# Protein Structure Elastic Network Models and the Rank 3 Positive Semidefinite Matrix Manifold

by

Xiao-Bo Li

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2018

**Examining Committee Membership**

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

| | |
|---|---|
| External Examiner: | Zhijun Wu<br>Professor, Dept. of Mathematics,<br>Iowa State University |
| Supervisor: | Forbes Burkowski<br>Associate Professor Emeritus, Cheriton School of Computer Science,<br>University of Waterloo |
| Internal Member: | Justin Wan<br>Professor, Cheriton School of Computer Science,<br>University of Waterloo |
| Internal Member: | Christopher Batty<br>Assistant Professor, Cheriton School of Computer Science,<br>University of Waterloo |
| Internal-External Member: | Spiro Karigiannis<br>Associate Professor , Dept. of Pure Mathematics,<br>University of Waterloo |

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

This thesis is a contribution to the study of protein dynamics using elastic network models (ENMs).

An ENM is an abstraction of a protein structure where inter-atomic interactions are assumed to be modelled by a Hookean potential energy which is a function of inter-atomic distances. This model has been studied by various authors, and despite being a very simple model, can nonetheless provide a realistic understanding of protein dynamics. For example, it was shown by Tirion, that the Hookean potential energy can reproduce the normal mode fluctuations of the more complicated semi-empirical potential. In addition, it was shown by Tekpinar and Zheng that linearly interpolating two Hookean potentials can correctly model the order of local conformational changes before global conformational changes during ATP-driven conformational changes.

The purpose of this thesis is to provide a second mathematical formulation for modelling ENMs. This thesis suggests removing the square-root in the Hookean potential which leads to a positive semidefinite (PSD) potential that is a function of quadrances rather than distances. There are many similarities between the two approaches, but also many differences. One main difference is PSD matrices are linearly related to quadrance, the square of distance, which opens the way to model the PSD potential using perceptrons whose weight matrix is a rank 3 PSD matrix. This interesting consequence is left as a topic of future research.

The PSD potential is just as appropriate for modelling ENMs as observed by the following two agreements: The PSD potential produces normal mode fluctuations that agree with the Hookean potential introduced by Tirion. This agreement suggests both potentials provide the same information about a protein structure's flexibility. The generalization of the Hookean iENM potential (introduced by Tekpinar and Zheng) to the PSD iENM potential also interpolates the local conformational changes before the global conformational changes, in agreement with the original Hookean observations.

Recall that the equations of motion in classical mechanics is formulated using an abstract Riemannian manifold. This abstraction gives modellers the flexibility to consider different Riemannian manifolds appropriate to the problem.

After the introduction of the Hookean potential, the study of protein dynamics still uses the 3n dimensional Euclidean space as the Riemannian manifold, the same Riemannian manifold used by the semi-empirical potential. This is because both the semi-empirical potential and the Hookean potential assume the atomic coordinates of a protein structure are represented by a 3n by 1 vector.

However, with the introduction of the PSD potential, the protein structure's atomic coordinates are represented as a point on the rank 3 n by n PSD matrix manifold. Consequently, a new Riemannian manifold for modelling protein dynamics has been proposed.

In order to model protein dynamics on the rank 3 PSD matrix manifold, the equations of motion needs to be defined. This thesis presents the geometric objects: horizontal projection, gradient, Hessian, and retraction required for formulating the equations of motion for protein structures as an optimization problem on the rank 3 PSD matrix manifold. These formulas are a modification of the original formulas introduced by Journée et al. to allow constraints relevant to a protein structure to be described. Rosen's correction to the constraint manifold was already introduced in 1961, and was reintroduced by Goldenthal et al. in 2007 under the name of "the fast projection algorithm". Rosen's, Goldenthal et al.'s, and Journée et al.'s work are all closely related but were developed independently. This thesis makes their relationship more apparent.

**Keywords**: elastic network model, Euclidean distance matrix, Gram matrix, matrix manifold optimization, protein structure, positive semidefinite, Riemannian manifold

**Acknowledgements**

I would like to thank my advisor, Professor Forbes Burkowski for introducing me to the field of structural bioinformatics. We spent many hours discussing various research papers which helped me to form the basis of my thesis. He also spent many hours helping me with editing my thesis. I would also like to thank Professor Henry Wolkowicz, who introduced me to semidefinite optimization. In addition, thanks to all my PhD Committee members for their valuable feedback and for investing their time. Professor Wan, Professor Batty, and Professor Karigiannis all provided me with questions which were of central importance during the revision of my thesis. Especially Professor Wan, his detailed questions provided the feedback I needed to reorganize my thesis.

## Dedication

In memory of my father. To my parents.

# Table of Contents

# List of Tables

# List of Figures

# List of Symbols

| | |
|---|---|
| $\mathbb{R}$ | The space of real numbers. |
| $\mathbb{R}^{3n}$ | The space of $3n$ vectors with real entries. |
| $\mathbb{R}^{n\times m}$ | The space of $n \times m$ matrices with real entries. |
| PSD | Positive semidefinite matrix. |
| EDM | Euclidean distance matrix. |
| $n$ | The number of atoms to be modelled. |
| $a$ | Atomic index, $a \in \{1, \ldots, n\}$. |
| $m$ | The number of constraints. |
| $i$ | Constraint index, $i \in \{1, \ldots, m\}$. |
| $a_i$, $b_i$ | Atomic indices for atoms in the $i$-th constraint. |
| $e_a = \begin{pmatrix} \ldots & 1 & \ldots \end{pmatrix}^T$ | An $n \times 1$ vector with zeros everywhere, except at the $a$-th position (also called the one-hot vector). |
| $e_{a,I_3} = \begin{pmatrix} \ldots & I_3 & \ldots \end{pmatrix}^T$ | A $3n \times 3$ matrix of $n$ $3 \times 3$ blocks, where the $a$-block is $I_3$. If row index starts at 0, $I_3$ occupides rows $3a : 3(a+1) - 1$, if row index starts at 1, $I_3$ occupies rows $3a - 2 : 3a$. |
| $p_a$ | Atomic coordinates for the $a$-th atom, $a \in \{1, \ldots, n\}$. |
| $\overrightarrow{p}$ | $\overrightarrow{p} = \begin{pmatrix} {p_1}^T, \ldots, {p_n}^T \end{pmatrix}^T \in \mathbb{R}^{3n}$ is a $3n \times 1$ vector of atomic coordinates. |
| $P$ | $P = \begin{pmatrix} p_1, \ldots, p_n \end{pmatrix}^T$ is an $n \times 3$ matrix of atomic coordinates. |
| $X = PP^T$ | Gram matrix. |
| $\mathcal{D}$ | The set containing pairs of atomic indices where the inter-atomic distance is within a threshold distance (in angstroms). |

| | |
|---|---|
| $d_{ab}$ | Distance between the $a$-th and $b$-th atom, $d_{ab} = \| p_a - p_b \|$. |
| $q_{ab}$ | the quadrance between the $a$-th and $b$-th atom, $q_{ab} = {d_{ab}}^2 = \| p_a - p_b \|^2$. |
| $t$ | Time. May be used *without a value*, such as in normal mode analysis, $\overrightarrow{p}(0)$ denotes the equilibrium conformation and $\overrightarrow{p}(t)$ denotes a perturbation of the $\overrightarrow{p}(0)$ conformation with a normal mode displacement. May *take on integer values*: $\overrightarrow{p}(0), \overrightarrow{p}(1), \overrightarrow{p}(2) \ldots$ when modelling conformational change. |
| $\overrightarrow{p}_t$ | $\overrightarrow{p}_t = \overrightarrow{p}(t)$. The vector $\overrightarrow{p}$ with a subscript is still a $3n \times 1$ vector, the subscript means time. This should not be confused with $p_a \in \mathbb{R}^3$, a $3 \times 1$ vector of the atomic coordinate of the $a$-th atom. |
| $A \circ B$ | Element-wise multiplication of two matrices $A$ and $B$. |
| $\mathbf{S}^n$ | The space of $n \times n$ symmetric matrices. |
| $\mathbf{S}^n_+$ | The cone of $n \times n$ PSD matrices. |
| $\mathbf{S}^{n,r}_+$ | The space of rank $r$ PSD matrices. |
| $\mathbf{S}^{n,n}_+$ or $\mathbf{S}^n_{++}$ | The space of $n \times n$ positive definite matrices. |
| $\mathbf{1}_n$, $\mathbf{1}_{n \times m}$ | $n \times 1$ matrix of all 1's, $m \times n$ matrix of all 1's respectively. |
| $\mathbf{0}_n$, $\mathbf{0}_{n \times m}$ | $n \times 1$ matrix of all 0's, $m \times n$ matrix of all 0's respectively. |
| $\sqrt[\circ]{\mathcal{E}^n}$ | The cone of $n \times n$ matrices whose $(a, b)$-th entry is $d_{ab}$, not a convex cone when $n > 3$. |
| $\mathcal{E}^n$ | The convex cone of $n \times n$ matrices whose $(a, b)$-th entry is $q_{ab}$ (EDMs). |

# Chapter 1

# Introduction

> Everything should be made as simple as possible, but not simpler.
>
> a quote often attributed to Albert Einstein

Mathematical science aims to understand our natural world through the use of mathematical models. The "discovery" of a mathematical model is guided by at least two heuristics: the search for simplicity, and the search for agreements.

The search for simplicity means when multiple mathematical models produce the same result, the simpler model is the preferred one. Simplicity is evident in the very famous transition from the geocentric model of the solar system to the heliocentric model. Figure 1.1 [1] shows a geocentric solar system with complicated epicycles describing the path of the Sun, Mercury, and Venus. This figure is a forever reminder that models of nature needs to be as simple as possible, a principle also emphasized in the quote attributed to Einstein at the beginning of this Chapter.

The search for agreements means when a newer, different, mathematical model of nature is presented, which produces results that agree with prior approaches, the different model often provides more insights into the phenomenon being modelled. An example

[1] Taken from the Wikipedia entry "Deferent and epicycle". James Ferguson (1710-1776), based on similar diagrams by Giovanni Cassini (1625-1712) and Dr Roger Long (1680-1770); engraved for the Encyclopedia by Andrew Bell. [Public domain], via Wikimedia Commons (https://commons.wikimedia.org/wiki/File:Cassini_apparent.jpg).

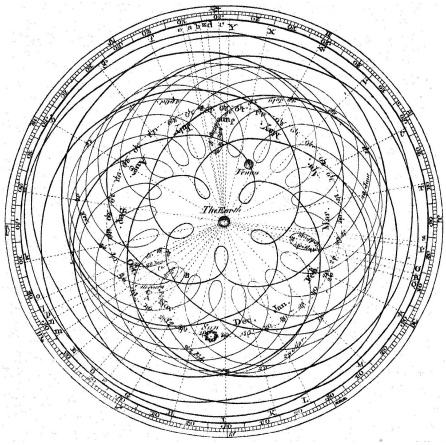


Figure 1.1: The geocentric solar system with epicycles is a complicated model of the solar system.

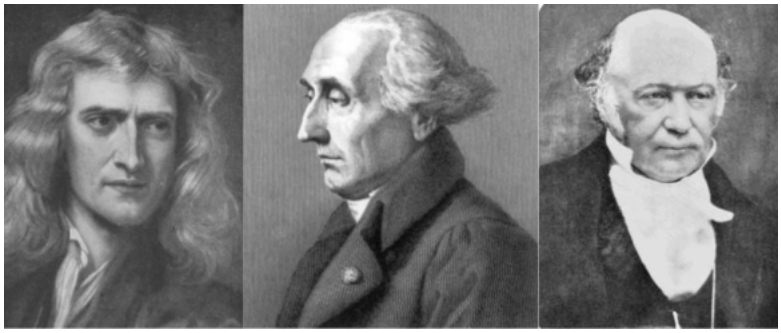


Figure 1.2: Newton (L), Lagrange (C), and Hamilton (R) used different approaches to arrive at classical mechanic's equations of motion.

is that of the agreement between Newton, Lagrange, and Hamilton (see Figure 1.2 [2]), in classical mechanic's equations of motion. A discussion of these three approaches to classical mechanics can be found in for example Taylor, 2005 [73]. Table 1.1 summarizes the different emphasis of Newton, Lagrange, and Hamilton.

[2]Image credits are as follows. Newton's image taken from Wikimedia Commons https://commons.wikimedia.org/wiki/File:SS-newton.jpg. Attribution: Arthur Shuster & Arthur E. Shipley: Britain's Heritage of Science. London, 1917. (A Temple of Worthies) [Public domain], via Wikimedia Commons. Lagrange's image is taken from Wikimedia Commons https://commons.wikimedia.org/wiki/File:Lagrange_portrait.jpg, see also: http://www-history.mcs.st-and.ac.uk/history/PictDisplay/Lagrange.html. Hamilton's image is taken from Wikimedia Commons https://commons.wikimedia.org/wiki/File:William_Rowan_Hamilton_portrait_oval_combined.png.

| Author | Emphasis |
|---|---|
| Newton | Forces and vectors, Cartesian coordinates. |
| Lagrange | The variational principle, coordinate independence. |
| Hamilton | Energy. |

Table 1.1: Newton, Lagrange, and Hamilton's formulation of the classical mechanics equations of motion emphasized different mathematical insights.

The following quote from Taylor[73] page 521-522 provide a flavor of the differences between these three formulations:

> "..the Newtonian form of mechanics...describes the world in terms of forces and acceleration (as related by the second law) and is primarily suited for use in Cartesian coordinates...Lagrangian formulation...is entirely equivalent to Newton's, in the sense that either one can be derived from the other, but the Lagrangian form is considerably more flexible with regard to choice of coordinates...Hamiltonian mechanics leads very naturally from classical mechanics into quantum mechanics."

The above two themes: the search for a simpler model, and the search for a different but agreeable approach in order to gain a deeper insight, will play an important role in this thesis.

The purpose of this thesis is to provide *a second mathematical formulation* for modelling protein structure elastic network models (ENMs). ENMs are abstract representations of a protein structure where inter-atomic interactions are modelled using a Hookean spring potential energy. This thesis suggests removing the square-root in the Hookean potential which leads to a positive semidefinite (PSD) potential. PSD matrices are linearly related to quadrance, the square of distance, which means this model is simple. However, the PSD potential produces results that agree with the Hookean potential (two examples, NMA and interpolating transitions are discussed). The linear relation between PSD matrices and quadrance means a perceptron (neuron) can be used to calculate the PSD potential. This interesting consequence is left as a topic of future research.

## 1.1 Notation

This section introduces some common notations and basic mathematical concepts used throughout this thesis.

The letter $n$ will mainly be used to refer to the number of atoms in a protein structure. The letter $a$ is used to index an atom in a summation, $a \in \{1, \ldots, n\}$. When more than one index is needed, the letter $b$ will be used. When more than two atomic indices are needed, it should be clear from the context which letters are atomic indices.

The letter $m$ is used to represent the number of constraints. This should not be confused with the mass of the $a$-th atom, which is denoted by $m_a$. Since these two contexts are very different, it should be clear from the context whether $m$ is the number of constraints, or the mass of an atom. The letter $i$ is used to index a constraint in a summation.

The matrix $\mathbf{0}_{n\times m}$ is an $n \times m$ matrix of all zeros, if $m = 1$, $\mathbf{0}_n = \mathbf{0}_{n\times 1}$. The matrices $\mathbf{1}_{n\times m}$ and $\mathbf{1}_n$ are defined similarly.

The matrix $I_n$ is the $n \times n$ identity matrix.

The matrix $A^T$ denotes the transpose of $A$. Trace($A$) denote the trace of $A$, which is the sum of the diagonal elements of $A$.

The atomic coordinates of the $a$-th atom is a $3 \times 1$ vector $p_a \in \mathbb{R}^3$. The vector $\overrightarrow{p} \in \mathbb{R}^{3n}$ is the $3n \times 1$ vector formed from stacking the atomic coordinates $p_a$, $a = 1, \ldots, n$.

The $n \times 3$ matrix $P$ is the matrix whose rows are the atomic coordinates transposed, $p_a^T$. The Gram matrix is the matrix $X = PP^T$.

The *rank* of a matrix $X$, denoted rank($X$), is the number of eigenvalues of $X$ greater than zero. Alternatively, the rank of a matrix is the number of columns or rows of the matrix that are linearly independent. Any nonzero vector will have rank 1. Since the atomic coordinates $p_a$ is in 3-D space, the matrix $P$ has three columns that are linearly independent, hence it has rank 3. The Gram matrix $X = PP^T$ also is of rank 3.

The *range space* of an $n \times n$ matrix $A$, denoted range($A$) or $A$, is a vector space whose elements are linear combinations of the columns of $A$. The *null space* of a matrix $A$ is the set null($A$) $= \{x \mid Ax = \mathbf{0}_n\}$. The null space and range space are orthogonal to each other, this is denoted with the symbol "$\perp$", e.g. null($A$)$\perp$range($A$).

The vector

$$e_a = \begin{pmatrix} \ldots & 1 & \ldots \end{pmatrix}^T \in \mathbb{R}^n \tag{1.1}$$

is an $n \times 1$ vector with a 1 at the $a$-th position, and 0 elsewhere. This vector is also called the "one-hot" vector in machine learning.

The matrix

$$e_{a,I_3} = \begin{pmatrix} \dots & I_3 & \dots \end{pmatrix}^T \in \mathbb{R}^{3n\times 3} \tag{1.2}$$

is the $3n \times 3$ matrix of $n$ $3 \times 3$ blocks, where the $a$-th block is $I_3$. If row index starts at 0, $I_3$ occupies rows $3a : 3(a+1) - 1$, if row index starts at 1, $I_3$ occupies rows $3a - 2 : 3a$.

The matrix

$$S_{ab} = (e_{a,I_3} - e_{b,I_3})(e_{a,I_3} - e_{b,I_3})^T \ , \tag{1.3}$$

is a $3n \times 3n$ matrix, also called a *stamp matrix* in [9]. The matrix

$$A_i = (e_{a_i} - e_{b_i})(e_{a_i} - e_{b_i})^T \ , \tag{1.4}$$

is the $n \times n$ constraint matrix for the $i$-th constraint.

The set $\mathbf{S}^n$ is the set of $n \times n$ symmetric matrices. An $n \times n$ matrix $X$ is PSD if it has the property that $v^T X v \geq 0$ for any vector $v \in \mathbb{R}^n$ and $v \neq \mathbf{0}_n$. The eigenvalues of a PSD matrix will all be greater than or equal to zero. The set of $n \times n$ PSD matrices is denoted $\mathbf{S}^n_+$. The set of $n \times n$ positive definite matrices is denoted $\mathbf{S}^n_{++}$. The set of fixed rank PSD matrices will be denoted $\mathbf{S}^{n,r}_+$, where $r$ is the fixed rank. The Gram matrix has rank 3, so it is an element of $\mathbf{S}^{n,3}_+$.

The distance between the $a$-th and $b$-th atoms is denoted $d_{ab}$. The square of the distance is called *quadrance*, it is denoted by $q_{ab}$ [79].

A matrix $D$ whose $ab$-th entry is *quadrance*, $q_{ab}$, is called a *Euclidean distance matrix* (EDM). Note the name does not use quadrance. The set $\mathcal{E}^n$ is the set of matrices whose $ab$-th entry is $q_{ab}$. Let $\sqrt[\circ]{\cdot}$ denote the entry-wise square-root operation. Closely related to an EDM $D$ is the matrix denoted $\sqrt[\circ]{D}$ whose $ab$-th entry is *distance*. The set of $\sqrt[\circ]{D}$ is denoted $\sqrt{\mathcal{E}^n}$. Unlike $\mathcal{E}^n$, $\sqrt{\mathcal{E}^n}$. is not convex for $n \geq 3$, see for example Dattorro [24] Section 6.3.

The set $\mathcal{D}$ (script D) contains atomic index pairs $(a, b)$, whose inter-atomic distance $d_{ab}$ is less than some specified threshold inter-atomic distance, these atoms may be bonded or nonbonded atoms. To avoid double counting pairs, $a < b$ is assumed.

The symbol $\nabla$ means taking the derivative of a mathematical expression. For example, consider a function $f : \mathcal{M} \to \mathbb{R}$, suppose $\mathcal{M} = \mathbb{R}^n$ and let $\overrightarrow{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ be a vector in $\mathbb{R}^n$. If taking the derivative of $f$ gives the gradient, then the gradient is denoted by $\nabla f(\overrightarrow{x})$ or $\mathrm{grad} f(\overrightarrow{x})$ inter-changeably:

$$\mathrm{grad} f(\overrightarrow{x}) = \nabla f(\overrightarrow{x})^T = \left( \frac{\partial f(\overrightarrow{x})}{x_1}, \dots, \frac{\partial f(\overrightarrow{x})}{\partial x_n} \right)^T \in \mathbb{R}^n \ , \tag{1.5}$$

where $\text{grad} f(\overrightarrow{x})$ is the preferred notation for matrix manifolds as seen in [1]. If the gradient of a function requires operations *in addition* to taking the derivative, such as a projection, then $\nabla f(\cdot)$ means taking the derivative *only*, and $\text{grad} f(\cdot)$ denotes the final gradient.

The symbol $\nabla^2$ means to take the derivative twice to arrive at a matrix of second derivatives. For example, when applied to the same function $f$, the symbol $\nabla^2 f$ is the Hessian matrix given by

$$\nabla^2 f(\overrightarrow{x}) = \text{Hess} f(\overrightarrow{x}) = \begin{pmatrix} \frac{\partial^2 f(\overrightarrow{x})}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(\overrightarrow{x})}{\partial x_1 \partial x_n} \\ & \ddots & \\ \frac{\partial^2 f(\overrightarrow{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\overrightarrow{x})}{\partial x_n \partial x_n} \end{pmatrix} . \tag{1.6}$$

The notation $\text{Hess} f(\overrightarrow{x})$ is preferred for matrix manifold optimization algorithms as seen in [1], where the Hessian matrix may also be a *linear operator* instead of a matrix.

## 1.2 Protein Structure Basics

Proteins serve important, life-sustaining functions in organisms, for example: hemoglobin and myoglobin transport oxygen to cells, actin and myosin allow muscles to contract, chaperones are proteins that assist other proteins to fold, and enzymes are proteins that help promote (catalyze) chemical reactions.

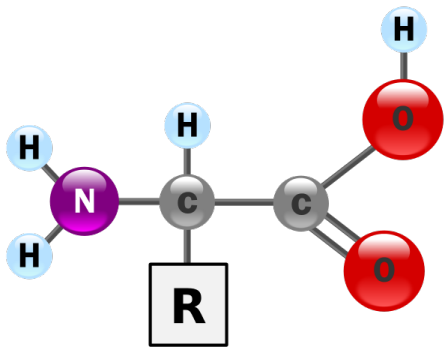



Figure 1.3: A generic amino acid.

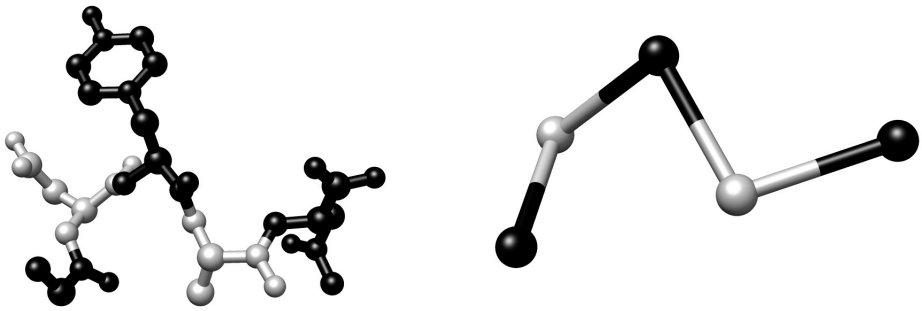
A protein structure is a chain of *amino acids*. Figure 1.3 [3] shows the structure of

[3] Figure taken from wikipedia, https://commons.wikimedia.org/wiki/File:AminoAcidball.svg.

a generic amino acid, it is made up of nitrogen, carbon, oxygen, and hydrogen atoms. Hydrogen atoms are often omitted when visualizing protein structures. The "R" component of the amino acid is called the *side chain*, the carbon atom that the side chain is attached to is called the $\alpha$-carbon, the carbon atom in the side chain that is attached to the $\alpha$-carbon is called the $\beta$-carbon, other atoms in the side chain are also named using Greek letter $\gamma$, $\delta$, etc.. All these atoms are connected by a *covalent bond*, which is a connection resulting from sharing of electrons.

The *backbone* of the protein is made up of the atoms not in the side chain, these are the nitrogen, $\alpha$-carbon, carbon, and oxygen atoms in the amino acid.

When two amino acids join, one water molecule is lost, therefore, each amino acid in a chain of amino acids is called a *residue*. A *coarse-grained* model of a protein structure is a model that represents each residue by its $\alpha$-carbon. Figure 1.4 a) shows an example of a short chain of amino acid residues, Figure 1.4 b) shows the corresponding *coarse-grained* representation, which is a chain of $\alpha$-carbons. A protein may have several hundred residues to tens of thousands of residues (titin has about 27000 residues, see page 78 of [70]).



(a) All atoms representation (balls are atoms).

(b) $\alpha$-carbon backbone representation (balls are atoms).

Figure 1.4: The polypeptide Glycyl-Aspartyl-Tyrosyl-Alanyl-Asparagine (GDYAN) visualized in UCSF Chimera.

A *conformation* of a protein is a particular three-dimensional arrangement of atoms, see for example page 76 of [35], or page 27 of [82] and the glossary therein. A particular conformation changes to another conformation if no covalent bonds are broken. The conformational change is due to a change in *dihedral angles*, also called *torsion angles*, which is the angle between two planes. There are two dihedral angles around a $\alpha$-carbon. The $\phi$ dihedral angle is defined as the angle between:

- the plane occupied by the carbon atom of the previous residue, plus the nitrogen atom and $\alpha$-carbon atom of the current residue and
- the plane occupied by nitrogen atom, the $\alpha$-carbon atom, and the carbon atom of the current residue.

The $\psi$ dihedral angle is defined as the angle between:

- the plane occupied by the nitrogen atom, the $\alpha$-carbon atom, and the carbon atom of the current residue and
- the plane occupied by the $\alpha$-carbon atom and the carbon atom of the current residue, plus the nitrogen atom of the next residue.

When modelling a protein's conformational change, the most obvious generalized coordinates to use are the dihedral angles mentioned above.

The *semi-empirical potential*, see for example [75, 70, 70, 13, 14, 60], is an example of a potential energy that models changes in dihedral angles. Let $E_p$ denote the semi-empirical potential, the following equation is one of its common expressions:

$$\begin{aligned} E_p =& \frac{1}{2} \sum_{(a,b)\in \text{bonds}} K_b (d_{ab} - \widehat{d}_{ab})^2 + \frac{1}{2} \sum_{(a,b,c)\in \text{angles}} K_\theta (\theta_{abc} - \widehat{\theta}_{abc})^2 \\ &+ \frac{1}{2} \sum_{(a,b,c,d)\in \text{dihedrals}} \sum_n K_\omega [1 + \cos(n_{abcd}\omega_{abcd} - \rho_{abcd})] \\ &+ \sum_{(a,b)\in\mathcal{D}} \left[ \left( \frac{C_{ab}}{d_{ab}^{12}} - \frac{D_{ab}}{d_{ab}^{6}} \right) + \frac{Q_a Q_b}{\epsilon_{ab} d_{ab}} \right] . \end{aligned} \tag{1.7}$$

The first three terms are for *bonded interactions*, they measure the distortion in bond lengths, bond angles, and dihedral angles respectively. The terms $d_{ab}$ represents the bond length between the $a$-th and $b$-th bonded atoms, $\theta_{abc}$ represents the bond angle formed by the $a$-th $b$-th and $c$-th atoms, and $\omega_{abcd}$ represents the dihedral angle between the plane occupied by the $a$-th $b$-th and $c$-th atoms, and the plane occupied by the $b$-th $c$-th and $d$-th atoms. The value $n_{abcd}$ is the periodicity of the dihedral angle, the term $\rho_{abcd}$ is a phase offset for the dihedral angle. The terms $\widehat{d}_{ab}$, $\widehat{\theta}_{abc}$, are equilibrium bond lengths and bond angles. The values $K_b$, $K_\theta$, $K_\omega$ are force constants.

The last summation is for *non-bonded interactions*, they are the Lennard Jones interaction and the electrostatic interaction respectively. The constants $C_{ab}$ and $D_{ab}$ are specific to the

interacting pair. The $a$-th and $b$-th atoms have charges denoted by $Q_a$ and $Q_b$ respectively, and $\epsilon_{ab}$ in the electrostatic interaction term is the dieletric constant. These nonbonded interactions are confined to atoms within a certain threshold distance.

## 1.3 The Defects of Using Dihedral Angles for Studying Protein Dynamics

Researchers have noted several defects in modelling protein conformational changes using dihedral angles, this Section reviews some of these defects.

According to Crippen [19], potential energies that are a function of dihedral angles have numerous local minima whereas potential energies that are a function of distance are a lot simpler.

Plantenga [64] has stated that using dihedral angles to study protein dynamics causes ill-conditioning because "a slight rotation of the dihedral moves neighboring atoms only a slight distance, but causes atoms farther down the chain to shift much larger distances; thus the potential energy can be highly sensitive to small changes in dihedral variables."

Neumaier [60] has pointed out further disadvantages of dihedral angles. Firstly, they are undefined if the bond angle is 180°, "[a]lthough equilibrium angles are typically far away from 180°, this is an important defect in global applications; for example, it ruins (or produces unpredictable results in) any local optimization routine if one of the angles in an intermediate calculation happens to come close to 180°. The second defect of dihedral angles as described by Neumaier is physically equivalent angles lead to different values of the potential energy. This leads to using trigonometric functions of angles rather than the angles themselves.

Kim [37] has stated that interpolating dihedral angles often leads to an infeasible transition pathway, and therefore proposed to interpolate inter-atomic distances rather than dihedral angles, this formulation is called "elastic network interpolation" (ENI).

## 1.4 Elastic Network Model and the Hookean Potential

ENMs focus on modelling the change in distance between atoms, rather than dihedral angles, and overcome many of the disadvantages of using angles.

In this modelling approach, the interaction between non-bonded atoms is modelled using a *Hookean potential*. This is equivalent to assuming these atoms are connected by springs.

ENMs further simplify the model by assuming a protein is represented as a chain of $\alpha$-carbons, as shown in Figure 1.4 b).

The Hookean potential is a *multi-purpose* potential. It has been used in normal mode analysis, to interpolate transitional conformations, and as shown by Tekpinar and Zheng [74] can also be used to avoid atomic collision.

Normal mode analysis (NMA) is an efficient tool for studying protein structure flexibility. It models the oscillation of the protein structure's atomic coordinates around an equilibrium conformation. It was shown by (Tirion, 1996 [75]) that a Hookean potential can reproduce the large amplitude atomic normal mode displacements, which were previously found from more the more complicated semi-empirical potentials.

NMA using the semi-empirical potential requires an energy minimization step to find a protein conformation at an energy minimum, e.g. optimal dihedral angles are needed. The Hookean potential energy can be used to find the normal modes of a protein more efficiently than the semi-empirical potential because the initial protein conformation is *already* at an energy minimum as measured by the Hookean potential, and any energy minimization of this initial structure is not required.

While the Hookean potential in NMA is concerned with deviations about an equilibrium protein structure, they have also been employed to model non-equilibrium dynamics, such as the modelling of transitional conformations between a given beginning and an ending conformation.

Protein macromolecular machines such as enzymes, channels, and pumps need to change their conformation to perform their functions. For example, many proteins are known to have "open" and "close" conformations. Understanding how the protein transitions from the "open" to the "close" conformations, or vice-versa, is very important for understanding the relationship between structure and function. Transitional conformations are relatively difficult to obtain compared with long-lived stable functional states, whose structures can be determined using X-ray crystallography or NMR.

**Definition 1.4.1** (The Transitional Conformations Problem)**.** *Let the atomic coordinates of the initial protein conformation at time* $t_{initial} = 0$ *be given by*

$$p_1(0), \ldots, p_n(0) \ ,$$

*and the final atomic coordinates be given by*

$$p_1(t_{final}), \ldots, p_n(t_{final}) \ .$$

*The* transitional conformations problem *is to find the intermediate conformations, where $t$ takes on integer values $t = 1, 2, \ldots, t_{final} - 1$ . The $t$-th intermediate conformation has atomic coordinates*

$$p_1(t), \ldots, p_n(t) \ ,$$

*Each intermediate conformation is found incrementally; that is, $p_a(t)$ is assumed to be known, and $p_a(t+1)$ is to be found.*

Examples of using the Hookean potential used to model the Transitional Conformations Problem include the formulation of Kim et al. called elastic network interpolation (ENI) [37, 40, 41, 38], Tekpinar and Zheng [74], and Bahar et al. [22]. Normal mode guided transition pathways have been explored in [47]. Hybrid ENMs have been used to model the transition pathways of rigid groups of atoms [71]. Interpolated ENM (iENM) have been shown to model the local conformational change of active sites before a global conformational change [74, 81].

Kim et al. [42, 37] provided a comparison between molecular dynamics (MD) simulation and ENI was made to evaluate the reasonableness of the intermediate protein conformations generated by ENI using the 16S ribosomal RNA. Their findings showed ENI has smoothed

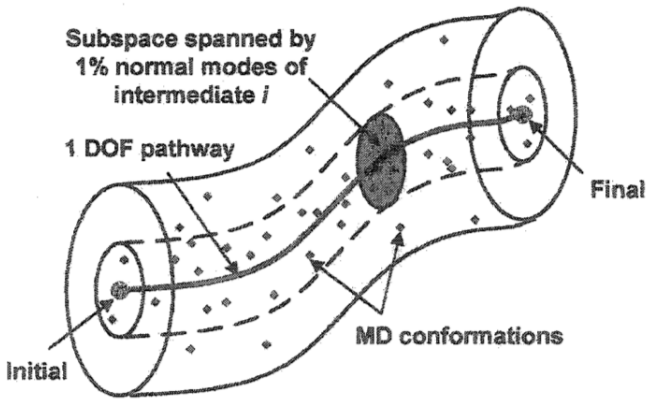



Figure 1.5: ENI produces structures that smooth out the structures generated from molecular dynamics. Figure 4.20 of (Kim, 2004[37]).

out the MD results, and generated an "average" MD pathway using much less computational

time than MD. This finding is summarized in Figure 9 of [42] and Figure 4.20 of [37], reproduced in Figure 1.5.

In order to show that the intermediate conformations generated by iENM are reasonable, three interpolation examples of ATP-driven conformational changes were presented in [74]. These examples showed that iENM generates intermediate structures where the local conformational change around an active site precedes global change. This result supports the idea that the active site drives the conformational change of molecular machines, and shows the iENM double-well potential pathway is a realistic model for conformational changes.

In Chapter 2, Tirion's Hookean potential, Kim et al.'s ENI Hookean potential, and Tekpinar and Zheng's iENM will be reviewed.

## 1.5 The Defects of Modelling ENMs on $\mathbb{R}^{3n}$

Currently, modelling ENMs is formulated in $\mathbb{R}^{3n}$; this formulation has a number of mathematical defects, these are listed in Table 1.2. Each of these defects will be address in this thesis. This thesis shows using $\mathbf{S}_{+}^{n,3}$ to model ENMs can remove all these defects.

## 1.6 Thesis Outline and Contributions

The first two chapters of this thesis provide an overview of needed background information.

- Chapter 2 introduces ENMs as they are currently modelled in $\mathbb{R}^{3n}$ with a Hookean potential energy. NMA and interpolation of transitional conformations are used as application examples of where the Hookean potential has appeared. This is followed by a discussion on enforcing constraints in $\mathbb{R}^{3n}$. Rosen introduced a "correction" to the constraint manifold as early as 1961 [67], therefore the "step and project" paradigm was already known at least since then. However, Ryckaert et al. did not apply this paradigm in the development of the SHAKE algorithm published in 1977 [68], and Goldenthal et al. [30] modified Ryckaert et al.'s formulation to adhere to the step and project paradigm in 2007, but they were also unaware of Rosen's work. The delay in using step and project to enforce distance constraints is a result of $\mathbb{R}^{3n}$ de-emphasizing introducing abstract concepts first, and then applying these concepts to different applications. Matrix manifold optimization algorithms do not have this

| **Considerations** | Defects of the current approach using $\mathbb{R}^{3n}$ | Thesis reference |
|---|---|---|
| Minimizing the potential is well-posed | No, problem is ill-posed. | Section 2.7, 5.5 |
| Distance or quadrance in potential | No preference. | Section 4.4 |
| Number of constraints for a rigid group of $n_q$ atoms | $3n_q - 6$, see [64] depends on $n_q$, i.e. $O(n_q)$. | Secton 5.6 |
| Rotation of rigid groups of atoms | Needs rotation matrices, and Lie group theory, additional complexity. [17]). | Section C.6.5 |
| Conservation of linear momentum | Additional constraint. | See discussions on ill-posedness. |
| Enforcing constraints | *Both* implicit and explicit treatment of the constraint force has been proposed. | Section 2.8.2, 6.3.5 |
| Redundancy | Yes. Mass matrix $M$ repeats mass of each atom three times, $e_{a,I_3}$ repeats the 1 three times. | Equation (2.19), (1.2), (5.14), (1.1) |
| Apply abstract concepts to different applications | No. Fast Projection was already presented by Rosen in 1961 [66] for general nonlinear constraints, but re-introduced by Goldenthal et al. [30] for distance constraints. | Section 2.8.1, 2.8.2, 2.8.5, Chapter 3 |

Table 1.2: The defects of modelling ENMs using $\mathbb{R}^{3n}$ to be addressed in this thesis.

defect, as shown by Absil et al. [1], these algorithms *start with abstract concepts*, and applies these concepts to *different* matrix manifolds.

The correction proposed by Rosen, and Fast Projection proposed by Goldenthal et al. is called a *retraction* in matrix manifold language, and *by definition* a retraction takes an unconstrained step followed by a correction to the matrix manifold. Therefore when implementing the retraction for enforcing constraints, implicitly treating the constraint-maintaining force is in fact the *only* option.

- Chapter 3 will present the mathematics for unconstrained optimization on the rank 3 PSD matrix manifold. This Chapter shows how abstract concepts from differential geometry are applied to $\mathbf{S}_+^{n,3}$ and is based on the publication by Mishra et al. [58]. For applications to other matrix manifolds, see Absil et al. [1].

Thereafter begins the contributions of this thesis:

- Firstly, Chapter 4 presents the PSD potential energy in the context of normal mode analysis because it was the same application used by Tirion [75] to introduce the Hookean potential. This Chapter also discusses why the PSD potential energy is different from the Hookean potential. The PSD potential prefers quadrance over distance because this choice leads to the simpler model on $\mathbf{S}_+^{n,3}$. Note that the PSD potential does not offer benefits in additional to the Hookean potential for NMA, rather NMA provides the context for introducing the PSD potential.

- The introduction of the PSD potential motivates the equations of motion to also be formulated on $\mathbf{S}_+^{n,3}$, this is the topic of Chapter 5. This Chapter also explains why the constraint for enforcing the conservation of linear momentum is not required on $\mathbf{S}_+^{n,3}$, and also why the number of constraints to keep a group of $n_q$ atoms rigid is independent of $n_q$, unlike for $\mathbb{R}^{3n}$.

- Chapter 6 describes the geometric objects required for constrained optimization on $\mathbf{S}_+^{n,3}$, these objects include the gradient, Hessian, projection onto the tangent space, and a retraction. The presentation in this Chapter closely follows the discussion by Journée et al. [34]. The agreements between enforcing constraints on $\mathbb{R}^{3n}$ and $\mathbf{S}_+^{n,3}$ are also discussed. These agreements are significant for the following reasons:
  - modelling protein ENMs on $\mathbf{S}_+^{n,3}$ is *not* more complicated than on $\mathbb{R}^{3n}$,
  - $\mathbf{S}_+^{n,3}$ can build on $\mathbb{R}^{3n}$, and not "reinvent the wheel",
  - additional mathematical insight can be gained by introducing $\mathbf{S}_+^{n,3}$.

- Chapter 7 uses the transitional conformations problem as an example to show how ENMs can be modelled on $\mathbf{S}_+^{n,3}$. An object-oriented implementation using `python` is given in the Appendix A.

Chapter 8 is the concluding Chapter, a summary of how modelling ENMs using $\mathbf{S}_+^{n,3}$ removes the corresponding defects in $\mathbb{R}^{3n}$ is provided. Directions of future research are also proposed.

# Chapter 2

# Elastic Network Models on $\mathbb{R}^{3n}$

## 2.1 Introduction

This Chapter introduces the mathematics for modelling protein structure ENMs as it is currently formulated on $\mathbb{R}^{3n}$.

An ENM is an abstraction of a protein structure where nonbonded inter-atomic interactions are modelled using a Hookean potential energy. The assumption made in this thesis is that the ENM models only the $\alpha$-carbons of a protein, as show in Figure 1.4 b). Because all atoms are the same, they have the same atomic mass, therefore without loss of generality, the mass of an $\alpha$-carbon is assumed to be 1 (unit mass). If the mass is not 1, the equations of motion can always be scaled to make the mass 1. In additional, inter-atomic distance does not change when atoms are rotated, and therefore the Hookean potential is not affected by gravity, hence mass is not a major modelling factor.

The generic Hookean potential is introduced first. Thereafter, the application of the Hookean potential to NMA as presented by Tirion [75] is reviewed. This is followed by a review of the application of this potential to the transitional conformations problem. Enforcing constraints on $\mathbb{R}^{3n}$ is then discussed.

In this Chapter, the parameter $t$ represents *time* abstractly, in that it is used to distinguish different protein conformations from each other, no specific time units are assumed.

When discussing NMA, the time is assumed to be a continuous variable, $t = 0$ for the equilibrium conformation, and $t \neq 0$ for the conformation perturbed by a normal mode displacement (no specific value for $t$ is needed).

When discussion the transitional conformations problem, $t$ is assume to take on *discrete* values, $t = 0, 1, \ldots$. The initial conformation is assumed to have $t = t_{initial} = 0$ and the final conformation is assumed to have $t = t_{final} = 100$, with 100 intermediate conformations generated in between.

The Hookean potential assumes the protein conformation's atomic coordinates are arranged as a $3n \times 1$ vector:

$$\overrightarrow{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \in \mathbb{R}^{3n} \ .$$

The atomic coordinates for a particular time $t$ conformation is denoted as:

$$\overrightarrow{p}_t = \overrightarrow{p}(t) = \begin{pmatrix} p_1(t) \\ \vdots \\ p_n(t) \end{pmatrix} \in \mathbb{R}^{3n} \ .$$

## 2.2 The Generic Hookean Potential

Define the function $E : \mathbb{R}^3 \to \mathbb{R}$:

$$E(x) = \frac{1}{2}\left(\sqrt{x^T x} - \widehat{d}_{ab}\right)^2 = \frac{1}{2}\left(\parallel x \parallel - \widehat{d}_{ab}\right)^2 \ .$$

The gradient and Hessian of this function is discussed in Appendix B.

Recall $e_{a,I_3}$ was defined in Equation (1.2). The generic *pairwise* Hookean potential energy describing the interaction between the $a$-th and $b$-th atom is given by substituting

$$x = (e_{a,I_3} - e_{b,I_3})^T \overrightarrow{p} = p_a - p_b \ ,$$

in $E(x)$:

$$E(p_a - p_b) = \frac{1}{2}\left(\sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} - \widehat{d}_{ab}\right)^2 = \frac{1}{2}\left(\parallel p_a - p_b \parallel - \widehat{d}_{ab}\right)^2 \ . \tag{2.1}$$

The stamp matrix $S_{ab}$ was defined in Equation (1.3) . The distance $\widehat{d}_{ab}$ is a "reference distance" defined based on the problem.

The gradient and Hessian matrix of the generic pairwise Hookean potential have been discussed by for example Kim [37]. They are found by using the formulas in Appendix B

with the substitution $x = p_a - p_b$. The $3 \times 1$ gradient vector $\nabla E(p_a - p_b)$ is given by:

$$\begin{aligned} \nabla E(p_a - p_b) &= \left( \parallel p_a - p_b \parallel - \widehat{d}_{ab} \right) \frac{(e_{a,I_3} - e_{b,I_3})^T \overrightarrow{p}}{\parallel p_a - p_b \parallel} \\ &= \left( 1 - \frac{\widehat{d}_{ab}}{\parallel p_a - p_b \parallel} \right) (p_a - p_b) \ , \end{aligned} \tag{2.2}$$

The $3 \times 3$ Hessian matrix of $E(p_a - p_b)$ is:

$$\nabla^2 E(p_a - p_b) = I_3 - \frac{\widehat{d}_{ab}}{\parallel p_a - p_b \parallel} Q(p_a - p_b) \in \mathbb{R}^{3 \times 3} \ , \tag{2.3}$$

where:

$$Q(x) = I_3 - \frac{xx^T}{\parallel x \parallel^2} \ . \tag{2.4}$$

Note that:

$$\nabla^2 E(p_a - p_b) = \nabla^2 E(p_b - p_a) \ .$$

The *total* Hookean potential energy for the entire protein structure is a function $E_H : \mathbb{R}^{3n} \to \mathbb{R}$ given by:

$$E_H(\overrightarrow{p}) = \sum_{(a,b) \in \mathcal{D}} w_{ab} E(p_a - p_b) \ . \tag{2.5}$$

The scalar $w_{ab}$ is the weight for the interaction pair $(a, b)$, which may be the same for all interacting pairs, further, $w_{ab} = 1$ can be assumed for all $(a, b) \in \mathcal{D}$, for example see [39, 37]. The summation is taken over the set $\mathcal{D}$ containing atom pairs that are within a threshold distance apart, they can be bonded or nonbonded pairs. The gradient $\nabla E_H(\overrightarrow{p})$ is a $3n \times 1$ vector given by:

$$\nabla E_H(\overrightarrow{p}) = \begin{pmatrix} \nabla E_H(\overrightarrow{p})_{[1]} \\ \vdots \\ \nabla E_H(\overrightarrow{p})_{[n]} \end{pmatrix} \in \mathbb{R}^{3n \times 1} \ , \tag{2.6}$$

where:

$$\begin{aligned}\nabla E_H(\overrightarrow{p})_{[a]} &= -\sum_{k=1}^{a-1} w_{ka}\nabla E(p_k - p_a) + \sum_{k=a+1}^{n} w_{ak}\nabla E(p_a - p_k) \\ &= \sum_{\substack{k=1 \\ k\neq a}}^{n} w_{ak}\nabla E(p_a - p_k) \quad a = 1, \ldots n \ .\end{aligned} \tag{2.7}$$

The notation $[a]$ is used to emphasize $E_H(\overrightarrow{p})_{[a]}$ is a $3\times 1$ vector, not a scalar. In other words, whenever $p_a$ appears first in $(p_a - p_k)$, derivative with respect to $p_a$ is a positive contribution from the pairwise potential, whenever $p_a$ appears as the second term in $(p_k - p_a)$, the derivative with respect to $p_a$ is a negative contribution of the pairwise potential. It is straighforward to show:

$$\nabla E_H(\overrightarrow{p})^T\overrightarrow{\delta} = \overrightarrow{\delta}^T\nabla E_H(\overrightarrow{p}) = \sum_{a<b} w_{ab}\nabla E(p_a - p_b)^T(\delta_a - \delta_b) \ , \tag{2.8}$$

The Hessian matrix, $\nabla^2 E_H(\overrightarrow{p})$, of the total Hookean potential has a Laplacian structure. For example, consider the case of just four atoms, $n = 4$. When all the atoms are neighbours, $\nabla^2 E_H(\overrightarrow{p})$ is given by the following $12 \times 12$ matrix of $3 \times 3$ blocks:

$$\nabla^2 E_H(\overrightarrow{p}) = \begin{pmatrix} \nabla^2 E_H(\overrightarrow{p})_{[11]} & -w_{12}\nabla^2 E(p_1 - p_2) & -w_{13}\nabla^2 E(p_1 - p_3) & -w_{14}\nabla^2 E(p_1 - p_4) \\ -w_{12}\nabla^2 E(p_1 - p_2) & \nabla^2 E_H(\overrightarrow{p})_{[22]} & -w_{23}\nabla^2 E(p_2 - p_3) & -w_{24}\nabla^2 E(p_2 - p_4) \\ -w_{13}\nabla^2 E(p_1 - p_3) & -w_{23}\nabla^2 E(p_2 - p_3) & \nabla^2 E_H(\overrightarrow{p})_{[33]} & -w_{34}\nabla^2 E(p_3 - p_4) \\ -w_{14}\nabla^2 E(p_1 - p_4) & -w_{24}\nabla^2 E(p_2 - p_4) & -w_{34}\nabla^2 E(p_3 - p_4) & \nabla^2 E_H(\overrightarrow{p})_{[44]} \end{pmatrix} .$$

The off-diagonal blocks are all the negative contributions of the Hessian of the pairwise Hookean potentials.

The diagonals are the sum of all the positive contributions defined as follows:

- $\nabla^2 E_H(\overrightarrow{p})_{[11]} = w_{12}\nabla^2 E(p_1 - p_2) + w_{13}\nabla^2 E(p_1 - p_3) + w_{14}\nabla^2 E(p_1 - p_4)$
- $\nabla^2 E_H(\overrightarrow{p})_{[22]} = w_{12}\nabla^2 E(p_1 - p_2) + w_{23}\nabla^2 E(p_2 - p_3) + w_{24}\nabla^2 E(p_2 - p_4)$
- $\nabla^2 E_H(\overrightarrow{p})_{[33]} = w_{13}\nabla^2 E(p_1 - p_3) + w_{23}\nabla^2 E(p_2 - p_3) + w_{34}\nabla^2 E(p_3 - p_4)$
- $\nabla^2 E_H(\overrightarrow{p})_{[44]} = w_{14}\nabla^2 E(p_1 - p_4) + w_{24}\nabla^2 E(p_2 - p_4) + w_{34}\nabla^2 E(p_3 - p_4)$

The sign for the pairwise Hessian $\nabla^2 E(p_a - p_b)$ is positive or negative for the same reason the sign for the pairwise gradient $\nabla E(p_a - p_b)$ is positive or negative, i.e. whether the

derivative is with respect to $p_a$ or $p_b$. For a vector $\overrightarrow{\delta} = (\delta_1^T, \delta_2^T, \delta_3^T, \delta_4^T)^T \in \mathbb{R}^{12}$, it is straightforward to show:

$$\overrightarrow{\delta}^T \nabla^2 E_H(\overrightarrow{p})\, \overrightarrow{\delta} = \sum_{(a,b)\in\mathcal{D}} (\delta_a - \delta_b)^T w_{ab} \nabla^2 E(p_a - p_b)(\delta_a - \delta_b) \; .$$

where for this simple example, $\mathcal{D}$ is all possible pairs of the 4 atoms such that $a < b$ to avoid double counting pairs. For a general $n$, the off-diagonal blocks are given by

$$\nabla^2 E_H(\overrightarrow{p})_{[ab]} = \begin{cases} -w_{ab} \nabla^2 E(p_a - p_b) & \text{if } (a,b) \in \mathcal{D} \\ \mathbf{0}_{3\times 3} & \text{if } (a,b) \notin \mathcal{D} \end{cases}$$

where the square bracket $[ab]$ is used to emphasize this is a $3 \times 3$ matrix and not a scalar. The diagonal blocks are given by:

$$\begin{aligned} \nabla^2 E_H(\overrightarrow{p})_{[aa]} &= \sum_{k=1}^{a-1} w_{ka} \nabla^2 E(p_k - p_a) + \sum_{k=a+1}^{n} w_{ak} \nabla^2 E(p_a - p_k) \\ &= \sum_{\substack{k=1 \\ k\neq a}}^{n} w_{ka} \nabla^2 E(p_k - p_a) \end{aligned}$$

The following convention is used:

$$\sum_{k=a}^{b} x_k = 0 \quad \text{if } a > b \; . \tag{2.9}$$

Similar to the $n = 4$ case, it is straightforward to show for any $n$:

$$\overrightarrow{\delta}^T \nabla^2 E_H(\overrightarrow{p})\, \overrightarrow{\delta} = \sum_{(a,b)\in\mathcal{D}} (\delta_a - \delta_b)^T w_{ab} \nabla^2 E(p_a - p_b)(\delta_a - \delta_b) \; . \tag{2.10}$$

The second order expansion of $E_H(\overrightarrow{p})$ for a small displacement $\overrightarrow{\delta} \in \mathbb{R}^{3n}$ is given by:

$$E_H(\overrightarrow{p} + \overrightarrow{\delta}) \approx \frac{1}{2} \overrightarrow{\delta}^T \nabla^2 E_H(\overrightarrow{p})\, \overrightarrow{\delta} + \overrightarrow{\delta}^T \nabla E_H(\overrightarrow{p}) + E_H(\overrightarrow{p}) \; . \tag{2.11}$$

Equation (2.11) can also be expressed in terms of the gradient and Hessian of the pairwise

potential using Equations (2.10) and (2.8):

$$\begin{aligned}
&\frac{1}{2}\overrightarrow{\delta}^T\nabla^2 E_H(\overrightarrow{p})\overrightarrow{\delta} + \overrightarrow{\delta}^T\nabla E_H(\overrightarrow{p}) + E_H(\overrightarrow{p}) \\
=&\frac{1}{2}\sum_{(a,b)\in\mathcal{D}}(\delta_a-\delta_b)^T\nabla^2 E(p_a-p_b)(\delta_a-\delta_b) + \sum_{(a,b)\in\mathcal{D}}\nabla E(p_a-p_b)^T(\delta_a-\delta_b) + \sum_{(a,b)\in\mathcal{D}}E(p_a-p_b) \\
=&\frac{1}{2}\sum_{(a,b)\in\mathcal{D}}\left[(\delta_a-\delta_b)^T\nabla^2 E(p_a-p_b)(\delta_a-\delta_b) + \nabla E(p_a-p_b)^T(\delta_a-\delta_b) + E(p_a-p_b)\right] ,
\end{aligned} \tag{2.12}$$

## 2.3 The Hookean Potential and Normal Mode Analysis

This section describes the Hookean potential introduced by Tirion et al. for NMA[75].

Let $\overrightarrow{p}(0)$ denote the vector of equilibrium atomic coordinates, these are the atomic coordinates stored in the protein data bank (PDB) files. Let the displaced atomic coordinates at a time $t > 0$, $t \in \mathbb{R}$, be $\overrightarrow{p}(t) = \overrightarrow{p}(0) + \overrightarrow{\delta}(t)$. The $3n \times 1$ vector $\overrightarrow{\delta}(t)$:

$$\overrightarrow{\delta}_t = \overrightarrow{\delta}(t) = \left(\delta_1(t)^T, \ldots, \delta_n(t)^T\right)^T \in \mathbb{R}^{3n} \quad \delta_a(t) \in \mathbb{R}^3 \quad a = 1, \ldots, n , \tag{2.13}$$

represents a small displacement. The pairwise NMA Hookean potential to model the interaction between the $a$-th and $b$-th atoms is given by:

$$\begin{aligned}
E_{ab,\text{H-NMA}}(p_a(t)-p_b(t)) &= \frac{1}{2}\left(\sqrt{\overrightarrow{p}(t)^T S_{ab}\overrightarrow{p}(t)} - \|p_a(0)-p_b(0)\|\right)^2 \\
&= \frac{1}{2}\left(\|p_a(t)-p_b(t)\| - \|p_a(0)-p_b(0)\|\right)^2 \\
&= \frac{1}{2}\left(\|p_a(t)-p_b(t)\| - d_{ab}(0)\right)^2 .
\end{aligned} \tag{2.14}$$

The total potential, $E_{\text{H-NMA}}(\overrightarrow{p}(t))$, is the sum of the pairwise potentials:

$$E_{\text{H-NMA}}(\overrightarrow{p}(t)) = \sum_{(a,b)\in\mathcal{D}} E_{ab,\text{H-NMA}}(p_a(t)-p_b(t)) , \tag{2.15}$$

This is the generic Hookean potential with the reference distance $\widehat{d}_{ab}$ set to $d_{ab}(0)$ and the atomic coordinates $\overrightarrow{p}$ set to $\overrightarrow{p}(t)$.

NMA requires using the second order approximation of the total potential near $\overrightarrow{p}(0)$ in the equations of motion to arrive at a closed form solution to the protein's motion. Consider the pairwise Hookean potential first, its second order approximation is given by:

$$\begin{aligned}
&E_{ab,\text{H-NMA}}((p_a(0) - p_b(0)) + (\delta_a(t) - \delta_a(t))) \\
&\approx \frac{1}{2}(\delta_a(t) - \delta_b(t))^T \nabla^2 E_{ab,\text{H-NMA}}(p_a(0) - p_b(0))(\delta_a(t) - \delta_b(t)) \\
&+ (\delta_a(t) - \delta_b(t))^T \nabla E_{ab,\text{H-NMA}}(p_a(0) - p_b(0)) \\
&+ E_{ab,\text{H-NMA}}(p_a(0) - p_b(0)) \ .
\end{aligned}$$

Note that

$$\|p_a(0) - p_b(0)\| - d_{ab}(0) = 0$$

The constant term of the second order expansion is:

$$E_{ab,\text{H-NMA}}(p_a(0) - p_b(0)) = \frac{1}{2}\left(\|p_a(0) - p_b(0)\| - d_{ab}(0)\right)^2 = 0 \ .$$

Applying Equation (2.2) to Equation (2.14) gives:

$$\begin{aligned}
\nabla E_{ab,\text{H-NMA}}(p_a(0) - p_b(0)) &= (\parallel p_a(0) - p_b(0) \parallel - d_{ab}(0)) \frac{(e_{a,I_3} - e_{b,I_3})^T \overrightarrow{p}(0)}{\parallel p_a(0) - p_b(0) \parallel} \\
&= 0 \times 1 = 0 \ .
\end{aligned}$$

Therefore, only the second order term is nonzero in the second order expansion of the pairwise Hookean potential. Applying Equation (2.3) to Equation (2.14) gives:

$$\begin{aligned}
\nabla^2 E_{ab,\text{H-NMA}}(p_a(0) - p_b(0)) &= I_3 - \frac{d_{ab}(0)}{\parallel p_a(0) - p_b(0) \parallel} Q(p_a(0) - p_b(0)) \\
&= I_3 - Q(p_a(0) - p_b(0)) \\
&= I_3 - I_3 + \frac{(p_a(0) - p_b(0))(p_a(0) - p_b(0))^T}{(p_a(0) - p_b(0))^T (p_a(0) - p_b(0))} \\
&= \frac{(p_a(0) - p_b(0))(p_a(0) - p_b(0))^T}{(p_a(0) - p_b(0))^T (p_a(0) - p_b(0))} \ .
\end{aligned} \tag{2.16}$$

Only the second order term is nonzero for the NMA Hookean potential.

$$E_{ab,\text{H-NMA}}((p_a(0)-p_b(0))+(\delta_a(t)-\delta_b(t))) \approx \frac{1}{2}(\delta_a(t)-\delta_b(t))^T\nabla^2 E_{ab,\text{H-NMA}}(\overrightarrow{p}(0))(\delta_a(t)-\delta_b(t)) \ .$$

Summing the second order approximation for all pairs in $\mathcal{D}$ gives the second order expansion for NMA:

$$E_{\text{H-NMA}}(\overrightarrow{p}(0)+\overrightarrow{\delta}(t)) \approx \frac{1}{2}\sum_{(a,b)\in\mathcal{D}}(\delta_a(t)-\delta_b(t))^T\nabla^2 E_{ab,\text{H-NMA}}(p_a(0)-p_b(0))(\delta_a(t)-\delta_b(t)) \ , \tag{2.17}$$

Equation (2.17) can be expressed in matrix form:

$$E_{\text{H-NMA}}(\overrightarrow{p}(0)+\overrightarrow{\delta}(t)) \approx \frac{1}{2}\overrightarrow{\delta}(t)^T\nabla^2 E_{\text{H-NMA}}(\overrightarrow{p}(0))\,\overrightarrow{\delta}(t) \ , \tag{2.18}$$

where $\nabla^2 E_{\text{H-NMA}}(\overrightarrow{p}(0))$ has the block Laplacian structure discussed in Section 2.2. Let $M$ be the $3n \times 3n$ mass matrix:

$$M = \begin{pmatrix} m_1 & & & & & & \\ & m_1 & & & & & \\ & & m_1 & & & & \\ & & & \ddots & & & \\ & & & & m_n & & \\ & & & & & m_n & \\ & & & & & & m_n \end{pmatrix} \ . \tag{2.19}$$

For the displacement vector $\overrightarrow{\delta}(t)$, $\dot{\overrightarrow{\delta}}(t)$ is the velocity, where the dot is a time derivative. Therefore, the kinetic energy of the ENM is given by:

$$E_k(\dot{\overrightarrow{\delta}}(t)) = \frac{1}{2}\dot{\overrightarrow{\delta}}(t)^T M\,\dot{\overrightarrow{\delta}}(t) \ . \tag{2.20}$$

The Lagrangian function is given by:

$$\begin{aligned} L &= \text{kinetic} - \text{potential} \\ &= E_k(\dot{\overrightarrow{\delta}}(t)) - E_{\text{H-NMA}}(\overrightarrow{p}(t)) \ . \end{aligned} \tag{2.21}$$

Using the second order approximation for the potential from Equation (2.18):

$$L = \frac{1}{2}\dot{\overrightarrow{\delta}}(t)^T M \dot{\overrightarrow{\delta}}(t) - \frac{1}{2}\overrightarrow{\delta}(t)^T \nabla^2 E_{\text{H-NMA}}(\overrightarrow{p}(0))\overrightarrow{\delta}(t) \ . \tag{2.22}$$

The equations of motion are then given by the Euler-Lagrange equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\overrightarrow{\delta}}(t)}\right) - \frac{\partial L}{\partial \overrightarrow{\delta}(t)} = 0 \ . \tag{2.23}$$

Substituting the kinetic and the second order approximated potential into the equations of motion gives:

$$M \ddot{\overrightarrow{\delta}}(t) + \nabla^2 E_{\text{H-NMA}}(\overrightarrow{p}(0))\overrightarrow{\delta}(t) = 0 \ . \tag{2.24}$$

Therefore, the normal mode displacements of the protein's atoms are given by the eigenvectors of $\nabla^2 E_{\text{H-NMA}}(\overrightarrow{p}(0))$. The eigenvectors of $\nabla^2 E_{\text{H-NMA}}(\overrightarrow{p}(0))$ corresponding to the low eigenvalues give the large amplitude motions of the protein structure.

## 2.4 The Hookean Potential for Elastic Network Interpolation (ENI)

Elastic Network Interpolation (ENI) [39, 37, 38, 41, 71, 40, 36] uses a Hookean potential with a linearly interpolated reference distance model conformational transitions. The *pair-wise* ENI Hookean potential used to find the $(t+1)$-st intermediate conformation is:

$$E_{ab,t+1}(p_a - p_b) = \frac{1}{2}(\sqrt{\overrightarrow{p}^T S_{ab}\overrightarrow{p}} - d_{ab}(t+1))^2 = \frac{1}{2}(\parallel p_a - p_b \parallel - d_{ab}(t+1))^2 \ , \tag{2.25}$$

The subscript $t+1$ is used to emphasize the potential is solving for the $(t+1)$-st transitional conformation. The distance $d_{ab}(t+1)$ is a linearly interpolated distance:

$$d_{ab}(t+1) = \left(1 - \frac{t+1}{100}\right) \parallel p_a(0) - p_b(0) \parallel + \left(\frac{t+1}{100}\right) \parallel p_a(100) - p_b(100) \parallel \ . \tag{2.26}$$

The denominator in the interpolation parameter is 100 since the last conformation's time index is assumed to be $t_{final} = 100$. The total potential energy used by ENI is:

$$E_{t+1}(\overrightarrow{p}) = \sum_{(a,b)\in\mathcal{D}} E_{ab,t+1}(p_a - p_b) \ . \tag{2.27}$$

The ENI Hookean potential is the generic Hookean potential with the following changes:

- the set $\mathcal{D}$ contains pairs of indices representing atoms within a threshold distance in *either* the beginning or ending conformations, they can be either bonded or nonbonded pairs (see [37] Section 3.2.1 page 65), these pairs do not change during the interpolation, and
- the reference distance is $\widehat{d}_{ab} = d_{ab}(t+1)$.

The meaning of the ENI Hookean potential in Equation (2.27) is to measure how close the inter-atomic distances of the atomic coordinates in $\overrightarrow{p}$ are to the *target* inter-atomic distances, $d_{ab}(t+1)$, for $(a,b) \in \mathcal{D}$. The atomic coordinates of the $(t+1)$-st intermediate conformation, denoted $\overrightarrow{p}_{t+1} = \overrightarrow{p}(t+1)$, minimizes this ENI Hookean potential, Equation (2.27). Let $\overrightarrow{\delta}_{t+1}$ be a small displacement, as defined in Equation (2.13), the $(t+1)$-st conformation is given by:

$$\overrightarrow{p}_{t+1} = \overrightarrow{p}_t + \overrightarrow{\delta}_{t+1} \ .$$

In order to find the next conformation $\overrightarrow{p}_{t+1}$, $\overrightarrow{\delta}_{t+1}$ needs to be determined. The $\overrightarrow{\delta}_{t+1}$ that gives the $\overrightarrow{p}_{t+1}$ which *minimizes* Equation (2.27) is referred to as the "optimal displacement".

The ENI approach to finding the optimal displacement $\overrightarrow{\delta}_{t+1}$ is to first expand Equation (2.27) to second order near $\overrightarrow{p}_t$:

$$\begin{aligned} E_{t+1}(\overrightarrow{p}_{t+1}) &= E_{t+1}(\overrightarrow{p}_t + \overrightarrow{\delta}_{t+1}) \\ &\approx \frac{1}{2}\overrightarrow{\delta}_{t+1}^{\ T} \nabla^2 E_{t+1}(\overrightarrow{p}_t)\, \overrightarrow{\delta}_{t+1} + \nabla E_{t+1}(\overrightarrow{p}_t)^T \overrightarrow{\delta}_{t+1} + E_{t+1}(\overrightarrow{p}_t) \ , \end{aligned} \tag{2.28}$$

then take the gradient of the second order approximation in Equation (2.28) with respect to $\overrightarrow{\delta}_{t+1}$:

$$\begin{aligned} &\frac{d}{d\,\overrightarrow{\delta}_{t+1}} \left( \frac{1}{2}\overrightarrow{\delta}_{t+1}^{\ T} \nabla^2 E_{t+1}(\overrightarrow{p}_t)\, \overrightarrow{\delta}_{t+1} + \nabla E_{t+1}(\overrightarrow{p}_t)^T \overrightarrow{\delta}_{t+1} + E_{t+1}(\overrightarrow{p}_t) \right) \\ &= \nabla^2 E_{t+1}(\overrightarrow{p}_t)\, \overrightarrow{\delta}_{t+1} + \nabla E_{t+1}(\overrightarrow{p}_t) \ . \end{aligned} \tag{2.29}$$

Finally, the $\overrightarrow{\delta}_{t+1}$ that sets Equation (2.29) to zero is the "optimal displacement" as defined by ENI. Setting Equation (2.29) to zero, then move $\nabla E_{t+1}(\overrightarrow{p}_t)$ to the other side shows $\overrightarrow{\delta}_{t+1}$ solves the linear system:

$$\nabla^2 E_{t+1}(\overrightarrow{p}_t)\,\overrightarrow{\delta}_{t+1} = -\nabla E_{t+1}(\overrightarrow{p}_t)\ , \tag{2.30}$$

## 2.5 The Hookean Potential for Interpolated ENM (iENM)

Interpolated ENM (iENM)[74, 81] generates a transition pathway using a *double-well* potential. Let $E_0(\overrightarrow{p})$ be a Hookean potential whose reference inter-atomic distances $\widehat{d}_{ab}$ are set to the beginning conformation's inter-atomic distances $d_{ab}(0)$:

$$\begin{aligned} E_0(\overrightarrow{p}) &= \frac{1}{2}\sum_{(a,b)\in\mathcal{D}_0} w_{ab}(\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}} - d_{ab}(0))^2 \\ &= \frac{1}{2}\sum_{(a,b)\in\mathcal{D}_0} w_{ab}(\parallel p_a - p_b \parallel - d_{ab}(0))^2\ , \end{aligned} \tag{2.31}$$

and let $E_{100}(\overrightarrow{p})$ be a Hookean potential whose reference inter-atomic distances $\widehat{d}_{ab}$ are set to the ending conformation's inter-atomic distances $d_{ab}(100)$:

$$\begin{aligned} E_{100}(\overrightarrow{p}) &= \frac{1}{2}\sum_{(a,b)\in\mathcal{D}_{100}} w_{ab}(\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}} - d_{ab}(100))^2 \\ &= \frac{1}{2}\sum_{(a,b)\in\mathcal{D}_{100}} w_{ab}(\parallel p_a - p_b \parallel - d_{ab}(100))^2\ . \end{aligned} \tag{2.32}$$

The weight $w_{ab}$ is set to 10 for bonded residues and 1 otherwise. The set of neighbour index pairs, $\mathcal{D}_0$ and $\mathcal{D}_{100}$, are calculated based on the beginning and ending conformation respectively. As with ENI, these sets of pairs do not change during the interpolation.

The physical meaning of $E_0(\overrightarrow{p})$ is to measure the deviations of an intermediate transitional protein conformation's inter-atomic distances, based on the atomic coordinates given in $\overrightarrow{p}$, from the beginning conformation. Likewise, $E_{100}(\overrightarrow{p})$ will measure the deviations of an intermediate transitional conformation's inter-atomic distances from the ending conformation.

The double-well potential used by iENM is a linear interpolation between $E_0(\overrightarrow{p})$ and $E_{100}(\overrightarrow{p})$. The definition of a double-well potential is given in Definition 2.5.1.

**Definition 2.5.1** (Double-Well Potential)**.** *An arbitrary double-well potential is a composite function,* $F(E_0(\overrightarrow{p}), E_{100}(\overrightarrow{p}))$*, that has the following property:*

$$F(E_0(\overrightarrow{p}), E_{100}(\overrightarrow{p})) \approx \begin{cases} E_0(\overrightarrow{p}) & \text{if } E_0(\overrightarrow{p}) \ll E_{100}(\overrightarrow{p}) \\ E_{100}(\overrightarrow{p}) & \text{if } E_0(\overrightarrow{p}) \gg E_{100}(\overrightarrow{p}) \end{cases} . \tag{2.33}$$

This property means that if $E_0(\overrightarrow{p}) \ll E_{100}(\overrightarrow{p})$, the intermediate conformation $\overrightarrow{p}$ is very similar to the beginning conformation $\overrightarrow{p}_0$, and therefore the double-well potential should produce a value very similar to $E_0(\overrightarrow{p})$. Likewise, if $E_0(\overrightarrow{p}) \gg E_{100}(\overrightarrow{p})$, the intermediate conformation $\overrightarrow{p}$ is very similar to the ending conformation $\overrightarrow{p}_{100}$ and therefore the double-well potential should produce a value very similar to $E_{100}(\overrightarrow{p})$.

Note that $E_0(\overrightarrow{p}_0) = 0$ and $E_{100}(\overrightarrow{p}_{100}) = 0$. Thus, the beginning conformation $\overrightarrow{p}_0$ and the ending conformation $\overrightarrow{p}_{100}$ are two minima of $F(E_0(\overrightarrow{p}), E_{100}(\overrightarrow{p}))$. The transitional conformations $\overrightarrow{p}_t$, $t \in (1, \ldots, 99)$, form a *sequence* of saddle points of $F(E_0(\overrightarrow{p}), E_{100}(\overrightarrow{p}))$, meaning they satisfy

$$\nabla F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t)) = \mathbf{0}_{3n} .$$

Using the chain rule:

$$\begin{aligned} &\nabla F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t)) \\ &= \left( \frac{\partial F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t))}{\partial E_0(\overrightarrow{p}_t)} \right) \nabla E_0(\overrightarrow{p}_t) + \left( \frac{\partial F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t))}{\partial E_{100}(\overrightarrow{p}_t))} \right) \nabla E_{100}(\overrightarrow{p}_t) \\ &= \mathbf{0}_{3n} . \end{aligned} \tag{2.34}$$

Tekpinar and Zheng have stated that based on previous studies, the transitional pathway generated by iENM is *independent* of the mathematical form of $F(E_0(\overrightarrow{p}), E_{100}(\overrightarrow{p}))$, see [74] and the references therein. Since simpler models of nature are always preferable to more complicated models, Tekpinar and Zheng proposed $E_0(\overrightarrow{p})$ and $E_{100}(\overrightarrow{p})$ should be connected by a linearly interpolated potential. Divide Equation (2.34) by

$$\frac{\partial F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t))}{\partial E_0(\overrightarrow{p}_t)} + \frac{\partial F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t))}{\partial E_{100}(\overrightarrow{p}_t)} ,$$

then define the parameter $\lambda_t$ to be:

$$\lambda_t = \frac{\frac{\partial F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t))}{\partial E_0(\overrightarrow{p}_t)}}{\frac{\partial F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t))}{\partial E_0(\overrightarrow{p}_t)} + \frac{\partial F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t)}{\partial E_{100}(\overrightarrow{p}_t)}} , \tag{2.35}$$

$\lambda_t$ is an interpolation parameter varying from 1 to 0. Equation (2.34) shows the saddle points satisfy:

$$\lambda_t \nabla E_0(\overrightarrow{p}_t) + (1-\lambda_t)\nabla E_{100}(\overrightarrow{p}_t) = \mathbf{0}_{3n} \ . \tag{2.36}$$

It follows the saddle points (transitional conformations) are critical points of the following *iENM Hookean potential energy*:

$$E_t(\overrightarrow{p}) = \lambda_t E_0(\overrightarrow{p}) + (1-\lambda_t)E_{100}(\overrightarrow{p}) \ . \tag{2.37}$$

Starting with $\overrightarrow{p}_0$ and $\lambda_0 = 1$, the original publication presents a way of getting $\lambda_1$ from which the next conformation $\overrightarrow{p}_1$ is determined, such a way of selecting the next $\lambda_t$ may not be uniform. However, since the iENM transition pathway is independent of the mathematical form of $F(E_0(\overrightarrow{p}_t), E_{100}(\overrightarrow{p}_t))$, this thesis will assume $\lambda_t$ changes uniformly; this means $\lambda_t$ is given by:

$$\lambda_t = 1 - \frac{t}{100} \quad \text{where } t = 0, 1, \ldots, 100 \ ,$$

which in turn gives the following iENM Hookean potential for finding the $t$-th transitional conformation:

$$E_t(\overrightarrow{p}) = \left(1 - \frac{t}{100}\right) E_0(\overrightarrow{p}) + \left(\frac{t}{100}\right) E_{100}(\overrightarrow{p}) \ . \tag{2.38}$$

Other simple functions have also been proposed, for example Das et al. [22] proposed:

$$\min(E_0(\overrightarrow{p}), E_{100}(\overrightarrow{p}))$$

which also satisfy the property given in Equation (2.33) that characterizes a double-well potential.

Tekpinar and Zheng added a steric collision energy to the iENM potential to ensure atoms in the transitional conformations do not collide, see Section 2.6.

The next conformation $\overrightarrow{p}_{t+1}$ is a critical point of the optimization problem of minimizing $E_{t+1}(\overrightarrow{p})$. Since the previous conformation $\overrightarrow{p}_t$ is known, finding the next conformation $\overrightarrow{p}_{t+1}$ is equivalent to finding $\overrightarrow{\delta}_{t+1}$ where $\overrightarrow{p}_{t+1} = \overrightarrow{p}_t + \overrightarrow{\delta}_{t+1}$ so the new conformation $\overrightarrow{p}_{t+1}$ sets the gradient of the potential energy to zero:

$$\nabla E_{t+1}(\overrightarrow{p}_t + \overrightarrow{\delta}_{t+1}) = \mathbf{0}_{3n} \ . \tag{2.39}$$

Equation (2.39) shows $\overrightarrow{\delta}_{t+1}$ can be viewed as the root of this Equation, and therefore Tekpinar and Zheng proposes to use the Newton-Raphson algorithm to find $\overrightarrow{\delta}_{t+1}$.

## 2.6 The Hookean Potential for Avoiding Inter-Atomic Collisions

Together with the iENM double-well potential, Tekpinar and Zheng also introduced a Hookean potential for avoiding atomic collisions[74, 81]. This potential is called the steric collision energy, denoted $E_{col}(\overrightarrow{p}_t)$, and is added to the iENM Hookean potential to ensure the non-bonded atoms of the intermediate conformations avoid colliding with each other during the transition. The steric collision energy for a pair of non-bonded atoms becomes nonzero when they are within the distance threshold given by $d_{col}$ measured in angstroms, this nonzero energy causes these atoms to be pushed apart. The formula for $E_{col}(\overrightarrow{p}_t)$ is given by:

$$E_{col}(\overrightarrow{p}_t) = \frac{1}{2}\sum_{a=3}^{n}\sum_{b=1}^{a-2} w_{ab}h(d_{col} - d_{ab}(t))(\parallel p_a(t) - p_b(t) \parallel - d_{col})^2 \,, \tag{2.40}$$

where $h(\cdot)$ is the Heaviside function meaning that:

$$h(d_{col} - d_{ab}(t)) = \begin{cases} 0 & \text{if } d_{col} \leq d_{ab}(t) \\ 1 & \text{if } d_{col} > d_{ab}(t) \end{cases} \tag{2.41}$$

$d_{col}$ is the threshold such that a collision between $\alpha$-carbons is assumed to have occurred, $d_{col} = 4\dot{A}$ is used by Tekpinar and Zheng [74]. $w_{ab}$ is set to 10 for non-bonded residues, i.e. $b \neq a \pm 1$. Since Tekpinar and Zheng have excluded bonded residues, it is possible for bonds to stretch or shrink during the interpolation, see Chapter 7.

## 2.7 Minimizing the Hookean Potential is Ill-Posed

Finding the next conformation by minimizing the Hookean potential is an ill-posed problem. Note that ill-posed here does not mean a solution cannot be found, only that the solution is not unique.

Since inter-atomic distance is invariant to translation of the atomic coordinates, any translation of the solution $\overrightarrow{\delta}_{t+1}$ for either ENI or iENM will produce *another* solution.

This can be seen in another way: the Hessian matrix $\nabla^2 E_{t+1}(\overrightarrow{p}_t)$ for ENI's Hookean potential in Equation (2.27) has six zero eigenvalues, and is therefore singular (not invertible).

Two modifications are suggested by Kim in [37] to ensure the solution $\overrightarrow{\delta}_{t+1}$ is unique.

1. A modification of the ENI Hookean potential, Equation (2.27), to make the matrix $\nabla^2 E_{t+1}(\overrightarrow{p}_t)$ invertible. The modified ENI Hookean potential adds a weighted Cartesian interpolation term to the ENI Hookean potential:

$$\widehat{E}_{t+1}(\overrightarrow{p}_{t+1}) = E_{t+1}(\overrightarrow{p}_{t+1}) + \frac{\epsilon}{2}\sum_{a=1}^{n} \parallel p_a(t) + \delta_a(t+1) - \widehat{p}_a(t+1) \parallel^2 \ . \tag{2.42}$$

   where $\epsilon = 0.1$ was used by Kim, and

$$\widehat{p}_a(t+1) = \left(\left(1 - \frac{t+1}{100}\right) p_a(0) + \left(\frac{t+1}{100}\right) p_a(100)\right) \ ,$$

   is the linearly interpolated time $t+1$ Cartesian coordinates. The gradient of this modified potential evaluated at $\overrightarrow{p}_t$ is:

$$\nabla \widehat{E}_{t+1}(\overrightarrow{p}_t) = \nabla E_{t+1}(\overrightarrow{p}_t) + \epsilon \begin{pmatrix} p_1(t) - \widehat{p}_1(t+1) \\ \vdots \\ p_n(t) - \widehat{p}_n(t+1) \end{pmatrix}$$

   and the Hessian matrix evaluated at $\overrightarrow{p}_t$ is:

$$\nabla^2 \widehat{E}_{t+1}(\overrightarrow{p}_t) = \nabla^2 E_{t+1}(\overrightarrow{p}_t) + \epsilon I_{3n} \ . \tag{2.43}$$

   The Hessian matrix is now invertible.

2. An alternative, second, modification is to add the constraint that the linear momentum is conserved:

$$\sum_{a=1}^{n} m_a \delta_a(t) = \mathbf{0}_3 \tag{2.44}$$

   Kim then makes the assumption that $m_a = 1$ for $a = 1, \ldots, n$.

The Hessian matrix for the iENM Hookean potential is also rank deficient. In order to use the Newton-Raphson algorithm, the Hessian matrix, $\nabla^2 E_{t+1}(\overrightarrow{p}_t)$, of Equation (2.37) was modified in [81]:

$$\epsilon I_{3n} + \nabla^2 E_{t+1}(\overrightarrow{p}_t) \ , \tag{2.45}$$

where $\epsilon$ is a small positive number. This is in fact the same modification as seen in Equation (2.43) for the ENI Hookean potential's non-invertible Hessian, both modifications adds a

weighted identity matrix to the Hessian of the Hookean potential to make the Hessian invertible. This correction is known as *Tikhonov's regularization*.

## 2.8 Enforcing Constraints $\mathbb{R}^{3n}$

The formulations for the transitional conformations problem by previous authors as discussed above do not have a step for enforcing inter-atomic distance constraints. Constrained optimization in $\mathbb{R}^{3n}$ is a well studied research topic. This section reviews the following results:

- the gradient projection method of J.B. Rosen and Rosen's correction (1961)[67] and
- the Fast Projection method of Goldenthal et al. (2007) [30],

These methods have close analogues to constrained optimization for PSD matrices introduced in Chapter 6.

Rosen's correction to the constraint manifold was for general nonlinear constraints, it used a "step and project" paradigm. However, their work was not built upon by Goldenthal et al., whom reintroduced the same correction for distance constraints in particular. It is a defect of $\mathbb{R}^{3n}$ that abstract geometric ideas are not emphasized and applied to different problems, and the *same* algorithm when applied to different problems are given *different* names.

Before these methods are reviewed, the notations used to express constraints are introduced here.

Consider a protein structure ENM with $n$ $\alpha$-carbons, indexed by $1, \ldots, n$. Let there be $m$ pairs of $\alpha$-carbon atoms where the distance between each pair of atoms is a constraint, these pairs of atoms are given as the following $m$ index pairs:

$$(a_1, b_1), \ldots, (a_m, b_m) \ .$$

Goldenthal et al. used the following formula to constrain the quadrance between the $a_i$-th and $b_i$-th atoms[30]:

$$C_i(\overrightarrow{p}) = \frac{\overrightarrow{p}^T S_{a_i b_i} \overrightarrow{p}}{d_{a_i b_i}} - d_{a_i b_i} = \frac{\| \ p_{a_i} - p_{b_i} \ \|^2}{d_{a_i b_i}} - d_{a_i b_i} \quad i = 1, \ldots, m. \tag{2.46}$$

For a vector $\overrightarrow{p} \in \mathbb{R}^{3n}$ of atomic coordinates, $C(\overrightarrow{p})$ is an $m \times 1$ vector describing all $m$ constraints, and is defined as follows in [30]:

$$C(\overrightarrow{p}) = \begin{pmatrix} C_1(\overrightarrow{p}) \\ \vdots \\ C_m(\overrightarrow{p}) \end{pmatrix} . \tag{2.47}$$

The *constraint manifold* is defined as the set:

$$\mathcal{C}(\mathbb{R}^{3n}) = \{ \overrightarrow{p} \mid C(\overrightarrow{p}) = \mathbf{0}_m \} . \tag{2.48}$$

Rewrite the constraint expression as:

$$C_i(\overrightarrow{p}) = \frac{1}{2} \left( \overrightarrow{p}^T S_{a_i b_i} \overrightarrow{p} - q_{a_i b_i} \right) = \frac{1}{2} \left( \| p_{a_i} - p_{b_i} \|^2 - q_{a_i b_i} \right) \quad i = 1, \ldots, m. \tag{2.49}$$

Clearly, this expression gives the same constraints as Goldenthal's original expression. Rearranging the constraint expression in this manner allows the gradient to be conveniently given by the following expression:

$$\nabla C_i(\overrightarrow{p}) = \overrightarrow{p}^T S_{a_i b_i} = (\overrightarrow{p}_{a_i} - \overrightarrow{p}_{b_i})^T (e_{a_i, I_3} - e_{b_i, I_3}) . \tag{2.50}$$

which is a $1 \times 3n$ vector. The gradient of all constraints is:

$$\nabla C(\overrightarrow{p}) = \begin{pmatrix} \nabla C_1(\overrightarrow{p}) \\ \vdots \\ \nabla C_m(\overrightarrow{p}) \end{pmatrix} = \begin{pmatrix} \overrightarrow{p}^T S_{a_1 b_1} \\ \vdots \\ \overrightarrow{p}^T S_{a_m b_m} \end{pmatrix} = \begin{pmatrix} (\overrightarrow{p}_{a_1} - \overrightarrow{p}_{b_1})^T (e_{a_1, I_3} - e_{b_1, I_3}) \\ \vdots \\ (\overrightarrow{p}_{a_m} - \overrightarrow{p}_{b_m})^T (e_{a_m, I_3} - e_{b_m, I_3}) \end{pmatrix} , \tag{2.51}$$

which is an $m \times 3n$ matrix.

### 2.8.1 J. B. Rosen's Gradient Projection (1960 and 1961)

J.B. Rosen's *Gradient Projection Method (GPM)* was first discussed in as early as 1957 in [65], then in more detailed papers in 1960 [66] and 1961[67].

Dantzig's *simplex method* was formulated in 1947, and published in 1951 [21]. Consequently, throughout the 1950s, linear programming was applied to a diverse range of problems. Quoting from Rosen [66]:

> [Applications of linear programming] include industrial applications, transportation problems, contract awards, military applications, agricultural applications, marketing analysis, production scheduling, inventory control and structural design. In many problems a far more realistic formulation is possible if a linearized approximation is not required. In such cases the optimum solution obtained by a nonlinear programming method should have considerably more validity than the solution to the approximate linear programming program. It is therefore anticipated that an efficient method for the solution of nonlinear programming problems will lead to *more realistic solutions* in many cases, as well as *extending the range of application* for programming methods. (emphasis added)

Rosen pointed to "problems in mechanics, physics and chemistry" as one example of a new area of application [66]. Given this historical context, Rosen's introduction of GPM was very much motivated by the Simplex Method. For example, see Section 6 of [66] where the author compares the GPM to the dual simplex method and the references therein.

Rosen's divide the GPM into two publications, Part I assumed constraints are linear [66], Part II assumed constraints are nonlinear [67]. Rosen also considered both equality and inequality constraints, the idea is that at a point $\overrightarrow{p} \in \mathbb{R}^{3n}$, only *active* inequality constraints are considered, these are the inequality constraints for which the equality holds true at the given point. The discussion in this section will therefore assume all constraints are equality constraints, this means the constraint manifold and the boundary of the constraint manifold are the same.

The main idea behind the GPM is to ensure the gradient always stays on the tangent space of the constraint manifold. In Part I of the publication [66], constraints are assumed to be linear, which means they can be written in the form:

$$L_i(\overrightarrow{p}) = \overrightarrow{\eta}_i^T \overrightarrow{p} - \alpha$$

where $\overrightarrow{\eta}_i \in \mathbb{R}^{3n}$ is a vector and $\alpha \in \mathbb{R}$ is a constant. In this case, let the constraint manifold be:

$$\mathcal{L}(\overrightarrow{p}) = \{\overrightarrow{p} \mid L_i(\overrightarrow{p}) = 0 \; i = 1, \ldots, m\}$$

A vector is on the boundary of the constraint manifold if it is perpendicular to the space span$\{\overrightarrow{\eta}_1, \ldots, \overrightarrow{\eta}_m\}$. Define the $3n \times m$ matrix:

$$N = \begin{pmatrix} \overrightarrow{\eta}_1 & \ldots & \overrightarrow{\eta}_m \end{pmatrix} ,$$

then the projection onto the constraint manifold is given by the projection matrix:

$$\text{Proj} = I_{3n} - N(N^T N)^{-1} N^T$$

Part II of the publication [67] allowed the constraints to be nonlinear. In this case, the projection will map a vector onto the *tangent space* of the constraint manifold, thereafter *a correction back to the constraint manifold* from the tangent space is required. For example, let the constraints be $C(\overrightarrow{p})$ as defined in Equation (2.47), make the substitution:

$$N = \nabla C(\overrightarrow{p})^T \ ,$$

where $\nabla C(\overrightarrow{p})$ is defined in Equation (2.51), then the projection onto the tangent space of the constraint manifold is given by:

$$\text{Proj} = I_{3n} - \nabla C(\overrightarrow{p})^T (\nabla C(\overrightarrow{p}) \nabla C(\overrightarrow{p})^T)^{-1} \nabla C(\overrightarrow{p}) \ . \tag{2.52}$$

The GPM requires a step to be taken in this direction, but since the constraint is not linear, this step leads to a point that is not on the constraint manifold. Rosen suggests "a correction back to the feasible region" (see Section 3 [67]). Rosen did not give a name to this procedure, we will refer to it as "Rosen's Correction". Rosen's Correction proceeds as follows: generate a sequence of points starting from the initial point on the tangent space $\overrightarrow{p}^{(0)}$ given by (see Equation (4.22) of [67]):

$$\overrightarrow{p}^{(k)} = \overrightarrow{p}^{(k-1)} - \nabla C(\overrightarrow{p}^{(k-1)})^T (\nabla C(\overrightarrow{p}^{(k-1)}) \nabla C(\overrightarrow{p}^{(k-1)})^T)^{-1} C(\overrightarrow{p}^{(k-1)}) \tag{2.53}$$

until $\| \ C(\overrightarrow{p}^{(k-1)}) \ \|$ is smaller than some tolerance. Rosen's Correction is reintroduced by Goldenthal et al. as the Fast Projection Algorithm to be reviewed next.

### 2.8.2 Goldenthal et al.'s Fast projection (2007)

Goldenthal et al. examined the modelling of cloth as a particle system with constraints on inter-particle distances in [30]. The equations of motion for this particle system can be arrived at from the augmented Lagrangian for modelling constrained dynamics in $\mathbb{R}^{3n}$ given by:

$$L(\overrightarrow{p}_t, \overrightarrow{v}_t) = \frac{1}{2} (\overrightarrow{v}_t)^T M \overrightarrow{v}_t - E(\overrightarrow{p}_t) - C(\overrightarrow{p}_t)^T \lambda \tag{2.54}$$

where $\overrightarrow{v}_t = \dot{\overrightarrow{p}}_t$ is the velocity, $M$ is the $3n \times 3n$ mass matrix with the mass of each particle repeated three times along the diagonal, see Equation (2.19). $E(\overrightarrow{p}_t)$ is the potential energy,

and $\lambda$ is the $m \times 1$ vector of Lagrange multipliers for the constraints.

The equations of motion are given by two Euler-Lagrange equations. The first one is given by:

$$M\ddot{\overrightarrow{p}}_t = -\nabla E(\overrightarrow{p}_t) - \nabla C(\overrightarrow{p}_t)^T \lambda \tag{2.55}$$

Equation (2.55) states the total force acting on the atoms is the sum of the force from the potential energy and the constraint force. The second Euler-Lagrange equation is

$$C(\overrightarrow{p}_t) = \mathbf{0}_m \tag{2.56}$$

The second equation states all constraints must be satisfied, equivalently, $\overrightarrow{p}_t$ is on the constraint manifold given by Equation (2.48).

The equations of motion are discretized over a time interval when used in computer simulation. SHAKE and RATTLE are examples of discretization methods [68, 6], both considers the constraint force to be evaluated *explicitly*, this means the coordinates of the particles being simulated are evaluated at the *beginning* of the time interval. Barth et al. pointed out four common geometric configurations where the SHAKE algorithm faces difficulty [9]. Goldenthal et al. [30] proposed that the constraint-maintaining force should be evaluated *implicitly* for SHAKE, that is, using the unconstrained coordinates at the *end* of the time interval instead of the beginning, to alleviate the problems described by Barth et al., this formulation is called Fast Projection.

The following discussion assumes $t$ takes on arbitrary discrete values $t_j$ and $t_{j+1}$, and $\Delta t = t_{j+1} - t_j$ indicates an arbitrary change in time. Define $\overrightarrow{p}_{j+1} = \overrightarrow{p}(t_{j+1})$ and $\overrightarrow{p}_j = \overrightarrow{p}(t_j)$. Let $\overrightarrow{v}_j = \overrightarrow{v}(t_j)$ be the time $t = t_j$ velocity, and $\overrightarrow{v}_{j+1} = \overrightarrow{v}(t_{j+1})$ be the time $t_{j+1}$ velocity.

To arrive at the Fast Projection algorithm, two algorithms are first presented in [30].

1. Firstly, evaluating the constraint force at the *end of the time interval*, $[t_j, t_{j+1}]$, this is the *implicitly constraint direction (ICD)* method.

2. Secondly, consider an alternative, but *equivalent*, perspective to constrained dynamics. Take an unconstrained step, then find the *closest* point on the constraint manifold to project to. This perspective leads to the *step and project (SAP) method*.

3. Finally, Fast Projection is an approximation of SAP.

### Implicit Constraint Direction (ICD)

The discretized equations of motion between time $t_j$ and $t_{j+1}$, as presented in [30], assumes the force due to the potential energy is treated explicitly, and the constraint-maintaining force is treated implicitly. Equation (2.55) which describes forces is discretized as:

$$M\frac{\overrightarrow{v}_{j+1}-\overrightarrow{v}_j}{\Delta t} = -\nabla E(\overrightarrow{p}_j) - \nabla C(\overrightarrow{p}_{j+1})^T\lambda \; . \tag{2.57}$$

Equation (2.57) shows how to find the new velocity $\overrightarrow{v}_{j+1}$ from the potential and constraint forces. This new velocity is then used to find the new coordinates $\overrightarrow{p}_{j+1}$.

$$\frac{\overrightarrow{p}_{j+1}-\overrightarrow{p}_j}{\Delta t} = \overrightarrow{v}_{j+1} \; . \tag{2.58}$$

Equation (2.56) when discretized implicitly uses the end of the time interval value:

$$C(\overrightarrow{p}_{j+1}) = \mathbf{0}_m \; . \tag{2.59}$$

Combining Equation (2.57) and (2.58) gives the following Equation for $\overrightarrow{p}_{j+1}$:

$$\overrightarrow{p}_{j+1} = \overrightarrow{p}_j + (\Delta t)\overrightarrow{v}_j - (\Delta t)^2 M^{-1}\nabla E(\overrightarrow{p}_j) - (\Delta t)^2 M^{-1}\nabla C(\overrightarrow{p}_{j+1})^T\lambda \; . \tag{2.60}$$

Equation (2.60) shows $\overrightarrow{p}_{j+1}$ can be separated into an *unconstrained step* given by:

$$\overrightarrow{p}^{(0)}_{j+1} = \overrightarrow{p}_j + (\Delta t)\overrightarrow{v}_j - (\Delta t)^2 M^{-1}\nabla E(\overrightarrow{p}_j) \; , \tag{2.61}$$

followed by a *correction* due to the constraint forces, denoted $\delta\overrightarrow{p}_{j+1}$, given by:

$$\begin{aligned} \delta\overrightarrow{p}_{j+1} &= \overrightarrow{p}_{j+1} - \overrightarrow{p}^{(0)}_{j+1} \\ &= -(\Delta t)^2 M^{-1}\nabla C(\overrightarrow{p}_{j+1})^T\lambda \; . \end{aligned} \tag{2.62}$$

Finally, the constraint enforcing step $\delta\overrightarrow{p}_{j+1}$ and the new Lagrange multipliers $\lambda$ can be written as the roots of the following system of equations:

$$\begin{aligned} &F(\delta\overrightarrow{p}_{j+1},\lambda) = \delta\overrightarrow{p}_{j+1} + (\Delta t)^2 M^{-1}\nabla C(\overrightarrow{p}_{j+1})^T\lambda = \mathbf{0}_{3n} \\ &C(\overrightarrow{p}_{j+1}) = \mathbf{0}_m \; . \end{aligned} \tag{2.63}$$

The meaning of $F(\delta\overrightarrow{p}_{j+1},\lambda)$ is to measure the deviation of the atomic coordinates away from the forces due to the potential and constraint forces, it states that the unconstrained

step is corrected by the constraint forces. The function $C(\overrightarrow{p}_{j+1})$ measures the deviation of $\overrightarrow{p}_{j+1}$ from the constraint manifold, defined in Equation (2.48).

### Step and Project(SAP)

Step and Project (SAP) is a different perspective from ICD. It views the *correction* of the unconstrained step, $\delta\overrightarrow{p}_{j+1}$, as a *projection* of $\overrightarrow{p}^{(0)}_{j+1}$ to a point on the constraint manifold *closest* to $\overrightarrow{p}^{(0)}_{j+1}$.

In other words, $\overrightarrow{p}^{(0)}_{j+1}$ and $\overrightarrow{p}_{j+1}$ are as close as possible, and this closeness is measured using the $L^2$ norm of the mass-weighted displacement $\delta\overrightarrow{p}_{j+1}$:

$$(\delta\overrightarrow{p}_{j+1})^T M(\delta\overrightarrow{p}_{j+1}) \ ,$$

where $\delta\overrightarrow{p}_{j+1} = \overrightarrow{p}_{j+1} - \overrightarrow{p}^{(0)}_{j+1}$ is the correction of the unconstrained step.

SAP solves the following optimization problem to finding the closest point on the constraint manifold to the unconstrained step:

$$\begin{aligned} \min \quad & \frac{1}{2(\Delta t)^2}(\delta\overrightarrow{p}_{j+1})^T M(\delta\overrightarrow{p}_{j+1}) \\ s.t. \quad & C(\overrightarrow{p}_{j+1}) = \mathbf{0}_m \ . \end{aligned} \tag{2.64}$$

The constant $1/2(\Delta t)^2$ will become clear in Theorem 2.8.1. The Lagrangian of the optimization problem for SAP in Equation (2.64) is:

$$L_{SAP}(\delta\overrightarrow{p}_{j+1}, \lambda) = \frac{1}{2(\Delta t)^2}(\delta\overrightarrow{p}_{j+1})^T M(\delta\overrightarrow{p}_{j+1}) + C(\overrightarrow{p}_{j+1})^T\lambda \ . \tag{2.65}$$

ICD and SAP will find the same $\delta\overrightarrow{p}_{j+1}$, and $\lambda$. This agreement is given as Theorem 1 in [30] and also presented as Theorem 2.8.1 below, where a more detailed explanation is given.

**Theorem 2.8.1** (Agreement between ICD and SAP)**.** *ICD and SAP finds the same solution.*

*Proof.* A stationary point $(\delta \overrightarrow{p}_{j+1}, \lambda)$ for SAP satisfies the following stationary equations:

$$\begin{aligned} \frac{\partial L_{SAP}(\delta \overrightarrow{p}_{j+1}, \lambda)}{\partial \delta \overrightarrow{p}_{j+1}} &= \mathbf{0}_{3n} \\ \frac{\partial L_{SAP}(\delta \overrightarrow{p}_{j+1}, \lambda)}{\partial \lambda} &= \mathbf{0}_{m} \ . \end{aligned} \tag{2.66}$$

Substituting in the expression for $L_{SAP}(\delta \overrightarrow{p}_{j+1}, \lambda)$ from Equation (2.65) shows the stationary equations are given by:

$$\begin{aligned} \frac{1}{(\Delta t)^2} M \delta \overrightarrow{p}_{j+1} + \nabla C(\overrightarrow{p}_{j+1})^T \lambda &= \mathbf{0}_{3n} \\ C(\overrightarrow{p}_{j+1}) &= \mathbf{0}_{m} \ . \end{aligned} \tag{2.67}$$

Rearranging gives:

$$\begin{aligned} \delta \overrightarrow{p}_{j+1} + (\Delta t)^2 M^{-1} \nabla C(\overrightarrow{p}_{j+1})^T \lambda &= \mathbf{0}_{3n} \\ C(\overrightarrow{p}_{j+1}) &= \mathbf{0}_{m} \ . \end{aligned} \tag{2.68}$$

The linear system of Equation (2.68) is the same as the ICD system given in Equation (2.63) and repeated below:

$$\begin{aligned} &F(\delta \overrightarrow{p}_{j+1}, \lambda) = \delta \overrightarrow{p}_{j+1} + (\Delta t)^2 M^{-1} \nabla C(\overrightarrow{p}_{j+1})^T \lambda = \mathbf{0}_{3n} \\ &C(\overrightarrow{p}_{j+1}) = \mathbf{0}_{m} \ . \end{aligned} \tag{2.69}$$

Therefore, $(\delta \overrightarrow{p}_{j+1},\ \lambda)$ are the roots of the ICD system and also a stationary point for SAP. □

Even though ICD and SAP finds the same solutions, the SAP paradigm is more powerful since it leads to the Fast Projection algorithm discussed next.

### Fast Projection (FP) is an Approximation of SAP

Fast Projection is an approximation of the SAP method. Instead of finding *the* closest point to the unconstrained step on the constraint manifold, $\overrightarrow{p}_{j+1}$, Fast Projection finds a sequence of *small* steps approaching the constraint manifold:

$$\overrightarrow{p}^{(0)}_{j+1},\ \overrightarrow{p}^{(1)}_{j+1},\ \overrightarrow{p}^{(2)}_{j+1},\ \overrightarrow{p}^{(3)}_{j+1}, \ldots$$

where $\overrightarrow{p}_{j+1}^{(0)}$ is given by Equation (2.61) and

$$\overrightarrow{p}_{j+1}^{(k)} = \overrightarrow{p}_{j+1}^{(k-1)} + \delta\overrightarrow{p}_{j+1}^{(k-1)} \quad \text{for } k = 1, 2, \dots ,$$

and $\delta\overrightarrow{p}_{j+1}^{(k-1)}$ is a small step.

The following discussion shows how to find $\overrightarrow{p}_{j+1}^{(1)}$ from $\overrightarrow{p}_{j+1}^{(0)}$. Since $\| \delta\overrightarrow{p}_{j+1}^{(0)} \|$ is small, $C(\overrightarrow{p}_{j+1}^{(1)})$ is very close to its linear approximation near $\overrightarrow{p}_{j+1}^{(0)}$:

$$C(\overrightarrow{p}_{j+1}^{(1)}) = C(\overrightarrow{p}_{j+1}^{(0)} + \delta\overrightarrow{p}_{j+1}^{(0)}) \approx C(\overrightarrow{p}_{j+1}^{(0)}) + \nabla C(\overrightarrow{p}_{j+1}^{(0)})\delta\overrightarrow{p}_{j+1}^{(0)} . \tag{2.70}$$

Therefore, Fast Projection replaces the constraint in Equation (2.64) with its linear approximation to solve an optimization problem very similar to SAP to find the optimal step $\delta\overrightarrow{p}_{j+1}^{(0)}$ :

$$\begin{aligned} \min \quad & \frac{1}{2(\Delta t)^2}(\delta\overrightarrow{p}_{j+1}^{(0)})^T M(\delta\overrightarrow{p}_{j+1}^{(0)}) \\ s.t. \quad & C(\overrightarrow{p}_{j+1}^{(0)}) + \nabla C(\overrightarrow{p}_{j+1}^{(0)})\delta\overrightarrow{p}_{j+1}^{(0)} = \mathbf{0}_m . \end{aligned} \tag{2.71}$$

The meaning of this Fast Projection optimization problem is to find a small step, $\delta\overrightarrow{p}_{j+1}^{(0)}$ such that the linearly approximated constraint near $\overrightarrow{p}_{j+1}^{(0)}$ is satisfied. This small step then gives a new point, $\overrightarrow{p}_{j+1}^{(1)}$, that satisfies the constraints a bit better than $\overrightarrow{p}_{j+1}^{(0)}$. The only difference between Equation (2.71) and Equation (2.64) is the constraint manifold has been linearly approximated. The Lagrangian for the optimization problem in Equation (2.71) is:

$$\begin{aligned} & L_{FP}(\delta\overrightarrow{p}_{j+1}^{(0)}, \delta\lambda^{(0)}) \\ & = \frac{1}{2(\Delta t)^2}(\delta\overrightarrow{p}_{j+1}^{(0)})^T M(\delta\overrightarrow{p}_{j+1}^{(0)}) + \left(C(\overrightarrow{p}_{j+1}^{(0)}) + \nabla C(\overrightarrow{p}_{j+1}^{(0)})\delta\overrightarrow{p}_{j+1}^{(0)}\right)^T \delta\lambda^{(0)} , \end{aligned} \tag{2.72}$$

where $\delta\lambda^{(0)} \in \mathbb{R}^m$ are $m$ Lagrange multipliers for the *small* constraint-maintaining force correcting $\overrightarrow{p}_{j+1}^{(0)}$. A stationary point of Equation (2.72), $(\delta\overrightarrow{p}_{j+1}^{(0)}, \delta\lambda^{(0)})$, satisfies the following two stationary equations simultaneously:

$$\frac{\partial L_{FP}(\delta\overrightarrow{p}_{j+1}^{(0)}, \delta\lambda^{(0)})}{\partial\delta\overrightarrow{p}_{j+1}^{(0)}} = \frac{1}{(\Delta t)^2}M(\delta\overrightarrow{p}_{j+1}^{(0)}) + \nabla C(\overrightarrow{p}_{j+1}^{(0)})^T\delta\lambda^{(0)} = \mathbf{0}_{3n} , \tag{2.73}$$

$$\frac{\partial L_{FP}(\delta \overrightarrow{p}^{(0)}_{j+1}, \delta\lambda^{(0)})}{\partial \delta\lambda^{(0)}} = C(\overrightarrow{p}^{(0)}_{j+1}) + \nabla C(\overrightarrow{p}^{(0)}_{j+1})\delta \overrightarrow{p}^{(0)}_{j+1} = \mathbf{0}_m \ . \tag{2.74}$$

Rearranging Equation (2.73) gives:

$$\delta \overrightarrow{p}^{(0)}_{j+1} = -(\Delta t)^2 M^{-1} \nabla C(\overrightarrow{p}^{(0)}_{j+1})^T \delta\lambda^{(0)} \ . \tag{2.75}$$

Rearranging Equation (2.74) gives:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1})\delta \overrightarrow{p}^{(0)}_{j+1} = -C(\overrightarrow{p}^{(0)}_{j+1}) \ . \tag{2.76}$$

Substituting Equation (2.75) into Equation (2.76) gives:

$$(\Delta t)^2 M^{-1} \left( \nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}^{(0)}_{j+1})^T \right) \delta\lambda^{(0)} = C(\overrightarrow{p}^{(0)}_{j+1}) \ . \tag{2.77}$$

The $\delta\lambda^{(0)}$ that is the solution to Equation (2.77) will be used to find the correction $\delta \overrightarrow{p}^{(0)}_{j+1}$ using Equation (2.75). Then, $\overrightarrow{p}^{(1)}_{j+1}$ is a point on the linearly approximated constraint manifold and is found by:

$$\overrightarrow{p}^{(1)}_{j+1} = \overrightarrow{p}^{(0)}_{j+1} + \delta \overrightarrow{p}^{(0)}_{j+1} \ .$$

The same process is used to find $\overrightarrow{p}^{(2)}_{j+1}, \overrightarrow{p}^{(3)}_{j+1}, \ldots$ until $\parallel C(\overrightarrow{p}^{(k)}_{j+1}) \parallel$ is smaller than a predefined threshold level, $\epsilon$, which is an input to the Fast Projection algorithm. Since a Fast Projection step is an approximation of SAP, $\delta\lambda^{(k)}$ found from the $k$-th iteration of Fast Projection can be optionally scaled by a factor, $\alpha \in \mathbb{R}$, $0 < \alpha \leq 1$ as needed to ensure the correction is small. Fast Projection is summarized in Algorithm 1.

### 2.8.3 Benefits of Fast Projection

Goldenthal et al. [30] provided two benefits of Fast Projection.

1. Firstly, the authors showed Fast Projection can handle three of the four cases where SHAKE will face difficulty as described in Barth et al.[9]. The remaining case can be avoided by taking a smaller unconstrained step.

2. Secondly, the authors presented comparisons of Fast Projection with other algorithms, and showed Fast Projection is faster and more accurate.

These are discussed in the following subsections.

**Algorithm 1** Fast Projection

**INPUT:** Initial iterate $\overrightarrow{p}_{j+1}^{(0)}$ resulting from an unconstrained step, see Equation (2.61). A small number $\epsilon$.

**OUTPUT:** A sequence of iterates $\overrightarrow{p}_{j+1}^{(0)}, \overrightarrow{p}_{j+1}^{(1)}, \ldots$ approaching the constraint manifold $\mathcal{C}(\mathbb{R}^{3n})$.

1: **for** $k = 0, 1, 2, \ldots$ **do**

2: Solve the linear system:

$$(\Delta t)^2 \left( \nabla C(\overrightarrow{p}_{j+1}^{(k)}) M^{-1} \nabla C(\overrightarrow{p}_{j+1}^{(k)})^T \right) \delta\lambda^{(k)} = C(\overrightarrow{p}_{j+1}^{(k)}) \, . \tag{2.78}$$

to find $\delta\lambda^{(k)}$.

3: Find the correction $\delta\overrightarrow{p}_{j+1}^{(k)}$:

$$\delta\overrightarrow{p}_{j+1}^{(k)} = -(\Delta t)^2 M^{-1} \nabla C(\overrightarrow{p}_{j+1}^{(k)})^T \delta\lambda^{(k)} \, . \tag{2.79}$$

4: Update $\overrightarrow{p}_{j+1}^{(k+1)} = \delta\overrightarrow{p}_{j+1}^{(k)} + \overrightarrow{p}_{j+1}^{(k)}$.

5: **if** $\|C(\overrightarrow{p}_{j+1}^{(k+1)})\| < \epsilon$ **then**

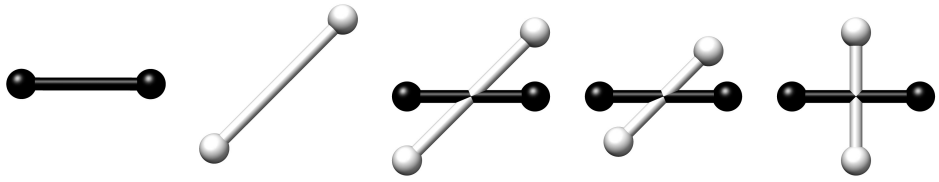
6: Return $\overrightarrow{p}_{j+1}^{(k+1)}$

7: **end if**

8: **end for**

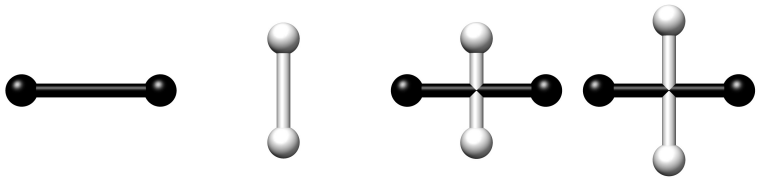
**Advantage of Fast Projection's Implicit Constraint Force over SHAKE's Explicit Constraint Force**

Barth et al.[9] discussed four cases where an explicit constraint force will not be able to produce a correct constraint force after the atomic coordinates have changed. Goldenthal et al. [30] discussed why treating the constraint force implicitly can lead to improvements, their discussions are summarized here.



(a) $\overrightarrow{p}_j$ (b) $\overrightarrow{p}^{(0)}_{j+1}$ (c) Superposition. (d) FP. (e) SHAKE.

Figure 2.1: Case 1: The explicit constraint force used by SHAKE, points in the wrong direction.



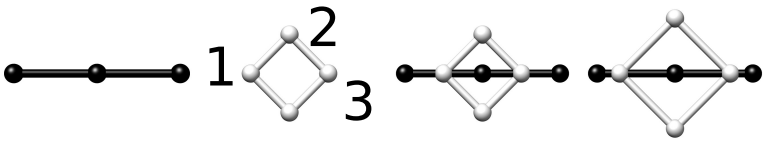
(a) $\overrightarrow{p}_j$ (b) $\overrightarrow{p}^{(0)}_{j+1}$ (c) Superposition. (d) FP.

Figure 2.2: Case 2: The matrix $\nabla C(\overrightarrow{p}^{(0)}_{j+1})\nabla C(\overrightarrow{p}_j)^T$ used by SHAKE is singular. The matrix $\nabla C(\overrightarrow{p}^{(0)}_{j+1})\nabla C(\overrightarrow{p}^{(0)}_{j+1})^T$ used by Fast Projection is nonsingular.

The four cases are presented in Figures 2.1, 2.2, 2.3, and 2.4. The black colored lattice structures in each figure is the configuration at the beginning of the time interval, with atomic coordinates denoted by the vector $\overrightarrow{p}_j$. The light gray structure is the the result of one unconstrained step and is denoted $\overrightarrow{p}^{(0)}_{j+1}$, see Equation (2.61). The exact values of $\Delta t$, $\overrightarrow{v}_j$, and the force due to the potential energy that caused the unconstrained step is not relevant to this discussion. Rather, the gradient $\nabla C(\overrightarrow{p})$ as given by Equation (2.51) and

(a) $\overrightarrow{p}_j$ (b) $\overrightarrow{p}^{(0)}_{j+1}$ (c) Superposition.

Figure 2.3: Case 3: Two atoms with the same coordinates causes the matrix $\nabla C(\overrightarrow{p}^{(0)}_{j+1})$ to be rank deficient. Both SHAKE and Fast Projection produces no solution. This can be avoided by taking a smaller unconstrained step.




(a) $\overrightarrow{p}_j$ (b) $\overrightarrow{p}^{(0)}_{j+1}$ (c) Superposition. (d) FP.

Figure 2.4: Case 4: $\nabla C(\overrightarrow{p}_j)$ is rank deficient.

the related matrix:

$$\nabla C(\overrightarrow{p})M^{-1}\nabla C(\overrightarrow{p})^T \ ,$$

corresponding to each of the configurations given in Figures 2.1, 2.2, 2.3, and 2.4 are used to illustrate how Fast Projection enforces constraints differently from SHAKE.

The following discussion assumes all atoms involved are the same, and therefore have the same mass $m$. This means $M = mI_{3n}$, where $m$ is scalar that can be absorbed by the Lagrange multiplier $\delta\lambda^{(k)}$, therefore, we can assume $M = I_{3n}$. Under this assumption, SHAKE's linear system at beginning ofthe first iteration uses the matrix,

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1})\nabla C(\overrightarrow{p}_j)^T \ ,$$

while Fast Projection's linear system at the beginning of the first iteration uses the matrix:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1})\nabla C(\overrightarrow{p}^{(0)}_{j+1})^T \ .$$

In addition, the constraints to be enforced are the bond lengths between consecutive atoms

as determined by the atomic coordinates in $\overrightarrow{p}_j$. This means $C(\overrightarrow{p}_j) = \mathbf{0}_m$ holds, whereas $C(\overrightarrow{p}^{(0)}_{j+1}) \neq \mathbf{0}_m$ holds.

1. Case 1 is presented in Figure 2.1. In this case, $\overrightarrow{p}^{(0)}_{j+1}$ results from a 45° rotation and a stretch of the bond length of $\overrightarrow{p}_j$. The main problem illustrated in this case is that an explicit force does not use the most current relevant information.

   Let the atomic coordinates of the black lattice be $(0,0,0)$ and $(4,0,0)$, and $\overrightarrow{p}_j = (0,0,0,4,0,0)^T$. Let the atomic coordinates of the light gray lattice be $(0,-2,0)$, and $(4,2,0)$, and $\overrightarrow{p}^{(0)}_{j+1} = (0,-2,0,4,2,0)^T$. Calculating $\nabla C(\overrightarrow{p}_j)$ shows:

$$\begin{aligned}
\nabla C(\overrightarrow{p}_j) = \overrightarrow{p}_j^T S_{12} &= (p_1(j) - p_2(j))^T \begin{pmatrix} I_3 & | -I_3 \end{pmatrix} \\
&= ((0,0,0) - (4,0,0)) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (-4,0,0) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (-4,0,0,4,0,0)
\end{aligned}$$

   The explicit constraint force $\nabla C(\overrightarrow{p}_j) = (-4,0,0,4,0,0)$ can only make a correction in the $x$ direction for each atomic coordinate in the unconstrained step. $\nabla C(\overrightarrow{p}^{(0)}_{j+1})$ is given by:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1}) = (\overrightarrow{p}^{(0)}_{j+1})^T S_{12} = (p_1^{(0)}(j+1) - p_2^{(0)}(j+1))^T \begin{pmatrix} I_3 & | -I_3 \end{pmatrix}$$

$$\begin{aligned}
&= ((0,-2,0) - (4,2,0)) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (-4,-4,0) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (-4,-4,0,4,4,0)
\end{aligned}$$

   Fast Projection's implicit constraint force $\nabla C(\overrightarrow{p}^{(0)}_{j+1}) = (-4,-4,0,4,4,0)$ can make a correction in *both* the $x$ and $y$ direction for each atomic coordinate in the lattice.

As a result, Fast Projection is able to adjust the bond in the diagonal direction, but SHAKE incorrectly makes a horizontal adjustment, rotating the bond's orientation.

2. Case 2 is presented in Figure 2.2. In this case, $\overrightarrow{p}_{j+1}^{(0)}$ results from a 90° rotation and a shrink in the bond length from the beginning of timestep conformation, with coordinates $\overrightarrow{p}_j$. The main problem illustrated in this case is that a large rotation due to the unconstrained step can cause the matrix used by SHAKE: $\nabla C(\overrightarrow{p}_{j+1}^{(0)})\nabla C(\overrightarrow{p}_j)^T$ to be come singular or ill-conditioned (near singular).

   Let the black lattice's atomic coordinates be $(0,0,0)$ and $(4,0,0)$, let the light gray lattice's atomic coordinates be $(2,1.5,0)$, and $(2,-1.5,0)$, giving $\overrightarrow{p}_j = (0,0,0,4,0,0)^T$ and $\overrightarrow{p}_{j+1}^{(0)} = (2,1.5,0,2,-1.5,0)^T$. $\nabla C(\overrightarrow{p}_j)$ is given by:

$$\begin{aligned}
\nabla C(\overrightarrow{p}_j) = \overrightarrow{p}_j^T S_{12} &= (p_1(j) - p_2(j))^T \begin{pmatrix} I_3 & | -I_3 \end{pmatrix} \\
&= ((0,0,0) - (4,0,0)) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (-4,0,0) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (-4,0,0,4,0,0)
\end{aligned}$$

   $\nabla C(\overrightarrow{p}_{j+1}^{(0)})$ is given by:

$$\nabla C(\overrightarrow{p}_{j+1}^{(0)}) = (\overrightarrow{p}_{j+1}^{(0)})^T S_{12} = (p_1^{(0)}(j+1) - p_2^{(0)}(j+1))^T \begin{pmatrix} I_3 & | -I_3 \end{pmatrix}$$

$$\begin{aligned}
&= ((2,1.5,0) - (2,-1.5,0)) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (0,3,0) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \\
&= (0,3,0,0,-3,0)
\end{aligned}$$

$C(\overrightarrow{p}^{(0)}_{j+1})$ is given by:

$$C(\overrightarrow{p}^{(0)}_{j+1}) = \frac{1}{2}(\| (2, 1.5, 0) - (2, -1.5, 0) \|^2 - \| (0, 0, 0) - (4, 0, 0) \|^2) = \frac{1}{2}(9 - 16) = -3.5$$

The linear system SHAKE solves at the first iteration is:

$$\begin{aligned}
&\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}_j)^T \lambda^{(0)} = C(\overrightarrow{p}^{(0)}_{j+1}) \\
&(0, 3, 0, 0, -3, 0)(-4, 0, 0, 4, 0, 0)^T \lambda^{(0)} = -3.5 \\
&0 = -3.5
\end{aligned}$$

This is a contradiction, thus SHAKE is *unable to find a solution.* On the other hand, Fast Projection's first iteration solves the system:

$$\begin{aligned}
&\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}^{(0)}_{j+1})^T \lambda^{(0)} = C(\overrightarrow{p}^{(0)}_{j+1}) \\
&(0, 3, 0, 0, -3, 0)(0, 3, 0, 0, -3, 0)^T \lambda^{(0)} = -3.5 \\
&18\lambda^{(0)} = -3.5
\end{aligned}$$

Therefore, large rotations due to an unconstrained step does not cause a problem for Fast Projection, but can cause the matrix used by SHAKE, $\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}_j)^T$ to become singular, or ill-conditioned.

3. In Figure 2.3, the atoms have collided resulting in a bond length of zero, after an upwards motion. This is a situation where *both* SHAKE and Fast Projection cannot find a solution, but can be avoided by taking a small unconstrained step.

   In this situation, the unconstrained step has the bond shrunken to have length 0, and the two atoms have collided. Therefore:

$$p^{(0)}_1(j + 1) - p^{(0)}_2(j + 1) = (0, 0, 0) \ .$$

The resulting $\nabla C(\overrightarrow{p}^{(0)}_{j+1})$ is given by:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1}) = (\overrightarrow{p}^{(0)}_{j+1})^T S_{12} = (p_1^{(0)}(j+1) - p_2^{(0)}(j+1))^T \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$= (0,0,0) \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$= (0,0,0,0,0,0)$$

Fast Projection's matrix, in this case a scalar, is singular:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}^{(0)}_{j+1})^T = 0$$

SHAKE's matrix is also singular:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}_j)^T = 0$$

Thus, if a bond length shrinks to 0, and two atoms have the same atomic coordinates, a row of zeros is added to the gradient of the constraints, making this gradient matrix rank deficient.

4. Case 4 is presented in Figure 2.4. This case involve a lattice with 4 atoms, three atoms are labelled in Figure 2.4 (b), where the leftmost atom is assumed to be the first atom, with label 1. The constraint pairs are the following index pairs: $(1,2)$, $(1,4)$, $(2,3)$, $(3,4)$. Since the vector $p_1(j) - p_2(j)$ and $p_1(j) - p_4(j)$ *are the same*, $\nabla C(\overrightarrow{p}_j)$ is rank deficient, causing the matrix used by SHAKE to also be rank deficient. The following calculation provides a further clarification.

   The atomic coordinates of the 1st to 4th atoms of the lattice at the beginning of the time interval are: $(0,0,0)$, $(4,0,0)$, $(8,0,0)$, $(4,0,0)$ respectively. The beginning of

the time interval atomic coordinates give the following $\nabla C(\overrightarrow{p}_j)$ matrix:

$$\nabla C(\overrightarrow{p}_j) = \begin{pmatrix} (\overrightarrow{p}_j)^T S_{12} \\ (\overrightarrow{p}_j)^T S_{14} \\ (\overrightarrow{p}_j)^T S_{23} \\ (\overrightarrow{p}_j)^T S_{34} \end{pmatrix} = \begin{pmatrix} (p_1^{(0)}(j+1) - p_2^{(0)}(j+1))^T \begin{pmatrix} I_3 & -I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \\ (p_1^{(0)}(j+1) - p_4^{(0)}(j+1))^T \begin{pmatrix} I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -I_3 \end{pmatrix} \\ (p_2^{(0)}(j+1) - p_3^{(0)}(j+1))^T \begin{pmatrix} \mathbf{0}_{3\times3} & I_3 & -I_3 & \mathbf{0}_{3\times3} \end{pmatrix} \\ (p_3^{(0)}(j+1) - p_4^{(0)}(j+1))^T \begin{pmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_3 & -I_3 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} ((0,0,0) - (4,0,0))^T \begin{pmatrix} I_3 & -I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \\ ((0,0,0) - (4,0,0))^T \begin{pmatrix} I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -I_3 \end{pmatrix} \\ ((4,0,0) - (8,0,0))^T \begin{pmatrix} \mathbf{0}_{3\times3} & I_3 & -I_3 & \mathbf{0}_{3\times3} \end{pmatrix} \\ ((8,0,0) - (4,0,0))^T \begin{pmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_3 & -I_3 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} (-4,0,0)^T \begin{pmatrix} I_3 & -I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \\ (-4,0,0)^T \begin{pmatrix} I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -I_3 \end{pmatrix} \\ (-4,0,0)^T \begin{pmatrix} \mathbf{0}_{3\times3} & I_3 & -I_3 & \mathbf{0}_{3\times3} \end{pmatrix} \\ (4,0,0)^T \begin{pmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_3 & -I_3 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} -4 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 4 & 0 & 0 & -4 & 0 & 0 \end{pmatrix}.$$

$\nabla C(\overrightarrow{p}_j)$ is a rank 3 matrix, this can be verified using singular value decomposition, thus it is rank deficient.

The atomic coordinates of the 1st to 4th atoms of the lattice at the end of the unconstrained step are: $(2,0,0)$, $(4,2,0)$, $(6,0,0)$, $(4,-2,0)$ respectively. The unconstrained

step atomic coordinates gives the following $\nabla C(\overrightarrow{p}^{(0)}_{j+1})$ matrix:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1}) = \begin{pmatrix} (\overrightarrow{p}^{(0)}_{j+1})^T S_{12} \\ (\overrightarrow{p}^{(0)}_{j+1})^T S_{14} \\ (\overrightarrow{p}^{(0)}_{j+1})^T S_{23} \\ (\overrightarrow{p}^{(0)}_{j+1})^T S_{34} \end{pmatrix} = \begin{pmatrix} (p^{(0)}_1(j+1) - p^{(0)}_2(j+1))^T \begin{pmatrix} I_3 & -I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \\ (p^{(0)}_1(j+1) - p^{(0)}_4(j+1))^T \begin{pmatrix} I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -I_3 \end{pmatrix} \\ (p^{(0)}_2(j+1) - p^{(0)}_3(j+1))^T \begin{pmatrix} \mathbf{0}_{3\times3} & I_3 & -I_3 & \mathbf{0}_{3\times3} \end{pmatrix} \\ (p^{(0)}_3(j+1) - p^{(0)}_4(j+1))^T \begin{pmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_3 & -I_3 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} ((2,0,0) - (4,2,0))^T \begin{pmatrix} I_3 & -I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \\ ((2,0,0) - (4,-2,0))^T \begin{pmatrix} I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -I_3 \end{pmatrix} \\ ((4,2,0) - (6,0,0))^T \begin{pmatrix} \mathbf{0}_{3\times3} & I_3 & -I_3 & \mathbf{0}_{3\times3} \end{pmatrix} \\ ((6,0,0) - (4,-2,0))^T \begin{pmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_3 & -I_3 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} (-2,-2,0) \begin{pmatrix} I_3 & -I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \\ (-2,2,0) \begin{pmatrix} I_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -I_3 \end{pmatrix} \\ (-2,2,0) \begin{pmatrix} \mathbf{0}_{3\times3} & I_3 & -I_3 & \mathbf{0}_{3\times3} \end{pmatrix} \\ (-2,-2,0) \begin{pmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & I_3 & -I_3 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} -2 & -2 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 \\ 0 & 0 & 0 & -2 & 2 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 & 2 & 2 & 0 \end{pmatrix}.$$

$\nabla C(\overrightarrow{p}^{(0)}_{j+1})$ is a full rank matrix. Multiplying $\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}^{(0)}_{j+1})^T$ shows this is also a full rank matrix:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}^{(0)}_{j+1})^T = \begin{pmatrix} 16 & 0 & 0 & 0 \\ 0 & 16 & 0 & 0 \\ 0 & 0 & 16 & 0 \\ 0 & 0 & 0 & 16 \end{pmatrix}$$

Multiplying out $\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}_j)^T$ gives:

$$\nabla C(\overrightarrow{p}^{(0)}_{j+1}) \nabla C(\overrightarrow{p}_j)^T = \begin{pmatrix} 16 & 8 & -8 & 0 \\ 8 & 16 & 0 & 0 \\ -8 & 0 & 16 & 8 \\ 0 & -8 & 8 & 16 \end{pmatrix},$$

which is not full rank.

This concludes the review of the four cases discussed by Barth et al.[9] which was used by

Goldenthal et al. [30] to show the implicit constraint force is more advantageous.

Note that these four cases showed two situations where the gradient of the constraint can become rank deficient. The first situation is as follows. Let $\overrightarrow{p}$ denote the vector of atomic coordinates, and let $p_a$ denote the atomic coordinate of the $a$-th atom. For a constraint on the distance between the $a$-th and $b$-th atom, if

$$p_a - p_b = \mathbf{0}_3 \ ,$$

then $\nabla C(\overrightarrow{p})$ will have a row of zeros and is therefore rank deficient. This means a force pushing these two atoms away from each other cannot be found.

In the second situation, suppose the $a$-th atom is involved in *more than one* constraint. For all constraints that the $a$-th atom is involved in, if there exists atomic indices $b_1$ and $b_2$ such that,

$$p_a - p_{b_1} = \alpha(p_a - p_{b_2}) \ ,$$

where $\alpha \in \mathbb{R}$ is an arbitrary constant, then $\nabla C(\overrightarrow{p})$ is rank deficient. This means in particular that three atoms cannot be constrained to be collinear.

#### Speed and Accuracy of Fast Projection

Fast Projection is faster and more accurate than other iterative methods. This is presented in Figure 5 of [30], where Fast Projection was compared with ICD, Shake, Jacobi, and Gauss-Siedel. Figure 5 has been reproduced in Figure 2.5. The main conclusions from Figure 5 are:

- Fast Projection uses less time than all other algorithms to enforce constraints to higher accuracy (less strain). See Figure 5 a) of [30].
- As $n$ and $m$ increase, Fast Projection's computing time increases the slowest among all four algorithms. See Figure 5 b) of [30].

### 2.8.4 Termination of Fast Projection

The convergences of Rosen's GPM, of which Fast Projection is a part of, is a research direction of itself. See for example Rosen's original work [66, 67], and also the publications from Du and Zhang [27, 29, 28].

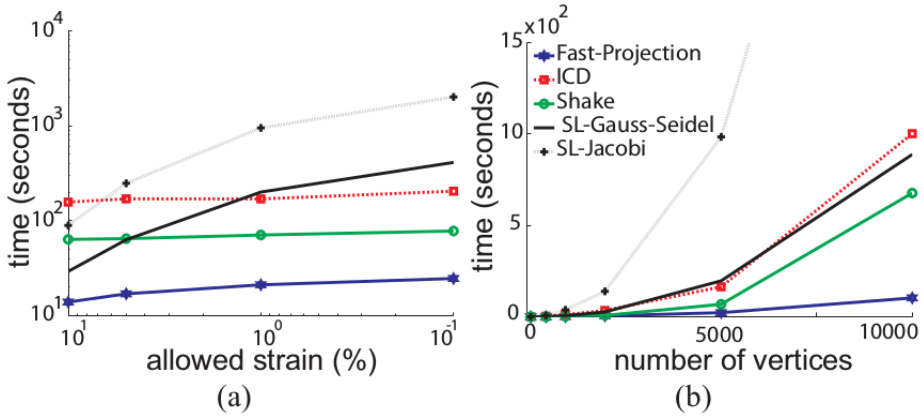



Figure 2.5: Figure 5 of [30]. This figure shows Fast Projection's performance is the fastest when compared with four other iterative constraint enforcing algorithms.

Assuming that the matrix in Fast Projection's linear system is full rank, and assuming a sequence of iterates converge to the constraint manifold exists, the following proof shows Fast Projection will find this sequence of iterates and terminate (converge).

**Theorem 2.8.2** (Termination of the Fast Projection Algorithm)**.** *Assume that the matrix in Fast Projection's linear system is full rank for all iterations $k$:*

$$\left(\nabla C(\overrightarrow{p}_{j+1}^{(k)})\nabla C(\overrightarrow{p}_{j+1}^{(k)})^T\right) .$$

*A sequence of iterates converge to the constraint manifold exists if and only if Fast Projection terminates.*

*Proof.* If a sequence of iterates approaching the constraint manifold exists, then

$$\delta\overrightarrow{p}_{j+1}^{(k)} \to \mathbf{0}_{3n} \text{ as } k \to \infty .$$

By Equation (2.76)

$$\nabla C(\overrightarrow{p}_{j+1}^{(k)})\delta\overrightarrow{p}_{j+1}^{(k)} = -C(\overrightarrow{p}_{j+1}^{(k)}) .$$

This means

$$C(\overrightarrow{p}_{j+1}^{(k)}) \to \mathbf{0}_m$$

and hence $\| C(\overrightarrow{p}_{j+1}^{(k)}) \| \to 0$, and Fast Projection will terminate.

Conversely, starting with the assumption that Fast Projection terminates, then the following must be true:

$$C(\overrightarrow{p}_{j+1}^{(k)}) \to \mathbf{0}_m \ .$$

But from Equation (2.78)

$$(\Delta t)^2 \left( \nabla C(\overrightarrow{p}_{j+1}^{(k)}) M^{-1} \nabla C(\overrightarrow{p}_{j+1}^{(k)})^T \right) \delta\lambda^{(k)} = C(\overrightarrow{p}_{j+1}^{(k)}) \to \mathbf{0}_m \ .$$

Since $\left( \nabla C(\overrightarrow{p}_{j+1}^{(k)}) \nabla C(\overrightarrow{p}_{j+1}^{(k)})^T \right)$ is full rank and invertible, this must mean

$$\delta\lambda^{(k)} = \frac{1}{(\Delta t)^2} \left( \nabla C(\overrightarrow{p}_{j+1}^{(k)}) \nabla C(\overrightarrow{p}_{j+1}^{(k)})^T \right)^{-1} C(\overrightarrow{p}_{j+1}^{(k)}) \to \mathbf{0}_m \text{ as } k \to \infty$$

From Equation (2.79), this also means

$$\delta \overrightarrow{p}_{j+1}^{(k)} = -(\Delta t)^2 M^{-1} \nabla C(\overrightarrow{p}_{j+1}^{(k)})^T \delta\lambda^{(k)} \to \mathbf{0}_{3n} \ .$$

Thus a sequence of iterates that approach the constraint manifold has been generated. □

### 2.8.5 Remark: Rosen's Correction and Fast Projection

Rosen's Correction and Fast Projection are the same algorithm developed independently (Goldenthal et al. did not cite Rosen's work). This independence is significant in its own right in that both authors emphasized different mathematics [1]. Rosen emphasized generalizing from linear constraints to nonlinear constraints; Goldenthal et al. emphasized using an implicit constraint force.

However, this independence also shows a serious defect of the approach on $\mathbb{R}^{3n}$: the de-emphasis of applying "abstract" concepts to different problem contexts.

This is not the case for the matrix manifold optimization paradigm, which applies abstract concepts from differential geometry to formulate optimization algorithms for a wide range of matrix manifolds. For example, the particular operation performed by both Rosen's Correction and Fast Projection is called a *retraction* [1, 2], different matrix manifolds implement their own retractions.

[1] Recall Newton, Lagrange, and Hamilton showed the equations of motion can be derived using different approaches.

## 2.9 Summary

This chapter described the modelling of ENMs on $\mathbb{R}^{3n}$. The Hookean potential energy was introduced, this potential has been used in NMA and to interpolate transitional conformations between a given beginning and ending conformation, as well as for avoiding atomic collisions.

Enforcing constraints on $\mathbb{R}^{3n}$ has also been discussed. A review of Goldenthal et al.'s derivation of the Fast Projection algorithm was given. Fast Projection was already known to Rosen as early as 1961. The fact that optimization algorithms in $\mathbb{R}^{3n}$ do start with abstract concepts, and apply these abstract concepts to different problems, is one of the serious defects of modelling protein ENMs using this space. In this case, Fast Projection could have been introduced earlier.

# Chapter 3

# Unconstrained Optimization with Fixed Rank PSD Matrices

## 3.1 Introduction

This Chapter reviews the result that the EDM completion (EDMC) problem can be formulated as a fixed rank PSD matrix manifold optimization problem. This Chapter's discussion is based on the publication from Mishra et al. [58]. Since a protein structure's Gram matrix is a point on $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n\times 3}/\mathbf{O}_3$, the mathematics reviewed in this Chapter are fundamental for modelling ENMs on the rank 3 PSD matrix manifold. Despite the close connection between the EDMC problem and modelling of ENMs, these two areas has to date developed independently from each other.

This Chapter is organized as follows:

1. Section 3.2 defines the EDMC problem.
2. Section 3.3 introduces the PSD objective function of [3].
3. Section 3.4 reviews the approach from [58] of solving the EDMC problem by solving a sequence of fixed rank PSD matrix manifold optimization problems. This formulation is reviewed as an example of formulating optimization problems on the matrix manifold $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$ without any constraints.
4. Section 3.5 introduces the Riemannian trust region (RTR) algorithm which was used in [58] to solve the fixed rank problems.

5. Section 3.6 discusses the geometric objects required by matrix manifold optimization algorithms, such as the RTR algorithm, for the fixed rank EDMC problem.

Constrained optimization on $\mathbf{S}_+^{n,r}$ has been discussed by Journée et. al. [34] not in the context of the EDMC problem, but for formulating the *max-cut SDP relation* problem and the *sparse PCA problem*. Journée et. al.'s formulas for geometric objects for constrained optimization will be discussed in Chapter 6 and modified in order to describe the constraints of a protein structure. See also Section C.6 for a discussion on facial reduction and the EDMC problem.

## 3.2 The EDM Completion (EDMC) Problem

The EDM and its embedding dimension has been defined in Section C.4. The definition of the EDMC problem is given in Definition 3.2.1. The EDMC problem is an example of the *imputation of missing values* task, one of the examples of machine learning problems given in Goodfellow et al. [32].

**Definition 3.2.1** (The EDM Completion (EDMC) Problem )**.** *For a collection of $n$ points, assume some of the quadrances between the points can be specified, while the quadrances between other pairs of points cannot be specified. The EDM for these points is therefore an incomplete EDM.*

*The EDMC problem takes as input an incomplete EDM, $\overline{D}$, where some matrix entries are specified, while other entries are unspecified. The EDMC problem produces an output EDM, $D$, such that if the $ab$-th entry in $\overline{D}$ is specified, then the $ab$-th entry in the output EDM is the same:*

$$D_{ab} = \overline{D}_{ab} \ ,$$

*and the unspecified entries in $\overline{D}$ are determined in the output EDM, $D$.*

*The EDMC problem can be defined using the language of* graph theory *(see for example [44] page 37). Associated with an $n \times n$ incomplete EDM is a weighted undirected graph*

$$G = (N, E, w)$$

*where $N$ is the set of nodes of the graph, defined as:*

$$N = \{1, \ldots, n\} \ .$$

*$E$ is the set of undirected edges, defined as:*

$$E = \{ab \mid a, b \in N, a \neq b \text{ and } \overline{D}_{ab} \text{ is specified.}\} .$$

*These edges are undirected so that the edge $ab$ is the same as the edge $ba$. The set $w$ is the set of weights for the edges.*

$$w = \{w_{ab} = \overline{D}_{ab} \quad \forall ab \in E\} .$$

*The $0-1$ adjacency matrix $H$ for the graph $G$ is defined as:*

$$H_{ab} = H_{ba} = \begin{cases} 1 & \text{if } ab \in E \\ 0 & \text{if } ab \notin E \end{cases}$$

*The graph $G$ is called the* input EDM graph.

*The EDMC problem takes the input EDM graph $G$ and produces as output a new graph where all possible pairs of undirected edges are present with their weights. If an edge, $ab \in E$, is already in the input graph, $E$, with weight $w_{ab}$, the edge appears in the output graph with the same weight.*

The problem of determining protein structures from Nuclear Magnetic Resonance (NMR) spectroscopy data can be formulated as an EDMC problem. This problem is defined in Definition 3.2.2. The Protein Structure-NMR problem is not the problem addressed in this thesis. It has been formulated as a PSD optimization problem by Alipanahi et al. in [5, 4].

**Definition 3.2.2** (Protein Structure Determination from Nuclear Magnetic Resonance (NMR) spectroscopy data (Protein Stucture-NMR))**.** *Nuclear Magnetic Resonance (NMR) can be used to measure inter-atomic distances between protein atoms that are within the range of 5 to 6 Å apart, where $1\text{Å} = 10^{-10}m$. These are the "specified entries" of a protein's EDM. Larger range inter-atomic distances are not known, and are the "unspecified entries" of a protein's EDM. These unspecified entries must be determined in order to determine the position of all of the protein's atoms.*

The EDMC problem has applications for areas outside of protein structure modelling as well. For example, another application is in sensor network localization. Further discussion of solving SNL via EDMC can be found in [45, 44, 46].

**Definition 3.2.3** (Sensor Network Localization (SNL))**.** *Assume locations of sensors are given by their 3-D Cartesian coordinates. Assume that sensors are aware of their distances*

*to nearby sensors. Assume also that there is a large number of sensors and the use of a global positioning system (GPS) to determine all of their locations is very expensive.*

*The SNL problem takes as input a list of distances between each sensor and its neighbours, subject to a threshold distance.*

*The SNL problem outputs the 3-D Cartesian coordinates of all the sensors.* Localization *means to determine the 3-D Cartesian coordinate of a sensor.*

### 3.2.1 Low Embedding Dimension EDMC

The SNL problem, given in Definition 3.2.3 and the Protein Stucture-NMR problem, given in Defintion 3.2.2 are both examples of EDMC problems where a *low embedding dimension* solution is required. This is because both sensors and protein atoms are points that exist in 3D space, and thus the EDM has embedding dimension 3, which is small compared to the total number of points. As discussed in Section 2.6.4 of Krislock [44], the low embedding dimension EDMC problem is *NP-hard* in general, but specific instances of this problem can be solved efficiently, as the work of Krislock and Wolkowicz [44, 45], and Mishra et al. [58] has shown. Using *rank minimizing heuristics* to formulate the EDMC problem is one research direction for finding low rank EDMs. This approach is reviewed in Section 2.6.7 of Krislock [44].

The approach proposed by Mishra et al. [58] finds low embedding dimension EDMs by solving a sequence of *fixed rank* PSD matrix manifold optimization problems, starting with rank 1. Since the mathematics of the fixed rank problem is fundamental for modelling ENMs, it is used as an example to introduce fixed-rank PSD matrix manifold optimization in Section 3.4 and Section 3.6.

## 3.3 The PSD Objective Function for EDMC

Suppose there are $n$ points, and the EDM of these points, $\overline{D}$, is incomplete. The EDMC problem was formulated as a semidefinite optimization problem in Alfakih (1999) [3] with the objective function:

$$f(X) = \| H \circ (\mathcal{K}(X) - \overline{D}) \|_F^2 \ . \tag{3.1}$$

The matrix $X$ is an $n \times n$ PSD matrix, $X \succeq 0$. The matrix $H$ is the $0-1$ adjacency matrix defined in Definition 3.2.1. The operation $\circ$ represents element-wise matrix multiplication.

The matrix $\overline{D}$ is the partial EDM whose missing entries needs to be determined, and $\mathcal{K}(\cdot)$ is the linear isomorphism between the PSD cone and the EDM cone, see Section C.5.

The objective function given by Equation (3.1) has the meaning of finding the closest EDM $\mathcal{K}(X)$ to the input incomplete EDM $\overline{D}$. The EDM $\mathcal{K}(X)$ and $\overline{D}$ are considered close if the squared Frobenius norm, $\|\cdot\|_F^2$, of their difference is small, which is what the objective function is measuring.

Other aspects of the EDMC problem discussed in [3] include:

- Ensuring that $X$ is a *centered Gram matrix*. This means $X\mathbf{1}_n = \mathbf{0}_n$, this constraint means the coordinates of the points is centered at the origin. This constraint reduces the problem from finding an $n \times n$ matrix $X$ to a problem of finding an $(n-1)\times(n-1)$ matrix $X$. That is, the problem is formulated on the cone $\mathbf{S}_+^{n-1}$ rather than $\mathbf{S}_+^n$.

  Note that this is the same constraint as the constraint for conservation of linear momentum used in modelling ENMs on $\mathbb{R}^{3n}$, see Section 2.7.

- Quadrance constraints can also be added to the optimization problem.

Finally, a primal-dual interior-point algorithm was developed to solve this optimization problem. This solution approach is not explored in this thesis.

Meyer [55] pointed out that because of the linear isomorphism $\mathcal{K}(\cdot)$, the EDMC problem when viewed as an optimization problem on $\mathbf{S}_+^n$ or $\mathbf{S}_+^{n,r}$ becomes a *linear regression problem*. See also Section 8.1.

## 3.4 Fixed Rank PSD Matrix Manifold Optimization

Solving low rank PSD optimization problems have been examined by Journée et al. [34] where an algorithm that solves a sequence of *fixed rank* PSD matrix manifold optimization problems was introduced. The authors did not give a name to that algorithm, referring to it only as a "meta algorithm". For ease of discussion, this meta algorithm is referred to here as "the rank incremental algorithm".

Mishra et al. [58] used this rank incremental algorithm to solving the low embedding dimension EDMC problem, the Matlab code can be found at `https://bamdevmishra.in/codes/edmcompletion/`. The geometry of the fixed rank problem is relevant to modelling protein structures because a protein's Gram matrix is a fixed rank (rank 3) PSD matrix. Since Gram matrices of other ranks are not relevant to modelling protein ENMs, the details

of the rank incremental algorithm will not be discussed, see Journée et al. [34], Mishra et al. [58], and the Matlab code for further details. The discussion from this point on will focus on solving the fixed rank problem only.

An optimization problem that requires its solution to be a fixed rank Gram matrix is an optimization problem on the Riemannian manifold $\mathbf{S}_+^{n,r}$, where $r$ denotes the fixed rank. Such an optimization problem can be solved using either the gradient descent algorithm or the trust region algorithm generalized to $\mathbf{S}_+^{n,r}$ [58].

Optimization algorithms on $\mathbb{R}^{3n}$ (equivalently $\mathbb{R}^n$) can be generalized to matrix manifolds, which are Riemannian manifolds where a point on the manifold can be represented by a matrix, see [1]. Appendix D provides the background needed to define a Riemannian manifold, and the compact Stiefel manifold, the non-compact Stiefel manifold, the Grassmann manifold, and the fixed rank PSD matrix manifolds are given as examples of matrix manifolds.

Absil et al. [1] has used abstract differential geometry as a foundation, and showed how such abstract ideas can be applied to various matrix manifolds.

The trust region algorithm generalized to Riemannian manifolds is called the *Riemannian trust region (RTR)* algorithm. The trust region algorithm has the desirable superlinear convergence properties of the Newton algorithm, but is much less sensitive to initial conditions than the Newton algorithm (see [1] Chapter 7). Another advantage of the trust region algorithm over Newton's algorithm is that the subproblem is solved by the tCG algorithm, which *does not require the Hessian matrix of the objective function to be formed explicitly*, nor the inverse of the Hessian matrix to be found. This means the tCG algorithm can be conveniently generalized to the situation where the Riemannian manifold is not $\mathbb{R}^{3n}$, where the Hessian matrix is instead given as a *linear operator* acting on a matrix or vector, as seen in Section 3.6.5. Due to these advantages, the trust region algorithm will be used in this thesis.

Mishra et al.[58] showed the trust region algorithm is preferable to gradient descent for the EDMC problem following reasons:

- The trust region algorithm requires less iterations for convergence than gradient descent in the EDMC problem (see discussion in [58] Section V and VI).
- As the number of points $n$ increases, the trust region algorithm has been observed to scale better than gradient descent for the EDMC problem. (see discussion in [58] Section VI).

In Section 3.5 and Section 3.6, and the rest of this section, the Gram matrix factorization will be denoted as:

$$X = YY^T \ ,$$

instead of:

$$X = PP^T .$$

The reason for this notation change is that $P$ has been defined to be an $n \times 3$ matrix in this thesis. However, the fixed rank problem currently being discussed is a subroutine called by the rank incremental algorithm. At the beginning of the rank incremental algorithm, the $Y$ given as input to the fixed rank problem is $n \times 1$, in the next iteration of the rank incremental algorithm $Y$ is $n \times 2$, and so on. Thus, $Y$ will be defined to be an $n \times r$ matrix, instead of an $n \times 3$ matrix.

The fixed rank problem solved by Mishra et al. [58] is:

$$\min_{YY^T \in \mathbf{S}_+^{n,r}} f(YY^T) = \| \ H \circ (\mathcal{K}(YY^T) - \overline{D}) \ \|_F^2 \ , \tag{3.2}$$

where $f(YY^T)$ is the objective function for the fixed rank problem. Equation (3.2) means the optimization problem will find the rank $r$ PSD matrix $YY^T$ whose corresponding EDM, $\mathcal{K}(YY^T)$, is closest to $\overline{D}$. The is the same objective function as Equation (3.1), except the Gram matrix is factorized.

The factorization of the Gram matrix as $X = YY^T$ is invariant to the transformation:

$$Y \to YQ \ ,$$

where $Q \in \mathbf{O}_r$, and $\mathbf{O}_r = \{Q | Q^T Q = QQ^T = I_r\}$, $I_r$ is the $r \times r$ identity matrix. Since $X$ is rank $r$ and $Y$ is an $n \times r$ matrix of *maximal rank* $r$, $Y$ is an element of the non-compact Stiefel manifold, $Y \in \mathbb{R}_*^{n \times r}$. The optimization problem in Equation (3.2) is defined on the quotient manifold:

$$\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n \times r} / \mathbf{O}_r$$

This quotient manifold has equivalence classes $[Y]$ defined as:

$$[Y] = \{YQ | Q \in \mathbf{O}_r\} \ . \tag{3.3}$$

That is, the matrix $YQ$ is the same point as the matrix $Y$ on the quotient manifold $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n \times r} / \mathbf{O}_r$, for any $Q \in \mathbf{O}_r$.

When solving optimization problems on quotient manifolds, *any* matrix from an equivalence class can be used to represent that equivalence class. This is given as the following definition

for a point on $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$.

**Definition 3.4.1** (A point on the quotient manifold $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$)**.** $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$ *is a quotient manifold and each point on the manifold is an equivalence classes given by Equation (3.3) with potentially an infinite number of elements. In numerical algorithms and when storing in computer memory,* any *matrix* $Y \in [Y]$ *can be used to represent its equivalence class, and mathematical formulas are expressed using this "representative matrix". This means a point on* $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$ *can be represented simply by the matrix* $Y$*.*

Following Definition 3.4.1, expressions like $[Y][Y]^T$ will not appear in the mathematical formulas relating to the fixed rank EDMC problem. The simpler notation $YY^T$ will still be used. The notation $f(YY^T)$ and $f(Y)$ will both be used to denote the objective function in Equation (3.2). The notation $f(Y)$ is preferable to $f(YY^T)$ since it is easier to read.

As discussed in [78] and Section D.6.5, $\mathbf{S}_+^{n,r}$ is diffeomorphic to many geometries. In this thesis, unless stated otherwise, the diffeomorphism $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$ will always be assumed, similarly for protein ENMs, $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n\times 3}/\mathbf{O}_3$ is assumed.

## 3.5 The Riemannian Trust Region (RTR) Algorithm

The RTR algorithm can be used to solve the fixed rank problem posed in Equation (3.2). The RTR algorithm requires the following geometric objects, they are to be defined in Section 3.6:

- The Riemannian metric on the tangent space. See Section 3.6.1.
- The tangent space, $T_Y\mathbf{S}_+^{n,r}$. See Section 3.6.2.
- A projection to the horizontal space, $\Pi_{\mathcal{H}_Y}(\cdot)$. See Section 3.6.3.
- The Riemannian gradient, $\mathrm{grad} f(Y)$. See Section 3.6.4.
- The Riemannian Hessian in a direction $\eta$, $\mathrm{Hess} f(Y)[\eta]$. See Section 3.6.5
- The retraction from the tangent space to the matrix manifold, $R_Y(\eta)$. See Section 3.6.6.

The reminder of this section presents a review of the RTR algorithm.

Since this optimization problem has no constraints, any point $Y \in \mathbf{S}_+^{n,r}$ is a *feasible point*, which means a point the optimization algorithm will consider as a candidate solution. The feasible region, which is the set of all feasible points, is the entire manifold $\mathbf{S}_+^{n,r}$. The feasible region is also called the *search space* since this is the space the optimization algorithm will search for the optimal $Y$ which will minimize the objective function.

For a given rank $r$, the RTR algorithm generates a sequence of iterates of $n \times r$ matrices, $Y_0, \ldots, Y_k$ such that:

$$\lim_{k\to\infty} \text{grad} f(Y_k) = \mathbf{0}_{n\times r} \ . \tag{3.4}$$

This means $Y_k$ approaches a critical point of the objective function $f(Y)$ defined in Equation (3.2) (see [1] Theorems 7.4.2 and 7.4.4). $Y_0$ is the initial iterate, it is an input to the algorithm.

At the $k$-th iteration of the RTR algorithm, an update vector $\eta$ is found by solving the trust region subproblem. Let the trust region radius at the $k$-th iteration be given by $\Delta_k$, the *trust region subproblem* is the following optimization problem:

$$\begin{aligned} &\min \quad m_k(\eta) = f(Y_k) + \langle \text{grad} f(Y_k), \eta\rangle + \frac{1}{2}\langle \eta, \text{Hess} f(Y_k)[\eta]\rangle \ , \\ &\text{s.t.} \quad \langle \eta, \eta\rangle \leq \Delta_k^2 \ , \\ &\text{where} \quad \eta \in T_{Y_k}\mathbf{S}_+^{n,r} \ . \end{aligned} \tag{3.5}$$

The tCG algorithm, given in Algorithm 3, is used to solve the trust region subproblem, this is following the suggestion by Absil et al. , see Algorithm 11 [1], see also page 205 of [18] which discusses some helpful recurrences.

Let $\eta^*$ be the output of the tCG algorithm that solves the trust region subproblem. This $\eta^*$ is accepted or rejected, and the trust region radius is also increased or decreased, based on the ratio $\rho_k$:

$$\rho_k = \frac{f(Y_k) - f(R_{Y_k}(\eta^*))}{m_k(0) - m_k(\eta^*)} \ . \tag{3.6}$$

The meaning of $\rho_k$ is to compare the decrease between the objective function $f(R_{Y_k}(\eta^*))$, with the quadratic approximation $m_k(\eta^*)$. This comparison is an indication of how well the quadratic model $m_k(\eta^*)$ is in agreement with the objective function. If the agreement is "very good", the trust region radius is increased, if the agreement is "not very good", the trust region radius is decreased. "Very good" or not is determined as follows. Let $\rho' \in [0, \frac{1}{4})$, e.g. $\rho' = 0.1$, and consider the following possibilities:

- If $\rho_k < \rho'$, the objective function, $f(R_{Y_k}(\eta^*))$, changed very little or may have even increased meaning the quadratic model $m_k(\eta^*)$ from Equation (3.5) is a very inaccurate approximation of the objective function. In this case, $\eta^*$ is rejected, and the trust-region radius is decreased.

- If $\rho_k > \rho'$ but it is still small (e.g. $\rho_k < \frac{1}{4}$), the decrease in the objective function is considered a reasonable progress, but the quadratic model $m_k(\eta^*)$ is still not a very good approximation of the objective function $f(R_{Y_k}(\eta^*))$. Therefore $\eta^*$ is accepted, but the trust region radius is still decreased.

- If $\rho_k > \rho'$ and $\rho_k \approx 1$, then there is good agreement between the quadratic model $m_k(\eta^*)$ from Equation (3.5) and the objective function $f(R_{Y_k}(\eta^*))$. Since $\rho_k > \rho'$, good progress has been made in decreasing the cost function so $\eta^*$ is accepted. Since the quadratic model $m_k(\eta^*)$ is in close agreement with the objective function, the trust region radius is increased, with the hope the quadratic model will still be a good approximation of the objective function for the bigger trust region.

- If $\rho_k \gg 1$, then the quadratic model $m_k(\eta^*)$ is a very inaccurate approximation of the objective function $f(R_{Y_k}(\eta^*))$, but nonetheless, the tCG algorithm has found a step that greatly reduces the objective function. In this case, the trust region radius is increased, with the hope a big decrease in the objective function can be found again.

The above ideas lead to the RTR algorithm in Algorithm 2. The RTR algorithm stops when the Riemannian gradient grad$f(Y_k)$ has a very small norm, see the condition given in Equation (3.4).

## 3.6 Geometric Objects for EDMC on $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$

In Absil et al.'s [1] development of matrix manifold optimization, differential geometry plays a central role. This abstraction allows the connection between optimization algorithms on $\mathbb{R}^{3n}$ and other matrix manifolds to be made clear. In Nocedal and Wright's well-known book [61], the discussion is restricted to $\mathbb{R}^{3n}$, and abstract differential geometry plays a lesser role.

Matrix manifold algorithms, such as the RTR algorithm discussed in Section 3.5 require various geometric objects to be defined for the matrix manifold in question. These objects are discussed in the following subsections for the EDMC problem on the matrix manifold $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$ .

**Algorithm 2** Riemannian Trust Region (RTR)

**INPUT:** Initial iterate $Y_0$, $\overline{\Delta} > 0$, $\Delta_0 \in (0, \overline{\Delta})$, and $\rho' \in [0, \frac{1}{4})$.
**OUTPUT:** Sequence of iterates $\{Y_k\}$

**for** $k = 0, 1, 2, \ldots$ **do**
  Obtain $\eta_k$ using Algorithm 3.
  Evaluate $\rho_k = \frac{f(Y_k) - f(R_{Y_k}(\eta_k))}{m_k(0) - m_k(\eta_k)}$.
  **if** $\rho_k < \frac{1}{4}$ **then**
    $\Delta_{k+1} = \frac{1}{4}\Delta_k$
  **else if** $\rho_k > \frac{3}{4}$ and $\| \eta_k \| = \Delta_k$ **then**
    $\Delta_{k+1} = \min(2\Delta_k, \overline{\Delta})$
  **else**
    $\Delta_{k+1} = \Delta_k$
  **end if**
  ---
  **if** $\rho_k > \rho'$ **then**
    Accept $Y_{k+1} = R_{Y_k}(\eta_k)$.
  **else**
    Reject $Y_{k+1} = Y_k$.
  **end if**
  Check stopping criterion, for example:
  **if** $\| \operatorname{grad} f(Y_{k+1}) \|_F < \epsilon$ for small $\epsilon$ **then**
    Algorithm Terminates.
  **end if**
**end for**

**Algorithm 3** Truncated Conjugate Gradient (tCG) for the trust-region subproblem

Set $j = 0$, $\eta^0 = 0$, $r_0 = \text{grad}f(Y_k)$, $\delta_0 = -r_0$.
**for** $j = 0, 1, 2, \ldots$ **do**
    Set $\alpha_j = \langle r_j, r_j \rangle / \langle \delta_j, \text{Hess}f(Y_k)[\delta_j] \rangle$.
    Set $\eta^{j+1} = \eta^j + \alpha_j \delta_j$.
    **if** $\langle \delta_j, \text{Hess}f(Y_k)[\delta_j] \rangle \leq 0$ or $\| \eta^{j+1} \| \geq \Delta$ **then**
        Compute $\tau \geq 0$ such that $\eta = \eta^j + \tau \delta_j$ satisfies $\| \eta \| = \Delta$.
        $\tau$ is given by the positive root of

$$\tau^2 \langle \delta_j, \delta_j \rangle + 2\tau \langle \eta^j, \delta_j \rangle = \Delta^2 - \langle \eta^j, \eta^j \rangle \tag{3.7}$$

        Return $\eta$
    **end if**
    Set $r_{j+1} = r_j + \alpha_j \text{Hess}f(Y_k)[\delta_j]$
    **if** $\| r_{j+1} \| \leq \min(\| r_0 \|^\theta, \kappa)$ **then**
        Return $\eta^{j+1}$.
    **end if**
    Set $\beta_{j+1} = \langle r_{j+1}, r_{j+1} \rangle / \langle r_j, r_j \rangle$
    Set $\delta_{j+1} = -r_{j+1} + \beta_{j+1} \delta_j$.
**end for**

In each subsection, the geometric object for an abstract quotient manifold is presented first, then the corresponding geometric object for the EDMC problem is presented. This order of presentation attempts to follow the presentation in Absil et al. [1] to highlight the importance of abstract differential geometry.

#### The Notation $\overline{\mathcal{M}}$ vs. $\mathcal{M}$

When discussing an abstract quotient manifold (see definition in Section D.6.4), the overline notation

$$\overline{x} \in \overline{\mathcal{M}} \ ,$$

is used to denote a point on the structure space $\overline{\mathcal{M}}$, where the line is placed on both the structure space and the point. The notation

$$x \in \mathcal{M} = \overline{\mathcal{M}} / \sim \ ,$$

is used for a point on the abstract quotient manifold. Similarly, an overline is placed on tangent vectors on the tangent space of the structure space,

$$\overline{\eta} \in T_{\overline{x}} \overline{\mathcal{M}} \ ,$$

but not on the tangent space of the quotient manifold:

$$\eta \in T_x \mathcal{M} \ .$$

However, when discussing $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n \times r} / \mathbf{O}_r$ in particular, the overline notation will not be used to differentiate between objects in the structure space and the quotient manifold. This is to make the notation easier to read, and to emphasize that when implementing these formulas in a computer program these objects are not differentiated. Recall from Defintion 3.4.1, a point on $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n \times r} / \mathbf{O}_r$ is represented by a point $Y \in \mathbb{R}_*^{n \times r}$ with no overline placed on the matrix.

### 3.6.1 Riemannian Metric

Riemannian metrics were introduced in Section D.5. For an abstract quotient manifold $\mathcal{M} = \overline{\mathcal{M}} / \sim$, let the Riemannian metric defined on the tangent space of a point $\overline{x} \in \overline{\mathcal{M}}$ be denoted

$$\overline{g}(\cdot, \cdot) \ ,$$

and the Riemannian metric on the tangent space of $x \in \mathcal{M} = \overline{\mathcal{M}}/\sim$ be denoted

$$g(\cdot,\cdot)\ .$$

If $\overline{g}(\cdot,\cdot)$ *does not* change along the equivalence class $\pi^{-1}(x)$, then the same metric can be used for the quotient manifold, see [1] page 49 and [78] page 27. That is:

$$g(\cdot,\cdot) = \overline{g}(\cdot,\cdot)\ .$$

The manifold $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$ is a quotient manifold with structure space $\mathbb{R}_*^{n\times r}$, which is an embedded submanifold of the Euclidean space $\mathbb{R}^{n\times r}$. The Euclidean metric from $\mathbb{R}^{n\times r}$ defined by:

$$\text{Trace}(\eta^T\xi)$$

where $\eta,\xi \in \mathbb{R}^{n\times r}$, does not depend on $Y$. This Euclidean metric will be used as the Riemannian metric on the tangent space of a point $Y \in \mathbf{S}_+^{n,r}$.

### 3.6.2 Tangent Space

Figure 3.1 is a schematic diagram of a tangent space to an abstract quotient manifold $\mathcal{M} = \overline{\mathcal{M}}/\sim$, with structure space $\overline{\mathcal{M}}$.

In Figure 3.1, $x \in \mathcal{M} = \overline{\mathcal{M}}/\sim$ is a point on the quotient manifold, and $\overline{x} \in \overline{\mathcal{M}}$ is a point on the structure space. An equivalence class $\pi^{-1}(x) \subset \overline{\mathcal{M}}$ is an embedded submanifold of $\overline{\mathcal{M}}$. The tangent space of $\pi^{-1}(x)$ at a point $\overline{x} \in \pi^{-1}(x) \subset \overline{\mathcal{M}}$ is denoted $\mathcal{V}_{\overline{x}}\overline{\mathcal{M}}$ and is called the *vertical space* at $\overline{x}$ (see for example [1] and [78]). Mathematically, this is written as:

$$\mathcal{V}_{\overline{x}}\overline{\mathcal{M}} = T_{\overline{x}}(\pi^{-1}(x))$$

The *horizontal space* is the orthogonal complement of the vertical space:

$$\mathcal{V}_{\overline{x}}\overline{\mathcal{M}} \perp \mathcal{H}_{\overline{x}}\overline{\mathcal{M}}$$

The horizontal space and vertical space together make up the tangent space:

$$\mathcal{H}_{\overline{x}}\overline{\mathcal{M}} \oplus \mathcal{V}_{\overline{x}}\overline{\mathcal{M}} = T_{\overline{x}}\overline{\mathcal{M}}\ , \tag{3.8}$$

A mapping that assigns to each $\overline{x} \in \overline{\mathcal{M}}$ a horizontal space $\mathcal{H}_{\overline{x}}\overline{\mathcal{M}}$ at $\overline{x}$ is called a *horizontal distribution*.

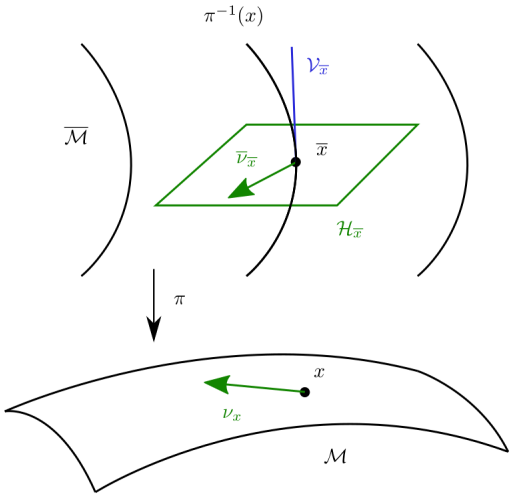



Figure 2.6: Tangent vector $\nu_x$ and its horizontal lift $\overline{\nu}_{\overline{x}}$.

Figure 3.1: Vertical and horizontal spaces. Figure 2.6 of (Vandereycken, 2010 [78]).

A tangent vector on the tangent space of the quotient manifold, $\eta \in T_x\mathcal{M}$, has a unique *horizontal lift* vector $\overline{\eta} \in \mathcal{H}_{\overline{x}}\overline{\mathcal{M}}$ on the horizontal space defined by the horizontal distribution. Thus, tangent vectors on the tangent space of the quotient manifold are represented by their horizontal lift in the horizontal space.[1]

Vectors in the horizontal space point to the direction of interest in optimization algorithms involving quotient manifolds.

For the quotient manifold $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$, the structure space is $\mathbb{R}_*^{n\times r}$. Tangent space of $\mathbb{R}_*^{n\times r}$ at a point $Y$ is denoted $T_Y\mathbb{R}_*^{n\times r}$. Proposition 3.6.1 shows $T_Y\mathbb{R}_*^{n\times r}$ is isomorphic to $\mathbb{R}^{n\times r}$.

**Proposition 3.6.1** ($T_Y\mathbb{R}_*^{n\times r} \simeq \mathbb{R}^{n\times r}$)**.** *Let $Y \in \mathbb{R}_*^{n\times r}$, the tangent space at the point $Y$, denoted $T_Y\mathbb{R}_*^{n\times r}$, is isomorphic to $\mathbb{R}^{n\times r}$, denoted:*

$$T_Y\mathbb{R}_*^{n\times r} \simeq \mathbb{R}^{n\times r} .$$

*Proof.* The first component of the vectors in $T_Y\mathbb{R}_*^{n\times r}$ are those vertical to the equivalence

[1]For $\mathcal{M} = \overline{\mathcal{M}}/\sim$, the notation $\mathcal{V}_{\overline{x}}\mathcal{M}$ and $\mathcal{H}_{\overline{x}}\mathcal{M}$ are seen in [58], [34], [55] page 28, [78] page 28. This notation is a bit confusing since the tangent space of a point on the quotient manifold should not have a vertical space. The notation $\mathcal{V}_{\overline{x}}\overline{\mathcal{M}}$ and $\mathcal{H}_{\overline{x}}\overline{\mathcal{M}}$ emphasizing the structure space seems to be a better notation.

classes defined in Equation (3.3):

$$[Y] = \{YQ \mid Q \in \mathbf{O}_r\} .$$

Let $\mathcal{V}_Y\mathbb{R}_*^{n\times r}$ denote the vertical space, these vectors are of the form:

$$\mathcal{V}_Y\mathbb{R}_*^{n\times r} = \{Y\Omega \mid \Omega \in \mathbb{R}^{r\times r} \ , \ \Omega^T = -\Omega\} . \tag{3.9}$$

The vertical space has $Y$ pre-determined, so its dimension is determined by the number of entries in the $r \times r$ skew symmetric matrix $\Omega$ that can be chosen freely. Thus, the dimension of $\mathcal{V}_Y\mathbb{R}_*^{n\times r}$ is:

$$\dim(\mathcal{V}_Y\mathbb{R}_*^{n\times r}) = \frac{r(r-1)}{2} . \tag{3.10}$$

Let $\xi \in \mathcal{H}_Y\mathbb{R}_*^{n\times r}$, then $\xi$ is orthogonal to $\mathcal{V}_Y\mathbb{R}^{n\times r}$ meaning that:

$$\text{Trace}(\xi^T Y\Omega) = 0 .$$

Since $\Omega$ is a skew symmetric matrix, this implies $\xi^T Y$ is a symmetric matrix. It follows then:

$$\mathcal{H}_Y\mathbb{R}_*^{n\times r} = \{\xi \mid \xi^T Y = Y^T\xi\} . \tag{3.11}$$

Let the matrix $Y^\perp$ the an $(n-r) \times r$ matrix whose columns are all orthogonal to $Y$, that is:

$$Y^T Y^\perp = \mathbf{0}_{r\times(n-r)} \text{ and } (Y^\perp)^T Y = \mathbf{0}_{(n-r)\times r} . \tag{3.12}$$

Let $\xi = Y^\perp B$, where $B$ is any $(n-r) \times r$ matrix then

$$\xi^T Y = Y^T\xi = \mathbf{0}_{r\times r} ,$$

hence, $Y^\perp B \in \mathcal{H}_Y\mathbb{R}_*^{n\times r}$. Let $\xi = Y(Y^TY)^{-1}A$, where $A$ is an $r \times r$ symmetric matrix, this expression comes from [78] Section 6.6.2, then:

$$\xi^T Y = Y^T\xi \neq \mathbf{0}_{r\times r} ,$$

Therefore, in general, $\xi \in \mathcal{H}_Y\mathbb{R}_*^{n\times r}$ has the expression:

$$\xi = Y(Y^TY)^{-1}A + Y^\perp B . \tag{3.13}$$

The dimension of $\mathcal{H}_Y\mathbb{R}_*^{n\times r}$ is determined as follows. Firstly, observe $B$ has $(n-r)r$ entries that can be chosen freely. Next, recall an $r \times r$ symmetric matrix has $r(r+1)/2$ entries

that can be chosen freely, therefore:

$$\dim(\mathcal{H}_Y\mathbb{R}_*^{n\times r}) = (n-r)r + \frac{r(r+1)}{2} \ . \tag{3.14}$$

The tangent space at $Y \in \mathbb{R}_*^{n\times r}$ is defined as:

$$T_Y\mathbb{R}_*^{n\times r} = \mathcal{V}_Y\mathbb{R}_*^{n\times r} \oplus \mathcal{H}_Y\mathbb{R}_*^{n\times r} \ .$$

Equation (3.10) and (3.14) together show:

$$\begin{aligned}\dim(T_Y\mathbb{R}_*^{n\times r}) &= \frac{r(r-1)}{2} + (n-r)r + \frac{r(r+1)}{2} \\ &= \frac{r^2}{2} - \frac{r}{2} + nr - r^2 + \frac{r^2}{2} + \frac{r}{2} \\ &= nr \ .\end{aligned} \tag{3.15}$$

$T_Y\mathbb{R}_*^{n\times r}$ is a vector space, and it has dimension $nr$, therefore, it is isomorphic to $\mathbb{R}^{n\times r}$. □

Horizontal vectors in $\mathcal{H}_Y\mathbb{R}_*^{n\times r}$ at a point $Y$ make up the tangent space to $\mathbf{S}_+^{n,r}$ at the point $Y$. This is formalized in the Definition 3.6.1.

**Definition 3.6.1** (The tangent space at $Y \in \mathbf{S}_+^{n,r}$, $T_Y\mathbf{S}_+^{n,r}$)**.** *The tangent space at $Y \in \mathbf{S}_+^{n,r}$, denoted $T_Y\mathbf{S}_+^{n,r}$, is defined to be the set of all vectors from $\mathcal{H}_Y\mathbb{R}_*^{n\times r}$:*

$$T_Y\mathbf{S}_+^{n,r} = \{\xi \in \mathbb{R}^{n\times r} : \xi^T Y = Y^T\xi\} \ .$$

### 3.6.3 Projection onto the Horizontal Space

For an abstract quotient manifold, since $\mathcal{H}_{\overline{x}}\overline{\mathcal{M}} \oplus \mathcal{V}_{\overline{x}}\overline{\mathcal{M}} = T_{\overline{x}}\overline{\mathcal{M}}$, a projection onto these subspaces can be defined. The horizontal projection is a map:

$$\Pi_{\overline{x}} : T_{\overline{x}}\overline{\mathcal{M}} \to \mathcal{H}_{\overline{x}}\overline{\mathcal{M}} \ , \tag{3.16}$$

For $\mathbf{S}^{n,3}$, the horizontal projection is a map $\Pi_Y : T_Y\mathbb{R}_*^{n\times r} \to \mathcal{H}_Y\mathbb{R}_*^{n\times r}$, or equivalently $\Pi_Y : \mathbb{R}^{n\times r} \to \mathcal{H}_Y\mathbb{R}_*^{n\times r}$, given by:

$$\Pi_Y(Z) = Z - Y\Omega \ . \tag{3.17}$$

Equation (3.17) has the meaning that after removing the vertical space, the horizontal space is what is left. Since $Z - P\Omega \in \mathcal{H}_Y \mathbb{R}_*^{n\times r}$,

$$(Z - Y\Omega)^T Y = Y^T (Z - Y\Omega)$$

holds. Thus, $\Omega$ solves:

$$Y^T Y\Omega + \Omega Y^T Y = Y^T Z - Z^T Y \ . \tag{3.18}$$

This is a special instance of the *Sylvester equation*. Solving this form of the Sylvester equation is discussed in Section E.1.

### 3.6.4 The Riemannian Gradient

Let $\overline{f} : \overline{\mathcal{M}} \to \mathbb{R}$ be a function whose domain is the structure space $\overline{\mathcal{M}}$. The *differential* of $\overline{f}$ at a point $\overline{x} \in \overline{\mathcal{M}}$ in a direction $\eta \in T_{\overline{x}}\overline{\mathcal{M}}$ is denoted (see Section D.3.2):

$$D\overline{f}(\overline{x})[\eta]$$

The gradient is defined using this directional derivative, see Definition 3.6.2.

**Definition 3.6.2** (Gradient of a function)**.** *Let $grad\overline{f}(\overline{x})$ be the* gradient *of $\overline{f}$ at a point $\overline{x} \in \overline{\mathcal{M}}$. The gradient $grad\overline{f}(\overline{x})$ is* defined *as the unique tangent vector:*

$$D\overline{f}(\overline{x})[\eta] = \overline{g}(grad\overline{f}(\overline{x}), \eta)$$

*where $\overline{g}(\cdot,\cdot)$ is the Riemannian metric on $T_{\overline{x}}\overline{\mathcal{M}}$.*

Definition 3.6.2 is a generalization of the definition from $\mathcal{M} = \mathbb{R}^{3n}$. Let $\overrightarrow{p} \in \mathbb{R}^{3n}$, recall for a function $F : \mathbb{R}^{3n} \to \mathbb{R}$, its directional derivative in the direction of a vector $\overrightarrow{\eta} \in \mathbb{R}^{3n}$, denoted $DF(\overrightarrow{p})[\overrightarrow{\eta}]$, is defined as the dot product between the gradient and the vector $\overrightarrow{\eta}$:

$$DF(\overrightarrow{p})[\overrightarrow{\eta}] = \nabla F(\overrightarrow{p})^T \overrightarrow{\eta} \tag{3.19}$$

where $\nabla F(\overrightarrow{p})$ is the gradient of $F(\overrightarrow{p})$. The definition for a function on an arbitrary Riemannian manifold is similar, except the Riemannian metric on the tangent space at the point $\overrightarrow{p}$ is used instead of the dot product.

Consider again the function $\overline{f} : \overline{\mathcal{M}} \to \mathbb{R}$ on an arbitrary Riemannian manifold $\mathcal{M}$, suppose $\overline{f}$ is constant along each of the equivalence classes (the fibers), i.e. $\overline{f}(\overline{x}_1) = \overline{f}(\overline{x}_2)$ where $\pi(\overline{x}_1) = \pi(\overline{x}_2)$, then $\overline{f}$ is said to *induce* a function $f : \overline{\mathcal{M}}/\sim \to \mathbb{R}$ on the quotient

manifold. The gradient $\overline{\text{grad}f}(\overline{x})$ is a vector on the horizontal space at $\overline{x} \in \overline{\mathcal{M}}$. It is the horizontal lift of $\text{grad}f(x)$, which is the gradient of the induced function $f$. This relationship is mathematically expressed as:

$$\overline{\text{grad}f(x)} = \text{grad}\overline{f}(\overline{x}) \ ,$$

where $\overline{x} \in \overline{\mathcal{M}}$ is a point on the structure space, and $x \in \overline{\mathcal{M}}/\sim$ is a point on the quotient manifold, and $\overline{\text{grad}f(x)}$ is the horizontal lift of $\text{grad}f(x)$, the gradient of the induced function $f : \overline{\mathcal{M}}/\sim \to \mathbb{R}$.

Consider the objective function $f(Y) = f(YY^T)$ [2] in Equation (3.2), this function is constant along the equivalence classes $[Y]$ since $YQ(YQ)^T = YQQ^TY^T = YY^T$ for any $Q \in \mathbf{Q}_r$. Thus $f(YY^T)$ induces a function on $\mathbf{S}_+^{n,r}$.

The differential of $f(Y)$ in Equation (3.2) in a direction $\eta$ is given as follows:

$$\begin{aligned}
Df(Y)[\eta] =& D \parallel H \circ (\mathcal{K}(YY^T) - \overline{D}) \parallel_F^2 [\eta] \\
=& D\langle H \circ (\mathcal{K}(YY^T) - \overline{D}), H \circ (\mathcal{K}(YY^T) - \overline{D})\rangle[\eta] \\
=& 2\langle D(H \circ (\mathcal{K}(YY^T) - \overline{D}))[\eta], H \circ (\mathcal{K}(YY^T) - \overline{D})\rangle \\
=& 2\langle H \circ \mathcal{K}(Y\eta^T + \eta Y^T), H \circ (\mathcal{K}(YY^T) - \overline{D})\rangle \\
=& 2\langle \mathcal{K}(Y\eta^T + \eta Y^T), H \circ H \circ (\mathcal{K}(YY^T) - \overline{D})\rangle \\
=& 2\langle \mathcal{K}(Y\eta^T + \eta Y^T), H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D})\rangle \\
=& 2\langle Y\eta^T + \eta Y^T, \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))\rangle \\
=& 2\langle Y\eta^T, \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))\rangle \\
& + 2\langle \eta Y^T, \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))\rangle \\
=& 2\text{Trace}(\eta Y^T \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))) \\
& + 2\text{Trace}(Y\eta^T \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))) \\
=& 4\text{Trace}(\eta Y^T \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))) \\
=& 4\text{Trace}(Y^T \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))\eta) \\
=& 4\text{Trace}(\eta^T \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))Y)
\end{aligned} \tag{3.20}$$

where $H^{(2)} = H \circ H$, and $\mathcal{K}^*$ is the adjoint of $\mathcal{K}$. Note that $\mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))$ is a

[2] As noted before, $f(YY^T)$ may also be denoted $f(Y)$ to make the notation easier to read.

symmetric matrix:

$$\mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D})) = \mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))^T \ . \tag{3.21}$$

Definition 3.6.2 together with the Riemannian metric from Section 3.6.1 gives the expression:

$$Df(Y)[\eta] = \text{Trace}(\eta^T \text{grad} f(Y)) \ .$$

It follows $\text{grad} f(Y)$ given by:

$$\text{grad} f(Y) = 4\mathcal{K}^*(H^{(2)} \circ (\mathcal{K}(YY^T) - \overline{D}))Y \ . \tag{3.22}$$

Equation (3.2) for the objective function and Equation (3.22) for the gradient suggests the entired Gram matrix $X = YY^T$ needs to be formed. For computer implementation, the following equivalent summation expression have been presented in (Meyer, 2011[55]). These formulas do not require the Gram matrix $X = YY^T$ to be formed explicitly. The objective function is given by the following summation:

$$f(Y) = \sum_{(a,b)\in\mathcal{D}} \text{err}(a,b)^2 \ , \tag{3.23}$$

where

$$\begin{aligned} \text{err}(a,b) &= (e_a - e_b)^T YY^T (e_a - e_b) - \overline{q}_{ab} \\ &= (y_a - y_b)^T (y_a - y_b) - \overline{q}_{ab} \ . \end{aligned} \tag{3.24}$$

The quadrance $\overline{q}_{ab}$ is a specified entry from the incomplete EDM $\overline{D}$, the input to the EDMC problem. Recall $e_a$ was defined in Section 1.1, and $y_a - y_b$ is the difference between the $a$-th and $b$-th row of the matrix $Y$.

Suppose the specified entries of the incomplete EDM $\overline{D}$ involve the following $N$ neighbour index pairs, where $N = |\mathcal{D}|$:

$$\mathcal{D} = \{(a_1, b_1), \ldots, (a_N, b_N)\} \ ,$$

then, the gradient is given by the following summation:

$$\begin{aligned}\text{grad} f(Y) &= 4 \sum_{(a,b)\in\mathcal{D}} \text{err}(a,b)(e_a - e_b)(e_a - e_b)^T Y \\ &= 4 \sum_{(a,b)\in\mathcal{D}} \text{err}(a,b)(e_a - e_b)(y_a - y_b)^T \\ &= 4(E\Sigma E^T)Y \ .\end{aligned} \tag{3.25}$$

The matrix $E$ is an $n \times |\mathcal{D}|$ sparse matrix with columns given by $(e_a - e_b)$ for all $(a,b) \in \mathcal{D}$. That is:

$$E = \begin{pmatrix} e_{a_1} - e_{b_1} & \dots & e_{a_N} - e_{b_N} \end{pmatrix} \ .$$

The diagonal matrix $\Sigma$ has shape $|\mathcal{D}| \times |\mathcal{D}|$ and its diagonal entries are the errors:

$$\Sigma = \begin{pmatrix} \text{err}(a_1, b_1) & & \\ & \ddots & \\ & & \text{err}(a_N, b_N) \end{pmatrix} \ . \tag{3.26}$$

The $k$-th diagonal entry of $\Sigma$ uses the same $(a_k, b_k)$ pair as the $k$-th column of $E$. The summation form of the objective function, Equation (3.23) and gradient, Equation (3.25) have assumed the weight on the error term $H_{ab} = 1$ if $(a,b) \in \mathcal{D}$.

Note that both Equation (3.22) and Equation (3.25) for the gradient has the form:

$$\text{grad} f(Y) = GY$$

where $G$ is an $n \times n$ symmetric matrix. When the gradient has this form the vertical space does not need to be removed because the gradient is on the horizontal space, see Definition 3.6.1.

$$Y^T(\text{grad} f(Y)) = (\text{grad} f(Y))^T Y \ . \tag{3.27}$$

However, this thesis will still provide the projection formula which assumes the vertical space needs to be removed in order to give the most general expression.

### 3.6.5 The Riemannian Hessian

A *vector field* $\eta$ is a map that assigns to each point $x \in \mathcal{M}$ a point on the tangent space at $x$,

$$\eta : \mathcal{M} \to T_x\mathcal{M} \ .$$

Let $\mathcal{V}(\mathcal{M})$ denote the set of all smooth vector fields on $\mathcal{M}$, an *affine connection* is defined to be a smooth mapping ("smooth" is defined in Section D.2):

$$\nabla : \mathcal{V}(\mathcal{M}) \times \mathcal{V}(\mathcal{M}) \to \mathcal{V}(\mathcal{M}) : (\xi, \eta) \to \nabla_\xi \eta \ ,$$

where $\nabla_\xi \eta$ is called the *covariant derivative* of $\eta$ with respect to $\xi$ for the affine connection $\nabla$. An affine connection $\nabla$ generalizes the idea of taking a directional derivative of a vector field from $\mathbb{R}^{3n}$ to a Riemannian manifold $\mathcal{M}$. It is *another* structure in addition to the differentiable structure of a manifold, which is discussed in Section D.2. Every Riemannian manifold has a *preferred* affine connection, called the *Riemannian connection*, or *Levi-Civita connection*.

When the Riemannian manifold is Euclidean space, its Riemmanian connection reduces to the directional derivative (see for example [1] page 94, 98):

$$\nabla_\xi \eta(x) = D\eta(x)[\xi(x)] = \lim_{t \to 0} \frac{\eta(x + t\xi(x)) - \eta(x)}{t} \ . \tag{3.28}$$

This is the case for $\mathbb{R}^{3n}$, and also for $\mathbb{R}^{n \times r}$, the space of $n \times r$ matrices. This definition is called the *Euclidean connection*. When the Riemannian manifold is an embedded submanifold or quotient manifold of $\mathbb{R}^{n \times r}$, their Riemannian connection also reduce to the Euclidean connection, followed by a projection to the tangent space of the embedded or quotient manifold. The case for a quotient manifold is discussed below. The structure space of $\mathbf{S}_+^{n,r}$ is $\mathbb{R}_*^{n \times r}$, which is an embedded submanifold of $\mathbb{R}^{n \times r}$, therefore both $\mathbf{S}_+^{n,r}$ and $\mathbb{R}_*^{n \times r}$ use the Euclidean connection, the Levi-Civita connection is not required and will not be discussed further in this thesis, further information can be found in for example [1, 78, 48].

For a Riemannian quotient manifold $\mathcal{M} = \overline{\mathcal{M}}/\sim$, let $\overline{\nabla}$ denote the Riemannian connection for $\overline{\mathcal{M}}$. For a point $\overline{x} \in \overline{\mathcal{M}}$, let $\overline{\eta}, \overline{\xi} \in \mathcal{H}_{\overline{x}}\overline{\mathcal{M}} \subset T_{\overline{x}}\overline{\mathcal{M}}$ denote the horizontal tangent vectors. Let $x = \pi^{-1}(\overline{x}) \in \mathcal{M}$, and let $\eta, \xi \in T_x\mathcal{M}$. The horizontal lift of $\nabla_\xi \eta(x)$ for the quotient manifold $\mathcal{M} = \overline{\mathcal{M}}/\sim$ is given by $\overline{\nabla}_{\overline{\xi}}\overline{\eta}(\overline{x})$ from the structure space, projected onto the horizontal space using Equation (3.16) (see [1] page 100, [78] page 30):

$$\overline{\nabla_\xi \eta(x)} = \Pi_{\overline{x}}(\overline{\nabla}_{\overline{\xi}}\overline{\eta}(\overline{x})) \ . \tag{3.29}$$

If $\overline{\mathcal{M}}$ is a Euclidean space, Equation (3.29) is written using the directional derivative:

$$\overline{\nabla_\xi \eta(x)} = \Pi_{\overline{x}}(D\overline{\eta}(\overline{x})[\overline{\xi}]) \ . \tag{3.30}$$

For $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$ the horizontal projection in Equation (3.17) is used to define the Riemannian connection.

$$\nabla_\xi \eta(Y) = \Pi_Y(D\eta(Y)[\xi]) \ , \tag{3.31}$$

where $\eta$, $\xi \in \mathcal{H}_Y\mathbb{R}_*^{n\times r}$. Recall from Section 3.6 that for $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n\times r}/\mathbf{O}_r$, a bar will not be used to distinguish between objects in the structure space and quotient space, this is to make the notation easier to read.

For an abstract quotient manifold, $\mathcal{M} = \overline{\mathcal{M}}/$, the Hessian is a *linear operator* defined by the map:

$$\mathrm{Hess} f(x) : T_x\mathcal{M} \to T_x\mathcal{M} \text{ where } x \in \mathcal{M} \ .$$

This map is given by the covariant derivative of the gradient with respect to a given tangent vector $\eta \in T_x\mathcal{M}$ (see Definition 5.5.1[1] and Definition 2.36 of [78]):

$$\mathrm{Hess} f(x)[\eta] = \nabla_\eta \mathrm{grad} f(x) \ . \tag{3.32}$$

Recall in $\mathbb{R}^{3n}$, the Hessian matrix is arrived at from differentiating the gradient.

The Riemannian Hessian of the objective function in Equation (3.2) in a tangent direction $\eta$ is given in [58]. The Hessian is defined using the Riemannian connection from Equation (3.31):

$$\mathrm{Hess} f(Y)[\eta] = \nabla_\eta \mathrm{grad} f(Y) = \Pi_Y(D\mathrm{grad} f(Y)[\eta]) \ . \tag{3.33}$$

Applying this definition to Equation (3.22) gives:

$$\mathrm{Hess} f(Y)[\eta] = 4\Pi_Y[\mathcal{K}^*(H \circ (\mathcal{K}(YY^T) - \overline{D}))\eta + \mathcal{K}^*(H \circ (\mathcal{K}(Y\eta^T + \eta Y^T) - \overline{D}))Y] \ . \tag{3.34}$$

Alternatively, differentiating Equation (3.25) gives (see also [55]):

$$\mathrm{Hess} f(Y)[\eta] = 4\Pi_Y(E\Sigma E^T\eta + E\widetilde{\Sigma}E^TY) \ , \tag{3.35}$$

where $\widetilde{\Sigma}$ is a diagonal matrix whose diagonal entries are given by:

$$\mathrm{diag}(\widetilde{\Sigma}) = 2\mathrm{diag}((E^TY)(\eta^TE)) \ . \tag{3.36}$$

In summary, the Riemannian Hessian for an arbitrary Riemannian manifold $\mathcal{M}$ is a linear operator. Only in the special case where $\mathcal{M} = \mathbb{R}^{3n}$, or $\mathcal{M} = \mathbb{R}^n$, does it appear as a matrix.

### 3.6.6 Retraction

In the context of matrix manifold optimization, a *retraction* is the notion of mapping from the tangent space back to the matrix manifold [1, 2]. Figure 3.2 shows an retraction on an abstract manifold.

Retractions are approximations to the exponential map, see Appendix D Section D.5.1 for a definition of the exponential map, see also [1] Section 4.1, Chapter 5 Section 5.4 and [48] Chapter 5. Computing the exponential involves solving nonlinear ordinary differential equations which is a challenging problem in itself.

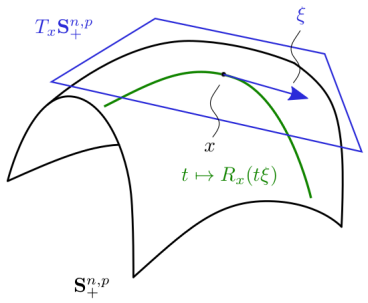



Figure 1.2: Retraction-based optimization.

Figure 3.2: A retraction on an abstract manifold. Figure 1.2 of (Vandereycken, 2010[78]).

The retraction for the EDMC problem maps from $T_Y\mathbf{S}_+^{n,r}$ to $\mathbf{S}_+^{n,r}$, and is given by [58]:

$$R_Y(\eta) = Y + \eta \,. \tag{3.37}$$

Note that when $Y$ is full rank, and when the entries in $\eta$ are small perturbations, $Y + \eta$ is still a full rank matrix.

## 3.7 Summary

This Chapter used the EDMC problem to introduce unconstrained optimization on the $\mathbf{S}_+^{n.r}$ matrix manifold. The matrix manifold optimization paradigm starts with abstract differential geometry concepts, and applies these concepts to a wide range of matrix manifolds.

# Chapter 4

# Normal Mode Analysis and the PSD Potential

## 4.1 Introduction

The Hookean potential was first introduced as an efficient potential for finding the normal modes of a protein structure in [75], where the author showed it produces root mean squared (RMS) fluctuations in agreement with the semi-empirical potential. The Hookean potential's benefit for NMA studies is it allows the costly energy minimization step to be skipped.

This chapter will also introduce the PSD potential using NMA as the application.

This Chapter does not claim any additional benefit is gained by using the PSD potential for NMA. Rather NMA is the setting used to introduce the PSD potential.

The purpose of this Chapter is to highlight the agreement between the Hookean potential and the PSD potential using the density of modes graph and the RMS fluctuation graphs which were also seen in for example Tirion [75] and Kim's [37] publications.

Section 4.2 describes the graphs for the density of modes, the RMS fluctuation per $\alpha$-carbon due to all modes, and the RMS fluctuation of all $\alpha$-carbons per mode.

Section 4.3 presents the PSD potential energy, by *removing the square-root* from the Hookean potential [1]. The density of modes graph and the RMS fluctuation graphs are presented and compared with the Hookean potential's graphs.

[1] In order to compute distance, the dot product is taken first, then the square-root is applied

Section 4.4 discusses why quadrance has special significance to the PSD potential.

This Chapter continues to use the assumption discussed in Section 2.1 that a protein structure is represented as an $\alpha$-carbon chain.

Some of the discussions in this Chapter has appeared in our previous publication [53].

## 4.2 Density of Modes and Root Mean Square (RMS) Fluctuations

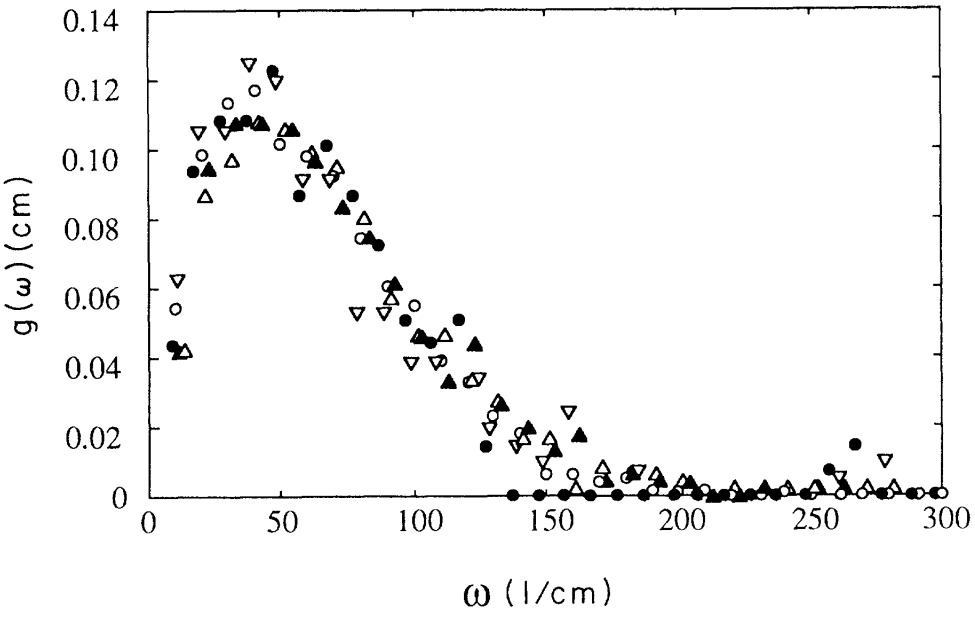
### 4.2.1 Density of modes

The density of normal modes curve for various proteins was compared by ben-Avraham in [10]. The result was presented in Figure 1 of [10] and is reproduced below in Figure 4.1. The author observed the shape among different proteins was very similar, suggesting it is a univeral property shared by proteins, and referred to this curve as a "univeral curve".

This curve is constructed by grouping the eigenvalues of the Hessian matrix of the potential energy into bins and counting the number of eigenvalues in each bin. The vertical axis of this curve, $g(\omega)$ is the number of eigenvalues within a range (a bin), divided by the total number of eigenvalues. The horizontal axis of this curve, $\omega$, represents the range of eigenvalues. A similar curve is also presented in [37] Figure 2.11, in a histogram form. The cumulative density version of this universal curve, denoted $G(\omega)$, was given in Figure 1 of Tirion's publication[75] rather than the density curve, $g(\omega)$.

The different points in the curve in Figure 4.1 represent different proteins:

- white circle: g-actin,
- white up-pointing triangle:lysozyme,
- black up-pointing triangle: ribonuclease I,
- white down-pointing triangle: BPTI,
- black circle: crambin.




(a) Density of modes for various proteins.

**FIG. 1. Density of vibrational normal modes, $g(\omega)$, of $g$ actin (○) lysozyme (△), ribonuclease I (▲), BPTI (▽), and crambin (●) as a function of frequency.**

(b) Original caption explaining the meaning of each shape.

Figure 4.1: The shape of the density of normal modes is similar for many proteins. Taken from [10].

### 4.2.2 RMS Fluctuations

RMS fluctuations of atoms are statistical measures of atomic displacements from the equilibrium atomic coordinates and indicates which atoms have more flexibility to move and which are stationary acting as pivot points. They are calculated using the eigenvalues and eigenvectors of the Hessian matrix of the potential energy (e.g. Hookean potential or PSD potential). Two fluctuations can be calculated:

- the fluctuation of the *i-th α-carbon* due to all modes, denoted $\sigma^i$, and
- the fluctuation of all $\alpha$-carbons for *the k-th mode* , denoted $\sigma_k$.

The formulas for these values have been described previously, for example [37, 76], and are summarized next.

The formula for $\sigma^i$ is defined as follows, the first 6 eigenvalues are zero representing rigid motions, and are ignored.

$$\sigma^i = \left( \sum_{k=7}^{3n} (\sigma_k^i)^2 \right)^{\frac{1}{2}} . \tag{4.1}$$

The formula for $\sigma_k$, the RMS fluctuation of all $\alpha$-carbons per normal mode $k$, is given by:

$$\sigma_k = \left( \sum_{i=1}^{n} \frac{(\sigma_k^i)^2}{n} \right)^{\frac{1}{2}} . \tag{4.2}$$

In the above formulas, $\sigma_k^i$ is given by:

$$\sigma_k^i = \left|\left| v_k^i \frac{\gamma_k}{\sqrt{2}} \right|\right| , \tag{4.3}$$

and $v_k = ((v_k^1)^T, \ldots, (v_k^n)^T)^T \in \mathbb{R}^{3n}$ is the eigenvector for mode $k$. The value of $\gamma_k$ is given by[37, 76]:

$$\gamma_k = \left( \frac{2k_BT}{\lambda_k} \right)^{\frac{1}{2}} , \tag{4.4}$$

where $\lambda_k$ is the $k$-th eigenvalue, $k_B$ is the Boltzmann constant, and $T$ is temperature. Since $2k_BT$ is a constant, setting this constant to 1 to use the $\gamma_k$ value of:

$$\gamma_k = \frac{1}{\sqrt{\lambda_k}} . \tag{4.5}$$

will not change the shape of the graphs. Note that $\gamma_k$ is a measure of the average *amplitude* of vibration of the $\alpha$-carbon atoms, therefore $\sqrt{\lambda_k}$ is a measure of the average *frequency* of vibration.

## 4.3 The Positive Semidefinite Potential Energy

Removing the square-root in Equation (2.14) gives the following pairwise PSD potential energy:

$$\begin{aligned} E_{ab,P-NMA}(p_a(t) - p_b(t)) &= \frac{1}{2}\left(\|p_a(t) - p_b(t)\|^2 - d_{ab}(0)^2\right)^2 \\ &= \frac{1}{2}\left(\|p_a(t) - p_b(t)\|^2 - q_{ab}(0)\right)^2 \end{aligned} \tag{4.6}$$

Quadrance can be expressed as a *linear* function of the Gram matrix $P(t)P(t)^T$:

$$\|p_a(t) - p_b(t)\|^2 = (e_a - e_b)^T P(t)P(t)^T(e_a - e_b)$$

where $P(t) = P(0) + \delta(t)$, and $\delta(t)$ is an $n \times 3$ matrix of perturbations, $\delta_a(t)$, $a = 1, \ldots, n$ now appear as the rows of $\delta(t)$ (compare this to Equation (2.13)).

$$\delta(t) = \begin{pmatrix} \delta_1(t) & \ldots & \delta_n(t) \end{pmatrix}^T \in \mathbb{R}^{n\times 3}. \tag{4.7}$$

The total potential energy is now a function of the time $t$ Gram matrix $P(t)P(t)^T$ given by:

$$\begin{aligned} E_{P-NMA}(P(t)) &= \sum_{(a,b)\in\mathcal{D}} E_{ab,P-NMA}(p_a(t) - p_b(t)) \\ &= \frac{1}{2}\sum_{(a,b)\in\mathcal{D}} \left((e_a - e_b)^T P(t)P(t)^T(e_a - e_b) - q_{ab}(0)\right)^2 . \end{aligned} \tag{4.8}$$

Since this potential energy is a function of the PSD Gram matrix, it will be referred to as the "PSD potential energy".

As discussed in Section 3.6.5, the Hessian is a linear operator for an arbitrary Riemannian manifold $\mathcal{M}$. NMA requires the matrix representation of this linear operator to be found, which is an $3n \times 3n$ matrix. The process is very similar to that for the Hookean potential in Section 2.3. The second order Taylor expansion for the pairwise PSD potential energy near $P(0)$, for a small perturbation $\delta(t)$ is found first, the $3 \times 3$ Hessian matrix for the pairwise potential will give the blocks for the $3n \times 3n$ Hessian matrix.

A simple way to find the second order expansion for the pairwise potential using the matrix representation of the Hessian to make the substitution $x = P(0)^T(e_a - e_b) \in \mathbb{R}^3$, and apply the formulas in Section B.5.

Firstly note that

$$E_{ab,P-NMA}(x) = (x^T x - q_{ab}(0))^2 = 0 \ ,$$

therefore, the constant term in the second order expansion is zero. Next, the gradient is given by:

$$\nabla E_{ab,P-NMA}(x) = 2(x^T x - q_{ab}(0))x = \mathbf{0}_3 \ .$$

Therefore, only the Hessian matrix is nonzero when evaluated at $P(0)$:

$$\begin{aligned} \nabla^2 E_{ab,P-NMA}(x) &= 2(x^T x - q_{ab}(0))I_3 + 4xx^T \\ &= 4xx^T \\ &= 4\left[p_a(0) - p_b(0)\right]\left[p_a(0) - p_b(0)\right]^T \ . \end{aligned} \tag{4.9}$$

A comparison with Equation (2.16) shows the pairwise Hessian matrix for the Hookean and PSD potential differ by a division of

$$4q_{ab}(0) = 4\left[p_a(0) - p_b(0)\right]^T\left[p_a(0) - p_b(0)\right] \ ,$$

which is a constant depending on the atomic index pair $(a, b)$. Therefore, the two Hessian matrices can be made equal by multiplying the pairwise interaction weights by the required scaling factor.

The corresponding second order expansion for the total PSD potential is given by:

$$\begin{aligned} E_{P-NMA}(P(t)) &\approx \frac{1}{2}\sum_{(a,b)\in\mathcal{D}} (\delta_a(t) - \delta_b(t))^T \nabla^2 E_{ab,P-NMA}(p_a(0) - p_b(0))(\delta_a(t) - \delta_b(t)) \\ &= 2\sum_{(a,b)\in\mathcal{D}} (\delta_a(t) - \delta_b(t))^T \left[p_a(0) - p_b(0)\right]\left[p_a(0) - p_b(0)\right]^T (\delta_a(t) - \delta_b(t)) \ , \end{aligned} \tag{4.10}$$

Equation (4.10) can be expressed in matrix form:

$$E_{P-NMA}(P(t)) \approx \frac{1}{2}\delta(t)^T \nabla^2 E_{P-NMA}(P(t))\delta(t) \ . \tag{4.11}$$

where $\nabla^2 E_{P-NMA}(P(t))$ is the $3n \times 3n$ Hessian matrix, which has the same Laplacian structure as the Hessian for the Hookean potential.

The density of modes graph and the RMS fluctuations graph will now be compared between the Hookean and the PSD potential for a number of example proteins. For 1ATN, all graphs are generated with a $10\mathring{A}$ cut-off distance while for the other proteins a $11\mathring{A}$ cut-off distance is used. The cut-off distance is a parameter chosen arbitrarily, it should not be too small to the extend that some atom's set of neighbours is disconnected from another. But a small number is chosen to keep the computational burden small.

In Figure 4.2 and 4.3, the density of modes curves is shown for various small proteins in histogram form. The Hookean potential energy and the PSD potential energy both produce the similar shape which was observed in [10] and presented in Figure 4.1. Recall only the $\alpha$-carbons of the protein are modelled. These histograms are generated using `matplotlib.pyplot`'s `hist` function [33]:

```
hist(eigval[6:], rwidth = 0.4, bins=40)
```

The vector `eigval` contains the eigenvalues from the Hessian matrix of the potential energy. The first six eigenvalues are zero, and are therefore ignored.

Figures 2 and 3 of [75] showed the root-mean-square (RMS) fluctuation graphs of the Hookean potential closely matched the semi-empirical L79 potential. Similar graphs are also seen in [37]. The Hookean potential and the PSD potential are also in agreement.

Figure 4.4 and 4.5, presents the $\sigma^i$ graphs for the same proteins in Figure 4.2 and Figure 4.3. It shows that the shape of the $\sigma^i$ graph for the original Hookean potential energy and the PSD potential energy are in agreement.

Figure 4.6 and 4.7, present the $\sigma_k$ graphs for the same corresponding proteins. These graphs decrease very fast indicating large amplitude backbone motions are due to the low modes. Once again, an agreement between the Hookean potential energy and the PSD potential energy is seen.

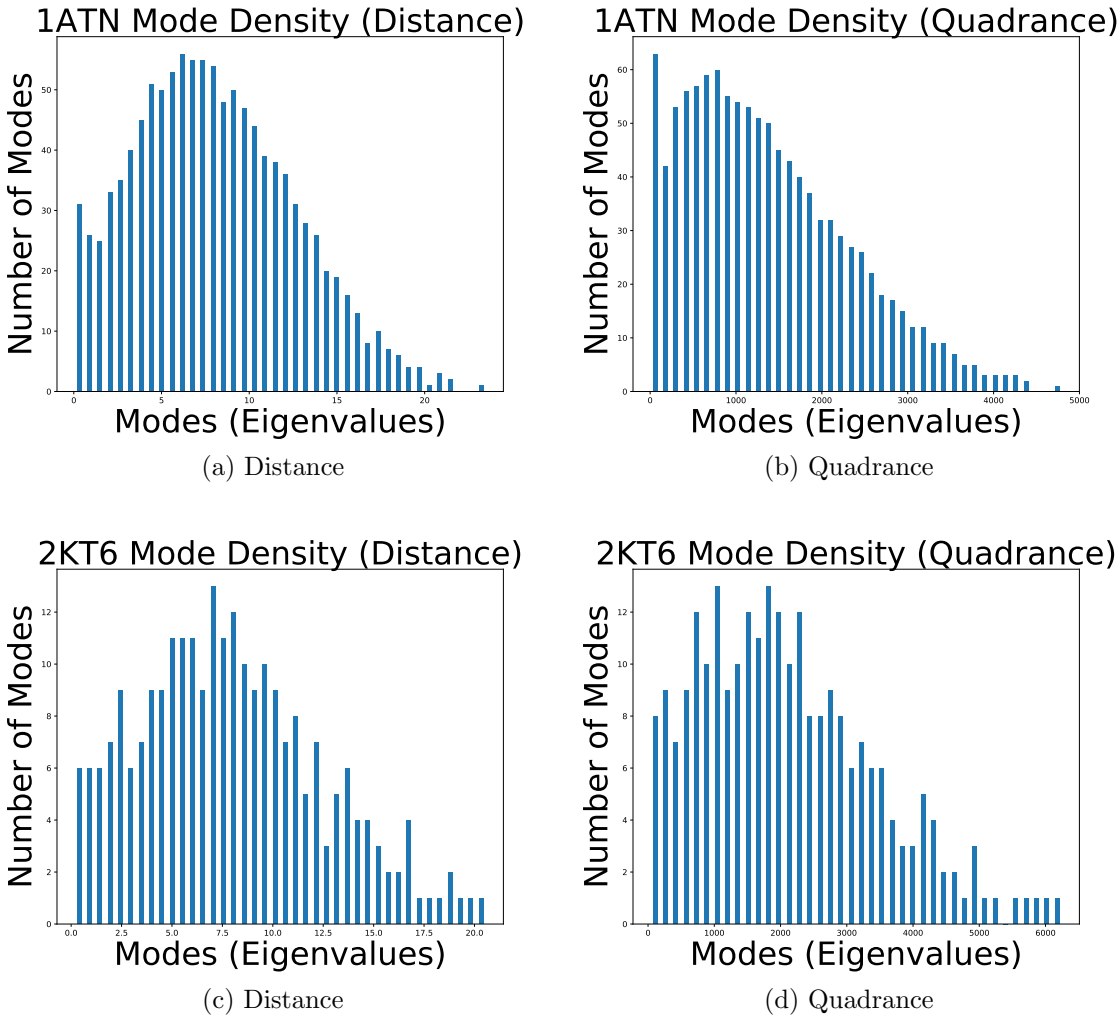
In all the figures, note that the vertical axis between the Hookean (distance) and PSD (quadrance) graphs are different, this is because the Hookean potential and PSD potential have different Hessian matrices.

## 4.4 The PSD Potential Prefers Quadrance

Although the Hookean and PSD potentials provide the same information about the atom's RMS fluctuation, these two potentials are very different mathematical models. This section discusses why quadrance has special significance to the PSD potential.

Let $\mathcal{K}_{ab}(\cdot)$ denote the function that maps a rank 3 Gram matrix $X \in \mathbf{S}_+^{n,3}$ to the quadrance between the $a$-th and $b$-th atoms.

$$\mathcal{K}_{ab}(X) = (e_a - e_b)^T X (e_a - e_b) = (p_a - p_b)^T (p_a - p_b) = \| p_a - p_b \|^2 \ . \tag{4.12}$$




(a) Distance

(b) Quadrance

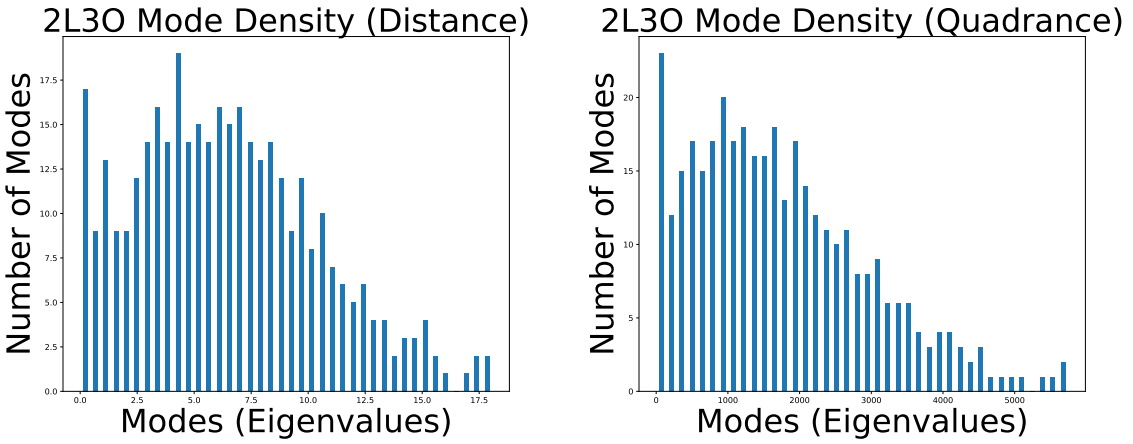
(c) Distance

(d) Quadrance

Figure 4.2: Density of normal modes for distance and quadrance.

Consider any two different $n \times n$ Gram matrices $X_1 = P_1 P_1^T$ and $X_2 = P_2 P_2^T$. Applying $\mathcal{K}_{ab}(X)$ to $X = X_1 + X_2$ gives:

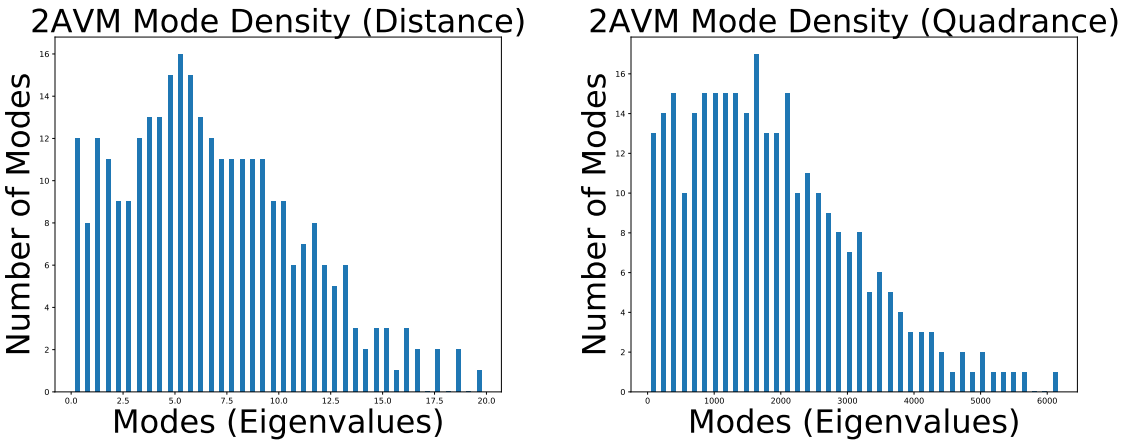
$$\begin{aligned} \mathcal{K}_{ab}(X_1 + X_2) &= (e_a - e_b)^T (X_1 + X_2)(e_a - e_b) \\ &= (e_a - e_b)^T X_1 (e_a - e_b) + (e_a - e_b)^T X_2 (e_a - e_b) \\ &= \mathcal{K}_{ab}(X_1) + \mathcal{K}_{ab}(X_2) \ . \end{aligned} \tag{4.13}$$




(a) Distance




(b) Quadrance

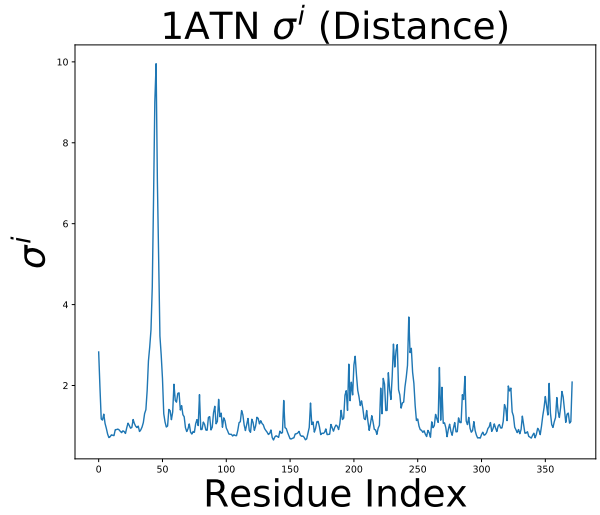



(c) Distance

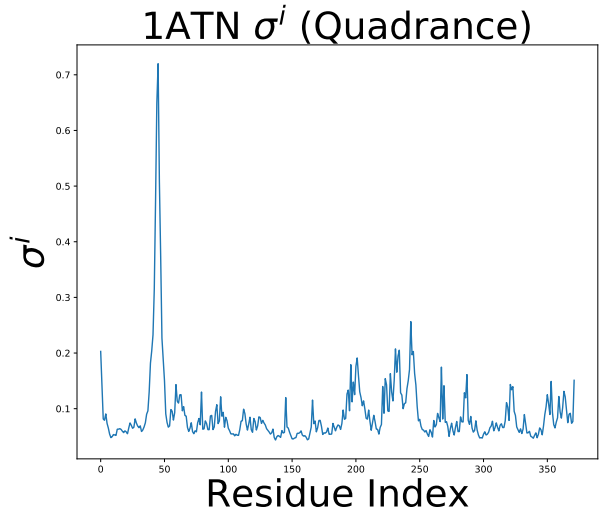



(d) Quadrance

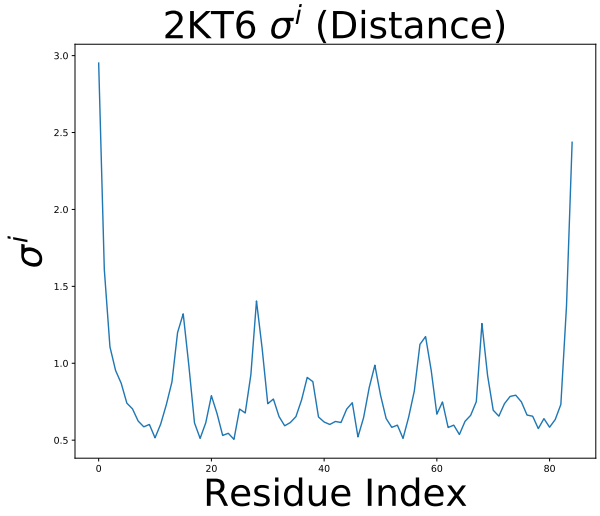
Figure 4.3: Density of normal modes for distance and quadrance (continued).



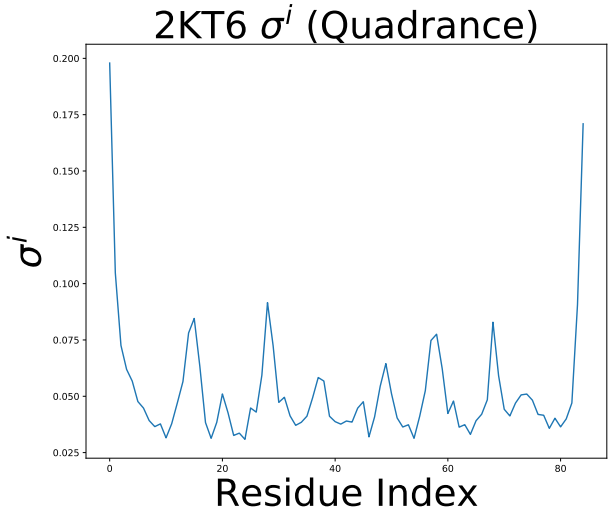

(a) Distance




(b) Quadrance




(c) Distance

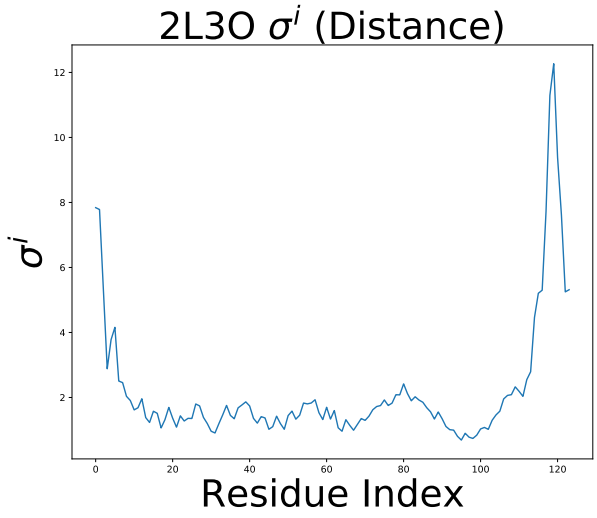



(d) Quadrance

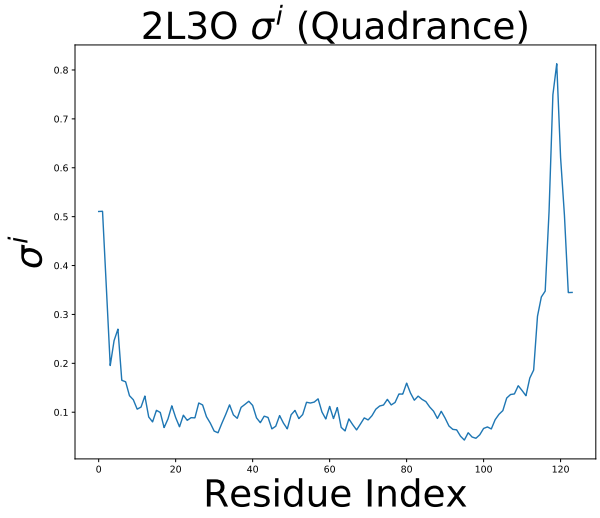
Figure 4.4: $\sigma^i$ graphs for various proteins.

These calculations show that $\mathcal{K}_{ab}$ is a linear function on the domain $\mathbf{S}_+^{n,3}$:

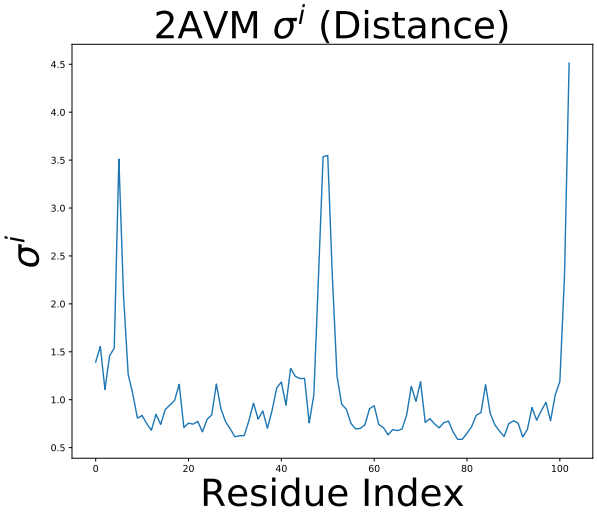
$$\mathcal{K}_{ab}(X_1 + X_2) = \mathcal{K}_{ab}(X_1) + \mathcal{K}_{ab}(X_2) \ . \tag{4.14}$$



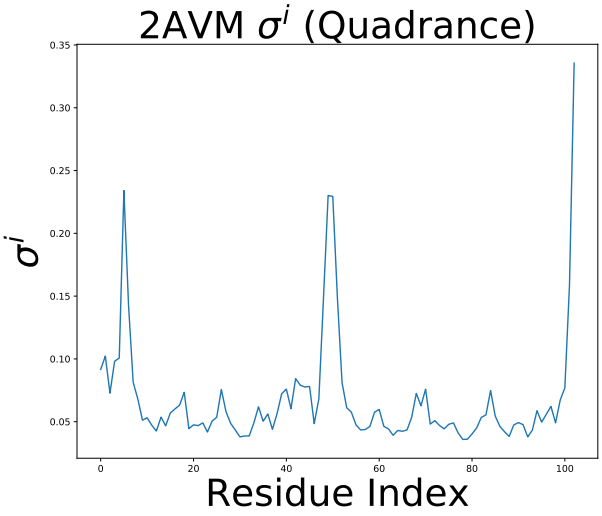

(a) Distance




(b) Quadrance

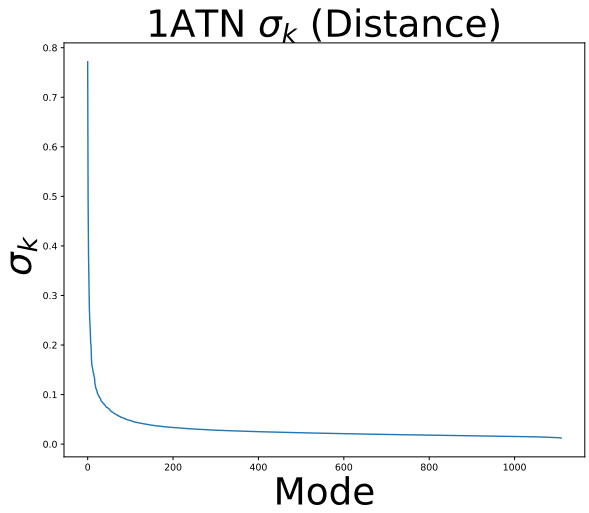



(c) Distance

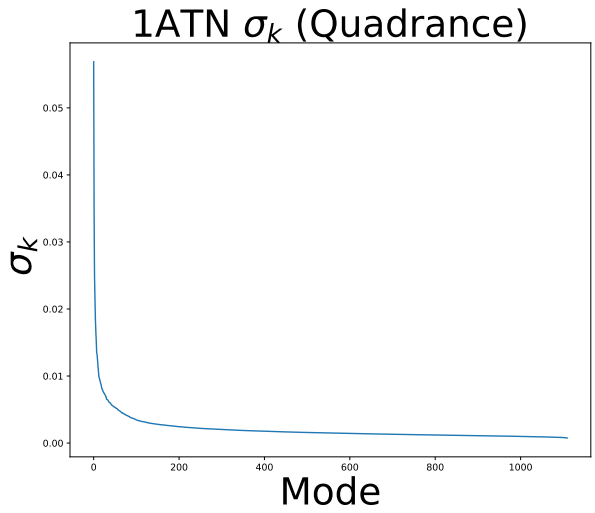



(d) Quadrance

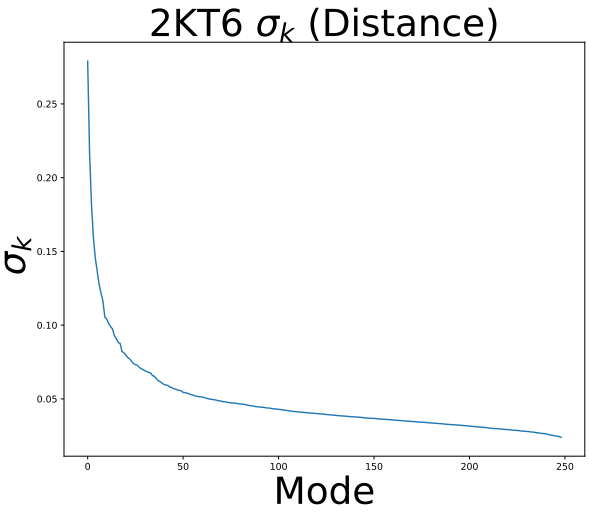
Figure 4.5: $\sigma^i$ graphs for various proteins (continued).



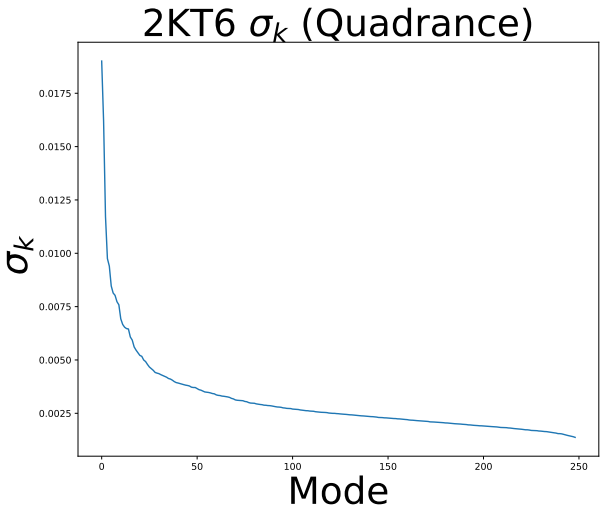

(a) Distance




(b) Quadrance



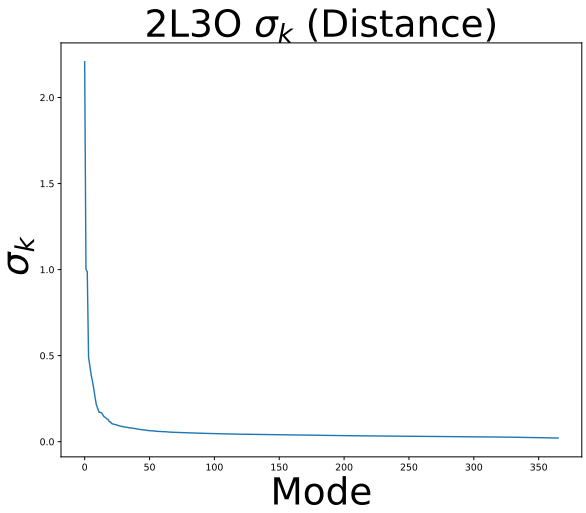

(c) Distance



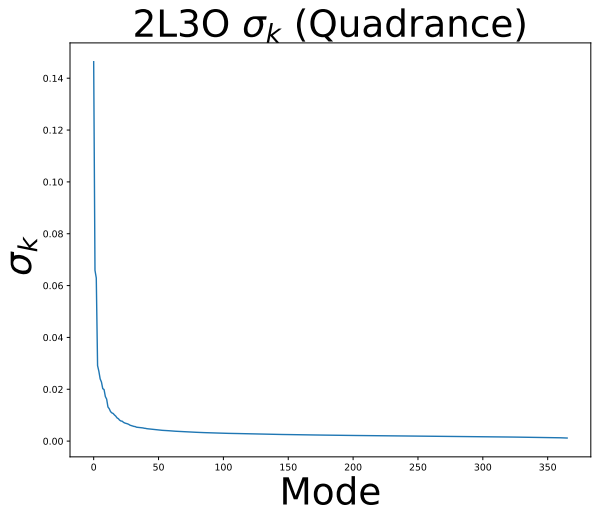

(d) Quadrance

Figure 4.6: $\sigma_k$ graphs for various proteins.

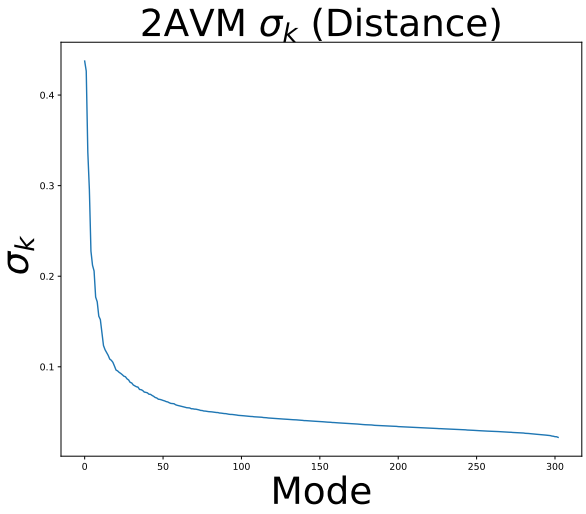
Let $\sqrt{\mathcal{K}}_{ab}(\cdot)$ denote the function that maps a rank 3 Gram matrix to the *distance* between



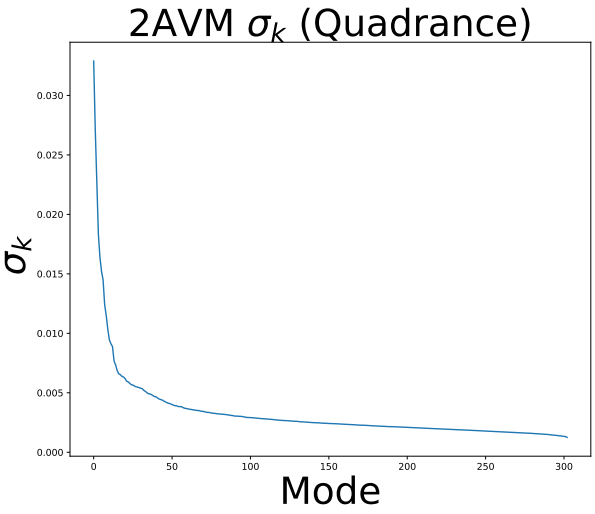

(a) Distance




(b) Quadrance




(c) Distance




(d) Quadrance

Figure 4.7: $\sigma_k$ graphs for various proteins (continued).

the $a$-th and $b$-th atoms:

$$\sqrt{\mathcal{K}}_{ab}(X) = \sqrt{(e_a - e_b)^T X (e_a - e_b)} = \sqrt{(p_a - p_b)^T (p_a - p_b)} = \| p_a - p_b \| \ . \qquad (4.15)$$

$\sqrt{\mathcal{K}_{ab}}(X)$ is *not* a linear function on $\mathbf{S}_+^{n,3}$. Therefore, choosing quadrance leads to a simpler function on $\mathbf{S}_+^{n,3}$.

The mapping $\mathcal{K}_{ab}(X)$ will produce the same quadrance if $P$ was translated. But if the points in $P$ have centroid at the origin, that is $P^T\mathbf{1}_n = \mathbf{0}_n$, then $X$ is a centered Gram matrix, see Equation (C.25), and in this case $K_{ab}$ will be a bijective map. The discussion in Section C.5 uses the mapping $\mathcal{K}$, this map is a linear isomorphism between $\mathbf{S}_+^n \cap \mathbf{S}_C^n$ and $\mathcal{E}^n$, see Krislock [44] Theorem 2.35 and Theorem 2.36.

Now consider the situation on $\mathbb{R}^{3n}$. Define a vector function $k_{ab}(\cdot)$ mapping a vector $\overrightarrow{p} \in \mathbb{R}^{3n}$ to the quadrance between the $a$-th and $b$-th atoms.

$$\begin{aligned} k_{ab}(\overrightarrow{p}) &= \left((e_{a,I_3} - e_{b,I_3})^T\overrightarrow{p}\right)^T (e_{a,I_3} - e_{b,I_3})^T\overrightarrow{p} \\ &= \overrightarrow{p}^T(e_{a,I_3} - e_{b,I_3})(e_{a,I_3} - e_{b,I_3})^T\overrightarrow{p} \\ &= \overrightarrow{p}^T S_{ab}\,\overrightarrow{p} \\ &= (p_a - p_b)^T(p_a - p_b) = \| p_a - p_b \|^2 \end{aligned} \tag{4.16}$$

Clearly, $k_{ab}(\overrightarrow{p})$ is not a linear function on $\mathbb{R}^{3n}$. Mapping from $\overrightarrow{p}$ to:

- the distance between the $a$-th and $b$-th atoms, or
- any power of distance larger than 2,

also do not use a linear function of $\overrightarrow{p} \in \mathbb{R}^{3n}$.

In summary, $\mathbb{R}^{3n}$ does not show preference for distance or quadrance, or any higher power of distance. But $\mathbf{S}_+^{n,3}$ does show a preference for using quadrance because a Gram matrix can be mapped linearly to quadrance, giving the simplest function when compared to distance or any higher power of distance.

## 4.5 Summary

Removing the square-root in the Hookean potential energy results in a PSD potential energy. The shape of the RMS fluctuations given by the PSD potential energy is observed to be in agreement with that given by the Hookean potential energy, suggesting both potentials output the same information about the $\alpha$-carbon atom's fluctuations. Despite the output being equal, the two potential energies are very different mathematical models. The PSD potential energy is a function on $\mathbf{S}_+^{n,3}$ where quadrance is linearly related to

the Gram matrix $PP^T$, choosing quadrance gives a simpler model than other powers of distance, and hence quadrance is the preferred quantity. However, on $\mathbb{R}^{3n}$, neither quadrance nor distance has a linear relation with $\overrightarrow{p}$.

# Chapter 5

# The Equations of Motion for the Riemannian Manifold $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n\times 3}/\mathbf{O}_3$

## 5.1 Introduction

An abstract potential energy is a smooth function on a Riemannian manifold, see Definition D.7.1. The equation of motion for classical mechanics is also formulated on a Riemannian manifold. Following the introduction of the PSD potential on $\mathbf{S}_+^{n,3}$ in Chapter 4, this Chapter presents the equations of motion on $\mathbf{S}_+^{n,3}$.

Let $f(P)$ be a function defined on the rank 3 PSD matrix manifold, $f : \mathbf{S}_+^{n,3} \to \mathbb{R}$, $P$ is an arbitrary $n \times 3$ atomic coordinates matrix.

**Definition 5.1.1** (The PSD ENM dynamics (PSD-ENM-D) problem)**.** *The PSD ENM dynamics (PSD-ENM-D) problem is an optimization problem that uses a PSD potential to model a protein's dynamics, with the assumption that the protein is represented as an ENM. The PSD-ENM-D problem seeks to find the $n \times 3$ matrix $P$ of atomic coordinates of a protein structure by solving a rank 3 PSD matrix manifold optimization problem with constraints. The generic form of the problem is given as follows:*

$$\begin{aligned} min \quad & f(P) \\ s.t. \quad & Trace(P^T A_i P) = q_{a_i b_i} \quad i = 1, \ldots, m \ . \end{aligned} \tag{5.1}$$

*In the above problem, $q_{a_i b_i}$ is the quadrance between the $a_i$-th atom and $b_i$-th atom, and $A_i$ is the $n \times n$ symmetric matrix defined in Equation (1.4).*

ENI and iENM were introduced in Chapter 2, and can be reformulated as PSD-ENM-D problems as shown in Chapter 7. The matrix $A_i$ in Equation (5.1) was introduced in Equation (1.4), note it serves the same purpose on $\mathbf{S}_+^{n,3}$ as the stamp matrix $S_{a_i b_i}$ on $\mathbb{R}^{3n}$.

Modelling ENMs on the rank 3 PSD matrix manifold removes all of the defects of the current model in $\mathbb{R}^{3n}$ as given in Table 1.2.

## 5.2 Revisiting ENMs in $\mathbb{R}^{3n}$

### 5.2.1 Revisiting the Rank Deficient Hessian of ENI and iENM

As shown by Equation (2.43) and Equation (2.45), the Hessian matrix for both the ENI Hookean potential and the iENM Hookean potential required an addition with the identity matrix, $I_{3n}$, in order to be invertible.

Consider the time interval $[t_j, t_{j+1}]$, let $\Delta t = t_{j+1} - t_j$. Let:

- $\overrightarrow{p}_j = \overrightarrow{p}(t_j)$ denote the matrix of atomic coordinates at the beginning of the time interval,
- $\overrightarrow{v}_j$ be the velocity at the beginning of the time interval, and
- $\overrightarrow{v}_{j+1}$ be the velocity at the end of the interval.

Let $\overrightarrow{p}_{j+1} = \overrightarrow{p}(t_{j+1})$ be the atomic coordinates at the end of the interval, this is the unknown to be determined. Consider the following equation of motion where the force due to the potential is evaluated *implicit*.

$$\frac{\overrightarrow{v}_{j+1} - \overrightarrow{v}_j}{\Delta t} = -\nabla E(\overrightarrow{p}_{j+1}) \tag{5.2}$$

$$\frac{\overrightarrow{p}_{j+1} - \overrightarrow{p}_j}{\Delta t} = \overrightarrow{v}_{j+1} \tag{5.3}$$

Substituting Equation (5.2) into Equation (5.3)

$$\frac{\overrightarrow{p}_{j+1} - \overrightarrow{p}_j}{\Delta t} = \overrightarrow{v}_j - (\Delta t)\nabla E(\overrightarrow{p}_{j+1}) \tag{5.4}$$

Rearrange Equation (5.4) by moving all terms to the left hand side gives:

$$\frac{\overrightarrow{p}_{j+1} - \overrightarrow{p}_j - (\Delta t)\overrightarrow{v}_j}{(\Delta t)^2} + \nabla E(\overrightarrow{p}_{j+1}) = \mathbf{0}_{3n} \tag{5.5}$$

Let:

$$\begin{aligned} f(\overrightarrow{p}) =& \frac{1}{2(\Delta t)^2}\langle \overrightarrow{p} - \overrightarrow{p}_j - (\Delta t)\overrightarrow{v}_j, \overrightarrow{p} - \overrightarrow{p}_j - (\Delta t)\overrightarrow{v}_j\rangle + E(\overrightarrow{p}) \\ =& \frac{1}{2(\Delta t)^2} \parallel \overrightarrow{p} - \overrightarrow{p}_j - (\Delta t)\overrightarrow{v}_j \parallel^2 + E(\overrightarrow{p}) \ . \end{aligned} \tag{5.6}$$

Equation (5.5) shows $\overrightarrow{p}_{j+1}$ satisfies:

$$\nabla f(\overrightarrow{p}_{j+1}) = \mathbf{0}_{3n} \ .$$

Thus, $\overrightarrow{p}_{j+1}$ the the critical point of the following optimization problem:

$$\min f(\overrightarrow{p}) \ . \tag{5.7}$$

The Hessian matrix of $f(\overrightarrow{p})$ is given by:

$$\nabla^2 f(\overrightarrow{p}) = \frac{1}{2(\Delta t)^2} I_{3n} + \nabla^2 E(\overrightarrow{p}) \ . \tag{5.8}$$

The resulting Hessian matrix is very similar to Equation (2.43) and Equation (2.45) in that an identity matrix, $I_{3n}$, is added to a rank deficient Hessian. *While adding $I_{3n}$ seemed very arbitrary for the ENI and iENM Hookean potential*, the discussion above shows the *same* conclusion can be arrived at by integrating the equations of motion in classical mechanics, with the force due to the potential treated *implicitly*.

### 5.2.2 The Constraint Force Has Two Purposes

Consider a vector in $\overrightarrow{\eta} \in \mathbb{R}^{3n}$, Rosen's projection, given in Equation (2.52) projects this vector onto the tangent space of the constraint manifold:

$$\text{Proj}(\overrightarrow{\eta}) = \overrightarrow{\eta} - \nabla C(\overrightarrow{p})^T (\nabla C(\overrightarrow{p}) \nabla C(\overrightarrow{p})^T)^{-1} \nabla C(\overrightarrow{p}) \overrightarrow{\eta} \ .$$

This expression can be written using Lagrange multipliers:

$$\text{Proj}(\overrightarrow{\eta}) = \overrightarrow{\eta} - \nabla C(\overrightarrow{p})^T \lambda \ .$$

where $\lambda \in \mathbb{R}^m$ with $m$ the number of constraints. $\lambda$ is given by the solution of the linear system:

$$\lambda = (\nabla C(\overrightarrow{p})\nabla C(\overrightarrow{p})^T)^{-1}\nabla C(\overrightarrow{p})\,\overrightarrow{\eta}\ . \tag{5.9}$$

Recall from Equation (2.55) and Equation (2.57) also serves the purpose of ensuring the atomic coordinates stay on the constraint manifold. Equation (2.57) is repeated below for convenience:

$$M\frac{\overrightarrow{v}_{j+1} - \overrightarrow{v}_j}{\Delta t} = -\nabla E(\overrightarrow{p}_j) - \nabla C(\overrightarrow{p}_{j+1})^T\lambda$$

Therefore, the constraint force:

$$\nabla C(\overrightarrow{p})^T\lambda$$

serves *two purposes*:

- Project a vector onto the *tangent space* of the constraint manifold. In this case, the Lagrange multipliers of the constraint force are given by Equation (5.9).
- Project (retract) a vector onto the constraint manifold *itself*. In this case, the Lagrange multipliers of the constraint force are given by the linear system in the Fast Projection algorithm, see Algorithm 1 Equation (2.78).

This observation carries over to $\mathbf{S}_+^{n,3}$ as well, see Theorem 5.4.1 where this observation is applied.

## 5.3 The Constraint Manifold

When constraints are placed on the manifold $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n\times 3}/\mathbf{O}_3$, the structure space is no longer $\mathbb{R}_*^{n\times 3}$. Rather for an arbitrary $n\times 3$ matrix $P$ of atomic coordinates, the constraints describes a structure space called the *constraint manifold*, denoted $\mathcal{C}(\mathbb{R}_*^{n\times 3})$. It is given by:

$$\mathcal{C}(\mathbb{R}_*^{n\times 3}) = \{P \in \mathbb{R}_*^{n\times 3} \mid \mathrm{Trace}(P^T A_i P) = q_{a_i b_i}\ i = 1,\ldots,m\}\ . \tag{5.10}$$

$\mathcal{C}(\mathbb{R}_*^{n\times 3})$ is an *embedded submanifold* of $\mathbb{R}_*^{n\times 3}$, see Section D.6. The optimization problem in Equation (5.1) is formulated on the quotient manifold:

$$\mathcal{C}(\mathbf{S}_+^{n,3}) = \mathcal{C}(\mathbb{R}_*^{n\times 3})/\mathbf{O}_3\ , \tag{5.11}$$

where $\mathcal{C}(\mathbf{S}_+^{n,3})$ denote the subset of rank 3 PSD Gram matrices from $\mathbf{S}_+^{n,3}$ that satisfy the constraints described in the set $\mathcal{C}(\mathbb{R}_*^{n\times 3})$.

Define the $m \times 1$ vector:

$$C(P) = \begin{pmatrix} C_1(P) \\ \vdots \\ C_m(P) \end{pmatrix} , \tag{5.12}$$

where

$$\begin{aligned} C_i(P) &= \frac{1}{2}\left(\text{Trace}(P^T A_i P) - q_{a_i b_i}\right) \\ &= \frac{1}{2}\left((e_{a_i} - e_{b_i})^T P P^T (e_{a_i} - e_{b_i}) - q_{a_i b_i}\right) \\ &= \frac{1}{2}\left(\parallel p_{a_i} - p_{b_i} \parallel^2 - q_{a_i b_i}\right) , \end{aligned} \tag{5.13}$$

where $A_i$ is an $n \times n$ symmetric matrix, see Equation (1.4). Then $P \in \mathcal{C}(\mathbf{S}_+^{n,3})$ if and only if $C(P) = \mathbf{0}_m$.

## 5.4 Equations of Motion

Informally, the equations of motion for constrained dynamics on $\mathbf{S}_+^{n,3}$ can be arrived at from the equations of motion for constrained dynamics on $\mathbb{R}^{3n}$ by replacing the $3n \times 1$ vector $\overrightarrow{p}$ with the $n \times 3$ matrix $P$ (see for example Equation (5.2) and Equation (5.3), and adding in the constraints). A more formal discussion is presented next.

The constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$ is the Riemannian manifold on which the equations of motion are formulated.

Consider the time interval $[t_j, t_{j+1}]$, let $\Delta t = t_{j+1} - t_j$. Let:

- $P_j = P(t_j)$ denote the matrix of atomic coordinates at the beginning of the time interval,
- $V_j$ be the velocity at the beginning of the time interval, and
- $V_{j+1}$ be the velocity at the end of the interval.

Let $P_{j+1} = P(t_{j+1})$ be the atomic coordinates at the end of the interval, this is the unknown to be determined.

Let there be $m$ constraints. The $i$-th quadrance constraint, $i \in \{1, \ldots, m\}$, involving atoms with atomic indices denoted by $a_i$ and $b_i$, is given by $C_i(P)$ in Equation (5.13):

**Theorem 5.4.1** (Equation of Motion for Constrained Dynamics on $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n\times 3}/\mathbf{O}_3$). *Let the constraint manifold be:*

$$\mathcal{C}(\mathbf{S}_+^{n,3}) = \mathcal{C}(\mathbb{R}_*^{n\times 3})/\mathbf{O}_3 \ .$$

*Let*

$$\widehat{M} = \begin{pmatrix} m_1 & & \\ & \ddots & \\ & & m_n \end{pmatrix} \ , \tag{5.14}$$

*be the $n \times n$ matrix of atomic masses, and $m_a$, $a = 1, \ldots, n$ be the mass of the $a$-th atom (do not confuse $m_a$ with $m$, the total number of constraints).*

*The following three equations are the equations of motion for the motion of a point on $\mathcal{C}(\mathbf{S}_+^{n,3})$.*

$$\widehat{M}\left(\frac{V_{j+1} - V_j}{\Delta t}\right) = -\nabla E(P_{j+1}) + P_{j+1}\Omega + \sum_{i=1}^{m} \lambda_i A_i P_{j+1} \ , \tag{5.15}$$

$$P_{j+1} = P_j + (\Delta t) V_{j+1} \ , \tag{5.16}$$

$$C_i(P_{j+1}) = 0 \text{ for } i = 1, \ldots m. \tag{5.17}$$

*where $\Omega$ is a skew symmetric matrix.*

*Proof.* The quantity $\nabla E(P_{j+1})$ is the force due to the potential energy, while $P_{j+1}\Omega$ is the component of $\nabla E(P_{j+1})$ in the direction vertical to the equivalence class at $P_{j+1}$. The component $\sum_{i=1}^{m} \lambda_i A_i P_{j+1}$ is a direction normal to the tangent space at $P_{j+1}$. Therefore, the force

$$\nabla E(P_{j+1}) - P_{j+1}\Omega - \sum_{i=1}^{m} \lambda_i A_i P_{j+1}$$

is a projection of $\nabla E(P_{j+1})$ in the horizontal space at $P_{j+1}$. See Section 6.3.1 for a description of these spaces, recall also Section 3.6.2.

Discretize the Euler-Lagrange Equations using the same approach as seen in Goldenthal et al. [30] Section 4.1, and also presented in Equations (2.57), (2.58), and (2.59) leads to Equations (5.15), (5.16), and (5.17). □

As discussed in Section 5.2.2, the constraint force serves two purposes: projecting onto the tangent space of the constraint manifold, and retracting onto the constraint manifold itself. Therefore, the Lagrange multipliers $\lambda_i$, $i = 1, \ldots, m$ in Equation 5.15 can be broken

down to two components: one component projects $\nabla E(P_{j+1})$ to the horizontal space, another component ensures Equation (5.17) holds. See Section 6.3.5.

Theorem 5.4.1 motivates the definition of the horizontal projection:

$$\text{Proj}_P(\eta) = \eta - P\Omega - \sum_{i=1}^{m} \lambda_i A_i P \ ,$$

where $\eta \in \mathbb{R}^{n\times 3}$. This projection will be defined formally in Section 6.3.2 after the tangent space is discussed.

**Proposition 5.4.1.** *Integrating the equations of motion from Theorem 5.4.1 is equivalent to solving the following constrained $\mathbf{S}_+^{n,3}$ optimization problem:*

$$\begin{aligned} &min \quad f(P) \\ &s.t. \quad Trace(P^T A_i P) = q_{a_i b_i} \quad i = 1, \ldots, m \ . \end{aligned} \tag{5.18}$$

*where*

$$f(P) = \frac{1}{(\Delta t)^2} Trace((P - P_j - (\Delta t)V_j)^T \widehat{M}(P - P_j - (\Delta t)V_j)) + E(P) \ . \tag{5.19}$$

*The optimal $P$ that minimizes this optimization problem is $P_{j+1}$.*

*Proof.* This can be seen as follows. Substituting $V_{j+1}$ in Equation (5.15) with Equation (5.16), then move every term to the left hand side of the equal sign gives:

$$\widehat{M}\left(\frac{P_{j+1} - P_j - (\Delta t)V_j}{(\Delta t)^2}\right) + \nabla E(P_{j+1}) - P_{j+1}\Omega - \sum_{i=1}^{m} \lambda_i A_i P_{j+1} = \mathbf{0}_{n\times 3} \ . \tag{5.20}$$

The Euclidean gradient $\nabla f(P)$ projected in the horizontal direction

$$\text{grad} f(P) = \text{Proj}_P(\nabla f(P)) \ ,$$

is the definition of the gradient on the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$; it is called the Riemannian gradient to distinguish it from $\nabla f(P)$, see Section 6.3.3. Equation (5.20) shows

$$\text{grad} f(P_{j+1}) = \mathbf{0}_{n\times 3}$$

which means $P_{j+1}$ is a critical point of the constrained optimization problem in Equation (5.18). □

Recall from Section 2.8.2 that the Fast Projection algorithm finds a sequence of *small* steps to the constraint manifold instead of one step. A similar idea is also used by the RTR algorithm. Recall from Section 3.5 the RTR algorithm generates a sequence of iterates $P_j^{(0)}, P_j^{(1)}, \ldots$ such that $\text{grad} f(P_j^{(k)}) \to \mathbf{0}_{n\times 3}$ as $k \to \infty$ and therefore $P_j^{(k)} \to P_{j+1}$. The following theorem shows the potential energy alone can be minimized to find $P_{j+1}$.

**Theorem 5.4.2.** *Suppose the current protein conformation is $P_j$ and $P_{j+1}$ is the conformation to be found using the RTR algorithm (Algorithm 2). Let the sequence of iterates generated by the RTR algorithm be given by $P_j^{(0)}, P_j^{(1)}, \ldots,$ such that $P_j^{(k)} \to P_{j+1}$ as $k \to \infty$, this means also that $P_j^{(k)} - P_j^{(k-1)} \to \mathbf{0}_{n\times 3}$.*

*$P_{j+1}$ is a critical point of the optimization problem from Equation (5.18) if and only if it is a critical point of the optimization problem:*

$$\begin{aligned} &min \quad E(P) \\ &s.t. \quad Trace(P^T A_i P) = q_{a_i b_i} \quad i = 1, \ldots, m \ , \end{aligned} \tag{5.21}$$

*where $E(P)$ is the PSD potential energy.*

*Proof.* Let $\mathcal{R}_P(\eta)$ denote the retraction which projects an unconstrained step $\eta$ onto the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$, see Section 6.3.5. The RTR algorithm uses the tCG algorithm (Algorithm 3) to propose an unconstrained step $\eta$. Let $\eta^{(k)}$ denote the $k$-th accepted unconstrained step proposed by the tCG algorithm. If $V_j \neq \mathbf{0}_{n\times 3}$, then the first constrained step is

$$P^{(1)} = \mathcal{R}_{P^{(0)}}((\Delta t)V_j + \eta^{(0)}) \ ,$$

otherwise, the first constrained step is:

$$P^{(1)} = \mathcal{R}_{P^{(0)}}(\eta^{(0)}) \ .$$

Subsequent steps are given by:

$$P^{(k)} = \mathcal{R}_{P^{(k-1)}}(\eta^{(k-1)})$$

Starting with $P_j^{(0)} = P_j$, the RTR algorithm generates a sequence of iterates $P_j^{(0)}, P_j^{(1)}, \ldots$

which gives a sequence of gradients:

$$\begin{aligned}
\text{grad}f(P_j^{(1)}) &= \frac{\widehat{M}}{(\Delta t)^2}\left(P_j^{(1)} - P_j^{(0)}\right) + \nabla E(P_j^{(1)}) - P_j^{(1)}\Omega - \sum_{i=1}^{m}\lambda_i A_i P_j^{(1)} \\
&\vdots \\
\text{grad}f(P_j^{(k)}) &= \frac{\widehat{M}}{(\Delta t)^2}\left(P_j^{(k)} - P_j^{(k-1)}\right) + \nabla E(P_j^{(k)}) - P_j^{(k)}\Omega - \sum_{i=1}^{m}\lambda_i A_i P_j^{(k)} \ .
\end{aligned}$$

As $k \to \infty$, $P_j^{(k)} - P_j^{(k-1)} \to \mathbf{0}_{n\times 3}$. Therefore,

$$\text{grad}f(P_j^{(k)}) \to \mathbf{0}_{n\times 3}$$

if and only if

$$\text{grad}E(P_j^{(k)}) = \nabla E(P_j^{(k)}) - P_j^{(k)}\Omega - \sum_{i=1}^{m}\lambda_i A_i P_j^{(k)} \to \mathbf{0}_{n\times 3} \ .$$

Therefore, as $P_j^{(k)} \to P_{j+1}$, $P_j^{(k)}$ approaches a critical point of the optimization problem given in Equation (5.18) if and only if it approaches a critical point of the optimization problem given in Equation (5.21), which minimizes $E(P)$ only. □

## 5.5 Removing the Conservation of Linear Momentum Constraint

As discussed in Section 2.7, minimizing the Hookean potential is ill-posed because a translated solution is also a solution. Note that ill-posed as used here means the solution is not unique, not that a solution cannot be found.

Kim proposed to add a constraint in order to conserve the linear moment, this constraint is given in Equation (2.44) and is repeated below for convenience.

$$\sum_{a=1}^{n}\delta_a(t) = \mathbf{0}_3$$

On $\mathbf{S}_+^{n,3}$, $\delta(t)$ is a $n \times 3$ matrix, whose rows are $\delta_a(t)$ (see Equation 4.7)). Therefore, on

$\mathbf{S}_+^{n,3}$ this constraint can be written in matrix notation:

$$\begin{pmatrix}\delta_1(t) & \dots & \delta_n(t)\end{pmatrix}\begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix} = \delta(t)^T\mathbf{1}_n = \mathbf{0}_n \tag{5.22}$$

On $\mathbf{S}_+^{n,3}$, facial reduction can be used to remove this constraint. This result is arrived at from the *Centring Constraint Reduction Theorem*, from Krislock [44] Theorem 4.13 and Alipanahi [4] Section 3.3.5).

Recall the problem context is to find $P_{j+1}$ from $P_j$. To simplify the discusion here, let $P = P_j$ (drop the subscript), and consider $\delta = \delta(t)$ (drop the time parameter $t$ as well). Now the problem under discussion is that of finding $\delta$ such that $(P+\delta)^T\mathbf{1}_n = \mathbf{0}_n$.

Recall Equation (C.25) gives the definition of $\mathbf{S}_C^n$, the space of centered symmetric matrices. Let $X = PP^T \in \mathbf{S}_+^{n,3} \cap \mathbf{S}_C^n$ be an $n \times n$ Gram matrix, this means $P^T\mathbf{1}_n = \mathbf{0}_n$. If $\delta$ satisfies Equation (5.22) then $(P+\delta)^T\mathbf{1}_n = \mathbf{0}_n$ as well, ensuring the Gram matrix of the next conformation is centered $(P+\delta)(P+\delta)^T \in \mathbf{S}_+^{n,3} \cap \mathbf{S}_C^n$.

In order to remove the constraint of Equation (5.22), first factorize $P$ as $P = US$ where $U$ is an $n \times (n-1)$ matrix, and $S$ is an $(n-1) \times 3$ matrix. Note that

$$P^T\mathbf{1}_n = \mathbf{0}_n \text{ if and only if } U^T\mathbf{1}_n = \mathbf{0}_n \ .$$

This is equivalent to the condition:

$$\text{range}(U)^\perp = \text{range}(\mathbf{1}_n) \ . \tag{5.23}$$

The required $U$ can thus be found by first performing QR decomposition on $\mathbf{1}_n$ to find an $n \times n$ matrix $Q$. Next take all columns of $Q$ except the first column and denote this matrix $U$, i.e. $U = Q[:, 2:n]$. Then the matrix $U$ satisfies Equation (5.23). Define:

$$\delta = U\delta_U$$

The original problem of finding $\delta$ now becomes the problem of finding $\delta_U$, without the centering constraint since:

$$(P+\delta)^T\mathbf{1}_n = (US + U\delta_U)^T\mathbf{1}_n = (S+\delta_U)^TU^T\mathbf{1}_n = \mathbf{0}_n \ .$$

Another option to ensure $(P+\delta)^T\mathbf{1}_n = \mathbf{0}_n$ is to use the "step and project" paradigm, also mentioned in Section 2.8.2. Proposition 2.30 of Krislock [44] gives the following projection

from $\mathbf{S}^n$ to $\mathbf{S}^n_C$:

$$\text{Proj}_{\mathbf{S}^n_C}(X) = JXJ$$

where $X \in \mathbf{S}^n$ and $J$ is given by:

$$J = I_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T \, .$$

For the rank 3 Gram matrix $X = PP^T \in \mathbf{S}^{n,3}_+$, this projection has the meaning of subtracting the centroid of the points in $P$ from each row of $P$, where each row is the atomic coordinates of one point. This is straightforward to see. Consider:

$$JPP^TJ = (JP)(JP)^T \, ,$$

then

$$JP = P - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^TP \, ,$$

where

$$\frac{1}{n}\mathbf{1}_n^TP$$

is the centroid. Starting with $P$, if after taking a step $\delta$, the new Gram matrix

$$(P+\delta)(P+\delta)^T$$

does not satisfy $(P+\delta)(P+\delta)^T\mathbf{1}_n = \mathbf{0}_n$, applying the centering projection

$$J(P+\delta)(P+\delta)^TJ$$

will give a new projected Gram matrix that is centered. The step and project method for ensuring the Gram matrix stays centered has the advantage that the matrix $J$ is easier to compute than finding the face matrix $U$ explicitly.

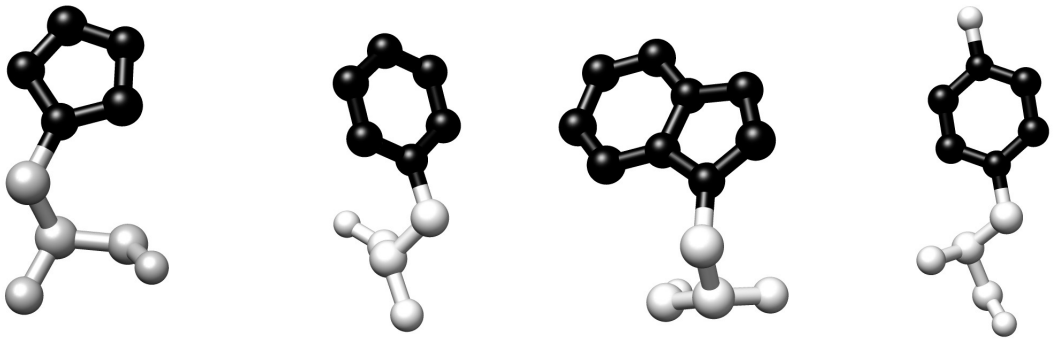
## 5.6 Constraint Reduction for Rigid Groups of Atoms

A *clique* is defined in the context of the EDMC problem in Definition C.6.1. A clique is called a *rigid cluster* by Kim with the introduction of rigid cluster ENI (see Section C.6.5 and [41, 37]). In the context of protein dynamics, a clique or rigid cluster is a group of atoms that move concurrently with their mutual distances not changing as a protein undergoes conformational change.

The reformulation of rigid cluster ENI using facial reduction of the PSD ENI potential [54] shows that modelling of rigid groups of atoms can be accomplished by using faces of $\mathbf{S}_+^{n,3}$, whereas faces of $\mathbb{R}^{3n}$ have no such meaning. This is reviewed in Section C.6.5.

The assumption about which groups of atoms in a protein structure should be modelled as rigid is a separate research problem and is not examined in this thesis. Amino acid side chains that contain aromatic rings are known to be rigid, and well be used as examples of rigid clusters in the discussion here to make the discussion more concrete (less abstract).

Figure 5.1 presents a number of amino acids with rigid (aromatic) side chains, the atoms in black make up the aromatic rings. These structures are visualized in UCSF Chimera [63] and are part of a larger protein, the hydrogen atoms have been omitted from the PDF file. For each side chain, the bond attaching the side chain to the protein backbone is made up of the $\beta$-carbon (gray atom) and $\gamma$-carbon (black atom) bond about which the side chain rotates. Therefore, the $\beta$-carbon is also a part of the side chain clique. Because these side chains are all 2D, in order to model their rotation in 3D about the $\beta$-carbon $\gamma$-carbon bond, one additional out-of-plane "pseudo-atom" is needed, which is also part of the side chain clique.



(a) Histidine (b) Phenylalanine (c) Tryptophan (d) Tyrosine

Figure 5.1: Amino acids with rigid side chains (aromatic rings are colored in black).

The number of constraints required to keep the aromatic side chain clique, including the $\beta$-carbon and the out-of-plane carbon, rigid is discussed next, and summarized in Table 5.2.

Let $\mathcal{C} \subset \{1, \ldots, n\}$ denote the subset of atomic indices for the atoms in a clique. The number of constraints needed on $\mathbb{R}^{3n}$ to enforce a rigid group of $|\mathcal{C}|$ atoms has been

discussed by Plantenga [64, 56]. This number is

$$3|\mathcal{C}| - 6 \ .$$

This is because a group of $|\mathcal{C}|$ atoms that move rigidly has only $3|\mathcal{C}| - 6$ degrees of freedom. The total number of possible pairwise constraints without repetition, is

$$\frac{|\mathcal{C}|(|\mathcal{C}| - 1)}{2} \ .$$

Therefore, if $|\mathcal{C}| > 4$, some constraints must be discarded. The optimal subset of constraints to keep is the subset of $3|\mathcal{C}| - 6$ gradients such that their constraint gradients *minimize the condition number* of the matrix of constraint gradients:

$$\nabla C(\overrightarrow{p})^T = \begin{pmatrix} \nabla C_{\sigma(1)}(\overrightarrow{p})^T & \dots & \nabla C_{\sigma(3|\mathcal{C}|-6)}(\overrightarrow{p})^T \end{pmatrix}$$

where $\sigma(1), \dots, \sigma(3|\mathcal{C}| - 6)$ are the indices of the constraints kept. The optimal set of constraints to keep are decided by first assembling all $|\mathcal{C}|(|\mathcal{C}| - 1)/2$ constraint gradients, the $3n \times 1$ vector $\nabla C_i(\overrightarrow{p})^T$ as defined in Equation (2.50), in a matrix. The columns can be arranged in any order. Then perform rank revealing QR factorization on this matrix, and choose the first $3|\mathcal{C}| - 6$ columns chosen by this factorization.

Section 2.8.3 discussed two other situations where the matrix $\nabla C(\overrightarrow{p})$ can be rank deficient. Table 5.1 summarizes these discussions.

| | When is the matrix $\nabla C(\overrightarrow{p})$ rank deficient or ill-conditioned |
|---|---|
| 1 | For a constraint on the distance between the $a$-th and $b$-th atom, if $p_a - p_b = \mathbf{0}_3$ , then $\nabla C(\overrightarrow{p})$ will have a row of zeros. |
| 2 | For all constraints that the $a$-th atom is involved in, if there exists atomic indices $b_1$ and $b_2$ such that, $p_a - p_{b_1} = \alpha(p_a - p_{b_2})$ , where $\alpha \in \mathbb{R}$ is an arbitrary constant. |
| 3 | In a clique of size $\|\mathcal{C}\|$, choosing more than $3\|\mathcal{C}\| - 6$ constraints, or choosing $3\|\mathcal{C}\| - 6$ constraints which do not minimize the condition number of $\nabla C(\overrightarrow{p})$. |

Table 5.1: A summary of when $\nabla C(\overrightarrow{p})$ is rank-deficient.

On $\mathbf{S}_+^{n,3} \cap \mathbf{S}_C^n$, the number of constraints required can be reduced by applying Theorem 4.16 of Krislock [44], the *Distance Constraint Reduction Theorem*. This theorem says to find a smaller clique $\mathcal{C}_{small} \subset \mathcal{C}$ such that the EDM for both cliques have the same embedding dimension, then enforcing constraints for this smaller clique $\mathcal{C}_{small}$ is all that is necessary to ensure the original clique $\mathcal{C}$ stays rigid. The smallest number of atoms to have an embedding dimension of 3 is *four* atoms, therefore $|\mathcal{C}_{small}| = 4$. This is *also* the number for which rank-revealing QR factorization is *not needed*. Thus, the Distance Constraint Reduction Theorem has removed the need to perform rank-revealing QR factorization.

| Amino acid | $\|\mathcal{C}\|$ | Number of constraints on $\mathbb{R}^{3n}$ $(3\|\mathcal{C}\| - 6)$ | Number of constraints on $\mathbf{S}_+^{n,3} \cap \mathbf{S}_C^n$ $(3(4) - 6)$ |
|---|---|---|---|
| Histidine | 7 | 15 | 6 |
| Phenylalanine | 8 | 18 | 6 |
| Tryptophan | 11 | 27 | 6 |
| Tyrosine | 9 | 21 | 6 |

Table 5.2: Number of constraints to enforce rigidity of $|\mathcal{C}|$ atoms in aromatic side chain (includes $\beta$-carbon and out-of-plane atom).

## 5.7 Summary

This Chapter formulated the equations of motion for protein ENMs on $\mathbf{S}_+^{n,3}$. Facial reduction theories were also used to remove the constraint for conservation of linear momentum, and to decrease the number of constraints required to keep a clique rigid.

# Chapter 6

# Constrained Optimization with Fixed Rank PSD Matrices

## 6.1 Introduction

The PSD-EDM-D problem, defined in Definition 5.1.1 is a constrained optimization problem on $\mathbf{S}_+^{n,3}$. Solving constrained fixed rank PSD matrix manifold optimization problems has been discussed in Journée et al. [34], where the geometric objects required by matrix manifold optimization algorithms, such as the RTR algorithm, were presented. The original discussion assumes matrices $A_i$ in the problem above satisfy:

$$A_i A_j = \mathbf{0}_{n\times n} \text{ if } i \neq j \ , \tag{6.1}$$

This assumption is given as "Assumption 2" of [34], and is appropriate for the problems they analyzed. However, this assumption is too restrictive to describe constraints for ENMs. This Chapter begins with a discussion on why the quadrance constraints of ENMs do not satisfy Equation (6.1). Thereafter, the geometric objects required by the RTR algorithm are presented.

This Chapter is organized as follows:

- Section 6.2 describes why protein structure constraints do not satisfy "Assumption 2" of Journée et al.[34].
- Section 6.3 describes the geometric objects required by matrix manifold algorithms,

such as the RTR algorithm, for constrained optimization with fixed rank PSD matrices without "Assumption 2".

- Section 6.3.1 describes the tangent space.
- Section 6.3.2 presents the horizontal projection.
- Section 6.3.3 presents the Riemannian gradient.
- Section 6.3.4 presents the Riemannian Hessian.
- Section 6.3.5 presents retraction. The full retraction is derived using the retraction from Journée et al.. The Fast Retraction is then given, which is an approximation to the full retraction that is more efficient for large $n$ and $m$, and is derived by linearly approximating the full retraction.
- Section 6.4 discusses the agreements between constrained optimization on $\mathbb{R}^{3n}$ and $\mathbf{S}_+^{n,3}$.

The background material from Chapter 3 should be read prior to this Chapter.

## 6.2 Removing Assumption 2 of Journée et al.

Suppose the $i$-th quadrance constraint involves atoms with atomic indices $a_i$ and $b_i$, and suppose there are $m$ such pairs of atomic indices:

$$(a_1, b_1), \ldots, (a_i, b_i), \ldots, (a_m, b_m)$$

where the subscript $i = 1, \ldots, m$ is used to index the constraints. The matrix $A_i$ for the $i$-th quadrance constraint is the $n \times n$ symmetric matrix:

$$A_i = (e_{a_i} - e_{b_i})(e_{a_i} - e_{b_i})^T \ ,$$

first introduced in Equation (1.4) together with the stamp matrix because of their similar roles.

Let the $j$-th quadrance constraint involve atomic indices $a_j$ and $b_j$. When the $i$-th quadrance constraint and $j$-th quadrance constraint do not share any atom, i.e. they are not adjacent bonds, then the atomic indices $a_i$, $b_i$, $a_j$, and $b_j$ are all different, resulting in

$$(e_{a_i} - e_{b_i})^T (e_{a_j} - e_{b_j}) = 0 \ ,$$

matrices $A_i$ and $A_j$ satisfying Equation (6.1).

A particular atomic index may appear in more than one quadrance constraint when bonds are adjacent. When the $i$-th quadrance constraint and $j$-th quadrance constraint are sharing one atomic index, one of the following conditions will be true for atomic indices $a_i$, $b_i$, $a_j$, and $b_j$ (despite these indices being labelled differently):

$$\begin{aligned} a_i &= a_j \text{ or} \\ a_i &= b_j \text{ or} \\ b_i &= a_j \text{ or} \\ b_i &= b_j \ . \end{aligned} \tag{6.2}$$

Since the atomic indices $a_i$, $b_i$, $a_j$, and $b_j$ are all labelled differently, the notation does not indicate which one of the situations in Equation (6.2) holds. This information is not needed for this discussion. It is only important to note in the situation described by Equation (6.2) "Assumption 2" does not hold:

$$A_i A_j \neq \mathbf{0}_{n\times n} \text{ if } i \neq j \ . \tag{6.3}$$

In order to model protein constraints, "Assumption 2" of [34] needs to be removed. This Chapter will discuss how to remove "Assumption 2" for the geometric objects discussed in [34].

## 6.3 Geometric Objects for Constrained Optimization on $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n\times 3}/\mathbf{O}_3$

### 6.3.1 The Tangent Space

The constraint manifold $\mathcal{C}(\mathbb{R}_*^{n\times 3})$ was define in Equation (5.10) and $\mathcal{C}(\mathbf{S}_+^{n,3})$ was defined in Equation (5.11). The tangent space of $\mathcal{C}(\mathbb{R}_*^{n\times 3})$ at a point $P$ is denoted $T_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$. It is given by differentiating the constraint equations describing $\mathcal{C}(\mathbb{R}_*^{n\times 3})$ in Equation (5.10):

$$T_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) = \{\eta \in \mathbb{R}^{n\times 3} \mid \text{Trace}(\eta^T A_i P) = 0 \ i = 1, \ldots, m\} \ . \tag{6.4}$$

$T_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ can be decomposed into two orthogonal spaces, the vertical space and the horizontal space. The vertical space, denoted $\mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ is given by:

$$\mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) = \{P\Omega \mid \Omega \in \mathbb{R}^{3\times 3}, \Omega = -\Omega^T\} \,. \tag{6.5}$$

The horizontal space is denoted by $\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$. From the discussion in Section 3.6.2 a vector in the horizontal space, $\eta \in \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ is orthogonal to $P\Omega \in \mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$, meaning that $\text{Trace}(\eta^T P\Omega) = 0$ holds. This orthogonality in turn requires $\eta^T P$ to be a symmetric matrix. Therefore, the horizontal space is given as (see [34]):

$$\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) = \{\eta \in T_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \mid \eta^T P = P^T\eta\} \,. \tag{6.6}$$

The normal space at $P$, $N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$, is orthogonal to $T_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ and is given by:

$$N_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) = \{\sum_{i=1}^{m} \lambda_i A_i P \mid \lambda_i \in \mathbb{R}\} \,, \tag{6.7}$$

where $\lambda_i$ are $m$ Lagrangian multipliers. This definition of the normal space, which is the definition originally presented in [34], is a subset of the horizontal space:

$$N_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \subset \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \,. \tag{6.8}$$

This inclusion can be verified as follows. Consider an arbitrary vector in the normal space:

$$\sum_{i=1}^{m} \lambda_i A_i P \,,$$

this vector satisfies:

$$(\sum_{i=1}^{m} \lambda_i A_i P)^T P = P^T(\sum_{i=1}^{m} \lambda_i A_i)P = P^T(\sum_{i=1}^{m} \lambda_i A_i P)$$

and is therefore in $\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ as well. It would be more precise to *redefine* $\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ as:

$$\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) = \{\eta \in T_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \mid \eta^T P = P^T\eta \text{ and } \text{Trace}(\eta^T A_i P) = 0 \; i = 1, \ldots, m\} \,. \tag{6.9}$$

This definition explicitly removes the vectors in the normal space. In practice, to ensure a tangent vector $\eta$ satisfies:

$$\text{Trace}(\eta^T A_i P) = 0 \; i = 1, \ldots, m$$

its normal space component can be removed using the horizontal projection, which is discussed in Section 6.3.2.

Note that vectors in the vertical space $P\Omega \in \mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ are already in the tangent space and not in the normal space since:

$$\text{Trace}(P^T A_i P\Omega) = 0$$

holds because $P^T A_i P$ is a symmetric matrix.

**Definition 6.3.1.** *The tangent space at a point $P$ of the quotient manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$, denoted $T_P\mathcal{C}(\mathbf{S}_+^{n,3})$, is the space of all vectors from the horizontal space $\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$.*

**Corollary 6.3.1** (Corollary to Proposition 3.6.1)**.**

$$T_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \simeq \mathbb{R}^{n\times 3} \; .$$

*Proof.* Vectors in the normal space are of the form:

$$\sum_{i=1}^{m} \lambda_i A_i P \; .$$

This is a linear combination of $\{A_1 P, \ldots, A_m P\}$. Hence,

$$\dim(N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) = m \tag{6.10}$$

Equation (3.9) shows $\mathcal{V}_P\mathbb{R}_*^{n\times 3} = \mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$. Therefore:

$$\dim(\mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) = \dim(\mathcal{V}_P\mathbb{R}_*^{n\times 3}) = \frac{3(3-1)}{2} \tag{6.11}$$

The horizontal space $\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ is defined in Equation (6.9) and $\mathcal{H}_P\mathbb{R}_*^{n\times 3}$ was defined in Equation (3.11). Therefore,

$$\mathcal{H}_P\mathbb{R}_*^{n\times 3} = \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \oplus N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$$

and

$$\begin{aligned}\dim(\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) &= \dim(\mathcal{H}_P\mathbb{R}_*^{n\times 3}) - \dim(N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) \\ &= (n-3)3 + \frac{3(3+1)}{2} - m\end{aligned} \tag{6.12}$$

From Equations (6.10), (6.11), (6.12), it follows:

$$\begin{aligned}\dim(T_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) &= \dim(N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) + \dim(\mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) + \dim(\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})) \\ &= m + \frac{3(3-1)}{2} + (n-3)3 + \frac{3(3+1)}{2} - m \\ &= \frac{3(3-1)}{2} + (n-3)3 + \frac{3(3+1)}{2} \\ &= \frac{9}{2} - \frac{3}{2} + 3n - 9 + \frac{9}{2} + \frac{3}{2} \\ &= 3n\ .\end{aligned} \tag{6.13}$$

Therefore, it follows that:

$$\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \oplus N_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \oplus \mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \simeq \mathbb{R}^{n\times 3}\ ,$$

as required. □

### 6.3.2 Projecting to the Horizontal Space

The horizontal projection is used to map a vector to the tangent space of the constraint manifold. The horizontal projection produces a vector orthogonal to the vertical space, $\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$, and the normal space, $N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$. The horizontal projection map, denoted $\text{Proj}_P$, has domain $T_P\mathcal{C}(\mathbb{R}_*^{n\times 3}) \simeq \mathbb{R}^{n\times 3}$ and image $\mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$:

$$\text{Proj}_P : \mathbb{R}^{n\times 3} \to \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})\ .$$

The map is given by (see also Theorem 9 of [34]):

$$\begin{aligned}\text{Proj}_P(\eta) &= \eta - P\Omega - \sum_{i=1}^{m} \lambda_i A_i P \\ &= \eta - P\Omega - E_b \text{Diag}(\lambda) E_b^T P\ ,\end{aligned} \tag{6.14}$$

where $\lambda = (\lambda_1, \ldots, \lambda_m)^T \in \mathbb{R}^m$. $E_b$ is an $n \times m$ sparse matrix defined as:

$$E_b = \begin{pmatrix} e_{a_1} - e_{b_1} & \ldots & e_{a_m} - e_{b_m} \end{pmatrix} \tag{6.15}$$

where $E_b$ has column $i$ given by $e_{a_i} - e_{b_i}$, $i = 1, \ldots, m$. The Diag operator was defined in Equation (C.31) and (C.32).

The formula for the horizontal projection has the meaning of removing the vertical space and the normal space, leaving behind only the horizontal space.

Since $\text{Proj}_P(\eta) \in \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$,

$$P^T\text{Proj}_P(\eta) = \text{Proj}_P(\eta)^T P$$

holds, where $\Omega$ is a skew-symmetric matrix that solves the Sylvester equation (see Section E.1) :

$$\Omega P^T P + P^T P \Omega = P^T\eta - \eta^T P \, . \tag{6.16}$$

Since $\text{Proj}_P(\eta) \in T_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$,

$$\text{Trace}(P^T A_i \text{Proj}_P(\eta)) = 0 \, ,$$

holds, and if $A_iA_j = \mathbf{0}_{n\times n}$ holds, $\lambda_i$ in Equation (6.14) is given by:

$$\lambda_i = \frac{\text{Trace}(\eta^T A_i P)}{\text{Trace}(P^T A_i^2 P)} \, . \tag{6.17}$$

If $A_iA_j \neq \mathbf{0}_{n\times n}$, Equation (6.17) cannot be used. Consider writing Equation (6.17) out in matrix form:

$$\begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{pmatrix} = \begin{pmatrix} \frac{1}{\text{Trace}(P^T A_1 A_1 P)} & & \\ & \ddots & \\ & & \frac{1}{\text{Trace}(P^T A_m A_m P)} \end{pmatrix} \begin{pmatrix} \text{Trace}(\eta^T A_1 P) \\ \vdots \\ \text{Trace}(\eta^T A_m P) \end{pmatrix}$$

This matrix equation shows $\lambda = (\lambda_1, \ldots, \lambda_m)^T \in \mathbb{R}^m$ is the solution to the system:

$$\begin{pmatrix} \text{Trace}(P^T A_1 A_1 P) & & \\ & \ddots & \\ & & \text{Trace}(P^T A_m A_m P) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{pmatrix} = \begin{pmatrix} \text{Trace}(\eta^T A_1 P) \\ \vdots \\ \text{Trace}(\eta^T A_m P) \end{pmatrix} \, . \tag{6.18}$$

Let

$$Q = \begin{pmatrix} \mathrm{Trace}(P^T A_1 A_1 P) & & \\ & \ddots & \\ & & \mathrm{Trace}(P^T A_m A_m P) \end{pmatrix} \tag{6.19}$$

$Q$ is an $m \times m$ matrix, and in additional, $Q$ is a diagonal matrix because the assumption $A_i A_j = \mathbf{0}_{n\times n}$ when $i \neq j$ holds. When $A_i A_j \neq \mathbf{0}_{n\times n}$, the matrix $Q$ is given by:

$$\begin{aligned} Q &= \begin{pmatrix} \mathrm{Trace}(P^T A_1 A_1 P) & \dots & \mathrm{Trace}(P^T A_m A_1 P) \\ \vdots & & \vdots \\ \mathrm{Trace}(P^T A_m A_1 P) & \dots & \mathrm{Trace}(P^T A_m A_m P) \end{pmatrix} \\ &= \begin{pmatrix} \varepsilon_{11}({p_{a_1b_1}}^T p_{a_1b_1}) & \dots & \varepsilon_{1m}({p_{a_1b_1}}^T p_{a_mb_m}) \\ \vdots & & \vdots \\ \varepsilon_{m1}({p_{a_mb_m}}^T p_{a_1b_1}) & \dots & \varepsilon_{mm}({p_{a_mb_m}}^T p_{a_mb_m}) \end{pmatrix} \end{aligned} \tag{6.20}$$

where $\varepsilon_{ij} = (e_{a_i} - e_{b_i})^T (e_{a_j} - e_{b_j})$ and ${p_{a_ib_i}}^T = (e_{a_i} - e_{b_i})^T P$.

Define the $m \times 1$ vector $\tau$ as:

$$\tau = \begin{pmatrix} \mathrm{Trace}(\eta^T A_1 P) \\ \vdots \\ \mathrm{Trace}(\eta^T A_m P) \end{pmatrix} = \begin{pmatrix} {\eta_{a_1b_1}}^T p_{a_1b_1} \\ \vdots \\ {\eta_{a_mb_m}}^T p_{a_mb_m} \end{pmatrix} , \tag{6.21}$$

where ${\eta_{a_ib_i}}^T = (e_{a_i} - e_{b_i})^T \eta$. Then, Equation (6.18) generalizes to the following linear system if $A_i A_j \neq \mathbf{0}_{n\times n}$ when $i \neq j$:

$$Q\lambda = \tau \tag{6.22}$$

**Proposition 6.3.1.** *The matrix $Q$ defined in Equation (6.20) is a sparse $m \times m$ matrix, with only $O(m)$ nonzero entries. The nonzero entries of $Q$ are the nonzero entries of $E_b^T E_b$.*

*Proof.* The $i$-th row of $Q$ is

$$\varepsilon_{i1}(p_{a_ib_i})^T p_{a_1b_1} \dots \varepsilon_{im}(p_{a_ib_i})^T p_{a_mb_m} .$$

The entry $\varepsilon_{ij}(p_{a_ib_i})^T p_{a_jb_j}$ is nonzero if and only if

$$\varepsilon_{ij} = (e_{a_i} - e_{b_i})^T (e_{a_j} - e_{b_j}) \neq 0 ,$$

which holds when the bond $(a_j, b_j)$ is adjacent to the bond $(a_i, b_i)$. Hence, the number of

nonzero entries per row is the number of bonds adjacent to the bond $(a_i, b_i)$. This number is small for all atoms in a protein's amino acid. For example, at most four bonds can be adjacent for the carbon atom. Denote this number by $c$, compared to the total number of constraints $m$, $c \ll m$. Since there are $m$ rows in $Q$, the total number of nonzero entries of $Q$ is $O(cm) = O(m)$. It follows that $Q$ has many zeros, and requires only $O(m)$ operations to form the nonzero entries.

Using the definition of $E_b$, it is straightforward to perform the multiplication $E_b^T E_b$, and see that the $ij$-th entries of this product is exactly $\varepsilon_{ij} = (e_{a_i} - e_{b_i})^T (e_{a_j} - e_{b_j})$. This means the position of nonzero entries for $Q$ and $E_b^T E_b$ coincide. □

The matrix Q is also given by the following matrix expression:

$$Q = (E_b^T E_b) \circ (E_b^T P P^T E_b) \ , \tag{6.23}$$

The operator $\circ$ is element-wise multiplication. However, since $Q$ has only $O(m)$ entries, another way of forming $Q$ is to only calculate these nonzero entries.

The vector $\tau$ is also given by the following matrix expression:

$$\tau = \text{sumrow}((E_b^T \eta) \circ (E_b^T P)) \ .$$

The operator sumrow$(\cdot)$ sums the row of the argument matrix.

### 6.3.3 The Riemannian Gradient

Let $\nabla f(P)$ be the Euclidean gradient of $f(P)$. The Riemannian gradient is a vector on $T_P\mathcal{C}(\mathbf{S}_+^{n,3})$ and by Definition 6.3.1 it is orthogonal to the vertical space, $\mathcal{V}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ and the normal space, $N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$. The horizontal projection from Equation (6.14) is applied to the Euclidean gradient to ensure these conditions are satisfied:

$$\text{grad} f(P) = \text{Proj}_P(\nabla f(P)) \ . \tag{6.24}$$

Let $\zeta = \nabla f(P)$, the formula for the horizontal projection from Section 6.3.2 gives the following expression for the Riemannian gradient:

$$\begin{aligned} \text{Proj}_P(\zeta) &= \zeta - P\Omega - \sum_{i=1}^{m} \lambda_i A_i P \\ &= \zeta - P\Omega - E_b \text{Diag}(\lambda) E_b^T P \ , \end{aligned} \tag{6.25}$$

where $\Omega$ now satisfies:

$$\Omega P^T P + P^T P \Omega = P^T \zeta - \zeta^T P \ . \tag{6.26}$$

The define the $m \times 1$ vector $\tau_{grad}$ by:

$$\tau_{grad} = \begin{pmatrix} \text{Trace}(\zeta^T A_1 P) \\ \vdots \\ \text{Trace}(\zeta^T A_m P) \end{pmatrix} = \begin{pmatrix} {\zeta_{a_1 b_1}}^T p_{a_1 b_1} \\ \vdots \\ {\zeta_{a_m b_m}}^T p_{a_m b_m} \end{pmatrix} = \text{sumrow}((E_b^T \zeta) \circ (E_b^T P)) \ , \tag{6.27}$$

where ${\zeta_{a_i b_i}}^T = (e_{a_i} - e_{b_i})^T \zeta$. Then, the required $\lambda$ for projecting the gradient is the solution to the system:

$$Q\lambda = \tau_{grad} \ . \tag{6.28}$$

### 6.3.4 The Riemannian Hessian

Recall from the discussion in Section 3.6.5, the Riemannian Hessian in the direction $\eta \in \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ is defined using the Riemannian connection. Similar to Equation (3.33) this is given by:

$$\text{Hess} f(P)[\eta] = \nabla_\eta \text{grad} f(P) = \text{Proj}_P(D(\text{grad} f(P))[\eta]) \ . \tag{6.29}$$

As Equations (6.24) and (6.29) show, evaluating the Hessian require the directional derivative of the projection operator in the direction $\eta \in \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$. Let $\zeta = \nabla f(P) \in \mathbb{R}^{n\times 3}$, let $D\zeta[\eta] \in \mathbb{R}^{n\times 3}$ be the directional derivative of $\zeta$ in the $\eta$ direction :

$$\begin{aligned} &D(\text{grad} f(P))[\eta] = D(\text{Proj}_P(\zeta))[\eta] \\ &= D(\zeta - P\Omega - \sum_{i=1}^m \lambda_i A_i P)[\eta] \\ &= D\zeta[\eta] - (\eta\Omega + PD\Omega[\eta]) - \sum_{i=1}^m \lambda_i A_i \eta - \sum_{i=1}^m D\lambda_i[\eta] A_i P \ , \\ &= D\zeta[\eta] - (\eta\Omega + PD\Omega[\eta]) - E_b \text{Diag}(\lambda) E_b^T \eta - E_b \text{Diag}(D\lambda[\eta]) E_b^T P \ . \end{aligned} \tag{6.30}$$

The skew-symmetric matrix $D\Omega[\eta] \in \mathbb{R}^{3\times 3}$ solves the Sylvester equation given by differentiating Equation (6.26) in the $\eta \in \mathcal{H}_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$ direction:

$$\begin{aligned} &D\Omega[\eta]P^TP + P^TPD\Omega[\eta] \\ &= (\eta^T\zeta - \zeta^T\eta) + (P^TD\zeta[\eta] - D\zeta[\eta]^TP) \\ &- \Omega(\eta^TP + P^T\eta) - (\eta^TP + P^T\eta)\Omega \ . \end{aligned} \tag{6.31}$$

The directional derivative $D\lambda_i[\eta]$, where $i = 1, \ldots, m$, is obtained by taking the directional derivative of $\lambda_i$ in the $\eta$ direction. Assuming $A_iA_j = \mathbf{0}_{n\times n}$ holds, then $\lambda_i$ is given by (recall Equation (6.17)):

$$\lambda_i = \frac{\text{Trace}(\zeta^TA_iP)}{\text{Trace}(P^TA_i^2P)} \ .$$

Taking the directional derivative gives [1]:

$$\begin{aligned} D\lambda_i[\eta] = &\frac{1}{\text{Trace}(P^TA_i^2P)}\text{Trace}(D\zeta[\eta]^TA_iP + \zeta^TA_i\eta) \\ &- \frac{\text{Trace}(\zeta^TA_iP)}{\text{Trace}(P^TA_i^2P)^2}\text{Trace}(\eta^TA_i^2P + P^TA_i^2\eta) \ . \end{aligned} \tag{6.32}$$

Consider writing Equation (6.32) out in matrix form:

$$\begin{aligned} \begin{pmatrix} D\lambda_1[\eta] \\ \vdots \\ D\lambda_m[\eta] \end{pmatrix} &= \text{Diag}\begin{pmatrix} \frac{1}{\text{Trace}(P^TA_1^2P)} \\ \vdots \\ \frac{1}{\text{Trace}(P^TA_m^2P)} \end{pmatrix}\begin{pmatrix} \text{Trace}(D\zeta[\eta]^TA_1P + \zeta^TA_1\eta) \\ \vdots \\ \text{Trace}(D\zeta[\eta]^TA_mP + \zeta^TA_m\eta) \end{pmatrix} \\ &- \text{Diag}\begin{pmatrix} \frac{1}{\text{Trace}(P^TA_1^2P)} \\ \vdots \\ \frac{1}{\text{Trace}(P^TA_1^2P)} \end{pmatrix}\text{Diag}\begin{pmatrix} \text{Trace}(\eta^TA_1^2P + P^TA_1^2\eta) \\ \vdots \\ \text{Trace}(\eta^TA_m^2P + P^TA_m^2\eta) \end{pmatrix}\text{Diag}\begin{pmatrix} \frac{1}{\text{Trace}(P^TA_1^2P)} \\ \vdots \\ \frac{1}{\text{Trace}(P^TA_m^2P)} \end{pmatrix}\tau_{grad} \end{aligned} \tag{6.33}$$

where

$$\text{Diag}\begin{pmatrix} \frac{1}{\text{Trace}(P^TA_1^2P)} \\ \vdots \\ \frac{1}{\text{Trace}(P^TA_m^2P)} \end{pmatrix} = \begin{pmatrix} \text{Trace}(P^TA_1^2P) & & \\ & \ddots & \\ & & \text{Trace}(P^TA_m^2P) \end{pmatrix}^{-1}$$

[1] This equation corrects the error of the corresponding equation in [34]

is the inverse of the matrix $Q$ already discussed in Equation (6.19) and Equation (6.20). The matrix:

$$\text{Diag}\begin{pmatrix}\text{Trace}(\eta^T A_1^2 P + P^T A_1^2 \eta)\\ \vdots \\ \text{Trace}(\eta^T A_m^2 P + P^T A_m^2 \eta)\end{pmatrix} = \begin{pmatrix}\text{Trace}(\eta^T A_1^2 P + P^T A_1^2 \eta) & & \\ & \ddots & \\ & & \text{Trace}(\eta^T A_m^2 P + P^T A_m^2 \eta)\end{pmatrix}$$

has diagonal entries that are given by taking the directional derivative of the diagonal entries of the matrix $Q$ in the $\eta$ directon. Therefore, this matrix can be denoted as $DQ[\eta]$:

$$DQ[\eta] = \begin{pmatrix}\text{Trace}(\eta^T A_1^2 P + P^T A_1^2 \eta) & & \\ & \ddots & \\ & & \text{Trace}(\eta^T A_m^2 P + P^T A_m^2 \eta)\end{pmatrix} \tag{6.34}$$

Similar to $Q$, $DQ[\eta]$ is a diagonal matrix only if the assumption $A_i A_j = \mathbf{0}_{n\times n}$ when $i \neq j$ holds, if this does not hold $DQ[\eta]$ will have off-diagonal terms:

$$\begin{aligned}
&DQ[\eta] \\
&= \begin{pmatrix}\text{Trace}(\eta^T A_1 A_1 P + P^T A_1 A_1 \eta) & \dots & \text{Trace}(\eta^T A_1 A_m P + P^T A_1 A_m \eta)\\ \vdots & & \vdots \\ \text{Trace}(\eta^T A_m A_1 P + P^T A_m A_1 \eta) & \dots & \text{Trace}(\eta^T A_m A_m P + P^T A_m A_m \eta)\end{pmatrix} \\
&= \begin{pmatrix}\varepsilon_{11}\left[{\eta_{a_1b_1}}^T p_{a_1b_1} + {p_{a_1b_1}}^T \eta_{a_1b_1}\right] & \dots & \varepsilon_{1m}\left[{\eta_{a_1b_1}}^T p_{a_mb_m} + {p_{a_1b_1}}^T \eta_{a_mb_m}\right]\\ \vdots & & \vdots \\ \varepsilon_{m1}\left[{\eta_{a_mb_m}}^T p_{a_1b_1} + {p_{a_mb_m}}^T \eta_{a_1b_1}\right] & \dots & \varepsilon_{mm}\left[{\eta_{a_mb_m}}^T p_{a_mb_m} + {p_{a_mb_m}}^T \eta_{a_mb_m}\right]\end{pmatrix}, \\
&= 2\begin{pmatrix}\varepsilon_{11}\left[{\eta_{a_1b_1}}^T p_{a_1b_1}\right] & \dots & \varepsilon_{1m}\left[{\eta_{a_1b_1}}^T p_{a_mb_m}\right]\\ \vdots & & \vdots \\ \varepsilon_{m1}\left[{\eta_{a_mb_m}}^T p_{a_1b_1}\right] & \dots & \varepsilon_{mm}\left[{\eta_{a_mb_m}}^T p_{a_mb_m}\right]\end{pmatrix}
\end{aligned} \tag{6.35}$$

where ${\eta_{a_ib_i}}^T = (e_{a_i} - e_{b_i})^T \eta$ and $\eta_{a_ib_i}^T p_{a_ib_i} = {p_{a_ib_i}}^T \eta_{a_ib_i}$.

Finally, the vector $\tau_{grad}$ for projecting the gradient $\zeta = \nabla f(P)$ is given in Equation (6.27) and reproduced below:

$$\tau_{grad} = \begin{pmatrix}\text{Trace}(\zeta^T A_1 P)\\ \vdots \\ \text{Trace}(\zeta^T A_m P)\end{pmatrix},$$

Define $D\tau_{grad}[\eta]$ to be the $m \times 1$ vector:

$$\begin{aligned} D\tau_{grad}[\eta] &= \begin{pmatrix} \text{Trace}(D\zeta[\eta]^T A_1 P + \zeta^T A_1 \eta) \\ \vdots \\ \text{Trace}(D\zeta[\eta]^T A_m P + \zeta^T A_m \eta) \end{pmatrix} \\ &= \begin{pmatrix} D\zeta[\eta]_{a_1 b_1}{}^T p_{a_1 b_1} + \zeta_{a_1 b_1}{}^T \eta_{a_1 b_1} \\ \vdots \\ D\zeta[\eta]_{a_m b_m}{}^T p_{a_m b_m} + \zeta_{a_m b_m}{}^T \eta_{a_m b_m} \end{pmatrix} \end{aligned} \tag{6.36}$$

where $\zeta_{a_i b_i}{}^T = (e_{a_i} - e_{b_i})^T \zeta$, and $D\zeta[\eta]_{a_i b_i}{}^T = (e_{a_i} - e_{b_i})^T D\zeta[\eta]$. Note that $D\tau_{grad}[\eta]$ is found from taking the directional derivative of the entries of $\tau_{grad}$, in the direction $\eta$.

Let $D\lambda[\eta]$ be the $m \times 1$ vector:

$$D\lambda[\eta] = \begin{pmatrix} D\lambda_1[\eta] \\ \vdots \\ D\lambda_m[\eta] \end{pmatrix} . \tag{6.37}$$

Using the matrices defined above, $D\lambda[\eta]$ solves the following system:

$$D\lambda[\eta] = Q^{-1} D\tau_{grad}[\eta] - Q^{-1} DQ[\eta] Q^{-1} \tau_{grad} . \tag{6.38}$$

which is a generalization of Equation (6.33) to the case where $A_i A_j = \mathbf{0}_{n \times n}$ does not hold when $i \neq j$.

Equivalently, noting that $\lambda = Q^{-1}\tau_{grad}$, $D\lambda[\eta]$ solves the linear system:

$$QD\lambda[\eta] = D\tau_{grad}[\eta] - DQ[\eta]\lambda . \tag{6.39}$$

**Proposition 6.3.2.** *The matrix $DQ[\eta]$ defined in Equation (6.35) is a sparse $m \times m$ matrix, with only $O(m)$ nonzero entries. The nonzero entries of $DQ[\eta]$ are the nonzero entries of $E_b^T E_b$.*

*Proof.* The proof is very similar to the argument provided for Proposition 6.3.1. □

The matrix $DQ[\eta]$ can be defined using the following matrix expression:

$$DQ[\eta] = 2(E_b^T E_b) \circ (E_b^T \eta P^T E_b) .$$

Since $DQ[\eta]$ has only $O(m)$ nonzero entries, only those entries need to be calculated and stored in a sparse matrix.

The vector $D\tau_{grad}[\eta]$ can be defined using the following matrix expression:

$$D\tau_{grad}[\eta] = \text{sumrow}((E_b^T D\zeta[\eta]) \circ (E_b^T P)) + \text{sumrow}((E_b^T \zeta) \circ (E_b^T \eta)) \ . \tag{6.40}$$

### 6.3.5 The Retraction

The retraction of a tangent vector $\eta \in T_P\mathcal{C}(\mathbf{S}_+^{n,3})$ to the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$, is the mapping

$$\mathcal{R}_P : T_P\mathcal{C}(\mathbf{S}_+^{n,3}) \to \mathcal{C}(\mathbf{S}_+^{n,3})$$

given by [1] :

$$\mathcal{R}_P(\eta) = P + \eta + \sum_{i=1}^{m} \lambda_i A_i (P + \eta) \ , \tag{6.41}$$

where $\eta$ is a matrix whose rows are the unconstrained atomic displacements. The retraction is a projection in the *normal direction* onto the constraint manifold (see definition of the normal space, $N_P\mathcal{C}(\mathbb{R}_*^{n\times 3})$, in Equation (6.7)), after an unconstrained step $P+\eta$. Therefore, the retraction is in agreement with the *step and project* paradigm.

Recall the discussion in Section 5.2.2 and Theorem 5.4.1 that the constraint force serves two roles. In the retraction, the constraint force serves the role of projecting *from* the tangent space onto the constraint manifold. In the horizontal projection, see Section 6.3.2, the constraint force projects from $\mathbb{R}^{n\times 3}$ *onto* the tangent space.

#### The Full Retraction

The Lagrange multipliers, $\lambda_i$ $i = 1, \ldots, m$, for the full retraction simultaneously satisfy $m$ equations, the $h$-th equation, $h = 1, \ldots, m$ is given by:

$$\text{Trace}(\mathcal{R}_P(\eta)^T A_h \mathcal{R}_P(\eta)) = q_{a_h b_h} \ , \tag{6.42}$$

Let $P^{(0)} = P + \eta$, then:

$$\text{Trace}((P^{(0)} + \sum_{i=1}^{m} \lambda_i A_i P^{(0)})^T A_h (P^{(0)} + \sum_{j=1}^{m} \lambda_j A_j P^{(0)})) = q_{a_h b_h} \ . \tag{6.43}$$

The 0 superscript will become clear in Section 6.3.5. Using the assumption $A_i A_j = \mathbf{0}_{n\times n}$ when $i \neq j$ from Equation (6.1), Equation (6.43) expands to:

$$\lambda_h^2 \text{Trace}({P^{(0)}}^T A_h^3 P^{(0)}) + 2\lambda_h \text{Trace}({P^{(0)}}^T A_h^2 P^{(0)}) + \text{Trace}({P^{(0)}}^T A_h P^{(0)}) - q_{a_h b_h} = 0 \ .$$

Define the function $\Phi_h : \mathbb{R} \to \mathbb{R}$:

$$\Phi_h(x) = x^2 \text{Trace}({P^{(0)}}^T A_h^3 P^{(0)}) + 2x \text{Trace}({P^{(0)}}^T A_h^2 P^{(0)}) + \text{Trace}({P^{(0)}}^T A_h P^{(0)}) - q_{a_h b_h} \tag{6.44}$$

Solving for $\lambda_h$ in Equation (6.43) is equivalent to finding the root of $\Phi_h(x)$:

$$\Phi_h(\lambda_h) = 0 \ .$$

If $A_i A_j \neq \mathbf{0}_{n\times n}$ when $i \neq j$, Equation (6.43) generalizes to:

$$\begin{aligned} &\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j \text{Trace}({P^{(0)}}^T A_i A_h A_j P^{(0)}) \\ &+2\sum_{i=1}^{m} \lambda_i \text{Trace}({P^{(0)}}^T A_h A_i P^{(0)}) \\ &+\text{Trace}({P^{(0)}}^T A_h P^{(0)}) - q_{a_h b_h} = 0 \ . \end{aligned}$$

Redefine the function $\Phi_h : \mathbb{R}^m \to \mathbb{R}$:

$$\begin{aligned} \Phi_h(\overrightarrow{x}) &= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} x_i x_j \text{Trace}({P^{(0)}}^T A_i A_h A_j P^{(0)}) \\ &+ \sum_{i=1}^{m} x_i \text{Trace}({P^{(0)}}^T A_h A_i P^{(0)}) \\ &+ \frac{1}{2}(\text{Trace}({P^{(0)}}^T A_h P^{(0)}) - q_{a_h b_h}) \\ &= \overrightarrow{x} T^{(0)} \overrightarrow{x} \\ &+ 2\left(\text{Trace}({P^{(0)}}^T A_h A_1 P^{(0)}) \quad \ldots \quad \text{Trace}({P^{(0)}}^T A_h A_m P^{(0)})\right) \overrightarrow{x} \\ &+ \frac{1}{2}(\text{Trace}({P^{(0)}}^T A_h P^{(0)}) - q_{a_h b_h}) \ . \end{aligned} \tag{6.45}$$

where $T_h^{(0)}$, where $h = 1, \ldots, m$, is an $m \times m$ matrix whose $ij$-th non-zero entry is given by:

$$T_{h,ij}^{(0)} = \frac{1}{2}\text{Trace}({P^{(0)}}^T A_i A_h A_j P^{(0)}) = \frac{1}{2}\varepsilon_{ihj}(p_{a_i b_i}^{(0)})^T p_{a_j b_j}^{(0)} \ , \tag{6.46}$$

where:

$$\varepsilon_{ihj} = (e_{a_i} - e_{b_i})^T (e_{a_h} - e_{b_h})(e_{a_h} - e_{b_h})^T (e_{a_j} - e_{b_j}) \ . \tag{6.47}$$

and $(p_{a_i b_i}^{(0)})^T = (e_{a_i} - e_{b_i})^T P^{(0)}$. Then, $\lambda = (\lambda_1, \ldots, \lambda_m)^T \in \mathbb{R}^m$ satisfies:

$$\Phi_h(\lambda) = 0 \text{ for } h = 1, \ldots, m \ . \tag{6.48}$$

Define the function $\Phi(\overrightarrow{x})$, $\Phi : \mathbb{R}^m \to \mathbb{R}^m$:

$$\Phi(\overrightarrow{x}) = \begin{pmatrix} \Phi_1(\overrightarrow{x}) \\ \vdots \\ \Phi_m(\overrightarrow{x}) \end{pmatrix} = \begin{pmatrix} \overrightarrow{x}^T T_1^{(0)} \overrightarrow{x} \\ \vdots \\ \overrightarrow{x}^T T_m^{(0)} \overrightarrow{x} \end{pmatrix} + Q^{(0)} \overrightarrow{x} + C(P^{(0)}) \ . \tag{6.49}$$

The $m \times m$ matrix $Q^{(0)}$ is given by:

$$\begin{aligned} Q^{(0)} &= \begin{pmatrix} \text{Trace}({P^{(0)}}^T A_1 A_1 P^{(0)}) & \ldots & \text{Trace}({P^{(0)}}^T A_1 A_m P^{(0)}) \\ & \ddots & \\ \text{Trace}({P^{(0)}}^T A_m A_1 P^{(0)}) & \ldots & \text{Trace}({P^{(0)}}^T A_m A_m P^{(0)}) \end{pmatrix} \\ &= \begin{pmatrix} \varepsilon_{11}(p_{a_1 b_1}^{(0)})^T (p_{a_1 b_1}^{(0)}) & \ldots & \varepsilon_{1m}(p_{a_1 b_1}^{(0)})^T (p_{a_m b_m}^{(0)}) \\ & \vdots & \\ \varepsilon_{m1}(p_{a_m b_m}^{(0)})^T (p_{a_1 b_1}^{(0)}) & & \varepsilon_{mm}(p_{a_m b_m}^{(0)})^T (p_{a_m b_m}^{(0)}) \end{pmatrix} \ . \end{aligned} \tag{6.50}$$

The $m \times 1$ vector $C(P^{(0)})$ is given by (see also Equation (5.12)):

$$C(P^{(0)}) = \begin{pmatrix} C_1(P^{(0)}) \\ \vdots \\ C_m(P^{(0)}) \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(\text{Trace}({P^{(0)}}^T A_1 P^{(0)}) - q_{a_1 b_1}) \\ \vdots \\ \frac{1}{2}(\text{Trace}({P^{(0)}}^T A_m P^{(0)}) - q_{a_m b_m}) \end{pmatrix} \ .$$

Then Equation (6.48) is equivalent to:

$$\Phi(\lambda) = \mathbf{0}_m \ . \tag{6.51}$$

Since the required $\lambda$ is a root of $\Phi(\overrightarrow{x})$, it can be found using Newton's method iteratively:

$$\lambda^{(k+1)} = \lambda^{(k)} - \nabla\Phi(\lambda^{(k)})^{-1}\Phi(\lambda^{(k)}) \,, \tag{6.52}$$

where $\lambda^{(k)}$ is the $\lambda$ vector from the $k$-th Newton iteration. The $m \times m$ gradient matrix $\nabla\Phi(\overrightarrow{x})$ is defined by:

$$\nabla\Phi(\overrightarrow{x}) = \begin{pmatrix} \frac{\partial h_1(\overrightarrow{x})}{\partial x_1} & \cdots & \frac{\partial h_1(\overrightarrow{x})}{\partial x_m} \\ \vdots & & \\ \frac{\partial h_m(\overrightarrow{x})}{\partial x_1} & \cdots & \frac{\partial h_m(\overrightarrow{x})}{\partial x_m} \end{pmatrix} \,. \tag{6.53}$$

Differentiating $\Phi(\overrightarrow{x})$, gives the following gradient matrix:

$$\nabla\Phi(\overrightarrow{x}) = 2 \begin{pmatrix} \overrightarrow{x}^T T_1^{(0)} \\ \vdots \\ \overrightarrow{x}^T T_m^{(0)} \end{pmatrix} + Q^{(0)} \,. \tag{6.54}$$

The Newton iteration for finding $\lambda$ is summarized in Algorithm 4.

**Algorithm 4** Full Retraction using Newton's Algorithm

**INPUT:** The unconstrained step $P^{(0)}$, an initial iterate $\lambda^{(0)}$, a small number $\epsilon$,.
**OUTPUT:** A point $P^{(k)}$ on the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$.

1: **for** $k = 1, 2, \ldots$ **do**
2: $\quad\lambda^{(k+1)} = \lambda^{(k)} - \nabla\Phi(\lambda^{(k)})^{-1}\Phi(\lambda^{(k)})$
3: $\quad$**if** $\| \lambda^{(k+1)} - \lambda^{(k)} \| < \epsilon$ **then**
4: $\quad\quad$Exit loop.
5: $\quad$**end if**
6: **end for**
7: Return $P^{(k+1)} = P^{(0)} + \sum_{i=1}^{m} \lambda_i^{(k+1)} A_i P^{(0)}$.

### The Fast Retraction

The matrix equation $\Phi(\overrightarrow{x})$ for the Full Retraction, given in Equation (6.49), can take up to $O(m^2)$ computations to evaluate. This bottleneck is caused by the matrix:

$$\begin{pmatrix} \overrightarrow{x}^T T_1^{(0)} \overrightarrow{x} \\ \vdots \\ \overrightarrow{x}^T T_m^{(0)} \overrightarrow{x} \end{pmatrix} ,$$

see Lemma 6.3.1 and Proposition 6.3.3. This means the Full Retraction is very inefficient to use for large $m$. It is not uncommon for a protein structure to have near 1000 residues. Thus, even if only the $\alpha$-carbons of each residue are modelled, as is the case for ENMs, $m$ will often be near 1000. One iteration of the Full Retraction is simply too expensive to evaluate. This section introduces the Fast Retraction, it is so named for two reasons:

- Fast Retraction avoids evaluating the matrix:

$$\begin{pmatrix} \overrightarrow{x}^T T_1^{(0)} \overrightarrow{x} \\ \vdots \\ \overrightarrow{x}^T T_m^{(0)} \overrightarrow{x} \end{pmatrix} ,$$

- this name draws attention to the close connection it has with Fast Projection, see Section 2.8.2.

Observe the quadratic term in $\Phi_h(\overrightarrow{x})$ in Equation (6.45) will be very small if $\| \overrightarrow{x} \|$ is very small, and therefore can be approximated by its linear approximation.

Let $\overrightarrow{\epsilon} \in \mathbb{R}^m$ be a vector where $\| \overrightarrow{\epsilon} \|$ is very close to zero. Define the following linear approximation of $\Phi_h(\overrightarrow{\epsilon})$, denoted by $\widehat{\Phi}_h(\overrightarrow{\epsilon})$:

$$\Phi_h(\overrightarrow{\epsilon}) \approx \widehat{\Phi}_h(\overrightarrow{\epsilon}) = \sum_{i=1}^{m} \epsilon_i \text{Trace}({P^{(0)}}^T A_h A_i P^{(0)}) + C_h(P^{(0)}) \ . \tag{6.55}$$

Define the matrix function:

$$\widehat{\Phi}(\overrightarrow{\epsilon}) = \begin{pmatrix} \widehat{\Phi}_1(\overrightarrow{\epsilon}) \\ \vdots \\ \widehat{\Phi}_m(\overrightarrow{\epsilon}) \end{pmatrix} = Q^{(0)} \overrightarrow{\epsilon} + C(P^{(0)}) \ . \tag{6.56}$$

A vector of small Lagrange multipliers, $\delta\lambda = (\delta\lambda_1, \ldots, \delta\lambda_m) \in \mathbb{R}^m$, that satisfy:

$$\widehat{\Phi}(\delta\lambda) = \mathbf{0}_m \ . \tag{6.57}$$

gives a small retraction called the "Fast Retraction". This means that $\delta\lambda$ is the solution to the linear system:

$$Q^{(0)}\overrightarrow{\epsilon} = -C(P^{(0)}) \ . \tag{6.58}$$

Because the Fast Retraction takes a very small step, the retracted point is likely not on the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$ yet. Let $P^{(0)}$ be the first unconstrained step. A sequence of Fast Retractions are needed, $P^{(1)}, P^{(2)}, \ldots$ to incrementally approaching the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$. At the $k$-th iteration of Fast Retraction, $\delta\lambda^{(k)}$ is found by solving the linear system:,

$$Q^{(k)}\overrightarrow{\epsilon} = -C(P^{(k)}) \ . \tag{6.59}$$

giving a corresponding $\delta P^{(k)}$:

$$\delta P^{(k)} = \sum_{i=1}^{m} \delta\lambda_i^{(k)} A_i P^{(k)} \ .$$

followed by the update:

$$P^{(k+1)} = P^{(k)} + \delta P^{(k)} = P^{(k)} + \sum_{i=1}^{m} \delta\lambda_i^{(k)} A_i P^{(k)} \ , \tag{6.60}$$

to arrive at $P^{(k+1)}$ which satisfies the constraints a bit better than $P^{(k)}$. The Lagrange multipliers $\delta\lambda^{(k)}$ can be scaled by a parameter $\alpha$, that the programmer determines, for example $\alpha = 0.1$, to ensure $\| \ \delta\lambda^{(k)} \ \|$ is small, which is the assumption required by Fast Retraction. These iterations can stop when the constraints are satisfied to some tolerance level. Fast Retraction is summarized in Algorithm 5.

The Fast Retraction update at the $k$-th iteration when integrating the equations of motion has the constraint force scaled by the matrix $(\Delta t)^2\widehat{M}^{-1}$ (which scales the constraint force into a displacement):

$$P^{(k+1)} = P^{(k)} + (\Delta t)^2\widehat{M}^{-1} \sum_{i=1}^{m} \delta\lambda_i^{(k)} A_i P^{(k)} \ . \tag{6.61}$$

Substituting Equation (6.61) into the constraint gives:

$$\begin{aligned}&\text{Trace}({P^{(k+1)}}^T A_k P^{(k+1)})\\ \approx&(\Delta t)^2 \sum_{i=1}^{m} \lambda_i^{(k)} \text{Trace}({P^{(k)}}^T A_k \widehat{M}^{-1} A_i P^{(k)}) + \text{Trace}({P^{(k)}}^T A_k P^{(k)}) - q_{a_k b_k}\end{aligned} \tag{6.62}$$

Putting all $m$ constraints together shows $\delta\lambda$ the following linear system :

$$\widehat{Q}^{(k)} \overrightarrow{\epsilon} = -C(P^{(k)}) \,, \tag{6.63}$$

where

$$\widehat{Q}^{(k)} = (\Delta t)^2 \begin{pmatrix} \text{Trace}({P^{(k)}}^T A_1 \widehat{M}^{-1} A_1 P^{(k)}) & \dots & \text{Trace}({P^{(k)}}^T A_1 \widehat{M}^{-1} A_m P^{(k)}) \\ \vdots & \ddots & \vdots \\ \text{Trace}({P^{(k)}}^T A_m \widehat{M}^{-1} A_1 P^{(k)}) & \dots & \text{Trace}({P^{(k)}}^T A_m \widehat{M}^{-1} A_m P^{(k)}) \end{pmatrix} \,,$$

which differs from Equation (6.59) in the use of $\widehat{Q}^{(k)}$ instead of $Q^{(k)}$.

In the lattice interpolation example discussed in Section 7.2, numerous calls to the retraction function are needed by the RTR algorithm for each conformation. If the Full Retraction is used, the maximum number of iterations for each conformation ranges from 3 to 9. But if the Fast Retraction is used, only 1 to 5 iterations were required. Note that the Full Retraction also requires a random initial guess for the Lagrange multipliers, while the Fast Retraction does not.

**Lemma 6.3.1.** *The $m \times 1$ vector:*

$$\begin{pmatrix} \overrightarrow{x}^T T_1^{(0)} \overrightarrow{x} \\ \vdots \\ \overrightarrow{x}^T T_m^{(0)} \overrightarrow{x} \end{pmatrix} \,,$$

*in $\Phi(\overrightarrow{x})$ defined in Equation (6.49) requires $O(m^2)$ steps to setup, where $m$ is the number of constraints.*

*Proof.* The $ij$-th entry of the matrix $T_h^{(0)}$, $T_{h,ij}^{(0)}$ from Equation (6.46) has nonzero entry if and only

$$\varepsilon_{ihj} = (e_{a_i} - e_{b_i})^T (e_{a_h} - e_{b_h})(e_{a_h} - e_{b_h})^T (e_{a_j} - e_{b_j}) \neq 0 \,.$$

Therefore, $T_{h,ij}^{(0)} \neq 0$ if bond $(a_h, b_h)$ is adjacent to *both* bond $(a_i, b_i)$ and $(a_j, b_j)$. This

**Algorithm 5** Fast Retraction

**INPUT:** Initial iterate $P^{(0)}$ resulting from an unconstrained step, see Equation (2.61). A small number $\epsilon$, a small numer $\alpha$.

**OUTPUT:** A sequence of iterates $P^{(1)}, P^{(2)}, \ldots$ approaching the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$.

1: **for** $k = 0, 1, 2, \ldots$ **do**

2: Solve the linear system for $\delta\lambda^{(k)}$.

$$Q^{(k)}\delta\lambda^{(k)} = -C(P^{(k)}) \tag{6.64}$$

3: Set $\delta\lambda^{(k)} = \alpha\delta\lambda^{(k)}$ to ensure $\| \delta\lambda^{(k)} \|$ is small.

4: Set the $k$-th correction $\delta P^{(k)}$ to be:

$$\delta P^{(k)} = \sum_{i=1}^{m} \delta\lambda_i^{(k)} A_i P^{(k)} \ . \tag{6.65}$$

5: Update $P^{(k+1)} = P^{(k)} + \delta P^{(k)}$.

6: **if** $\|C(P^{(k+1)})\| < \epsilon$ **then**

7: Return $P^{(k+1)}$

8: **end if**

9: **end for**

means each row of $T_h^{(0)}$ has very few nonzero elements, hence each row of the multiplication $T_h^{(0)}\overrightarrow{x}$ can be achieved in $O(1)$ computations, allowing $T_h^{(0)}\overrightarrow{x}$ to be constructed in $O(m)$ computations. Since $T_h^{(0)}\overrightarrow{x}$ is an $m \times 1$ vector, the multiplication of $\overrightarrow{x}^T\left(T_h^{(0)}\overrightarrow{x}\right)$ also require $O(m)$ computations. The matrix

$$\begin{pmatrix} \overrightarrow{x}^T T_1^{(0)}\overrightarrow{x} \\ \vdots \\ \overrightarrow{x}^T T_m^{(0)}\overrightarrow{x} \end{pmatrix},$$

repeats this multiplication $m$ times, hence, requires $mO(m) = O(m^2)$ computations to construct. □

**Proposition 6.3.3.** *The Full Retraction system requires $O(m^2)$ computations to evaluate, while the Fast Retraction's linear system requires only $O(m)$ computations to evaluate.*

*Proof.* As Lemma 6.3.1 has shown, the matrix

$$\begin{pmatrix} \overrightarrow{x}^T T_1^{(0)}\overrightarrow{x} \\ \vdots \\ \overrightarrow{x}^T T_m^{(0)}\overrightarrow{x} \end{pmatrix},$$

required to compute $\Phi(\overrightarrow{x})$ requires $O(m^2)$ computations to set up. The matrix $Q^{(0)}$ from Equation (6.50) has the same sparsity structure as the matrix Q of Equation (6.20) which from Proposition 6.3.1 has $O(m)$ entries, and therefore requires $O(m)$ computations to setup. □

## 6.4 Agreements Between $\mathcal{C}(\mathbb{R}^{3n})$ and $\mathcal{C}(\mathbf{S}_+^{n,3})$

This section discusses the agreements between constraint optimization on $\mathcal{C}(\mathbb{R}^{3n})$ and $\mathcal{C}(\mathbf{S}_+^{n,3})$. The superscript $k$ will be dropped where it is convenient to do so to make the expressions easier to read.

**Lemma 6.4.1.** *The matrix:*

$$\nabla C(\overrightarrow{p})M^{-1}\nabla C(\overrightarrow{p})^T,$$

*and the matrix:*

$$\widehat{Q} = \begin{pmatrix} Trace(P^T A_1 \widehat{M}^{-1} A_1 P) & \dots & Trace(P^T A_1 \widehat{M}^{-1} A_m P) \\ \vdots & \ddots & \vdots \\ Trace(P^T A_m \widehat{M}^{-1} A_1 P) & \dots & Trace(P^T A_m \widehat{M}^{-1} A_m P) \end{pmatrix} ,$$

*are the same matrix.*

*Proof.* This requires showing the $ij$-th entry of these two matrices are the same.

From Equation (2.51),

$$\nabla C(\overrightarrow{p}) = \begin{pmatrix} \overrightarrow{p}^T S_{a_1 b_1} \\ \vdots \\ \overrightarrow{p}^T S_{a_m b_m} \end{pmatrix} .$$

The structure of the matrix:

$$\nabla C(\overrightarrow{p}) M^{-1} \nabla C(\overrightarrow{p})^T ,$$

has been discussed in, for example Barth et al. [9]:

$$\begin{aligned} \nabla C(\overrightarrow{p}) M^{-1} \nabla C(\overrightarrow{p})^T &= \begin{pmatrix} \overrightarrow{p}^T S_{a_1 b_1} \\ \vdots \\ \overrightarrow{p}^T S_{a_m b_m} \end{pmatrix} M^{-1} \begin{pmatrix} S_{a_1 b_1} \overrightarrow{p} & \dots & S_{a_m b_m} \overrightarrow{p} \end{pmatrix} \\ &= \begin{pmatrix} \overrightarrow{p}^T S_{a_1 b_1} M^{-1} S_{a_1 b_1} \overrightarrow{p} & \dots & \overrightarrow{p}^T S_{a_1 b_1} M^{-1} S_{a_m b_m} \overrightarrow{p} \\ \vdots & \ddots & \vdots \\ \overrightarrow{p}^T S_{a_m b_m} M^{-1} S_{a_1 b_1} \overrightarrow{p} & \dots & \overrightarrow{p}^T S_{a_m b_m} M^{-1} S_{a_m b_m} \overrightarrow{p} \end{pmatrix} \end{aligned}$$

The $ij$-th entry is:

$$\overrightarrow{p}^T S_{a_i b_i} M^{-1} S_{a_j b_j} \overrightarrow{p} .$$

Evaluating this expression gives:

$$\begin{aligned} &\overrightarrow{p}^T S_{a_i b_i} M^{-1} S_{a_j b_j} \overrightarrow{p} \\ &= \overrightarrow{p}^T (e_{a_i, I_3} - e_{b_i, I_3})(e_{a_i, I_3} - e_{b_i, I_3})^T M^{-1} (e_{a_j, I_3} - e_{b_j, I_3})(e_{a_j, I_3} - e_{b_j, I_3})^T \overrightarrow{p} \\ &= (p_{a_i} - p_{b_i})^T (e_{a_i, I_3} - e_{b_i, I_3})^T M^{-1} (e_{a_j, I_3} - e_{b_j, I_3})(p_{a_i} - p_{b_i}) . \end{aligned}$$

For $i = j$, the diagonal entries are given by:

$$\begin{aligned}
&(p_{a_i} - p_{b_i})^T (e_{a_i,I_3} - e_{b_i,I_3})^T M^{-1} (e_{a_i,I_3} - e_{b_i,I_3})(p_{a_i} - p_{b_i}) \\
=&\begin{pmatrix} \frac{1}{m_{a_i}}(p_{a_i} - p_{b_i})^T & \dots & -\frac{1}{m_{b_i}}(p_{a_i} - p_{b_i})^T \end{pmatrix} \begin{pmatrix} (p_{a_i} - p_{b_i}) \\ \vdots \\ -(p_{a_i} - p_{b_i}) \end{pmatrix} \\
=&\frac{1}{m_{a_i}}(p_{a_i} - p_{b_i})^T (p_{a_i} - p_{b_i}) - \frac{1}{m_{b_i}}(p_{a_i} - p_{b_i})^T (p_{a_i} - p_{b_i}) \\
=&\left( \frac{1}{m_{a_i}} + \frac{1}{m_{b_i}} \right) (p_{a_i} - p_{b_i})^T (p_{a_i} - p_{b_i}) \\
=&\left( \frac{1}{m_{a_i}} + \frac{1}{m_{b_i}} \right) q_{a_i b_i}
\end{aligned}$$

For off-diagonal entries, only the blocks that appear in the same position in $(e_{a_i,I_3} - e_{b_i,I_3})$ and $(e_{a_j,I_3} - e_{b_j,I_3})$ will contribute to the result. For example, if $a_i = a_j$:

$$\begin{aligned}
&(p_{a_i} - p_{b_i})^T (e_{a_i,I_3} - e_{b_i,I_3})^T M^{-1} (e_{a_j,I_3} - e_{b_j,I_3})(p_{a_j} - p_{b_j}) \\
=&\begin{pmatrix} \frac{1}{m_{a_i}}(p_{a_i} - p_{b_i})^T & \dots & -\frac{1}{m_{b_i}}(p_{a_i} - p_{b_i})^T \end{pmatrix} \begin{pmatrix} (p_{a_j} - p_{b_j}) \\ \vdots \\ -(p_{a_j} - p_{b_j}) \end{pmatrix} \\
=&\left( \frac{1}{m_{a_i}} \right) (p_{a_i} - p_{b_i})^T (p_{a_j} - p_{b_j}) \ .
\end{aligned}$$

As another example, if $b_i = a_j$,

$$\begin{aligned}
&(p_{a_i} - p_{b_i})^T (e_{a_i,I_3} - e_{b_i,I_3})^T M^{-1} (e_{a_j,I_3} - e_{b_j,I_3})(p_{a_j} - p_{b_j}) \\
=&\begin{pmatrix} \frac{1}{m_{a_i}}(p_{a_i} - p_{b_i})^T & \dots & -\frac{1}{m_{b_i}}(p_{a_i} - p_{b_i})^T \end{pmatrix} \begin{pmatrix} (p_{a_j} - p_{b_j}) \\ \vdots \\ -(p_{a_j} - p_{b_j}) \end{pmatrix} \\
=&\left( \frac{-1}{m_{b_i}} \right) (p_{a_i} - p_{b_i})^T (p_{a_j} - p_{b_j})
\end{aligned}$$

For the matrix $\widehat{Q}$, the $ij$-th entry simplifies as follows:

$$\begin{aligned}
&\text{Trace}(P^T A_i \widehat{M}^{-1} A_j P) \\
&= \text{Trace}(P^T (e_{a_i} - e_{b_i})(e_{a_i} - e_{b_i})^T \widehat{M}^{-1} (e_{a_j} - e_{b_j})(e_{a_j} - e_{b_j})^T P) \\
&= \text{Trace}((e_{a_i} - e_{b_i})^T \widehat{M}^{-1} (e_{a_j} - e_{b_j})(e_{a_j} - e_{b_j})^T P P^T (e_{a_i} - e_{b_i})) \\
&= \text{Trace}((e_{a_i} - e_{b_i})^T \widehat{M}^{-1} (e_{a_j} - e_{b_j}))(p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \\
&= \text{Trace}(\widehat{M}^{-1} (e_{a_j} - e_{b_j})(e_{a_i} - e_{b_i})^T)(p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \ .
\end{aligned}$$

If $i = j$,

$$\begin{aligned}
&\text{Trace}(\widehat{M}^{-1} (e_{a_i} - e_{b_i})(e_{a_i} - e_{b_i})^T)(p_{a_i} - p_{b_i})^T (p_{a_i} - p_{b_i}) \\
=&\text{Trace}(\widehat{M}^{-1} (e_{a_i} - e_{b_i})(e_{a_i} - e_{b_i})^T) q_{a_i b_i} \\
=&\left( \frac{1}{m_{a_i}} + \frac{1}{m_{b_i}} \right) q_{a_i b_i} \ .
\end{aligned}$$

For off-diagonal entries, only the entries that appear in the same position in $(e_{a_j} - e_{b_j})$ and $(e_{a_i} - e_{b_i})$ will contribute to the result. Using the same examples as for $\nabla C(\overrightarrow{p}) M^{-1} \nabla C(\overrightarrow{p})^T$, if $a_i = a_j$:

$$\begin{aligned}
&\text{Trace}(\widehat{M}^{-1} (e_{a_j} - e_{b_j})(e_{a_i} - e_{b_i})^T)(p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \\
&= \text{Trace}(\widehat{M}^{-1} (e_{a_j} - e_{b_j})(e_{a_i} - e_{b_i})^T)(p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \\
&= \left( \frac{1}{m_{a_i}} \right) (p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \ .
\end{aligned}$$

If $b_i = a_j$:

$$\begin{aligned}
&\text{Trace}(\widehat{M}^{-1} (e_{a_j} - e_{b_j})(e_{a_i} - e_{b_i})^T)(p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \\
&= \text{Trace}(\widehat{M}^{-1} (e_{a_j} - e_{b_j})(e_{a_i} - e_{b_i})^T)(p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \\
&= \left( \frac{-1}{m_{b_i}} \right) (p_{a_j} - p_{b_j})^T (p_{a_i} - p_{b_i}) \ .
\end{aligned}$$

Therefore, $\nabla C(\overrightarrow{p}) M^{-1} \nabla C(\overrightarrow{p})^T$ and $\widehat{Q}$ are the same matrix. □

**Proposition 6.4.1.** *The matrix:*

$$\nabla C(\overrightarrow{p})\nabla C(\overrightarrow{p})^T \ ,$$

*and the matrix:*

$$\begin{pmatrix} Trace(P^T A_1 A_1 P) & \dots & Trace(P^T A_1 A_m P) \\ \vdots & \ddots & \vdots \\ Trace(P^T A_m A_1 P) & \dots & Trace(P^T A_m A_m P) \end{pmatrix} \ ,$$

*are the same matrix.*

*Proof.* Apply Lemma 6.4.1 with $M = I_{3n}$ and $\widehat{M} = I_n$. □

**Proposition 6.4.2** (The Agreement between the Lagrange multipliers for projecting onto the Tangents Space of the Constraint Manifold)**.** *The Lagrange multipliers for projecting onto the tangents space of the constraint manifold $\mathcal{C}(\mathbb{R}^{3n})$ given by Equation (5.9) agrees with the Lagrange multipliers for projecting onto the tangents space of the constraint manifold $\mathcal{C}(\mathbf{S}_+^{n,3})$.*

*Proof.* From Equation (5.9) $\lambda$ for the projection in $\overrightarrow{\eta} \in \mathbb{R}^{3n}$ solves:

$$\lambda = (\nabla C(\overrightarrow{p})\nabla C(\overrightarrow{p})^T)^{-1}\nabla C(\overrightarrow{p})\overrightarrow{\eta}$$

From Equation (6.22), the Lagrange multiplier for projecting an $n \times 3$ matrix $\eta$ is

$$\lambda = Q^{-1}\tau \ .$$

where $\tau$ is defined in Equation (6.21).

By Proposition 6.4.1,

$$Q = \nabla C(\overrightarrow{p})\nabla C(\overrightarrow{p})^T \ .$$

To see that $\tau$ and $\nabla C(\overrightarrow{p})\overrightarrow{\eta}$ are the same vectors, consider:

$$\nabla C(\overrightarrow{p})\overrightarrow{\eta}$$

From Equation (2.51),

$$\nabla C(\overrightarrow{p}) = \begin{pmatrix} \overrightarrow{p}^T S_{a_1 b_1} \\ \vdots \\ \overrightarrow{p}^T S_{a_m b_m} \end{pmatrix} \ .$$

Therefore:

$$\nabla C(\overrightarrow{p})\overrightarrow{\eta} = \begin{pmatrix} \overrightarrow{p}^T S_{a_1 b_1} \\ \vdots \\ \overrightarrow{p}^T S_{a_m b_m} \end{pmatrix} \overrightarrow{\eta}$$
$$= \begin{pmatrix} \overrightarrow{p}^T S_{a_1 b_1} \overrightarrow{\eta} \\ \vdots \\ \overrightarrow{p}^T S_{a_m b_m} \overrightarrow{\eta} \end{pmatrix} = \begin{pmatrix} (p_{a_1} - p_{b_1})^T (\eta_{a_1} - \eta_{b_1}) \\ \vdots \\ (p_{a_m} - p_{b_m})^T (\eta_{a_m} - \eta_{b_m}) \end{pmatrix}$$

This is equivalent to Equation (6.21) which defines $\tau$ when projection an $n \times 3$ matrix $\eta$.

Therefore, the Lagrange multipliers $\lambda$ solves the same linear system on both $\mathcal{C}(\mathbb{R}^{3n})$ and $\mathcal{C}(\mathbf{S}_+^{n,3})$. □

**Proposition 6.4.3** (The agreement between the derivative of the Lagrange multipliers)**.** *Let $\overrightarrow{\zeta}, \overrightarrow{\eta} \in \mathbb{R}^{3n}$, and let the projection of $\overrightarrow{\zeta}$ be:*

$$Proj(\overrightarrow{\zeta}) = \overrightarrow{\zeta} - \nabla C(\overrightarrow{p})^T \lambda \, .$$

*Let $D\lambda[\overrightarrow{\eta}]$ be the directional derivative of $\lambda$ in the direction $\overrightarrow{\eta}$. The linear system for solving for $D\lambda[\overrightarrow{\eta}]$ on $\mathcal{C}(\mathbb{R}^{3n})$ is the same linear system used to solve for $D\lambda[\eta]$ on $\mathcal{C}(\mathbf{S}_+^{n,3})$ given in Equation (6.38) and Equation (6.39).*

*Proof.* This proof follows the same approach as the proof for Proposition 6.4.2. Details are omitted. □

**Proposition 6.4.4** (The Agreement Between Fast Projection and Fast Retraction)**.** *The linear system solved by Fast Projection and the linear system solved by Fast Retraction are the same linear systems.*

*Proof.* Fast Projection's $k$ correction is given by:

$$\overrightarrow{p}^{(k+1)} = \overrightarrow{p}^{(k)} - (\Delta t)^2 M^{-1} \nabla C(\overrightarrow{p}^{(k)})^T \delta\lambda \, .$$

Fast Retraction has been defined with a sign difference (see Equation (6.61)):

$$P^{(k+1)} = P^{(k)} + (\Delta t)^2 \widehat{M}^{-1} \sum_{i=1}^{m} \delta\lambda_i^{(k)} A_i P^{(k)} \, .$$

But since $\delta\lambda_i^{(k)}$ is an unknown vector to be found, the sign difference can be moved to be associated this unknown vector, so Fast Retraction can be redefined with a minus sign to be consistent with Fast Projection:

$$P^{(k+1)} = P^{(k)} - (\Delta t)^2 \widehat{M}^{-1} \sum_{i=1}^{m} (-\delta\lambda_i^{(k)}) A_i P^{(k)} \ .$$

Fast Retraction's linear system is now:

$$(\Delta t)^2 \begin{pmatrix} \text{Trace}({P^{(k)}}^T A_1 \widehat{M}^{-1} A_1 P^{(k)}) & \dots & \text{Trace}({P^{(k)}}^T A_1 \widehat{M}^{-1} A_m P^{(k)}) \\ \vdots & \ddots & \vdots \\ \text{Trace}({P^{(k)}}^T A_m \widehat{M}^{-1} A_1 P^{(k)}) & \dots & \text{Trace}({P^{(k)}}^T A_m \widehat{M}^{-1} A_m P^{(k)}) \end{pmatrix} (-\delta\lambda^{(k)}) = -C(P^{(k)}) \ .$$

After cancelling the minus sign on both sides of the equation, Fast Retraction's linear system is then:

$$(\Delta t)^2 \begin{pmatrix} \text{Trace}({P^{(k)}}^T A_1 \widehat{M}^{-1} A_1 P^{(k)}) & \dots & \text{Trace}({P^{(k)}}^T A_1 \widehat{M}^{-1} A_m P^{(k)}) \\ \vdots & \ddots & \vdots \\ \text{Trace}({P^{(k)}}^T A_m \widehat{M}^{-1} A_1 P^{(k)}) & \dots & \text{Trace}({P^{(k)}}^T A_m \widehat{M}^{-1} A_m P^{(k)}) \end{pmatrix} (\delta\lambda^{(k)}) = C(P^{(k)}) \ .$$

By Proposition 6.4.1, the matrix on the left hand side of the linear system for Fast Projection and Fast Retraction are the same.

On the right hand side, Equation 2.49 gives:

$$C_i(\overrightarrow{p}) = \frac{1}{2} \left( \| \ p_{a_i} - p_{b_i} \ \|^2 - q_{a_i b_i} \right) \ ,$$

while Equation 5.13 gives:

$$C_i(P) = \frac{1}{2} \left( \| \ p_{a_i} - p_{b_i} \ \|^2 - q_{a_i b_i} \right) \ .$$

Fast Projection's right hand side and Fast Retraction's right hand side are therefore equivalent:

$$C(\overrightarrow{p}) = C(P) \ .$$

In summary, Fast Projection and Fast Retraction solves the same linear system. □

Although Fast Projection and Fast Retraction both solve the same linear system, they

are arrived at from different approaches. Table 6.1 provides a comparison of these two paradigms.

| **Fast Projection** | **Fast Retraction** |
|---|---|
| Approximates SAP, which solved for both the Lagrange multipliers *and* the atomic positions. | Approximates the full retraction, which was *only* solving for the Lagrange multipliers. |
| The potential energy is a function on $\mathbb{R}^{3n}$ | The potential energy is a function on $\mathbf{S}_{+}^{n,3}$ |

Table 6.1: A comparison of Fast Projection and Fast Retraction.

## 6.5 Summary

This Chapter presented the gradient, Hessian, horizontal projection, and retraction required for optimization on $\mathcal{C}(\mathbf{S}_{+}^{n,3})$. Assumption 2 from [34] was removed to describe the constraints of a protein structure. This Chapter also discussed the agreements between constraint optimization on $\mathcal{C}(\mathbb{R}^{3n})$ and $\mathcal{C}(\mathbf{S}_{+}^{n,3})$.

# Chapter 7

# Interpolating Transitional Conformations: An Example of Modelling ENMs on $\mathbf{S}_+^{n,3}$

## 7.1 Introduction

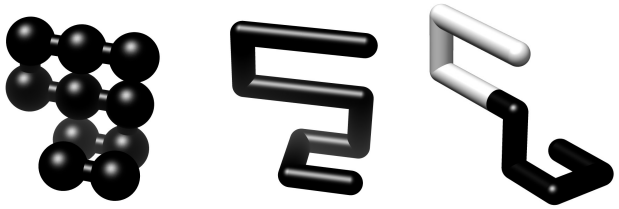
This Chapter uses the example of ENI and iENM to illustrate the modelling of protein dynamics on $\mathbf{S}_+^{n,3}$. A `python` implementation is provided, using an object-oriented paradigm, in Appendix A.

In Section 7.2, a lattice example is used to illustrate interpolating with and without constraints. A comparison of the PSD ENI and PSD iENM potentials in Section 7.3 shows that PSD iENM generates local conformational changes before global changes, as was also seen in the original iENM formulation[74]. This order of change supports the idea that the active site changes first to drive the protein's conformational change. PSD ENI however, does not produce such an order of change.

## 7.2 Lattice Example

In this section, a lattice model will be used to illustrate using the constraint manifold to enforce constraints. Section 8.1 discusses that the PSD collision avoidance energy can be modified to include enforcing equality constraints.

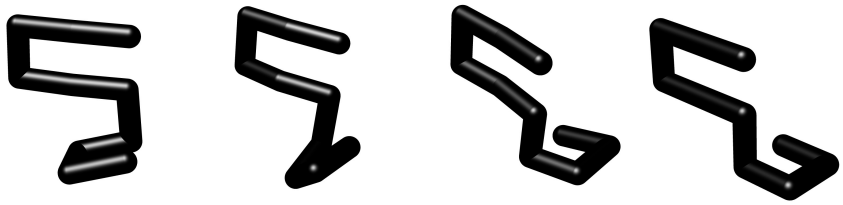
Figure 7.1 presented the lattice model, this lattice model is an $\alpha$-carbon chain. It is called "lattice1_small" because the atoms are placed 1 $\mathring{A}$ apart, a shorter distance than the 3.8 $\mathring{A}$ for a real protein.



(a) (b) (c)

Figure 7.1: The lattice structure "lattice1_small". a) ball and stick model of the starting conformation. b) stick model of the starting conformation. c) stick model of the ending conformation.

Figure 7.1 a) gives the beginning conformation as a "ball and stick" representation, where the balls represent the $\alpha$-carbons. Figure 7.1 b) gives the beginning conformation using a "stick" representation, where the atoms are not visible. The stick representation allows the orientation of the backbone to be shown. Figure 7.1 c) is the ending conformation, where the bottom part of the lattice, coloured in black, has undergone a 180° rotation.



(a) $t = 0.25$ (b) $t = 0.79$ (c) $t = 0.80$ (d) $t = 1.00$

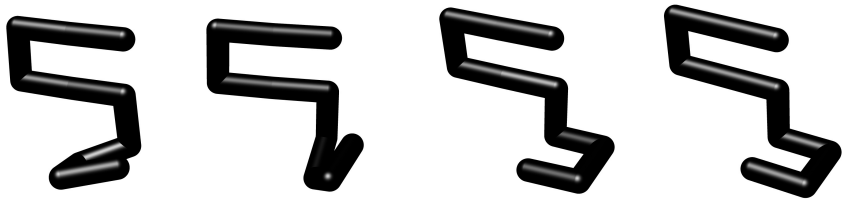
Figure 7.2: Transitions generated by Hookean ENI without the constraint manifold for lattice1_small. The final conformation is in the correct orientation after an abrupt flip at $t = 0.79$.

In order for the ending conformation to be in the correct orientation, the part of the lattice structure coloured in black in Figure 7.1 c) must remain rigid throughout the interpolation. Let the atoms of this structure be indexed starting from the top from 1

to 10, then constraining the bond length between consecutive $\alpha$ carbons, together with constraining the bottom piece colored in black to be rigid requires the following pairs of constraints:

$$(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10),(6,8),(6,9),(6,10),(7,9),(8,10)$$

Figure 7.2 shows the transition generated using Hookean ENI without constraints, but with the steric collision energy. An interesting observation from this interpolation is that at $t = 0.79$, the bottom piece is abruptly flipped to the other side to arrive at the final conformation. Hookean iENM has generated the same transition and therefore images of the intermediate conformations will be omitted.



(a) $t = 0.25$ (b) $t = 0.60$ (c) $t = 0.86$ (d) $t = 1.00$

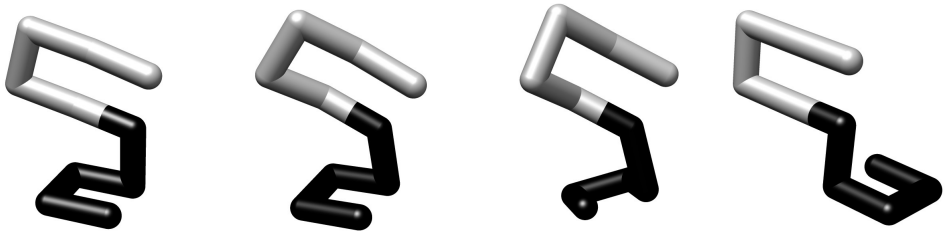
Figure 7.3: Transitions generated by PSD ENI without the constraint manifold for lattice1_small. The final conformation is in the wrong orientation.

Figure 7.3 shows the transition generated by PSD ENI without the constraint manifold. In this case, because the bottom piece has not been constrained to be rigid, the final conformation generated is in the wrong orientation. PSD iENM has generated the same transition and therefore images of the intermediate conformations will be omitted.

Figure 7.4 shows the transitions generated by Hookean ENI from moving on the constraint manifold. The intermediate conformations generated in this case has the bottom piece, which is coloured in black in Figure 7.4, stay rigid for all intermediate conformations, and therefore, the ending conformation is in the correct orientation.

Note that the top part of the structure in Figure 7.4, colored gray, *does not stay rigid*. Recall from the discussion in Section 2.8.3 that when three atoms are collinear, the matrix used by Fast Projection to solve for the Lagrange multipliers becomes singular, therefore the top part of lattice1_small cannot be constrained to be rigid. Since this lattice is not a real protein, we will not explore this issue further in this thesis.

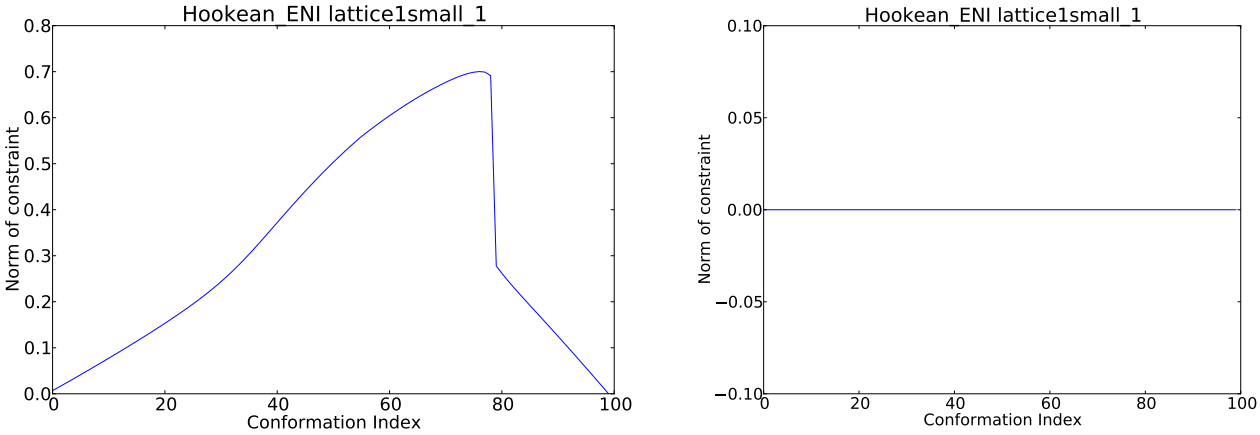
The transitional conformations generated by Hookean iENM are the same as that generated by Hookean ENI, the figures have therefore been omitted. The transitions



(a) $t = 0.25$ (b) $t = 0.50$ (c) $t = 0.65$ (d) $t = 0.99$

Figure 7.4: Transitions generated by Hookean ENI with the constraint manifold for lattice1_small. The black coloured part of the structure is constrained to be rigid to allow it to rotate 180°.

generated by PSD ENI, and PSD iENM also show the same principle, and therefore these images are also omitted. In both cases, the final conformation is in the correct conformation, unlike in Figure 7.3.

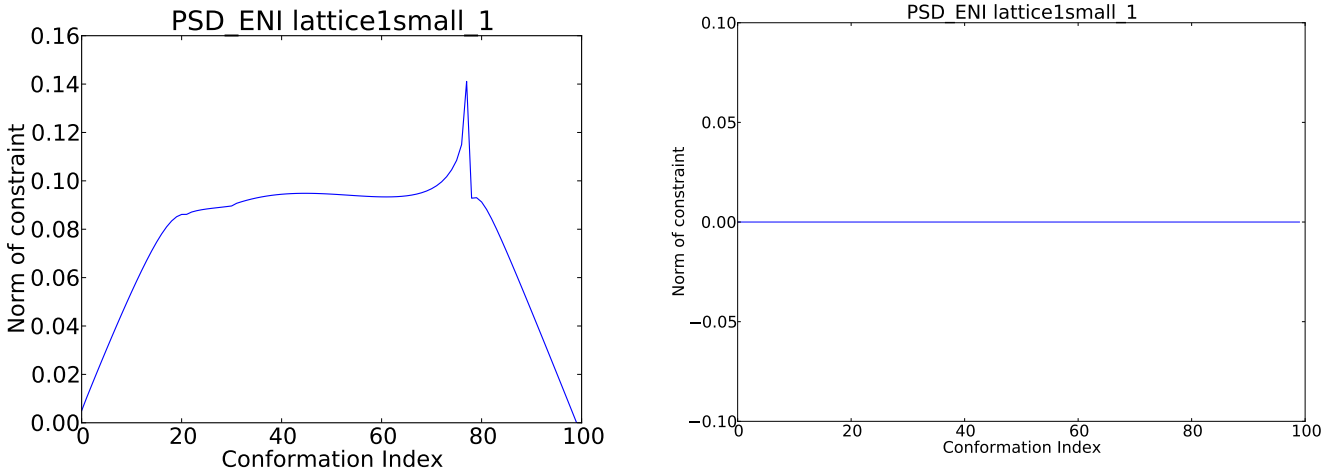



(a) Without constraint manifold. (b) With constraint manifold

Figure 7.5: $\| C(\overrightarrow{p}_t) \|$ for each transitional conformation $\overrightarrow{p}_t$ generated by Hookean ENI. The same graph was generated by Hookean iENM.

Figure 7.5 shows the norm of the constraint, $\| C(\overrightarrow{p}) \|$ for each transitional conformation generated by Hookean ENI (recall Equation (2.47). Hookean iENM produced the same graph and is therefore omitted. Figure 7.5 a) shows that despite the final conformation

being in the correct orientation, the intermediate conformations do not stay on the constraint manifold. Note the peak of the graph around $t = 0.79$ and $t = 0.8$, where the abrupt flip happened as shown in Figure 7.2. Figure 7.5 b) shows after enforcing constraints, $\| C(\overrightarrow{p}) \|$ stays very close to zero for each transitional conformation, with no abrupt changes.




(a) Without constraint manifold.

(b) With constraint manifold

Figure 7.6: $\| C(P_t) \|$ for each transitional conformation $P_t$ generated by PSD ENI. The same graph was generated by PSD iENM.

Figure 7.6 shows $\| C(P_t) \|$ for each $P_t$ generated by PSD ENI. Figure 7.6 a) and b) show respectively that if enforcing constraints is omitted, the intermediate conformations are not guaranteed to be on the constraint manifold.

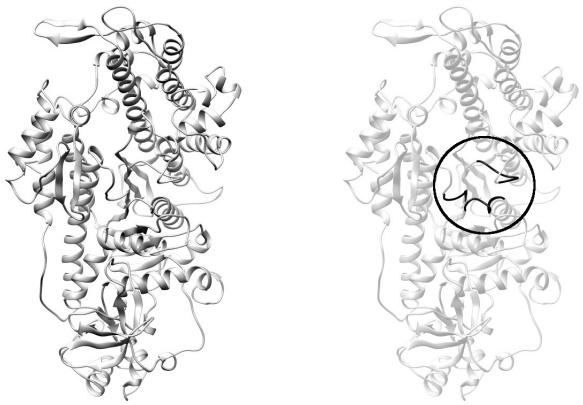
## 7.3 Local and Global Conformational Changes

The protein structure constraints modelled in this section are the quadrances between consecutive $\alpha$-carbons, being fixed at $(3.8)^2\mathring{A}$ apart. Distances between geminal $\alpha$-carbons can also be constrained to keep the bond angles fixed, but is omitted in this example to keep the number of constraints small. The range of motion being interpolated is small, hence this modelling choice does not have an impact on the result.

The Hookean iENM potential produces intermediate structures that model local conformational change at an active site before a global conformational change. The ability of the model to

distinguish local versus global conformational changes supports the idea that local motions at an active site *drive* larger global conformational changes. When the original iENM was introduced [74], three Adenosine-triphosphate (ATP) driven structural transitions were modelled to demonstrate iENM can model local change at an active site before a global change. These three transitions are summarized below, and are used to show PSD iENM also preserves the order of local change before global change.

CASE 1: MYOSIN Myosin is a superfamily of actin-based Adenosine-triphosphate (ATP) powered motor proteins involved in diverse functions from muscle contraction to intracellular transportation. The various biochemical states this protein can have are described in [74], where the transition from the state given by 1FMW to the state given by 1VOM, is modelled by the authors. During this transition, ATP hydrolysis is accompanied by the closing of the active site (local change), and an upward rotation of the converter subdomain (global change). A difference in the order of local and global conformational change has been observed by different modelling approaches. The iENM approach produces the local conformational change before the global change. According to the authors, this is "consistent with a causal relationship by which ATP hydrolysis causes a subsequent upward rotation of the converter, which is energetically more favorable than an alternative order by which an upward rotation of the converter precedes the closing of active site." The active site is located at residues 179-186, 233-238, and 454-459. Figure 7.7 shows the active site residues for 1FMW.



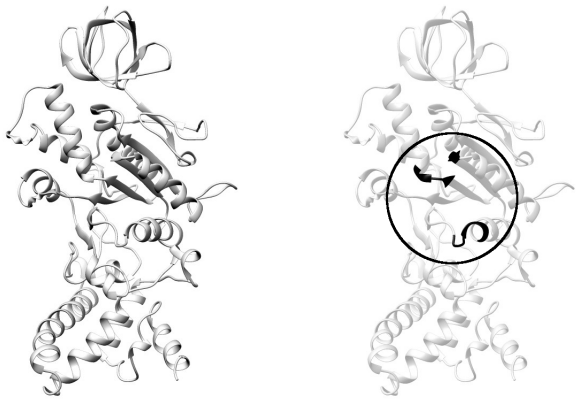
(a) 1FMW (b) Active site circled.

Figure 7.7: 1FMW active site.

CASE 2: 1BMF $F_1$ ATPase ($F_1$) is a globular catalytic moiety of $F_0F_1$ ATP synthase,

which is a molecular machine that uses the energy of proton motive force across the mitochondrial membrane to synthesize ATP. It has three $\beta$ subunits, three $\alpha$ subunits, and a central stalk of $\gamma\delta\epsilon$ subunits. Three catalytic sites and three non-catalytic sites are situated at the interface of the $\beta$ and $\alpha$ subunits. The bovine $F_1$ crystal structure, (PDB code 1BMF), captures the catalytic sites at the open and close conformations. The empty E-site, and the associated $\beta_E$ subunit adopted the open conformation. The other two sites, referred to as the TP-site and DP-site, and their associated $\beta_{TP}$ and $\beta_{DP}$ subunits adopted the closed conformation. The iENM model from the $\beta_E$ open conformation to the $\beta_{TP}$ closed conformation shows closing of the catalytic E-site precedes the open-to-close transition of the $\beta$ subunit. The catalytic site (active site) is located at residues 159-164, 186-188, 256-259, and 308-309. Figure 7.8 shows the active site for 1BMF.

The $\beta_E$ subunit is captured in chain E of the PDB structure, and the $\beta_{TP}$ subunit is captured in chain F of the PDB structure. The conformation given by these two chains are interpolated.

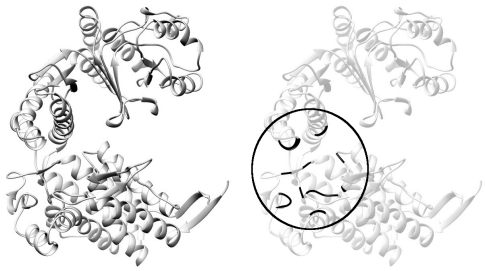


(a) 1BMF chain E. (b) Active site circled.

Figure 7.8: 1BMF (chain E, $\beta_E$ subunit) active site.

CASE 3: 1AON GroEL is a chaperone protein that assists in protein folding. ATP binding leads to the transition from the T state to the R state, then binding of the GroES protein and ATP hydrolysis leads to the R$''$ state. Chain H of 1AON is in the T state conformation, and chain A of 1AON is in the R$''$ state conformation. The iENM transition from the T state to the R$''$ state shows the closing of the active site before a global conformational change, supporting the idea that ATP binding is the cause of the global

structural change. The active site is located at residues 31-33, 87, 91, 150, 151, 398, 415, 454, 479-481, 493, 495. Figure 7.9 shows the active site for 1AON.



(a) 1AON chain H. (b) Active site circled.

Figure 7.9: 1AON (chain H) active site.

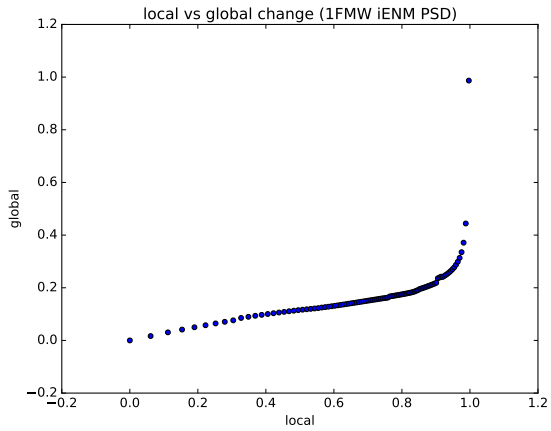
Let $\mathcal{I} \subset \{1, \ldots, n\}$, be a subset of the residue indices for which their progress towards the end conformation is to be measured. Let $P_0$ be the initial conformation, $P_j$ be the $j$-th conformation, and $P_{100}$ be the final conformation. Let $P_0[\mathcal{I}, :]$, $P_j[\mathcal{I}, :]$, $P_{100}[\mathcal{I}, :]$ be the corresponding submatrices with the rows in $\mathcal{I}$ present. The reaction coordinate formula from [74] for measuring the progress of residues indexed by $\mathcal{I}$ towards the ending conformation can be expressed in the following matrix expression:

$$RC^j_{\mathcal{I}} = \frac{\langle P_j[\mathcal{I}, :]Q_j - P_0[\mathcal{I}, :], P_{100}[\mathcal{I}, :]Q_{100} - P_0[\mathcal{I}, :]\rangle}{\langle P_{100}[\mathcal{I}, :]Q_{100} - P_0[\mathcal{I}, :], P_{100}[\mathcal{I}, :]Q_{100} - P_0[\mathcal{I}, :]\rangle} \quad j = 0, \ldots, 100\ . \tag{7.1}$$

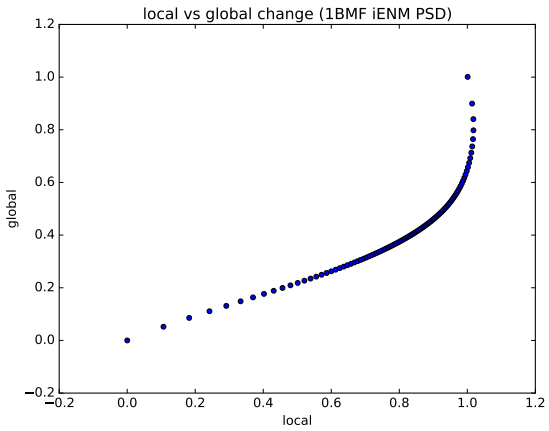
The matrices $Q_{100}$ and $Q_j$ are Procrustes rotations applied to $P_{100}$ and $P_j$ respectively. The matrix $P_{100}[\mathcal{I}, :]Q_{100} - P_0[\mathcal{I}, :]$ measures the total displacement from the beginning conformation to the end conformation. The matrix $P_j[\mathcal{I}, :]Q_j - P_0[\mathcal{I}, :]$ measures the displacement up to the $j$-th conformation. $RC^j_{\mathcal{I}}$ thus measures how much the $j$-th conformation's residues with indices from $\mathcal{I}$ has moved, with respect to how much they need to move in total.

Figure 7.10 and 7.11 both plot $RC^j_{Local}$ on the horizontal axis versus $RC^j_{Global}$ on the vertical axis. The local and global residue indices for each protein were discussed previously in the CASE 1, CASE 2, CASE 3 summaries.

Figure 7.10 shows the local versus global reaction coordinate scatter plot for the transitional conformations generated by PSD iENM for 1FMW's and 1BMF's transitions. For 1FMW, the global change is measured for residues 81 to 747, following [74]. This figure shows




(a) 1FMW to 1VOM, global motion delayed.




(b) 1BMF chain E to F, global motion delayed.

Figure 7.10: PSD iENM delays global motion for 1FMW and 1BMF, in agreement with Hookean iENM.

the global motion is delayed until the very end of the interpolation for both proteins, in agreement with the original Hookean iENM.

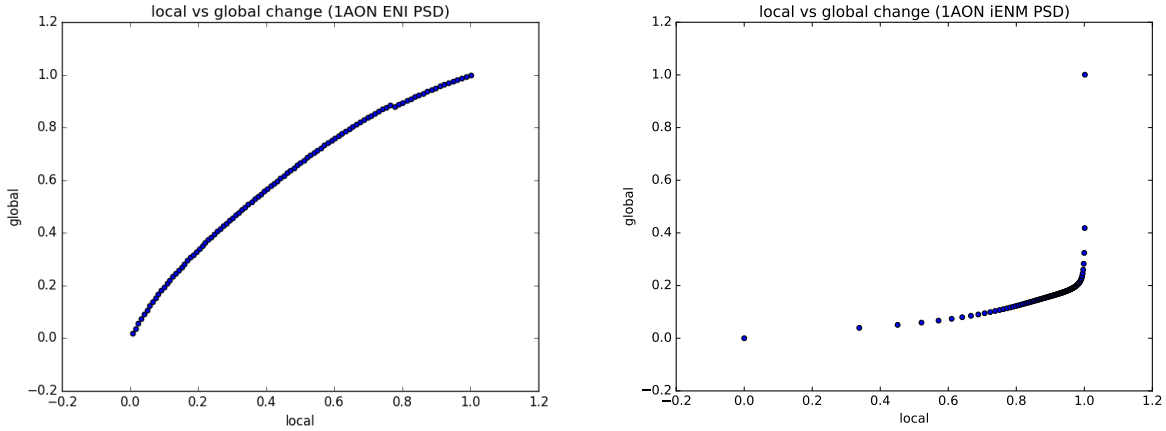
Figure 7.11 shows a comparison of the local versus global change for 1AON generated by PSD ENI and PSD iENM. As can be seen, PSD ENI does not delay the global motion towards the final stages of the interpolation.

In Figure 7.13, intermediate conformations generated by PSD ENI are presented for the 1AON chain H to 1AON chain A interpolation. These conformations show the global movement happens very early. Figure 7.13 is the aligned beginning and ending conformation before and after interpolation.

Figure 7.14 presents the PSD iENM generated interpolation for 1AON chain A to chain H. These intermediate structures show PSD iENM delays the global motion towards the final stages of the interpolation.

## 7.4 Summary

This Chapter used the transitional conformations problem as an example of modelling ENMs on $\mathbf{S}_{+}^{n,3}$. This Chapter has also observed the agreement between the Hookean iENM and PSD iENM potentials.

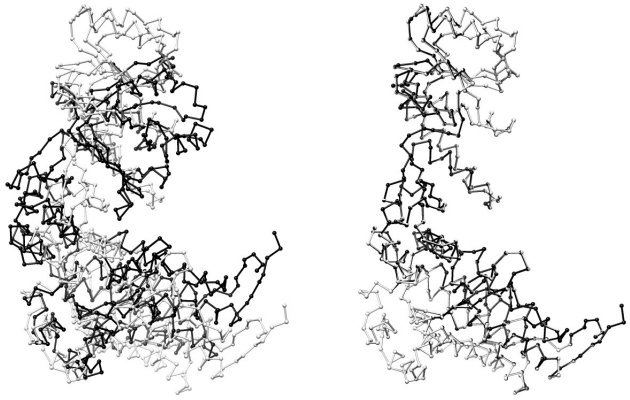



(a) PSD ENI does not delay global motion (b) PSD iENM generates global motion near the end.

Figure 7.11: Local and global motion comparison for 1AON from chain H to chain A.

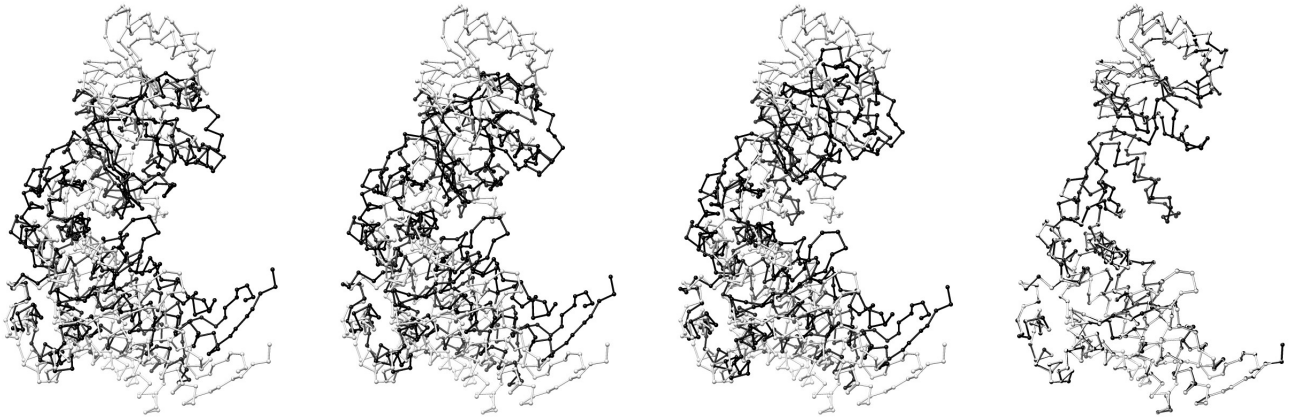
(a) $t = 0.25$ (b) $t = 0.50$ (c) $t = 0.70$

Figure 7.12: Sample of transitional conformations for 1AON Chain H (black) to Chain A with PSD ENI showing global motion already at $t = 0.50$.



(a) $t = 0.00$ (b) $t = 1.00$

Figure 7.13: 1AON Chain H (black) superimposed onto Chain A before and after interpolation, showing the range of global motion.



(a) $t = 0.25$ (b) $t = 0.90$ (c) $t = 0.99$ (d) $t = 1.00$

Figure 7.14: Sample transitional conformations for 1AON Chain H to Chain A with PSD iENM showing global motion delayed to *beyond* the 90th transitional conformation.

# Chapter 8

# Conclusion

The $\mathbb{R}^{3n}$ Riemannian manifold presents many mathematical defects for modelling ENMs. These defects are removed by changing to the $\mathbf{S}_+^{n,3}$ Riemannian manifold. Table 8.1 summarizes defects discussed in this thesis.

## 8.1 Future Research

The Riemannian manifold $\mathbf{S}_+^{n,3}$ opens up many interesting directions of future research.

Meyer[55] generalized linear regression to many matrix manifolds, including $\mathbf{S}_+^{n,r}$. Recall in $\mathbb{R}^{3n}$, linear regression is given by the optimization problem:

$$\min \sum_{i=1}^{N} (\overrightarrow{w}^T \overrightarrow{\eta}_i - \overline{y}_i)^2 \ , \tag{8.1}$$

where $\overrightarrow{\eta}_i \in \mathbb{R}^{3n}$ are the training features, $\overline{y}_i \in \mathbb{R}$ $i = 1, \ldots, N$ are the training (expected) outputs and $\overrightarrow{w} \in \mathbb{R}^{3n}$ are the weights to be learned by the linear regression model. The generalization of linear regression to $\mathbf{S}_+^{n,r}$ provided by Meyer[55] is given by:

$$\min \sum_{i=1}^{N} \parallel \text{Trace}(W^T \text{Sym}(A_i)) - \overline{y}_i \parallel_F^2 \ , \tag{8.2}$$

where $W \in \mathbf{S}_+^{n,r}$ is the low rank weight matrix, and $A_i$ are now matrix inputs, where $\text{sym}(A) = (A + A^T)/2$ ensures $A_i$ is a symmetric matrix, and the expected training output is

| **Considerations** | Defects of the current approach using $\mathbb{R}^{3n}$ | The new approach in this thesis using $\mathbf{S}_+^{n,3} \cap \mathbf{S}_C$ |
|---|---|---|
| Minimizing the potential is well-posed | No, problem is ill-posed. | Yes, problem is well-posed. |
| Distance or quadrance in potential | No preference. | Quadrance is preferred because of the Linear isomorphism mapping $\mathcal{K} : \mathbf{S}_+^n \cap \mathbf{S}_C^n \to \mathcal{E}^n$. |
| Number of constraints for a rigid group of $n_q$ atoms | $3n_q - 6$, see [64] depends on $n_q$, i.e. $O(n_q)$. | Use facial reduction to *further reduce* to $3(4) - 6 = 6$ constraints |
| Rotation of rigid groups of atoms | Needs rotation matrices, and Lie group theory, additional complexity. [17]). | Use a face of the $\mathbf{S}_+^n$ cone. No additional mathematical objects required, model remains simple. |
| Conservation of linear momentum | Additional constraint. | Using facial reduction to remove this constraint. |
| Enforcing constraints | *Both* implicit and explicit treatment of the constraint force has been proposed. | The retraction is by definition a "step and project" operation, implicit treatment of constraint force is the only option. |
| Redundancy | Yes. Mass matrix $M$ repeats mass of each atom three times, $e_{a,I_3}$ repeats the 1 three times. | No. See definition of $\widehat{M}$ and $e_a$. |
| Apply abstract concepts to different applications | No. Fast Projection was already presented by Rosen in 1961 [66] for general nonlinear constraints, but re-introduced by Goldenthal et al. [30] for distance constraints. | Yes. Matrix manifold optimization uses abstract differential geometry as a foundation. |

Table 8.1: The advantages of modelling ENMs using $\mathbf{S}_+^{n,3}$ over $\mathbb{R}^{3n}$ is summarized.

$\overline{y}_i$. Meyer pointed out that the rank $r$ EDMC problem is in fact a *linear regression problem on* $\mathbf{S}_+^{n,r}$, where the specified quadrances, $\overline{q}_{ab}$, are the expected training outputs, $(e_a - e_b)(e_a - e_b)^T$ for $(a, b) \in \mathcal{D}$ are the matrix training inputs, $W = YY^T$ is the rank $r$ *weight matrix*

to be learned. In other words, Meyer [55] has pointed out that the optimization problem:

$$\begin{aligned} &\min \sum_{(a,b)\in\mathcal{D}} (\text{Trace}(YY^T(e_a - e_b)(e_a - e_b)^T) - \overline{q}_{ab})^2 \\ =&\min \sum_{(a,b)\in\mathcal{D}} ((e_a - e_b)^T YY^T(e_a - e_b) - \overline{q}_{ab})^2 \ , \end{aligned} \tag{8.3}$$

is a linear regression problem on $\mathbf{S}_+^{n,r}$.

Given input $\overrightarrow{\eta}_i \in \mathbb{R}^{3n}$, a perceptron in $\mathbb{R}^{3n}$ that does not have an activation function *also* outputs the calculation:

$$\overrightarrow{w}^T \overrightarrow{\eta}_i \ . \tag{8.4}$$

Therefore, this means the output of a perceptron whose weight matrix is on $\mathbf{S}_+^{n,r}$ is:

$$\text{Trace}(W^T \text{Sym}(A_i)) \ . \tag{8.5}$$

The PSD potential uses a perceptron which produces the output

$$\text{Trace}(PP^T(e_a - e_b)(e_a - e_b)^T)$$

which is of the form given in Equation (8.5) with $W = PP^T$. Therefore, the PSD potential can be viewed as the sum of squared errors of a perceptron whose training input is given by the matrices $(e_a - e_b)(e_a - e_b)^T$, and whose training output is the reference quadrances from atomic index pairs in $\mathcal{D}$, from which the rank 3 "weight matrix" $PP^T$ of atomic coordinates is to be learned.

Meyer's generalization motives Definition 8.1.1 of a rank $r$ PSD perceptron.

**Definition 8.1.1** (Rank $r$ PSD Perceptron)**.** *A rank $r$ PSD perceptron is a perceptron whose weight matrix is a rank $r$ PSD matrix. The input to the perceptron is a symmetric matrix. The output of the perceptron is a scalar real number. There may or may not be an activation function applied to the output.*

The PSD collision avoidance potential can be modified to include checking for violations of equality constraints. Define the following sets which are evaluated at $P$. Let $q_{col}$ be the threshold quadrance for which atomic collision is assumed. Define $\mathcal{D}_{col}(P)$ to be the set of atomic pairs for which the inter-atomic quadrance is less than $q_{col}$:

$$\mathcal{D}_{col}(P) = \{(a,b) \mid (e_a - e_b)^T PP^T(e_a - e_b) < q_{col}\} \ .$$

Define $\mathcal{D}_E(P)$ to be the set of pairs that have violated their equality constraint, given by $\widetilde{q}_{ab}$:

$$\mathcal{D}_E(P) = \{(a,b) \mid (e_a - e_b)^T PP^T (e_a - e_b) > \widetilde{q}_{ab} \text{ or } (e_a - e_b)^T PP^T (e_a - e_b) < \widetilde{q}_{ab}\} .$$

These sets can be used to define the activation function of a PSD "penalty perceptron", which now penalizes both collisions and violation of equality constraints.

Now consider the situation on $\mathbb{R}^{3n}$. The Hookean potential has no simple relation to a perceptron whose weight vector is from $\mathbb{R}^{3n}$.

Because the PSD potential uses PSD perceptrons, and minimizing the PSD potential is equivalent to minimizing the sum of squared errors outputted by these PSD perceptrons, this problem can be implemented on hardware optimized for neural network training which are expected to be faster than CPUs This implementation is left as a topic for future research. Other future research directions include adapting the PSD potential to different problems in protein dynamics, for example protein-ligand docking [57, 69], and a further investigation of applying facial reduction to modelling rigid groups of atoms.

# References


[1] P.A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008.

[2] P.A. Absil and J. Malick. Projection-like retractions on matrix manifolds. *SIAM Journal on Optimization*, 22(1):135–158, 2012.

[3] A. Alfakih, A. Khandani, and H. Wolkowicz. Solving Euclidean distance matrix completion problems via semidefinite programming. *Computational Optimization and Applications*, 12(1-3):13–30, January 1999.

[4] B. Alipanahi. *New Approaches to Protein NMR Automation*. PhD thesis, University of Waterloo, 2011.

[5] B. Alipanahi, N. Krislock, A. Ghodsi, H. Wolkowicz, L. Donaldson, and M. Li. Determining protein structures from noesy distance constraints by semidefinite programming. *Journal of Computational Biology*, 20(4):296–310, 2013.

[6] H.C. Andersen. Rattle: A velocity version of the shake algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52(1):24–34, 1983.

[7] V.I. Arnold. *Mathematical Methods of Classical Mechanics, Second Edition*. Springer-Verlag, 1989.

[8] R.H. Bartels and G.W. Stewart. Solution of the matrix equation AX + XB = C [f4]. *Commun. ACM*, 15(9):820–826, September 1972.

[9] E. Barth, K. Kuczera, B. Leimkuhler, and R.D. Skeel. Algorithms for constrained molecular dynamics. *Journal of Computational Chemistry*, 16(10):1192–1209, 1995.

[10] D. ben-Avraham. Vibrational normal-mode spectrum of globular proteins. *Physical Review B*, 47(21):14559, 1993.

[11] S. Bonnabel and R. Sepulchre. Riemannian metric and geometric mean for positive semidefinite matrices of fixed rank. *SIAM J. Matrix Anal. Appl.*, 31(3):1055–1070, August 2009.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[13] F.J. Burkowski. *Structural Bioinformatics: An Algorithmic Approach*. Chapman & Hall/CRC, 2009.

[14] F.J. Burkowski. *Computational and Visualization Techniques for Structural Bioinformatics Using Chimera*. CRC Press, 2015.

[15] O. Calin and D.C. Chang. *Geometric mechanics on Riemannian manifolds: applications to partial differential equations*. Springer Science & Business Media, 2006.

[16] Y.L. Cheung, D. Drusvyatskiy, N. Krislock, and H. Wolkowicz. Noisy sensor network localization: robust facial reduction and the pareto frontier. *arXiv preprint arXiv:1410.6852*, 2014.

[17] G.S. Chirikjian. Group theory and biomolecular conformation: I. mathematical and computational models. *Journal of Physics: Condensed Matter*, 22(32):323103, 2010.

[18] A.R. Conn, N.I.M. Gould, and P.L. Toint. *Trust-Region Methods*. MPS/SIAM Series on Optimization, 2000.

[19] G.M. Crippen. *Distance geometry and conformational calculations*. Research Studies Press, 1981.

[20] F. Critchley. On certain linear mappings between inner-product and squared-distance matrices. *Linear Algebra and its Applications*, 105:91–107, 1988.

[21] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 1951.

[22] A. Das, M. Gur, M.H.Y. Cheng, S.H. Jo, I. Bahar, and B. Roux. Exploring the conformational transitions of biomolecular systems using a simple two-state anisotropic network model. *PLoS computational biology*, 10(4):e1003521, 2014.

[23] J. Dattorro. Equality relating Euclidean distance cone to positive semidefinite cone. *Linear Algebra and its Applications*, 428(11):2597–2600, 2008.

[24] J. Dattorro. *Convex optimization and Euclidean distance geometry, 2nd Edition*. MeBoo, 2017.

[25] D. Drusvyatskiy, N. Krislock, Y.-L. Voronin, and H. Wolkowicz. Noisy Euclidean distance realization: robust facial reduction and the pareto frontier. *SIAM Journal on Optimization*, 27(4):2301–2331, 2017.

[26] D. Drusvyatskiy, G. Pataki, and H. Wolkowicz. Coordinate shadows of semidefinite and Euclidean distance matrices. *SIAM Journal on Optimization*, 25(2):1160–1178, 2015.

[27] D.Z. Du. Remarks on the convergence of Rosen's gradient projection method. *Acta Mathematicae Applicatae Sinica*, 3(3):270–279, Jul 1987.

[28] D.Z. Du and X.S. Zhang. A convergence theorem of Rosens gradient projection method. *Mathematical programming*, 36(2):135–144, 1986.

[29] D.Z. Du and X.S. Zhang. Global convergence of Rosen's gradient projection method. *Mathematical Programming*, 44(1-3):357–366, 1989.

[30] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient simulation of inextensible cloth. *ACM Transactions on Graphics (TOG)*, 26(3):49, 2007.

[31] G.H. Golub and C.F. van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[32] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press Cambridge, 2016.

[33] J.D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[34] M. Journée, F. Bach, P.A. Absil, and R. Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM J. OPTIM*, 20(5):2327–2351, May 2010.

[35] A. Kessel and N. Ben-Tal. *Introduction to Proteins: structure, function, and motion*. CRC Press, 2011.

[36] M.H. Kim and M.K. Kim. Review: Elastic network model for protein structural dynamics. *JSM Enzymol Protein Sci*, 1(1), 2014.

[37] M.K. Kim. *Elastic Network Models of Biomolecular Structure and Dynamics*. PhD thesis, The Johns Hopkins University, 2004.

[38] M.K. Kim, G.S. Chirikjian, and R.L. Jernigan. Elastic models of conformational transitions in macromolecules. *Journal of Molecular Graphics and Modelling*, 21(2):151–160, 2002.

[39] M.K. Kim, Y.H. Jang, and J.I. Jeong. Using harmonic analysis and optimization to study macromolecular dynamics. *International Journal of Control Automation and Systems*, 4(3):382–393, 2006.

[40] M.K. Kim, R. L. Jernigan, and G.S. Chirikjian. Efficient generation of feasible pathways for protein conformational transitions. *Biophysical Journal*, 83(3):1620–1630, 2002.

[41] M.K. Kim, R.L. Jernigan, and G.S. Chirikjian. Rigid-cluster models of conformational transitions in macromolecular machines and assemblies. *Biophysical journal*, 89(1):43–55, 2005.

[42] M.K. Kim, L. Wen, B.A. Shapiro, and G.S. Chirikjian. A comparison between elastic network interpolation and MD simulation of 16S ribosomal RNA. *Journal of biomolecular structure & dynamics*, 21(3):395–405, 2003.

[43] N. Krislock. *Numerical solution of semidefinite constrained least squares problems*. PhD thesis, University of British Columbia, 2003.

[44] N. Krislock. *Semidefinite facial reduction for Low-Rank Euclidean Distance Matrix Completion*. PhD thesis, University of Waterloo, 2010.

[45] N. Krislock and H. Wolkowicz. Explicit sensor network localization using semidefinite representations and facial reductions. *SIAM Journal on Optimization*, 20(5):2679–2708, 2010.

[46] N. Krislock and H. Wolkowicz. Euclidean distance matrices and applications. In *Handbook on semidefinite, conic and polynomial optimization*, pages 879–914. Springer, 2012.

[47] B.H. Lee, S.J. Seo, M.H. Kim, Y.J. Kim, S.J. Jo, M.K. Choi, H.M. Lee, J.B. Choi, and M.K. Kim. Normal mode-guided transition pathway generation in proteins. *PloS one*, 12(10):e0185658, 2017.

[48] J.M. Lee. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 1997.

[49] J.M. Lee. *Introduction to Smooth Manifolds*. Springer, 2003.

[50] J.M. Lee. *Introduction to Topological Manifolds, Second Edition*. Springer, 2010.

[51] X.B. Li and F.J. Burkowski. Conformational transitions and principal geodesic analysis on the positive semidefinite matrix manifold. In M. Basu, Y. Pan, and J. Wang, editors, *Bioinformatics Research and Applications*, volume 8492 of *Lecture Notes in Computer Science*, pages 334–345. Springer International Publishing, 2014.

[52] X.B. Li and F.J. Burkowski. Generating conformational transitions using the Euclidean distance matrix. *IEEE transactions on nanobioscience*, 2015.

[53] X.B. Li, F.J. Burkowski, and H. Wolkowicz. Protein structure normal mode analysis on the positive semidefinite matrix manifold. In *8th International Conference on Bioinformatics and Computational Biology (BICOB 2016)*, 2016.

[54] X.B. Li, F.J. Burkowski, and H. Wolkowicz. Semidefinite facial reduction and rigid cluster elastic network interpolation of protein structures. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 132–136. IEEE, 2016.

[55] G. Meyer. *Geometric optimization algorithms for linear regression on fixed-rank matrices*. PhD thesis, University of Liège, 2011.

[56] J.C. Meza, T.D. Plantenga, and R.S. Judson. Novel applications of optimization to molecule design. In *Large-Scale Optimization with Applications*, pages 73–97. Springer, 1997.

[57] H. Mirzaei, D. Beglov, I. Paschalidis, S. Vajda, P. Vakili, and D. Kozakov. Rigid body energy minimization on manifolds for molecular docking. *Journal of chemical theory and computation*, 8(11):4374–4380, 2012.

[58] B. Mishra, G. Meyer, and R. Sepulchre. Low-rank optimization for distance matrix completion. In *2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 4455–4460, Orlando, FL, USA, December 12-15 2011.

[59] M. Moakher and P.G. Batchelor. Symmetric positive-definite matrices: From geometry to applications and visualization. In *Visualization and Processing of Tensor Fields*, pages 285–298. Springer, 2006.

[60] A. Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM review*, 39(3):407–460, 1997.

[61] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[62] R. Parhizkar. *Euclidean Distance Matrices: Properties, Algorithms and Applications*. PhD thesis, École Polytechnique Fédérake de Lausanne, 2013.

[63] E.F. Pettersen, T.D. Goddard, C.C. Huang, G.S. Couch, D.M. Greenblatt, E.C. Meng, and T.E. Ferrin. UCSF chimera a visualization system for exploratory research and analysis. *J Comp Chem*, 25(13):1605–1612, 2004.

[64] T.D. Plantenga. Fast energy minimization of large polymers using constrained optimization. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA, 1998.

[65] J.B. Rosen. Nonlinear programming. The gradient projection method (abstract 80). *Bulletin of the American Mathematical Society*, 63:25–26, 1957.

[66] J.B. Rosen. The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8(1):181–217, 1960.

[67] J.B. Rosen. The gradient projection method for nonlinear programming. part ii. nonlinear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):514–532, 1961.

[68] J.P. Ryckaert, G. Ciccotti, and H.J.C. Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23(3):327–341, 1977.

[69] B. Sadjad. *Robust Search Methods for Rational Drug Design Applications*. PhD thesis, University of Waterloo, 2009.

[70] T. Schlick. *Molecular modeling and simulation: an interdisciplinary guide*, volume 21. Springer Science & Business Media, 2010.

[71] S.J. Seo, Y.H. Jang, P.F. Qian, W.K. Liu, J.B. Choi, B.S. Lim, and M.K. Kim. Efficient prediction of protein conformational pathways based on the hybrid elastic network model. *Journal of Molecular Graphics and Modelling*, 47:25–36, 2014.

[72] V.E. Shestopal. Solution of the matrix equation AX- XB= C. *Mathematical notes of the Academy of Sciences of the USSR*, 19(3):275–276, 1976.

[73] J.R. Taylor. *Classical Mechanics*. University Science Books, 2005.

[74] M. Tekpinar and W.J. Zheng. Predicting order of conformational changes during protein conformational transitions using an interpolated elastic network model. *Proteins: Structure, Function, and Bioinformatics*, 78(11):2469–2481, 2010.

[75] M.M. Tirion. Large amplitude elastic motions in proteins from a single-parameter, atomic analysis. *Physical Review Letters*, 77(9):1905–1908, August 1996.

[76] M.M. Tirion and D. ben-Avraham. Normal mode analysis of G-actin. *Journal of molecular biology*, 230(1):186–195, 1993.

[77] L. Tu. *An Introduction to Manifolds, Second Edition*. Springer, 2011.

[78] B. Vandereycken. *Riemannian and multilevel optimization for rank-constrained matrix problems*. PhD thesis, Department of Computer Science, KU Leuven, 2010.

[79] N.J. Wildberger. *Divine Proportions: Rational Trigonometry to Universal Geometry*. Wild Egg, 2005.

[80] V. Yip and R. Elber. Calculations of a list of neighbors in molecular dynamics simulations. *Journal of Computational Chemistry*, 10(7):921–927, 1989.

[81] W.J. Zheng and M. Tekpinar. Analysis of protein conformational transitions using elastic network model. In D. Livesay (eds), editor, *Methods in molecular biology (Methods and Protocols)*, volume 1084, pages 159–172. Humana Press, 01 2014.

[82] M. Zvelebil and J. O. Baum. *Understanding bioinformatics*. Garland Science, 2008.

# Appendix A

# Interpolating transitions: An Object-Oriented Implementation in `python`

Sample `python` code for implementing ENI and iENM is presented in this Appendix.

## A.1 Background

The implementation choice made here is to treat the Hessian as a linear operator as discussed in Section 3.6.5. In Plantenga's algorithm for minimizing the semi-empirical potential, the author used the conjugate gradient algorithm since it only required the Hessian-vector product, and not the Hessian matrix explicitly. The author proposed to approximate the Hessian-vector product with a finite difference approximation:

$$\nabla^2 f(\overrightarrow{p})\eta = \frac{\nabla f(\overrightarrow{p} + \epsilon\overrightarrow{\eta}) - \nabla f(\overrightarrow{p})}{\epsilon} \ ,$$

where $\epsilon = 2\max\{1, \parallel \overrightarrow{p} \parallel\}\sqrt{\text{machine precisions}}/ \parallel \overrightarrow{\eta} \parallel$. This approximation is motivated by the definition of the directional derivative of a vector field, see Equation (3.28). Absil et al.[1] has provided a more formal development for arbitrary Riemannian manifolds.

The generalization of the truncated conjugate gradient algorithm (and the conjugate gradient algorithm) to an arbitrary Riemannian manifold, and to $\mathbf{S}_+^{n,3}$ in particular, is

straightforward because of this "inverse-free" property, which was also noted by Absil et al. [1] page 144:

> Notice that the truncated CG algorithm is "inverse-free", as it uses $H_k$ only in the computation of $H_k[\delta_k]$.

Recall from Section 3.6.5, the Riemannian connection generalizes the idea of taking the directional derivative of a vector field from $\mathbb{R}^{3n}$ to an arbitrary Riemannian manifold. Further, for submanifolds and quotient manifolds of Euclidean space, the Riemannian connection reduces to the Euclidean connection, which is the directional derivative followed by the appropriate projection onto the tangent space of the submanifold or quotient manifold. Therefore, for $\mathbf{S}_+^{n,3}$, $\text{Hess} f(P)[\eta]$ can be found exactly without the need for approximation methods. In fact, the same approach can be used for the Hookean potential. For the Hookean potential and the PSD potential, these directional derivatives have a closed form and will be presented in this Chapter.

The implementation in this Appendix shows the Hookean potentials can also be minimized *without* the regularizations discussed in Section 2.7, ill-posed does not mean a solution cannot be found, only the solution is not unique. The Hessian-free implementation overcomes this because the tCG algorithm makes adjustments to the Hessian-vector product, rather than inverting the Hessian.

As for previous authors in this area, a web server for ENI has been setup at `http://bioengineering.skku.ac.kr/kosmos/tutorial.php` by Kim. Matlab code for ENI is available from the "tutorial" tab. This code does form the Hessian matrix explicitly[1]. A web server has been set up for iENM at `https://enm.lobos.nih.gov`, but the source code is not available.

## A.2 Data Input and Data Output

The required input data to the transitional conformations problem are:

[1] A naive way to interpolate two protein conformations is to interpolate the beginning and ending conformation's Gram matrix or atomic coordinates directly, this is called "range interpolation" and was discussed in our previous publication [51]. A preliminary discussion of generalizing Hookean ENI to PSD ENI has appeared in our previous publication [52]. When Kim's matlab code was used to interpolated the lattice structures in this publication, anomalous stretching of bonds were observed. Implementing Hookean ENI with the Hessian matrix formed without an intermediate matrix, as was the case in Kim's implementation, does not give this stretching. The reason for this difference is unclear, and is not explored further in this thesis since the implementation here does not form the Hessian matrix explicitly.

- the atomic coordinates of the initial protein structure,
- the atomic coordinates of the ending protein structure, and
- information about which inter-atomic distance of these atoms are constrained.

This information is assumed to be available in textfiles.

This information can be obtained from a protein structure's Protein databank (PDB) file. *UCSF Chimera* [63] can be used to open a PDB file, extract the needed input data, and print them out to text files, UCSF Chimera allows all these steps to be written using `python`.

The output of the transitional conformations problem is a sequence of textfiles, each file contains the atomic coordinates of one intermediate conformation. These atomic coordinates can also be visualized using UCSF Chimera by updating the atomic coordinates of the initial conformation.

For more information on how to use UCSF Chimera to access PDB files and visualize protein structures see [14].

## A.3 The `R3n_Calculations` Class

This class implements calculations related to the generic Hookean potential. The `cost` function implements Equation (2.5), the generic Hookean potential:

$$E_H(\overrightarrow{p}) = \frac{1}{2} \sum_{(a,b)\in\mathcal{D}} w_{ab} \left( \sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} - \widehat{d}_{ab} \right)^2 ,$$

where $\widehat{d}_{ab}$ is the reference distance defined based on the application. The parameters of `cost` are `weight_L` a list of the weights for each pair of neighbours in $\mathcal{D}$, `distance_ref_L` a list of reference distances, `nbrs_L` a list representing the set $\mathcal{D}$, and finally `pvec` is the $3n \times 1$ vector $\overrightarrow{p}$.

The function `grad` implements forming the gradient of $E_H(\overrightarrow{p})$, grad$E_H(\overrightarrow{p})$, which is a $3n \times 1$ vector. The presentation of this gradient in Section 2.2 was based on the discussion in [37] where each $3 \times 1$ block was defined. The equation below is a more concise expression for the gradient, as *one summation*, derived by taking the vector derivative of the equation

for $E_H(\overrightarrow{p})$:

$$\text{grad}E_H(\overrightarrow{p}) \;\; = \sum_{(a,b)\in\mathcal{D}} w_{ab}\left(\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}} - \widehat{d}_{ab}\right)\frac{S_{ab}\,\overrightarrow{p}}{\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}}}\;. \tag{A.1}$$

The `grad` function uses a loop to calculate this summation. The definition of the stamp matrix $S_{ab}$, shows for each iteration of the loop, the $a$-th $3\times 1$ block of the $3n\times 1$ gradient vector receives the vector update:

$$w_{ab}\left(\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}} - \widehat{d}_{ab}\right)\frac{(e_{a,I_3} - e_{b,I_3})\overrightarrow{p}}{\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}}} = w_{ab}\left(\parallel p_a - p_b \parallel - \widehat{d}_{ab}\right)\frac{p_a - p_b}{\parallel p_a - p_b \parallel}\;,$$

whereas the $b$-th $3\times 1$ block gets the update with a minus sign:

$$-w_{ab}\left(\parallel p_a - p_b \parallel - \widehat{d}_{ab}\right)\frac{p_a - p_b}{\parallel p_a - p_b \parallel}\;.$$

The Hessian matrix is not formed. Instead, the `Hess` function implements the Euclidean directional derivative of the gradient in the direction $\overrightarrow{\eta}$, denoted

$$\text{Hess}E_H(\overrightarrow{p})[\overrightarrow{\eta}] = D\text{grad}E_H(\overrightarrow{p})[\overrightarrow{\eta}]\;,$$

which is a $3n\times 1$ vector (Plantenga called this value the "Hessian-vector" product). This expression is given by:

$$\begin{aligned}
&\text{Hess}E_H(\overrightarrow{p})[\overrightarrow{\eta}] = D\text{grad}E_H(\overrightarrow{p})[\overrightarrow{\eta}] \\
&= \sum_{(a,b)\in\mathcal{D}} w_{ab}\left[\left(\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}} - \widehat{d}_{ab}\right)\left(\frac{(\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}})S_{ab}\,\overrightarrow{\eta} - \left(\frac{\overrightarrow{p}^T S_{ab}\,\overrightarrow{\eta}}{\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}}}\right)S_{ab}\,\overrightarrow{p}}{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}}\right)\right. \\
&\qquad\left. + \left(\frac{\overrightarrow{p}^T S_{ab}\,\overrightarrow{\eta}}{\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}}}\right)\left(\frac{S_{ab}\,\overrightarrow{p}}{\sqrt{\overrightarrow{p}^T S_{ab}\,\overrightarrow{p}}}\right)\right]\;.
\end{aligned}$$

This expression is arrived at using the product rule and the quotient rule for taking

derivatives, it can be further simplified as follows:

$$
\begin{aligned}
&\text{Hess}E_H(\overrightarrow{p})[\overrightarrow{\eta}] \\
&= \sum_{(a,b)\in\mathcal{D}} w_{ab} \left[ \left( \sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} - \widehat{d}_{ab} \right) \left( \frac{S_{ab}\overrightarrow{\eta}}{\sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}}} - \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\left( \sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right)^3} \right) + \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right] \\
&= \sum_{(a,b)\in\mathcal{D}} w_{ab} \left[ \left( 1 - \frac{\widehat{d}_{ab}}{\sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}}} \right) \left( S_{ab}\overrightarrow{\eta} - \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right) + \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right] \\
&= \sum_{(a,b)\in\mathcal{D}} w_{ab} \left[ S_{ab}\overrightarrow{\eta} - \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} - \frac{\widehat{d}_{ab}}{\sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}}} \left( S_{ab}\overrightarrow{\eta} - \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right) + \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right] \\
&= \sum_{(a,b)\in\mathcal{D}} w_{ab} \left[ S_{ab}\overrightarrow{\eta} - \frac{\widehat{d}_{ab}}{\sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}}} \left( S_{ab}\overrightarrow{\eta} - \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right) \right]
\end{aligned}
$$

The final expression is used to implemented the `Hess` function, which returns the Hessian-vector product. This product is implemented using a loop. The definition of the stamp matrix $S_{ab}$ shows for each iteration of the loop, the $a$-th $3 \times 1$ block of the $3n \times 1$ vector $\text{Hess}E_H(\overrightarrow{p})[\overrightarrow{\eta}]$ gets the update:

$$
\begin{aligned}
&w_{ab} \left[ (e_{a,I_3} - e_{b,I_3})\overrightarrow{\eta} - \frac{\widehat{d}_{ab}}{\sqrt{\overrightarrow{p}^T S_{ab} \overrightarrow{p}}} \left( (e_{a,I_3} - e_{b,I_3})\overrightarrow{\eta} - \frac{\overrightarrow{p}^T S_{ab} \overrightarrow{\eta}\, S_{ab} \overrightarrow{p}}{\overrightarrow{p}^T S_{ab} \overrightarrow{p}} \right) \right] \\
&= w_{ab} \left[ (\eta_a - \eta_b) - \frac{\widehat{d}_{ab}}{\| p_a - p_b \|} \left( (\eta_a - \eta_b) - \left( \frac{(p_a - p_b)^T (\eta_a - \eta_b)}{\| p_a - p_b \|^2} \right) (p_a - p_b) \right) \right] ,
\end{aligned}
$$

while the $b$-th $3 \times 1$ block gets the update:

$$
-w_{ab} \left[ (\eta_a - \eta_b) - \frac{\widehat{d}_{ab}}{\| p_a - p_b \|} \left( (\eta_a - \eta_b) - \left( \frac{(p_a - p_b)^T (\eta_a - \eta_b)}{\| p_a - p_b \|^2} \right) (p_a - p_b) \right) \right]
$$

All summations can be divided by $|\mathcal{D}|$ to prevent these terms from getting too big.

---

```
class R3n_Calculations:

    def __init__(self):
```

```
    pass


def cost(self, weight_L, distance_ref_L, nbrs_L,pvec):

    if len(nbrs_L) == 0:
        return 0
    Total = 0
    for i in range(len(nbrs_L)):
        nbrsi = nbrs_L[i]
        a = nbrsi [0]
        b = nbrsi [1]
        dab = la.norm(pvec[3*a:3*(a+1),:]-pvec[3*b:3*(b+1),:])
        erri = dab - distance_ref_L[i]
        Total = Total + weight_L[i]* (erri **2)

    return 0.5*Total


def grad(self, weight_L, distance_ref_L, nbrs_L, pvec):

    grad =0.0*pvec
    if len(nbrs_L)==0:
        return grad

    for i in range(len(nbrs_L)):
        nbrsi = nbrs_L[i]
        a = nbrsi [0]
        b = nbrsi [1]
        pvec_ab = pvec[3*a:3*(a+1),:]-pvec[3*b:3*(b+1),:]
        dab = la.norm(pvec_ab)
        erri = dab - distance_ref_L[i]

        entry = weight_L[i]* erri/dab * (pvec_ab)

        # update 3 by 1 blocks
        grad[3*a:3*(a+1),:] =grad[3*a:3*(a+1),:]+ entry
        grad[3*b:3*(b+1),:] =grad[3*b:3*(b+1),:]- entry

    return grad
```

```
def Hess(self, weight_L, distance_ref_L, nbrs_L, pvec, dirvec):

    HessDir = 0.0*pvec
    if len(nbrs_L) ==0:
        return HessDir
    for i in range(len(nbrs_L)):
        nbrsi = nbrs_L[i]
        a = nbrsi [0]
        b = nbrsi [1]

        pvec_ab   = pvec[3*a:3*(a+1),:]-pvec[3*b:3*(b+1),:]
        dirvec_ab = dirvec[3*a:3*(a+1),:]-dirvec[3*b:3*(b+1),:]

        dab        = la.norm(pvec_ab)

        cross = sp.sum(sp.multiply(pvec_ab, dirvec_ab))/dab**2

        entry = weight_L[i]*(dirvec_ab - (distance_ref_L[i]/dab)*(dirvec_ab
            - cross * pvec_ab) )

        # update 3 by 1 blocks
        HessDir[3*a:3*(a+1),:] = HessDir[3*a:3*(a+1),:] + entry
        HessDir[3*b:3*(b+1),:] = HessDir[3*b:3*(b+1),:] - entry

    return HessDir
```

## A.4 The PSD3_Calculations Class

This class implements calculations relating to the generic PSD potential. The `cost` function implements calculating the generic PSD potential:

$$E_{PSD}(P) = \frac{1}{2} \sum_{(a,b)\in\mathcal{D}} w_{ab} \left(\mathrm{err}(a,b)\right)^2 \ ,$$

where

$$\text{err}(a,b) = (e_a - e_b)PP^T(e_a - e_b) - \widehat{q}_{ab}$$

where $\widehat{q}_{ab}$ is a reference quadrance defined based on the application. The parameters of the PSD `cost` function is the same as those used in the `R3n_Calculations`, with the matrix $P$ replacing the vector $\overrightarrow{p}$.

The `grad` function implements the calculation:

$$\text{grad}E_{PSD}(P) = \sum_{(a,b)\in\mathcal{D}} w_{ab}\text{err}(a,b)(e_a - e_b)(e_a - e_b)^T P$$

The `Hess` function implements the calculation of the Euclidean directional derivative:

$$\text{Hess}E_{PSD}(P)[\eta] = D\text{grad}E_{PSD}(P)[\eta]$$

which is given by:

$$D\text{grad}E_{PSD}(P)[\eta] = \sum_{(a,b)\in\mathcal{D}} w_{ab}\left[\text{err}(a,b)(e_a - e_b)(e_a - e_b)^T\eta + \text{cross}(a,b)(e_a - e_b)(e_a - e_b)^T P\right] ,$$

where

$$\text{cross}(a,b) = 2(e_a - e_b)^T P\eta^T(e_a - e_b) = 2(p_a - p_b)^T(\eta_a - \eta_b) .$$

---

```
class PSD3_Calculations:

    def __init__(self):
        pass

    def cost(self, weight_L, quadrance_ref_L, nbrs_L,P):

        if len(nbrs_L) == 0:
            return 0
        Total = 0
        for i in range(len(nbrs_L)):
            a = nbrs_L[i][0]
            b = nbrs_L[i][1]
            qab = sp.sum(sp.square(P[a,:]-P[b,:]))
            erri = qab - quadrance_ref_L[i]
            Total = Total + weight_L[i]* (erri **2)
```

```
    return 0.5*Total


def grad(self, weight_L, quadrance_ref_L, nbrs_L, P):

    grad =0.0*P
    if len(nbrs_L)==0:
        return grad

    for i in range(len(nbrs_L)):
        a = nbrs_L[i][0]
        b = nbrs_L[i][1]

        qab = sp.sum(sp.square(P[a,:]-P[b,:]))
        erri = qab - quadrance_ref_L[i]
        entry = weight_L[i]* erri * (P[a,:]-P[b,:])
        grad[a,:] =grad[a,:]+ entry
        grad[b,:] =grad[b,:]- entry

    return 2*grad

def Hess(self, weight_L, quadrance_ref_L, nbrs_L, P, Dir):

    HessDir = 0.0*P
    if len(nbrs_L) ==0:
        return HessDir

    for i in range(len(nbrs_L)):
        a = nbrs_L[i][0]
        b = nbrs_L[i][1]
        qab = sp.sum(sp.square(P[a,:]-P[b,:]))
        erri = qab - quadrance_ref_L[i]

        crossab = 2* sp.sum(sp.multiply(P[a,:]-P[b,:], Dir[a,:]-Dir[b,:]))

        entry = weight_L[i]* erri * (Dir[a,:]-Dir[b,:]) + weight_L[i]*
            crossab * (P[a,:]-P[b,:])
        HessDir[a,:] = HessDir[a,:] + 2*entry
        HessDir[b,:] = HessDir[b,:] - 2*entry
```

```
    return HessDir
```

## A.5 The `Potential` Base Class

The `Potential` base class is a class from which the Hookean and PSD potentials inherit from. This class has functions common to both potentials, these functions relate to finding which atoms are neighbours with which other atom.

A neighbour of an atom is defined to be another atom whose distance is less than some threshold distance away. The neighbour may be bonded or not bonded. Finding the neighbours of each atom exhaustively is very inefficient. For a protein with $n$ atoms, labelled $1, \ldots, n$, determining the neighbours of the first atom requires a check of its distance with $n-1$ atoms, determining the neighbours of the second atom requires a check of its distance with $n-2$ atoms, for the third atom, there are $n-3$ checks, and so on, resulting in:

$$(n-1) + (n-2) + (n-3) + \cdots = O(n^2)$$

computations. A faster method has been discussed in detail in Chapter 3 Section 3.4 of Burkowski[14] and also appears in other papers, for example Yip and Elber[80]. This method encloses the protein with a bounding box (bbox) as shown in Figure A.1. The corners of this bbox are the points

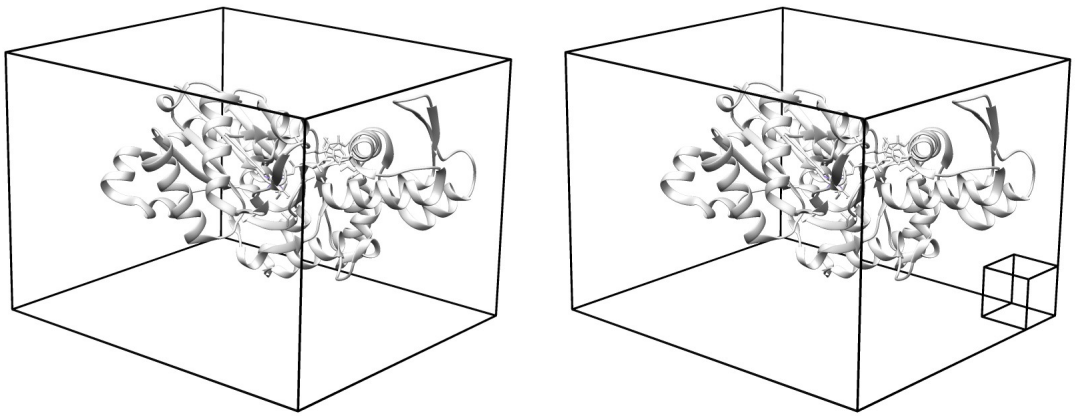
$$(x, y, z) \text{ where } x \in \{x_{min}, x_{max}\}, y \in \{y_{min}, y_{max}\}, z \in \{z_{min}, z_{max}\},$$

where $x_{min}$ is the minimum value of the $x$ coordinate of all atoms in the protein, $x_{max}$ is the maximum value of all the $x$ coordinates of the protein rounded up to an *exact multiple* of the threshold neighbour distance; $y_{min}$, $z_{min}$ $y_{max}$, $z_{max}$ are defined similarly. This bbox is then further divided uniformly into a grid of *smaller* cubes where each side of the cube has length equal to the threshold distance for an atom to be considered a neighbour. Figure A.1 a) shows a protein surrounded by a bbox, Figure A.1 b) shows one grid cube. These cubes are index by 3 coordinates $(c_x, c_y, c_z)$, called the "cube coordinates". Each cube coordinate serves as a *key* into a `dictionary` (hashmap) with the corresponding *value* being a list of atomic indices representing the atoms that are enclosed by this grid cube. The construction of this dictionary requires iterating through all atoms, and therefore this method requires $O(n)$ operations to set up.

When determining the neighbours of a particular atom, rather than checking all atoms to see if they have distance less than the threshold distance, the atoms in this particular

atom's grid cube, and the surrounding grid cubes (at most 26) are the only ones that needs to be checked. This number is independent of $n$, therefore, finding the neighbours of a particular atom requires $O(1)$ operations. Finding the neighbours of all $n$ atoms thus require $O(n)$ operations.

The `Potential` class assumes atomic coordinates are given as a list of tuples. It does not have knowledge of either the $\mathbb{R}^{3n}$ or $\mathbf{S}_{+}^{n,3}$ geometries.



(a) A bounding box. (b) A bounding box with one grid cube.

Figure A.1: A bounding box around a protein.

```
class Potential(object):

    def __init__(self, name, thres):
        self.name = name

        # threshold distance for neighbours
        self.thres = thres

    # Returns the type of potential
    def getName(self):
        return "potential base class "+self.name+" threshold interaction
            distance "+str(self.thres)
    # ----------------------------------------
```

```
# Finds the smallest coords, this is one corner of the bounding box.
def findmintuple(self,coord_L):
    # initialize to infinity
    xmin = ymin = zmin = float('inf')
    for a in coord_L:
        if a[0]< xmin: xmin = a[0]
        if a[1]< ymin: ymin = a[1]
        if a[2]< zmin: zmin = a[2]
    return (xmin, ymin, zmin)

# ----------------------------------------
# Finds the largest coords, this is another corner of the bounding box.
def findmaxtuple(self,coord_L):
    # initialize to negative infinity
    xmin, ymin, zmin = self.findmintuple(coord_L)

    xmax = ymax = zmax = -float('inf')
    for a in coord_L:
        if a[0] > xmax: xmax = a[0]
        if a[1] > ymax: ymax = a[1]
        if a[2] > zmax: zmax = a[2]

    multiple_x = sp.ceil((xmax - xmin)/self.thres)
    multiple_y = sp.ceil((ymax - ymin)/self.thres)
    multiple_z = sp.ceil((zmax - zmin)/self.thres)
    xmax_exactmult = xmin + multiple_x * self.thres
    ymax_exactmult = ymin + multiple_y * self.thres
    zmax_exactmult = zmin + multiple_z * self.thres

    return (xmax_exactmult, ymax_exactmult, zmax_exactmult)

# Given a list of atomic coordinates, updates which atom belong to which
# cube in the grid of cubes and return list of neighbours for each atom.
def updateGrid(self, coord_L):

    self.mintuple  = self.findmintuple(coord_L)
    self.maxtuple  = self.findmaxtuple(coord_L)
    self.grid      = self.buildGrid(coord_L)

    nbr_pairs = self.getAllNbrs(coord_L)
```

```
    return nbr_pairs

# Get the neighbours of all atoms
def getAllNbrs(self, coord_L):

    # nbr_L is one big list of pairs, each pair are two neighbour atoms.
    nbr_L = []
    for i in range(len(coord_L)):

        # Get atom i's neighbours.
        nbr = self.getListofIndicesOfNbrAtoms(i, coord_L)

        # Joint the current atom's neighbour index with the global
        # neighbour list
        nbr_L=nbr_L + nbr

    return nbr_L

# ----------------------------------------
# Take a atomic coord convert it to the cube it is in.
def convCoordtoGridCoord(self, coord):
    cubex = int(coord[0] - self.mintuple[0])/self.thres
    cubey = int(coord[1] - self.mintuple[1])/self.thres
    cubez = int(coord[2] - self.mintuple[2])/self.thres

    return (cubex, cubey, cubez)
# ----------------------------------------
# Build grid for atomic coords. The grid is a dictionary (hashmap).
# key: grid cube coordinates,
# values: list of atomic indices for atoms in this cube
def buildGrid(self, coord_L):
    grid = {}
    for i in range(len(coord_L)):

        key = self.convCoordtoGridCoord(coord_L[i])

        grid.setdefault(key, []).append(i)
    return grid
# ----------------------------------------
```

```
# Get neighbours using grid cubes around atom index center_ix.
# Returns a list of tuples (center_ix, nbr_ix).
def getListofIndicesOfNbrAtoms(self, center_ix, coord_L):

    # return tuple list, each tuple is (center_ix, nbr_ix) where
    # nbr_ix is center atom's neighbour atom index
    ngbrpair_L =[]

    cube = self.convCoordtoGridCoord(coord_L[center_ix])
    center_coord = coord_L[center_ix]

    x,y,z = cube[0], cube[1], cube[2] # cube coord

    for i in range(x-1, x+2):
        for j in range(y-1, y+2):
            for k in range(z-1, z+2):

                key = (i,j,k)

                if self.grid.has_key(key):

                    ngbr_L = self.grid[key]
                    sorted(ngbr_L)

                    for ix in ngbr_L:
                        if ix <= center_ix:
                            # skip nbrs that are smaller index
                            # skip center_ix as well
                            continue

                        nbrcoord =coord_L[ix]
                        distance = sp.sqrt((center_coord[0] - nbrcoord[0])**2\
                                        + (center_coord[1] - nbrcoord[1])**2\
                                        + (center_coord[2] - nbrcoord[2])**2)
                        if distance < self.thres:
                            # atom ix is within theshold so include
                            # it in nbrs list

                            ngbrpair_L.append((center_ix, ix))
```

```
        ngbrpair_L = sorted(ngbrpair_L)

        return ngbrpair_L
```

There are three types of Hookean potentials inheriting from the `Potential` class: the `Hookean_ENI` potential for Hookean ENI, the `Hookean_iENM` potential for Hookean iENM, and the `Hookean_Collision` energy for avoiding atomic collisions. All three classes requires the `R3n_Calculations` class to be imported.

There are also three types of PSD potentials inheriting from the `Potential` class: the `PSD_ENI` potential for PSD ENI, the `PSD_iENM` potential for PSD iENM, and the `PSD_Collision` energy for avoiding atomic collisions. All three classes require the `PSD3_Calculations` class to be imported.

These classes are described below. Assume the `Potential` class is in a file called `potential.py`, the `R3n_Calculations` class is in a file called `r3nCalculations.py`, and the `PSD3_Calculations` class is in a file called `psd3Calculations.py`.

## A.6 The `Hookean_ENI` Class

This class implements the Hookean ENI potential calculations, see Section 2.4. The purpose of this class is to pass the required data to the `R3n_Calculations` class which is performing the actual calculations.

The parent class is used to determine neighbour atoms. Kim suggests the ENI potential should use the union of the starting and ending conformation's neighbour lists, see [37] page 65, this is the convention used for this class.

```
import r3nCalculations as r3ncalc
from potential import Potential

class Hookean_ENI(Potential):

    def __init__(self, name, bonds_L, thres, coord0_L, coord1_L, bondweight,
        nonbondweight):

        Potential.__init__(self, name, thres)
        self.r3ncalc = r3ncalc.R3n_Calculations()
        self.bondweight = bondweight
```

```
        self.nonbondweight = nonbondweight

        # determine neighbours
        nbrs0_L = super(Hookean_ENI, self).updateGrid(coord0_L)
        nbrs1_L = super(Hookean_ENI, self).updateGrid(coord1_L)

        # ENI uses the union of the neighbour list.
        self.nbrs_L = sorted(list(set().union(nbrs0_L, nbrs1_L)))
        self.weight_interactions = []
        # Convert bonds_L to a set so membership check is O(1),
        # see "https://wiki.python.org/moin/TimeComplexity".
        self.bonds_L = set(bonds_L)

        for pair in self.nbrs_L:
            if pair in self.bonds_L:
                self.weight_interactions.append(bondweight)
            else:
                self.weight_interactions.append(nonbondweight)
        # put atomic coordinates into 3n by 1 vectors
        self.pvec0 = sp.matrix(coord0_L).reshape(len(coord0_L)*3, 1)
        self.pvec1 = sp.matrix(coord1_L).reshape(len(coord1_L)*3, 1)

    def getStartConformation(self):
        return self.pvec0

    def getEndConformation(self):
        return self.pvec1
    # Returns the linearly interpolated reference distance.
    def get_distance_ref(self, t):
        d_ref = []

        for pairs in self.nbrs_L:
            a = pairs[0]
            b = pairs[1]

            dab0 = la.norm(self.pvec0[3*a:3*(a+1),:]-self.pvec0[3*b:3*(b+1),:])
            dab1 = la.norm(self.pvec1[3*a:3*(a+1),:]-self.pvec1[3*b:3*(b+1),:])
            d_ref.append(dab0 * (1-t)+ dab1 *t)

        return d_ref
```

```
    def cost(self, pvec, t):

        distance_ref = self.get_distance_ref(t)
        return self.r3ncalc.cost(self.weight_interactions, distance_ref,
            self.nbrs_L, pvec)

    def grad(self, pvec, t):

        distance_ref = self.get_distance_ref(t)
        return self.r3ncalc.grad(self.weight_interactions, distance_ref,
            self.nbrs_L, pvec)

    def Hess(self, pvec, dirvec, t):

        distance_ref = self.get_distance_ref(t)
        return self.r3ncalc.Hess(self.weight_interactions, distance_ref,
            self.nbrs_L, pvec, dirvec)
```

## A.7 The Hookean_iENM Class

This class implements the linearly interpolated Hookean iENM potential calculations, see Section 2.5. The required data are passed to the R3n_Calculations class which will perform the actual calculations.

```
import r3nCalculations as r3ncalc
from potential import Potential
# This class implements the Hookean iENM model
class Hookean_iENM(Potential):
    def __init__(self, name, bonds_L, thres, coord0_L, coord1_L, bondweights,
        nonbondweights):

        Potential.__init__(self, name, thres)
        # Initialize R3n_Calculations isntance for generic Hookean potential
            calculations
        self.r3ncalc = r3ncalc.R3n_Calculations()
        self.bonds_L = bonds_L
        self.nbrs0_L = super(Hookean_iENM, self).updateGrid(coord0_L)
```

```
        self.nbrs1_L = super(Hookean_iENM, self).updateGrid(coord1_L)

        # put atomic coordinates into 3n by 1 vectors
        self.pvec0 = sp.matrix(coord0_L).reshape(len(coord0_L)*3, 1)
        self.pvec1 = sp.matrix(coord1_L).reshape(len(coord1_L)*3, 1)

        # set the weight vector for the beginning structure's interaction pairs
        self.weight_interactions0 = self.get_interaction_weights(bondweights,
            nonbondweights, self.nbrs0_L)
        # set the weight vector for the ending structure's interaction pairs
        self.weight_interactions1 = self.get_interaction_weights(bondweights,
            nonbondweights, self.nbrs1_L)

        # set reference distance for the beginning and ending structures
        self.distance0 = self.get_distance_ref(self.pvec0, self.nbrs0_L)
        self.distance1 = self.get_distance_ref(self.pvec1, self.nbrs1_L)

    def getStartConformation(self):
        return self.pvec0

    def getEndConformation(self):
        return self.pvec1

    def get_interaction_weights(self, bondweights, nonbondweights, nbrs_L):
        weight_L = []
        for pair in nbrs_L:

            if pair in self.bonds_L:
                weight_L.append(bondweights) # weight for bond
            else:
                weight_L.append(nonbondweights) # weight for nonbonded
                    interaction

        return weight_L
    # Reference distance based on atomic coordinates in nbrs_L.
    def get_distance_ref(self, pvec, nbrs_L):
        d_ref =[]
        for pairs in nbrs_L:
            a = pairs[0]
            b = pairs[1]
```

```
            dab = la.norm(pvec[3*a:3*(a+1),:]-pvec[3*b:3*(b+1),:])
            d_ref.append(dab)
        return d_ref

    def cost(self, pvec, t):

        cost0 = self.r3ncalc.cost(self.weight_interactions0,self.distance0,
            self.nbrs0_L, pvec)
        cost1 = self.r3ncalc.cost(self.weight_interactions1,self.distance1,
            self.nbrs1_L, pvec)
        return (1-t)*cost0 + t*cost1

    def grad(self, pvec, t):

        grad0 = self.r3ncalc.grad(self.weight_interactions0,self.distance0,
            self.nbrs0_L, pvec)
        grad1 = self.r3ncalc.grad(self.weight_interactions1,self.distance1,
            self.nbrs1_L, pvec)
        return (1-t)*grad0 + t*grad1

    def Hess(self, pvec, dirvec, t):

        Hess0 = self.r3ncalc.Hess(self.weight_interactions0, self.distance0,
            self.nbrs0_L, pvec, dirvec)
        Hess1 = self.r3ncalc.Hess(self.weight_interactions1, self.distance1,
            self.nbrs1_L, pvec, dirvec)

        return (1-t)* Hess0 + t * Hess1
```

## A.8 The Hookean_Collision Energy

This class implements the steric collision energy of Tekpinar and Zheng [74, 81], which prevents atoms from colliding by penalizing atoms that are within a given collision threshold distance as given by the `thres` parameter, see Section 2.6.

```
import r3nCalculations as r3ncalc
from potential import Potential
class Hookean_Collision(Potential):
```

```
def __init__(self, name, bonds_L, thres, weight):

    Potential.__init__(self, name, thres)
    # Convert bonds_L to a set so membership check is O(1),
    # see "https://wiki.python.org/moin/TimeComplexity".
    self.bonds_L = set(bonds_L)
    self.weight = weight
    self.r3ncalc = r3ncalc.R3n_Calculations()
    self.thres = thres
    self.checkPairs_L =[]# pairs of atoms within collision threshold
    self.collision_weights =[] # weight applied to pairs in
        self.checkPairs_L

# Check if any atoms have collided, ignore bonded atoms.
def updateCollisionPairs(self, pvec):

    # Reshape pvec from 3n by 1 to n by 3 in order to convert to list of
        coords.
    P = pvec.reshape(len(pvec)/3, 3)
    coord_L = P.tolist()

    pairs_L = super(Hookean_Collision, self).updateGrid(coord_L)

    self.checkPairs_L =[]
    self.collision_weights =[]
    for pair in pairs_L:
        if not pair in self.bonds_L:
            self.checkPairs_L.append(pair)
            self.collision_weights.append(self.weight)

    self.distance_ref = [self.thres]*len(self.checkPairs_L)

def cost(self, pvec):

    return self.r3ncalc.cost(self.collision_weights, self.distance_ref,
        self.checkPairs_L, pvec)
def grad(self, pvec):

    return self.r3ncalc.grad(self.collision_weights, self.distance_ref,
```

```
            self.checkPairs_L, pvec)
    def Hess(self, pvec, dirvec):

        return self.r3ncalc.Hess(self.collision_weights, self.distance_ref,
            self.checkPairs_L, pvec, dirvec)
```

## A.9 The PSD_ENI Class

This class implements the PSD ENI potential, which is the generic PSD potential with the reference quadrance set to the linearly interpolated quadrance between the beginning and ending conformations. Data are passed to the PSD3_Calculations class which performs the actual calculations.

```
import psd3Calculations as psdcalc
from potential import Potential
class PSD_ENI(Potential):
    def  __init__(self, name, bonds_L, thres, coord0_L, coord1_L, bondweight,
        nonbondweight):

        Potential.__init__(self, name, thres)
        self.bondweight = bondweight
        self.nonbondweight = nonbondweight

        self.psdcalc = psdcalc.PSD3_Calculations()

        # determine neighbours
        nbrs0_L = super(PSD_ENI, self).updateGrid(coord0_L)
        nbrs1_L = super(PSD_ENI, self).updateGrid(coord1_L)

        # ENI uses the union of the neighbour list.

        self.nbrs_L = sorted(list(set().union(nbrs0_L, nbrs1_L)))

        self.weight_interactions = []

        # Convert bonds_L to a set so membership check is O(1),
        # see "https://wiki.python.org/moin/TimeComplexity".
        self.bonds_L = set(bonds_L)
```

```
    for pair in self.nbrs_L:

        if pair in self.bonds_L:

            self.weight_interactions.append(bondweight)
        else:

            self.weight_interactions.append(nonbondweight)

    # convert coord to matrix
    self.P0 = sp.matrix(coord0_L)
    self.P1 = sp.matrix(coord1_L)

def getStartConformation(self):
    return self.P0

def getEndConformation(self):
    return self.P1

def getBonds(self):
    return self.bonds_L

# Get interpolated quadrance with parameter t
def get_quadrance_ref(self, t):

    q_ref = []
    for pairs in self.nbrs_L:
        a = pairs[0]
        b = pairs[1]
        qab0 = sp.sum(sp.square(self.P0[a,:]-self.P0[b,:]))
        qab1 = sp.sum(sp.square(self.P1[a,:]-self.P1[b,:]))
        q_ref.append(qab0*(1-t) + qab1 * t)

    return q_ref

def cost(self,P, t):

    quadrance_ref =self.get_quadrance_ref(t)

    return self.psdcalc.cost(self.weight_interactions, quadrance_ref,
```

```
            self.nbrs_L, P)

    def grad(self, P, t):

        quadrance_ref = self.get_quadrance_ref(t)
        return self.psdcalc.grad(self.weight_interactions, quadrance_ref,
            self.nbrs_L, P)

    def Hess(self, P, Dir, t):

        quadrance_ref =self.get_quadrance_ref(t)

        return self.psdcalc.Hess(self.weight_interactions, quadrance_ref,
            self.nbrs_L, P, Dir)
```

## A.10 The PSD_iENM Class

The PSD_iENM Class implements the linearly interpolated PSD iENM potential. Data are passed to the PSD3_Calculations class, which performs the actual calculations.

```
class PSD_iENM(Potential):
    def  __init__(self, name, bonds_L, thres, coord0_L, coord1_L, bondweights,
        nonbondweights):
        print "PSD iENM constructor"

        Potential. __init__(self, name, thres)

        self.psdcalc = psdcalc.PSD3_Calculations()

        self.bonds_L = bonds_L

        self.nbrs0_L = super(PSD_iENM, self).updateGrid(coord0_L)
        self.nbrs1_L = super(PSD_iENM, self).updateGrid(coord1_L)

        self.P0 = sp.matrix(coord0_L)
        self.P1 = sp.matrix(coord1_L))

        # set the weight vector for the beginning structure's interaction pairs
```

```
        self.weight_interactions0 = self.get_interaction_weights(bondweights,
            nonbondweights, self.nbrs0_L)
        # set the weight vector for the ending structure's interaction pairs
        self.weight_interactions1 = self.get_interaction_weights(bondweights,
            nonbondweights, self.nbrs1_L)


        self.quadrance0 = self.get_quadrance_ref(self.P0, self.nbrs0_L)
        self.quadrance1 = self.get_quadrance_ref(self.P1, self.nbrs1_L)

    def getStartConformation(self):
        return self.P0

    def getEndConformation(self):
        return self.P1

    def get_interaction_weights(self, bondweights, nonbondweights, nbrs_L):
        weight_L = []
        for pair in nbrs_L:

            if pair in self.bonds_L:
                # weight for bond
                weight_L.append(bondweights)
            else:
                # weight for nonbonded interaction
                weight_L.append(nonbondweights)

        return weight_L

    def get_quadrance_ref(self, P, nbrs_L):
        q_ref =[]
        for pairs in nbrs_L:
            a = pairs[0]
            b = pairs[1]
            qab = sp.sum(sp.square(P[a,:]-P[b,:]))
            q_ref.append(qab)
        return q_ref

    def cost(self,P, t):
```

```
        cost0 = self.psdcalc.cost(self.weight_interactions0,self.quadrance0,
            self.nbrs0_L, P)
        cost1 = self.psdcalc.cost(self.weight_interactions1,self.quadrance1,
            self.nbrs1_L, P)

        return (1-t)*cost0 + t*cost1

    def grad(self, P, t):

        grad0 = self.psdcalc.grad(self.weight_interactions0,self.quadrance0,
            self.nbrs0_L, P)
        grad1 = self.psdcalc.grad(self.weight_interactions1,self.quadrance1,
            self.nbrs1_L, P)
        return (1-t)* grad0 + t*grad1

    def Hess(self, P, Dir, t):

        Hess0 = self.psdcalc.Hess(self.weight_interactions0, self.quadrance0,
            self.nbrs0_L, P, Dir)
        Hess1 = self.psdcalc.Hess(self.weight_interactions1, self.quadrance1,
            self.nbrs1_L, P, Dir)


        return (1-t)* Hess0 + t * Hess1
```

## A.11 The PSD_Collision Class

This class implements the PSD collision avoidance energy. It is the generic PSD potential with reference distance equal to the threshold distance for which a collision is assumed to have occurred between atoms.

The collision threshold distance, `thres`, is the distance used to construct the grid of cubes.

```
import psd3Calculations as psdcalc
from potential import Potential
class PSD_Collision(Potential):
```

```
def __init__(self, name, bonds_L, thres, weight):

    Potential.__init__(self, name, thres)
    # Convert bonds_L to a set so membership check is O(1),
    # see "https://wiki.python.org/moin/TimeComplexity".
    self.bonds_L = set(bonds_L)
    self.weight = weight
    self.psdcalc = psdcalc.PSD3_Calculations()
    self.thres = thres
    # pairs of atoms within collision threshold
    self.checkPairs_L =[]
    # weight applied to pairs in self.checkPairs_L
    self.collision_weights =[]



# Check if any atoms have collided, ignore bonded atoms.
def updateCollisionPairs(self, P):

    coord_L = P.tolist()
    pairs_L = super(PSD_Collision, self).updateGrid(coord_L)
    self.checkPairs_L =[]
    self.collision_weights =[]
    for pair in pairs_L:
        if not pair in self.bonds_L:
            self.checkPairs_L.append(pair)
            self.collision_weights.append(self.weight)

    self.quadrance_ref = [self.thres**2]*len(self.checkPairs_L)

def cost(self, P):

    return self.psdcalc.cost(self.collision_weights, self.quadrance_ref,
        self.checkPairs_L, P)

def grad(self, P):

    return self.psdcalc.grad(self.collision_weights, self.quadrance_ref,
        self.checkPairs_L, P)
```

```
    def Hess(self, P, Dir):

        return self.psdcalc.Hess(self.collision_weights, self.quadrance_ref,
            self.checkPairs_L, P, Dir)
```

## A.12 The `Manifold` Class

This is the base Manifold class. It implements the functions common to both $\mathcal{C}(\mathbb{R}^{3n})$ and $\mathcal{C}(\mathbf{S}_+^{n,3})$, as discussed in Section 6.4.

The adjacency list `self.adjacencylist_D` is a dictionary with key an atomic index, and the value being a list of atomic indices of atoms that are involved in a constraint with the key index. This list is used to create `self.edgeAdjacencyList_D`, which is another adjacency list where the key is a pair of atomic indices for two atoms in a constraint, and the value is a list of pairs, each pair is a constraint that shares an atom with the key pair. These lists help to construct the sparse matrix $Q$ and its directional derivative.

```
class Manifold(object):

    def __init__(self, name, constraints_L):

        self.name = name
        self.constraints_L = constraints_L

        self.makeAdjacencyList()
        self.makeEdgeAdjacencyList()

    # Adjacency list of the protein, implemented using a
    # dictionary.

    def makeAdjacencyList(self):
        self.adjacencylist_D = {}

        for constr in self.constraints_L:
            a = constr[0]
            b = constr[1]

            if a in self.adjacencylist_D:
```

```
            self.adjacencylist_D[a].append(b)
        else:
            self.adjacencylist_D[a]=[b]
        if b in self.adjacencylist_D:
            self.adjacencylist_D[b].append(a)
        else:
            self.adjacencylist_D[b]=[a]

def makeEdgeAdjacencyList(self):
    self.edgeAdjacencyList_D = {}
    for pair in self.constraints_L:

        a = pair[0]
        b = pair[1]

        nbr_c_L =[]

        # get edges ajacent to a
        for val in self.adjacencylist_D[a]:

            nbr_c = tuple(sorted((a, val)))
            nbr_c_L.append(nbr_c)

        # get edges adjacent to b
        for val in self.adjacencylist_D[b]:

            nbr_c = tuple(sorted((b, val)))
            if not nbr_c in nbr_c_L:
                nbr_c_L.append(nbr_c)

        if pair in self.edgeAdjacencyList_D:
            self.edgeAdjacencyList_D[pair]=self.edgeAdjacencyList_D[pair] +
                nbr_c_L
        else:
            self.edgeAdjacencyList_D[pair] = nbr_c_L


# Return the Q matrix for solving for Lagrange multipliers
# as a sparse matrix
def makeQ(self, p1, p2):
```

```
# reshape to n by 3 arrays for easier handling
r1,c1 = p1.shape
r2,c2 = p1.shape
if c1 != 3:
    p1 = p1.reshape((len(p1)/3, 3))
if c2 != 3:
    p2 = p2.reshape((len(p2)/3, 3))

const_I_L =[]
const_J_L = []
entry_IJ_L = []
for pairI in self.edgeAdjacencyList_D :

    c_i = self.constraints_L.index(pairI)
    diff_I = p1[pairI[0],:]-p1[pairI[1],:]

    for pairJ in self.edgeAdjacencyList_D[pairI]:
        c_j = self.constraints_L.index(pairJ)

        diff_J = p2[pairJ[0],:]-p2[pairJ[1],:]

        dot = sp.sum(sp.multiply(diff_I, diff_J))

        sign = -1
        if c_i == c_j:
            sign = 2
        elif pairI[0]==pairJ[0] or pairI[1]==pairJ[1]:
            sign = 1
        entry_ij = sign * dot

        const_I_L.append(c_i)
        const_J_L.append(c_j)
        entry_IJ_L.append(entry_ij)

Q = sparse.csc_matrix((entry_IJ_L, (const_I_L, const_J_L)),
    shape=(len(self.constraints_L), len(self.constraints_L)) )
return Q
```

```
def make_tau(self, P, Z):
    rp,cp = P.shape
    rz,cz = Z.shape
    if cp != 3:
        P = P.reshape((len(P)/3, 3))
    if cz != 3:
        Z = Z.reshape((len(Z)/3, 3))

    tau = []

    for pair in self.constraints_L:

        a = pair[0]
        b = pair[1]

        Zab = Z[a,:]-Z[b,:]
        Pab = P[a,:]-P[b,:]
        tau_ij = sp.sum(sp.multiply(Zab, Pab))
        tau.append(tau_ij)

    tau = sp.matrix(tau)
    tau = tau.T
    return tau

def make_dtau(self, P, Z, dEgrad, Egrad ):
    # Ensure arrays are n by 3
    rp, cp       = P.shape
    rz, cz       = Z.shape
    rgrad, cgrad = Egrad.shape
    rdg, cdg     = dEgrad.shape

    if cp !=3:
        P = P.reshape(len(P)/3, 3)
    if cz !=3:
        Z = Z.reshape(len(Z)/3, 3)
    if cgrad !=3:
        Egrad = Egrad.reshape(len(Egrad)/3, 3)
    if cp !=3:
        dEgrad = dEgrad.reshape(len(dEgrad)/3, 3)
    dtau = []
```

```
        for pair in self.constraints_L:
            a = pair[0]
            b = pair[1]
            Zab = Z[a,:]-Z[b,:]
            Pab = P[a,:]-P[b,:]

            dEgrad_ab = dEgrad[a,:] - dEgrad[b,:]
            Egrad_ab = Egrad[a,:]-Egrad[b,:]
            dtau_ab = sp.sum(sp.multiply(dEgrad_ab, Pab)) \
                     + sp.sum(sp.multiply(Egrad_ab, Zab))
            dtau.append(dtau_ab)
        dtau = sp.matrix(dtau)
        dtau = dtau.T
        return dtau

    def projection_mult(self, P, Z):

        tau = self.make_tau(P, Z)
        Q   = self.makeQ(P,P)
        mult = sla.spsolve(Q, tau)
        mult = sp.matrix(mult)
        mult = mult.T


        return mult

    def projection_dmult(self, P, Z, Egrad, dEgrad, mult):

        dtau  = self.make_dtau(P, Z, dEgrad, Egrad)

        dQ = self.makeQ(P, Z)*2
        dQ_mult = dQ * mult
        RHS   = dtau - dQ_mult

        Q = self.makeQ(P,P)
        dmult = sla.spsolve(Q, RHS)
        dmult = sp.matrix(dmult)
        dmult = dmult.T

        return dmult
```

## A.13 The R3n_manifold Class

This class implements functions relevant for enforcing constraints in $\mathcal{C}(\mathbb{R}^{3n})$, for example, projecting the gradient and Hessian, forming the constraint force, and retracting to the constraint manifolds. These function names match their corresponding function names in $\mathcal{C}(\mathbf{S}_{+}^{n,3})$. The optimizer (Section A.15) does not need to known the exact Riemannian manifold (recall matrix manifold optimization algorithms can be implemented for a variety of matrix manifolds).

---

```
from manifold import Manifold
class R3n_manifold(Manifold):
    # initialize with a potential energy and a collision energy
    def __init__(self, potential, collision, constraints_L):
        Manifold.__init__(self, 'R3n Manifold', constraints_L)
        self.potential = potential
        self.collision = collision
    def getName(self):
        return self.name
    def getStartConformation(self):
        return self.potential.getStartConformation()
    def getEndConformation(self):
        return self.potential.getEndConformation()

    # Update which pairs to check for collision
    def updateCollisionCheck(self,pvec):
        self.collision.updateCollisionPairs(pvec)

    def cost(self, pvec, pvec_pre, t):
        return self.potential.cost(pvec, t)+self.collision.cost(pvec)

    def Egrad(self, pvec, pvec_pre, t):
        return self.potential.grad(pvec, t) + self.collision.grad(pvec)

    def grad(self, pvec, pvec_pre, t):
        return self.projection(pvec, self.Egrad(pvec, pvec_pre, t))

    def Hess(self, pvec, pvec_pre, dirvec, t):

        Egrad = self.Egrad(pvec, pvec_pre, t)
```

```
    dEgrad = self.potential.Hess(pvec, dirvec, t)+self.collision.Hess(pvec,
        dirvec)

    mult = self.projection_mult(pvec, Egrad)
    dmult = self.projection_dmult(pvec, dirvec, Egrad, dEgrad, mult)

    dconstraintForce = self.getConstraintForce(mult,
        dirvec)+self.getConstraintForce(dmult, pvec)

    return self.projection(pvec, dEgrad-dconstraintForce)

def getConstraintForce(self, mult, pvec):
    constraint_force = 0*pvec

    for i in range(len(self.constraints_L)):

        pair = self.constraints_L[i]

        multi = sp.asscalar(mult[i,0])


        a = pair[0]
        b = pair[1]
        entry = multi*(pvec[3*a:3*(a+1), :] - pvec[3*b:3*(b+1), :])
        constraint_force[3*a:3*(a+1), :] = constraint_force[3*a:3*(a+1), :]
            + entry
        constraint_force[3*b:3*(b+1), :] = constraint_force[3*b:3*(b+1), :]
            - entry
    return constraint_force
# Project to tangent space of constraint manifold and returns 3n by 1 vector
def projection(self, pvec, z):

    mult = self.projection_mult(pvec, z)
    constraint_force = self.getConstraintForce(mult, pvec)

    return z - constraint_force

# Implements fast projection and returns 3n by 1 vector
def retraction(self, pvecz):
    pveczk = pvecz
```

```
        # use the starting conformation's reference distances
        p0 = self.getStartConformation()

        for k in range(10):

            error = []

            for pair in self.constraints_L:
                a = pair[0]
                b = pair[1]
                qab  = sp.sum(sp.square(pveczk[3*a:3*(a+1),:]-
                    pveczk[3*b:3*(b+1),:]))
                q0ab = sp.sum(sp.square(p0[3*a:3*(a+1),:]- p0[3*b:3*(b+1),:]))
                error.append(0.5*(qab-q0ab))
            error = sp.matrix(error)
            error = error.T

            errornorm = la.norm(error)

            if errornorm < 1e-6:
                break
            #print "norm error ", errornorm
            Q = self.makeQ(pveczk, pveczk)
            mult = sla.spsolve(Q, error)
            mult = sp.matrix(mult)
            mult = mult.T
            dpveczk = self.getConstraintForce(mult, pveczk)
            pveczk = pveczk -dpveczk


        return pveczk

    def inner_product(self, avec, bvec):
        return sp.trace(avec.T*bvec)
```

## A.14 The PSD3_manifold Class

This class implements relevant functions needed for enforcing constraints on $\mathcal{C}(\mathbf{S}_+^{n,3})$.

```
# The rank 3 PSD manifold provides the geometric objects needed
# for optimization algorithms on this manifold.
from manifold import Manifold
class PSD3_manifold(Manifold):
    def __init__(self, potential, collision, constraints_L):

        Manifold.__init__(self, 'PSD3 Manifold', constraints_L)

        self.potential = potential
        self.collision = collision

    # Return the name of this potential.

    def getName(self):
        return self.name

    # Return coordinates of the starting conformation.
    def getStartConformation(self):
        return self.potential.getStartConformation()

    # Return coordinates of the ending conformation.
    def getEndConformation(self):
        return self.potential.getEndConformation()

    # Update which pairs to check for collision
    def updateCollisionCheck(self,P):
        self.collision.updateCollisionPairs(P)

    def cost(self, P, Ppre, t):

        return self.potential.cost(P,t)+self.collision.cost(P)

    def Egrad(self, P, Ppre, t):

        return self.potential.grad(P, t)+self.collision.grad(P)

    def grad(self, P, Ppre, t):

        return self.projection(P, self.Egrad(P, Ppre, t))
```

```
def Hess(self, P, Ppre, Dir, t):

    Egrad = self.Egrad(P, Ppre, t)
    dEgrad = self.potential.Hess(P, Dir, t) + self.collision.Hess(P, Dir)

    mult  = self.projection_mult(P,Egrad)
    dmult = self.projection_dmult(P, Dir, Egrad, dEgrad, mult)

    SS = P.T*P
    AS = P.T*Egrad - Egrad.T*P
    Omega = self.sylvester(SS, AS)

    AS2 = (Dir.T*Egrad - Egrad.T*Dir) + (P.T*dEgrad - dEgrad.T*P) -
        Omega*(Dir.T*P+P.T*Dir) - (Dir.T*P+P.T*Dir)*Omega
    dOmega = self.sylvester(SS, AS2)
    dconstraintforce = self.getConstraintForce(mult,Dir)+
        self.getConstraintForce(dmult,P)

    dprojgrad = dEgrad-P*dOmega-Dir*Omega - dconstraintforce

    return self.projection(P,dprojgrad)


def getConstraintForce(self, mult, P):
    constraint_force = 0*P

    for i in range(len(self.constraints_L)):

        pair = self.constraints_L[i]
        multi = sp.asscalar(mult[i])

        a = pair[0]
        b = pair[1]
        entry = multi*(P[a, :] - P[b, :])
        constraint_force[a,:] = constraint_force[a,:] + entry
        constraint_force[b,:] = constraint_force[b,:] - entry
    return constraint_force
```

```
# Projection to tangent space of constraint manifold and returns n by 3
    matrix.
def projection(self, P, Z):

    SS = P.T*P
    AS = P.T*Z - Z.T*P

    Omega = self.sylvester(SS, AS)

    # construct Q matrix

    mult = self.projection_mult(P,Z)

    constraint_force = self.getConstraintForce(mult,P)

    return Z - P*Omega - constraint_force

# Retracts to constraint manifold and returns n by 3 matrix
def retraction(self, PZ):
    PZk = PZ
    # use the starting conformation's reference distances
    P0 = self.getStartConformation()

    for k in range(10):
        error = []
        for pair in self.constraints_L:
            a = pair[0]
            b = pair[1]
            qab = sp.sum(sp.square(PZk[a,:]-PZk[b,:]))
            q0ab = sp.sum(sp.square(P0[a,:]-P0[b,:]))
            error.append(0.5*(qab - q0ab))

        error = sp.matrix(error)
        error = error.T

        errornorm = la.norm(error)
        if errornorm < 1e-6:
            break

        Q = self.makeQ(PZk, PZk)
```

```
        mult = sla.spsolve(Q, error)
        mult = sp.matrix(mult)
        mult = mult.T

        dPZk = self.getConstraintForce(mult, PZk)
        PZk = PZk -dPZk


    return PZk

def inner_product(self, A, B):
    A = sp.matrix(A)
    B = sp.matrix(B)
    return sp.trace(A.T*B)

# Solve Sylvester equation
def sylvester(self, Sym_mat, Asym_mat):
    Sym = sp.matrix(Sym_mat)
    Asym = sp.matrix(Asym_mat)

    row, col = Sym.shape
    p = col

    V, sig, Vh = la.svd(Sym)
    V= sp.matrix(V)

    O = sp.zeros((p,p))
    O2 = V.T*Asym
    O2 = O2*V

    for i in range(p-1):
        for j in range(i+1, p):
            A = sig[i] + sig[j]
            if not A == 0:
                O[i,j]=O2[i,j]/float(A)
            else:
                O[i,j] = 0
    O = O - O.T
    O = sp.matrix(O)
```

```
        omega = V*O*V.T

        return omega
```

## A.15 The Optimzer Class

The implementation of the trust region algorithm and the tCG algorithm below is based on the implementation from Mishra et al. at https://bamdevmishra.in/codes/edmcompletion/. In the implementation below, the trust region algorithm is using the norm of the gradient as the stopping criterion. This is based on the theoretical result discussed in Section 3.5, see Equation (3.4) and related references. However, this is not the stopping criterion used by Mishra et al.. In their implementation, the stopping criterion is:

```
if accept && k > min_outer && (abs(rhonum) < epsilon || abs(rhonum)/fx_old <
    vepsilon)}
```

This stopping criterion is checking firstly whether the minimum number of iterations has been tried, and then checking if `rhonum`, which is the numerator of $\rho_k$ is small enough. Recall from Equation (3.6), the numerator of $\rho_k$ is defined as the difference between the function at the current iterate, and the function at the proposed iterate. If this numerator is very small, it means the RTR algorithm is not making very significant progress in decreasing the objective function. Therefore, Mishra's stopping criterion terminates once it sees the algorithm has decreased the objective function as much as possible.

```
# The RTR Opt class performs RTR optimizer using
# geometric objects from the manifold.
class RTR_Opt:
    def __init__(self, manifold):

        self.manifold = manifold


    # Given previous conformation xpre at time t-1,
    # find the next conformation at time t.
    def rtr(self, xpre, t):

        # ------------------------------------------------
```

```
#  Default trust region radius parameters

TR_Radius0 = xpre.shape[1] # initial trust region radius
TR_Radius_max = (2**5)*TR_Radius0 # max trust region radius
rho_prime = 0.1 # accept or reject parameter

#-----------------------------------------------------------

# initialize trust region radius
TR_Radius = TR_Radius0

# initialize cost, grad, and norm of grad
x = xpre

self.manifold.updateCollisionCheck(x)

fx = self.manifold.cost(x, xpre, t)
grad = self.manifold.grad(x, xpre, t)

normgrad = la.norm(grad)

tr_iteration = 0 # track number of iterations
max_tr_iteration = 500 # maximum iterations of RTR

accept = 0
reject_count = 0

tcg_iterations_L =[]

while True:

    tr_iteration = tr_iteration + 1


    fx_old = fx

    eta = self.tCG(x, xpre, t, TR_Radius, grad)


    x_prop = self.manifold.retraction(x + eta)
```

```
Heta = self.manifold.Hess(x, xpre, eta, t)

fx_prop = self.manifold.cost(x_prop, xpre, t)

rho_numerator = fx-fx_prop
# rho_denominator is fx - quadratic approximation of fx
rho_denominator = -self.manifold.inner_product(eta, grad)\
               - 0.5*self.manifold.inner_product(eta, Heta)
rho = rho_numerator / rho_denominator

if rho < 0.25:
    # quadratic approximation is inaccurate, decrease radius
    TR_Radius = 0.25*TR_Radius
elif rho > 0.75 and self.manifold.inner_product(eta, eta) ==
    sp.power(TR_Radius, 2):
    # quadratic approximation is accurate, increase radius
    TR_Radius = sp.minimum(2.0*TR_Radius, TR_Radius_max)

# Accept or reject new iterate x_prop
if rho > rho_prime:
    # accept x_prop
    xpre = x
    x =x_prop
    fx = fx_prop
    grad = self.manifold.grad(x, xpre, t)
    normgrad = la.norm(grad)
    accept = 1
    reject_count = 0
else:

    accept = 0
    reject_count = reject_count + 1


if norm_Rgrad < 1e-4:
    break
if tr_iteration > max_tr_iteration :
    # prevent too many iterations
    break
if reject_count > 30:
```

```
            # prevent too many rejections
            break

    return x

# The tCG algorithm for solving the trust region subproblem.
def tCG(self, x, xpre, t, TR_Radius, grad):

    # for evaluating stopping criterion
    kappa = 0.1
    theta = 1

    max_iterations = 500
    min_iterations = 0

    eta = 0.0*grad
    e_e = 0.0
    r = grad
    r_r = self.manifold.inner_product(r, r)
    norm_r = sp.sqrt(r_r)
    norm_r0 = norm_r
    d_d = r_r

    # initial search direction
    delta = -r
    e_d = 0.0
    tcg_iteration = 0

    while True:

        Hd = self.manifold.Hess(x, xpre, delta, t)
        dHd = self.manifold.inner_product(delta, Hd)

        if dHd <=0:
            tau = (-e_d + sp.sqrt(e_d*e_d + d_d *((TR_Radius**2)-e_e)))/d_d
            eta = eta + tau* delta

            break
        alpha = r_r / dHd
        e_e_new = e_e + 2.0*alpha * e_d +alpha * alpha *d_d
```

```
        if e_e_new >= TR_Radius**2:
            tau = (-e_d + sp.sqrt(e_d*e_d + d_d *((TR_Radius**2)-e_e)))/d_d
            eta = eta + tau* delta

            break
        e_e = e_e_new

        eta = eta + alpha * delta


        r = r+ alpha * Hd
        r = self.manifold.projection(x, r)

        r_r_old = r_r
        r_r = self.manifold.inner_product(r,r)
        norm_r = sp.sqrt(r_r)

        # check stopping criteria
        if tcg_iteration > max_iterations:
            break

        if tcg_iteration > min_iterations and norm_r <=
            norm_r0*sp.minimum(norm_r0**theta,kappa):
            break

        # updates

        beta = r_r /r_r_old
        delta = -r + beta*delta
        e_d = beta*(e_d + alpha*d_d)
        d_d = r_r + beta * beta * d_d
        tcg_iteration = tcg_iteration + 1

    return eta
```

## A.16 Running the Interpolation

A main class is needed to run the interpolation. The details are left to the reader. The idea is that instances of the potentials are created and passed to the manifold, this manifold is in turn passed to an optimizer, and the optimizer can then call the manifold to calculate costs, gradients, Hessians in the RTR and tCG algorithms.

# Appendix B

# Derivatives Involving Distance and Quadrance

## B.1 Derivative of Distance

Let $x \in \mathbb{R}^3$,

$$\frac{d \parallel x \parallel}{dx} = \frac{d\sqrt{x^T x}}{dx} = \frac{x}{\sqrt{x^T x}} \, . \tag{B.1}$$

## B.2 Derivative of Quadrance

Let $x \in \mathbb{R}^3$,

$$\frac{d \parallel x \parallel^2}{dx} = \frac{d(x^T x)}{dx} = 2x \, . \tag{B.2}$$

## B.3 Derivative of Distance times Vector

Let $x \in \mathbb{R}^3$ and $I_3$ be the $3 \times 3$ identity matrix. Note that $\parallel x \parallel$ is a scalar so the order of multiplication does not matter. The following expression uses the product rule for taking derivatives:

$$\frac{d(x \parallel x \parallel)}{dx} = \frac{d(\parallel x \parallel x)}{dx} = \parallel x \parallel I_3 + \frac{xx^T}{\parallel x \parallel^2} \, . \tag{B.3}$$

## B.4 Second Order Expansion for the Hookean Potential Energy

Let $d \in \mathbb{R}$ be a scalar and $x, \delta \in \mathbb{R}^3$ be vectors, and the function to be expanded be:

$$f(x) = \frac{1}{2} \left( \| x \| - d \right)^2 \ . \tag{B.4}$$

The second order expansion near $x$ is:

$$f(x + \delta) \approx f(x) + \text{grad} f(x)^T \delta + \frac{1}{2} \delta^T \text{Hess} f(x) \delta \ . \tag{B.5}$$

The constant term $f(x)$ is given by:

$$f(x) = \frac{1}{2} \left( \| x \| - d \right)^2 \ . \tag{B.6}$$

The gradient $\text{grad} f(x)$ is a $3 \times 1$ vector given by:

$$\text{grad} f(x) = \left( \| x \| - d \right) \frac{x}{\| x \|} \ . \tag{B.7}$$

The Hessian $\text{Hess} f(x)$ is a $3 \times 3$ matrix given by, using the product rule on $\text{grad} f(x)$:

$$\begin{aligned} \text{Hess} f(x) &= \left( \| x \| - d \right) \left( \frac{\| x \| \, I_3 - \frac{x x^T}{\| x \|}}{\| x \|^2} \right) + \frac{x x^T}{\| x \|^2} \\ &= I_3 - \frac{d}{\| x \|} \left( I_3 - \frac{x x^T}{\| x \|^2} \right) \ , \end{aligned} \tag{B.8}$$

## B.5 Second Order Expansion for the PSD Potential Energy

Let $d \in \mathbb{R}$ be a scalar and $x, \delta \in \mathbb{R}^3$ be vectors, and the function to be expanded be:

$$f(x) = \frac{1}{2} \left( x^T x - d \right)^2 \ . \tag{B.9}$$

The second order expansion near $x$ is:

$$f(x+\delta) \approx f(x) + \mathrm{grad} f(x)^T \delta + \frac{1}{2}\delta^T \mathrm{Hess} f(x) \delta \ . \tag{B.10}$$

The constant term $f(x)$ is given by:

$$f(x) = \frac{1}{2}\left(x^T x - d\right)^2 \ . \tag{B.11}$$

The gradient $\mathrm{grad} f(x)$ is a $3 \times 1$ vector given by:

$$\mathrm{grad} f(0) = 2\left(x^T x - d\right) x \ . \tag{B.12}$$

The Hessian $\mathrm{Hess} f(0)$ is a $3 \times 3$ matrix given by:

$$\mathrm{Hess} f(0) = 2\left(x^T x - d\right) I_3 + 4 x x^T \ . \tag{B.13}$$

# Appendix C

# The $\mathbf{S}_+^n$ Cone and the EDM Cone

The PSD cone and the EDM cone are both examples of convex cones. This Appendix reviews some basic mathematics relating to convex cones and their faces. The faces of PSD cones are also described. Further reading can be found in [44]. Matrices whose entries are distance rather than quadrance are very similar to EDMs, these matrices have been discussed in for example [24].

## C.1 Introduction to Convexity

Let the blackboard bold font denote an arbitrary Euclidean space, e.g. $\mathbb{E}$, and assume an inner product is defined in $\mathbb{E}$, $\langle \cdot, \cdot \rangle : \mathbb{E} \times \mathbb{E} \to \mathbb{R}$. Recall the following familiar Euclidean spaces:

- $\mathbb{R}^n$, the space of $n$-vectors, with dot product $\langle x, y \rangle = x^T y = \sum_{i=1}^n x_i y_i$ .
- $\mathbb{R}^{m \times n}$, the space of $m \times n$ real matrices, and $\langle A, B \rangle = \text{Trace}(A^T B) = \sum_{i=1}^m \sum_{j=1}^n A_{ij} B_{ij}$.
- $\mathbf{S}^n = \{X \in \mathbb{R}^{n \times n} : X = X^T\}$, the space of $n \times n$ real symmetric matrices, and $\langle A, B \rangle = \text{Trace}(AB) = \sum_{i,j=1}^n A_{ij} B_{ij}$..

Note that $\mathbb{R}$ denotes the set of real numbers, and $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$ denotes the set of real numbers greater than or equal to zero, and $\mathbb{R}_{++} = \{x \in \mathbb{R} : x > 0\}$ denotes the set of real numbers strictly greater than zero.

A set $A \subset \mathbb{E}$ is *affine* if:

$$\alpha x + (1-\alpha)y \in A \quad \forall x, y \in A \quad \alpha \in \mathbb{R} \; . \tag{C.1}$$

The affine hull of a set $S \subset \mathbb{E}$, denoted $\text{aff}(S)$, is the smallest affine set in $\mathbb{E}$ containing $S$. A set $S \subset \mathbb{E}$ is *convex* if:

$$\alpha x + (1-\alpha)y \in S \quad \forall x, y \in S \quad 0 \leq \alpha \leq 1 \; . \tag{C.2}$$

The convex hull of a set $S \subset \mathbb{E}$, $\text{conv}(S)$, is the smallest convex set containing $S$. The unit ball in $\mathbb{E}$ is defined as:

$$\mathbb{B} := \{x \in \mathbb{E} : \| \; x \; \| \leq 1\} \; . \tag{C.3}$$

The ball around $x \in \mathbb{E}$ with radius $\epsilon > 0$ is then given by:

$$x + \epsilon\mathbb{B} = \{y \in \mathbb{E} : \| \; x - y \; \| \leq \epsilon\} \; . \tag{C.4}$$

The relative interior of a convex set $S \subseteq E$ is:

$$\text{relint}(S) := \{x \in \text{aff}(S) : (x + \epsilon\mathbb{B}) \cap \text{aff}(S) \subseteq S, \text{ for some } \epsilon > 0\} \; . \tag{C.5}$$

An alternative definition of the relative interior is given by [44] Theorem 2.8:

$$\begin{aligned} \text{relint}(S) &:= \{x \in S : \forall y \in S, \exists \mu > 1 \text{ such that } \mu x + (1-\mu)y \in S\} \\ &:= \{x \in S : \forall y \in S, \exists \mu > 1 \text{ such that } y + \mu(x-y) \in S\} \; . \end{aligned} \tag{C.6}$$

This definition says that in the relative interior, we can move in the direction $d = x - y$ slightly, and still remain in the set $S$, when $x, y \in S$.

A set $K \subseteq \mathbb{E}$ is a cone if $\mathbb{R}_+ K = K$, where:

$$\mathbb{R}_+ K = \{\alpha x : \alpha \in \mathbb{R}_+, x \in K\} \; . \tag{C.7}$$

Let $\langle \cdot, \cdot \rangle$ denote the inner product in $\mathbb{E}$. The dual cone $K^*$ is defined by:

$$K^* = \{y \in \mathbb{E} : \langle x, y \rangle \geq 0, \forall x \in K\} \; . \tag{C.8}$$

A cone $K$ is self-dual if $K^* = K$.

**Definition C.1.1** (A Face of a Convex Cone)**.** *Let $K \subset \mathbb{E}$ be a convex cone. $F$ is a face*

*of $K$, denoted $F \trianglelefteq K$, if $F$ is a convex cone, $F \subseteq K$, and:*

$$x, y \in K \text{ and } x + y \in F \implies x, y \in F \ . \tag{C.9}$$

*If $F \neq K$, this is denoted by $F \triangleleft K$.*

**Definition C.1.2** (Exposed Face)**.** *A face $F \trianglelefteq K$ is exposed if:*

$$\exists v \in K^*, \ \ F = \{x \in K : \langle v, x \rangle = 0\} = K \cap \{v\}^\perp \ . \tag{C.10}$$

*A cone $K$ is facially exposed if every face $F \trianglelefteq K$ is exposed.*

A face $F \trianglelefteq K$, has an associated conjugate face $F^c \trianglelefteq K^*$ given by:

$$F^c = K^* \cap F^\perp \ . \tag{C.11}$$

Each point in the conjugate face $F^c$ defines an exposed face that contains the face $F$, see [44] Proposition 2.11. Mathematically, let $K$ be a convex cone. If $F \trianglelefteq K$ and $\phi \in F^c$, then

$$F \trianglelefteq K \cap \{\phi\}^\perp \trianglelefteq K \ . \tag{C.12}$$

**Definition C.1.3** (Minimal Face)**.** *The minimal face of a convex cone $K$ containing a set $S \subseteq K$ is denoted face$(S)$ or face$(S, K)$ if $K$ is to be specified explicitly. It is given by:*

$$\mathit{face}(S) = \mathit{face}(S, K) = \bigcap_{S \subseteq F \trianglelefteq K} F \ . \tag{C.13}$$

## C.2 The $\mathbb{R}^n_+$ Cone and its Faces

The nonnegative orthant is defined as:

$$\mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x \geq 0\} \ , \tag{C.14}$$

and the positive orthant is defined as:

$$\mathbb{R}^n_{++} = \{x \in \mathbb{R}^n : x > 0\} \ . \tag{C.15}$$

$\mathbb{R}^n_+$ is a convex cone and also self-dual (Proposition 2.1.1 of [43]), and $\mathbb{R}^n_{++}$ is it's relative interior.

Faces of $\mathbb{R}^n_+$ are given by:

$$F_{\mathcal{I}} = \{x \in \mathbb{R}^n_+ : x_i = 0 \quad \forall i \in \mathcal{I}\} \quad \text{where} \quad \mathcal{I} \subseteq \{1, \ldots, n\} \; . \tag{C.16}$$

That is, a point in $\mathbb{R}^n_+$ has $n$ coordinates: $(x_1, \ldots, x_n)$. The faces of $\mathbb{R}^n_+$ are given by restricting a subset of these $n$ coordinates to be equal to zero. The faces of $\mathbb{R}^n_+$ are essentially $\mathbb{R}^k_+$, for $k \leq n$.

As an example, $\mathbb{R}^3_+$ has the following faces:

- The $\mathbb{R}^2_+$ cones: $\{(x, y)|x \geq 0, y \geq 0\}$, $\{(y, z)|y \geq 0, z \geq 0\}$, and $\{(x, z)|x \geq 0, z \geq 0\}$ are all faces of $\mathbb{R}^3_+$.
- The $\mathbb{R}_+$ cones: $\{x|x \geq 0\}$, $\{y|y \geq 0\}$, and $\{z|z \geq 0\}$ are faces of $\mathbb{R}^3_+$.
- The point $(0, 0, 0)$ is a face of $\mathbb{R}^3_+$.
- The entire cone $\mathbb{R}^3_+$ is a face of $\mathbb{R}^3_+$.

All of the faces of $\mathbb{R}^3_+$ are exposed.

## C.3 The PSD Cone and its Faces

The convex cone of $n \times n$ PSD matrices, $\mathbf{S}^n_+$, is given by:

$$\mathbf{S}^n_+ = \{X \in \mathbf{S}^n : v^T X v \geq 0, \forall v \in \mathbb{R}^n\} \subset \mathbf{S}^n \; . \tag{C.17}$$

The relative interior of $\mathbf{S}^n_+$, denoted $\mathbf{S}^n_{++}$, is given by:

$$\mathbf{S}^n_{++} = \{X \in \mathbf{S}^n : v^T X v > 0, \forall v \neq 0, v \in \mathbb{R}^n\} \; . \tag{C.18}$$

It is well-known that the cone $\mathbf{S}^n_+$ is self-dual, $(\mathbf{S}^n_+)^* = \mathbf{S}^n_+$, see for example Proposition 2.1.1 of [43].

Just as $\mathbb{R}^n_+$ has faces that are $\mathbb{R}^k_+$, where $k \leq n$, the $\mathbf{S}^n_+$ cone has faces that are $\mathbf{S}^k_+$ cones, for $k \leq n$.

A more formal definition is given in Proposition 2.15 from [44], where proofs are also provided. Some key statements from that Proposition are restated below in Proposition C.3.1.

**Proposition C.3.1** (The Faces and Conjugate Faces of PSD Cones)**.** *Let $Q \in \mathbb{R}^{n\times n}$ be an orthogonal matrix. Partition $Q$ as $Q = \begin{pmatrix} U & V \end{pmatrix}$, where $U$ is an $n \times k$ full column rank matrix and $V$ is an $n \times (n-k)$ full column rank matrix. The value of $k$ depends on the application.*

1. *The PSD cone $\mathbf{S}_+^n$ has faces $F \trianglelefteq \mathbf{S}_+^n$ defined by:*

$$F = U\mathbf{S}_+^k U^T = \left\{ Q \begin{bmatrix} A & \mathbf{0}_{k\times(n-k)} \\ \mathbf{0}_{(n-k)\times k} & \mathbf{0}_{(n-k)\times(n-k)} \end{bmatrix} Q^T : A \in \mathbf{S}_+^k \right\} , \tag{C.19}$$

   *where $\mathbf{S}_+^k$ is the cone of $k \times k$ PSD matrices.*

2. *The conjugate face $F^c \trianglelefteq \mathbf{S}_+^n$ is given by:*

$$F^c = V\mathbf{S}_+^{n-k} V^T = \left\{ Q \begin{bmatrix} \mathbf{0}_{k\times k} & \mathbf{0}_{k\times(n-k)} \\ \mathbf{0}_{(n-k)\times k} & B \end{bmatrix} Q^T : B \in \mathbf{S}_+^{n-k} \right\} , \tag{C.20}$$

3. *The relative interior of a face $F$ is given by:*

$$\mathit{relint}(F) = U\mathbf{S}_{++}^k U^T . \tag{C.21}$$

4. *The relative interior of a conjugate face $F^c$ is given by:*

$$\mathit{relint}(F^c) = U\mathbf{S}_{++}^{n-k} U^T . \tag{C.22}$$

   *Note that $F$ and $F^c$ are closely related.*

5. *Any matrix in the conjugate face $F^c$ exposes a face $F$:*
   *$F \trianglelefteq \mathbf{S}_+^n \cap \{Y\}^\perp$, $\forall Y \in F^c$; in addition, $F = \mathbf{S}_+^n \cap \{Y\}^\perp$, $\forall Y \in \mathit{relint}(F^c)$.*

6. *Similar to above, any matrix in the face $F$ exposes a conjugate face $F^c$:*
   *$F^c \trianglelefteq \mathbf{S}_+^n \cap \{X\}^\perp$, $\forall X \in F$; in addition, $F^c = \mathbf{S}_+^n \cap \{X\}^\perp$, $\forall X \in \mathit{relint}(F)$.*
   *$Y$ and $X$ are also called* exposing vectors.

*Proof.* Proofs for each of the above statements are given in the proof for Proposition 2.15 in [44]. □

## C.4 The Euclidean Distance Matrix

Given a set of points, $p_1, \ldots, p_n \in \mathbb{R}^r$, the matrix $D \in \mathbf{S}^n$ whose $(a,b)$-th entry is given by $D_{ab} = q_{ab} = \| p_a - p_b \|^2$ is called the Euclidean distance matrix (EDM)[3, 24, 26, 44]. Note that the EDM's entries are quadrances and not distances.

The *embedding dimension* of an EDM $D$ is the smallest $r$ such that the EDM exists, see [44] Section 2.5.

$$\text{embdim}(D) = \min\{r : \exists p_1, \ldots, p_n \in \mathbb{R}^r \text{ s.t. } D_{ab} = \| p_a - p_b \|^2 \; \forall a, b\} \; . \tag{C.23}$$

Denote the set of $n \times n$ EDMs by $\mathcal{E}^n$. The rank of the Gram matrix is equivalent to the embedding dimension of the corresponding EDM $D$, both are denoted by $r$.

## C.5 The Linear Isomorphism between $\mathbf{S}^n_C \cap \mathbf{S}^n_+$ and $\mathcal{E}^n$

A Gram matrix $X = PP^T$ is *centered* if the atomic coordinates in $P$ have centroid at the origin. This can be expressed as $X\mathbf{1}_n = PP^T\mathbf{1}_n = \mathbf{0}_n$, where $X = PP^T$ is an $n \times n$ PSD Gram matrix, $\mathbf{1}_n$ is the $n \times 1$ matrix of all ones, and $\mathbf{0}_n$ is the $n \times 1$ matrix of all zeros. For an $n \times n$ EDM, $D \in \mathcal{E}^n$, a linear mapping $\mathcal{K}(\cdot)$ [20, 23, 44] relates a centered Gram matrix to its corresponding EDM:

$$\begin{aligned}
D = \mathcal{K}(PP^T) &= \text{diag}(PP^T)\mathbf{1}_n^T + \mathbf{1}_n\text{diag}(PP^T)^T - 2PP^T \\
&= \begin{pmatrix} {p_1}^T p_1 \\ \vdots \\ {p_n}^T p_n \end{pmatrix} \mathbf{1}_n^T + \mathbf{1}_n \begin{pmatrix} {p_1}^T p_1 & \ldots & {p_n}^T p_n \end{pmatrix} - 2 \begin{pmatrix} {p_1}^T p_1 & \ldots & {p_n}^T p_1 \\ \vdots & \ddots & \vdots \\ {p_n}^T p_1 & \ldots & {p_n}^T p_n \end{pmatrix} \\
&= \begin{pmatrix} {p_1}^T p_1 & \ldots & {p_1}^T p_1 \\ \vdots & & \vdots \\ {p_n}^T p_n & \ldots & {p_n}^T p_n \end{pmatrix} + \begin{pmatrix} {p_1}^T p_1 & \ldots & {p_n}^T p_n \\ \vdots & & \vdots \\ {p_1}^T p_1 & \ldots & {p_n}^T p_n \end{pmatrix} - 2 \begin{pmatrix} {p_1}^T p_1 & \ldots & {p_n}^T p_1 \\ \vdots & \ddots & \vdots \\ {p_n}^T p_1 & \ldots & {p_n}^T p_n \end{pmatrix} \\
&= \begin{pmatrix} {p_a}^T p_a + {p_b}^T p_b - 2{p_a}^T p_b \end{pmatrix} = \begin{pmatrix} (p_a - p_b)^T (p_a - p_b) \end{pmatrix} = \begin{pmatrix} q_{ab} \end{pmatrix} \; .
\end{aligned} \tag{C.24}$$

Here, $\text{diag}(\cdot)$ is an operator that extracts the diagonal of the argument matrix and returns the $n \times 1$ vector containing the diagonal entries.

The mapping $\mathcal{K}(\cdot)$ is a *linear isomorphism* from $\mathbf{S}^n_C \cap \mathbf{S}^n_+$ to $\mathcal{E}^n$ [26, 44], where

$$\mathbf{S}^n_C = \{X \in \mathbf{S}^n : X\mathbf{1}_n = \mathbf{0}_n\} \; . \tag{C.25}$$

This linear isomorphism means *both* $\mathbf{S}_C^n \cap \mathbf{S}_+^n$ and $\mathcal{E}^n$ are convex cones.

The mapping $\mathcal{K}(\cdot)$ has a Moore-Penrose pseudoinverse, denoted $\mathcal{K}^\dagger(\cdot)$, given by:

$$\mathcal{K}^\dagger(D) = -\frac{1}{2} J \text{offDiag}(D) J \ , \tag{C.26}$$

where:

$$J = I_n - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n{}^T \ , \tag{C.27}$$

and $I_n$ is the $n \times n$ identity matrix. The operator offDiag$(\cdot)$ is defined as:

$$\text{offDiag}(D)_{ab} = \begin{cases} 0 & \text{if } a = b \\ D_{ab} & \text{if } a \neq b \ . \end{cases} \tag{C.28}$$

Let $\mathcal{A}$ be a linear map $\mathcal{A} : \mathbb{X} \to \mathbb{Y}$, where $\mathbb{X}$ and $\mathbb{Y}$ are Euclidean spaces and let $\langle \cdot, \cdot \rangle$ denote the dot product. The adjoint of $\mathcal{A}$, denoted $\mathcal{A}^*$, is given by:

$$\langle \mathcal{A}x, y \rangle = \langle x, \mathcal{A}^* y \rangle \quad x \in \mathbb{X} \quad y \in \mathbb{Y} \ . \tag{C.29}$$

The mapping $\mathcal{K}(\cdot)$ has an adjoint given by:

$$\mathcal{K}^*(A) = 2(\text{Diag}(A\mathbf{1}_n) - A) \ , \tag{C.30}$$

where $A$ is an $n \times n$ matrix. The diagonal operator, Diag$(\cdot)$, is the operator that takes an $n \times 1$ vector $x$, and returns an $n \times n$ diagonal matrix with the vector entries along the diagonal. The $ab$-th entry of the matrix Diag$(x)$ is:

$$[\text{Diag}(x)]_{ab} = \delta_{ab} x_a \ , \tag{C.31}$$

where $\delta_{ab}$ is the Kronecker delta. For example, for the $n \times 1$ vector $x = (x_1, \ldots, x_n)^T$, we have:

$$\text{Diag}(x) = \begin{pmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{pmatrix} \ . \tag{C.32}$$

Further discussions of $\mathcal{K}(\cdot)$ can be found in [3, 16, 20, 26, 44, 45].

## C.6 EDM Completion and Facial Reduction

The EDMC problem was given in Definition 3.2.1 where the matrix manifold approach for solving this problem was discussed. The study of the EDMC problem also led to the study of facial reduction of PSD matrices, which is a research direction that developed independently of matrix manifold optimization. This Section provides a brief review of some key results.

This Section is organized as follows:

1. Section C.6.1 discusses the original motivation for facial reduction.
2. Section C.6.2 discusses the application of facial reduction to the SNL problem (Definition 3.2.3) formulated by Krislock and Wolkowicz in [44, 45].
3. Section C.6.3 discusses the application of facial reduction to the protein structure-NMR problem (Definition 3.2.2) by Alipanahi et al. [5, 4] ,
4. Section C.6.4 discusses facial reduction in the noisy SNL problem, where the sensors may have noise in their distance measurement with other sensors. The facial reduction introduced in Cheung et al. [16] and Drusvyatskiy et al. [26] is more robust to noise.
5. Section C.6.5 reviews the connection between Rigid cluster ENI [37, 40, 41, 38], which is a modification of ENI to interpolate the rotation and translation of rigid clusters in the protein structure, to facial reduction. We showed in [54] if a protein structure is represented using a Gram matrix, as a point on $\mathbf{S}_+^{n,3}$ instead of $\mathbb{R}^{3n}$, then rigid cluster ENI can be reformulated using facial reduction.

These subsections are only a brief review, the interested reader should consult the original publications for a more detailed discussion. In order to use facial reduction for protein structures, the information about which groups of atoms in a protein structure are assumed to be rigid is required to be given as input, this is a separate problem of itself.

### C.6.1 The Original Motivation for Facial Reduction

Semidefinite optimization problems are one type of convex optimization problem, this is because $\mathbf{S}_+^n$, with no rank constraint, is a convex cone.

In the semidefinite relaxation of the EDMC problem, the feasible point $X$ can have potentially any rank (relaxation means remove the rank constraint), and therefore the

entire cone $\mathbf{S}_+^n$ is the feasible region, provided the specified distances in the input incomplete EDM $\overline{D}$ are satisfied by $X$. In the paper by Alfakih et al.[3], the feasible region for the EDMC problem as a semidefinite optimization problem is given as follows:

$$F = \{X \in \mathbf{S}_+^n \mid \mathcal{K}_{ab}(X) = \overline{D}_{ab} \quad \forall (a,b) \in \mathcal{D}\} \ ,$$

where $\mathcal{K}_{ab}(X)$ is the mapping defined in Equation (4.15).

For convex optimization in general, *Slater's constraint qualification* is a sufficient condition for strong duality, see for example Boyd and Vandenberghe [12] Section 5.2.3 for more information on strong duality. Slater's constraint qualification requires the interior of the feasible region of the optimization problem to have at least one feasible point, also called a *Slater point*, i.e. the interior of the feasible region must be nonempty. For semidefinite optimization problems in particular, knowing whether the problem being modelled satisfies Slater's constraint qualification is not only important for checking whether strong duality holds, but is also a condition required by interior point algorithms to ensure a solution can be found.

A Slater point for the EDMC problem is given by:

$$\exists X \in \mathbf{S}_{++}^n \quad \text{s. t. } \mathcal{K}_{ab}(X) = \overline{D}_{ab} \quad \forall (a,b) \in \mathcal{D} \ ,$$

This definition says that if a full rank, rank $n$, positive definite matrix $X$ exists that is also feasible, then the EDMC problem satisfies Slater's constraint qualification. The full rank matrix $X$ is an "interior point", also called a "Slater point" because it is in the relative interior of the $\mathbf{S}_+^n$ cone.

**Definition C.6.1** (A Clique in the EDM)**.** *A* clique *is defined as a graph where all nodes are adjacent, in other words, all possible undirected edges are present. For an EDM matrix, a clique means the distance between all points have been determined.*

**Definition C.6.2** (Principal Submatrix)**.** *For a $n \times n$ matrix $A$, let $\mathcal{C} \subset \{1, \ldots, n\}$, the matrix $A[\mathcal{C}, \mathcal{C}]$ formed by taking the rows whose indices are in the set $\mathcal{C}$, and the columns whose indices are also in the set $\mathcal{C}$ is called a* principal submatrix.

If the input EDM graph contains subgraphs that are themselves cliques, or equivalently, if the input incomplete EDM, $\overline{D}$, contains principal submatrices that have all entries specified, then the rank of the feasible matrices is decreased. This means the EDMC problem will not have a Slater point $X \in \mathbf{S}_{++}^n$, and Slater's constraint qualification is not satisfied. This result is given as Theorem 4.1 of [44] and Theorem 2.3 [45] for the case of

one clique in the EDM graph. This theorem is called the "Single Clique Facial Reducton" theorem, and is restated below as a definition.

**Definition C.6.3** (Single Clique Facial Reduction)**.** *Let $\overline{D}$ be an incomplete input EDM to the EDMC problem. Suppose the input EDM $\overline{D}$ has one clique, let $\mathcal{C} \subset \{1, \ldots, n\}$ be the set containing the indices of the points in this clique. Since these indices can always be permuted, assume without loss of generality that $\mathcal{C} = \{1, \ldots, q\}$ and $|\mathcal{C}| = q$. Let $P_{\mathcal{C}}$ be the matrix whose rows are the atomic coordinates of the points in clique $\mathcal{C}$. Diagonalize $X_{\mathcal{C}} = P_{\mathcal{C}} P_{\mathcal{C}}{}^T$ using either eigendecomposition or singular value decomposition,*

$$X_{\mathcal{C}} = P_{\mathcal{C}} P_{\mathcal{C}}{}^T = U_{\mathcal{C}} D U_{\mathcal{C}}{}^T.$$

*Let:*

$$\bar{U}_{\mathcal{C}} = \begin{array}{c} \\ q \end{array} \begin{array}{c} \begin{array}{cc} k & 1 \end{array} \\ \begin{pmatrix} U_{\mathcal{C}} & \frac{\mathbf{1}_q}{\sqrt{q}} \end{pmatrix} \end{array}, \tag{C.33}$$

*where $\mathbf{1}_q$ is a $q \times 1$ vector of all ones and $k$ is the embedding dimension of the points. The division by $\sqrt{q}$ ensures that $\bar{U}_{\mathcal{C}}$ is orthonormal.*

*Let:*

$$U = \begin{array}{c} \\ q \\ n-q \end{array} \begin{array}{c} \begin{array}{cc} k+1 & n-q \end{array} \\ \begin{pmatrix} \bar{U}_{\mathcal{C}} & \mathbf{0}_{q \times (n-q)} \\ \mathbf{0}_{(n-q) \times (k+1)} & I_{n-q} \end{pmatrix} \end{array}. \tag{C.34}$$

*Then, the feasible $n \times n$ Gram matrices that solve this EDMC problem are found on the face:*

$$U \mathbf{S}_+^{n-q+k+1} U^T$$

Definition C.6.3 says that if the incomplete EDM has just one clique, Slater's constraint qualification cannot be satisfied because the feasible Gram matrix $X$ that solves the EDMC problem is not in the relative interior $\mathbf{S}_{++}^n$, but rather on a lower dimensional face. Mathematically, this is expressed as:

$$X \in U \mathbf{S}_+^{n-q+k+1} U^T \quad X \notin \mathbf{S}_{++}^n$$

If the input incomplete EDM contains more than one clique, then each clique will define a face that contains the feasible Gram matrices of the EDMC problem, given by Definiton C.6.3. Recall the minimal face was defined in Definition C.1.3, *facial reduction* is the process of intersecting all these faces to find the one unique minimal face containing the

feasible Gram matrices. Suppose there are $l$ cliques:

$$\mathcal{C}_1, \ldots, \mathcal{C}_l \ .$$

Using Definition C.6.3, each clique has a subspace representation given by Equation (C.34):

$$U_1, \ldots, U_l \ .$$

Then, facial reduction finds the subspace $U$ such that:

$$\text{range}(U) = \bigcap_{i=1}^{l} \text{range}(U_i) \ ,$$

where range$(\cdot)$ denote the range of a matrix. Once the matrix $U$ has been found, the EDMC problem can be formulated on a smaller cone. Suppose the resulting $U$ found from facial reduction is an $n \times p$ matrix, then the EDMC problem can be formulated as a semidefinite optimization problem on the smaller cone:

$$U\mathbf{S}_+^p U \ .$$

Slater's constraint qualification will hold for this smaller cone, i.e. the smaller cone will have an interior point that is a feasible Gram matrix to the EDMC problem.

### C.6.2 Cliques in the SNL problem (Krislock and Wolkowicz [44, 45])

In the SNL problem, a sensor is aware of its neighbours up to a threshold distance called the "radio range". There may be many sensors within this radio range, and they all known each other's inter-sensor distances, thus forming a clique in the input EDM graph, causing the Slater's contraint qualification to fail.

The main contribution of Krislock and Wolkowicz [44, 45] is to show that when the SNL problem has *exact distance data*, i.e. no noise in the distance measurements, and the input incomplete EDM graph has cliques (Slater's constraint qualification fails), facial reduction can be used to find a face of the PSD cone where a Slater's point *does* exist. It may even be possible to determine all sensor positions after facial reduction, without the need for a semidefinite optimization solver.

The idea of Krislock and Wolkowicz's algorithm is to start with each sensor as a clique

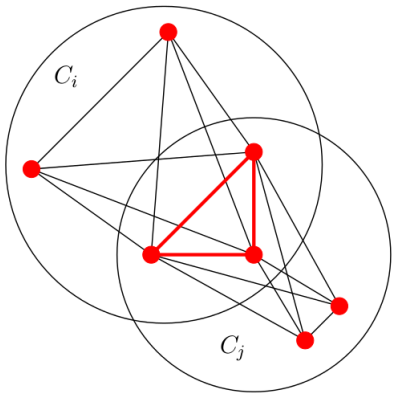



Figure C.1: A rigid intersection between two abstract cliques. Figure 2.2 of [45].

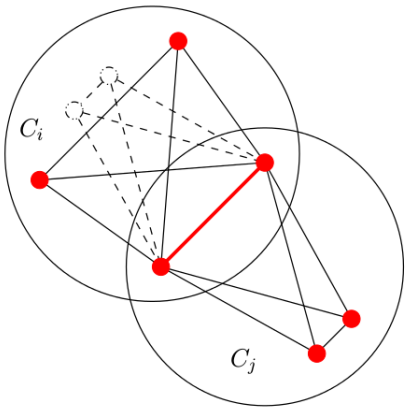



Figure C.2: A non-rigid intersection between two abstract cliques. Figure 2.3 of [45].

of one sensor (a trivial clique), it then attempts to make this clique bigger by checking if there are sensors not in the clique, but is within the radio range of all the sensors in the clique (adjacent to all the sensors in the clique). After the cliques have achieved a maximum size the first step of the algorithm is complete.

The next step of the algorithm is to determine the subspace representations of these cliques using Definition C.6.3.

Thereafter, cliques are checked to see if they have any common sensors, if they do then these cliques intersect. There are two ways two cliques may intersect: *rigidly*, as shown in Figure C.1. "Rigid" intersection means there is only one orientation the two cliques can intersect, as the triangle in the diagram shows. The other way is to intersect *non-rigidly*,

as shown in Figure C.2, the dotted lines show the second possible orientation to intersect. This diagram assumes the points are in 2D space, and so intermediate solutions where one clique is out of the plane, e.g. one clique is oriented 90 degrees with respect to the other clique, is not possible.

Consider two cliques, $U_i$ and $U_j$, and suppose they have a subset of points in common, i.e. they intersect. Let $U_i$ be the subspace of the clique $\mathcal{C}_i$ found using Definition C.6.3, and $U_j$ be the subspace of the clique $\mathcal{C}_j$ also found using Definition C.6.3. Let

$$\mathcal{C} = \mathcal{C}_i \cap \mathcal{C}_j$$

denote the indices of the points in their intersection. Let

$$U_i[\mathcal{C}, :]$$

be the matrix formed from taking the rows of $U_i$ specified in $\mathcal{C}$, and let

$$U_j[\mathcal{C}, :]$$

be the matrix formed from taking the rows of $U_j$ specified in $\mathcal{C}$. $U_i[\mathcal{C}, :]$ and $U_j[\mathcal{C}, :]$ are the subspace representation of the same clique, given by the index set $\mathcal{C} = \mathcal{C}_i \cap \mathcal{C}_j$. This means they should be the same matrix. If they are not, they must differ by a rotation and translation.

If the intersection of these two cliques is rigid, see Figure C.1, a *unique* rotation and translation can be found to rotate one of the matrices, say $U_j$, so that after the rotation $U_i[\mathcal{C}, :]$ and $U_j[\mathcal{C}, :]$ are the same matrix. This idea is explained in more detail in Section 4.8 of [44]. Once the subspace has been rotated, an expression for

$$\text{range}(U) = \text{range}(U_i) \cap \text{range}(U_j)$$

can be found, which leads to finding the Cartesian coordinates of the points in the cliques themselves. The details for this will not be discussed here, see [44].

If the intersection of these two cliques is non-rigid, see Figure C.2, a similar idea of rotating one of the subspaces is used in [44], however, now there are two possible solutions to how the cliques orient with respect to each other, this is given by the dotted lines in Figure C.2. If one of these two solutions can be determined to be the correct solution, for example distance constraints can be checked, then the subspace representation of the face can be found just as in the rigid intersection case and these points can be localized. If the correct solution cannot be determined, a semidefinite solver may be needed.

### C.6.3 Cliques in the Protein Structure-NMR problem (Alipanahi et al. [5, 4])

In [44] the algorithm for localizing sensors with facial reduction was also applied to the protein structure-NMR problem, where the incomplete input EDM has specified entries if the inter-atomic distance is within a distance $R$, the values of $R$ examined were $5\mathring{A}$, $6\mathring{A}$, $7\mathring{A}$, and $8\mathring{A}$.

The protein structure-NMR problem was re-examined by Alipanahi et al.[5, 4]. The definition of the cliques used in this approach is different from the original approach in [44], the cliques were defined using the protein structure's *chemical structures*. This is shown in Figure C.3, where three cliques are presented. The two cliques bounded by a square are peptide planes, and the one clique bounded by a circle is a tetrahedral carbon, the bond angles of a carbon atom has a tetrahedral shape.

Note that all three cliques intersect non-rigidly, compare Figure C.3 with Figure C.2. Protein cliques will always intersect nonrigidly about a bond, since any motion of the protein structure is from rotations about a bond.

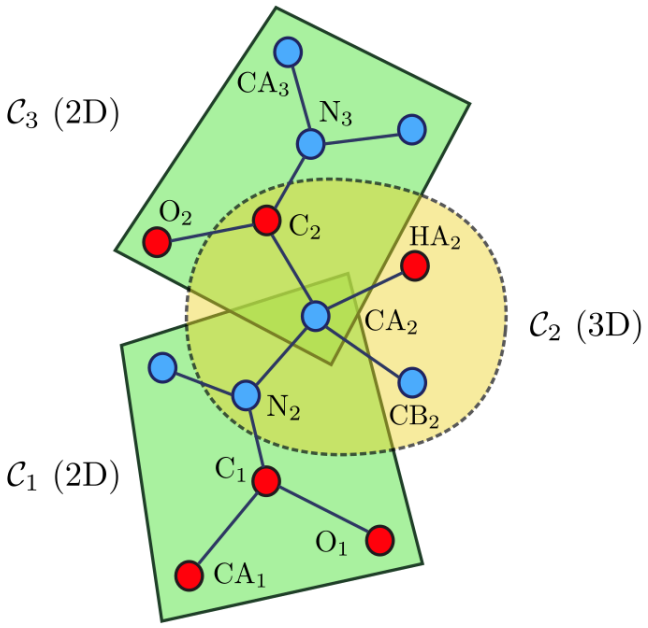



Figure C.3: Protein clique example. Figure 3.13 of (Alipanahi, 2011[4]).

The method used for intersecting cliques by Alipanahi [4] is the *subspace intersection*

*method*, see Theorem 5 of [4] and Algorithm 12.4.3 of [31]. This method is summarize next.

Suppose there are two cliques, whose atomic indices are given by the sets $\mathcal{C}_1$ and $\mathcal{C}_2$. Suppose further without loss of generality these two sets form a consecutive set of integers. In addition, since this discussion only involve two cliques, assume no other cliques in the protein, such that $\mathcal{C}_1 \cup \mathcal{C}_2 = \{1, \ldots, n\}$. This approach is applied recursively to intersect more than two cliques.

Define the following cardinal values:

$$\begin{aligned} q_1 &= |\mathcal{C}_1| \\ q_2 &= |\mathcal{C}_2| \\ q &= |\mathcal{C}_1 \cap \mathcal{C}_2| \\ q_U &= |\mathcal{C}_1 \cup \mathcal{C}_2| \end{aligned} \tag{C.35}$$

Using Definition C.6.3 and Equation (C.34) when applied to these two cliques gives the following subspace representation for the clique $\mathcal{C}_1$:

$$U_1 = \begin{array}{c} \\ q_1 \\ q_U - q \\ n - q_U \end{array} \begin{array}{c} \begin{array}{ccc} k_1 + 1 & q_U - q & n - q_U \end{array} \\ \begin{pmatrix} \bar{U}_{\mathcal{C}_1} & \mathbf{0}_{q_1 \times (q_U - q)} & \mathbf{0}_{q_1 \times (n - q_U)} \\ \mathbf{0}_{(q_U - q) \times (k_1 + 1)} & I_{q_U - q} & \mathbf{0}_{(q_U - q) \times (n - q_U)} \\ \mathbf{0}_{(n - q_U) \times (k_1 + 1)} & \mathbf{0}_{n - q_U \times (k_2 + 1)} & I_{n - q_U} \end{pmatrix} \end{array}, \tag{C.36}$$

the following subspace representations for the clique $\mathcal{C}_2$

$$U_2 = \begin{array}{c} \\ q_U - q \\ q_2 \\ n - q_U \end{array} \begin{array}{c} \begin{array}{ccc} q_U - q & k_2 + 1 & n - q_U \end{array} \\ \begin{pmatrix} I_{q_U - q} & \mathbf{0}_{(q_U - q) \times (k_2 + 1)} & \mathbf{0}_{(q_U - q) \times (q_U - q)} \\ \mathbf{0}_{q_2 \times (q_U - q)} & \bar{U}_{\mathcal{C}_2} & \mathbf{0}_{q_2 \times (n - q_U)} \\ \mathbf{0}_{q_2 \times (q_U - q)} & \mathbf{0}_{n - q_U \times (k_2 + 1)} & I_{n - q_U} \end{pmatrix} \end{array}. \tag{C.37}$$

In order to find a matrix $U$ such that range$(U)$ =range$(U_1) \cap$ range$(U_2)$. Theorem 5 of [4] and Algorithm 12.4.3 of [31], proceeds as follows:

1. Ensure $U_1$ and $U_2$ have orthonormal columns. If not, find the QR-decomposition for $U_1 = Q_1 R_1$ and $U_2 = Q_2 R_2$, and use $Q_1$ and $Q_2$ instead.
2. Form the matrix $C = U_1^T U_2$, and find its singular value decomposition: $C = Y \Sigma Z^T$.
3. Let $Y_{\sigma=1}$ denote the columns of $Y$ where the singular values are 1, then $U = U_1 Y_{\sigma=1}$

is the required matrix $U$ such that range($U$) =range($U_1$)$\cap$ range($U_2$). The number of singular values that are 1 therefore determines the number of columns of the intersected subspace $U$.

The matrix $U$ is now the subspace representation of the clique $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. In order to intersect $U$ with another subspace, $U_3$, with clique indices $\mathcal{C}_3$, the above method is applied recursively. Even if the protein has more than two cliques, only two cliques are intersected at a time and for each intersection, using the same formulas as above.

Note that when intersecting $U_1$ and $U_2$, the last $n - q_U$ rows and columns are the same and therefore, the *smaller* matrices:

$$\widetilde{U}_1 = \begin{array}{c} \\ q_1 \\ q_U - q \end{array} \begin{array}{c} \begin{array}{cc} k_1 + 1 & q_U - q \end{array} \\ \begin{pmatrix} \bar{U}_{\mathcal{C}_1} & \mathbf{0}_{q_1 \times (q_U - q)} \\ \mathbf{0}_{(q_U - q) \times (k_1 + 1)} & I_{q_U - q} \end{pmatrix} \end{array}, \tag{C.38}$$

and

$$\widetilde{U}_2 = \begin{array}{c} \\ q_U - q \\ q_2 \end{array} \begin{array}{c} \begin{array}{cc} q_U - q & k_2 + 1 \end{array} \\ \begin{pmatrix} I_{q_U - q} & \mathbf{0}_{(q_U - q) \times (k_2 + 1)} \\ \mathbf{0}_{q_2 \times (q_U - q)} & \bar{U}_{\mathcal{C}_2} \end{pmatrix} \end{array}. \tag{C.39}$$

can be intersected instead using the above method. Then, suppose the matrix $\widetilde{U}$ is the resulting matrix such that

$$\text{range}(\widetilde{U}) = \text{range}(\widetilde{U}_1) \cap \text{range}(\widetilde{U}_2) \ ,$$

it is expanded as follows to arrive at the subspace representation for the face containing the $n \times n$ Gram matrix

$$U = \begin{array}{c} \\ q_U \\ n - q_U \end{array} \begin{array}{c} \begin{array}{cc} k & n - q_U \end{array} \\ \begin{pmatrix} \widetilde{U} & \mathbf{0}_{q_U \times (n - q_U)} \\ \mathbf{0}_{(n - q_U) \times k} & I_{n - q_U} \end{pmatrix} \end{array}.$$

The number $k$ is the number of columns of the intersected subspace $\widetilde{U}$, this number was already discussed above. See also Algorithm 3 of [4] on finding the number of columns of the intersected subspace.

### C.6.4 Noisy SNL and Cliques (Cheung et al. [16], Drusvyatskiy et al. [26, 25])

The conjugate face can also be used for facial reduction.

The face and conjugate of a PSD cone are defined in Proposition C.3.1. The face is given by:

$$F = U\mathbf{S}_+^k U^T = \left\{ Q \begin{bmatrix} A & \mathbf{0}_{k\times(n-k)} \\ \mathbf{0}_{(n-k)\times k} & \mathbf{0}_{(n-k)\times(n-k)} \end{bmatrix} Q^T : A \in \mathbf{S}_+^k \right\}$$

and the conjuate face is given by:

$$F^c = V\mathbf{S}_+^{n-k} V^T = \left\{ Q \begin{bmatrix} \mathbf{0}_{k\times k} & \mathbf{0}_{k\times(n-k)} \\ \mathbf{0}_{(n-k)\times k} & B \end{bmatrix} Q^T : B \in \mathbf{S}_+^{n-k} \right\} ,$$

In addition, recall the following statement from the same Proposition:

$$F \trianglelefteq \mathbf{S}_+^n \cap \{Y\}^\perp \quad \forall Y \in F^c \quad \text{ in addition } F = \mathbf{S}_+^n \cap \{Y\}^\perp \quad \forall Y \in \text{relint}(F^c).$$

This statement shows the conjugate face can be used to define the face.

Let $\bar{U}_\mathcal{C}$ be the $q \times (k+1)$ matrix given by Equation (C.33). Let $\bar{V} \in \mathbb{R}^{q\times(q-k-1)}$ be the complement subspace, that is

$$\text{range}(\bar{U}_\mathcal{C}) = \text{range}(\bar{V})^\perp.$$

The face given by the primal expression $F = \bar{U}_\mathcal{C}\mathbf{S}_+^{k+1}\bar{U}_\mathcal{C}^T$ can be exposed by $\bar{V}\bar{V}^T \in F^c$. This is mathematically written as:

$$\mathbf{S}_+^q \cap \{\bar{V}\bar{V}^T\}^\perp = \bar{U}_\mathcal{C}\mathbf{S}_+^{k+1}\bar{U}_\mathcal{C}^T \ . \tag{C.40}$$

Since the entire protein structure has $n$ atoms, define the matrix:

$$V = \begin{matrix} \\ q \\ n-q \end{matrix} \begin{matrix} q-k-1 \\ \left( \begin{matrix} \bar{V} \\ \mathbf{0}_{(n-q)\times(q-k-1)} \end{matrix} \right) \end{matrix} . \tag{C.41}$$

Then the exposing vector for the face containing the Gram matrix of all $n$ atoms as defined

just by clique $\mathcal{C}$ is:

$$VV^T = \begin{array}{c} \\ q \\ n-q \end{array} \begin{array}{c} \begin{array}{cc} q & n-q \end{array} \\ \begin{pmatrix} \bar{V}\bar{V}^T & \mathbf{0}_{q\times(n-q)} \\ \mathbf{0}_{(n-q)\times q} & \mathbf{0}_{(n-q)\times(n-q)} \end{pmatrix} \end{array} . \tag{C.42}$$

Let $U$ be defined by equation (C.34), the primal expression for the face $U\mathbf{S}_+^{n-q+k+1}U^T$ is related to the exposing vector $VV^T$ from Equation (C.42) via the following mathematical expression:

$$\mathbf{S}_+^n \cap \{VV^T\}^\perp = U\mathbf{S}_+^{n-q+k+1}U^T \tag{C.43}$$

See for example Theorem 1 of [16], Theorem 3.1 of [25].

The exposing vector for the intersection of two cliques is found from adding up the exposing vectors for both cliques. Consider again the example in Section C.6.3 of two cliques, with their atomic indices given by the sets $\mathcal{C}_1$ and $\mathcal{C}_2$. Define matrices $\bar{V}_1$, $\bar{V}_2$ such that:

$$\text{range}(\bar{U}_{\mathcal{C}_1}) = \text{range}(\bar{V}_1)^\perp$$

and

$$\text{range}(\bar{U}_{\mathcal{C}_2}) = \text{range}(\bar{V}_2)^\perp .$$

The matrix $V_1{V_1}^T$ exposing the face defined by the cliques given by indices $\mathcal{C}_1$ is given by:

$${V_1}{V_1}^T = \begin{array}{c} \\ q_1 \\ n-q \end{array} \begin{array}{c} \begin{array}{cc} q_1 & n-q \end{array} \\ \begin{pmatrix} \bar{V}_1\bar{V}_1^T & \mathbf{0}_{q_1\times(n-q)} \\ \mathbf{0}_{(n-q)\times q_1} & \mathbf{0}_{(n-q)\times(n-q)} \end{pmatrix} \end{array} , \tag{C.44}$$

and the matrix $V_2{V_2}^T$ exposing the face defined by the cliques given by indices $\mathcal{C}_2$ is given by:

$${V_2}{V_2}^T = \begin{array}{c} \\ q_U-q \\ q_2 \\ n-q_U \end{array} \begin{array}{c} \begin{array}{ccc} q_U-q & q_2 & n-q_U \end{array} \\ \begin{pmatrix} \mathbf{0}_{(q_U-q)\times(q_U-q)} & \mathbf{0}_{(q_U-q)\times q_2} & \mathbf{0}_{(q_U-q)\times(n-q_U)} \\ \mathbf{0}_{q_2\times(q_U-q)} & \bar{V}_2\bar{V}_2^T & \mathbf{0}_{q_2\times(n-q_U)} \\ \mathbf{0}_{(n-q_U)\times(q_U-q)} & \mathbf{0}_{(n-q_U)\times q_2} & \mathbf{0}_{(n-q_U)\times(n-q_U)} \end{pmatrix} \end{array} . \tag{C.45}$$

Their sum ${V_1}{V_1}^T + {V_2}{V_2}^T$ exposes the subspace representing the intersection of the two cliques, that is

$$({V_1}{V_1}^T + {V_2}{V_2}^T)\perp\text{range}(U_1) \cap \text{range}(U_2) .$$

If there are $l$ cliques in total, the sum of all their exposing vectors exposes the intersection

of all $l$ clique's subspaces. That is, let

$$VV^T = (V_1{V_1}^T + \cdots + V_l{V_l}^T) \tag{C.46}$$

and let:

$$\text{range}(U) = \text{range}(U_1) \cap \cdots \cap \text{range}(U_l) \ .$$

Then:

$$VV^T \perp \text{range}(U) \ .$$

The matrix $U$ is found from diagonalizing the matrix $VV^T$ given by Equation C.46, then the eigenvectors that corresponds to eigenvalues that are 0 are used to form the columns of $U$.

*The main advantage of this method is it is robust to noise in the distance measurements*, see [16] where this type of facial reduction was applied to the noisy SNL problem.

*The main disadvantage of this method is the final exposing vector, given by Equation (C.46), needs to be diagonalized to recover* $V$, from which the subspace of the intersection of the spaces, $U$ is found.

### C.6.5 Rigid Clusters and ENI [41, 37]

Elastic network interpolation (ENI) as proposed by Kim et al. [37, 40, 41, 38] was reviewed in Chapter 2 Section 2.4. Rigid cluster ENI is a variation of ENI that interpolates the rotation and translations of rigid groups of atoms in a protein structure [41, 37].

This section discusses the connection between facial reduction and rigid cluster ENI. This discussion has also appeared in our previous publication [54].

Suppose the $a$-th atom belongs to the $r$-th clique, let $p_a(t) \in \mathbb{R}^3$ denote its coordinates at an arbitrary time $t$, and let there be $q$ atoms in this clique. Let $t_1 - t_0$ be a *small* time increment. The position of atom $a$ at time $t_1$, $p_a(t_1)$, relative to $p_a(t_0)$ at time $t_0$ is given by:

$$p_a(t_1) = R(\omega_r(t_1 - t_0))\,(p_a(t_0) - c_r(t_0)) + c_r(t_0) + \delta c_r(t_1 - t_0) \ . \tag{C.47}$$

Here, $c_r(t)$ is the $r$-th clique's centroid at time $t$, and $\delta c_r(t_1 - t_0) = c_r(t_1) - c_r(t_0)$ denote the relative translation of the clique's centroid from time $t_0$ to $t_1$. The axis-angle vector for this clique at time $t_0$ is denoted by

$$\omega_r(t_1 - t_0) = \theta(t_1 - t_0)u_r(t_0) \ ,$$

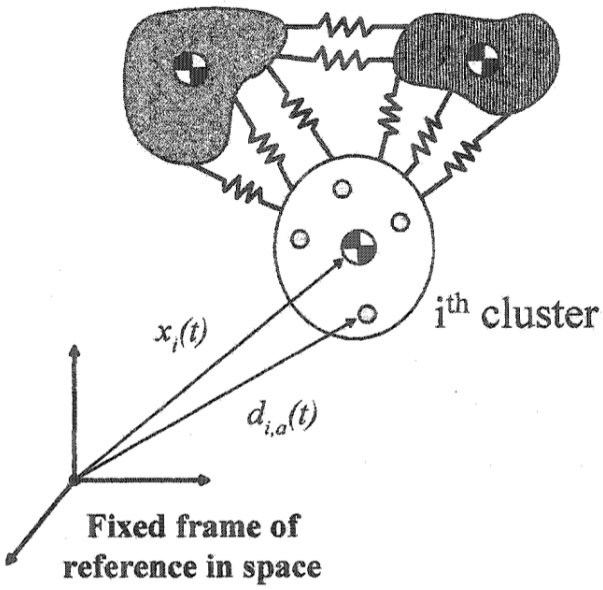



Figure C.4: A schematic diagram of a protein with three rigid clusters (a rigid group of atoms that move concurrently) connected by springs. Figure 6.1 of (Kim, 2004[37]).

where $u_r(t_0) = (x_r(t_0), y_r(t_0), z_r(t_0))^T \in \mathbb{R}^3$ is the axis of rotation, $\parallel u_r(t_0) \parallel = 1$, and $\theta(t_1 - t_0) \in \mathbb{R}$, a scalar, is the counterclockwise angle of rotation from time $t_0$ to time $t_1$. The rotation matrix $R(\omega_r(t_1 - t_0))$ to rotate any point within the $r$-th clique from time $t_0$ to $t_1$ is given by the well-known Rodrigues' rotation formula:

$$\begin{aligned} & R(\omega_r(t_1 - t_0)) \\ & = I + (\sin(\theta(t_1 - t_0))\text{skew}(u_r(t_0)) + (1 - \cos(\theta(t_1 - t_0))\text{skew}(u_r(t_0))^2 \\ & = \exp(\theta(t_1 - t_0)\text{skew}(u_r(t_0))) \\ & = \exp(\text{skew}(\theta(t_1 - t_0)u_r(t_0))) \\ & = \exp(\text{skew}(\omega_r(t_1 - t_0))) \ , \end{aligned} \tag{C.48}$$

where for $\overrightarrow{x} = (x, y, z)^T \in \mathbb{R}^3$, skew$(\overrightarrow{x})$ is defined as:

$$\text{skew}(\overrightarrow{x}) = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} , \tag{C.49}$$

The function $\exp(\cdot)$ is the matrix exponential. Since we assumed $t_1 - t_0$ is a small time

increment, we will also assume the rotation between time $t_0$ and $t_1$ is small, the rotation matrix can then be approximated as:

$$R(\omega_r(t_1 - t_0)) \approx I_3 + \text{skew}(\omega_r(t_1 - t_0)) \ , \tag{C.50}$$

where $I_3$ is the $3 \times 3$ identity matrix. This approximation simplifies equation (C.47) for $p_i(t_1)$ to:

$$\begin{aligned} p_a(t_1) &\approx (p_a(t_0) - c_r(t_0)) + \text{skew}(\omega_r(t_1 - t_0))\,(p_a(t_0) - c_r(t_0)) + c_r(t_0) + \delta c_r(t_1 - t_0) \\ &= p_a(t_0) + \text{skew}(\omega_r(t_1 - t_0))\,(p_a(t_0) - c_r(t_0)) + \delta c_r(t_1 - t_0) \ . \end{aligned} \tag{C.51}$$

Note that for $v, w \in \mathbb{R}^3$, $\text{skew}(w)v = -\text{skew}(v)w$. More concisely, we can express $p_i(t_1)$ as:

$$\begin{aligned} p_a(t_1) &\approx p_a(t_0) - \text{skew}\,(p_a(t_0) - c_r(t_0))\,\omega_r(t_1 - t_0) + \delta c_r(t_1 - t_0) \\ &= p_a(t_0) + H_a(t_0) y(t_1) \end{aligned} \tag{C.52}$$

where:

$$H_a(t_0) = \begin{pmatrix} -\text{skew}\,(p_a(t_0) - c_r(t_0)) & I_3 \end{pmatrix} \in \mathbb{R}^{3\times 6} \ , \tag{C.53}$$

and:

$$y(t_1) = \begin{pmatrix} \omega_r(t_1 - t_0) \\ \delta c_r(t_1 - t_0) \end{pmatrix} \in \mathbb{R}^6 \ . \tag{C.54}$$

The same change $y(t_1)$ is applied to all atoms in the clique, so we can form the matrix expression:

$$\begin{pmatrix} p_1(t_1) \\ \vdots \\ p_q(t_1) \end{pmatrix} \approx \begin{pmatrix} p_1(t_0) \\ \vdots \\ p_q(t_0) \end{pmatrix} + \begin{pmatrix} H_1(t_0) \\ \vdots \\ H_q(t_0) \end{pmatrix} y(t_1) \ . \tag{C.55}$$

Equation (C.55) shows to find the time $t_1$ coordinates from time $t_0$ for the points in the clique, we only need to find the much smaller $y(t_1)$.

### A Second Expression for Rigid Clusters

Consider the transpose of equation (C.51):

$$p_a(t_1)^T \approx p_a(t_0)^T + (p_a(t_0) - c_r(t_0))^T \,\text{skew}(\omega_r(t_1 - t_0))^T + \delta c_r(t_1 - t_0)^T \tag{C.56}$$

Since all atoms that belong to one clique are rotated by the same rotation matrix, $\text{skew}(\omega_r(t_1 - t_0))^T$, and translated by the same vector $\delta c_r(t_1 - t_0)^T$, we can place all $q$ equations into a matrix:

$$\begin{pmatrix} p_1(t_1)^T \\ \vdots \\ p_q(t_1)^T \end{pmatrix} \approx \begin{pmatrix} p_1(t_0)^T \\ \vdots \\ p_q(t_0)^T \end{pmatrix} + \begin{pmatrix} (p_a(t_0) - c_r(t_0))^T & 1 \\ \vdots & \vdots \\ (p_q(t_0) - c_r(t_0))^T & 1 \end{pmatrix} \begin{pmatrix} \text{skew}(\omega_r(t_1 - t_0))^T \\ \delta c_r(t_1 - t_0)^T \end{pmatrix} , \quad \text{(C.57)}$$

Define the matrix $U_r$:

$$U_r = \begin{pmatrix} (p_1(t_0) - c_r(t_0))^T & 1 \\ \vdots & \vdots \\ (p_q(t_0) - c_r(t_0))^T & 1 \end{pmatrix} . \quad \text{(C.58)}$$

$U_r$ is called the *point representation* of the face of the $r$-th clique, see [44] Section 4.12. Analogous to $y(t_1)$, we define $Y(t_1)$ as:

$$Y(t_1) = \begin{pmatrix} \text{skew}(\omega_r(t_1 - t_0))^T \\ \delta c_r(t_1 - t_0)^T \end{pmatrix} . \quad \text{(C.59)}$$

Then we have the matrix expression:

$$P(t_1) = P(t_0) + U_r Y(t_1) . \quad \text{(C.60)}$$

Equation (C.60) shows that when advancing from time $t_0$ to time $t_1$, if we can factor $P(t_0) = U_r S(t_0)$, then equaton (C.60) becomes:

$$P(t_1) = U_r(S(t_0) + Y(t_1)) . \quad \text{(C.61)}$$

Equation (C.61) shows we only need to change $S(t_0)$ to arrive at $P(t_1)$. Although $U_r$ has been defined based on time $t_0$, it can be defined with respect to a reference time, for example $t = 0$.

Facial reduction only requires the face matrix $U_r$ to be found, rotation matrices or axis angle vectors do not need to be introduced explicitly.

# Appendix D

# An Introduction to Riemannian Manifolds

The purpose of this Appendix is to provide enough differential geometry background to define Riemannian manifolds. Further reading can be found in [50, 49, 48, 1, 77, 78]. Formulating the Euler-Lagrange equations on Riemannian manifolds is also discussed in this Appendix. Further reading on classical mechanics and Riemannian manifolds can be found in for example [7, 15].

## D.1 Topological Manifolds

*Topologies* are abstract objects that generalize the notion of open intervals in $\mathbb{R}$. More precisely, a *topology* on a space $\mathcal{M}$ is a collection $\mathcal{T}$, of subsets of $\mathcal{M}$, these subsets are called "open sets". These open sets have the following properties:

1. The subsets $\mathcal{M}$ and $\emptyset$ belong to $\mathcal{T}$.
2. For subsets $U_1, U_2, \cdots \in \mathcal{T}$, their *union* is also in $\mathcal{T}$, $U_1 \cup U_2 \cup \cdots \in \mathcal{T}$, the union may be finite or infinite.
3. For *finitely* many subsets $U_1, U_2, \cdots \in \mathcal{T}$, their intersection is also in $\mathcal{T}$, $U_1 \cap \cdots \cap U_n \in \mathcal{T}$.

Elements of $\mathcal{M}$ are called *points*, denoted $p \in \mathcal{M}$. A *neighborhood* of $p$ is an open subset $U \subset \mathcal{M}$ that contains $p$. Neighborhoods give a feel of "nearness" without the need for a metric. A statement is true "near" a point $p$ if it is true in a neighborhood of $p$.

Open sets also give a convenient definition of *continuity*. A map $f : \mathcal{M} \to \mathcal{N}$ between topological spaces $\mathcal{M}$ and $\mathcal{N}$ is continuous if for every open subset $U \subseteq \mathcal{N}$, its preimage $f^{-1}(U)$ is open in $\mathcal{M}$.

A topological space is *Hausdorff* if distinct points can always be separated by disjoint open sets.

A topological space has a non-unique special collection $\mathcal{B}$ of subsets of $\mathcal{M}$ called a *basis*. This collection has the property that:

1. each $x \in \mathcal{M}$ belongs to at least one element of $\mathcal{B}$
2. if $x \in (B_1 \cap B_2)$ with $B_1, B_2 \in \mathcal{B}$, then there exists $B_3 \in \mathcal{B}$ such that $x \in B_3 \subseteq B_1 \cap B_2$

A topological space is *second countable* if it admits a countable basis.

Two topological spaces $\mathcal{M}$ and $\mathcal{N}$ are *homeomorphic* if there is a bijective map $\varphi : \mathcal{M} \to \mathcal{N}$ such that both $\varphi$ and $\varphi^{-1}$ are continuous functions.

A topological space $\mathcal{M}$ is *locally Euclidean of dimension $n$* if every point of $\mathcal{M}$ has a neighborhood homeomorphic to an open subset of $\mathbb{R}^n$.

The special homeomorphism $\varphi$ that maps $U \subset \mathcal{M}$ to $\widetilde{U} = \varphi(U) \subset \mathbb{R}^n$ is called a (local) *coordinate map*. The pair $(U, \varphi)$ is called a *coordinate chart*. $U$ is called the *coordinate domain*, or *coordinate neighborhood* or *coordinate ball* if $\varphi(U)$ is an open ball in $\mathbb{R}^n$. For a point $p$, the component functions of $\varphi$,

$$x^1 = x^1(p), \ldots, x^n = x^n(p) \tag{D.1}$$

are called the *local coordinates* of $p$.

A *topological $n$-manifold* is a *second countable, Hausdorff space* that is *locally Euclidean* of dimension $n$.

## D.2 Smooth Manifolds

Recall from calculus a function $F$ is *smooth* (denoted $C^\infty$) if each of its component functions has continuous partial derivatives of all orders. A *smooth manifold* is a topological manifold with *extra* structure for deciding which functions on the manifold are smooth.

An obvious way to extend smoothness to manifolds for a function $f : \mathcal{M} \to \mathbb{R}$ is to say $f$ is smooth if and only if $f \circ \varphi^{-1} : \widetilde{U} \to \mathbb{R}$ is smooth, where $\varphi$ is a coordinate map defined above. But the choice of $\varphi$ must be limited to only a collection of *smooth* charts. This collection is the new structure to be defined.

Given a topological manifold and two charts $(U, \varphi)$, $(V, \psi)$ with $U \cap V \neq \emptyset$, the *transition map* from $\varphi$ to $\psi$ is the composite function

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \to \psi(U \cap V)$$

A *diffeomorphism* is a smooth bijective function, with a smooth inverse. If either $U \cap V = \emptyset$ or $\psi \circ \varphi^{-1}$ is a diffeomorphism, then $(U, \varphi)$ and $(V, \psi)$ are *smoothly compatible*.

An *atlas* $\mathcal{A}$ for a topological manifold $\mathcal{M}$ is a collection of charts $(U_\alpha, \varphi_\alpha)$ such that the union of the coordinate domains cover $\mathcal{M}$.

$$\bigcup_\alpha U_\alpha = \mathcal{M}$$

If in addition, any two charts are smoothly compatible then this atlas $\mathcal{A}$ is called a *smooth atlas*.

There may be many atlases that give the same collection of smooth functions. This ambiguity is avoided by defining a *maximal smooth atlas* $\mathcal{A}$, which is a smooth atlas that contains all charts that are smoothly compatible; the maximal atlas is not contained in any strictly larger smooth atlas. A maximal smooth atlas is also called a *smooth structure* or a *differentiable structure*.

A *smooth manifold* is a pair $(\mathcal{M}, \mathcal{A})$, where $\mathcal{M}$ is a topological manifold and $\mathcal{A}$ is a smooth structure.

Any chart $(U, \varphi)$ contained in the maximal smooth atlas is called a *smooth chart*. The coordinate map $\varphi$ is called a *smooth coordinate map*, and $U$ is called a *smooth coordinate domain* or *smooth coordinate neighborhood* or a *smooth coordinate ball* if $\varphi(U)$ is a ball in Euclidean space.

Finally, given this smooth atlas, define a function $f : \mathcal{M} \to \mathbb{R}$ to be smooth if and only if for every $p \in \mathcal{M}$ there exists a smooth chart $(U, \varphi)$ with $p \in U$, and $f \circ \varphi^{-1}$ is smooth for each coordinate chart $(U, \varphi)$. The set of all smooth real-valued functions $f : \mathcal{M} \to \mathbb{R}$ is denoted $C^\infty(\mathcal{M})$. $C^\infty(\mathcal{M})$ is a vector space. The function $\widehat{f} = f \circ \varphi^{-1}$ is called the *coordinate representation* of $f$. This concept of smoothness generalizes easily to maps between manifolds. let $F : \mathcal{M} \to \mathcal{N}$ be a map between manifolds $\mathcal{M}$ and $\mathcal{N}$. For any

$p \in \mathcal{M}$, let $(U, \varphi)$ be a smooth chart containing $p$ and $(V, \psi)$ be a smooth chart containing $F(p)$ such that $F(U) \subset V$. $F$ is smooth if for all $p \in \mathcal{M}$ the composite map $\psi \circ F \circ \varphi^{-1}$ is smooth from $\varphi(U)$ to $\psi(V)$.

Oftentimes, for simplicity, the coordinate domain $U$ is thought of as simultaneously being an open subset of the manifold $\mathcal{M}$ and as an open subset of $\mathbb{R}^n$. As well, the coordinates $(x^1, \ldots, x^n) = \varphi(p)$ are thought of as *being* $p \in U$, called the *coordinate representation*. Similarly, the notation $\widehat{f}$ may also be suppressed when there is no confusion.

# D.3 Geometric Objects

## D.3.1 Tangent space

The tangent space to a manifold at a point is a *linear approximation* of the manifold at that point.

In Euclidean space, $\mathbb{R}^n$, the tangent space at a point $p$ can be thought of as a "copy" of $\mathbb{R}^n$ with the origin at $p$. For example, a tangent plane to a sphere in 3D space, $\mathbb{S}^2 \subset \mathbb{R}^3$, at a point $p$ is a copy of $\mathbb{R}^2$ centered at $p$. This motivates the definition of the *geometric tangent space* to $\mathbb{R}^n$ at a point $p \in \mathbb{R}^n$ to be the set $\mathbb{R}^n_p = \{p\} \times \mathbb{R}^n$ (see Chapter 3 page 61 of [49]):

$$\mathbb{R}^n_p = \{(p, v) : v \in \mathbb{R}^n\} , \tag{D.2}$$

$(p, v)$ may be denoted $v_p$ or $v|_p$. Note that $\mathbb{R}^n_p$ is a vector space:

$$u_p + v_p = (u + v)_p \quad \text{where } u_p, v_p, (u + v)_p \in \mathbb{R}^n_p$$

$$c(v_p) = (cv)_p \quad \text{where } c \in \mathbb{R}$$

There is an *isomorphism* between $\mathbb{R}^n_p$ and $\mathbb{R}^n$, $\mathbb{R}^n_p \cong \mathbb{R}^n$.

This geometric view does not generalize to arbitrary abstract smooth manifolds with no ambient space. For that, the concept of a *directional derivative* is used. Recall a directional derivative is a *linear map* $D_v|_p : C^\infty(\mathbb{R}^n) \to \mathbb{R}$ such that for any $p \in \mathbb{R}^n_p$, in the direction $v$ at $p$ we have:

$$D_v|_p f = D_v f(p) = \left.\frac{d}{dt}\right|_{t=0} f(p + tv) \tag{D.3}$$

This linear operator satisfies the *product rule*

$$D_v|_p (fg) = f(p)\, D_v|_p\, g + g(p)\, D_v|_p\, f \tag{D.4}$$

With this motivation, define a new object, called a *derivation*, which is a linear map $X : C^\infty(\mathbb{R}^n) \to \mathbb{R}$ that satisfies the *product rule*:

$$X(fg) = f(p)Xg + g(p)Xf$$

Let $T_p(\mathbb{R}^n)$ denote the set of all derivations of $C^\infty(\mathbb{R}^n)$ at $p$. $T_p(\mathbb{R}^n)$ is a vector space under the operations:

$$(X + Y)f = Xf + Yf$$
$$(cX)f = c(Xf)$$

**Proposition D.3.1.** *The geometric tangent space $\mathbb{R}^n_p$ is isomorphic to $T_p(\mathbb{R}^n)$ (see [49] Proposition 3.2).*

These derivations are used to define a tangent space.

**Definition D.3.1** (Tangent Space)**.** *The* tangent space *at a point $p$ is the set of all derivations at $p$. This space is a vector space denoted $T_p\mathcal{M}$. An element of $T_p\mathcal{M}$ is called a* tangent vector *at $p$.*

Derivations by themselves are abstract. But they can be represented by something concrete: tangent vectors to curves (see [49] page 75).

Let $J \subset \mathbb{R}$ be an interval, then a *curve* is a smooth map $\gamma : J \to \mathcal{M}$. The tangent vector to $\gamma$ at $t_0$, denoted $\gamma'(t_0)$, is a derivation at $\gamma(t_0)$ that can *act on functions* by:

$$\gamma'(t_0)f = \frac{d}{dt}\bigg|_{t_0}(f \circ \gamma) = \frac{d(f \circ \gamma)}{dt}(t_0) \tag{D.5}$$

Each point $p \in \mathcal{M}$ has its own tangent space, and these spaces are all disjoint. Taking the disjoint union gives a new object called the *tangent bundle*:

$$T\mathcal{M} = \coprod_{p \in \mathcal{M}} T_p\mathcal{M}$$

The tangent bundle comes with a natural projection $\pi : T\mathcal{M} \to \mathcal{M}$ given by $\pi(p, \eta) = p$ which sends every vector $(p, \eta) = \eta_p$ in $T_p\mathcal{M}$ to the point $p$.

**Lemma D.3.1.** *For any smooth $n$-manifold, the tangent bundle is a smooth $2n$-manifold (see Lemma 4.1 [49]).*

### D.3.2 Pushforwards

A tangent space gives a local linear approximation of the manifold. A map $F : \mathcal{M} \to \mathcal{N}$ between two manifolds can also be approximated by a linear map called the *pushforward* between the two tangent spaces, near a point $p \in \mathcal{M}$ and $F(p) \in \mathcal{N}$. The pushforward is a linear map

$$F_* : T_p\mathcal{M} \to T_{F(p)}\mathcal{N}$$

defined by:

$$(F_*\eta)(f) = \eta(f \circ F)$$

The pushforward is also called the *differential* or *total derivative* of a function $F$. For functions mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$, $F : \mathbb{R}^n \to \mathbb{R}^m$, the pushforward is represented by the *Jacobian matrix* of $F$. There are at least two notations for the pushforward:

- $F_*\eta_p$ is favored by authors like Lee [49] when discussing abstract manifolds,
- $DF(p)[\eta_p]$ is favored by authors like Absil [1] when discussing matrix manifolds.

See [49] page 65 to 72 for additional discussions of the pushforward. Some properties of the pushforward are given by Lemma 3.5 of [49], restated below.

**Lemma D.3.2.** *Let $F : \mathcal{M} \to \mathcal{N}$ and $G : \mathcal{N} \to \mathcal{P}$ be smooth maps for manifolds $\mathcal{M}$, $\mathcal{N}$, and $\mathcal{P}$, and $p \in \mathcal{M}$ (see Lemma 3.5 of [49]) .*

1. *$F_* : T_pM \to T_{F(p)}N$ is linear*
2. *$(G \circ F)_* = G_* \circ F_* = T_pM \to T_{G \circ F(p)}P$*
3. *$(Id_M)_* = Id_{T_pM} : T_pM \to T_pM$*
4. *If $F$ is a diffeomorphism, then $F_* : T_pM \to T_{F(p)}N$ is an isomorphism.*

Using the pushforward notation, a tangent vector to a curve $\gamma$ at $t_0 \in J$ is denoted:

$$\gamma'(t_0) = \gamma_* \left( \left. \frac{d}{dt} \right|_{t_0} \right) \in T_{\gamma(t_0)}\mathcal{M} \ , \tag{D.6}$$

and the corresponding linear action is:

$$\gamma'(t_0)f = \gamma_* \left( \left. \frac{d}{dt} \right|_{t_0} \right) f \ . \tag{D.7}$$

**Lemma D.3.3.** *Let $\mathcal{M}$ be a smooth manifold and $p \in \mathcal{M}$. Every $X \in T_p\mathcal{M}$ is the tangent vector to some smooth curve $\mathcal{M}$ (see Lemma 3.11 of [49]).*

As a consequence of Lemma D.3.3, $F_*X$ can be computed for any $X \in T_p\mathcal{M}$ and smooth map $F : \mathcal{M} \to \mathcal{N}$ by finding a smooth curve $\gamma$ such that $X = \gamma'(0)$

$$F_*X = F_*(\gamma'(0)) = F_* \circ \gamma_* \left( \frac{d}{dt}\bigg|_{t_0} \right) = (F \circ \gamma)_* \left( \frac{d}{dt}\bigg|_{t_0} \right) = (F \circ \gamma)'(0) \tag{D.8}$$

The pushforward $F_*$ can be used to determine the *rank* of $F$.

**Definition D.3.2** (Rank of a Differentiable Function)**.** *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ denote two manifolds with dimension $d_1$ and $d_2$ respectively. Consider a differentiable function $F : \mathcal{M}_1 \to \mathcal{M}_2$ with charts $\varphi_1$ and $\varphi_2$ for $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively. Let the coordinate representation of $F$ be $\hat{F} = \varphi_2 \circ F \circ \varphi_1^{-1}$. The* rank *of $F$ is the dimension of the range of $D\hat{F}(\varphi_1(x))[\cdot] : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$. $D\hat{F}(\varphi_1(x))$ is the differential of $\hat{F}$ at $\varphi(x)$.*

**Definition D.3.3** (Immersions and Submersions)**.** *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ denote two manifolds with dimension $d_1$ and $d_2$ respectively.*

- *$F$ is an* immersion *if its rank is $dim(\mathcal{M}_1) = d_1$; immersions are locally like injective linear maps.*
- *$F$ is a* submersion *if its rank is $dim(\mathcal{M}_2) = d_2$; submersions are locally like surjective linear maps.*

## D.4 Examples of smooth manifolds

The following are some objects that satisfy the definition of a smooth manifold. These examples are taken from [49] p 17.

- (Lee[49] Example 1.13) $\mathbb{R}^n$ is a smooth $n$-manifold with the atlas consisting of just a single chart $(\mathbb{R}^n, Id_{\mathbb{R}^n})$. This is called the *standard smooth structure*.
- (Lee[49] Example 1.14) Let $\psi : \mathbb{R} \to \mathbb{R}$ be $\psi(x) = x^3$. The atlas of the single chart $(\mathbb{R}, \psi)$ is a smooth structure on $\mathbb{R}$ This chart is not smoothly compatible with the standard smooth structure $Id_{\mathbb{R}}$ because the transition map $Id_{\mathbb{R}} \circ \psi^{-1}(x) = x^{1/3}$ is not smooth at the origin.

- (Lee[49] Example 1.15) A *finite-dimensional vector space* $V$ with any norm determines a topology. $V$ is a smooth manifold where the smooth structure is the atlas with the single chart $(V, E^{-1})$, where $E : \mathbb{R}^n \to V$ is a basis isomorphism given by:

$$E(x) = \sum_{i=1}^{n} x^i E_i \ , \tag{D.9}$$

for any ordered basis $(E_1, \ldots, E_n)$ of $V$. This map is a homeomorphism and so the single chart $(V, E^{-1})$ is a smooth structure.

The same element $E(x) \in V$ can be expressed in a *different* ordered basis $(\widetilde{E}_1, \ldots, \widetilde{E}_n)$ so that $\widetilde{E}(\widetilde{x}) = E(x) \in V$. Let $E_i = \sum_j A_i^j \widetilde{E}_j$ where $A = (A_i^j)$ is an invertible matrix. This means:

$$\sum_{j=1}^{n} \widetilde{x}^j \widetilde{E}_j = \sum_{i=1}^{n} x^i E_i = \sum_{j=1}^{n} \sum_{i=1}^{n} x^i A_i^j \widetilde{E}_j \tag{D.10}$$

So it follows:

$$\widetilde{x}^j = \sum_{i=1}^{n} x^i A_i^j \tag{D.11}$$

The transition map between these two charts is:

$$\widetilde{E}^{-1} \circ E(x) = \widetilde{x} = (\widetilde{x}^1, \ldots, \widetilde{x}^n) = (\sum_i A_i^1 x^i, \ldots, \sum_i A_i^n x^i) = Ax \tag{D.12}$$

So the map from $x$ to $\widetilde{x}$ is an invertible linear map making the two charts smoothly compatible. Thus, all bases determine the same smooth structure.

- (Lee[49] Example 1.16, Absil [1] Section 3.1.5) The space of $n \times m$ matrices with real entries, denoted $M(n \times m, \mathbb{R})$ or $\mathbb{R}^{n \times m}$ [1] is a vector space under matrix addition and scalar multiplication, and so is a smooth manifold of dimension $n \times m$. When $m = n$, the notation $M(n, \mathbb{R})$ is also used. Note the notation $\mathbb{R}^{nm}$ without the $\times$ symbol means the space of $mn \times 1$ vectors.

- (Lee[49] Example 1.17) When $\mathcal{M}$ is a smooth $n$-manifold, any open subset $U \subset \mathcal{M}$ is a smooth manifold called the open submanifold of $\mathcal{M}$ with atlas given by:

$$\mathcal{A}_U = \{\text{smooth charts } (V, \varphi) \text{ for } M \text{ such that } V \subset U\}.$$

- (Lee[49] Example 1.18) The *general linear group*, denoted $\mathbf{GL}_n$, is the set of *invertible*

*$n \times n$ matrices.* This set is a smooth $n^2$-manifold since it is an open subset of $M(n, \mathbb{R})$, the set where the determinant is nonzero.

## D.5 Riemannian Manifolds

A *Riemannian metric* is a 2-tensor field that is symmetric, $g(X, Y) = g(Y, X)$, and positive definite at each point, $g(X, X) > 0$ if $X > 0$. This metric is defined on the tangent space at point $p$, so $X$ and $Y$ are tangent vectors on this tangent space.

A *Riemannian manifold* $(\mathcal{M}, g)$ is a smooth manifold $\mathcal{M}$ with a Riemannian metric $g(\cdot, \cdot)$ defined on the tangent space of every point $p$ on the smooth manifold.

### D.5.1 The Exponential Map

Exponential maps are defined for an abstract Riemannian manifold $\mathcal{M}$ as follows. Let $\gamma : I \to \mathcal{M}$, where $I \subset \mathbb{R}$, be a curve on the manifold $\mathcal{M}$. See Section D.3.1 for the definition of a curve on a manifold. Define a *geodesic* to be a curve on a manifold $\gamma$ with zero acceleration. This definition depends on the Riemannian connection $\nabla$. Denote the geodesic with intial point $p$ and initial velocity $V$ by $\gamma_V$. Let $T\mathcal{M}$ denote the tangent bundle of $\mathcal{M}$, see Section D.3.1 for the definition. Define the set:

$$U = \{V \in T\mathcal{M} : \gamma_V \text{ is defined on an interval containing } [0, 1]\} .$$

Then, the exponential map, is the map $\text{Exp} : U \to \mathcal{M}$ defined by:

$$\text{Exp}(V) = \gamma_V(1) . \tag{D.13}$$

Equation (D.13) means for a point on $p \in \mathcal{M}$, the exponential map determines the point on $\mathcal{M}$ away from $p$, in the direction $V \in T_p\mathcal{M}$, where the geodesic has travelled by time $t = 1$, $\gamma_V(1) \in \mathcal{M}$.

## D.6 Matrix Manifolds

A matrix manifold is a manifold as defined in classical differential geometry, each *element* of the manifold can be represented by a *matrix*. Optimization algorithms in $\mathbb{R}^n$ work with inner products and length. Therefore, a Riemannian metric on the tangent space of the

matrix manifold needs to be defined when generalizing optimization algorithms in $\mathbb{R}^n$ to matrix manifold. Thus, matrix manifolds are also Riemannian manifolds.

Some examples of matrix manifolds are presented below.

### D.6.1 $\mathbb{R}^{3n}$

Recall from the above discussion in Section D.4 that $\mathbb{R}^n$, or equivalently $\mathbb{R}^{3n}$, is a smooth manifold,

For an arbitrary point $\overrightarrow{p} \in \mathbb{R}^{3n}$, the tangent space at $\overrightarrow{p}$ is another copy of $\mathbb{R}^{3n}$ with the origin at $\overrightarrow{p}$, see for example [49] Chapter 3. The dot product for $\mathbb{R}^{3n}$ can be defined to be the Riemannian metric on the tangent space. The smooth manifold $\mathbb{R}^{3n}$ with a Riemannian metric defined on all its tangent spaces is a Riemannian manifold.

Note that $\mathbb{R}^{3n}$ is also a matrix manifold since each point is represented by a $3n \times 1$ matrix. No constraints are required to describe $\mathbb{R}^{3n}$ and $\mathbb{R}^{3n}$ also has no quotient structure.

### D.6.2 $\mathbb{R}^{n\times m}$

The set of $n \times m$ matrices with real entries, $\mathbb{R}^{n\times m}$, discussed above (Lee[49] Example 1.16) is a matrix manifold. $\mathbb{R}^{n\times m}$ is a $nm$-dimensional vector space, any vector space is a smooth manifold as discussed above.

Define $\varphi(Y) = \text{vec}(Y)$ to be the function that stack the columns of $Y$ below each other to form an $nm \times 1$ vector. Any matrix $Y \in \mathbb{R}^{n\times m}$ can be turned into an $nm \times 1$ vectors $y \in \mathbb{R}^{nm}$, using the function $\varphi(Y) = \text{vec}(Y)$. Thus, $\varphi(Y)$ defines a chart for $\mathbb{R}^{n\times m}$.

For two matrix $A$, $B$ on the tangent space of a point $Y$, the inner product (Euclidean metric):

$$\text{Trace}(A^T B) = \text{vec}(A)^T \text{vec}(B)$$

is defined as the Riemannian metric, making $\mathbb{R}^{n\times m}$ a Riemannian manifold.

Two ways to construct matrix manifolds from $\mathbb{R}^{n\times m}$ are by the operations of taking *embedded submanifolds* and *quotient manifolds* of this space.

If $\mathcal{M}$ is an embedded submanifold of $\mathbb{R}^{n\times m}$, then $\mathbb{R}^{n\times m}$ is referred to as the *embedding space*. If $\mathcal{M}$ is a quotient manifold of $\mathbb{R}^{n\times m}$, then $\mathbb{R}^{n\times m}$ is referred to as the *total space*. Following the convention in [1] page 29, both an embedding space and a total space can also be called a *structure space*.

### D.6.3 Embedded Submanifolds of $\mathbb{R}^{n\times m}$

Let $\mathcal{M}$ be a manifold with topology $\mathcal{T}$. Let $\mathcal{N} \subset \mathcal{M}$. The subspace topology $\mathcal{T}_{\mathcal{N}}$ is defined as:

$$\mathcal{T}_{\mathcal{N}} = \{U \subset \mathcal{N} : U = \mathcal{N} \cap V \text{ where } V \in \mathcal{T}\} .$$

This expression means the open subsets of in the topology $\mathcal{T}_{\mathcal{N}}$ are the open subsets of the bigger space $\mathcal{M}$ intersected with $\mathcal{N}$.

Let $(\mathcal{M}, \mathcal{A})$ and $(\mathcal{N}, \mathcal{B})$ be manifolds and their respective maximal atlases. If $\mathcal{N} \subset \mathcal{M}$, and the inclusion map $i : \mathcal{N} \to \mathcal{M} : x \to x$ is an immersion, then $(\mathcal{N}, \mathcal{B})$ is an *immersed submanifold*. If the manifold topology of $\mathcal{N}$ coincides with the subspace topology induced from the topological space $\mathcal{M}$, then $\mathcal{N}$ is called an *embedded submanifold*.

Embedded submanifolds of $\mathbb{R}^{n\times m}$ can be described by constraint sets on the space $\mathbb{R}^{n\times m}$.

#### The Compact and Non-compact Stiefel Manifolds are Embedded Submanifold of $\mathbb{R}^{n\times m}$

Both the compact and non-compact Stiefel manifolds are embedded submanifolds of $\mathbb{R}^{n\times m}$.

Assume $m \leq n$. The compact Stiefel manifold, denoted $St(m, n)$, is the set of all $n \times m$ orthonormal matrices. It is described by the following constraint set:

$$St(m, n) = \left\{Y \in \mathbb{R}^{n\times m} : Y^T Y = I_m\right\} = \left\{Y \in \mathbb{R}^{n\times m} : \det(Y^T Y) = 1\right\} , \tag{D.14}$$

where $\det(\cdot)$ takes the determinant of the argument.

When $m = 1$, $St(m, n)$ reduces to the unit sphere. When $m = n$, $St(m, n)$ becomes the orthogonal group $\mathbf{O}_n$.

The noncompact Stiefel manifold $\mathbb{R}_*^{n\times m}$ (Absil Section 3.1.5 [1] and Lee Example 1.19 [49]) is the set of $n \times m$ full rank matrices. The constraint describing this manifold is given by:

$$\mathbb{R}_*^{n\times m} = \left\{Y \in \mathbb{R}^{n\times m} : \det(Y^T Y) \neq 0\right\} \tag{D.15}$$

When $m = 1$, $\mathbb{R}_*^{n\times m}$ reduces to the Euclidean space $\mathbb{R}^n$ with the original removed, also known as the *real projective space*. When $m = n$, it becomes the general linear group $\mathbf{GL}_n$.

### D.6.4 Quotient Manifolds of $\mathbb{R}^{n\times m}$

Let $\mathcal{M}$ be a Riemannian manifold. Let $\sim$ be an equivalence relation defined between points on $\mathcal{M}$.

**Definition D.6.1** (Equivalence Relation)**.** *For a manifold $\mathcal{M}$, an equivalence relation, $\sim$, is a relation between points on the manifold $\mathcal{M}$ with the following properties:*

- *Reflexive: $x \sim x$, $\forall x \in \mathcal{M}$.*
- *Symmetric: $x \sim y$ if and only if $y \sim x$, $\forall x, y \in \mathcal{M}$.*
- *Transitive: if $x \sim y$ and $y \sim z$, then $x \sim z$, $\forall x, y, z \in \mathcal{M}$.*

An equivalence class is a set where all elements have an equivalence relationship with each other:

$$[x] = \{y \in \mathcal{M} : x \sim y\}$$

The quotient of $\mathcal{M}$ by $\sim$ is the set of equivalence classes, and is denoted $\mathcal{M}/\sim$:

$$\mathcal{M}/\sim = \{[x] : x \in \mathcal{M}\} .$$

If the *natural projection* $\pi : \mathcal{M} \to \mathcal{M}/\sim$ given by:

$$\pi(x) = [x] ,$$

is also a submersion, then $\mathcal{M}/\sim$ is a *quotient manifold*. Each of the equivalence classes $[x]$ are an embedded submanifold of $\mathcal{M}$ (see Proposition 3.4.4 of [1]).

Figure D.1 (taken from Figure 2.5 of [78]) is a schematic diagram representing a quotient manifold. In this Figure, $\mathcal{M}$ is the structure space, $\mathcal{M}/\sim$ is the quotient manifold, $[x] = \pi^{-1}\pi(x)$ is an equivalence class, with $x$ being a representative element of the equivalence class. The equivalence class $[x]$ when regarded as a subset of $\mathcal{M}$ is called a *fiber*. A point on the quotient manifold is denoted $\pi(x)$.

In applications of matrix manifolds, *any* matrix from an equivalence class can be used to represent that equivalence class. Calculations relating to the quotient matrix manifold are expressed in terms of these "representative matrices".

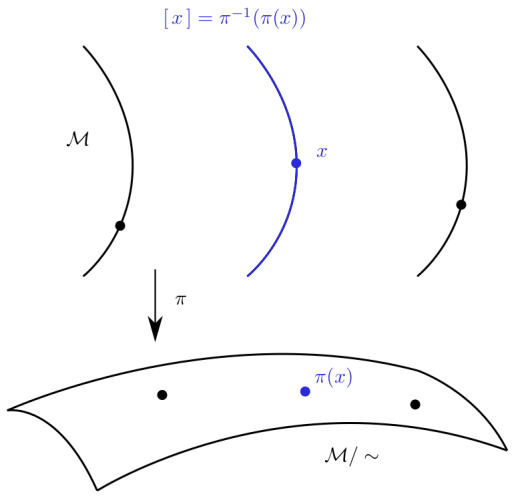



Figure 2.5: A quotient space $\mathcal{M}/\sim$ and its fibers $\pi^{-1}(\pi(x))$.

Figure D.1: A schematic diagram of a quotient manifold. Taken from Figure 2.5 of [78].

**The Grassmann manifold is a Quotient Manifold of $\mathbb{R}^{n\times m}$**

The Grassmann manifold $Grass(m,n)$ is the set of all $m$-dimensional subspaces of $\mathbb{R}^n$ (Section 3.4.4 [1]). For $n \times m$ matrices $X$ and $Y$, if $Y = XM$ where $M \in \mathbf{GL}_m$, then they describe the same $m$-dimensional subspace. That is $span(Y) = span(X)$. Any two matrices that can be expressed by such a relation are in the same equivalence class. Each of these equivalence classes is one element of the Grassmann manifold. Thus, $Grass(m,n)$ is a quotient manifold with the quotient structure [1]:

$$Grass(m,n) \simeq \mathbb{R}_*^{n\times m}/\mathbf{GL}_m \tag{D.16}$$

### D.6.5 The rank $r$ Positive Semidefinite Matrix Manifold $\mathbf{S}_+^{n,r}$

All matrices in the relative interior, $\mathbf{S}_{++}^n$, have full rank $n$. The boundary of $\mathbf{S}_+^n$ is given by:

$$\mathbf{S}_+^n \setminus \mathbf{S}_{++}^n = \bigcup_{r=0}^{n-1} \mathbf{S}_+^{n,r} \ , \tag{D.17}$$

where:

$$\mathbf{S}_+^{n,r} = \{X \in \mathbf{S}_+^n : \text{rank}(X) = r\} \ . \tag{D.18}$$

The boundary contains all the rank deficient matrices in $\mathbf{S}_+^n$. The notation $\mathbf{S}_+^{n,n}$ is the same as $\mathbf{S}_{++}^n$

It is also well known that $\mathbf{S}_{++}^n$ is a Riemannian manifold [59]. It was shown in [78] Section 3.2 that $\mathbf{S}_+^{n,r}$ is a smooth manifold embedded in the set of $n \times n$ real matrices, $\mathbb{R}^{n \times n}$, for every $r < n$.

The fixed rank PSD matrix manifold *does not* have a natural geometry. This was discussed in [78] Section 6.6, where various geometries $\mathbf{S}_+^{n,r}$ is diffeomorphic to has been presented (see page 17 of [78], where the symbol $\simeq$ is used to denote diffeomorphism between two manifolds). Two quotient geometries that $\mathbf{S}_+^{n,r}$ is diffeomorphic to are given below.

### The quotient manifold $\mathbf{S}_+^{n,r} \simeq (St(r,n) \times \mathbf{S}_+^{r,r})/\mathbf{O}_r$

A positive semidefinite matrix $X$ can be factored as $X = USU^T$, where $U \in St(r,n)$ is an element of the compact Stiefel manifold, and $S \in \mathbf{S}_+^{r,r}$ is a $r \times r$ positive definite matrix. This factorization is unique up to the action of the orthogonal group: $U \to UQ$, $S \to Q^T S Q$ for $Q \in \mathbf{O}_r$. See [11] for further discussions.

### The quotient manifold $\mathbf{S}_+^{n,r} \simeq \mathbb{R}_*^{n \times r}/\mathbf{O}_r$

A positive semidefinite matrix $X$ can be factored as $X = YY^T$, where $Y \in \mathbb{R}_*^{n \times r}$. This factorization is unique up to the action of the orthogonal group: $Y \to YQ$, where $Q$ is an $r \times r$ orthogonal matrix, $Q \in \mathbf{O}_r$. This geometry has appeared in [34, 58].

The atomic coordinates of a protein can be represented by a $3n \times 1$ vector $\overrightarrow{p}$ when the Riemannian manifold $\mathbb{R}^{3n}$ is being used to model ENMs.

Alternatively, the atomic coordinates can be represented by an $n \times 3$ matrix $P$, or equivalently the Gram matrix $PP^T$, when the Riemannian manifold $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n \times r}/\mathbf{O}_r$ is being used to model ENMs.

Define the Riemannian metric on the tangent space at $P$ to be given by $\text{Trace}(X^T Y)$, where $X$ and $Y$ are $n \times 3$ matrices on the tangent space at $P$, and $\text{Trace}(\cdot)$ takes the trace of the argument. The manifold $\mathbf{S}_+^{n,3} \simeq \mathbb{R}_*^{n \times r}/\mathbf{O}_r$ together with this metric has the structure of a Riemannian manifold.

## D.7 Riemannian Manifolds and Classical Mechanics

The study of the equations of motion for classical mechanics on Riemannian manifolds can be found in various sources, two examples are [7, 15].

In classical mechanics, the motion of a particle (or particle system) is completely described by its position, $p$, and velocity $v$. The position $p$ belongs to a space called the coordinate space, which is a Riemannian manifold $\mathcal{M}$. The space of position and velocity, $(p, v)$ is called the phase space, which is identified with the tangent bundle $T\mathcal{M}$.

**Definition D.7.1.** *The* potential energy *is defined as a differentiable function mapping from the Riemannian manifold $\mathcal{M}$ to $\mathbb{R}$, $U : \mathcal{M} \to \mathbb{R}$.*

**Definition D.7.2.** *Let $\mathcal{M}$ be a Riemannian manifold. The* kinetic energy *at a point $p$ is given by:*

$$T(\xi) = \frac{1}{2} g(\xi, \xi) \ , \quad \xi \in T_p\mathcal{M} \ , \tag{D.19}$$

*where $g(\cdot, \cdot)$ is the Riemannian metric defined in Section D.5.*

Consider an abstract particle with position $p(t) \in \mathcal{M}$, its velocity is given by $v(t) = \dot{p}(t) \in T_{p(t)}\mathcal{M}$, where the dot indicates a time derivative. The *Lagrangian* is a function from the tangent bundle to the real numbers, $L : T\mathcal{M} \to \mathbb{R}$, given by:

$$L(p(t), \dot{p}(t)) = \frac{1}{2} g(\dot{p}(t), \dot{p}(t)) - U(p(t)) \ . \tag{D.20}$$

The equation of motion of this particle is given:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{p}(t)} = \frac{\partial L}{\partial p(t)} \ . \tag{D.21}$$

# Appendix E

# Miscellaneous Mathematics

## E.1 Solving the Sylvester Equation via Diagonalization

Let $A$, $B$, $C$, and $X$ be matrices with the appropriate shape. The Sylvester equation given by:

$$AX - XB = C$$

has a solution $X$, if an only if A and B do not have any eigenvalues in common, see for example [72, 8]. The Sylvester equation that appears in this thesis has the form:

$$(Y^T Y)\Omega + \Omega(Y^T Y) = Z \ , \tag{E.1}$$

where $Y$ is an $n \times r$ matrix, $\Omega$ is an $r \times r$ skew symmetric matrix, and $Z$ is an $r \times r$ skew symmetric matrix. Note that $A = Y^T Y$ and $B = -A$, therefore $A$ and $B$ will not have any eigenvalues in common.

To solve this form of the Sylvester equation [1], first diagonalize $Y^T Y$ (e.g. using singular value decomposition (SVD)) as $Y^T Y = V \text{Diag}(\sigma) V^T$, where $\text{Diag}(\sigma)$ is a diagonal matrix with the singular values along the diagonal, and $V$ is an $n \times r$ matrix such that $VV^T = I_n$, $V^T V = I_r$. Equation (E.1) can now be rewritten as:

$$\text{Diag}(\sigma) V^T \Omega V + V^T \Omega V \text{Diag}(\sigma) = V^T Z V \tag{E.2}$$

[1] This approach for solving the Sylvester equation is implemented in the matlab code for the EDMC problem by Mishra's https://bamdevmishra.in/codes/edmcompletion/

Define:

$$\Omega_v = V^T \Omega V$$

and

$$Z_v = V^T Z V$$

and Equation (E.1) becomes the simpler equation:

$$\text{Diag}(\sigma)\Omega_v + \Omega_v \text{Diag}(\sigma) = Z_v \tag{E.3}$$

Equation (E.3) can be expressed using element-wise matrix multiplication, $\circ$, as:

$$\Omega_v \circ (\sigma \mathbf{1}_3^T) + \Omega_v \circ (\mathbf{1}_3 \sigma^T) = Z_v \ . \tag{E.4}$$

Factor out $\Omega_v$:

$$\Omega_v \circ (\sigma \mathbf{1}_3^T + \mathbf{1}_3 \sigma^T) = Z_v \ . \tag{E.5}$$

Let $\oslash$ denote element-wise division, then:

$$\Omega_v = Z_v \oslash (\sigma \mathbf{1}_3^T + \mathbf{1}_3 \sigma^T) \ . \tag{E.6}$$

Since $\Omega_v = V^T \Omega V$, $\Omega$ is recovered by the matrix multiplication

$$\Omega = V \Omega_v V^T \ .$$

The Python code below implements this approach using scipy. Only the top triangle of the $\Omega_v$ matrix is calculated, then the lower triangle of $\Omega_v$ is determined using:

$$\Omega_v = \Omega_v - \Omega_v^T \ .$$

```
# Solve Sylvester equation
def sylvester(Sym_mat, Skew_mat):
    Sym = sp.matrix(Sym_mat)
    Skew = sp.matrix(Skew_mat)

    row, col = Sym.shape
    r = col

    V, sig, Vh = la.svd(Sym)
    V= sp.matrix(V)
```

```
O = sp.zeros((r,r))
O2 = V.T*Skew
O2 = O2*V
# only the upper triangular portion of O is calculated
for i in range(r-1):
    for j in range(i+1, r):
        A = sig[i] + sig[j]
        if not A == 0:
            O[i,j]=O2[i,j]/float(A)
        else:
            O[i,j] = 0
O = O - O.T
O = sp.matrix(O)
omega = V*O*V.T
return omega
```

## E.2 The Procrustes Problem

Suppose $A, B \in \mathbb{R}^{n\times m}$, then the Procrustes problem is to find an orthogonal $m \times m$ matrix $Q \in \mathbf{O}_m$ to rotate $B$ to be as close to $A$ as possible.

For example, suppose $A = P_1$ is an $n\times 3$ matrix with rows representing the 3D Cartesian coordinates of $n$ points. Suppose $B = P_2$ is another $n \times 3$ matrix also with the rows representing the 3D Cartesian coordinates of another group of $n$ points. The Procrustes problem is to find an $3 \times 3$ rotation matrix $Q \in \mathbf{O}_3$ to rotate $B = P_2$ such that the two sets of points are as close as possible.

The Procrustes problem can be formulated as the following optimization problem:

$$\begin{aligned} &\min_{Q} \parallel A - BQ \parallel_F \\ &\text{s. t. } Q^TQ = I_m \ . \end{aligned} \tag{E.7}$$

The solution to the optimization problem is:

$$Q = UV^T$$

where $U$ and $V$ come from the singular value decomposition of $B^TA = U\Sigma V^T$. See [44] Section 3.1.1 and [31] Section 12.4.1 for a proof of this result.

# Appendix F

# Quadrance in Other Research Areas

## F.1 The Stress Function and the S-Stress Function

The study of the Hookean potential and the PSD potential is not unique to this thesis. In (Parhizkar 2013 [62]) the author discussed the stress function and the s-stress function.