Computation Reduction through Data Re-use for Scientific Applications

By

Misha Ahmadian, B.S.C.S.

A Thesis

In

Computer Science

Submitted to the Graduate Faculty of Texas Tech University in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCES

Approved

Dr. Yong Chen
Chairperson of Committee

Dr. Yu Zhuang
Co-Chairperson of Committee

Dr. William L. Hase

Mark Sheridan
Dean of the Graduate School

December, 2015
Copyright 2015, Misha Ahmadian
ACKNOWLEDGMENTS

I would like to express my deepest gratitude to some remarkable people during my study at Texas Tech University, who had outstanding impacts on my master program study, research, and on-campus experiences.

First of all I wish to thank Prof. Bill Hase for his financial support, supervision, and guidance through all my research study at department of chemistry and biochemistry as well as giving me an extraordinary experience opportunity through my work position as a computational cluster administrator, and application developer.

I am extremely thankful to my major advisor, Dr. Yong Chen, for his advice and guidance during the course of my master studies. He did not only give me this opportunity to work with him and his research group as a researcher, but also helped and encouraged me a lot with his inspirations through all these years of my studies and work at Texas Tech University.

I am also extremely thankful for my co-chair Dr. Yu Zhuang for his all helps and supervision in my thesis. I had this opportunity to work with him in one of his research endeavors, data reduction through increased data utilization in scientific applications. Without his advice, guidance, and crucial contribution this thesis would never have been finished.

National Science Foundation, Air Force Office of Science Research, and Robert A. Welch Foundation all need a special acknowledgment here for providing funding for this research through NSF grants CHE-0957521, OISE-0730114, CNS-1338078, and CNS-1526055, and Robert A. Welch Foundation Grant D-0005.

Finally, I would like to extend my gratitude to my awesome parents, and my brother for their love and care. Their unconditional support during all these years really deserves special appreciation.
# TABLE OF CONTENTS

ACKNOWLEDGMENTS .................................................................................................................. ii

ABSTRACT ................................................................................................................................... v

TABLE OF FIGURES ...................................................................................................................... vi

1. INTRODUCTION ....................................................................................................................... 1

2. DATA MODELING TECHNIQUE AND METHODOLOGY ......................................................... 3
   2.1 Introduction ............................................................................................................................ 3
   2.2 Definition ............................................................................................................................... 3
   2.3 Methodology ......................................................................................................................... 3
   2.4 The approach ......................................................................................................................... 5
   2.5 Implementation .................................................................................................................... 7

3. CLASSICAL TRAJECTORY CHEMICAL DYNAMICS SIMULATION ................................. 8
   3.1 Introduction ........................................................................................................................... 8
   3.2 VENUS, the general classical trajectory program ............................................................... 9

4. DATA MODELING FOR CHEMICAL DYNAMICS SIMULATION ..................................... 12
   4.1 Introduction .......................................................................................................................... 12
   4.2 Apply Data Modeling approach ............................................................................................ 13
       4.2.1 Data Interpolation and Extrapolation ........................................................................ 13
       4.2.2 Choosing To-be-modeled Data ....................................................................................... 14
       4.2.3 Store to-be-modeled data in a Dynamic In-Memory Database ................................ 16
       4.2.4 Search for closest data from dataset ............................................................................. 17
       4.2.5 Data modeling in-place of expensive calculation ......................................................... 18

5. EXPERIMENTAL RESULTS FOR TWO ATOMS OF HYDROGEN (H₂) ...................... 21
   5.1 Results without Training set: ............................................................................................... 21
5.2 Experimental Training Set for $H_2$ Data Modeling in VENUS .................. 22

6. CONCLUSION AND FUTURE WORK ................................................. 27

   6.1 Conclusion .................................................................................. 27
   6.2 Future Work .............................................................................. 27

REFERENCES ...................................................................................... 30

A. DATA MODELING CODE FOR VENUS ............................................ 34
ABSTRACT

Many scientific applications consist of heavy computational and analysis workload on data, and also they often require to produce intermediate data for ongoing calculations. For instance, chemical dynamics simulations are known as heavy workload applications in terms of calculation in many aspects. There is a strong desire to seek a solution to minimize expensive calculations by replacing them with light-weight ones. VENUS is one of these chemical dynamic simulation software packages known as classical chemical dynamics simulation, with scalar executing code and heavy calculation process. In this thesis research, we introduce an innovative approximation method by storing, managing, and leveraging intermediate data (results) to speed up expensive calculations. The key idea is a newly introduced data interpolation method that leverages data points from previous calculations. The newly proposed method is a general approach that can be applied to a variety of scientific applications and disciplines. In this research, we focus on chemical dynamics simulations and the VENUS code and have developed an early prototype of the data interpolation method for reduced computations. The proposed computation reduction method through increased data re-use can increase the efficiency and productivity of scientific simulations, thus can have an impact on scientific discovery powered by high performance computing simulations.
# TABLE OF FIGURES

2.1: Replace heavy computation by Data Modeling .................................................. 5
3.1: Flow chart of VENUS Program ........................................................................ 9
4.1: Convert $\frac{\partial V}{\partial q}$ to $\frac{\partial V}{\partial r}$ ..................................................... 15
4.2: MemTable Structure ......................................................................................... 16
4.3: Interpolation Algorithm .................................................................................... 19
4.4: Flow chart of New Structure of VENUS ......................................................... 20
5.1: Potential energy calculation for two atoms of hydrogen (H2) in VENUS ........ 21
5.2: Potential energy calculation for two atoms of hydrogen (H2) in VENUS with
      Threshold 0.7 ................................................................................................. 23
5.3: Potential energy calculation for two atoms of hydrogen (H2) in VENUS with
      Threshold 0.2 ................................................................................................. 24
5.4: Potential energy calculation for two atoms of hydrogen (H2) in VENUS with
      Threshold 0.4 ................................................................................................. 24
5.5: Potential energy calculation for two atoms of hydrogen (H2) in VENUS with
      Threshold 0.2 ................................................................................................. 25
5.6: Number of Original Calculations versus Number of Interpolation calls for two
      atoms of Hydrogen H2 in 1000 step ................................................................ 26
CHAPTER 1

INTRODUCTION

Scientific applications increasingly utilize large-scale data in various fields including physics, astrophysics, climate studies, bioinformatics [1], and chemistry. In many of these science and engineering investigations, there are data that are of critical importance but highly expensive to generate, e.g. obtainable through time-consuming experimental or computational processes. Examples of highly expensive data includes those produced by heavy scientific calculations, which might take hours, days, or even weeks to compute. If such critical and expensive datasets are also very large, there is an incentive to reduce the amount of such data, since reduction of expensively computed data also means reduction of computation time. But what is critical is that the reduction of data should not lead to, within a problem-dependent threshold, the loss of the information that is carried by the original bigger set of data. Thus, for scientific applications with large computationally expensive datasets, the approach of this research is to develop a computationally efficient data modeling procedure to replace the computation of some, hopefully as many as possibly, data computed through highly expensive processes, and this data modeling procedure has to be accurate so that the model data carries almost the same scientific information as the original expensive data.

In this research we focus our approach of replacing expensive data computation by efficient procedure of data modeling in chemical dynamics simulations, where the potential energy needed in every time step of a simulation has to be computed using a highly expensive computation procedure. Potential energy computation is the most expensive part in each step of a chemical dynamics simulation, whether using empirical formulas or using quantum mechanical electronic structure theories. Parallelization of potential energy calculation is a widely adopted approach to reducing the computation time, as done in many electronic structure calculation packages including NWChem [2, 3] and in analytic empirical formulas [4]. Potential energy data calculated based on a quantum mechanical electronic structure theory are called ab initio potential energy data and they are far more computation intensive than empirical formula-based calculations.
In this paper, the ab initio data are our targets to be replaced by modeled data at as many time steps as possible while trying to maintain a good simulation accuracy.

While data modeling methods are problem dependent and require specific domain science knowledge to design a good one, this research focuses on machine learning techniques for problems where the originally expensive data have a broadly existent property. Using our techniques, we developed a method for modeling ab initio potential energy data, and implemented our data modeling method in VENUS, a chemical dynamics simulation software package. VENUS [2, 5, 6] has its own in-package suit of analytic potential energy models and is also linked with several electronic structure calculation packages, including NWChem [2, 3], for generating accurate ab initio potential energy data. In the implementation, our modeled data are to replace the ab initio data generated by NWChem.

The rest of this thesis is organized as follows. We will explain our data modeling technique and methodology in Chapter 2. The descriptions of chemical dynamics simulations and the VENUS simulation software are given in Chapter 3. The method of using modeling data to replace expensively generated data, including its application to chemical dynamics simulations, is described in Chapter 4. Experimental results are presented in Chapter 5 and Chapter 6 is the conclusion, and future work.
CHAPTER 2
DATA MODELING TECHNIQUE AND METHODOLOGY

2.1 Introduction
In this research, our strategy for reducing the costly computations for data generation is to replace as many of them as possible by data modeling. While data modeling methods do need domain specific information to achieve high modeling accuracy, there is a feature that exits widely in scientific data.

2.2 Definition
The process of learning about data is called Data Modeling, and the end result of data modeling is the Data Model. For those specific projects that require a standard defining and analyzing process of data, using Data Modeling is strongly recommended. In another words, Data Modeling techniques and methodologies are being used to manage the data resources by modeling the data in a standard, consistent, and predictable form [7, 25, 26, 27, 28].

2.3 Methodology
Many data generated in scientific or engineering processes consist of data from controllable parameters of the scientific or engineering processes, called input parameter data, and other data observed or measured as the results of processes with the controllable parameters taking those input parameter data. In many cases, the measured resulting data are continuous functions of the input parameter data. In other words, the causal relationship between the input data and the resulting data may have a relationship that is of a continuous function. Actually, even for discrete data in input-output pairs, relations resembling continuous functions also exist. For instance, many discrete optimization methods, e.g. the simulated annealing and the genetic algorithm, are based on the assumption that close discrete input data will, in high probability, lead to close output data. In this paper, we restrict our investigation only to input data and
output data related as continuous functions. Investigation on continuous function-like relation for discrete input-output data will be carried out in future research projects.

For data generated through an expensive computational or experimental process, usually the controllable parameter data incur almost no cost but the resulting output data consume a huge cost to produce. Thus, our data modeling procedure is to model the resulting data using the input parameter data as values of input variables of an unknown but to-be-modeled continuous functions. This strategy was introduced in earlier work [8, 9, 10, 11] for ab initio chemical dynamics studies, but the data modeling in [8, 9, 10, 11] used only ab initio data from the most recent time steps to model the potential data needed at the current time step. In this paper, we are investigating the possibility of using ab initio data from any earlier time steps as long as those to-be-used data are close enough to the to-be-modeled data in terms of the input parameter data. Thus, the range of data selection for the modeling is increased as compared with the modeling procedure in earlier work [8, 9, 10, 11]. With an enlarged data selection range, an ab initio data item will hence have higher chance to be re-used in more time steps, as compared with possible re-uses only for a few close time steps. Thus, increased utilization of the computation expensive data is one of the features of our new implementation strategy for the modeling-replacing-computation approach introduced in [8, 9, 10, 11]. And from the perspective of the data to be modeled, the larger data selection range makes it more likely for the modeling procedure to find close data for accurate modeling, leading to replacement of the highly expensive ab initio computation by the efficient data modeling at more time steps, and consequently resulting in a higher reduction of the computation expensive data. Thus, the new strategy we are investigating in this paper for implementing the modeling-replacing-computation approach is, in short, computation reduction through data re-using.

To increase utilization of the expensive ab initio data, all computed ab initio data have to be saved so that they can be used at later time steps in modeling. After the saved ab initio data reach a certain amount, at every time step of the dynamics simulation, the ab initio dataset is searched to decide if there are data that are close enough to produce
accurate data modeling. If it is determined that an accurate modeling is achievable, the potential energy data are modeled and the expensive ab initio calculation is not performed; otherwise, the ab initio calculation is carried out to generate potential energy data. Figure 2.1 shows the flow chart of this data modeling, and technical details of this procedure are described in following chapters.

![Flow Chart](image)

**Figure 2.1: Replace heavy computation by Data Modeling**

### 2.4 The approach

According to our methodology for data modeling of calculated data from scientific applications, there would be three essential steps besides the original procedures of software package code to let us replace the huge-cost calculations with new data modeling algorithms as many as possible:
• **Store the to-be-modeled data for future use:** The first step for this data modeling approach is finding those calculated data on each step of computation, which would be re-used in the light-calculating algorithm as an alternative to expensive calculations. Then we need to memorize these to-be-modeled data by storing them in a proper designed database for future use. This is important to know that we would not need all the data for our technique, therefore those necessary data should be selected wisely, and as little as possible.

• **Search for close enough data from database:** After storing certain number of data in database, a procedure would be called for each step of calculation to decide whether there is any close data in the database with respect to current calculated data of that step. Then those data will be retrieved from the database to be used by data modeling algorithm. This search procedure should be called on each step of calculation, after having enough data in the database and before calling any expensive calculation algorithm. In terms of fining close data from database, search algorithm should be designed precisely to let data modeling algorithm give out more accurate results, without losing any information. In some cases, applying a training set would be useful to gain more precise resulting data. Training set will be discussed in more details in chapter 4.

• **Replace highly expensive calculation with data modeling algorithm:** The final step for this approach is applying those data, which were found by search algorithm from database, in a light-calculating data modeling algorithm. In fact if there was enough close data from the database, then we can prevent an expensive original calculation algorithm from being called, and trigger light data modeling procedures from using those data. In other words, if not enough data could be found in each step, then expensive calculations should be called, and the resulting data must be stored. Since the data modeling algorithm is a sort of approximation, we do not store the output from data modeling to keep the final results more accurate.
2.5 Implementation

In this research we will discuss how we model and store intermediate calculated data into a database to re-use them in place of further heavy calculation by data learning algorithms. These algorithms after some amount of real calculation, try to search for some appropriate data out of the database to see if those data are able to be utilized in a new lighter form of calculation instead of original burdensome calculations. We assume this method could help the scientific applications to eliminate their unnecessary calculations by replacing them with new methods, and also reduce the dependency of the scalar codes by modeling calculated data.

In this thesis the goal is to apply our alternative method in one of the chemical dynamic simulations, which is strictly running sequential with expensive calculations on each trajectory integration. This method would have two prominent features:

1. Modeling data by memorizing the calculated data on each steps and re-using them in the future steps to decide if it could be possible to replace the calculation at the current step.

2. Replace some of the possible heavy calculation steps with data modes.

We expect this machine learning method reduce the workloads of heavy calculations in this chemical dynamic simulation, and also let the majority of code be independent in terms of calculation to be able to run on HPC environments. Since we use machine learning techniques for each step, an exact output as the original calculation would not be guaranteed, but giving out the closest outcome is highly desired.
CHAPTER 3

CLASSICAL TRAJECTORY CHEMICAL DYNAMICS SIMULATION

3.1 Introduction

Classical trajectory chemical dynamics simulations is what provides all processes needed to calculate the simulations including gas-surface collisions [12], energy transfer and chemical reaction in gas-phase [13], intramolecular vibrational energy distribution [8], unimolecular decomposition and conformational change [14, 15], and, intramolecular energy transfer and chemical reaction [16, 17]. For these calculations the potential energy function $V$, the potential gradient $\frac{dv}{dq}$, and in some cases the Hessian $H$, are highly required to propagate an ensemble of trajectories, and each trajectory will be determined by numerically integrating the classical equations of motions [2].

A general and practical approach to perform classical trajectory chemical dynamics simulation is to calculate the potential energy function, the potential gradient, and Hessian directly from electronic structure theory. Since the potential energy, gradient, and Hessian are not needed in an analytic potential energy function, they all could be calculated by a single software package such as VENUS, however, electronic structure calculation requires a quantum chemistry software packages such as NWChem [3], MOLPRO [18], GAMASS [19].

In the classical chemical dynamics simulation initial conditions of the reactants for chemical reactions are being defined to calculate an ensemble of trajectories. Each trajectory is evaluated by numerically integrating either Hamilton’s or Newton’s equations of motion. Since the classical trajectory program solves the equation of the motion for particles, energy and energy gradients calculated in electronic structure program (i.e. NWChem).
3.2 VENUS, the general classical trajectory program

VENUS software package [2, 6] is known as general Monte Carlo classical trajectory program, which is widely being used in dynamics studies to simulate the dynamics of unimolecular reactions, bimolecular collisions and reactions, gas-surface collisions and reactions, and dynamics initiated at a potential energy barrier. VENUS numerically solves Hamilton’s equations of motion based on a system of $N$ atoms, besides the Cartesian coordinates and momenta.

![Flow chart of VENUS Program](image)

*Figure 3.1: Flow chart of VENUS Program*
1. Harmonic Stretch  \[ V = \frac{1}{2} f_2 (r - r_0)^2 \]

2. Morse Stretch  \[ V = D[1 - \exp(-D/r)]^2, \quad \beta = c_1 + c_2 \Delta r^2 + c_3 \Delta r^4 + c_4 \Delta r^6, \quad \Delta r = r - r_0 \]

3. Harmonic Bend  \[ V = \frac{1}{2} f_2 (\theta - \theta_0)^2, \quad f_2 = f_2^0 S(r) S(\theta) \text{ where } r \text{ and } r_0 \text{ define the bend.} \]

4. Harmonic alpha bend (wag)  \[ V = \frac{1}{2} f_2 (\alpha - \alpha_0)^2 \]

5. Generalized Lennard-Jones  \[ V = a_{\text{LJ}} + b_{\text{LJ}}^2 + c_{\text{LJ}} \]

6. 2-fold torsion (double bond) and 3-fold torsion (methyl group)  \[ V = V_0 (1 - \cos 2\theta)/2 = V_0 \sin^2 \tau \]

7. Generalized exponential repulsion and attraction  \[ V = a \exp (-b r) + c/r^a \]

8. Ghost pair interaction [23]  \[ V = q_1/q_2 + q_2 (1/r_1 + 1/r_2 + 1/r_3 + 1/r_4) \]

9. Tetrahedral Center [24]  \[ V = \frac{1}{2} \sum_{i=1}^{4} f_i \left( r_i - r_i^0 \right)^2 + \frac{1}{2} \sum_{i<j}^{4} g_{ij} \left( r_{ij} - r_{ij}^0 \right)^2 + \frac{1}{2} \sum_{k<l}^{4} h_{kl} \left( r_{kl} - r_{kl}^0 \right)^2 + \sum_{i,j} \delta_{ij} \Delta_{ij}^2 + \sum_{j,k} \delta_{jk} \Delta_{jk}^2 + \sum_{k,l} \delta_{kl} \Delta_{kl}^2 + \sum_{m,n} \delta_{mn} \Delta_{mn}^2 \]

10. Non-diagonal stretch-stretch interaction  \[ V = f_{ij} (r_i - r_i^0)(r_j - r_j^0), \quad f_{ij} = f_{ij}^0 S(r_i) S(r_j) \]

11. Non-diagonal stretch-bend interaction  \[ V = f_{i} \theta_i (r_i - r_i^0)(\theta_i - \theta_i^0), \quad f_\theta = f_\theta^0 S(r_i) S(\theta_i) \]

12. Non-diagonal bend-bend interaction  \[ V = f_{ij} \theta_{ij} (r_{ij} - r_{ij}^0)(\theta_{ij} - \theta_{ij}^0), \quad f_\theta = f_\theta^0 S(r_{ij}) S(\theta_{ij}) \]

13. Torsion for dihedral angle  \[ V = \sum_{n=1}^{N} \frac{\alpha_n^2}{2} \left[ 1 + \cos(n \phi - \gamma_n) \right] \]

Table 1.1: Different type of potential energy functions in VENUS

Execution of the program begins by reading Cartesian coordinates \( q \) and moments of inertia \( p \). The selection of initial conditions calls integration of classical equations of motion by calculating the potential energy \( V \), and its gradient \[ \frac{dv}{dq} \] and Hessian \( H \) to give out the vibrational energy within the molecule. At the final stage of the program, cross sections, scattering angels, product energy distribution, rate constant, etc. would be analyzed. The flow chart of the VENUS process can be found in Figure 3.1 [6].

Based on physical systems of interest in molecular dynamics simulations, VENUS provides a variety of potential energy functions to build blocks of potential surfaces, and molecular model systems. Table 1.1 shows some of the functions implemented in VENUS to let the user build an arbitrary potential energy surface, and
its first order derivative with respect to Cartesian coordinates [5]. All these potential energy functions are defined as $V(r, \theta, \alpha, \tau, \phi)$, where $r$ is known as interatomic distance, and $\theta, \alpha, \tau, \phi$ are represented as bond angles [20]. Therefore, the derivatives of the potential energy with respect to Cartesian coordinates would be:

$$\frac{\partial V}{\partial q_i} = \frac{\partial V}{\partial r} \frac{\partial r}{\partial q_i} + \frac{\partial V}{\partial \theta} \frac{\partial \theta}{\partial q_i} + \frac{\partial V}{\partial \alpha} \frac{\partial \alpha}{\partial q_i} + \frac{\partial V}{\partial \tau} \frac{\partial \tau}{\partial q_i} + \frac{\partial V}{\partial \phi} \frac{\partial \phi}{\partial q_i}$$

One of the critical issues about VENUS is running sequentially. The scalar code prevents the program from executing in parallel especially with other parallel chemical software packages (like NWChem). Furthermore, it should be of concern that each iteration of classical trajectory computation is tightly coupled with the previous step. In other words, producing the output on each trajectory calculation is based on the output from the steps before. This could be assumed to be the main reason in VENUS that forces the code to remain sequential, and prevents the program to be running on HPC platforms.

Another considerable point is the expensive calculations of potential energy, and the potential energy gradients in each step. Since trajectory integrations consist of many potential energy and its gradient calculations, a heavy workload would be expected for VENUS, and because of the scalar nature of code, the parallelization would be almost impossible to handle this issue on HPC clusters.
CHAPTER 4

DATA MODELING FOR CHEMICAL DYNAMICS SIMULATION

4.1 Introduction

The relationship between the continuing function of the input parameter data, and the output resulting file in terms of data modeling for calculation reduction is a challenge. Thus, the aim of this paper would be introducing an efficient data modeling methodology to be applied to the chemical dynamics simulation (i.e. VENUS), as a replacement of expensive computations. However, the first step is to know which calculated data would be essential for the modeling.

Classical dynamics trajectory simulation in VENUS calculates several components on each steps, including the potential energy, potential energy gradient, coordinates of atoms, momenta and etc. All these calculations are based on three dimension coordinates of atoms on each steps, which are all expensive to be computed.

Reducing a heavy calculation process of potential energy $V$, potential energy gradient $dvdq$, and making each steps of trajectory integrations independent (not for all but at least most of the calculations) is the goal of this new methodology, which we will introduce in this paper.

The traditional form of calculation (let’s say calculating the potential energy $V$ and potential energy gradient $dvdq$) is a serial code, in which each steps depends on previous calculated outputs. Observing Figure 2 would show this fact that after collecting the parameters from input, based on given information, VENUS calculates all the potential energy, and potential energy gradient with respect to calculating other parameters, and this will continue until last specified steps for all numbers of trajectories.

Since these calculations are assumed as heavy workload calculations, we tried to implement a data modeling algorithm, besides the current calculation process for $V$ and $dvdq$, then store the calculated data, and utilize those data in the middle of the
trajectory integration process to perform interpolation approximation for $V$ and $dvdq$, instead of heavy traditional calculations. However, devising new approach to provide a capability of approximation out of some trajectory steps, causes the reduction of more computational workload, and lead the application to have more independent calculation on each steps.

4.2 Apply Data Modeling approach

As we discussed in chapter 2, our data modeling technique has three main steps, such as Storing to-be-modeled data in a database on each steps of calculations, searching the database to find the closest data to be re-used for the data modeling algorithm, and finally replace expensive calculations with a light algorithm, which uses discovered data from the database. In this chapter we will show how all these phases will be applied to chemical dynamic simulation (VENUS), to reduce the highly cost calculations, by re-using the data. However, before going through each of these steps, a mathematical technique will be discussed roughly, which is going to be considered as a base idea of final ligh-computing algorithm that would be replaced by expensive calculations.

4.2.1 Data Interpolation and Extrapolation

Sometimes it is simple to get the value of function $f(x)$ at a set of points such as $x_1, x_2, ..., x_n$, which $(x_1 < x_2 < ... < x_n)$, however, sometimes $f(x)$ is more complex that might be a result from long numerical calculation or some physical measurement, which would be very hard to setup a simple functional form. In this case $f(x)$ needs to be estimated for arbitrary $x$, to let the function gives out a draw of smooth curve through the $x_i$ or maybe beyond it. If desired $x$ is in between largest and smallest of $x_i$’s, then the problem called Interpolation, otherwise if the $x$ is out of the range then the problem is called Extrapolation [21].

In this case, finding a sufficiently general form of schemes to model the function after is required to approximate large classes of functions in practice. However, it is very important to know that the Interpolation is related to, but distinct from function approximation. An approximate function is an easily computable function to use instead
of a more complex one with any desired points to calculate the approximation, but in case of Interpolation, the function \( f \) would be given the points which are not chosen by us.

This method could be used as a Machine Learning technique for our data modeling process, since we try to predict/approximate the next result from current saved results. However, it is important to know what the criteria is for storing to-be-modeled data in chemical dynamics simulation at the closest points, to be re-used for interpolation algorithm.

### 4.2.2 Choosing To-be-modeled Data

The first step for data modeling is to find those calculated data, which are necessary to be used by the data modeling algorithm. In the case of classical trajectory chemical dynamics simulation, and reducing the potential energy, and potential energy gradient calls, first we need to calculate the distance between atoms instead of using coordinates as one of the to-be-modeled data. This also causes data reduction without losing any information:

\[
R = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}
\]

Then we need to calculate the potential gradient with respect to distance \( R \), which is much lighter in terms of calculation. In other words:

\[
\frac{\partial V}{\partial q_i} \rightarrow \frac{\partial V}{\partial r_i}
\]

Figure 4.1 shows the code of converting potential gradient based on distance from gradient based on coordinates. This code demonstrates how inter-atomic distance
and Cartesian coordinates of the atoms in addition to other atomic parameters such as potential energy and gradient of the potential energy for multiple atoms, are given as input to give out derivative of potential energy with respect to inter-atomic distance ($dvdr$). As it could be perceived from this code, calculation of DVDR is fairly light in terms of calculation. Moreover, since the calculation of $\frac{\partial v}{\partial q_i}$ is based on the number of carterian Cartesian coordinate points, it would give out multiple-dimension data, the calculation of $\frac{\partial v}{\partial r_i}$ causes data reduction by applying distance ($R$), without losing any data.

Based on calculated $R$ and $dvdr$ we would be able to implement the interpolation algorithm to calculate the approximate value of the current potential energy, and $dvdr$ for each step, based on sufficient data from previous steps. Here, what we mean by sufficient data is to have some initial calculation for trajectories on some of the first steps in the traditional way, then whenever we reach a reliable data value to start approximation, then the interpolation algorithm is applied for that. Figure 4.3 demonstrates the code for Interpolation algorithm. This algorithm will be discussed later more in detail.
4.2.3 Store to-be-modeled data in a Dynamic In-Memory Database

Basically, in-memory computing refers to storing data on RAM to have a faster process over data. In many of the Big Data applications, which quick access to extensive amount of data is highly desired, in-memory processing is very common. In other words, in-memory processing is being used to speed up the access results from data-intensive problems like data analytics, and decision making. Since quick access to our modeled dataset is highly required, we try to store our data in a very simple two-dimensional array dataset.

In this case we designed a dynamic In-Memory table (simply called MemTable) to keep to-be-modeled data for future re-use. Figure 4.2 shows the structure of this MemTable.

<table>
<thead>
<tr>
<th>Dynamic array of distances between atoms (Sorted)</th>
<th>$\frac{dv}{dr}$ with respect to $\frac{dv}{dq_i}$</th>
<th>Potential Energy $V$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>$dv r_1$</td>
<td>$V_1$</td>
</tr>
<tr>
<td>$R_2$</td>
<td>$dv r_2$</td>
<td>$V_2$</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>$R_n$</td>
<td>$dv r_n$</td>
<td>$V_n$</td>
</tr>
</tbody>
</table>

As it is shown in this figure, MemTable consists of distance between atoms $R_i$, which is coming from Cartesian coordinates in $i_{th}$ step. $dv r_i$ with respect to $dv dq_i$, and potential energy $V_i$ of the $i_{th}$ step in trajectory integration. Since the number of bounds between atoms could be different according to the input parameters, and because the number of records per table is unpredictable, we cannot define the number of calculation preemptively. Therefore, MemTable should be able to adjust its size dynamically. This actually implemented for this method, and let MemTable to grow up dynamically. This means that after a specific number of records was inserted, if there was no more space.
to keep the data table, another amount of space would be added, which is defined as a default in the program.

The goal is to perform calculation for $V$ and $dvdq$ at least for two steps, and store distance $R$, potential energy $V$, and $dvdr$ into the MemTable. Consider that on each data insertion process, records should be sorted based on $Rs$. To get the highest efficiency on algorithm of sort while inserting data dynamically and sorted into the MemTable after each calculation, we implement the *Heapsort*.

### 4.2.4 Search for closest data from dataset

In the next two steps, the program automatically starts to search for the two closest $Rs$ in proportion to the current $R$. To make this clear, consider that on each step of the trajectories, atoms get new coordinates which results in a new $R$, let’s call it $R^*$. We change the traditional flow of the program in such a way that before doing any further calculation, MemTable should be searched for two inter-atomic distance, let’s say $R_1$ and $R_2$, which are:

$$R_1 < R^* < R_2$$

$R_1$, and $R_2$ will be opted in such a way that there must be no more closet $R$ with respect to $R^*$. If $R_1$, and $R_2$ were both found successfully, we would say enough data is found from the database, and we can retrieve all the data in the same record as $R_1$, and $R_2$ to re-use them for data modeling. However, the criteria for finding two closest data could be different for different cases, especially for different number of atoms, but this rule works well for calculation of two atoms. The another point that should be considered is the accuracy of data; finding close enough data from the database, may cause loss of information in further steps of calculation while the database is also growing up, and number of calls for data modeling increases. To solve this issue, we need to design a robust training set to reduce the chance of information loss, and increase the accuracy of resulting data up to the final step.

Training set is a part of our Machine Learning procedure to discover potentially predictive relationships between resulting data [22]. Based on type of algorithm and
number of atoms, this training test could be different, however, devising a good training set to give out more accurate results in this research would be based on experience. We will show our experimental training set in chapter 5, which would be suitable to be applied for finding closest data from database for data modeling of two atoms.

Therefore, if \( R_1 \) and \( R_2 \) were found from the database, and they met the training set criteria, then Data Modeling procedure would be called on, and the expensive calculation for potential energy \( V \) and \( dvdq \) will be prevented. In other words, if neither \( R_1 \) nor \( R_2 \) was presented in MemTable nor they were satisfied by the provided training set, then the original calculation would be called and resulting data should be stored in MemTable. For better and more precise results, we ignore interpolated outputs to be saved in MemTable, and we only try to keep calculated data.

### 4.2.5 Data modeling in-place of expensive calculation

In scientific applications, which final results are very crucial to scientists, approximating the results in terms of reducing the calculation workload should be defined sophisticatedly enough to cause a smooth curve through the desired output. In this work, a specific Interpolation formula has been devised to be assumed as a fair approximating algorithm in place of traditional calculation for potential energy \( V \), and potential energy gradient \( dvdq \). To do so, we first need to calculate the inter-atomic distance \( R \) between the atoms out of Cartesian coordinates of the current state of trajectory. While inter-atomic distance would be one of the main parameters for Interpolation, we must calculate the potential energy gradient with respect to inter-atomic distance instead of Cartesian coordinate data. This is being done by \( dvdq2dvdr \) algorithm (Figure 4.1), which is already discussed.

Figure 4.3 shows the algorithm for interpolating potential energy \( V \), and potential energy gradient (which is defined by \( g \)). Before calling this function, two closest \( R \) from the MemTable should be found, which \( R_1 < R^* < R_2 \). If any of these Rs couldn’t be found, traditional calculation must be triggered instead of interpolation. After obtaining \( R_1 \), and \( R_2 \) other data fields in the same record of \( R_1 \), and \( R_2 \) should be
retrieved and used as input parameters of Interpolation function. In other words, if we assume $R_i$, and $R_j$ are two records in $i_th$ and $j_th$ level of MemTable, the $dvdr_i$, and $dvdr_j$, in addition to $V_i$, and $V_j$ would be retrieved from MemTable to be used as another input parameters, then potential energy $V$, and $dvdq$ (which is named $g$ in this code) are being interpolated as the outputs.

Figure 4.4 shows the flow of the VENUS process after implementing the new Data Modeling method.
Setup

Loop over number of trajectories

Initializing trajectory

Loop over time $t_m$

Is $t_m > 2$?

Yes:

Search MemTable and find $R_1$ and $R_2$ for current $R^*$, which $R_1 < R^* < R_2$

No:

Interpolation/Approximation for $V$ and $dvdq$

Adams-Moulton integration $\Rightarrow q(t_m), p(t_m)$

Terminate Trajectory?

Yes:

Final State analysis

Another Trajectory?

Yes:

No:

End

No:

DVDQ at $t_m \Rightarrow \frac{\delta q_i}{\delta t}, \frac{\delta p_i}{\delta t}$

ENERGY $\Rightarrow V(r, \theta, \alpha, \tau, \phi)$

Store $R$, $dvd\alpha$, and $V$ in MemTable and sort based on $R_s$

Figure 4.4: Flow chart of New Structure of VENUS
CHAPTER 5

EXPERIMENTAL RESULTS FOR TWO ATOMS OF HYDROGEN (H₂)

The final part of this thesis is dedicated to the demonstration of our results out of current changes in the program. In this case we have chosen an input file for calculation of two atoms of H₂, with one trajectory integration, and 1000 steps. We have executed the VENUS codes on a system with following configurations:

- 2x Intel Xeon Quad core 2.66 GHz CPU
- 16 GB RAM

5.1 Results without Training set:

We first show the importance of a training set in terms of the accuracy of the final results. Figure 5.1 illustrates the final results for calculation of potential energy V of two hydrogen atoms (H₂) by new implementation of Data Modeling and interpolation algorithm in VENUS:

![Figure 5.1: Potential energy calculation for two atoms of hydrogen (H₂) in VENUS](image)

*Figure 5.1: Potential energy calculation for two atoms of hydrogen (H₂) in VENUS*
5.2 Experimental Training Set for \( \text{H}_2 \) Data Modeling in VENUS

As shown in Figure 5.1, the results are beyond what we expected, because the training set was not applied. This means that, we need to define whether R\(_1\), and R\(_2\) are in a specific range to be assumed as proper data. To specify this range we applied a new restriction in the search function described as follows:

First, we try to find the maximum between difference of R\(_1\), and R\(^*\) with R\(_2\), and R\(^*\):

\[
M = \text{MAX} (|R_1 - R^*|, |R_2 - R^*|)
\]

Then we calculate the average of all the maximum results up to \( k \) steps:

\[
\text{Average} = \frac{\sum_{i=1}^{k} \text{MAX} (|R_1 - R_i|, |R_2 - R_i|)}{k}
\]

For \( k \) steps, we keep finding the maximum between difference of R\(_1\), and R\(^*\) with R\(_2\), and R\(^*\), but this time instead of calculating the average of all \( M \), we start comparing the maximum values with a Threshold Constant \( C \) times final calculated average value \( \text{Avg} \).

\[
\begin{cases} 
M_i \leq (C \times \text{Avg}) & i > k \rightarrow \text{Interpolation} \\
\text{Otherwise} & \rightarrow \text{Calculation}
\end{cases}
\]

\( C \) as threshold constant is a number between 0.1, and 0.7, and in this work we kept \( k = 200 \) as a constant for total steps that average of \( M_i \)'s should be calculated. If \( M_i \) after \( k \) steps was in the range of \( C \) times \( \text{Avg} \), then interpolation will be triggered, otherwise traditional calculation would be called. The goal of this practice is mining good data out of our MemTable database, which lead the interpolation to get a more accurate approximation. Increasing this Threshold Constant \( C \) increments the number of Interpolation and decreases the total calculation number, however, it causes less
accurate results in further steps. On the other hand, small $C$ increases the number of calculations versus interpolations, but gives out a more precise result.

Figures 5.2-5.5 show the results for potential energy of two hydrogen atoms ($H_2$) atoms in 1000 steps, with different Threshold Constant $C$ values of 0.7, 0.4, 0.2, and 0.1.

![Graph showing potential energy calculation for two atoms of hydrogen ($H_2$) in VENUS with Threshold 0.7](image)

*Figure 5.2: Potential energy calculation for two atoms of hydrogen ($H_2$) in VENUS with Threshold 0.7*
Figure 5.3: Potential energy calculation for two atoms of hydrogen (H2) in VENUS with Threshold 0.4

Figure 5.4: Potential energy calculation for two atoms of hydrogen (H2) in VENUS with Threshold 0.2
As shown in Figures 5.2-5.5, smaller threshold constant gives out better results as well as threshold 0.1, which caused the data modeling results to almost overlap the original calculation for potential energy. So, these figures show us the training set play an important role in terms of data modeling and learning from data. More precise training sets leads the data modeling process to more accurate resulting.

Figure 5.6 demonstrates the number of original expensive calculations versus the number of interpolation calls as the data modeling process for two atoms of hydrogen in 1000 steps of one trajectory integration.

As shown in this figure, increasing the threshold in training set for data modeling algorithm causes more reduction in the number of expensive calculations, however, the accuracy of final output results would be decreased. Although the current training set is devised very precisely to provide very good data modeling results, it should be considered that it leads the interpolation process into a sort of trade-off between number of calculation and level of accuracy.
Figure 5.6: Number of Original Calculations versus Number of Interpolation calls for two atoms of Hydrogen H₂ in 1000 step
CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Today scientific computing is an inevitable part of most of the modern sciences, such as physics, astrophysics, climate studies, and chemistry. However, many scientific applications in these fields generate large-scale of output data with highly expensive computations, which may take long time to be finished, and also may consume a huge space for final output data.

Due to this issue, there is an incentive to reduce those costly calculations in most of the scientific applications. In this thesis research we introduced a new Data Modeling algorithm based on Machine Learning technique (i.e. data interpolation), to reduce highly expensive scientific calculations, by re-using the resulting data through all the processes of application.

We proposed a data modeling technique in this research, and we have applied our model into one of the Classical Trajectory Chemical Dynamics Simulations software packages (VENUS), which is very popular among computational chemistry scientists. This application needs to calculate several parameters of atoms in different trajectory integrations. However, in some cases such as the calculation of potential energy and potential energy gradient, the computation process is very expensive. The new method of data modeling technique is already applied into this application, to reduce those costly calculations with light-weight interpolation calls.

With devising a proper Training Set for our data modeling, we could get effective results in terms of expensive computation reduction, and final accurate results compared to their original ones. However, a trade-off can be made between computation reduction, and data accuracy based on current training set.

6.2 Future Work

In this research we have demonstrated the results of new Data Modeling technique for two atoms of hydrogen (H₂) in a thousand steps of one trajectory.
However, two major tasks would be the future works. First we need to study the results for more than two atoms. The current Data Modeling technique (Interpolation algorithm) that we devised, only works well with two atoms, and it would go far away from accuracy when we increase the number of atoms, even with a good training set. This means that these current algorithms could be efficient at least for 3-4 atoms, and any additional ones need a new method to get more precise data modeling. Therefore, devising new approach is significant task for the future.

Furthermore, since we can decrease the dependency of the code by adding the Interpolation algorithm, parallelization would be more desirable for the current sequential nature of code. Therefore, parallelizing the program in such a way to be able to run the VENUS on distributed HPC clusters should be a considerable task in the future.
REFERENCES


APPENDIX

DATA MODELING CODE FOR VENUS

The following is the code of new subroutines, which are added to VENUS software package to handle the new data modeling algorithms, including Interpolation, Dynamic In-Memory management and store procedure, DVDQ to DVDR convertor, Search, and Sort algorithms. All these codes are written in Fortran programming language. According to our latest tests, it is out of any execution errors or exceptions.

```fortran
SUBROUTINE NEWCALC(MODE)
  IMPLICIT DOUBLE PRECISION (A-H,O-Z)
  INCLUDE 'SIZES'

  BY: Misha Ahmadian

  This subroutine could be a proper replacement for current Energy and Gradient calculation, by implementing Interpolation

  MODE:
  1: ONLY DVDQ
  2: ONLY ENERGY
  3: BOTH DVDQ AND ENERGY

  COMMON/PRLIST/T,V,H,TIME,NTZ,NT,ISEED0(8),NC,NX
  COMMON/QPDOT/Q(NDA3),PDOT(NDA3)
  COMMON/PQDOT/P(NDA3),QDOT(NDA3),W(NDA)
  COMMON/GDD/DIATM(MXDIATOM,4),GSTOP(MXPTH,6),IATMP(MXPTH,NDA),
  * IDIATM(MXDIATOM,2),IATSP(MXPTH,3),ISOTOPE(NDA),NPTh,NDIATOM,
  * NMEM,NDISK,NS,NCNKSP,NHESS,NCPU,NIISO,NIISO,METHOD
  COMMON/PRFLAG/NFQP,NCOOR,NFR,NUMR,NFB,NUMB,NFA,NUMA,NFTAU,
  * NUMTAU,NFNT,NUMTD,NUMHD,NTMD,NUMHT
  COMMON/FORCES/NATOMS,I3N,NST,NM,NB,NA,NLJ,NTAU,NEXP,NGHOST,NTET,
  * NVRR,NVRT,NVTT,NANG,NAXT,NSN2,NRYD,NHFD,NLEPSA,NLEPSB,NDMBE,
  * NAXT,NM,NMO,NRCC0
  COMMON/RISTS/NPATOM,LDATA,LDATA2,DATA,NRECL

  INTERFACE
    SUBROUTINE REALLOC(MEMTBL,INXTBL,KEYFLD,NFIELD
    * ,NEWSIZE,DEFINC)
    DOUBLE PRECISION, ALLOCATABLE, INTENT(INOUT)::
    * MEMTBL(:,:)
    INTEGER,INTENT(INOUT) :: INXTBL(:)
    DOUBLE PRECISION, ALLOCATABLE :: TMPMEM(:,:)
    INTEGER, ALLOCATABLE :: TMPINX
```

34
INTEGER KEYFLD, NFIELD, DEFINC, NEWSIZE, OLDSIZE, I
END SUBROUTINE REALLOC
END INTERFACE

C
C   DYNAMIC MEMORY ARRAY SPACE TO KEEP CALCULATED DATA
DOUBLE PRECISION, ALLOCATABLE, SAVE :: MEMTBL(:, :)
DOUBLE PRECISION, ALLOCATABLE :: R(:)
DOUBLE PRECISION R1, R2, V1, V2, DVOR, DVDR1, DVDR2
DOUBLE PRECISION REC1(LDATA2), REC2(LDATA2)
INTEGER, ALLOCATABLE, SAVE :: INXTBL(:)
INTEGER, SAVE :: COUNTER = 0, ARRPOS = 0, NEWSIZE
INTEGER MODE, DBLPR, KEYFLD, DEFINC, LOG2, STRMOD, THRSHLD
LOGICAL HASR1, HASR2, ISUPDT, INTERPL, SAVCALC

C
C   - DEFAULT SIZE OF DOUBLE PRECISION IS 8-BYTE
C   - THE KEY FILED (KEYFLD) IS THE FILED OF EACH RECORD THAT SHOULD
C      BE SORTED. AT THE TIME THE FIRST FILED IS CONSIDERED.
PARAMETER (DBLPR = 8, KEYFLD = 1, DEFINC = 100)
C
IF (.NOT. ALLOCATED(MEMTBL)) THEN
   NEWSIZE = DEFINC
   ALLOCATE(MEMTBL(NEWSIZE, LDATA2))
ENDIF
IF (.NOT. ALLOCATED(INXTBL)) ALLOCATE(INXTBL(NEWSIZE))
IF (.NOT. ALLOCATED(R)) ALLOCATE(R(NPATOM))

C
WRITE(*,*) "******************************"
HASR1 = .FALSE.
HASR2 = .FALSE.
ISUPDT = .FALSE.
INTERPL = .TRUE. !JUST FOR TEST & DEBUG
SAVCALC = .TRUE. !JUST FOR TEST & DEBUG
C
AFTER FIRST CALCULATION
IF (INTERPL .AND. (COUNTER .GT. 0)) THEN
   CALL DISTANCE(Q, NATOMS, NPATOM, R)
ENDIF

C
IF AT LEAST TWO RECORDS WAS STORED, THEN WE CAN FIND
TWO CLOSEST R1 & R2 TO THE CURRENT R
IF (ARRPOS .GE. 2) THEN
   CALL SEARCH(MEMTBL, INXTBL, LDATA2, NEWSIZE, ARRPOS, KEYFLD
*          , R(1), REC1, REC2, HASR1, HASR2)
ENDIF

ENDIF
C
IF WE COULD FIND TWO CLOSE POINTS TO THE CURRENT R
THEN WE CAN CALL INTERPOLATION ALGORITHM, OTHERWISE
WE NEED TO CALL POTENTIAL ENERGY CALCULATION
IF (INTERPL.AND.HASR1.AND.HASR2) THEN
C ***** INTERPOLATION *****
R1=REC1(1)
V1=REC1(NPATOM+1)
DVDR1=REC1(NPATOM+2)
R2=REC2(1)
V2=REC2(NPATOM+1)
DVDR2=REC2(NPATOM+2)
CALL INPOLR(NATOMS,MODE,R1,V1,DVDR1,R2,V2,DVDR2)
*  "***********************"
C  " INTERPOLATION INCURRED"
C  "***********************"
ELSE
C ***** CALCULATION *****
CALCULATED DISTANCE BETWEEN CURRENT ATOMS
IF (INTERPL.AND.(COUNTER.EQ.0)) THEN
CALL DISTANCE(Q,NATOMS,NPATOM,R)
ENDIF
C  "***********************"
IF (MODE.EQ.1) THEN
"MODE1: ONLY DVDQ"
ISUPDT=.FALSE.
CALL DVDQ
ELSEIF (INTERPL) CALL ENERGY
ELSEIF (MODE.EQ.2) THEN
"MODE2: ONLY ENERGY"
ISUPDT=.TRUE.
CALL ENERGY
ELSE
"MODE3: BOTH DVDQ & ENERGY"
ISUPDT=.FALSE.
CALL DVDQ
CALL ENERGY
ENDIF
C  "***********************"
CALCULATE DVDR W.R.T DVDQ
IF (INTERPL) CALL DVDR2R(Q,PDOT,NATOMS,DVDR)
C
C ------------STORE LAST CALCULATED VALUES IN MEMORY-------------
IF (INTERPL.OR.SAVCALC) THEN
INCREASE THE MEMORY SPACE WHENEVER THE CURRENT
MEMTBL SIZE WAS NOT ENOUGH BIG TO STROE MORE DATA
IF(ARRPOS.EQ.NEWSIZE) THEN
CALL REALLOC(MEMTBL,INXTBL,KEYFLD,LDATA2
,NEWSIZE,DEFINC)
ENDIF
C
CALL STORE(MEMTBL,INXTBL,NEWSIZE,LDATA2,ARRPOS,KEYFLD
*,R,V,DVDR,Q,NATOMS,NPATOM,STRMOD,ISUPDT)
ENDIF
C
ENDIF
C
COUNTER=COUNTER+1
C
SAVE THE MEMTBL ON DISK FOR JUST TEST AND DEBUG
IF (SAVCALC.AND.(NC.EQ.NS)) THEN
  CALL SAVE(MEMTBL,INXTBL,NEWSIZE,LDATA2,ARRPOS)
ENDIF
C
END
C-----------------------------------
C       THE SIZE OF THE MEMORY TABLE WOULD BE UNPREDICTABLE, SINCE
C       WE DON'T KNOW HOW MANY TIMES THE ONO-INTERPOLATION CALCULATION
C       TRIGGERS TO STORE THE DATA. THEREFORE, WE NEED TO MANAGE THE
C       MEMORY DYNAMICALLY TO INCREASE THE SIZE OF MEMORY ON DEMAND.
C-----------------------------------

SUBROUTINE REALLOC(MEMTBL,INXTBL,KEYFLD,NFIELD,NEWSIZE,DEFINC)
DOUBLE PRECISION, ALLOCATABLE, INTENT(INOUT):: MEMTBL(:, :)
INTEGER,ALLOCATABLE,INTENT(INOUT) :: INXTBL(:)
DOUBLE PRECISION, ALLOCATABLE :: TMPMEM(:, :)
INTEGER, ALLOCATABLE :: TMPINX(:)
INTEGER KEYFLD,NFIELD,DEFINC,NEWSIZE,OLDSIZE,I

OLDSIZE=NEWSIZE
NEWSIZE=NEWSIZE+DEFINC
C
RESIZE MEMTBL EVERY 'SIZEINC' STEPS
ALLOCATE(TMPMEM(OLDSIZE,NFIELD))
TMPMEM(1:OLDSIZE,1:NFIELD)=MEMTBL
DEALLOCATE(MEMTBL)
ALLOCATE(MEMTBL(NEWSIZE,NFIELD))
MEMTBL(1:OLDSIZE,1:NFIELD)=TMPMEM
DEALLOCATE(TMPMEM)
C
RESIZE INXTBL EVERY 'SIZEINC' STEPS
ALLOCATE(TMPINX(OLDSIZE))
TMPINX=INXTBL
DEALLOCATE(INXTBL)
ALLOCATE(INXTBL(NEWSIZE))
INXTBL(1:OLDSIZE)=TMPINX
DEALLOCATE(TMPINX)
END
C-----------------------------------
C     MAKE INDEX TABLE OF SORTED DATA:
C       THE PURPOSE IS TO MAKE A TABLE OF INDEX INCLUDING THE INDEX OF
C       EACH RECORD SORTED BY REQUESTED FIELD (KEYREC)
C
THIS SUBROUTINE IS BASED ON "QUICK SORT" ALGORITHM IMPLEMENTED
BY (WILLIAM H.PRESS, SAUL A. TEUKOLSKY, WILLIAM T. VETTERLING,
BRIAN P. FLANNERY: "NUMERICAL RECIPES IN FORTRAN", 1992
SUBROUTINE QKSORT(ARRAY, INXTBL, NDATA, NFIELD, KEYFLD)
DOUBLE PRECISION ARRAY(NDATA,NFIELD),ADATA
INTEGER, ALLOCATABLE :: ISTACK(:),TMPSTK(:)
INTEGER NDATA,NFIELD,INXTBL(NDATA),MAXARR,NSTACK,KEYFLD
INTEGER I,J,K,INDX,N,INXTMP,JSTACK
C
WHEN A SUBARRAY HAS GOTTEN DOWN TO SOME SIZE MAXARR, IT BECOMES
FASTER TO SORT IT BY STRAIGHT INSERTION. THE OPTIMAL SETTING
OF MAXARR IS MACHINE DEPENDENT BUT (MAXARR = 7) IS NOT TOO FAR
WRONG!!
PARAMETER ( MAXARR=7)
QUICKSORT REQUIRES AN AUXILIARY ARRAY OF STORAGE, OF LENGTH
2*LOG2(N), WHICH IS USED AS PUSH-DOWN STACK FOR KEEPING TRACK
OF THE PENDING SUBARRAYS.
NSTACK=2*(LOG2(NEWSIZE)+1)
C
ALLOCATE(ISTACK(NSTACK))
C
INITIALIZE VARIABLES
DO J=1,NDATA
  INXTBL(J)=J
ENDDO
JSTACK=0
N=NDATA
L=1
C
INSERTION SORT WHEN SUBARRAY IS SMALL ENOUGH
IF (N-L.LT.MAXARR) THEN
  DO J=L+1,N
    INDX=INXTBL(J)
   ADATA=ARRAY(INDX,KEYFLD)
    DO I=J-1,1,-1
      IF (ARRAY(INXTBL(I),KEYFLD).LE.ADATA) GOTO 2030
      INXTBL(I+1)=INXTBL(I)
    ENDDO
    I=0
  2030 INXTBL(I+1)=INDX
  ENDDO
C
IF (JSTACK.EQ.0) THEN
  DEALLOCATE(ISTACK)
  RETURN
ENDIF
C
POP STACK AND BEGIN A NEW ROUND OF PARTITIONING
N=ISTACK(JSTACK)
L=ISTACK(JSTACK-1)
JSTACK=JSTACK-2
C OTHERWISE THE QUICK SORT ALGORITHM WILL BE EXECUTED
ELSE
C CHOOSE MEDIAN OF LEFT, CENTER, AND RIGHT ELEMENTS AS
C PARTITIONING ELEMENT (ARRAY).
K=(L+N)/2
INXTMP=INXTBL(K)
INXTBL(K)=INXTBL(L+1)
INXTBL(L+1)=INXTMP
C REARRANGE SO THAT ARRAY(L+1) <= ARRAY(L) <= ARRAY(N)
IF (ARRAY(INXTBL(L+1),KEYFLD).GT.ARRAY(INXTBL(N),KEYFLD)) THEN
INXTMP=INXTBL(L+1)
INXTBL(L+1)=INXTBL(N)
INXTBL(N)=INXTMP
ENDIF
C IF (ARRAY(INXTBL(L),KEYFLD).GT.ARRAY(INXTBL(N),KEYFLD)) THEN
INXTMP=INXTBL(L)
INXTBL(L)=INXTBL(N)
INXTBL(N)=INXTMP
ENDIF
C IF (ARRAY(INXTBL(L+1),KEYFLD).GT.ARRAY(INXTBL(L),KEYFLD)) THEN
INXTMP=INXTBL(L+1)
INXTBL(L+1)=INXTBL(L)
INXTBL(L)=INXTMP
ENDIF
C INITIALIZE POINTERS FOR PARTITIONING
I=L+1
J=N
INDX=INXTBL(L)
C PARTITIONING ELEMENT
ADATA=ARRAY(INDX,KEYFLD)
C BEGINING OF INNERMOST LOOP
2031 CONTINUE
C SCAN UP TO FIND ELEMENT > ADATA
C IF (I+1.GT.N) GOTO 2032
I=I+1
IF (ARRAY(INXTBL(I),KEYFLD).LT.ADATA) GOTO 2031
C 2032 CONTINUE
C SCAN DOWN TO FIND ELEMENT < ADATA
C IF (J-1.LT.1) GOTO 2033
J=J-1
IF (ARRAY(INXTBL(J),KEYFLD).GT.ADATA) GOTO 2032
C POINTERS CROSSED. EXIT WITH PARTITIONING COMPLETE.
2033 IF (J.LT.1) GOTO 2034
C EXCHANGE ELEMEVENTS
INXTMP=INXTBL(I)
INXTBL(I)=INXTBL(J)
INXTBL(J)=INXTMP

C     END OF INNERMOST LOOP
GOTO 2031
C
C     INSERT PARTITIONING ELEMENT
2034  INXTBL(L)=INXTBL(J)
INXTBL(J)=INDX
JSTACK=JSTACK+2
C
C     'AUXILIARY ARRAY OF STORAGE IS SMALL'
C     THIS WILL NOT HAPPEN, BUT FOR SOME RARE CASE COULD BE USEFUL
IF (JSTACK.GT.NSTACK) THEN
  PAUSE 'AUXILIARY ARRAY OF STORAGE IS SMALL'
  ALLOCATE(TMPSTK(NSTACK))
  TMPSTK=ISTACK
  DEALLOCATE(ISTACK)
  ALLOCATE(ISTACK(JSTACK))
  ISTACK(1:NSTACK)=TMPSTK
  DEALLOCATE(TMPSTK)
  NSTACK=JSTACK
ENDIF
C
C     PUSH POINTERS TO LARGE SUBARRAY ON STACK, PROCESS SMALLER
C     SUBARRAY IMMEDIATELY
IF (N-I+1.GT.J-1) THEN
  ISTACK(JSTACK)=N
  ISTACK(JSTACK-1)=I
  N=J-1
ELSE
  ISTACK(JSTACK)=J-1
  ISTACK(JSTACK-1)=L
  L=I
ENDIF
ENDIF
GOTO 2029
END

C-----------------------------------------------------------------------
C     THE SIZE OF THE MEMORY ALLOCATED TO MEMTBL WOULD BE DYNAMIC
C     IN THIS CASE ITS BETTER TO USE HEAP SORT ALGORITHM WHICH COULD
C     BE SUITABLE FOR INPLACE SORT TO REDUCE EXTRA SPACE, AND SORTING
C     INSERTED DATA INTO A SORTED ARRAY.
C-----------------------------------------------------------------------
SUBROUTINE HPSORT(MEMTBL,INXTBL,ARRSZ,ARRWID,ARRLEN,KEYFLD)
  INTEGER ARRSZ,INXTBL(ARRSZ),ARRWID,KEYFLD,START,BOTTOM
  INTEGER ARRLEN,TMP,L,I,J,IR
  DOUBLE PRECISION MEMTBL(ARRSZ,ARRWID)
C
C     THE ARRAY SHOULD HAVE AT LEAST 2 ITEMS
IF (ARRLEN.LT.2) RETURN
C

L = (ARRLEN/2) + 1
IR = ARRLEN

C
2040  CONTINUE
IF (L .GT. 1) THEN
    L = L - 1
    TMP = INXTBL(L)
ELSE
    TMP = INXTBL(IR)
    INXTBL(IR) = INXTBL(1)
    IR = IR - 1
    IF (IR .EQ. 1) THEN
        INXTBL(1) = TMP
        RETURN
    ENDF
ENDIF
ENDIF

C
STIFDOWN ELEMENT MEMTBL(TMP,KEYFLD) TO ITS PROPER LEVEL
I = L
J = L + L
DO WHILE (J .LE. IR)
    IF (J .LT. IR) THEN
        IF (MEMTBL(INXTBL(J),KEYFLD) .LT. MEMTBL(INXTBL(J+1),KEYFLD)) J = J + 1
    ENDF
    IF (MEMTBL(TMP,KEYFLD) .LT. MEMTBL(INXTBL(J),KEYFLD)) THEN
        INXTBL(I) = INXTBL(J)
        I = J
        J = J + J
    ELSE
        J = IR + 1
    ENDF
ENDDO
INXTBL(I) = TMP

C
GOTO 2040
END

C BECAUSE THERE IS ONLY LOG(a) AND LOG10(a) AS NATURAL AND COMMON
C FORTRAN BUILT-IN LOGARITHM FUNCTIONS, HERE WE TRY TO IMPLEMENT
C THE LOG2(a) (LOG BASE 2 OF A) AND LET QUICKSORT USE THIS AT THE
C BEGINNING.

C
INTEGER FUNCTION LOG2(N)
INTEGER N, I, J
C
I = N
J = 0

C
2028 IF (I .NE. 1) THEN
    J = J + 1
THE LOG BASE 2 OF AN INTEGER IS THE SAME AS THE POSITION OF THE HIGHEST BIT SET (OR MOST SIGNIFICANT BIT SET, MSB).

```fortran
I = ISHFT(I, -1)
GOTO 2028
ENDIF
LOG2 = J + 1
END
```

-----------------------------------------------------------------------

CALCULATION OF DISTANCE BETWEEN ATOMS WITH \( \frac{N(N-1)}{2} \) PATHS

```fortran
SUBROUTINE DISTANCE(Q, NATOMS, NPATOM, R)
IMPLICIT DOUBLE PRECISION (A-H, O-Z)
INTEGER NATOMS, NPATOM
DOUBLE PRECISION Q(3*NATOMS), R(NPATOM)

I = 1
C = 1
DO K = 1, NPATOM
    X1 = Q(I)
    Y1 = Q(I+1)
    Z1 = Q(I+2)
    J = I+3
    DO L = 1, NATOMS - K
        X2 = Q(J)
        Y2 = Q(J+1)
        Z2 = Q(J+2)
        R(C) = SQRT(((X2 - X1)**2 + (Y2 - Y1)**2 + (Z2 - Z1)**2))
        J = J + 3
        C = C + 1
    ENDDO
    I = I + 3
ENDDO
END
```

-----------------------------------------------------------------------

CALCULATE DV/DR ACCORDING TO CURRENT DVDQ AND DISTANCE R

ALGO BY: Dr. Yu Zhuang     IMPL: Misha Ahmadian

INPUT: RATOMS, Q, V, PDOT, NATOMS

OUTPUT DVDR: DERIVATIVE OF POTENTIAL ENERGY W.R.T INTER-ATOMIC DISTANCE

```fortran
SUBROUTINE DVDQ2R(Q, PDOT, NATOMS, DVDR)
DOUBLE PRECISION DVDR, TMP, Q(NATOMS*3), PDOT(NATOMS*3), Y(3)
INTEGER NATOMS

Y(1) = Q(4) - Q(1)
Y(1) = Q(5) - Q(2)
Y(1) = Q(6) - Q(3)

DVDR = SQRT((PDOT(1)**2 + PDOT(2)**2 + PDOT(3)**2))
```
TMP = (Y(1)*PDOT(1)) + (Y(2)*PDOT(2)) + (Y(3)*PDOT(3))

IF (TMP.LT.0.0) DVDR = -DVDR

END

-----------------------------------------------------------------------

ALGO BY: Dr. Yu Zhuang     IMPL: Misha Ahmadian

INPUT: R1,V1,G1VAL, R2,V2,G2VAL, R,Q.

MUST: R1< R < R2

R: INTER-ATOMIC DISTANCE BETWEEN THE TWO ATOMS
Q: CARTESIAN COORDINATES OF THE TWO ATOMS AT WHICH V,G ARE TO BE FITTED
V: POTENTIAL ENERGY OF THE DIATOMS
GVAL: DERIVATIVE OF POTENTIAL ENERGY W.R.T INTER-ATOMIC DISTANCE

OUTPUT: V,G,GVAL

GVAL: DERIVATIVE OF POTENTIAL ENERGY W.R.T. INTER-ATOMIC DISTANCE
G: GRADIENT OF THE POTENTIAL ENERGY IN CARTESIAN COORDINATES

-----------------------------------------------------------------------

SUBROUTINE INPOLR(MODE,NATOMS,R1,V1,G1VAL,R2,V2,G2VAL,R,Q,V,G,GVAL)

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
INCLUDE 'SIZES'

DOUBLE PRECISION R1,V1,G1VAL,R2,V2,G2VAL,R,V,GVAL,CURRV
DOUBLE PRECISION Q(3*NATOMS),G(3*NATOMS),CURRPD(3*NATOMS)
DOUBLE PRECISION TMP,Y(3)
INTEGER MODE,NATOMS

V = ((R2-R)*V1+(R-R1)*V2)/(R2-R1)

Y(1) = Q(4) - Q(1)
Y(2) = Q(5) - Q(2)
Y(3) = Q(6) - Q(3)

TMP = SQRT((Y(1)**2) + (Y(2)**2) + (Y(3)**2))
Y(1) = Y(1)/TMP
Y(2) = Y(2)/TMP
Y(3) = Y(3)/TMP

GVAL = ((R2-R)*G1VAL+(R-R1)*G2VAL)/(R2-R1)

MODE 2: IF IT CALLS ONLY ENERGY, KEEP CURRENT PDOT
IF (MODE.NE.2) THEN
  G(1) = Y(1)*GVAL
  G(2) = Y(2)*GVAL
  G(3) = Y(3)*GVAL
  G(4) = -G(1)
  G(5) = -G(2)
\( G(6) = -G(3) \)

ENDIF

C END

C-----------------------------------------------------------------------
C STORE CALCULATED VALUES FROM EACH TRAJECTORY I MEMORY TO USE THEM LATER FOR FIND TWO CLOSE POINTS FOR THE CURRENT DISTANCE OF THE ATOMS ON EACH TRAJECTORY PHASE
C
C STRMOD:
C 1) LEAVE POTENTIAL ENERGY EMPTY
C 2) DO NOT INSERT
C *) INSERT RECORD IN MEMTBL
C-----------------------------------------------------------------------

SUBROUTINE STORE(MEMTBL, INXTBL, ARRSIZ, ARRWID, ARRPOS, KEYFLD
* R, V, DVDR, Q, NATOMS, NPATOM, STRMOD, ISUPDT)
INTEGER ARRSIZ, ARRWID, ARRPOS, KEYFLD, NATOMS, NPATOM, I, J
INTEGER INXTBL(ARRSIZ), STRMOD
DOUBLE PRECISION MEMTBL(ARRSIZ, ARRWID)
* R(NPATOM), V, DVDR, Q(NATOMS*3)
LOGICAL ISUPDT

C CHECK TO SEE IF STORE IS ADDING DATA OR UPDATING A ROW
IF (.NOT. ISUPDT) ARRPOS=ARRPOS+1
IF (ARRPOS.EQ.0) RETURN

C ADD ITEMS TO THE GLOBAL MEMORY ARRAY
DO I=1, NPATOM
  MEMTBL(ARRPOS, KEYFLD)=R(I)
ENDDO
MEMTBL(ARRPOS, I)=V
MEMTBL(ARRPOS, I+1)=DVDR
DO J=1, NATOMS*3
  MEMTBL(ARRPOS, J+I+1)=Q(J)
ENDDO

C INXTBL(ARRPOS)=ARRPOS
C SORT THE MEMORY ARRAY BASE ON R
CALL HPSORT(MEMTBL, INXTBL, ARRSIZ, ARRWID, ARRPOS, KEYFLD)

C END

C-----------------------------------------------------------------------
C KEEP THE CALCULATED VALUES STORED IN MEMORY INTO A SEPARATE FILE NAMED ‘DATA.OUT’ JUST FOR TEST AND DEBUG
C-----------------------------------------------------------------------

SUBROUTINE SAVE(MEMTBL, INXTBL, ARRSIZ, LDATA2, LASTPOS)
INTEGER ARRSIZ, INXTBL(ARRSIZ), LDATA2
DOUBLE PRECISION MEMTBL(ARRSIZ, LDATA2)

C 2024 FORMAT(5X, 'ERROR: (DATA.out) file cannot be opened/created!')
2027 FORMAT(9F11.6, $)
C
OPEN(70, FILE='DATA.out', FORM='FORMATTED', ERR=2023)
GOTO 2100
C IF ERROR OCCURED WHILE OPENING THE FILE
WRITE(6,2024)
STOP
C 2100 DO I=1, LASTPOS
   WRITE(70,2027) (MEMTBL(INXTBL(I),J), J=1, LDATA2)
   C          NEXT(NEW) LINE
   WRITE(70, '(A)') ''
ENDDO
CLOSE(70)
END
C-----------------------------------------------------------------------
C       BINARY SEARCH TO FIND THE CLOSEST ATOMS TO THE CURRENT
C       DISTANCE BETWEEN ATOMS
C-----------------------------------------------------------------------
SUBROUTINE SEARCH(MEMTBL, INXTBL, LDATA2, ARRSIZ, LENGTH, KEYFLD, R, REC1, REC2, HASR1, HASR2)
   INTEGER ARRSIZ, INXTBL(ARRSIZ), LDATA2, LENGTH, KEYFLD, LEFT
   , MIDDLE, RIGHT, THRSHLD
   DOUBLE PRECISION MEMTBL(ARRSIZ, LDATA2), R, R1, R2, EMPTY
   DOUBLE PRECISION REC1(LDATA2), REC2(LDATA2), MX, AVGTHR
   DOUBLE PRECISION, SAVE :: AVGR=0.0
   INTEGER, SAVE :: CNT=1
   LOGICAL HASR1, HASR2, HASTHR
C PARAMETER(THRSHLD=200, AVGTHR=0.1) ! AVGTH COULD BE CHANGED
C
LEFT=1
RIGHT=LENGTH
EMPTY=TINY(0.0D0)
HASR1=.TRUE.
HASR2=.TRUE.
HASTHR=.TRUE.

C DO WHILE (RIGHT.GT.LEFT)
   MIDDLE=NINT((LEFT+RIGHT)/2.0)
   IF (MEMTBL(INXTBL(MIDDLE),KEYFLD).GE.R) THEN
      RIGHT=MIDDLE-1
   ELSE
      LEFT=MIDDLE
   ENDIF
ENDDO
C IF ((RIGHT+1).LE.LENGTH) THEN
   LEFT=RIGHT+1
ELSE
   LEFT=RIGHT
ENDIF
DO WHILE ((MEMTBL(INXTBL(LEFT),KEYFLD).LE.R)
* .AND.(LEFT.LT.LENGTH))
\texttt{LEFT=LEFT+1}
\texttt{ENDDO}

\texttt{IF (.NOT. (MEMTBL(INXTBL(RIGHT),KEYFLD).LT.R)) HASR1=.FALSE.}
\texttt{IF (.NOT. (MEMTBL(INXTBL(LEFT),KEYFLD).GT.R)) HASR2=.FALSE.}

\texttt{CALCULATE THE AVERAGE OF MAX(|R1-R|, |R2-R|) FOR 'THRSHLD' TIMES. AFTER THEN CHECK IF THE AVG IS LESS THAN THIS THE CALCULATION SHOULD INCCURE INSTEAD OF INTERPOLATION}

\texttt{IF (HASR1.AND.HASR2.AND.HASTHR) THEN}
\texttt{"******************************"}
\texttt{R1=MEMTBL(INXTBL(RIGHT),KEYFLD)}
\texttt{R2=MEMTBL(INXTBL(LEFT),KEYFLD)}
\texttt{MX=MAX(ABS(R1-R),ABS(R2-R))}

\texttt{IF (CNT.LE.THRSHLD) THEN}
\texttt{AVGR=DBLE(((AVGR*(CNT-1)) + MX)/CNT)}
\texttt{CNT=CNT+1}
\texttt{ELSE}
\texttt{IF (MX.GE.(AVGTHR*AVGR)) THEN}
\texttt{HASR1=.FALSE.}
\texttt{HASR2=.FALSE.}
\texttt{ELSE}
\texttt{ENDIF}
\texttt{ENDIF}
\texttt{"******************************"}

\texttt{ENDIF}

\texttt{DO \texttt{I=1,LDATA2}}
\texttt{REC1(I)=MEMTBL(INXTBL(RIGHT),I)}
\texttt{REC2(I)=MEMTBL(INXTBL(LEFT),I)}
\texttt{ENDDO}
\texttt{END}