted) *vs* number of data points (2D un-partitioned data: `2dunpart.pdf`). The data and plot after timing high energy data is also available in the folder (2D partitioned data: `Partitioned_Dataset_....pdf`).

The program also separates the boundary elements and adds them to the training dataset to prevent boundary errors. The plot for boundary elements (θ `vs` R plot for 2D NN boundary: `2dnnbd.pdf`) is shown in Fig. 4.30(b). The current example shows 2D outputs, and has single plot for boundary separation. For 4D PES and PES with multiple energies, the program generates multiple plots (and its data files) in the folder `NN_files/NN_plots/`.



(a) Untrimmed PES (b) Boundary separation

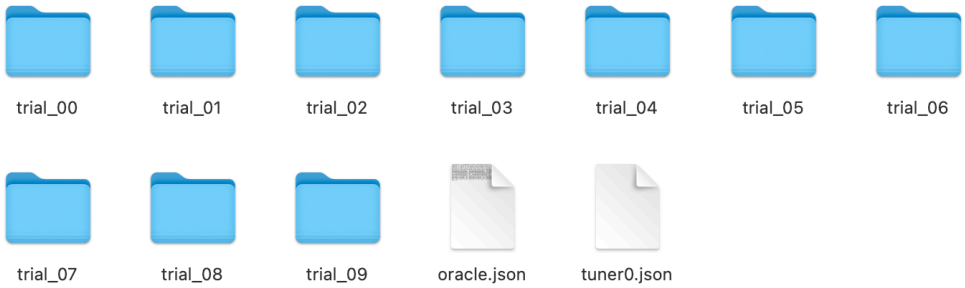
Figure 4.30. NN plots for (a) energy range (sorted) of full PES dataset and (b) extracted boundary data (boundary elements are merged with the training dataset).

The input file command to trim (partition) high-energy region of the PES is provided below:

```
#----------------------------- Partition commands ---------------------------------#
partition_req    = True       # partition data -> High Energy(HE) and minima region
High_E_cutoff    = 300        # E in cm-1 ---> Model Energies lower than this cutoff
reference_output = 2          # Column number for sorting and separating minima/HE
                              # Column numbering starts from 0
#--------------------------------------------------------------------------------#
#HE_train        = False      # High energy training off (default)
                              # High energy region approximated using R constraint
#--------------------------------------------------------------------------------#
```

Once the datasets have been prepared (boundary extracted, high-energy region trimmed and features scaled), the program will ask users for a folder name where NN model will be saved (`$NNrun`). In Fig. 4.31, the folder containing trial model data is shown. Fig. 4.31(a) shows a preview of `NN_trial_models` folder, where *mod_i* folders present for each base models (based on input file where we mention split ratio for datasets). The preview inside `mod_0` is shown in Fig. 4.31(b). The full path of the same will be `NN_trial_models/$NNrun/mod_0/`, where `$NNrun` is the user entered NN folder model name for that run (asked in terminal during execution).

(a) `../NN_trial_models/`



(b) `../NN_trial_models/mod_0/`

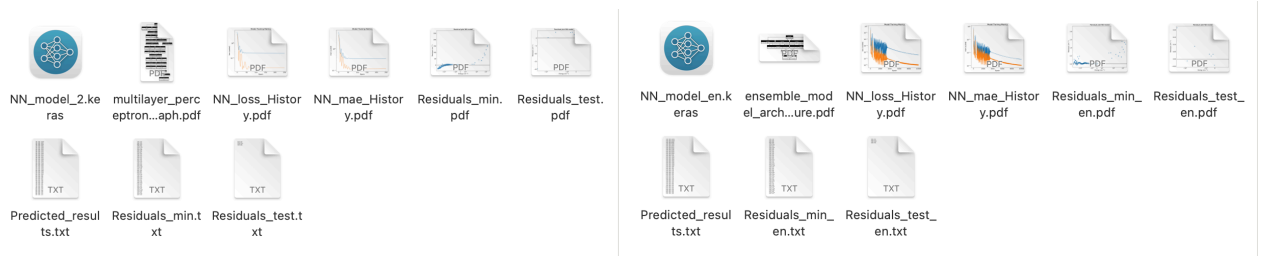
Figure 4.31. Preview of (a)`NN_trial_models` and (b) `NN_trial_models/mod_0` folders inside `NN_files` folder. The subplot (a) shows four folders for each base model, while subplot (b) shows that 10 trial models are generated to search optimum architecture for each base model.

For each *mod_i*, the program runs 10 (default) trials for searching the best architecture, which are shown as `trial_ii` folders. These files are generated and read by keras-tuner library. The final (best) architecture for that split ratio is then used for base model and trained inside `NN_final_model/$NNrun/` folder, as shown in Fig. 4.32.

The base models are trained and saved as `0,1,2,` etc. folders, and the final model is saved in the `Ensemble_tuned` folder. The scaling parameters of input and output features is saved in a `feature_scaler` and `target_scaler` pickle (`.pkl`) files, respectively.

Figure 4.32. Preview of `/NN_files/NN_final_model/$NNrun` folder, where `$Nnrun` is user entered NN project name. The 4 base models are stored inside the numbered folders and the final ensemble model along with augmented PES data is present in `Ensemble_tuned` folder.

A preview inside one of the base models and final ensemble model folder is provided in Fig. 4.33, where both base and ensemble models have a Keras (model file), training loss/mae and residuals plots along with predicted results. The architecture of the model is also saved in `pdf` format. The architecture along with coefficients can also be visualized by opening the `.keras` file using software like Netron.



(a) Base Model Output Folder (b) Ensemble Model Output Folder

Figure 4.33. Output files for base and ensemble NN models.

The architecture plots for one of the the base model is shown below in Fig. 4.34 . The base model in Fig. 4.34 is trained with a constrain on R.

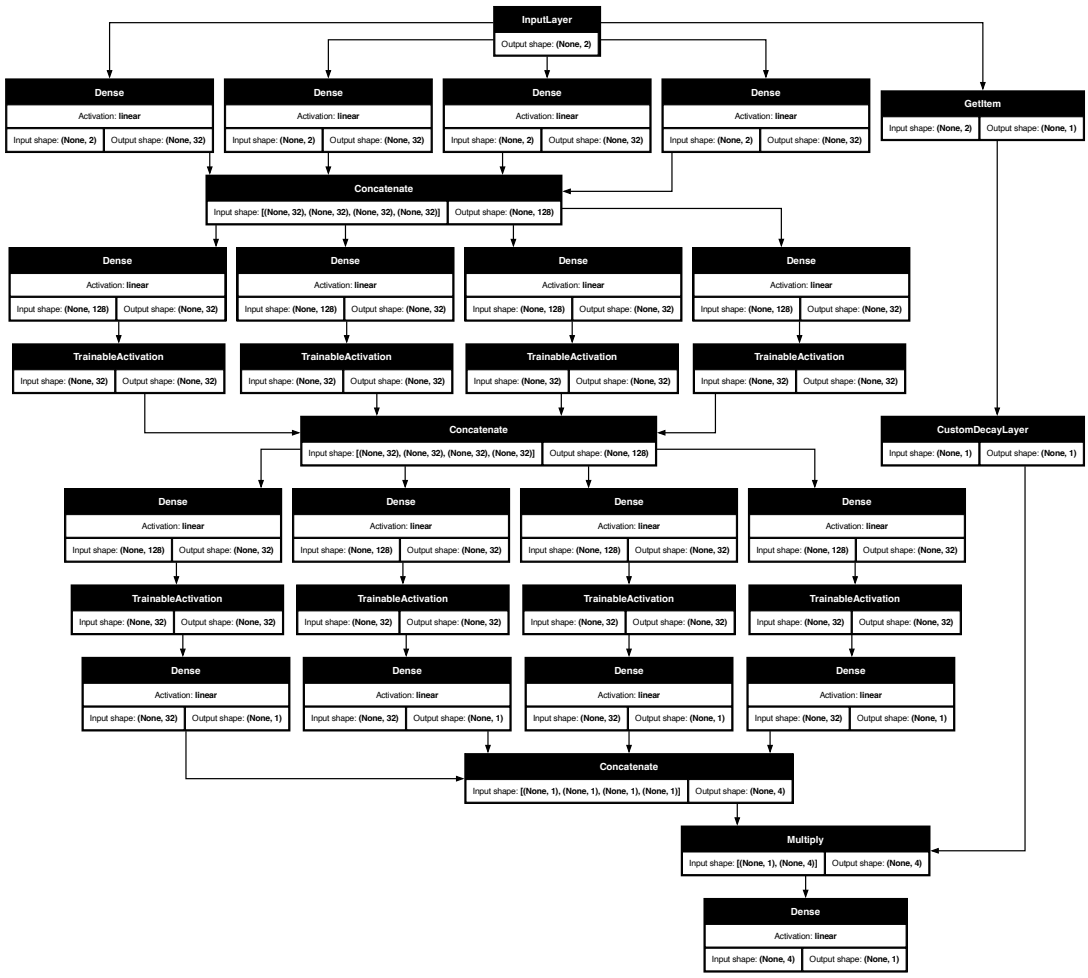


Figure 4.34. NN architecture for one of the base models.

The architecture plots for ensemble model is shown in Fig. 4.35. Each Functional layer in the ensemble model represents a base model as shown in Fig. 4.34. Users can also modify the code and play around with different ensemble architectures to see if a more complex layer or a simple average layer gives better results.

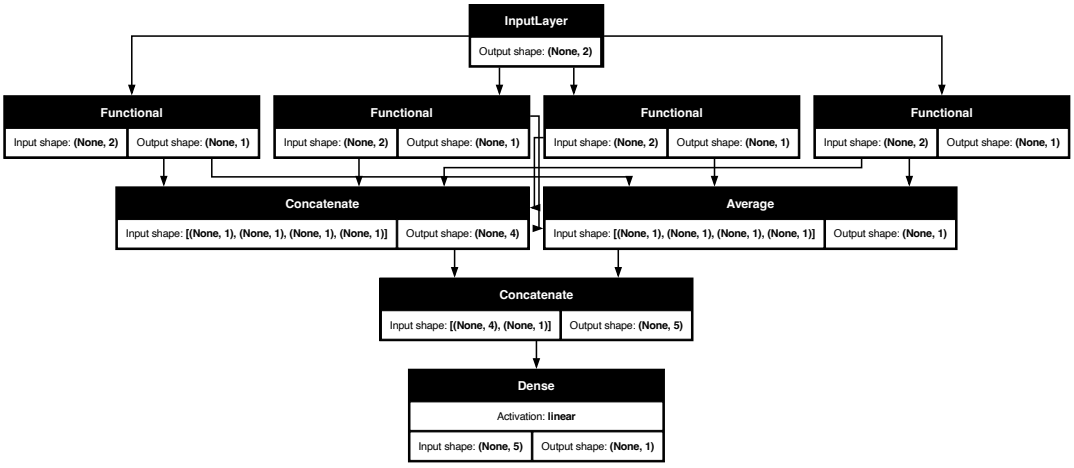
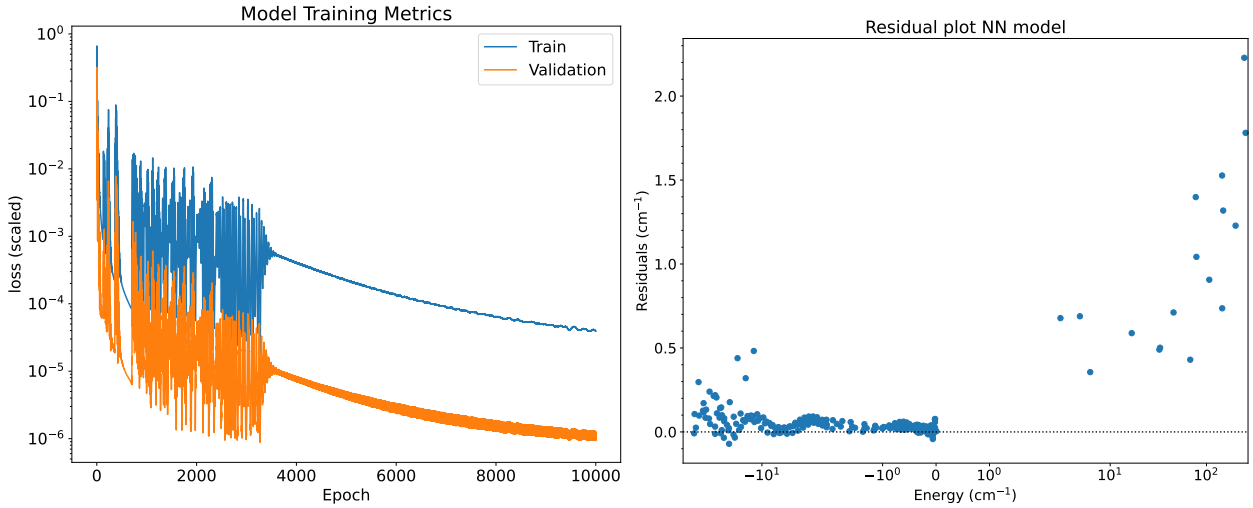


Figure 4.35. NN architecture for the ensemble model.

Since we have used the default template where the base models are only trained for a few epochs before they are trained with all other models in the ensemble model training, we shall see how the training proceeded in Fig. 4.36. The two plots are taken from `Ensemble_tuned` folder, where Fig. 4.36(a) shows how training error changes with number of epochs and Fig. 4.36(b) shows the residual plot (error in data after model has been fully trained).



(a) Training History (b) Residual Plot

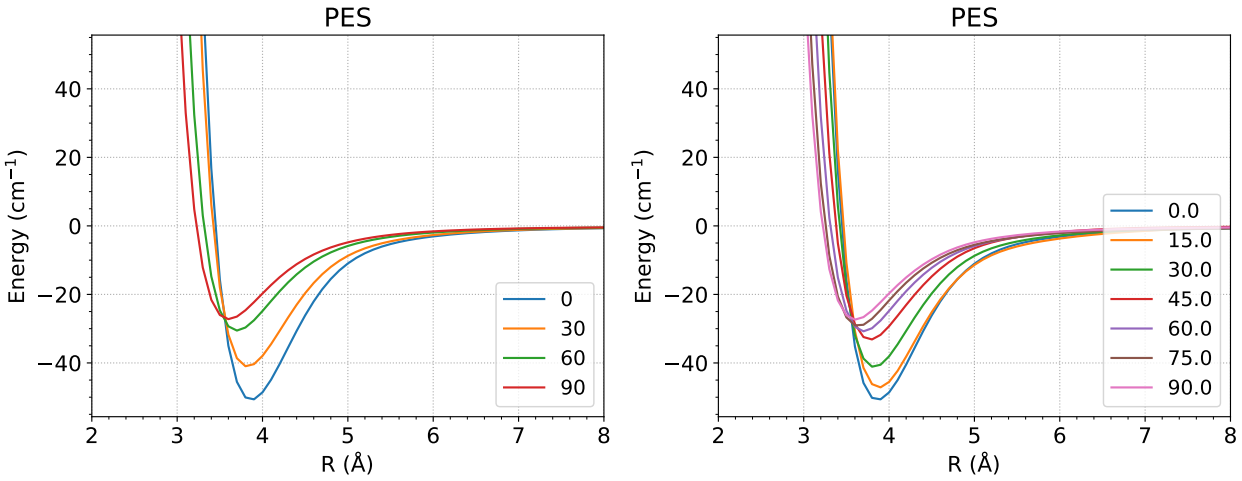
Figure 4.36. Example of (a) an ensemble model training history where validation dataset is very small, resulting in smaller loss than train dataset throughout NN training epochs (10,000), and (b) residuals based on its predicted results.

In the plot, we can see that the training loss is higher than the validation loss throughout the training. One of the main reasons for this behavior is that the split dataset has very few points in the validation dataset, and therefore, its loss is much smaller. Since the ensemble model uses the split ratio of the very first

model (the user input with multiple split ratios of train and test datasets), it is recommended to balance the number of test and validation datasets, otherwise the NN models will not train with maximum efficiency.

In the end, the residual plots are what give us the idea of the NN fitting. The same are available inside the base and ensemble model folders. In the current example, 4 base models and default settings (for architecture search, number of training epochs, and early stopping) have augmented the 2D PES within spectroscopic accuracy. However, for molecules with large anisotropies in the PES, more angles (i.e., more *ab initio* data points) and more number of epochs will be needed.

The final output, i.e., ensemble model's predicted results are saved as `NN_Predicted_results.txt` in the main project directory. The same cn be plotted using `opt_NNfit.py` input file and the PES will be saved in the `plots/` folder (inside the main project directory). The NN augmented PES for 2D collision (C_2-He) next to *ab initio* PES is shown in Fig. 4.37.



(a) *Ab initio* PES (b) NN augmented PES

Figure 4.37. *Ab initio* PES for 2D collision (C_2-He) next to NN augmented PES.

The resulting plot shows that the augmented angles (15°, 45°, and 75°,) fit perfectly in between the the training data (0°, 30°, 60°, and 95°,). The PES shows no artificial minima/maxima and retains the shape (and physical behavior) of the PES even when extrapolated to extreme R values. A 3D plot for the same (zoomed at minima region) is provided in 4.38.

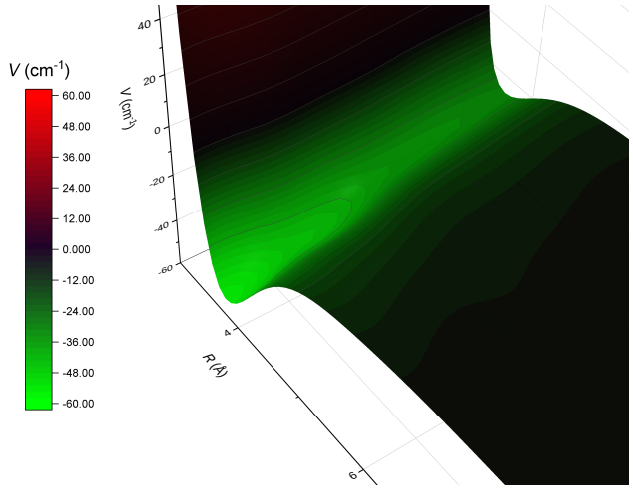


Figure 4.38. NN augmented PES ($\Delta\theta = 1°$) trained using *ab initio* PES ($\Delta\theta = 30°$) for C_2-He collision.

The saved Keras model! Users can also open the .keras file to visualize the architecture and optimized coefficients using external software if needed. The output for one of the base model visualized using netron (app) is shown in Fig. 4.39.

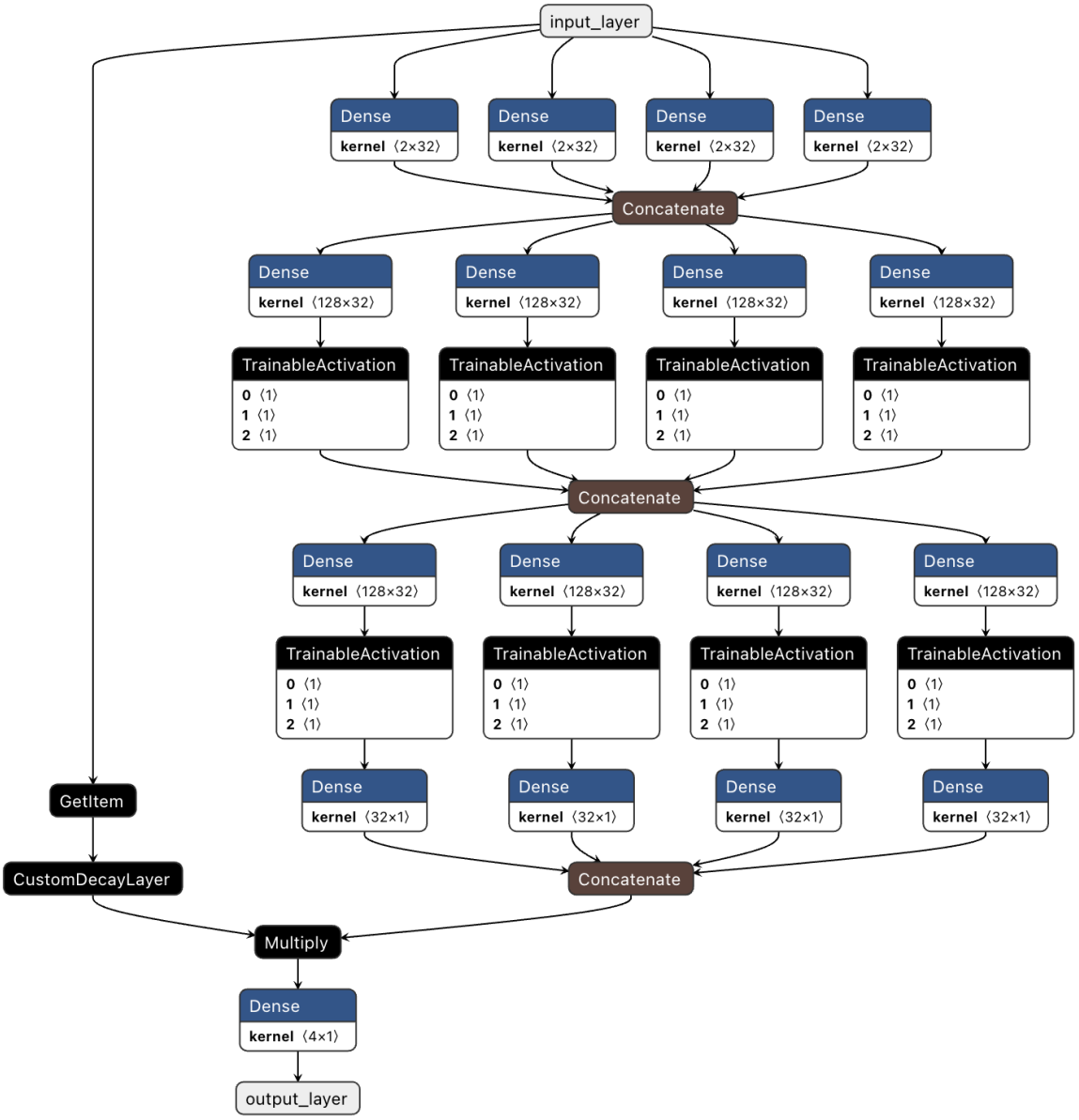


Figure 4.39. Base model (.keras) file visualized using external application (netron).

If required, the same can be loaded externally to predict the PES based on new coordinates. The reference code for the same can be found commented inside the main `PES2MP.py` file.

4.5 Discussion

Apart from examples discussed above, please also look at the example 999_PES discussed in the previous Chapter in Section 3.3.2, where the challenges of fitting the PES using suitable functions are discussed in detail. That section covers input and output files for analytical fitting of PES in the (a) full range of PES ('custom' function), (b) high-energy region, and (c) long-range region. It also describes how β coefficients must be set, for example, a very small β coefficient decays slowly, and can be used for mimicking long-range behavior in charged systems. So, including them in the 'custom' function is a good practice. Also, the range of β coefficients must be such that it covers the positions of minima. Suppose a very long rigid rotor has minima ranging from $R = 2 - 6$ Å, then using multiple Slater functions with β coefficients in this range is recommended. First, verify the sanity of the fit with fewer functions, as some angles may fit better than others, and then increase the functions carefully to check when fitting error saturates and overfitting begins.

4.5.1 Rate Coefficients, Einstein coefficients, and RADEX code!

The rate coefficients

In time-independent dynamics, we study the collision of interstellar molecules with abundant colliders such as H_2, He, $H^{\bullet}$, or e^{-} that can alter the population of rotational energy levels. The efficiency of rotational excitation and de-excitation is quantified by collisional rate coefficients. In ISM, such transitions depend on temperature [182–184] (more temperature - more kinetic energy) and density [185] (more dense - more collisions) of that region. The rate coefficients (experimental or theoretical) are therefore needed to model the radiative processes in ISM (that we ultimately observe on Earth through probes/satellites).

Einstein coefficients

Spontaneous Emission – Einstein A coefficient: An excited (higher rotational level) molecule can return to a lower state by emitting a photon. The probability per unit time of this process is quantified by the Einstein A coefficient, and is important for low-density regions where collisions are rare. Therefore, in highly diffuse clouds, spontaneous emission becomes the dominant de-excitation mechanism.

Stimulated Emission – Einstein B coefficient: Einstein B coefficients describe two processes, one for absorption (exciting the molecule to a higher rotational state by absorbing a photon) and another for stimulated emission (inducing the molecule to drop to a lower state while emitting a photon). These coefficients describe the probability of rotational transitions in the presence of the radiation field.

RADEX code

In ISM, both collisional and radiative processes occur simultaneously. Einstein coefficients provide a measure of the inherent radiative decay properties of the molecule, while the collisional rate coefficients describe the non-radiative energy exchange. The balance between these processes is often summarized by the concept of "critical density", i.e., the density at which the rate of collisional de-excitation equals the rate of spontaneous emission [180]. In environments below the critical density, radiative processes dominate, whereas above this threshold, collisions drive the excitation conditions toward thermal equilibrium.

Using collisional and radiative data, RADEX [186] computes the populations of the rotational levels. These populations are then used to predict the brightness and shape of spectral lines. Such predictions are essential for interpreting astronomical observations, as they allow researchers to infer physical conditions like density [181], temperature, and the radiation environment within molecular clouds under non-LTE conditions.

Also check out Python implemented RADEX code.

Bibliography

[1] J. D. Hunter, "Matplotlib: A 2D graphics environment", Comput. Sci. En **9**, 90–95 (2007) (cit. on p. 1).

[2] D. G. A. Smith, L. A. Burns, A. C. Simmonett, R. M. Parrish, M. C. Schieber, R. Galvelis, P. Kraus, H. Kruse, R. Di Remigio, A. Alenaizan, A. M. James, S. Lehtola, J. P. Misiewicz, M. Scheurer, R. A. Shaw, J. B. Schriber, Y. Xie, Z. L. Glick, D. A. Sirianni, J. S. O'Brien, J. M. Waldrop, A. Kumar, E. G. Hohenstein, B. P. Pritchard, B. R. Brooks, H. F. Schaefer, A. Y. Sokolov, K. Patkowski, A. E. DePrince, U. Bozkaya, R. A. King, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, "PSI4 1.4: Open-source software for high-throughput quantum chemistry", J. Chem. Phys. **152**, 184108 (2020) (cit. on pp. 1, 27).

[3] E. Caldeweyher, C. Bannwarth, and S. Grimme, "Extension of the D3 dispersion coefficient model", J. Chem. Phys. **147**, 034112 (2017) (cit. on p. 1).

[4] E. Caldeweyher, S. Ehlert, A. Hansen, H. Neugebauer, S. Spicher, C. Bannwarth, and S. Grimme, "A generally applicable atomic-charge dependent London dispersion correction", J. Chem. Phys. **150**, 154122 (2019) (cit. on p. 1).

[5] M. Friede, S. Ehlert, S. Grimme, and J.-M. Mewes, "Do Optimally Tuned Range-Separated Hybrid Functionals Require a Reparametrization of the Dispersion Correction? It Depends", J. Chem. Theory Comput **19**, 8097–8107 (2023) (cit. on p. 1).

[6] L. Wittmann, I. Gordiy, M. Friede, B. Helmich-Paris, S. Grimme, A. Hansen, and M. Bursch, "Extension of the D3 and D4 London dispersion corrections to the full actinides series", Phys. Chem. Chem. Phys. **26**, 21379–21394 (2024) (cit. on p. 1).

[7] P. Raybaut, "Spyder-documentation", GitHub.com: Spyder (2009) (cit. on p. 1).

[8] E. Carrera, "PyDot", GitHub.com: PyDot (2004) (cit. on p. 1).

[9] S. Bank, "graphviz - Simple Python interface for Graphviz", PyPi.com: Graphviz (2015) (cit. on p. 1).

[10] C. O. da Costa-Luis, "tqdm: A Fast, Extensible Progress Meter for Python and CLI", J. Open Source Softw **4**, 1277 (2019) (cit. on p. 1).

[11] M. A. Wieczorek and M. Meschede, "SHTools: Tools for Working with Spherical Harmonics", Geochem. Geophys. Geosyst. **19**, 2574–2592 (2018) (cit. on p. 1).

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python", J. Mach. Learn. Res. **12**, 2825–2830 (2011) (cit. on p. 1).

[13] W. McKinney, "Data Structures for Statistical Computing in Python", in Proceedings of the 9th Python in Science Conference, edited by S. van der Walt and J. Millman (2010), pp. 56–61 (cit. on p. 1).

[14] M. Newville, T. Stensitzki, D. B. Allen, M. Rawlik, A. Ingargiola, and A. Nelson, "Lmfit: Non-Linear Least-Square Minimization and Curve-Fitting for Python", Astrophys. Source Code Libr., ascl:1606.014 (2016) (cit. on p. 1).

[15] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy", Nature **585**, 357–362 (2020) (cit. on p. 1).

[16] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", Nature Methods **17**, 261–272 (2020) (cit. on p. 1).

[17] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter Notebooks – a publishing format for reproducible computational workflows", in Positioning and power in academic publishing: players, agents and agendas, edited by F. Loizides and B. Schmidt (IOS Press, 2016), pp. 87–90 (cit. on p. 1).

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, `http://tensorflow.org/`, Software available from tensorflow.org, 2015 (cit. on p. 1).

[19]T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, *Kerastuner*, https://github.com/keras-team/keras-tuner, 2019 (cit. on p. 1).

[20]N. Richardson, I. Cook, N. Crane, D. Dunnington, R. François, J. Keane, D. Moldovan-Grünfeld, J. Ooms, J. Wujciak-Jens, and Apache Arrow, *arrow: Integration to 'Apache' 'Arrow'*, R package version 19.0.1 (2025) (cit. on p. 1).

[21]D. R. Flower, "Molecular collision processes in interstellar clouds", Phys. Rep. **174**, 1–66 (1989) (cit. on p. 5).

[22]E. Herbst, "The chemistry of interstellar space", Chem. Soc. Rev. **30**, 168–176 (2001) (cit. on p. 5).

[23]T. E. Holzer, "Neutral hydrogen in interplanetary space", Rev. Geophys. **15**, 467–490 (1977) (cit. on p. 5).

[24]W. D. Watson, "Interstellar molecule reactions", Rev. Mod. Phys. **48**, 513–552 (1976) (cit. on p. 5).

[25]S. Petrie, T. J. Millar, and A. J. Markwick, "NCCN in TMC-1 and IRC+10216", Mon. Not. R. Astron. Soc. **341**, 609–616 (2003) (cit. on p. 5).

[26]L. Gonzlez-Snchez, N. Sathyamurthy, and F. A. Gianturco, "The role of small molecular cations in the chemical flow of the interstellar environments", Phys. Chem. Chem. Phys. **25**, 23370–23383 (2023) (cit. on p. 5).

[27]I. W. M. Smith, "Gas-Phase Reactions in the ISM: Rate Coefficients, Temperature Dependences, and Reaction Products", Proc. Int. Astron. Union **7**, 361–371 (2011) (cit. on p. 5).

[28]E. Herbst, "Rate Equation Models", in *Encyclopedia of Astrobiology* (Springer, Berlin, Germany, July 2023), pp. 2627–2628 (cit. on p. 5).

[29]A. Faure, "Collisional rate coefficients for astrophysics", Proc. Int. Astron. Union **18**, 123–132 (2022) (cit. on p. 5).

[30]E. Herbst, "Chemistry in the Interstellar Medium", Annu. Rev. Phys. Chem., 27–54 (1995) (cit. on p. 5).

[31]P. Pratap, J. E. Dickens, R. L. Snell, M. P. Miralles, E. A. Bergin, W. M. Irvine, and F. P. Schloerb, "A Study of the Physics and Chemistry of TMC-1", Astrophys. J. **486**, 862 (1997) (cit. on p. 5).

[32]B. T. Draine, "Interstellar Dust Grains", Annu. Rev. Astron. Astrophys, 241–289 (2003) (cit. on p. 5).

[33]G. M. Green, "dustmaps: A Python interface for maps of interstellar dust", J. Open Source Softw **3**, 695 (2018) (cit. on p. 5).

[34]E. Herbst, "The synthesis of large interstellar molecules", Int. Rev. Phys. Chem. (2017) (cit. on p. 6).

[35]P. Ehrenfreund and S. B. Charnley, "Organic Molecules in the Interstellar Medium, Comets, and Meteorites: A Voyage from Dark Clouds to the Early Earth", Annu. Rev. Astron. Astrophys, 427–483 (2000) (cit. on p. 6).

[36]E. Herbst and E. F. van Dishoeck, "Complex Organic Interstellar Molecules", Annu. Rev. Astron. Astrophys, 427–480 (2009) (cit. on p. 6).

[37]P. Chahal and T. J. Dhilip Kumar, "PO^+-He collision: *ab initio* potential energy surface and inelastic rotational rate coefficients", Mon. Not. R. Astron. Soc. **523**, 5869–5875 (2023) (cit. on p. 6).

[38]J. Crovisier, "Rotational and vibrational synthetic spectra of linear parent molecules in comets", Astron. Astrophys. Suppl. Ser. **68**, 223–258 (1987) (cit. on p. 6).

[39]N. Teanby, P. Irwin, R. de Kok, A. Jolly, B. Bézard, C. Nixon, and S. Calcutt, "Titan's stratospheric C_2N_2, C_3H_4, and C_4H_2 abundances from Cassini/CIRS far-infrared spectra", Icarus **202**, 620–631 (2009) (cit. on p. 6).

[40]H. S. P. Mller, S. Thorwirth, D. A. Roth, and G. Winnewisser, "The Cologne Database for Molecular Spectroscopy, CDMS", Astron. Astrophys. **370**, L49–L52 (2001) (cit. on p. 6).

[41]D. Flower, *Molecular Collisions in the Interstellar Medium* (Cambridge University Press, 2007) (cit. on p. 6).

[42]F. Lique and A. Faure, *Gas-Phase Chemistry in Space – From elementary particles to complex organic molecules* (IOP Publishing, 2019) (cit. on p. 6).

[43]A. M. Shaw, *Astrochemistry: The Physical Chemistry of the Universe* (Wiley, 2021) (cit. on p. 6).

[44]A. McKellar, "Evidence for the Molecular Origin of Some Hitherto Unidentified Interstellar Lines", Publ. Astron. Soc. Pac. **52**, 187 (1940) (cit. on p. 7).

[45]K. B. Jefferts, A. A. Penzias, and R. W. Wilson, "Observation of the CN Radical in the Orion Nebula and W51", Astrophys. J. **161**, L87 (1970) (cit. on p. 7).

[46]B. A. McGuire, R. A. Loomis, A. M. Burkhardt, K. L. K. Lee, C. N. Shingledecker, S. B. Charnley, I. R. Cooke, M. A. Cordiner, E. Herbst, S. Kalenskii, M. A. Siebert, E. R. Willis, C. Xue, A. J. Remijan, and M. C. McCarthy, "Detection of two interstellar polycyclic aromatic hydrocarbons via spectral matched filtering", Science **371**, 1265–1269 (2021) (cit. on p. 7).

[47]M. Gulin and J. Cernicharo, "Organic Molecules in Interstellar Space: Latest Advances", Front. Astron. Space Sci. **9**, 787567 (2022) (cit. on p. 7).

[48]Agúndez, M., Cernicharo, J., de Vicente, P., Marcelino, N., Roueff, E., Fuente, A., Gerin, M., Guélin, M., Albo, C., Barcia, A., Barbas, L., Bolaño, R., Colomer, F., Diez, M. C., Gallego, J. D., Gómez-González, J., López-Fernández, I., López-Fernández, J. A., López-Pérez, J. A., Malo, I., Serna, J. M., and Tercero, F., "Probing non-polar interstellar molecules through their protonated form: Detection of protonated cyanogen ($NCCNH^+$)", Astron. Astrophys. **579**, L10 (2015) (cit. on p. 7).

[49]M. B. Bell, P. A. Feldman, M. J. Travers, M. C. McCarthy, C. A. Gottlieb, and P. Thaddeus, "Detection of $HC_{11}N$ in the Cold Dust Cloud TMC-1", Astrophys. J. **483**, L61 (1997) (cit. on p. 7).

[50]R. Kołos and Z. R. Grabowski, "The chemistry and prospects for interstellar detection of some dicyanoacetylenes and other cyanoacetylene-related species", Astrophys. Space Sci. **271**, 65–72 (2000) (cit. on p. 7).

[51]V. G. Kunde, A. C. Aikin, R. A. Hanel, D. E. Jennings, W. C. Maguire, and R. E. Samuelson, "C_4H_2, HC_3N and C_2N_2 in Titan's atmosphere", Nature **292**, 686–688 (1981) (cit. on p. 7).

[52]D. L. Lambert, Y. Sheffer, and S. R. Federman, "Hubble Space Telescope Observations of C_2 Molecules in Diffuse Interstellar Clouds", Astrophys. J. **438**, 740 (1995) (cit. on p. 7).

[53]M. F. A'Hearn, S. Hoban, P. V. Birch, C. Bowers, R. Martin, and D. A. Klinglesmith, "Cyanogen jets in comet Halley", Nature **324**, 649–651 (1986) (cit. on p. 7).

[54]E. F. Van Dishoeck and G. A. Blake, "Chemical evolution of star-forming regions", Annu. Rev. Astron. Astrophys. **36**, 317–368 (1998) (cit. on p. 7).

[55]P. A. M. Dirac, "Is there an Æther?", Nature **168**, 906–907 (1951) (cit. on p. 7).

[56]P. A. M. Dirac, "The Cosmological Constants", Nature **139**, 323 (1937) (cit. on p. 7).

[57]S. Staggs, J. Dunkley, and L. Page, "Recent discoveries from the cosmic microwave background: a review of recent progress", Rep. Prog. Phys. **81**, 044901 (2018) (cit. on p. 8).

[58]E. Gawiser and J. Silk, "The cosmic microwave background radiation", Phys. Rep. **333-334**, 245–267 (2000) (cit. on p. 8).

[59]F. Hoyle, "The Universe: Past and Present Reflections", Annu. Rev. Astron. Astrophys., 1–36 (1982) (cit. on p. 8).

[60]S. W. Hawking, "The Beginning of the Universe", in *Primordial Nucleosynthesis and Evolution of Early Universe* (Springer, Dordrecht, The Netherlands, 1991), pp. 129–139 (cit. on p. 8).

[61]S. W. Hawking, "Arrow of time in cosmology", Phys. Rev. D **32**, 2489–2495 (1985) (cit. on p. 8).

[62]R. Penrose, "The Big Bang and its Dark-Matter Content: Whence, Whither, and Wherefore", Found. Phys. **48**, 1177–1190 (2018) (cit. on p. 8).

[63]A. Van Orden and R. J. Saykally, "Small Carbon Clusters: Spectroscopy, Structure, and Energetics", Chem. Rev. **98**, 2313–2358 (1998) (cit. on p. 8).

[64]P. Vonlanthen, T. Rauscher, C. Winteler, D. Puy, M. Signore, and V. Dubrovich, "Chemistry of heavy elements in the Dark Ages", Astron. Astrophys. **503**, 47–59 (2009) (cit. on p. 8).

[65]F. Zwicky, "On the Thermodynamic Equilibrium in the Universe", Proc. Natl. Acad. Sci. U.S.A. **14**, 592–597 (1928) (cit. on p. 8).

[66]L. Spitzer Jr., "The Dynamics of the Interstellar Medium. I. Local Equilibrium.", Astrophys. J. **93**, 369 (1941) (cit. on p. 8).

[67]L. Spitzer Jr., "The Dynamics of the Interstellar Medium. II. Radiation Pressure.", Astrophys. J. **94**, 232 (1941) (cit. on p. 8).

[68]L. Spitzer Jr., "The Dynamics of the Interstellar Medium. III. Galactic Distribution.", Astrophys. J. **95**, 329 (1942) (cit. on p. 8).

[69]C. Puzzarini, J. Bloino, N. Tasinato, and V. Barone, "Accuracy and Interpretability: The Devil and the Holy Grail. New Routes across Old Boundaries in Computational Spectroscopy", Chem. Rev. **119**, 8131–8191 (2019) (cit. on pp. 9, 21, 40, 58).

[70]A. D. Buckingham, "Theory of long-range dispersion forces", Discuss. Faraday Soc. **40**, 232–238 (1965) (cit. on p. 9).

[71]P. M. Morse and H. Feshbach, *Methods of Theoretical Physics* (McGraw-Hill, Maidenhead, England, UK, 1953) (cit. on p. 9).

[72]E. Gerjuoy and S. Stein, "Rotational Excitation by Slow Electrons", Phys. Rev. **97**, 1671–1679 (1955) (cit. on p. 9).

[73]J. M. Hutson and C. R. Le Sueur, "MOLSCAT: A program for non-reactive quantum scattering calculations on atomic and molecular collisions", Comput. Phys. Commun. **241**, 9–18 (2019) (cit. on pp. 9, 64).

[74]L. D. Carr, D. DeMille, R. V. Krems, and J. Ye, "Cold and ultracold molecules: science, technology and applications", New J. Phys. **11**, 055049 (2009) (cit. on p. 9).

[75]K. H. Walter, "The chemistry of the interstellar medium", Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences **325**, 405–421 (1988) (cit. on p. 9).

[76]G. W. F. Drake, *Springer Handbook of Atomic, Molecular, and Optical Physics* (Springer, New York, NY, USA, Feb. 2007) (cit. on p. 9).

[77]F. Lique, P. Honvault, and A. Faure, "Ortho–para-H_2 conversion processes in astrophysical media", Int. Rev. Phys. Chem. **33**, 125–149 (2014) (cit. on p. 10).

[78]N. Balakrishnan, "Perspective: Ultracold molecules and the dawn of cold controlled chemistry", J. Chem. Phys. **145**, 10.1063/1.4964096 (2016) (cit. on pp. 11, 12).

[79]R. V. Krems, "Cold controlled chemistry", Phys. Chem. Chem. Phys. **10**, 4079–4092 (2008) (cit. on p. 11).

[80]G. Qumner and P. S. Julienne, "Ultracold Molecules under Control!", Chem. Rev. **112**, 4949–5011 (2012) (cit. on p. 11).

[81]R. V. Krems, "Ultracold controlled chemistry", Physics **3** (2010) (cit. on p. 11).

[82]D. A. Williams and E. Herbst, "It's a dusty Universe: surface science in space", Surf. Sci. **500**, 823–837 (2002) (cit. on p. 12).

[83]A. Kushwaha, S. Chhabra, and T. J. Dhilip Kumar, "Interaction of cyanogen (NCCN) with proton: A new *ab initio* potential energy surface", Chem. Phys. Lett. **761**, 138013 (2020) (cit. on p. 12).

[84]P. Chahal and T. J. Dhilip Kumar, "Non-adiabatic interactions in $H^+ + C_3$ system: An *ab initio* study", Chem. Phys. **571**, 111941 (2023) (cit. on p. 12).

[85]B. Mandal, K. Patkowski, P. G. Jambrina, F. J. Aoiz, and N. Balakrishnan, "Stereodynamics of cold HD and D2 collisions with He", J. Chem. Phys. **162**, 10.1063/5.0250522 (2025) (cit. on p. 12).

[86]B. Yang, P. Zhang, X. Wang, P. C. Stancil, J. M. Bowman, N. Balakrishnan, and R. C. Forrey, "Quantum dynamics of CO–H2 in full dimensionality", Nat. Commun. **6**, 1–8 (2015) (cit. on p. 12).

[87]Q. Yao, M. Morita, C. Xie, N. Balakrishnan, and H. Guo, "Globally Accurate Full-Dimensional Potential Energy Surface for H2 + HCl Inelastic Scattering", J. Phys. Chem. A **123**, 6578–6586 (2019) (cit. on p. 12).

[88]E. Roueff and F. Lique, "Molecular Excitation in the Interstellar Medium: Recent Advances in Collisional, Radiative, and Chemical Processes", Chem. Rev. **113**, 8906–8938 (2013) (cit. on p. 12).

[89]V. I. Balykin, V. G. Minogin, and V. S. Letokhov, "Electromagnetic trapping of cold atoms", Rep. Prog. Phys. **63**, 1429 (2000) (cit. on p. 12).

[90]G. Qumner and P. S. Julienne, "Ultracold Molecules under Control!", Chem. Rev. **112**, 4949–5011 (2012) (cit. on p. 12).

[91]E. P. Wigner, "On the Behavior of Cross Sections Near Thresholds", Phys. Rev. **73**, 1002–1009 (1948) (cit. on pp. 12, 64, 67).

[92]J. Prodan, A. Migdall, W. D. Phillips, I. So, H. Metcalf, and J. Dalibard, "Stopping Atoms with Laser Light", Phys. Rev. Lett. **54**, 992–995 (1985) (cit. on p. 12).

[93]N. Balakrishnan, C. Kalyanaraman, and N. Sathyamurthy, "Time-dependent quantum mechanical approach to reactive scattering and related processes", Phys. Rep. **280**, 79–144 (1997) (cit. on p. 16).

[94]N. Sathyamurthy and S. Mahapatra, "Time-dependent quantum mechanical wave packet dynamics", Phys. Chem. Chem. Phys. **23**, 7586–7614 (2021) (cit. on p. 16).

[95]M. Born, "Quantum mechanics of collision processes", Uspekhi Fizich **1926**, 456 (1926) (cit. on p. 16).

[96]P. A. M. Dirac, "THE MATHEMATICAL FOUNDATIONS OF QUANTUM THEORY", in *Mathematical Foundations of Quantum Theory* (Academic Press, Cambridge, MA, USA, Jan. 1978), pp. 1–8 (cit. on p. 17).

[97]P. A. M. Dirac, "The fundamental equations of quantum mechanics", Proc. R. Soc. London A **109**, 642–653 (1925) (cit. on p. 17).

[98]P. A. M. Dirac, "The quantum theory of the electron", Proc. R. Soc. London A **117**, 610–624 (1928) (cit. on p. 17).

[99]P. A. M. Dirac, "Quantum mechanics of many-electron systems", Proc. R. Soc. London A **123**, 714–733 (1929) (cit. on p. 17).

[100]J. C. Slater, "The Virial and Molecular Structure", J. Chem. Phys. **1**, 687–691 (1933) (cit. on p. 18).

[101]J. C. Slater, "A Simplification of the Hartree-Fock Method", Phys. Rev. **81**, 385–390 (1951) (cit. on pp. 18, 19).

[102]R. F. Borkman and R. G. Parr, "Toward an Understanding of Potential-Energy Functions for Diatomic Molecules", J. Chem. Phys. **48**, 1116–1126 (1968) (cit. on p. 18).

[103]B. S. Francis, "The integral formulae for the variational solution of the molecular many-electron wave equation in terms of Gaussian functions with direct electronic correlation", Proc. R. Soc. London A - Math. Phys. Sci. **258**, 402–411 (1960) (cit. on p. 18).

[104]K. Singer, "The use of Gaussian (exponential quadratic) wave functions in molecular problems - I. General formulae for the evaluation of integrals", Proc. R. Soc. London A - Math. Phys. Sci. **258**, 412–420 (1960) (cit. on p. 18).

[105]A. LANDÉ, "The physical significance of the reciprocal lattice of crystals", American Scientist **37**, 414–416 (1949) (cit. on p. 18).

[106]C. Puzzarini, "Extrapolation to the Complete Basis Set Limit of Structural Parameters: Comparison of Different Approaches", J. Phys. Chem. A **113**, 14530–14535 (2009) (cit. on pp. 18, 24).

[107]J. A. Pople and D. L. Beveridge, "Molecular orbital theory", C0., NY (1970) (cit. on p. 18).

[108]T. H. Dunning, "Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen", J. Chem. Phys. **90**, 1007–1023 (1989) (cit. on p. 18).

[109]W. J. Hehre, R. Ditchfield, and J. A. Pople, "Self—Consistent Molecular Orbital Methods. XII. Further Extensions of Gaussian—Type Basis Sets for Use in Molecular Orbital Studies of Organic Molecules", J. Chem. Phys. **56**, 2257–2261 (1972) (cit. on p. 19).

[110]D. L. Strout and G. E. Scuseria, "A quantitative study of the scaling properties of the Hartree–Fock method", J. Chem. Phys. **102**, 8448–8452 (1995) (cit. on p. 19).

[111]F. Weigend, "A fully direct RI-HF algorithm: Implementation, optimised auxiliary basis sets, demonstration of accuracy and efficiency", Phys. Chem. Chem. Phys. **4**, 4285–4291 (2002) (cit. on p. 19).

[112]W. Kohn, A. D. Becke, and R. G. Parr, "Density Functional Theory of Electronic Structure", J. Phys. Chem. **100**, 12974–12980 (1996) (cit. on p. 19).

[113]M. L. Leininger, W. D. Allen, H. F. Schaefer, and C. D. Sherrill, "Is Mo/ller–Plesset perturbation theory a convergent ab initio method?", J. Chem. Phys. **112**, 9213–9222 (2000) (cit. on p. 19).

[114]C. David Sherrill and H. F. Schaefer, "The Configuration Interaction Method: Advances in Highly Correlated Approaches", in *Advances in Quantum Chemistry*, Vol. 34 (Academic Press, Cambridge, MA, USA, Jan. 1999), pp. 143–269 (cit. on p. 20).

[115]P. J. Knowles and N. C. Handy, "A new determinant-based full configuration interaction method", Chem. Phys. Lett. **111**, 315–321 (1984) (cit. on p. 20).

[116]G. D. Purvis and R. J. Bartlett, "A full coupled-cluster singles and doubles model: The inclusion of disconnected triples", J. Chem. Phys. **76**, 1910–1918 (1982) (cit. on p. 20).

[117]G. E. Scuseria, A. C. Scheiner, T. J. Lee, J. E. Rice, and H. F. Schaefer, "The closed-shell coupled cluster single and double excitation (CCSD) model for the description of electron correlation. A comparison with configuration interaction (CISD) results", J. Chem. Phys. **86**, 2881–2890 (1987) (cit. on p. 20).

[118]G. E. Scuseria and H. F. Schaefer, "A new implementation of the full CCSDT model for molecular electronic structure", Chem. Phys. Lett. **152**, 382–386 (1988) (cit. on p. 20).

[119]R. J. Bartlett and M. Musiał, "Coupled-cluster theory in quantum chemistry", Rev. Mod. Phys. **79**, 291–352 (2007) (cit. on p. 20).

[120]G. E. Scuseria, C. L. Janssen, and H. F. Schaefer, "An efficient reformulation of the closed-shell coupled cluster single and double excitation (CCSD) equations", J. Chem. Phys. **89**, 7382–7387 (1988) (cit. on p. 20).

[121]J. A. Gomez, T. M. Henderson, and G. E. Scuseria, "Recoupling the singlet- and triplet-pairing channels in single-reference coupled cluster theory", J. Chem. Phys. **145**, 10.1063/1.4963870 (2016) (cit. on p. 20).

[122]C. A. Jimnez-Hoyos and G. E. Scuseria, "Cluster-based mean-field and perturbative description of strongly correlated fermion systems: Application to the one- and two-dimensional Hubbard model", Phys. Rev. B **92**, 085101 (2015) (cit. on p. 20).

[123]H.-J. Werner and P. J. Knowles, "An efficient internally contracted multiconfiguration–reference configuration interaction method", J. Chem. Phys. **89**, 5803–5814 (1988) (cit. on p. 20).

[124]T. B. Adler, G. Knizia, and H.-J. Werner, "A simple and efficient CCSD(T)-F12 approximation", J. Chem. Phys. **127**, 221106 (2007) (cit. on pp. 21, 26).

[125]S. Boys and F. Bernardi, "The calculation of small molecular interactions by the differences of separate total energies. Some procedures with reduced errors", Mol. Phys. **19**, 553–566 (1970) (cit. on p. 24).

[126]M. Kllay, R. A. Horvth, L. Gyevi-Nagy, and P. R. Nagy, "Size-consistent explicitly correlated triple excitation correction", J. Chem. Phys. **155**, 034107 (2021) (cit. on p. 25).

[127]T. Shiozaki and H.-J. Werner, "Multireference explicitly correlated F12 theories", Mol. Phys. (2013) (cit. on p. 25).

[128]E. Quintas-Snchez and R. Dawes, "AUTOSURF: A Freely Available Program To Construct Potential Energy Surfaces", J. Chem. Inf. Model. **59**, 262–271 (2019) (cit. on p. 27).

[129]H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, and M. Schtz, "Molpro: a general-purpose quantum chemistry program package", Wiley Interdiscip. Rev. Comput. Mol. Sci. **2**, 242–253 (2012) (cit. on p. 27).

[130]M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox, *Gaussian 16 Revision B.01*, Gaussian Inc. Wallingford CT, 2016 (cit. on p. 27).

[131]R. Dennington, T. A. Keith, and J. M. Millam, *Gaussview Version 6*, Semichem Inc. Shawnee Mission KS, 2016 (cit. on p. 27).

[132]A. J. C. Varandas, "Energy switching approach to potential surfaces: An accurate single-valued function for the water molecule", J. Chem. Phys. **105**, 3524–3531 (1996) (cit. on p. 28).

[133]H. M. Hulburt and J. O. Hirschfelder, "Potential Energy Functions for Diatomic Molecules", J. Chem. Phys. **9**, 61–69 (1941) (cit. on p. 28).

[134]Y. P. Varshni, "Comparative Study of Potential Energy Functions for Diatomic Molecules", Rev. Mod. Phys. **29**, 664–682 (1957) (cit. on p. 28).

[135]S. Thareja and N. Sathyamurthy, "Fitting repulsive potential-energy curves and surfaces", J. Chem. Soc., Faraday Trans. 2 **81**, 717–723 (1985) (cit. on p. 32).

[136]S. K. Upadhyay and N. Sathyamurthy, "The challenge of fitting AB initio surfaces. A test of the utility of akima's bivariate interpolation method to dynamical studies", Chem. Phys. Lett. **92**, 631–636 (1982) (cit. on p. 33).

[137]C. P. Shukla, A. K. Bachhawat, and N. Sathyamurthy, "The challenge of fitting ab initio surfaces.I. Rigid-rotor xn–CO2H2-rz10a potential", Chem. Phys. **70**, 83–91 (1982) (cit. on p. 33).

[138]A. Kushwaha and T. J. Dhilip Kumar, "Benchmarking PES-Learn's machine learning models predicting accurate potential energy surface for quantum scattering", Int. J. Quantum Chem. **123**, e27007 (2023) (cit. on pp. 34, 45, 64).

[139]N. Sathyamurthy and L. M. Raff, "Quasiclassical trajectory studies using 3D spline interpolation of *ab initio* surfaces", J. Chem. Phys. **63**, 464–473 (1975) (cit. on p. 33).

[140]N. Balakrishnan, G. Qumner, R. C. Forrey, R. J. Hinde, and P. C. Stancil, "Full-dimensional quantum dynamics calculations of H_2–H_2 collisions", J. Chem. Phys. **134**, 014301 (2011) (cit. on p. 36).

[141]J. Gasteiger and J. Zupan, "Neural Networks in Chemistry", Angew. Chem., Int. Ed. Engl. **32**, 503–527 (1993) (cit. on p. 39).

[142]M. Meuwly, "Machine Learning for Chemical Reactions", Chem. Rev. **121**, 10218–10239 (2021) (cit. on p. 39).

[143]F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.", Psychol. Rev. **65**, 386 (1958) (cit. on pp. 39, 40, 43).

[144]C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, Adaptive Computation and Machine Learning series (MIT Press, 2005) (cit. on pp. 39, 40, 44).

[145]J. A. Keith, V. Vassilev-Galindo, B. Cheng, S. Chmiela, M. Gastegger, K.-R. Müller, and A. Tkatchenko, "Combining machine learning and computational chemistry for predictive insights into chemical systems", Chem. Rev. **121**, 9816–9872 (2021) (cit. on p. 40).

[146]A. Jasinski, J. Montaner, R. C. Forrey, B. H. Yang, P. C. Stancil, N. Balakrishnan, J. Dai, R. A. Vargas-Hernndez, and R. V. Krems, "Machine learning corrected quantum dynamics calculations", Phys. Rev. Res. **2**, 032051 (2020) (cit. on p. 40).

[147]A. S. Abbott, J. M. Turney, B. Zhang, D. G. A. Smith, D. Altarawy, and H. F. I. Schaefer, "PES-Learn: An Open-Source Software Package for the Automated Generation of Machine Learning Models of Molecular Potential Energy Surfaces", J. Chem. Theory Comput. **15**, 4386–4398 (2019) (cit. on pp. 40, 45).

[148]J. Behler, "Atom-centered symmetry functions for constructing high-dimensional neural network potentials", J. Chem. Phys. **134**, 074106 (2011) (cit. on p. 40).

[149]J. Behler, "Representing potential energy surfaces by high-dimensional neural network potentials", J. Phys.: Condens. Matter **26**, 183001 (2014) (cit. on p. 40).

[150]B. Jiang, J. Li, and H. Guo, "Potential energy surfaces from high fidelity fitting of *ab initio* points: the permutation invariant polynomial-neural network approach", Int. Rev. Phys. Chem. **35**, 479–506 (2016) (cit. on p. 40).

[151]K. Shao, J. Chen, Z. Zhao, and D. H. Zhang, "Communication: Fitting potential energy surfaces with fundamental invariant neural network", J. Chem. Phys. **145**, 10.1063/1.4961454 (2016) (cit. on p. 40).

[152]P. O. Dral, A. Owens, S. N. Yurchenko, and W. Thiel, "Structure-based sampling and self-correcting machine learning for accurate calculations of potential energy surfaces and vibrational levels", J. Chem. Phys. **146**, 244108 (2017) (cit. on p. 40).

[153]J. Cui and R. V. Krems, "Efficient non-parametric fitting of potential energy surfaces for polyatomic molecules with Gaussian processes", J. Phys. B: At. Mol. Opt. Phys. **49**, 224001 (2016) (cit. on p. 40).

[154]B. Kolb, P. Marshall, B. Zhao, B. Jiang, and H. Guo, "Representing Global Reactive Potential Energy Surfaces Using Gaussian Processes", J. Phys. Chem. A **121**, 2552–2557 (2017) (cit. on pp. 40, 44).

[155]C. Qu, Q. Yu, B. L. Van Hoozen Jr., J. M. Bowman, and R. A. Vargas-Hernndez, "Assessing Gaussian Process Regression and Permutationally Invariant Polynomial Approaches To Represent High-Dimensional Potential Energy Surfaces", J. Chem. Theory Comput. **14**, 3381–3396 (2018) (cit. on p. 40).

[156]Y. Guan, S. Yang, and D. H. Zhang, "Construction of reactive potential energy surfaces with gaussian process regression: active data selection", Mol. Phys. **116**, 823–834 (2018) (cit. on p. 40).

[157]A. Kamath, R. A. Vargas-Hernndez, R. V. Krems, T. Carrington, and S. Manzhos, "Neural networks vs Gaussian process regression for representing potential energy surfaces: A comparative study of fit quality and vibrational spectrum accuracy", J. Chem. Phys. **148**, 241702 (2018) (cit. on p. 40).

[158]J. Li, K. Song, and J. Behler, "A critical comparison of neural network potentials for molecular reaction dynamics with exact permutation symmetry", Phys. Chem. Chem. Phys. **21**, 9672–9682 (2019) (cit. on p. 40).

[159]A. Kushwaha and T. J. Dhilip Kumar, "4D potential energy surface of NCCN–H_2 collision: Rotational dynamics by p-H_2 and o-H_2 at interstellar temperatures", J. Chem. Phys. **159**, 074304 (2023) (cit. on p. 42).

[160]R. Reed and R. J. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks* (MIT Press, 1999) (cit. on p. 43).

[161]R. V. Krems, "Bayesian machine learning for quantum molecular dynamics", Phys. Chem. Chem. Phys. **21**, 13392–13410 (2019) (cit. on p. 45).

[162]O. B. Sheynin, "Laplace's theory of errors", Arch. Hist. Exact Sci. **17**, 1–61 (1977) (cit. on p. 45).

[163]J. Ahlberg, E. Nilson, and J. Walsh, *The Theory of Splines and Their Applications*, Mathematics in Science and Engineering (Elsevier, 1967) (cit. on p. 45).

[164]R. Biswas, F. A. Gianturco, K. Giri, L. Gonzlez-Snchez, U. Lourderaj, N. Sathyamurthy, and E. Yurtsever, "An improved artificial neural network fit of the *ab initio* potential energy surface points for $HeH^+ + H_2$ and its ensuing rigid rotors quantum dynamics", Artificial Intelligence Chemistry **1**, 100017 (2023) (cit. on p. 46).

[165]J. Dai and R. V. Krems, "Neural network Gaussian processes as efficient models of potential energy surfaces for polyatomic molecules", Mach. Learn.: Sci. Technol. **4**, 045027 (2023) (cit. on p. 46).

[166]F. Jensen, *Introduction to Computational Chemistry* (Wiley, Chichester, England, UK, Feb. 2017) (cit. on p. 58).

[167]P. R. P. Barreto, A. C. P. S. Cruz, H. O. Euclides, A. F. Albernaz, and E. Correa, "Spherical harmonics representation of the potential energy surface for the H2···H2 van der Waals complex", J. Mol. Model. **26**, 1–8 (2020) (cit. on p. 58).

[168]S. Green, "Rotational excitation in H_2–H_2 collisions: Close-coupling calculations", J. Chem. Phys. **62**, 2271–2277 (1975) (cit. on p. 60).

[169]N. Sathyamurthy, "Computational fitting of *ab initio* potential energy surfaces", Comput. Phys. Rep. **3**, 1–69 (1985) (cit. on p. 61).

[170]A. R. Edmonds, *Angular Momentum in Quantum Mechanics*, Investigations in Physics (Princeton University Press, 2016) (cit. on p. 61).

[171]E. U. Condon and P. M. Morse, "Quantum Mechanics of Collision Processes I. scattering of particles in a definite force field", Rev. Mod. Phys. **3**, 43–88 (1931) (cit. on p. 64).

[172]P. M. Morse, "Quantum Mechanics of Collision Processes Part II", Rev. Mod. Phys. **4**, 577–634 (1932) (cit. on p. 64).

[173]A. Arthurs and A. Dalgarno, "The theory of scattering by a rigid rotator", Proc. R. Soc. Lond., A Math. Phys. Sci. **256**, 540–551 (1960) (cit. on p. 64).

[174]M. H. Alexander and D. E. Manolopoulos, "A stable linear reference potential algorithm for solution of the quantum close-coupled equations in molecular scattering theory", J. Chem. Phys. **86**, 2044–2050 (1987) (cit. on p. 64).

[175]D. E. Manolopoulos and S. K. Gray, "Symplectic integrators for the multichannel Schrdinger equation", J. Chem. Phys. **102**, 9214–9227 (1995) (cit. on p. 64).

[176]J. H. Van Vleck and V. F. Weisskopf, "On the Shape of Collision-Broadened Lines", Rev. Mod. Phys. **17**, 227–236 (1945) (cit. on p. 66).

[177]J. R. Luyten, "Colisions and equilibrium", Ann. Phys. **77**, 354–379 (1973) (cit. on p. 66).

[178]J. S. Rowlinson, "The Maxwell–Boltzmann distribution", Mol. Phys. (2005) (cit. on p. 66).

[179]N. Balakrishnan, R. C. Forrey, and A. Dalgarno, "Threshold phenomena in ultracold atom–molecule collisions", Chem. Phys. Lett. **280**, 1–4 (1997) (cit. on p. 67).

[180]Y. L. Shirley, "The Critical Density and the Effective Excitation Density of Commonly Observed Molecular Dense Gas Tracers", Publ. Astron. Soc. Pac. **127**, 299 (2015) (cit. on pp. 67, 162).

[181]J. G. Mangum and Y. L. Shirley, "How to Calculate Molecular Column Density", Publ. Astron. Soc. Pac. **127**, 266 (2015) (cit. on pp. 67, 162).

[182]L. Spitzer Jr., "The Temperature of Interstellar Matter. I.", Astrophys. J. **107**, 6 (1948) (cit. on p. 162).

[183]L. Spitzer Jr., "The Temperature of Interstellar Matter. II.", Astrophys. J. **109**, 337 (1949) (cit. on p. 162).

[184]L. Spitzer Jr. and M. P. Savedoff, "The Temperature of Interstellar Matter. III.", Astrophys. J. **111**, 593 (1950) (cit. on p. 162).

[185]D. R. Bates and L. Spitzer Jr., "The Density of Molecules in Interstellar Space.", Astrophys. J. **113**, 441 (1951) (cit. on p. 162).

[186]F. F. S. van der Tak, J. H. Black, F. L. Schier, D. J. Jansen, and E. F. van Dishoeck, "A computer program for fast non-LTE analysis of interstellar line spectra - With diagnostic plots to interpret observed line intensity ratios", Astron. Astrophys. **468**, 627–635 (2007) (cit. on p. 162).

Appendix A - Python: Introduction and Basics

Python was given its name by its creator, Van Rossum, after Monty Python (their movies are a must-watch - surreal and comedic). He wanted the programming to be, most importantly, fun, and if you're using Python for the first time, remember:

'As theoreticians, your code must work as intended and should execute within a reasonable time frame. It doesn't need to be the best out there. With time, it will get better!'

A.1 Anaconda Distribution

A.1.1 What is Anaconda?

The first step towards running Python is to install the Python compiler needed to convert the human-readable lines of code to machine-understandable language. The Anaconda software is a simple solution to get started with Python programming. The software contains (a) a simple editor (**Atom**), (b) an advanced editor with compilation ability and variable explorer (**Spyder**), and (c) an interactive Python editor (**Jupyter-Notebook**).

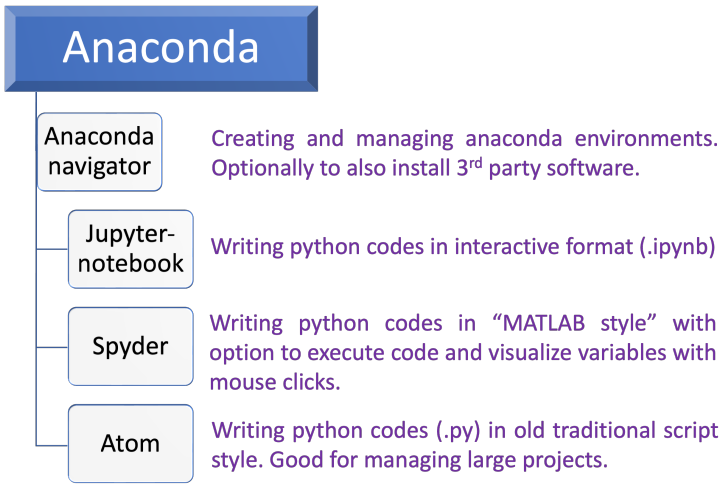


Figure A.1. A summary of software provided by Anaconda for editing and compiling Python codes.

Apart from Python distribution, Anaconda is also a package/environment manager that contains a standard collection of many preinstalled open-source packages. Off the shelf, users can use many useful packages like **NumPy** (advanced mathematical library for multi-dimensional arrays and numerical analysis), **Pandas** (2D data structure management), **SciPy** (advanced scientific package), **SymPy** (symbolic python library for tensor calculus), etc.

Additional libraries can be installed in different environments using Anaconda. This avoids conflicts among incompatible libraries and maintains a stable environment for any program/code. E.g., separate versions of libraries compatible with an old version of Python (version 2.7) and newer Python (version 3.9) can be separately installed by creating two environments, i.e., '*py*2_7' and '*py*3_9'. The Anaconda takes care of finding and installing compatible packages for any specific Python version.

A.1.2 How to Install Anaconda?

Use the following steps depending on the Windows/Linux OS. The PES2MP has been written considering Linux-style folder paths and will not work on Windows (unless the folder paths are changed to Windows format in the main and driver files.)

Windows

- **Step1:** Go to Anaconda website and then download for windows.
- **Step2:** Locate the setup file (Anaconda.exe) and install it.
- **Step3:** After accepting the terms of the agreement, install with the recommended options. Now you can run the Anaconda environment and install any packages like Jupyter Notebook, Spyder, etc.

Linux

- **Step1:** Go to Anaconda website and then download for Linux.
- **Step2:** Go to the download folder and open the terminal in that folder.
- **Step3:** **(Optional)** To check for any security and integrity issues in the file, run the hash encryption check, using the command: `sha256sum` *`filename`*
  e.g.: `sha256sum Anaconda_3_10_XX.sh`
  Example of output hashes: `fbadbfae5992a8c96af0a4621262080eea44e22baee2172e3dfb640f5cf8d22d`
  Then you will generate an output called hashes for the particular version of the downloaded file, which needs to be matched with the hashes available on the Anaconda documentation.
- **Step4:** Run the following command to complete the installation: `bash` *`filename`*
  e.g.: `bash Anaconda_3_10_XX.sh`

A.1.3 Anaconda as Package Manager

Anaconda can install, update, and remove packages.

Install Packages

Open a command prompt/anaconda prompt(Windows) or a terminal (Linux) and run the command below-
`conda install` *`package_name`* e.g.: `conda install numpy`

Uninstall packages

Run the command below to uninstall packages.
`conda uninstall` *`package_name`* e.g.: `conda uninstall numpy`

Update packages

Run the command below to update a package.
`conda update` *`package_name`* e.g.: `conda update numpy` .

Conda:

The command conda checks for the Python version for an environment and accesses conflicts with already

installed packages to find the most suitable version for the intended package. The **pip** command, on the other hand, does not take into account these conflicts and can therefore lead to problems when the packages are imported or libraries are executed. However, with time, the packages are becoming inter-compatible.

A.1.4 Anaconda as Environment Manager

If I am using two Python versions (3.6 and 3.9), upgrading the package to one version may affect the other version. Some packages rely on older versions for their functioning. For example, in my previous project, the PES-Learn software runs on Python 3.6. If we upgrade the Python version, then this software will not run due to conflicts with newer packages. So we always create a virtual environment using the Anaconda environment manager, which will be Python version 3.6, and we can use PES-Learn without worrying about conflicts. We shall name these environments such that we can remember what software they will be used for, followed by a number that may indicate an old/new version.

This way we can easily switch between different versions of Python.

Create a new environment

Run the following command to create a new Python environment.

```
conda create --name Peslearn python=3.6
```

and then activate using the command-

```
conda activate Peslearn
```

Note:*The package installed in one environment will not affect packages present in the other environments.*

Example: Installing PES-Learn using Anaconda Environment

After creating and activating the Python 3.6 environment, install the PES-Learn package using the following command:

```
git clone https://github.com/adabbot/PES-Learn.git
cd PES-Learn
python setup.py install
pip install -e
```

Note: run this command in the same folder as the setup.py file.

Note: If git is not installed, then install it using the command:

```
conda install git
```

Example: A simple Python code generating 2D PES

Below is an example file that creates input files for Gaussian. The files, when executed, will generate PES for PN collision with He at $\theta = 0°$. The initial value of R is 2 Å and the step size is 0.1 Å. In total, 100 files

will be created (**i in range (0,101)**). Try running it as an example using Spyder/jupyter-notebook. (Note: The base (default) environment will have NumPy and will automatically run the script. Alternatively, use **conda install numpy** after creating a test environment, and the code will run.)

```
import numpy as np
theta = 0                                   # angle in degrees
R_ini = 2                                   # R value starts from 2 Angstroms
for i in range(0,101):                      # loop for generating R points
    R = i/10 + R_ini                        # R value increment 0.1
    x = np.sin(np.radians(theta))           # sin projection of He
    y = np.cos(np.radians(theta))           # cos projection of He
    with open('{}.com'.format(i),'w') as z: # generating input file
        z.write("""%nprocshared=8           # writing inside input file
            %mem=2GB
            # ccsd(t)/aug-cc-pvqz

            NP pes (Comment Line)

            0 1
            P               0.000     0.000    -0.4638
            N               0.000     0.000    1.0271
            He              0.000     {}       {}
            """.format(x,y))
```

Listing A.1. A simple Python code generating 2D PES

A.2 Structure of Python

Before beginning with the coding part, let's check out how a program runs on a computer.

A.2.1 What is Programming?

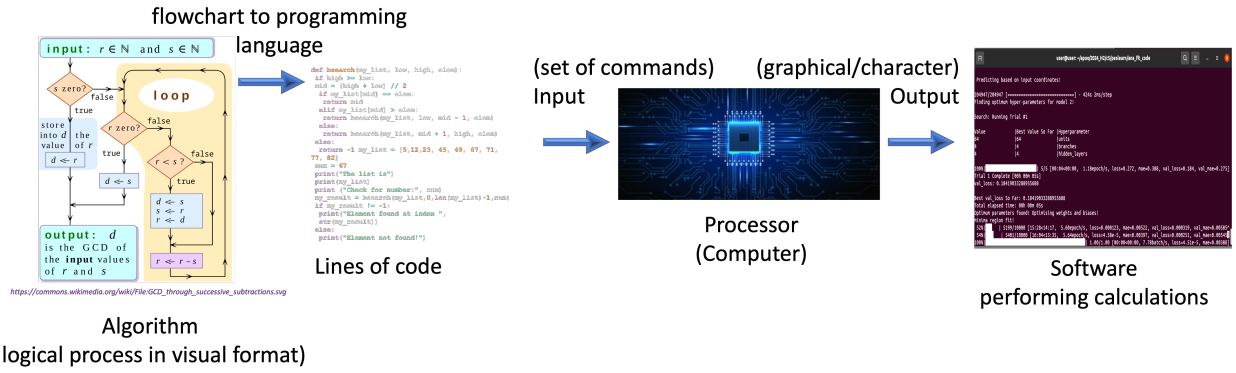


Figure A.2. How programming works!

- Programming: To create a logical set of commands that can perform any task on a computer.
- The basic flowchart describing a program is called an algorithm.
- The sets of commands are called codes.

- A compiler converts the code into a machine-readable format and sends it to a processor.
- A processor (CPU), along with RAM (keeps temporary variables) and ROM (writes output file), performs instructions on the code.
- A mature end product of code (software) can perform complex operations while being stable (handle errors).

What are programming languages???

- Low Level: Driver files | BIOS
  - Advantage: Highly optimized for direct hardware control with minimal footprint
  - Disadvantage: Complex, less portable, and non-universal (machine dependent)
- Low-Mid Level: FORTRAN 77 | C++ | JAVA
  - Disadvantage: Requires extensive knowledge of hardware and syntax.
  - Advantage: Fast and memory efficient
- High Level: Python | MATLAB
  - Disadvantage: Slower and Memory Demanding
  - Advantage: Easier syntax and universal compatibility
  - Major Advantage: Large number of libraries that can perform complex operations and are optimized to be as fast as low-level programming languages.

A.2.2 Python Basics

Type of data and their variable attributes

- Number: Integer/ Float/ Double
  - a = 5 — Integer
  - a = 5.0 — Float (default Float32)
  - a = double(5.0) — forced to be Double (64-bit precision)
- String: 'abcd' mixed with numbers and symbols
  - a = "hello world! The date is 16/10/2024 … "
- To inter-convert data types:
  - A = 5.0 — A is float with value 5.0000…. (32 bit)
  - B = int (5.0) or B = int(A) — B is integer with value 5
  - B = str(A) — B becomes a string with value '5.0'

Strings

- Putting variables inside a string

- A = int(5)
- B = ‘My age is ’.format(A) Output: My age is 5

- Special symbols
  - Inserting newline (\n) B1 = ‘ My age is 12. \n Maybe 13 ’
  - Inserting ‘ \’ B2 = ‘ My address is 12\\14 ’
  - Inserting ‘ {} ’ B3 = ‘ My address is {{12-14}} ’

Conditional Statements (if else:)

```
X = 1                # X is integer 5
if (X<10):           # ':' denotes the beginning of the if conditional statement
  print(X)           # print command : print X if x<10
  X = X+1            # updating the value of X (increment by 1)
else:                # 'else' statement: if the value of X is >= 10
  pass               # pass means do nothing
print('\n')          # new line
print('Code done!')    # Final print command
```

Listing A.2. A simple Python code with conditional statements

Extra points:

- ‘==’ checks if two (values/variables) have the same value.
- ‘>=’ checks if the value is greater than or equal to
- Do not confuse with ‘=’ which assigns value to a variable
- The code inside the conditional statement will run only if the condition is true
- # Anything written after ‘#’ is a comment line and is not a code

For loops

```
for i in range (5):          # loop will run from 0 to X-1 (default step size 1)
    print(i)                 # print statement (i)
```

Listing A.3. A simple Python code with nested loop

Python loop start from 0 and go to penultimate value (4). The value of `i` is updated in each loop (by +1). Loop runs if `i<5` resulting in 5 values `0 1 2 3 4`.

Nested loops (loop inside the loop)

```
X, Y = 2, 3                  # X = 2 and Y = 3
for i in range (X):          # outer loop will run from 0 to X-1 (1)
    for j in range (Y):      # inner loop will run from 0 to Y-1 (2)
        print(i, j)          # print statement inside two loops

Output:
(i, j) <- reference
--------------------------------------------------------------------------
0, 0     <- initial value (i,j)
0, 1     <- inner loop (Y) increment by 1
0, 2     <- inner loop (Y) increment by 1 (final value of inner loop)
1, 0     <- outer loop (X) increment by 1 (inner loop variable (Y) resets)
1, 1     <- inner loop (Y) increment by 1
1, 2     <- inner loop (Y) increment by 1 (final value of both inner & outer loop)
```

Listing A.4. A simple Python code with nested loop

A.2.3 Python Structure

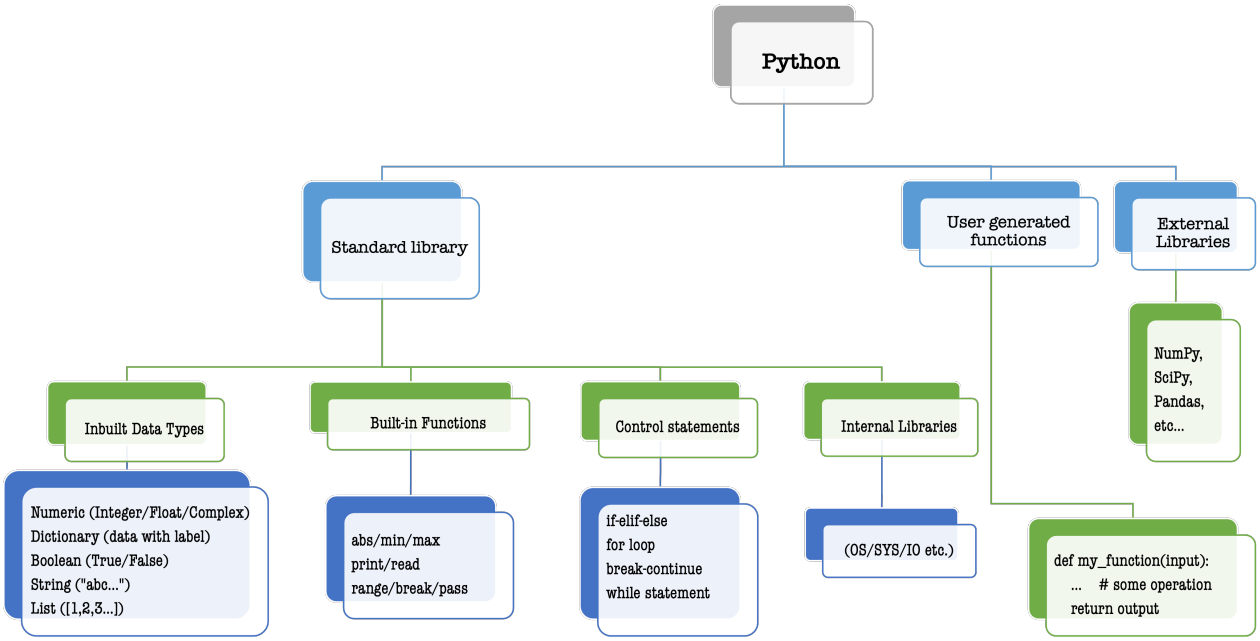


Figure A.3. Structure of Python!

Libraries

- Basic commands for importing the package:

  1. import numpy
  2. import numpy as np
  3. from numpy import zeros

- **Difference?**: E.g. To create a 1D array of length 10 and values 0 using the NumPy package 'zeros'!

  1. numpy.zeros(10)
  2. np.zeros(10)
  3. zeros(10)

- Warning: Two different libraries with the same package name can create confusion in code. E.g.

  – from XYZ import zeros
  – from ABC import zeros

  Here, Python cannot differentiate between the two packages (zeros) of ABC and XYZ.

Running a Python script

Once Anaconda and the required libraries are installed, any Python script file can be executed in the terminal using:

```
$ python3 script.py
```

Appendix B - Python: Jupyter-Notebook templates

This chapter explains how pieces of Python code can be run in 'Jupyter-Notebook' to debug or play around. Consider the Jupyter Notebook as a more versatile calculator. Here, pieces of code can be run multiple times, and output(s) for each are printed just below. The contents of this chapter would be useful in debugging any section of the PES2MP codes and in running the individual Jupyter-Notebook files provided in the PES2MP-IPYNB package in $/GitHub/apoorv-kushwaha$ and $/GitHub/QuantumDynamicsLab$. The installation of Anaconda and creation of Python environments were briefly covered in the previous Appendix, and therefore we shall see in detail with images how the same can be done using GUI.

B.1 Running Python Codes in Interactive Python Editor

B.1.1 Jupyter-Notebook

There are two methods for installing and running the Jupyter-Notebook.

1. GUI (Graphical User Interface)
   (a) Open anaconda-navigator (GUI) by:
      i. MacOS: Finding an icon for the same in Launchpad
      ii. Linux: typing `$ anaconda-navigator` in command prompt.
   (b) Install Jupyter-Notebook for any environment (default: '*base*' environment)
      i. Click on Environments to choose any specific environment (Optional)
      ii. Click on '*install*' option below jupyter-notebook icon. Once installed, the '*launch*' icon will appear.
      iii. Click on the '*launch*' icon to run Jupyter-Notebook on a local internet browser.
2. CUI (Character User Interface)
   (a) Open terminal ('base' will appear before $ after installing Anaconda)
   (b) base indicates base (default) environment. Users are encouraged to create a new environment for each program/software
   (c) Creating a new environment: `$ conda create -n pes2mp_test python=3.10 anaconda`
   (d) list environment (Optional): `$ conda env list`
   (e) Select environment: `$ conda activate pes2mp_test`

(f) Install Jupyter: `$ conda install anaconda::notebook` or, alternatively, use the code below for a quicker install (installs from Python source and not conda).

i. `$ conda install -y -c anaconda python`

ii. `$ python -m pip install jupyter`

(g) Run jupyter: `$ jupyter-notebook`

Creating a new environment and installing Jupyter Notebook in that environment (GUI)

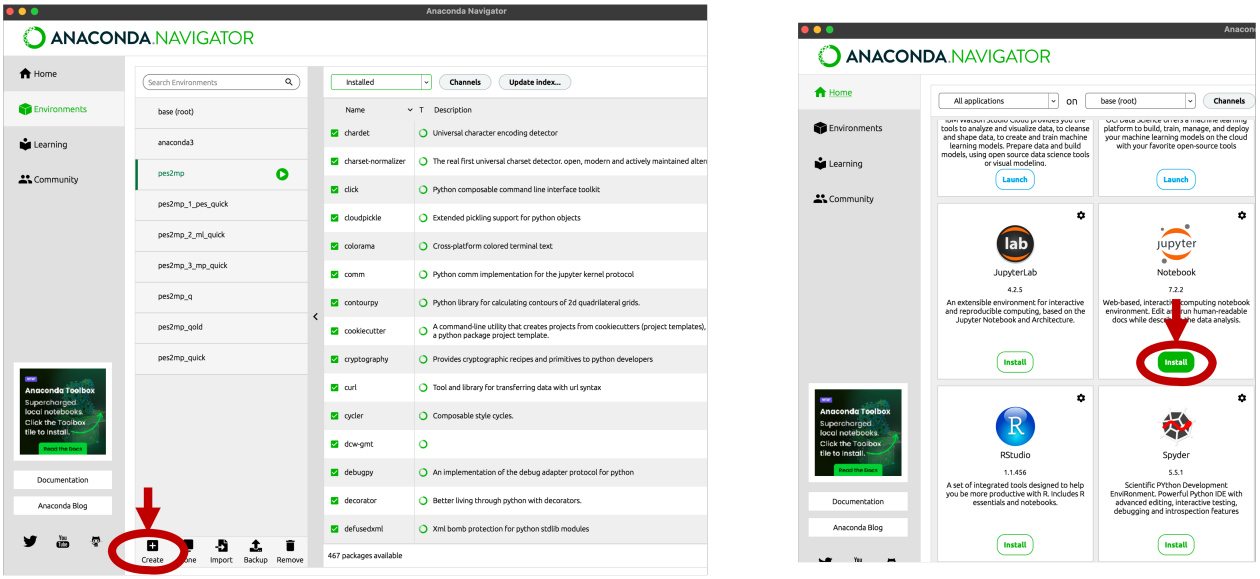


Figure B.1. Anaconda-Navigator option to (a) create new environments and (b) install Jupyter-Notebook and other packages like spyder/atom.

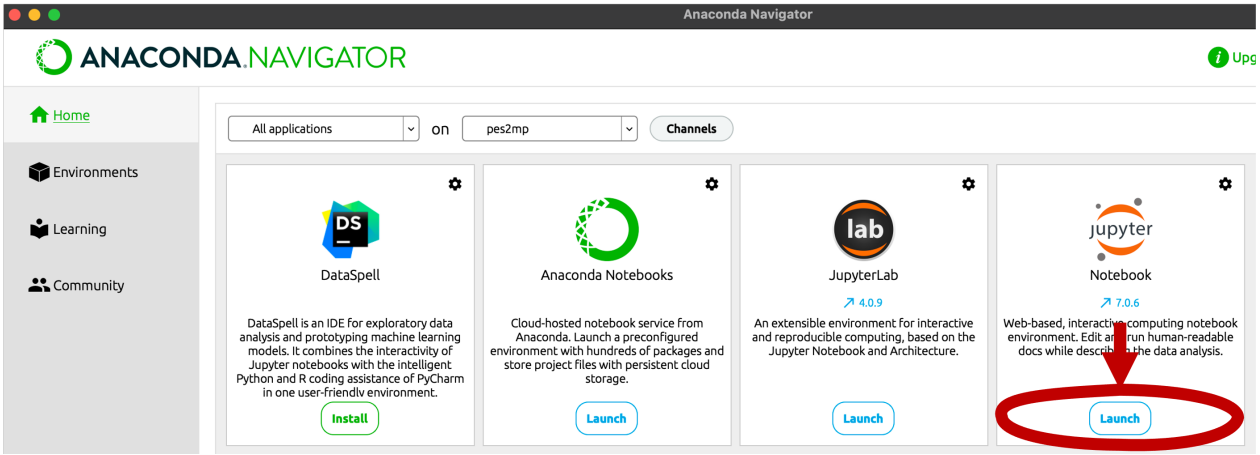


Figure B.2. Launching jupyter-notebook

Creating a new Jupyter Notebook file and running the code.

Below are some important highlights of the Jupyter Notebook to get started.

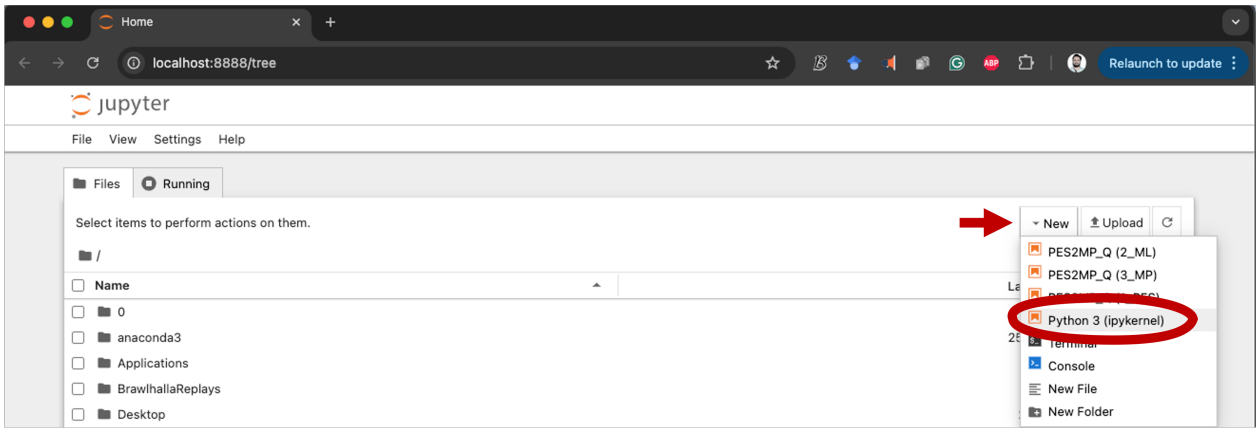
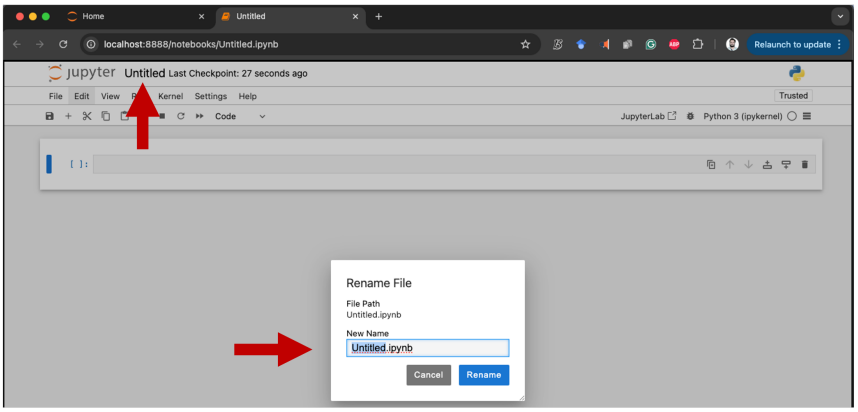


Figure B.3. Navigating to any folder and creating a new Jupyter-Notebook (.ipynb) file.



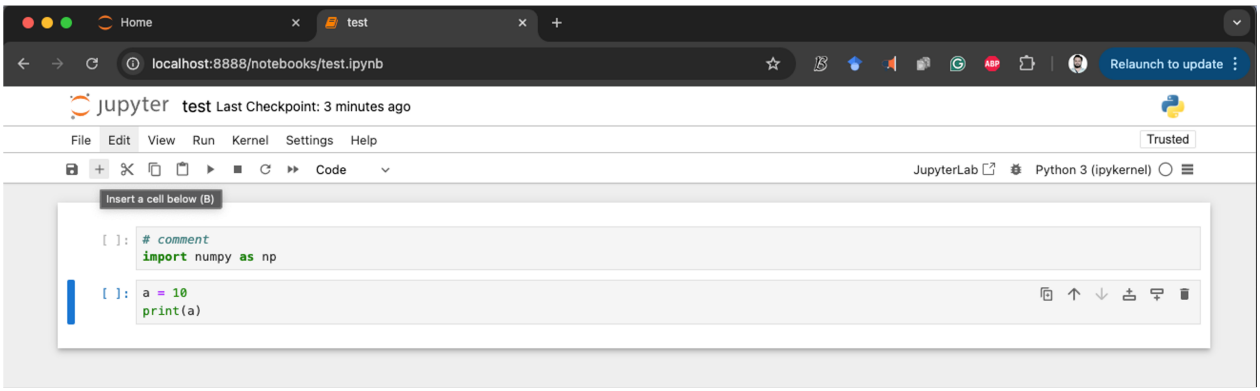


Figure B.4. Using jupyter-notebook

- *Shift + Enter*: Run the cell.
- Running a cell with just a variable name will return its attributes (properties) and value(s).

B.1.2 Google-Colab

The online version of Jupyter-Notebook. If you have a Gmail (Google) account (Sure you do!), you can directly run Google Colab with a Google search, and the interactive Python files (.ipynb) are saved into your Google Drive '*MyDrive/ColabNotebooks/*' folder. When you run any code, calculations run on one of Google's remote computers (some resources like CPU/storage/RAM) are allocated for free (for now!). This

is great if you have a basic computer/laptop and you need to run basic Python codes without the hassle of installing Anaconda and maintaining packages.

Important: Each time the Colab connects to a new remote server. Therefore, any generated/uploaded data file is lost if the server disconnects. Load and save all files using Google Drive or make sure to download the files once the calculations are done.

Installing Libraries in Google Colab

- `!pip install numpy` or
- `!pip install numpy --quiet` for silent/less verbose installation

However, to take full advantage of Python (and your overclocked/parallelized Linux personal/cluster build), Anaconda is the way to go. There is a chance that something may go wrong with time, and your environment will not work correctly (unexpected package conflicts/warnings/errors). But codes can always be rewritten to be compatible with the current version of Python with more robust libraries/environments.

B.2 Debugging code

"Programming is 1% coding and 99% debugging".

To debug a part of the PES2MP code (or any other code in general), print everything at every step. With the knowledge of output, the code can be debugged easily.
The general steps for debugging PES2MP code are:

1. activate 'PES2MP' environment
2. launch 'jupyter-notebook'
3. copy the contents of the input file into a cell
4. copy the section of the code (NNGen, PESGen,...etc.) from 'PES2MP.py' where the error exists
5. distribute the code into smaller fragments over multiple cells
6. run each cell till you encounter the error you want to debug
7. if some external functions are required, copy/paste them from 'PES2MP_driver.py'
8. change (functions calling) from `'driver.fun_x'` to simply `'fun_x'` .
9. with Jupyter-Notebook, you can modify and run the cell multiple times
10. continue the trial and error method (and check variables with variable explorer) each time
11. to understand the error(s) better, take help from the 'stack community' and/or AI assistant tools
12. continue with this approach till the error can be removed.

All the best!

Indexing