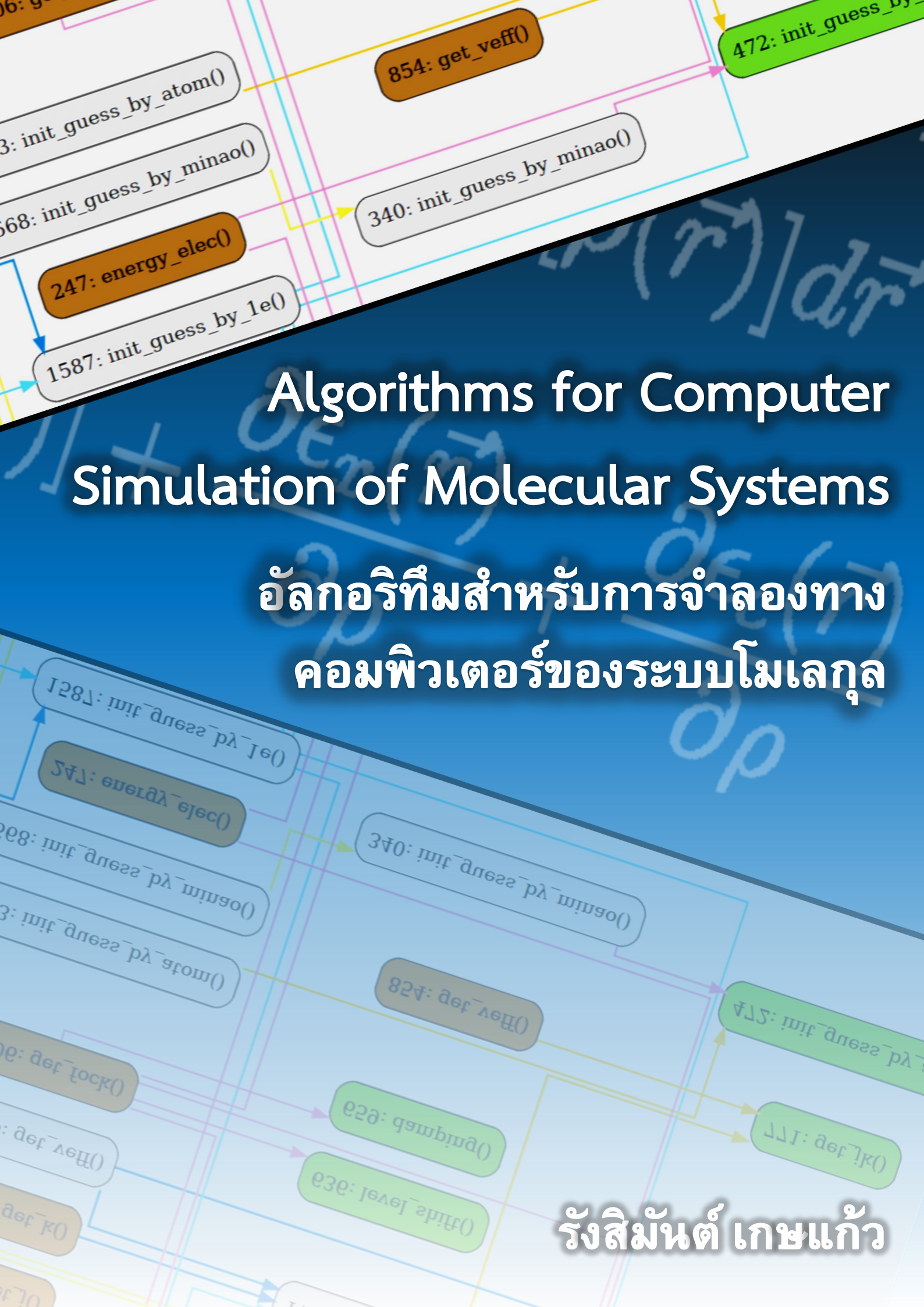



# Algorithms for Computer Simulation of Molecular Systems

# อัลกอริทึมสำหรับการจำลองทางคอมพิวเตอร์ของระบบโมเลกุล

รังสิมันต์ เกษแก้ว

# Algorithms for Computer Simulation of Molecular Systems

# อัลกอริทึมสำหรับการจำลองทางคอมพิวเตอร์ของระบบโมเลกุล

รังสิมันต์ เกษแก้ว

ฉบับพิมพ์ครั้งที่ 1

รังสิมันต์ เกษแก้ว

Algorithms for Computer Simulation of Molecular Systems
อัลกอริทึมสำหรับการจำลองทางคอมพิวเตอร์ของระบบโมเลกุล

ฉบับพิมพ์ครั้งที่ 1 พ.ศ. 2567




ซอร์สโค้ดของหนังสือเล่มนี้ถูกเขียนขึ้นโดยใช้ภาษา LaTeX เผยแพร่ที่ https://github.com/rangsimanketkaew/algo-sim-mol-book และไฟล์ PDF ถูกสร้างขึ้นโดยใช้ XeLaTeX เผยแพร่ที่ https://rangsimanketkaew.github.io/algo-sim-mol-book

หากต้องการติดต่อผู้เขียน กรุณาส่งอีเมลมาที่ rangsiman1993[at]gmail[dot]com

# สารบัญ


**คำนำ** vii

**กิตติกรรมประกาศ** ix

**สิ่งที่ควรทราบเกี่ยวกับหนังสือเล่มนี้** x

**บทที่ 1 กลศาสตร์ควอนตัมเชิงโมเลกุล** 1

1.1 การจำลองเชิงตัวเลขและเทคนิคเชิงคอมพิวเตอร์ . . . . . 1

1.2 พื้นฐานคณิตศาสตร์ที่ต้องรู้ . . . . . 3

1.3 หน่วยอะตอม . . . . . 14

1.4 สมการชโรดิงเงอร์ . . . . . 15

1.5 แฮมิลโทเนียนเชิงโมเลกุล . . . . . 17

1.6 คุณสมบัติพื้นฐานของฟังก์ชันคลื่น . . . . . 18

1.7 การประมาณของบอร์น-ออพเพนไฮเมอร์ . . . . . 21

1.8 ออร์บิทัลเชิงอะตอม . . . . . 22

1.9 ออร์บิทัลเชิงโมเลกุล . . . . . 26

1.10 หลักการผันแปร . . . . . 36

1.11 การประมาณของฮาร์ทรี-ฟ็อค . . . . . 37

1.12 เบซิสเซท . . . . . 43

1.13 พลังงานสหสัมพันธ์ของอิเล็กตรอน . . . . . 49

1.14 ทฤษฎีฟังก์ชันคลื่น . . . . . 50

1.15 ทฤษฎีฟังก์ชันนอลความหนาแน่น . . . . . 61

1.16 ฟังก์ชันนอล . . . . . 64
1.17 แบบฝึกหัด . . . . . 68

**บทที่ 2 พลวัตเชิงโมเลกุลแบบดั้งเดิม 69**

2.1 การประยุกต์ใช้ Molecular Dynamics . . . . . 69
2.2 ประวัติศาสตร์ของ Molecular Dynamics . . . . . 70
2.3 สนามแรง . . . . . 71
2.4 สนามแรงสำหรับพันธะโควาเลนท์ . . . . . 73
2.5 อันตรกิริยาระหว่างโมเลกุล . . . . . 79
2.6 สมการของการเคลื่อนที่ . . . . . 90
2.7 ข้อจำกัดของ MD . . . . . 90
2.8 ขั้นตอนการจำลอง MD . . . . . 91
2.9 การวิเคราะห์ผลการจำลอง MD . . . . . 94
2.10 ศึกษาเพิ่มเติมเกี่ยวกับงานวิจัยทางด้าน MD . . . . . 94
2.11 แบบฝึกหัด . . . . . 95

**บทที่ 3 พลวัตเชิงโมเลกุลแบบแอบ อินิชิโอ 96**

3.1 ทำไมต้อง *Ab Initio* Molecular Dynamics . . . . . 96
3.2 จาก MD สู่ *Ab Initio* MD . . . . . 98
3.3 มาเจาะลึก Classical Molecular Dynamics . . . . . 100
3.4 พลวัตเชิงโมเลกุลแบบเออเรนเฟสต์ . . . . . 103
3.5 พลวัตเชิงโมเลกุลแบบบอร์น-ออปเพนไฮเมอร์ . . . . . 104
3.6 พลวัตเชิงโมเลกุลแบบคาร์-พาร์ริเนลโล . . . . . 107
3.7 แล้ว Hellmann-Feynman Force ล่ะ? . . . . . 108
3.8 การสุ่มตัวอย่างแบบมีประสิทธิภาพ . . . . . 111
3.9 แบบฝึกหัด . . . . . 114

**บทที่ 4 การพัฒนาซอฟต์แวร์สำหรับเคมีเชิงคำนวณ 115**

4.1 การเขียนโปรแกรมทางเคมีเชิงคำนวณ . . . . . 115
4.2 การวางโครงสร้างโปรแกรม . . . . . 117
4.3 ทักษะและเครื่องมือสำหรับการเขียนโปรแกรมคำนวณทางวิทยาศาสตร์ . . . . . 119
4.4 เขียนโปรแกรมวิเคราะห์ Molecular Geometry (ภาษา C++) . . . . . 122
4.5 เขียนโปรแกรม Hartree-Fock SCF (ภาษา Python) . . . . . 137
4.6 เขียนโปรแกรม Direct Inversion of the Iterative Subspace . . . . . 154
4.7 เขียนโปรแกรม Møller-Plesset Perturbation (ภาษา Python) . . . . . 168
4.8 เขียนโปรแกรม Coupled Cluster (ภาษา Python) . . . . . 171
4.9 เขียนโปรแกรม Kohn-Sham DFT (ภาษา Python) . . . . . 183
4.10 เขียนโปรแกรมคำนวณ Molecular Quantum Integrals . . . . . 191
4.11 การเพิ่มประสิทธิภาพการแปลง Two-Electron Integral . . . . . 210
4.12 คำแนะนำสำหรับการคำนวณเคมีควอนตัม . . . . . 214
4.13 เรื่องที่หลายคนเข้าใจผิดเกี่ยวกับโปรแกรมเคมีควอนตัม . . . . . 217
4.14 แบบฝึกหัด . . . . . 218

**บทที่ 5 การคำนวณวิทยาศาสตร์สมรรถนะสูง 219**

5.1 ทำไมต้อง High-Performance Computing ? . . . . . 219
5.2 ทักษะและเครื่องมือสำหรับการเขียนโปรแกรมสำหรับ HPC . . . . . 221
5.3 การทำให้เกิดเมทริกซ์รูปทแยง (Matrix Diagonalization) . . . . . 222
5.4 การวัดประสิทธิภาพไลบรารี่สำหรับ Matrix Diagonalization . . . . . 224
5.5 การประยุกต์ใช้ Matrix Diagonalization . . . . . 226
5.6 การวัดประสิทธิภาพโปรแกรม *Ab Initio* Molecular Dynamics . . . . . 227
5.7 เทคนิคการเขียนโปรแกรม *Ab Initio* Molecular Dynamics . . . . . 231
5.8 การเขียนโปรแกรมเคมีควอนตัมบน GPU . . . . . 232
5.9 การวัดประสิทธิภาพ(ซุปเปอร์)คอมพิวเตอร์ . . . . . 237
5.10 ศึกษาเพิ่มเติมเกี่ยวกับการคำนวณแบบขนาน . . . . . 239

5.11 แบบฝึกหัด . . . . . 240

**ภาคผนวก** **241**

**บทที่ A เทคนิคทางโครงสร้างเชิงอิเล็กทรอนิกส์** **242**

1 Static Correlation กับ Dynamic Correlation . . . . . 242

2 Density Matrix Renormalization Group . . . . . 244

3 Density Matrix Functional Theory . . . . . 245

**บทที่ B ทฤษฎีและโครงสร้างของโปรแกรม CP2K** **249**

1 Hamiltonian และ Energy Functional . . . . . 249

2 ทำความเข้าใจและวิเคราะห์โปรแกรม CP2K . . . . . 250

3 การพัฒนาโปรแกรม CP2K . . . . . 250

**บทที่ C Equation of State สำหรับของเหลว** **251**

**บรรณานุกรม** **256**

**ดรรชนีภาษาไทย** **260**

**ดรรชนีภาษาอังกฤษ** **262**

# คำนำ

การจำลองและคำนวณคุณสมบัติเชิงโครงสร้างและอิเล็กทรอนิกส์ของระบบโมเลกุลด้วยวิธีทางคอมพิวเตอร์นั้นช่วยให้นักเคมีเข้าใจและอธิบายผลการทดลองได้ วิธีคำนวณทางเคมีในระดับอะตอมนั้นสามารถแบ่งออกเป็นสองวิธีคือวิธีเคมีควอนตัม (Quantum Chemistry) และวิธีพลวัตเชิงโมเลกุล (Molecular Dynamics) ซึ่งทั้งสองวิธีนี้มีความแตกต่างกันทั้งในแง่ทฤษฎี การนำไปใช้งานและความถูกต้องของการคำนวณ โดยวิธี Quantum Chemistry นั้นถูกนำมาใช้ในการศึกษาคุณสมบัติของโมเลกุลที่อธิบายด้วยโครงสร้างเชิงอิเล็กทรอนิกส์ซึ่งจะเกี่ยวข้องกับการคำนวณอันตรกิริยาระหว่างอิเล็กตรอนโดยใช้วิธีการประมาณ วิธีการนี้จะให้ผลการคำนวณที่ถูกต้องและใกล้เคียงกับผลการทดลองมาก สำหรับวิธี Molecular Dynamics นั้นถูกนำมาใช้ศึกษาระบบโมเลกุลที่มีขนาดใหญ่และคุณสมบัติในระดับ Microscopis ของระบบด้วยการจำลองการเคลื่อนที่ของอะตอมโดยอ้างอิงหลักกลศาสตร์แบบดั้งเดิม วิธีการนี้ช่วยให้เราศึกษาคุณสมบัติของโมเลกุลได้ในระดับของขนาดที่ใหญ่ขึ้นและระดับของระยะเวลาที่กว้างขึ้นด้วย

วิธี Quantum Chemistry นั้นจะเริ่มต้นจากหลักการพื้นฐานของฟังก์ชันคลื่น (Wavefunction) ที่ใช้สมการชโรดิงเงอร์ในการอธิบายซึ่งในปัจจุบันนั้นได้พัฒนามาเป็นวิชาโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) การแก้สมการชโรดิงเงอร์สำหรับระบบที่มีอิเล็กตรอนมากกว่าหนึ่งตัวนั้นสามารถทำได้เพียงแค่การประมาณค่าโดยที่ยังไม่สามารถหาผลเฉลยแบบแม่นตรงได้นั่นก็เพราะว่าเราไม่สามารถหาผลเฉลยของเทอมที่เป็นอันตรกิริยาระหว่างอิเล็กตรอนได้ ดังนั้นนักเคมีทฤษฎีจึงได้พัฒนาวิธีหรือเทคนิคต่าง ๆ เพื่อนำมาใช้ในการประมาณค่าเทอมดังกล่าวนี้ให้ได้คำตอบที่ถูกต้องและแม่นยำมากที่สุดเท่าที่จะทำได้ซึ่งก็เป็นหนึ่งในหัวข้องานวิจัยของเคมีเชิงคำนวณ

การจำลองทางคอมพิวเตอร์เพื่อศึกษาระบบโมเลกุลด้วยวิธี Molecular Dynamics แบบดั้งเดิมนั้นจะใช้แนวคิดของสนามแรง (Force Field) ซึ่งเป็นรูปแบบของฟังก์ชันทางคณิตศาสตร์ที่มีพารามิเตอร์ที่สามารถอธิบายพลังงานศักย์ของโมเลกุล (กลุ่มอะตอม) ซึ่งพลังงานศักย์ที่ได้มาจากการคำนวณ Force Field นี้คือสิ่งสำคัญที่เรามานำใช้ในการคำนวณแรงของอะตอมแต่ละอะตอมเพื่อใช้ในการหาโครงสร้างของโมเลกุล ณ จุดเวลาต่อ ๆ ไปได้ ซึ่งวิธี Molecular Dynamics แบบที่อ้างอิงกับ Force Field นั้นถูกนำมาใช้อย่างแพร่หลายในการศึกษาระบบต่าง ๆ เช่น การเปลี่ยนสถานะของสสาร ปฏิกิริยาเคมีหรือการเปลี่ยนแปลงเชิงโครงสร้างของชีวโมเลกุล นอกจากนี้การพัฒนา Force Field ก็เป็นหนึ่งในหัวข้องานวิจัยที่ได้รับความสนใจมาจนถึงปัจจุบัน

แนวคิดของการรวมวิธีการคำนวณ Quantum Chemistry เข้ากับ Molecular Dynamics เพื่อเพิ่ม

ความถูกต้องให้กับการคำนวณโดยการรวมนำข้อดีของทั้งสองวิธีมารวมกันนั้นมีมานานมากกว่า 40 ปีแล้ว โดยวิธีการคำนวณที่ถูกพัฒนาขึ้นมาใหม่นี้มีชื่อเรียกว่า *ab initio* Molecular Dynamics (AIMD) หรือพลวัตเชิงโมเลกุลแบบแอบ อินิชิโอ โดยในปัจจุบันวิธี AIMD นั้นมีความสำคัญและได้ถูกนำมาใช้อย่างแพร่หลายอย่างมากในการจำลองระบบโมเลกุล โดยเฉพาะทางด้านวัสดุศาสตร์และฟิสิกส์ เช่น โครงสร้างผลึกและสภาวะของแข็งของโมเลกุล เพื่อศึกษาคุณสมบัติต่าง ๆ ของระบบเหล่านั้นก่อนที่จะมีการนำไปศึกษาจริงในห้องปฏิบัติการ ช่วยให้ประหยัดงบประมาณในการทำงานวิจัยได้อย่างมหาศาลและช่วยให้นักวิทยาศาสตร์เข้าใจถึงพฤติกรรมและอธิบายถึงปัจจัยที่ส่งผลต่อคุณสมบัติของโมเลกุลได้ด้วย

เนื่องจากว่าในปัจจุบันนั้นแหล่งความรู้สำหรับการศึกษาอัลกอริทึมการจำลองทางคอมพิวเตอร์ของระบบโมเลกุลนั้นมักจะอยู่ในรูปของบทความวิชาการเป็นส่วนใหญ่ ซึ่งบทความวิชาการเหล่านั้นส่วนใหญ่แล้วจะมีเนื้อหาที่ซับซ้อนและอาจจะทำให้ยากต่อการทำความเข้าใจของผู้ที่เพิ่มเริ่มศึกษา ยิ่งไปกว่านั้นผู้เขียนพบว่า (เท่าที่ผู้เขียนทราบ) ยังไม่มีหนังสือหรือตำราทางวิชาการภาษาไทยที่อธิบายการจำลองทางคอมพิวเตอร์ของระบบโมเลกุลและการเขียนโปรแกรมด้าน Electronic Structure ได้อย่างครอบคลุม ดังนั้นหนังสือเล่มนี้จึงเป็นอีกช่องทางหนึ่งสำหรับผู้ที่สนใจและต้องการศึกษาอัลกอริทึมของการคำนวณ Electronic Structure และ Molecular Dynamics เพื่อนำไปใช้ต่อยอดในงานวิจัยทางด้านเคมีทฤษฎีและเคมีเชิงคำนวณ รวมไปถึงสาขาที่เกี่ยวข้องด้วย

สุดท้ายนี้ผู้เขียนหวังเป็นอย่างยิ่งว่าหนังสือเล่มนี้จะช่วยให้ผู้อ่านได้รับความรู้ที่ถูกต้องและครบถ้วนเกี่ยวกับวิธีทางเคมีเชิงคำนวณ ถ้าหากผู้อ่านมีข้อเสนอแนะ ข้อสงสัยหรือพบข้อผิดพลาดเกี่ยวกับเนื้อหาในหนังสือเล่มนี้สามารถติดต่อผู้เขียนได้โดยตรงผ่านทางอีเมล rangsiman1993[at]gmail[dot]com

รังสิมันต์ เกษแก้ว
6 กุมภาพันธ์ พ.ศ. 2567

# กิตติกรรมประกาศ

ความรู้และแรงบันดาลใจในการเขียนหนังสือเล่มนี้ของผู้เขียนมาจากแรงผลักดันและการสนับสนุนของบุคคลหลายท่าน การเขียนหนังสือเล่มนี้จะไม่เกิดขึ้นหรือสำเร็จไม่ได้ถ้าหากขาดบุคคลดังต่อไปนี้

ครอบครัวของผู้เขียนที่สนับสนุนให้ผู้เขียนได้ทำตามความฝันในการเรียนต่อระดับอุดมศึกษา ทั้งในระดับปริญญาโทและปริญญาเอก โดยเฉพาะการเห็นคุณค่าของการเรียนและการทำงานวิจัยทางด้านวิทยาศาสตร์พื้นฐาน

รศ.ดร. ยุทธนา ตันติรุ่งโรจน์ชัย บุคคลผู้เป็นต้นแบบด้านการเรียนและเป็นผู้สร้างแรงบันดาลใจให้ผู้เขียนเรียนต่อต่างประเทศและทำงานวิจัยทางด้านเคมีทฤษฎีและเคมีคอมพิวเตอร์

อาจารย์และเพื่อน ๆ ในช่วงมัธยมศึกษา (ตอนต้น-ปลาย) ที่โรงเรียนพนัสพิทยาคาร และช่วงปริญญาตรี-โท ที่มหาวิทยาลัยธรรมศาสตร์ สำหรับความทรงจำอันดีงามและความเป็นกัลยาณมิตรที่ดีเสมอมา

เพื่อน ๆ ที่เมืองซูริค ประเทศสวิตเซอร์แลนด์ สำหรับมิตรภาพอันดีงาม รอยยิ้มและเสียงหัวเราะที่เกิดขึ้น รวมไปถึงกิจกรรมที่ได้ทำร่วมกันในระหว่างที่ผู้เขียนกำลังศึกษาปริญญาเอกซึ่งเป็นช่วงเวลาเดียวกันกับที่ผู้เขียนกำลังเขียนหนังสือเล่มนี้

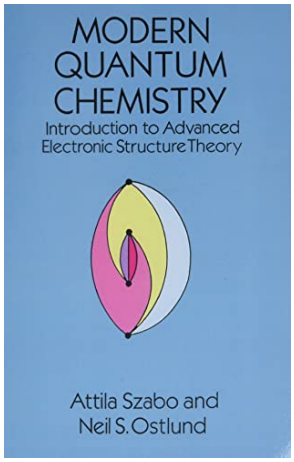
เพื่อนร่วมงานทั้งนักศึกษาปริญญาโท ปริญญาเอก และนักวิจัยหลังปริญญาเอกของกลุ่มวิจัยของผู้เขียนที่ภาควิชาเคมี มหาวิทยาลัยแห่งซูริค สำหรับการแลกเปลี่ยนความรู้ ไอเดียใหม่ ๆ และการช่วยเหลือเกี่ยวกับงานวิจัยทางด้านเคมีทฤษฎี

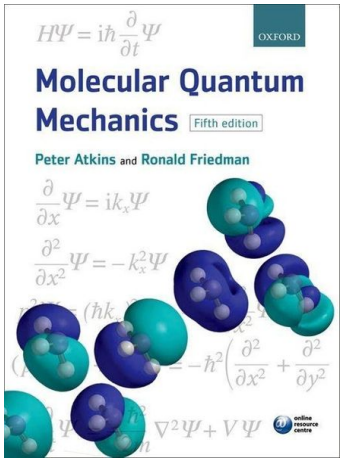
รังสิมันต์ เกษแก้ว

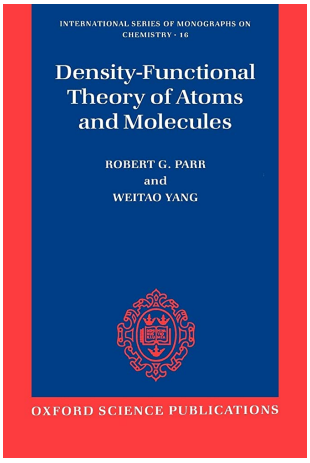
# สิ่งที่ควรทราบเกี่ยวกับหนังสือเล่มนี้

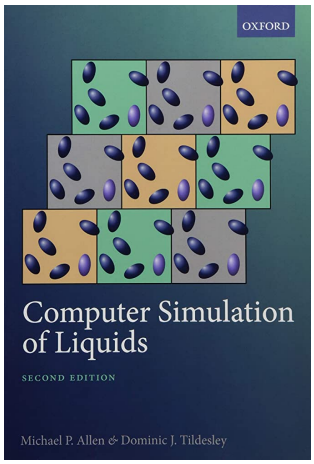
หนังสือเล่มนี้มีหนังสือต่างประเทศหลายเล่มเป็นแหล่งข้อมูลอ้างอิงทางวิชาการเพื่อให้มั่นใจได้ว่าความรู้ที่ผู้อ่านจะได้ศึกษาจากหนังสือเล่มนี้นั้นถูกต้อง อย่างไรก็ตามผมไม่สามารถการันตีได้ 100 เปอร์เซ็นต์ว่าหนังสือเล่มนี้ไม่มีข้อผิดพลาดเลยแม้แต่จุดเดียว ข้อผิดพลาดเล็กน้อยอาจจะเกิดจากการพิมพ์ผิด เป็นต้น ส่วนข้อผิดพลาดทางเนื้อหานั้นผมค่อนข้างมั่นใจว่ามีน้อยมากเพราะว่าผมได้ใช้หนังสือได้ที่รับการยอมรับและความนิยมในกลุ่มนักวิจัยทางด้าน Quantum Chemistry, Electronic Structure และ *Ab Initio* Molecular Dynamics หลาย ๆ เล่มมาเป็นหนังสืออ้างอิง รวมถึงได้พูดคุยและสอบถามกับนักวิจัยด้านเคมีควอนตัมคนอื่น ๆ ทั้งที่เป็นเพื่อนร่วมงานและคนที่ผมได้ไปพบเจอตามงานประชุมวิชาการ จึงทำให้มั่นใจได้ว่าความรู้ที่ถ่ายทอดผ่านหนังสือเล่มนี้นั้นถูกต้อง สำหรับสไตล์การใช้ภาษาในการเขียนหนังสือเล่มนี้ไม่วิชาการมากเกินไป ซึ่งผมพยายามเลือกใช้คำง่าย ๆ ให้มากที่สุด แต่ถึงอย่างนั้นก็ตามผมก็ไม่อาจที่จะหลีกเลี่ยงการใช้คำศัพท์เชิงเทคนิคได้

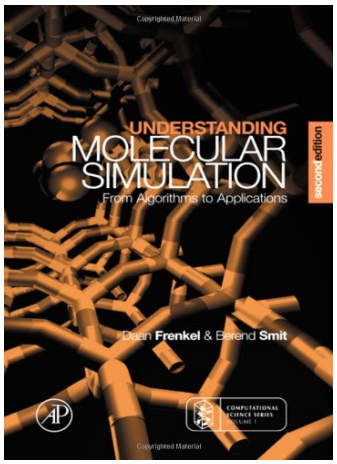
หนังสือ 6 เล่มที่ผมคิดว่าเขียนได้ดีมาก ๆ และผมใช้เป็นหนังสืออ้างอิงไม่เพียงแค่สำหรับเขียนหนังสือเล่มนี้แต่ยังใช้ในการทำงานวิจัยของผมเองด้วยมีดังนี้

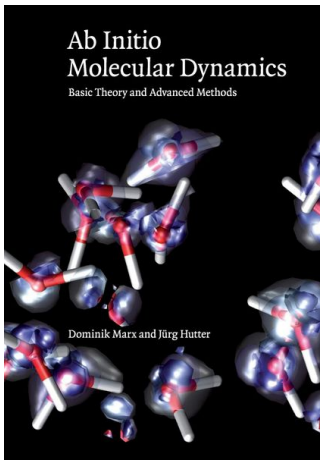



1. Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory แต่งโดย Attila Szabo และ Neil S. Ostlund[1]

2. Molecular Quantum Mechanics (5th Edition) แต่งโดย Peter W. Atkins และ Ronald S. Friedman[2]

3. Density-Functional Theory of Atoms and Molecules แต่งโดย Robert G. Parr และ Yang Weitao[3]

4. Computer Simulation of Liquids (2nd Edition) แต่งโดย Michael P. Allen และ Dominic J. Tildesley[4]

5. Understanding Molecular Simulation: From Algorithms to Applications (2nd Edition) แต่งโดย Daan Frenkel และ Berend Smit[5]

6. *Ab Initio* Molecular Dynamics: Basic Theory and Advanced Methods แต่งโดย Dominik Marx และ Juerg Hutter[6]

หนังสือเล่มนี้เหมาะสำหรับนักศึกษา นักวิจัย ผู้ที่สนใจในสาขาเคมีทฤษฎีและเคมีคำนวณและผ่านการเรียนวิชาเคมีเชิงฟิสิกส์มาแล้ว โดยเนื้อหาในหนังสือเล่มนี้เน้นไปที่การอธิบายทฤษฎีและพิสูจน์ที่มาของสมการทางกลศาสตร์ควอนตัมและโครงสร้างเชิงอิเล็กทรอนิกส์ของอะตอมและโมเลกุลรวมถึงแบบจำลองของระบบโมเลกุล นอกจากนี้ยังอธิบายการประยุกต์ใช้อัลกอริทึมทางคอมพิวเตอร์สำหรับการเขียนโปรแกรมคำนวณและการประมวลผลบนคอมพิวเตอร์สมรรถนะสูงอีกด้วย

หนังสือเล่มนี้ไม่เหมาะสำหรับผู้ที่ยังไม่เคยเรียนวิชาเคมีเชิงฟิสิกส์หรือเคมีควอนตัมมาก่อนเพราะว่าอาจจะตกใจกับคำศัพท์ทางเทคนิคและสมการต่าง ๆ ได้ อย่างไรก็ตามถ้าหากผู้อ่านสามารถทำความเข้าใจเนื้อหาของหนังสือเล่มนี้ได้โดยที่ยังไม่เคยศึกษาเคมีควอนตัมมาก่อนเลยก็ถือว่าเยี่ยมมาก ๆ นอกจากนี้แล้วในการทำความเข้าใจเนื้อหาในหนังสือเล่มนี้นั้นจะต้องอาศัยความรู้ทางคณิตศาสตร์พีชคณิตเยอะมาก ๆ โดยเฉพาะความรู้เกี่ยวกับเวกเตอร์ เมทริกซ์ แคลคูลัส รวมไปถึงเทคนิคต่าง ๆ ทางคณิตศาสตร์ เช่น Optimization, Decomposition, และ Diagonalization ดังนั้นผู้อ่านควรจะต้องมีความรู้คณิศาสตร์เหล่านี้มาบ้างแล้ว

คำแนะนำในการศึกษาทฤษฎีทางเคมีควอนตัมจากประสบการณ์ส่วนตัวของผมที่อยากจะแบ่งปันมี 3 ข้อคือ

1. อ่านตำรา (Textbook) ทีละเล่ม : สำหรับผู้ที่เริ่มต้นนั้นควรเลือกหนังสือเพียงเล่มเดียว ถ้าหากว่าอ่านหลายเล่มพร้อม ๆ กันแล้วสลับอ่านไปมาก็อาจจะทำให้เกิดความสับสนได้เพราะว่าหนังสือแต่ละเล่มนั้นจะมีการใช้สัญลักษณ์หรือคำศัพท์เฉพาะทางที่ต่างกันได้

2. อ่านบทความวิชาการเพื่ออัพเดทความรู้ : ปฏิเสธไม่ได้เลยว่าการทำวิจัยในปัจจุบันนั้นดำเนินไปอย่างรวดเร็วมาก ๆ เนื่องจากว่าเรามีคอมพิวเตอร์ที่มีประสิทธิภาพมากขึ้น รวมถึงกระบวนการตีพิมพ์งานวิชาการนั้นก็ไม่ได้ช้าเหมือนในอดีต ดังนั้นการอ่านบทความวิชาการ เช่น บทความวิจัย, บทความรีวิว รวมถึงสรุปข่าวต่าง ๆ จึงมีความสำคัญอย่างมากเพราะว่าเราจะได้อัพเดทความรู้ของเราตลอดเวลา

3. การเขียนโปรแกรม : การเขียนโปรแกรมนั้นจะมีลำดับขั้นตอนที่ชัดเจน ตรงไปตรงมา และไม่สามารถข้ามขั้นตอนได้ ถ้าหากว่าเราศึกษาการเขียนโปรแกรมควบคู่ไปกับการทฤษฎีที่สนใจก็จะทำให้เราเข้าใจที่มาที่ไปได้ง่ายขึ้น

*Erwin with his $\psi$ can do*
*Calculations quite a few.*
*But one thing has not been seen,*
*Just what does $\psi$ really mean.*

*- Walter Hückel (1895 - 1973)*

# บทที่ 1

# กลศาสตร์ควอนตัมเชิงโมเลกุล

## 1.1 การจำลองเชิงตัวเลขและเทคนิคเชิงคอมพิวเตอร์

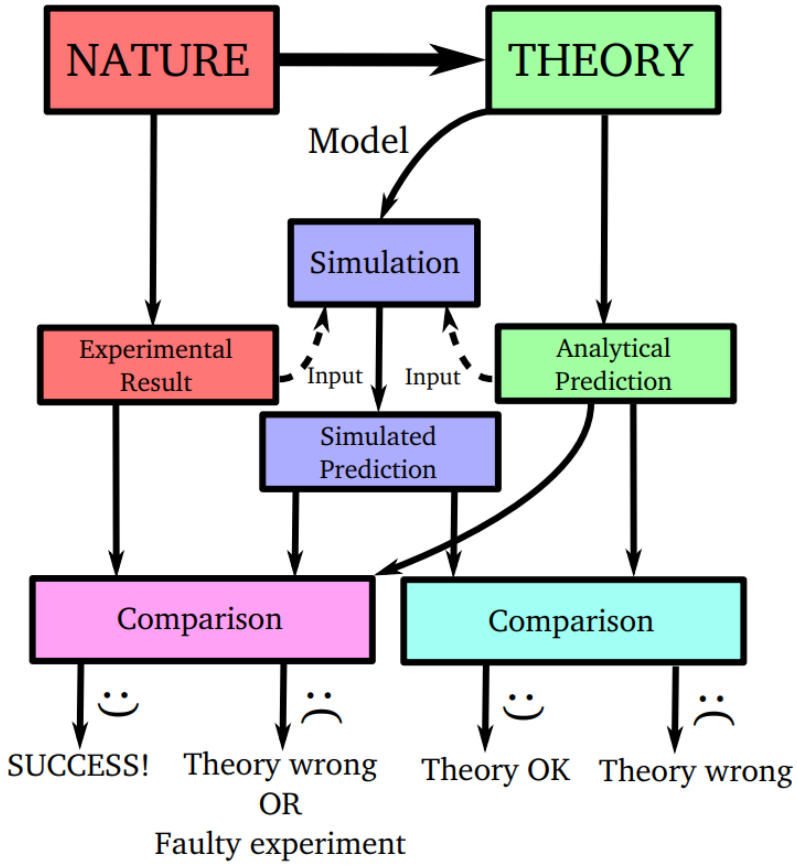
การจำลองเชิงตัวเลข (Numerical Modeling) คือวิธีการที่เราใช้แก้ปัญหาทางคณิตศาสตร์และฟิสิกส์แบบต่าง ๆ เช่น นำมาใช้แก้สมการเชิงอนุพันธ์ (Differential Equations) ที่ใช้ในการอธิบายปรากฏการณ์ต่าง ๆ ทางธรรมชาติซึ่งอาจจะมีความยากหรืออาจจะไม่มีทางแก้ได้ด้วยวิธีเชิงวิเคราะห์ (Analytical Method)

สำหรับการจำลองด้วยเทคนิคเชิงคอมพิวเตอร์ (Computer Simulation) นั้นเป็นการศึกษาการตอบสนองเชิงพลวัต (Dynamic Response) ของระบบที่เราศึกษาและจำลองภายใต้เงื่อนไขเริ่มต้น (Initial Conditions) ที่เราได้กำหนดไว้ โดยเงื่อนไขเริ่มต้นนี้สอดคล้องกับสภาวะจริงของระบบนั้น

ภาพที่ 1.1 แสดงแผนผังเชื่อมโยงความแตกต่างระหว่าง Numerical Modeling กับ Computer Simulation นั้นก็คือใน Simulation นั้นระบบจำลองของเราจะถูกสร้างขึ้นมา เช่น เราสร้างระบบที่เป็นโมเลกุลน้ำหลาย ๆ โมเลกุลเกาะกลุ่มรวมกัน (Water Cluster) โดยเราหวังว่า Water Cluster ที่เราสร้างขึ้นมานี้จะสามารถเป็นตัวแทนของระบบของโมเลกุลน้ำจริง ๆ ได้ ซึ่งก็จะทำให้เราสามารถศึกษาคุณสมบัติต่าง ๆ ของโมเลกุลน้ำได้ตามต้องการ ส่วนการจำลองเชิงตัวหรือ Numerical Simulation นั้นจะเป็นการสร้างการทดลองเสมือนจริง (Virtual Experiments) ของระบบจำลองขึ้นมา อย่างไรก็ตามในบทความวิชาการทางด้านเคมีเชิงคำนวณหรือชีวเชิงคำนวณนั้นเรามักจะพบว่าคำว่า Modeling นั้นสามารถถูกแทนด้วยคำว่า Simulation ได้เช่นกัน

คำถามสำคัญที่หลายคนมักจะถาม โดยเฉพาะนักเคมีที่ทำงานวิจัยโดยการทดลองในห้องปฏิบัติการ (Experimentalists) ก็คือ “ทำไมการจำลองทางคอมพิวเตอร์ถึงมีความสำคัญ?” ซึ่งคำตอบนั้นมีด้วยกันหลายข้อ โดยผมขอสรุปเป็นประเด็นตามนี้

1. การจำลองทางคอมพิวเตอร์นั้นเปรียบเสมือนเป็นสะพานเชื่อมโยงระหว่างทฤษฎีกับการทดลอง




ภาพ 1.1 แผนผังความเชื่อมโยงของระบบที่เราต้องศึกษา (Nature), ทฤษฎีหรือวิธีที่ใช้ในการศึกษา (Theory), แบบจำลองหรือโมเดล (Model), ผลการทดลอง (Experimental Result), และผลการคำนวณหรือผลการทำนาย (Computational Results หรือ Prediction)

2. การทำการทดลองบางอย่างนั้นมีค่าใช้จ่ายที่สูงมากและมีความยากเพราะว่าตัวทฤษฎีนั้นซับซ้อนเกินไป ดังนั้นการจำลองทางคอมพิวเตอร์นั้นจะเข้ามาช่วยในการจำลองการทดลองและทดสอบสมมติฐานเพื่อยืนยันทฤษฎีด้วย

3. การจำลองทางคอมพิวเตอร์นั้นช่วยหาปัจจัยและเงื่อนไขที่เหมาะสมสำหรับการทดลองได้

4. การจำลองทางคอมพิวเตอร์สามารถแสดงกระบวนการของระบบที่เราสนใจได้ ซึ่งอาจจะทำได้ยากในเชิงการทดลอง

5. การจำลองทางคอมพิวเตอร์สามารถนำมาใช้ในการศึกษาปรากฏการณ์ที่การทดลองนั้นอาจจะให้ผลการทดลองที่ไม่ละเอียดพอ

อย่างไรก็ตาม ผมอยากจะสรุปเพิ่มเติมด้วยว่าการใช้แบบจำลองทางคอมพิวเตอร์เพียงอย่างเดียวนั้นจะเปล่าประโยชน์ ถ้าหากว่าไม่มีผลการทดลองที่น่าเชื่อมายืนยันความถูกต้องของผลการคำนวณ ดังนั้นเคมีเชิงการทดลองกับเคมีเชิงคำนวณนั้นจึงเป็นศาสตร์ที่ต้องพึ่งพาอาศัยกัน

## 1.2 พื้นฐานคณิตศาสตร์ที่ต้องรู้

### 1.2.1 ทำไมคณิตศาสตร์จึงสำคัญต่อการเรียนเคมีควอนตัม

“พื้นฐานอะไรที่สำคัญที่สุดในการเริ่มศึกษาและทำวิจัยด้านกลศาสตร์เคมีควอนตัม?” คำถามนี้เป็นคำถามง่าย ๆ แต่ว่าหลายคนนั้นมองข้ามไป ผมเริ่มต้นด้วยคำถามนี้ก็เพราะว่าอยากให้ผู้อ่านนั้นตระหนักก่อนว่าเนื้อหาของวิชากลศาสตร์ควอนตัมโดยเฉพาะเคมีควอนตัมนั้นมีความยากและซับซ้อนมาก ๆ ซึ่งในการเริ่มศึกษานั้น นอกจากความรู้เคมีที่เราจะต้องมีแล้ว เราจะต้องมีความรู้ทางคณิตศาสตร์ด้วย เพราะว่ากลศาสตร์ควอนตัมเกี่ยวข้องกับสมการต่าง ๆ มากมาย มีทั้งการพิสูจน์ การดำเนินการต่าง ๆ รวมถึงการคำนวณที่ซับซ้อน

คำถามถัดมาคือ “แล้วหัวข้อไหนในคณิตศาสตร์ที่สำคัญและจำเป็นที่สุดสำหรับการศึกษากลศาสตร์ควอนตัมล่ะ?” ในการตอบคำถามนี้ ผู้อ่านต้องเข้าใจก่อนว่าในเคมีควอนตัมนั้นใช้คณิตศาสตร์หลายหัวข้อมาก ๆ ในการพิสูจน์และแก้สมการต่าง ๆ ซึ่งมีความจำเป็นและจะมองข้ามไปไม่ได้เลย เท่าที่ผมไปอ่านโพสต์ในฟอรั่มของต่างประเทศที่ได้พูดคุยกันเกี่ยวกับสกิลพื้นฐานที่จำเป็นสำหรับการเรียนเคมีควอนตัม ผมจับใจความได้ดังนี้ (เป็นพื้นฐานที่จำเป็นสำหรับการศึกษาควอนตัมทุกแขนง ไม่เพียงแค่เคมีควอนตัมเท่านั้น)

1. Linear Algebra : อย่างน้อย ๆ เลยสิ่งที่ต้องรู้นั่นก็คือพื้นฐานเรื่องเมทริกซ์ (Matrix), ปริภูมิเวกเตอร์ (Vector Spaces), Eigenvalues, Eigenvectors นั่นก็เพราะว่ากลศาสตร์ควอนตัมนั้นเกี่ยวข้องกับเวกเตอร์ ส่วนความรู้ Eigenvalues กับ Eigenvectors นั้นเราจะใช้ในการแก้สมการชโรดิงเงอร์สำหรับ Stationary State ของอะตอมหรือโมเลกุล

2. Calculus: Derivatives, Integrals, Taylor Expansions เราใช้แคลคูลัสเยอะมากในการดำเนินการหรือตรวจสอบคุณสมบัติของฟังก์ชันคลื่น รวมถึงเงื่อนไขขอบเขต (Boundary Condition) ต่าง ๆ

3. Differential Equations และ Partial Differential Equations ใช้ในการแก้สมการอนุพันธ์ทั้งแบบเชิงเส้นและไม่เชิงเส้น

4. ความรู้ด้านพิกัดระบบต่าง ๆ เช่น Cartesian Coordinates, Cylindrical Coordinates และ Spherical Coordinates ซึ่งระบบพิกัดต่าง ๆ นี้ก็จำเป็นสำหรับการศึกษาอนุภาค

5. พื้นฐานเรื่องความน่าจะเป็น (Probability) : ความน่าจะเป็นนั้นใช้เยอะมากในกลศาสตร์ควอนตัมเชิงสถิติ (Statistical Quantum Mechanics) รวมไปถึงการนำกลศาสตร์ไปอธิบายกับทฤษฎีหรือปรากฏการณ์ต่าง ๆ

6. หัวข้ออื่น ๆ : นอกจากนี้ยังมีหัวข้ออื่น ๆ อีกที่จำเป็นต้องรู้สำหรับการศึกษากลศาสตร์ควอนตัมขั้นสูง เช่น Relativistic Quantum Mechanics

   (a) Complex Analysis (โดยเฉพาะ Complex Integration)

   (b) Functional Integration

   (c) Functional Analysis

   (d) Group Theory

   (e) Calculus of Variations หรือ Variational Calculus

   (f) Functional Integration

   (g) Tensor Calculus

### 1.2.2 ทุกอย่างเกี่ยวข้องกับเวกเตอร์และเมทริกซ์

ถ้าต้องให้ผมเลือกหนึ่งหัวข้อที่คิดว่าสำคัญมาก ๆ ทั้งในแง่การนำมาใช้ในการพัฒนาทฤษฎีรวมถึงการนำไปประยุกต์ใช้งานจริง ผมคิดว่าพีชคณิตเชิงเส้น (Linear Algebra) โดยเฉพาะเรื่องเวกเตอร์และเมทริกซ์นั้นน่าจะสำคัญมากที่สุด เหตุผลก็คือผมคิดว่าเมทริกซ์นั้นเป็นพื้นฐานมากที่สุดในกลศาสตร์ควอนตัม เริ่มตั้งแต่การกำหนดหรือนิยามตัวแปรและพารามิเตอร์ต่าง ๆ เลยก็ว่าได้ ซึ่งเราใช้เวกเตอร์หรือเมทริกซ์ทั้งนั้น นอกจากนี้ในกลศาสตร์ควอนตัมนั้นเรามักจะวนเวียนและวุ่นวายอยู่กับฟังก์ชันคลื่น (Wavefunction) เริ่มตั้งแต่การ Represent ฟังก์ชันคลื่นเราก็ใช้เวกเตอร์และเมทริกซ์, การดำเนินการ (Operation) ต่าง ๆ ก็ใช้เอกลักษณ์และคุณสมบัติของเมทริกซ์ ดังนั้นความรู้เกี่ยวกับเมทริกซ์นั้นจึงเป็นสิ่งที่เราจำเป็นต้องใช้และไม่มีใครหนีพ้น ซึ่งนี่ยังไม่รวมถึงการนำไปใช้งานจริงในกลศาสตร์ควอนตัมเชิงการคำนวณ (Computational Quantum Mechanics) ซึ่งเกี่ยวข้องกับการเขียนโปรแกรมที่เราจำเป็นที่จะต้องแก้สมการเชิงเส้นต่าง ๆ เพื่อหาคำตอบออกมา

ตัวอย่างอันหนึ่งที่เห็นภาพได้ชัดก็คือสมการชโรดิงเงอร์ (ผมจะยังไม่ลงรายละเอียด ณ ตอนนี้)

$$\texttt{Hamiltonian} \cdot \texttt{Wavefunction} = \texttt{Energy} \cdot \texttt{Wavefunction} \tag{1.2.1}$$

จะเห็นได้ว่าเรามีทั้ง Hamiltonian, Wavefunction, และ Energy อยู่ในสมการ ซึ่งทั้ง 3 พารามิเตอร์นี้นั้นในสามารถถูกเขียนได้ง่าย ๆ เลยด้วยการใช้เมทริกซ์ซึ่งทำให้ง่ายต่อการตีความ เมื่อตัวแปรทั้ง 3 ตัวเป็นแค่เมทริกซ์ เวลาที่เราจะเขียนฟังก์ชันคลื่นให้อยู่ในรูปการกระจายในฟอร์มต่าง ๆ แบบไหนก็ตามก็ทำได้ง่ายขึ้น นอกจากนี้เรายังสามารถใช้คุณสมบัติของเมทริกซ์เข้ามาช่วยได้อีก เช่น สำหรับวิชาโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) ซึ่งเป็นหัวข้อหนึ่งของเคมีควอนตัมนั้น เราสามารถเขียนฟังก์ชันคลื่นให้อยู่ในรูปของผลรวมเชิงเส้นของผลคูณระหว่าง Basis Function กับสัมประสิทธิ์ออร์บิทัลเชิงโมเลกุล (Molecular Orbitals Coefficient) ได้

นอกจากนี้แล้ว การที่เราเข้าใจเรื่องเมทริกซ์นั้นยังช่วยให้เข้าใจวิธีหรือกระบวนการในการแก้สมการต่าง ๆ ในกลศาสตร์ควอนตัมอย่างเป็นขั้นเป็นตอน โดยเฉพาะตอนที่เราจะต้องเขียนโปรแกรมเพื่อแก้สมการนั้นเราก็ใช้เมทริกซ์เยอะมาก ๆ

### 1.2.3 เบซิส

สิ่งแรกที่ต้องทำความเข้าใจเกี่ยวกับตัวแปรทางเคมีควอนตัมก็คือปริภูมิของเวกเตอร์ (Vector) โดยเราเริ่มต้นด้วยการใช้ Dirac Notation สำหรับ Vector Space $S$ เรากำหนดให้ $\alpha \left|a\right\rangle$ เป็นสมาชิกของ $S$ และเราทำการกำหนดสิ่งที่เรียกว่า **Dual Vector** $\alpha^* \left\langle a\right|$ โดยที่ $\alpha^*$ เป็น Complex Conjugate ของ $\alpha$

สำหรับการดำเนินการทางคณิตศาสตร์ของเวกเตอร์ สิ่งแรกที่เราควรจะต้องรู้ก็คือการคูณเวกเตอร์ 2 อันแบบภายใน **Inner Product** (เช่น $\left|a\right\rangle$ กับ $\left|b\right\rangle$) ซึ่งจะได้ผลลัพธ์เป็นตัวเลขหรือปริมาณสเกลาร์ ดังนี้

$$\left\langle a|b\right\rangle = \alpha \in \mathbb{C} \tag{1.2.2}$$

ซึ่งการคูณแบบนี้เป็นการคูณแบบจุดระหว่างเวกเตอร์นั่นเอง $(\boldsymbol{u} \cdot \boldsymbol{v})$ และการนิยามการคูณแบบนี้จะทำให้การสลับที่ของเวกเตอร์นั้นยังคงมี Complex Conjugate ที่เหมือนกันอยู่

$$\left\langle b|a\right\rangle = \left\langle a|b\right\rangle^* = \alpha^* \tag{1.2.3}$$

โดยเราเรียกเวกเตอร์ $\left|a\right\rangle$ กับ $\left|b\right\rangle$ ที่มีผลลัพธ์เป็น 0 (Zero Inner Product) หรือ $\left\langle a|b\right\rangle = 0$ ว่าเป็นเวกเตอร์แบบตั้งฉาก (**Orthogonal**) ซึ่งมีความอิสระต่อหรือว่าไม่ขึ้นต่อกันนั่นเอง นอกจากนี้ Inner Product ของทั้งสองเวกเตอร์นั้นยังมีความเป็นเส้นตรง (**Linear**) อีกด้วย ดังนี้

$$\langle c|\alpha a + \beta b\rangle = \alpha \langle c|a\rangle + \beta \langle c|b\rangle \tag{1.2.4}$$

และมีความเป็น **Anti-Linear**[1] ดังนี้

$$\langle \alpha a + \beta b|c\rangle = \alpha^* \langle a|c\rangle + \beta^* \langle b|c\rangle \tag{1.2.5}$$

นอกจากนี้ยังเราสามารถนิยาม Norm ของเวกเตอร์ได้ด้วย (โดยที่เรายังใช้ Notation ของ Inner Product อยู่) ดังนี้

$$\| |a\rangle \| = \sqrt{\langle a|a\rangle} \in \mathbb{R} \tag{1.2.6}$$

ซึ่ง Norm ตามที่นิยามนี้มีเงื่อนไขดังต่อไปนี้

$$\begin{aligned} \| |a\rangle \| &> 0 \text{ if } |a\rangle \neq |0\rangle \\ \| |a\rangle \| &= 0 \text{ if } |a\rangle = |0\rangle \end{aligned} \tag{1.2.7}$$

และเราก็เรียกเวกเตอร์ $|a\rangle$ ที่มี Unit Norm ($\langle a|a\rangle = 0$) ว่าเป็น **Normalized Vector** หรือเวกเตอร์ที่ถูกทำให้เป็นปกติ

ลำดับต่อไปที่ผมอยากจะให้ผู้อ่านทำความเข้าใจก็คือการ Represent ปริภูมิเวกเตอร์ในทางกลศาสตร์ควอนตัม ต้องเกริ่นก่อนว่าปริภูมิเวกเตอร์ $S$ นั้นจริง ๆ แล้วก็เป็นแค่เซตของเวกเตอร์หลาย ๆ อันนั่นเอง (เราเลยเรียกว่าปริภูมิ) ซึ่งถ้าหากว่าเซตของเวกเตอร์ของเรานั้นมีหลาย ๆ เวกเตอร์ ก็จะทำให้ปริภูมิเวกเตอร์ของเรานั้นมีจำนวนมิติที่มากตามไปด้วย โดยจำนวนมิติของปริภูมิเวกเตอร์นั้นก็จะเท่ากับจำนวนของเวกเตอร์ $(N)$ ดังนั้นเราจึงเขียน **เบซิส** เป็นภาษาทางคณิตศาสตร์ได้ดังนี้ $\{|\phi_i\rangle, i = 1, \ldots, N\}$ โดยที่เราสามารถเขียนเวกเตอร์ $|a\rangle \in S$ ได้ในรูปของผลรวมเชิงเส้นของสมาชิกแต่ละตัวในเบซิส (ซึ่งสมาชิกแต่ละตัวก็คือเวกเตอร์นั่นเอง)

$$|a\rangle = \sum_{i=1}^{N} a_i |\phi_i\rangle \tag{1.2.8}$$

โดยที่ $a_i$ นั้นคือค่าสัมประสิทธิ์ของสมาชิกของเบซิสแต่ละตัวซึ่งจะมีค่าจำเพาะและไม่เหมือนกัน ผมอยากให้ลองคิดตามง่าย ๆ ว่า *จริง ๆ แล้ว* $|a\rangle$ *นั้นก็เป็นแค่ผลลัพธ์ที่ได้จากการคูณแบบจุด (Dot Product) ของเวก*

[1] ผมไม่รู้ว่าจะแปลคำว่า Anti-Linear ยังไงดี

*เตอร์แบบแถวของฟังก์ชันเบซิส* $|\boldsymbol{\phi}\rangle = (|\phi_1\rangle, |\phi_2\rangle, \ldots, |\phi_N\rangle)$ *กับเวกเตอร์แบบหลักของสัมประสิทธิ์* $\boldsymbol{a} = (a_1, a_2, \ldots, a_N)^T$

$$|a\rangle = (|\phi_1\rangle, |\phi_2\rangle, \ldots, |\phi_N\rangle) \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} = |\boldsymbol{\phi}\rangle \boldsymbol{a} \tag{1.2.9}$$

โดยที่ $\langle\boldsymbol{\phi}|$ คือเวกเตอร์แบบหลักของ Dual Vector ซึ่งมีหน้าตาดังต่อไปนี้

$$\langle\boldsymbol{\phi}| = \begin{pmatrix} \langle\phi_1| \\ \langle\phi_2| \\ \vdots \\ \langle\phi_N| \end{pmatrix} \tag{1.2.10}$$

ปริมาณอีกอันหนึ่งที่สำคัญมาก ๆ ในเคมีควอนตัมเพราะเป็นปริมาณที่สามารถบ่งบอกได้ถึงคุณลักษณะของเบซิสนั่นก็คือเมทริกซ์ซ้อนทับ (Overlap Matrix) ซึ่งมีนิยามดังนี้

$$S_{ij} = \langle\phi_i|\phi_j\rangle \tag{1.2.11}$$

และเมื่อผู้อ่านศึกษาเคมีควอนตัมไปเรื่อย ๆ จะพบว่าบ่อยครั้งที่เราจะต้องเจอกับสิ่งที่เรียกว่า **Orthonormal Bases** ซึ่งก็คือเบซิส พูดง่าย ๆ คือ Orthonormal Basis นั้นก็คือเบซิสที่มีคุณสมบัติ Orthogonal และ Normal ผสมกันอยู่นั่นเอง

$$\begin{aligned} \langle\phi_i|\phi_i\rangle &= 1 \quad \forall i \\ \langle\phi_i|\phi_j\rangle &= 0 \quad \forall ij, i \neq j \end{aligned} \tag{1.2.12}$$

แต่เรามักจะพบเห็นการเขียนเขียนเงื่อนไขของ Orthonormality ตามตำราหรือบทความงานวิจัยที่สั้นและกระชับกว่า ดังนี้

$$\langle\phi_i|\phi_j\rangle = \delta_{ij} \quad \forall ij \tag{1.2.13}$$

โดยที่ $\delta_{ij}$ นั้นมีชื่อเรียกว่า Kronecker Delta Function

เรายังสามารถที่จะเขียน Overlap Matrix ได้โดยการใช้ Expression ดังต่อไปนี้

$$\boldsymbol{S} = \langle \boldsymbol{\phi} | \boldsymbol{\phi} \rangle \tag{1.2.14}$$

$$= \begin{pmatrix} \langle \phi_1 | \\ \langle \phi_2 | \\ \vdots \\ \langle \phi_N | \end{pmatrix} \left( |\phi_1\rangle, |\phi_2\rangle, \ldots, |\phi_N\rangle \right) \tag{1.2.15}$$

$$= \begin{pmatrix} \langle \phi_1 | \phi_1 \rangle & \langle \phi_1 | \phi_2 \rangle & \cdots & \langle \phi_1 | \phi_N \rangle \\ \langle \phi_2 | \phi_1 \rangle & \langle \phi_2 | \phi_2 \rangle & \cdots & \langle \phi_2 | \phi_N \rangle \\ \vdots & & \ddots & \vdots \\ \langle \phi_N | \phi_1 \rangle & \langle \phi_N | \phi_2 \rangle & \cdots & \langle \phi_N | \phi_N \rangle \end{pmatrix} \tag{1.2.16}$$

จากข้อมูลที่เรามีเกี่ยวกับ Overlap Matrix ถ้าหากว่าเรามี **Orthonormal Basis** เราจะสามารถหาค่าของสัมประสิทธิ์ตามสมการ (1.2.8) ได้โดยการใช้ Inner Product ของ $|a\rangle$ กับสมาชิกแต่ละตัวของเบซิส

$$\langle \phi_k | a \rangle = \sum_{i=1}^{N} a_i \langle \phi_k | \phi_i \rangle \tag{1.2.17}$$

$$= \sum_{i=1}^{N} a_i \delta_{ki} \tag{1.2.18}$$

$$= a_1\delta_{k1} + a_2\delta_{k2} + \ldots + a_k \underbrace{\delta_{kk}}_{1} + \ldots \tag{1.2.19}$$

$$= a_k \tag{1.2.20}$$

จะเห็นได้ว่าสมการด้านบนนั้นถูกลดรูปจากอนุกรมเหลือแค่ $a_k$ ซึ่งถ้าสังเกตให้ดีจะพบว่าทริคที่เราใช้ในการ Simplify สมการชุดนี้ก็คือว่าจะมีแค่เทอมที่ Index $i$ มีค่าเท่ากับ $k$ เท่านั้นที่จะมีค่าเท่ากับ 1 (ตามเงื่อนไข Kronecker Delta Function ก่อนหน้านี้) ซึ่งการทำ Simplication แบบนี้เป็นเทคนิคอย่างหนึ่งที่ผู้อ่านควรจะต้องทำความคุ้นเคยให้ดี เพราะว่าเราจะใช้ทริคนี้อีกเยอะเลยในเคมีควอนตัม สรุปคือสมการ (1.2.17) นั้นแสดงให้เราเห็นว่าสัมประสิทธิ์ $a_k$ นั้นเป็น Inner Product $\langle \phi_k | a \rangle$ ในการกระจายของเวกเตอร์ $|a\rangle$

ถ้าหากว่าเรามีเวกเตอร์ 2 อัน เช่น $|a\rangle$ กับ $|b\rangle$ ที่ถูกเขียนให้กระจายในรูปของ Orthonormal Basis ที่เหมือนกัน ดังนี้

$$\begin{aligned} |a\rangle &= \sum_{i=1}^{N} a_i |\phi_i\rangle \\ |b\rangle &= \sum_{i=1}^{N} b_i |\phi_i\rangle \end{aligned} \tag{1.2.21}$$

เราจะสามารถเขียน Inner Product ของเวกเตอร์ทั้ง 2 อันนี้ได้ตามนี้

$$\begin{aligned}\langle a|b\rangle &= \left\langle \sum_{i=1}^{N} a_i\phi_i \middle| \sum_{j=1}^{N} b_j\phi_j \right\rangle = \sum_{i=1}^{N}\sum_{j=1}^{N} a_i^* \underbrace{\langle \phi_i|\phi_j\rangle}_{\delta_{ij}} b_j \\ &= \sum_{i=1}^{N} a_i^* b_i = (\alpha_1^*, \alpha_2^*, \ldots)\begin{pmatrix} b_1 \\ b_2 \\ \vdots \end{pmatrix} = \boldsymbol{a}^\dagger \boldsymbol{b}\end{aligned} \tag{1.2.22}$$

โดยเทอมขวาสุดของสมการนี้เป็นการเขียน Inner Product ในเทอมของ Dot Product ระหว่างเวกเตอร์ ($\boldsymbol{a}^\dagger\boldsymbol{b}$) แล้วก็ตัวห้อย "$^\dagger$" นั้นคือ **Conjugate Transpose ของเวกเตอร์**[1] ซึ่งนี่แสดงให้เห็นว่าในการหา Orthonormal Basis นั้น เราสามารถเขียนกระจาย Inner Product ของเวกเตอร์ 2 อันได้โดยการใช้ Inner Product ของสัมประสิทธิ์ของเวกเตอร์

### 1.2.4 ตัวดำเนินการเชิงเส้น

"ตัวดำเนินการ (Operators) คืออะไร?" ในหัวข้อนี้เราจะมาหาคำตอบกัน ผมคิดว่านิยามอย่างเป็นทางการของ "ตัวดำเนินการ" ที่พอจะเข้าใจได้ง่ายหน่อยนั่นก็คือ "วัตถุทางคณิตศาสตร์ (Mathematical Objects) แบบหนึ่งที่สามารถแปลงเวกเตอร์อันหนึ่งไปเป็นเวกเตอร์อีกอันหนึ่งได้ (เปลี่ยนเวกเตอร์อันเก่าให้เป็นอันใหม่)"

$$\hat{A}\,|a\rangle = |b\rangle \tag{1.2.23}$$

สำหรับเคมีควอนตัมนั้นเราจะสนใจเฉพาะ **ตัวดำเนินการเชิงเส้น** (Linear Operators) เป็นพิเศษ ซึ่งจะต้องสอดคล้องกับเงื่อนดังต่อไปนี้ด้วย

$$\hat{A}(\alpha\,|a\rangle + \beta\,|b\rangle) = \alpha\hat{A}\,|a\rangle + \beta\hat{A}\,|b\rangle \tag{1.2.24}$$

นอกจากนี้แล้วยังมีเรื่องของการกระทำ (Action) ของตัวดำเนินการต่อเวกเตอร์อีกด้วย โดยเราจะใช้สิ่งที่เรียกว่า **Matrix Representation** ซึ่งผมอยากจะให้ผู้อ่านลองดูสมการด้านล่างต่อไปนี้ก่อน

[1] เราสามารถคำนวณ Conjugate Tranpose ได้โดยการนำเมทริกซ์มาทำการ Tranpose แล้วคำนวณ Complex Conjugate ของสมาชิกแต่ละตัว

$$\begin{aligned}\sum_{i=1}^{N} \underbrace{\langle\phi_k| \hat{A} |\phi_i\rangle}_{A_{ki}} a_i &= \sum_i b_i \langle\phi_k|\phi_i\rangle \\ \sum_{i=1}^{N} A_{ki} a_i &= b_k\end{aligned} \tag{1.2.25}$$

จะเห็นได้ว่าเราสามารถทำการคำนวณ Inner Product ระหว่างเวกเตอร์และโอเปอร์เรเตอร์ $\hat{A}$ ได้ ซึ่งสิ่งที่เราได้ออกมานั้นคือ $A_{ki}$ ซึ่งก็คือ Matrix Representation ของโอเปอร์เรเตอร์ $\hat{A}$ นั่นเอง โดย Matrix Representation อันนี้มีนิยามคือ $A_{ki} = \langle\phi_k| \hat{A} |\phi_i\rangle$ แล้วเรายังสามารถเขียนสมการด้านบนนี้ให้กระชับกว่านี้ได้โดยการใช้ Matrix-Vector Notation ดังนี้

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{a} \tag{1.2.26}$$

โดยที่ $\boldsymbol{A}$ คือเมทริกซ์ที่มีสมาชิกแต่ละตัวเป็น $(\boldsymbol{A})_{ki} = \langle\phi_k| \hat{A} |\phi_i\rangle$

ลำดับต่อไปเราจะมาดูรายละเอียดว่าเราจะสามารถเขียนกระจายโอเปอร์เรเตอร์ในรูปของ Outer Product ของสมาชิกแต่ละตัวของเบซิส ซึ่ง **Outer Product** ของเวกเตอร์ 2 อัน ($|a\rangle$ กับ $|b\rangle$) นั้นมีสมการดังต่อไปนี้

$$|a\rangle \langle b| \tag{1.2.27}$$

เมื่อเรานำ Outer Product มาคูณกับ Ket Vector (โดยให้ Ket Vector นั้นอยู่ทางด้านขวา) เราจะได้ Inner Product เกิดขึ้นมาในทางด้านซ้ายนั้นเอง เช่น

$$(|a\rangle \langle b|) |c\rangle = |a\rangle \langle b|c\rangle \tag{1.2.28}$$

และในทำนองเดียวกัน ถ้าหากว่าเรานำ Ket Vector ไปคูณทางด้านซ้ายของ Outer Product เราก็จะได้ Inner Product ที่คล้ายกัน นอกจากนี้เรายังพบอีกว่าโอเปอร์เรเตอร์ $\hat{A}$ ที่เรานำไปใช้กับ State $|a\rangle$ นั้นจะให้ผลลัพธ์ที่เป็น State $|b\rangle$ ออกมานั่นเอง กล่าวคือ $\hat{A} |a\rangle = |b\rangle$ สามารถเขียนได้ดังนี้

$$\hat{A} = |b\rangle \langle a| \tag{1.2.29}$$

นั่นก็เพราะว่า

$$\hat{A} |a\rangle = |b\rangle \underbrace{\langle a|a\rangle}_{=1} = |b\rangle \tag{1.2.30}$$

ปกติแล้วเราสามารถ Represent ตัวดำเนินการเชิงเส้นได้โดยการใช้ผลรวมของ Outer Product นำมาคูณด้วย Matrix Elements ตัวอย่างเช่น ให้ผู้อ่านลองพิจารณา Orthonormal Basis $\{\phi_i\}$ และผลรวมของ Outer Product ของสมาชิกของแต่ละเบซิส ($\{\phi_i\}$) ดังนี้

$$\hat{1} = \sum_i |\phi_i\rangle \langle\phi_i| \tag{1.2.31}$$

ซึ่งเราจะได้นิยามปริมาณอันใหม่นี้ได้ว่าเป็น **ตัวดำเนินการเอกลักษณ์** (Identity Operator) และเพื่อยืนยันว่าสมการที่ (1.2.31) นั้นถูกต้อง เราสามารถตรวจสอบได้โดยการนำ $\hat{1}$ ไปกระทำกับเวกเตอร์ทั่วไป $|a\rangle$ ดังนี้

$$\hat{1}\,|a\rangle = \sum_i |\phi_i\rangle \underbrace{\langle\phi_i|a\rangle}_{a_i} = \sum_i |\phi_i\rangle\, a_i = |a\rangle \tag{1.2.32}$$

แล้วผมก็อยากจะเน้นด้วยว่าเราจะใช้เงื่อนไข $\hat{1} = \sum_i |\phi_i\rangle \langle\phi_i|$ ได้เฉพาะกับ Orthonormal Basis เท่านั้น

### 1.2.5 ความเชื่อมโยงระหว่าง Representations ของฟังก์ชันคลื่น

หัวข้อย่อยถัดมาที่ผมอยากให้ผู้อ่านศึกษาก็คือ Formalism ที่เกี่ยวข้องกับกลศาสตร์ควอนตัม โดยเฉพาะ Representation ที่เรานำมาใช้ในการแสดงฟังก์ชันคลื่น (Wavefunction) เพื่ออธิบายให้เห็นภาพมากขึ้น เราจะเริ่มด้วยการพิจารณาเบซิสเชิงตำแหน่ง (Position Basis) $\{|x\rangle, -\infty < x < \infty\}$ ซึ่งเบซิสอันนี้เป็น Orthogonal ด้วย กล่าวคือ $\langle x|x'\rangle = 0$ ถ้าหากว่า $x \neq x'$ แล้วเราจะสามารถทำการ Normalize เบซิสอันนี้ให้เป็น Delta ได้ด้วย ดังนี้

$$\langle x|x'\rangle = \delta(x - x') \tag{1.2.33}$$

ซึ่งจะทำให้อินทิกรัลของเบซิสด้านบนนี้นั้นจะมีค่าเท่ากับ 1 ดังนี้

$$\int \langle x|x'\rangle\, dx = 1 \tag{1.2.34}$$

ดังนั้นผมจึงอยากจะสรุปให้เข้าใจง่าย ๆ ว่าแท้จริงแล้วนั้น *แล้วฟังก์ชันคลื่นในทางกลศาสตร์ควอนตัมนั้นไม่ใช่อะไรเลย แต่เป็นแค่ Representation ของเวกเตอร์ $|\Psi\rangle$ บนเบซิสเชิงตำแหน่ง* ดังนี้

$$\Psi(x) \equiv \langle x|\Psi\rangle \tag{1.2.35}$$

ซึ่งเราสามารถเขียน Identity Operator ในรูปของเบซิสเชิงตำแหน่งได้ดังนี้

$$\hat{1} = \int |x\rangle \langle x| \, dx \tag{1.2.36}$$

แล้วเราก็สามารถเขียน Inner Product $\langle \Psi | \Psi \rangle$ ในทางกลศาสตร์ควอนตัมได้ดังนี้

$$\langle \Psi | \Psi \rangle = \langle \Psi | \left( \int |x\rangle \langle x| \, dx \right) |\Psi\rangle \tag{1.2.37}$$

$$= \int \langle \Psi | x \rangle \langle x | \Psi \rangle \, dx \tag{1.2.38}$$

$$= \int \Psi(x)^* \Psi(x) dx \tag{1.2.39}$$

$$= \int |\Psi(x)|^2 dx \tag{1.2.40}$$

### 1.2.6 ตัวดำเนินการพิเศษ

ลำดับต่อไปคือเรื่องของตัวดำเนินการพิเศษแบบต่าง ๆ ที่เราจะต้องเจอในกลศาสตร์เคมีควอนตัม โดยตัวดำเนินการแรกนั้นก็คือเมทริกซ์พิเศษที่เรียกว่า **คอนจูเกตเอร์มีเชียน** (Hermitian Conjugate) เริ่มต้นด้วยการกำหนดให้มีตัวดำเนินการ $\hat{A}$ แล้วเราจะทำการนิยามว่า Hermitian Conjugate ของตัวดำเนินการนี้ก็คือ $\hat{A}^\dagger$ ซึ่งกระทำอยู่บน Dual Vector ($\hat{A}\,|a\rangle$) โดย Hermitian Conjugate นั้นจะต้องสอดคล้องกับเงื่อนไขต่อไปนี้

$$\langle a | \, \hat{A} \, | b \rangle = \langle b | \, \hat{A}^\dagger \, | a \rangle^* \tag{1.2.41}$$

แล้วก็คุณสมบัติที่สำคัญอย่างหนึ่งของ Hermitian Conjugate ก็คือเราสามารถสลับลำดับตำแหน่งของตัวดำเนินการ พูดง่าย ๆ คือ Hermitian Conjugate ของเวกเตอร์ 2 อันที่คูณกันนั้นจะเท่ากับ Hermitian Conjugate ของเวกเตอร์แต่ละตัวที่สลับตำแหน่งกันแล้วมาคูณกัน ดังนี้

$$(\hat{A}\hat{B})^\dagger = \hat{B}^\dagger \hat{A}^\dagger \tag{1.2.42}$$

ตัวดำเนินการพิเศษที่สำคัญมากอันหนึ่งก็คือ **ตัวดำเนินการเอร์มีเชียน** (Hermitian Operator) ซึ่งมีนิยามคือต้องเป็น Hermitian Conjugate ที่เหมือนกันกับตัวดำเนินการอันเดิม

$$\hat{A}^{\dagger} = \hat{A} \tag{1.2.43}$$

และด้วยคุณสมบัติพิเศษอันนี้นี่เองที่ทำให้ Hermitian Operator นั้นถูกนำมาใช้เยอะมากในเคมีควอนตัม เพราะว่ามีค่าไอเกนแบบจริง (Real Eigenvalues) ซึ่งเป็นผลเฉลยหรือคำตอบของสมการไอเกนเวกเตอร์ ดังนี้

$$\hat{A}\left|a\right\rangle = \lambda\left|a\right\rangle \tag{1.2.44}$$

ตัวดำเนินการพิเศษอีกตัวที่เราควรรู้ก็คือ **ตัวดำเนินการแบบเดี่ยว** (Unitary Operator) ซึ่งมีนิยามคือต้องเป็น Hermitian Conjugate ที่อินเวอร์ส (Inverse) ของตัวเองนั้นต้องเท่ากับตัวดำเนินการอันเดิม

$$\hat{A}^{\dagger} = \hat{A}^{-1} \tag{1.2.45}$$

สำหรับ Unitary Operator นั้นเรามีเงื่อนไขเพิ่มเติมต่อไปนี้ด้วย

$$\begin{aligned} \hat{A}^{\dagger}\hat{A} &= \hat{A}^{-1}\hat{A} = 1 \\ \hat{A}\hat{A}^{\dagger} &= \hat{A}\hat{A}^{-1} = 1 \end{aligned} \tag{1.2.46}$$

นอกจากนี้แล้ว Unitary Operator นั้นยังไม่เปลี่ยนค่า Norm ของ State ที่เรานำมันไปกระทำอีกด้วย ซึ่งสามารถพิสูจน์ให้ดูได้จากตัวอย่างดังต่อไปนี้

$$\hat{U}\left|a\right\rangle = \left|b\right\rangle \tag{1.2.47}$$

เมื่อเรานำ Unitary Operator เข้าไปกระทำกับ State $\left|a\right\rangle$ จะพบว่าเราก็จะยังได้ค่า Norm ของ $\left|a\right\rangle$ ที่เหมือนเดิม

$$\|\left|b\right\rangle\| = \sqrt{\langle b|b\rangle} = \sqrt{\langle a|\,\hat{U}^{\dagger}\hat{U}\,|a\rangle} = \sqrt{\langle a|\,\underbrace{\hat{U}^{-1}\hat{U}}_{\hat{1}}\,|a\rangle} = \sqrt{\langle a|a\rangle} = \|\left|a\right\rangle\| \tag{1.2.48}$$

เราจึงสรุปว่าได้ Unitary Operator มีความสามารถในการรักษาค่า Norm ของ State

สุดท้ายนี้ผมขอสรุปเกี่ยวกับความสำคัญของคณิตศาสตร์ต่อเคมีควอนตัมว่า ท้ายที่สุดแล้วคณิตศาสตร์ทุกหัวข้อนั้นก็มีความสำคัญเท่ากันหมดและเป็นรากฐาน (Foundation) ที่สำคัญในการพัฒนาวิธีใหม่ ๆ ให้ดีกว่าวิธีเดิม ๆ แต่ว่าเราจะได้นำความรู้ของแต่ละหัวข้อมาใช้มากหรือน้อยนั้นก็ขึ้นอยู่กับหัวข้อเฉพาะทางของ

กลศาสตร์ที่เรานั้นสนใจหรือทำงานอยู่ แม้ว่าอาจจะมีหัวข้อหลาย ๆ อันที่เราจำเป็นต้องใช้ตลอดเวลา เช่น เมทริกซ์หรือแคลคูลัส แต่ถ้าหากว่าเราไม่มีความรู้ของบางหัวข้อที่เราคิดว่าจะไม่ได้ใช้ เราก็คงจะมีปัญหาแน่ ๆ เมื่อถึงเวลาที่เราจำเป็นต้องใช้ความรู้เหล่านั้น

## 1.3 หน่วยอะตอม

ในการศึกษาอะตอมและโมเลกุลนั้น นักเคมีเชิงทฤษฎีมักจะมีการกำหนดหน่วยขึ้นมาเพื่อให้ง่ายต่อการศึกษาคุณสมบัติต่าง ๆ ของอะตอม เช่น ตำแหน่ง, มวล, โมเมนตัม ซึ่งหน่วย (Units) ที่จะช่วยให้ชีวิตของนักเคมีทฤษฎีง่ายมากมากขึ้นในการพัฒนาทฤษฎีและทำให้การคำนวณง่ายขึ้นนั้นควรจะต้องทำให้ค่าของคุณสมบัติต่าง ๆ ที่ได้กล่าวมานั้นมีค่าเท่ากับหรือเข้าใกล้ 1 ให้มากที่สุด ซึ่งจะช่วยให้เราไม่ต้องไปใช้ค่าจริง ๆ ของปริมาณต่าง ๆ ในการคำนวณ ตัวอย่างเช่น เราก็ไม่จำเป็นที่จะต้องใช้ค่ามวลจริง ๆ ของอิเล็กตรอน

หน่วยที่ถูกนำมาใช้มากที่สุดในเคมีควอนตัมนั้นก็คือหน่วยอะตอม (Atomic Units ย่อสั้น ๆ เป็น a.u.) โดย Atomic Units นี้ถูกกำหนดค่าดังนี้

$$\text{Electron Mass} = m_e = 1 \tag{1.3.1}$$

$$\text{Electron Charge} = e = 1 \tag{1.3.2}$$

$$\text{Action} = \hbar = \frac{h}{2\pi} = 1 \tag{1.3.3}$$

$$\text{Coulomb's Constant} = k_e = \frac{1}{4\pi\epsilon_0} = 1 \tag{1.3.4}$$

| Dimension | สัญลักษณ์ (ชื่อ) | ค่าในหน่วยอื่น |
|---|---|---|
| Length | $a_0$ (bohr) | 0.52918 Å = 0.52918 $10^{-10}$ m |
| Mass | $m_e$ | $9.1095 \times 10^{-31}$ Kg |
| Charge | $e$ | $1.6022 \times 10^{-19}$ C |
| Action | $\hbar$ | $1.05457 \times 10^{-34}$ J · s |
| Energy | $E_\mathrm{h}$ (Hartree) | 627.51 kcal/mol |
| | | 27.211 eV |
| | | 219474.63 $\mathrm{cm}^{-1}$ |
| | | $4.3598 \times 10^{-18}$ J |
| Time | | $2.41889 \times 10^{-17}$ s $\approx 1/41.3$ fs |

ตาราง 1.1 แสดง Conversion Factor ระหว่าง Atomic Units กับหน่วยอื่น ๆ และความเร็วของแสงในหน่วย Atomic Units คือ $\alpha^{-1} \approx 137$ a.u.

## 1.4 สมการชโรดิงเงอร์

สมการชโรดิงเงอร์เป็นสิ่งที่ช่วยให้เราสามารถเข้าใจพฤติกรรมของโมเลกุลได้ การที่เรารู้คำตอบหรือผลเฉลยของสมการนั้นนำไปสู่การเข้าใจข้อมูลต่าง ๆ ของโมเลกุล (พูดให้ครอบคลุมกว่านี้คือระบบแบบ Microscopic) ที่อุณหภูมิ 0 K โดยสมการชโรดิงเงอร์ที่ขึ้นกับเวลา (Time-Dependent Schrödinger Equation) นั้นมีหน้าตาดังนี้

$$\hat{H}\Psi\left(\vec{r}_{1\ldots N}, t\right) = \mathrm{i}\hbar\frac{\partial\Psi\left(\vec{r}_{1\ldots N}, t\right)}{\partial t} \tag{1.4.1}$$

โดยที่ตัวแปรในสมการมีดังนี้

- $\hat{H}$ คือโอเปอร์เรเตอร์ของพลังงาน
- $\Psi$ คือฟังก์ชันคลื่นที่ขึ้นอยู่กับพิกัดหรือตำแหน่งของอนุภาค (ในที่นี้คืออิเล็กตรอน) ทั้งหมด $N$ ตัว เราจึงใช้เวกเตอร์แทน $\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N$
- $i$ คือหน่วยจินตภาพ $\left(\sqrt{-1}\right)$
- $\hbar$ คือค่าคงที่ของพลังค์แบบลดรูป (Reduced Planck's constant) มีค่าเท่ากับ $1.054\,571\,82 \times 10^{-34}\ \mathrm{m^2\,kg\,s^{-1}}$

ตัวแปรที่น่าจะมีความสำคัญที่สุดก็คือ $\hat{H}$ ซึ่งเป็นโอเปอร์เรเตอร์ที่เป็นผลรวมของโอเปอร์เรเตอร์พลังงานศักย์และโอเปอร์เรเตอร์พลังงานจลน์ ดังนี้

$$\hat{H} = \hat{T} + \hat{V} \tag{1.4.2}$$

โดยที่โอเปอร์เรเตอร์พลังงานจลน์ของระบบ $\hat{T}$ นั้นก็คือผลรวมของโอเปอร์เรเตอร์พลังงานจลน์ของอนุภาคแต่ละตัวนั่นเอง

$$\hat{T} = \sum_{i=1}^{N}\frac{-\hbar^2}{2m_i}\nabla_i^2 \tag{1.4.3}$$

โดยที่ $m_i$ คือมวลของอนุภาค $i$, $N$ คือจำนวนของอนุภาค และ $\nabla_i^2$ คือ Laplacian ในพิกัดคาร์ทีเซียนของอนุภาค $i$ ซึ่งมีสมการดังนี้

$$\nabla_i^2 = \frac{\partial^2}{\partial x_i^2} + \frac{\partial^2}{\partial y_i^2} + \frac{\partial^2}{\partial z_i^2} \tag{1.4.4}$$

โดยที่ $\vec{r}_i = (x_i, y_i, z_i)$ คือเวกเตอร์ของตำแหน่งในพิกัดคาร์ทีเซียน

ส่วนพลังงานจลน์ของระบบ $(\hat{V}(\vec{r}_{1\ldots N}))$ นั้นจริง ๆ แล้วมีความซับซ้อนมากเพราะว่าประกอบไปด้วยพลังงานจลน์หลาย ๆ รูปแบบมารวมกันแล้วก็จะมีความเฉพาะต่อระบบที่เราศึกษา สำหรับเคมีนั้นระบบที่เราสนใจศึกษาคือโมเลกุล ดังนั้นโอเปอร์เรเตอร์พลังงานศักย์นั้นจะต้องสอดคล้องกับพลังงานศักย์ของนิวเคลียสและอิเล็กตรอนเป็นหลักซึ่งผู้อ่านจะได้ศึกษาในหัวข้อที่

คราวนี้เรากลับมาดูที่ฟังก์ชันคลื่นกันต่อ ถ้าหากว่าฟังก์ชันคลื่นของเรานั้นเป็นฟังก์ชันที่ขึ้นอยู่กับตำแหน่งของอนุภาคเพียงอย่างเดียวและไม่ขึ้นกับเวลา เราสามารถเขียนฟังก์ชันคลื่นของทั้งระบบให้อยู่ในรูปผลคูณของฟังก์ชันคลื่นของอนุภาคแต่ละตัวได้โดยใช้เทคนิคที่เรียกว่า Separation of Variables ซึ่งเราจะได้สมการดังนี้

$$\Psi(\vec{r}_{1\ldots N}, t) = \psi(\vec{r}_{1\ldots N})\,\theta(t) \tag{1.4.5}$$

ซึ่งเมื่อเรานำสมการด้านบนแทนเข้าไปในสมการชโรดิงเงอร์เราจะได้สมการดังนี้

$$\frac{1}{\psi}\hat{H}\psi = \mathrm{i}\hbar\frac{1}{\theta}\frac{\partial\theta}{\partial t} \tag{1.4.6}$$

เนื่องจากว่าฝั่งซ้ายของสมการนั้นเป็นฟังก์ชันที่ขึ้นกับ $\vec{r}_{1\ldots N}$ อย่างเดียวและฝั่งขวานั้นเป็นฟังก์ชันที่ขึ้นกับ $t$ ดังนั้นทั้งสองฝั่งของสมการนั้นจะต้องมีค่าเท่ากับค่าคงที่ค่าหนึ่งซึ่งก็คือพลังงานของระบบ $E$ แล้วเราจะได้ว่าสมการโชรดิงเงอร์ที่ขึ้นกับเวลานั้นจะเปลี่ยนแบบไม่ขึ้นกับเวลา (Time-Independent Schrödinger Equation) ซึ่งมีสมการดังนี้

$$\hat{H}\psi = E\psi \tag{1.4.7}$$

สำหรับ Hamiltonians เกือบทั้งหมด (ไม่ใช้ทุกอัน) นั้นจะมีผลเฉลยสำหรับ Time-Independent Schrödinger Equation ที่มีค่าที่แน่นอน (Quantized) สำหรับแต่ละ State $n$ ของอนุภาค ดังนี้

$$\hat{H}\psi_n = E_n\psi_n \tag{1.4.8}$$

ซึ่งเราสามารถตีความสมการที่ (1.4.8) ด้านบนได้ว่าอนุภาคควอนตัมที่อยู่ในสถานะที่ $n$ จะมีค่าพลังงานที่แน่นอนนั้น $E_n$ นอกจากนี้แล้วสมการด้านบนนั้นเป็นปัญหาแบบค่าไอเกน (Eigenvalue Problem) โดยที่ $E_n$ คือค่าไอเกนและ $\psi_n$ คือฟังก์ชันไอเกนของโอเปอร์เรเตอร์ $\hat{H}$ ซึ่งการที่เราจะแก้สมการ Time-Independent Schrödinger Equation นั้นจะต้องอาศัยเทคนิคพิเศษซึ่งจะได้ศึกษาต่อในบทต่อ ๆ ไป[1]

[1] ตั้งแต่ส่วนนี้ของหนังสือเป็นต้นไปสมการโชรดิงเงอร์ (Schrödinger Equation) นั้นจะหมายถึงสมการชโรดิงเงอร์แบบที่ไม่

## 1.5 แฮมิลโทเนียนเชิงโมเลกุล

ในวิชาเคมีควอนตัมนั้นเราจะนิยามว่าโมเลกุลนั้นประกอบไปด้วยอิเล็กตรอน $n$ ตัวและนิวเคลียส $N$ ตัว โดยมีคุณสมบัติดังต่อไปนี้

- อิเล็กตรอนมีประจุเท่ากับ $-e$
- อิเล็กตรอนมีมวลเท่ากับ $m_e$
- นิวเคลียสตัวที่ $I$ นั้นมีประจุเท่ากับ $Z_I e$
- นิวเคลียสตัวที่ $I$ มีมวลเท่ากับ $m_I$

โดยที่อิเล็กตรอนกับนิวเคลียสนั้นจะถูกพิจารณาว่าเป็นจุดประจุ (Point Charges)

โอเปอร์เรเตอร์พลังงานจลน์ $\hat{T}$ สำหรับโมเลกุลนั้นเราสามารถประยุกต์ใช้สมการที่ (1.4.3) ได้ ซึ่งก็คือพลังงานจลน์ของทั้งอิเล็กตรอนและนิวเคลียสรวมกัน ดังนี้

$$\hat{T} = \underbrace{\sum_{I=1}^{N} \frac{-\hbar^2}{2m_I}\nabla_I^2}_{\text{Nuclei}} + \underbrace{\sum_{i=1}^{n} \frac{-\hbar^2}{2m_e}\nabla_i^2}_{\text{Electrons}} \tag{1.5.1}$$

ส่วนโอเปอร์เรเตอร์พลังงานศักย์ $\hat{V}$ สำหรับโมเลกุลนั้นก็จะเป็นอันตรกิริยาคูลอมบ์ (Coulomb Interaction) ระหว่างจุดประจุ ดังนี้

$$\hat{V} = \underbrace{\sum_{I=1,1}^{N} \frac{Z_I Z_J e^2}{4\pi\varepsilon_0 R_{IJ}}}_{\text{Nucleus-Nucleus}} + \underbrace{\sum_{i=1}^{n}\sum_{I=1}^{N} \frac{-Z_I e^2}{4\pi\varepsilon_0 r_{iI}}}_{\text{Electron-Nucleus}} + \underbrace{\sum_{\substack{i=1,1\\ j=i+1}}^{n} \frac{e^2}{4\pi\varepsilon_0 r_{ij}}}_{\text{Electron-Electron}} \tag{1.5.2}$$

ซึ่งก็คืออันตรกิริยาระหว่างทุกคู่ที่เป็นไปได้ นั่นคือ Nucleus-Nucleus, Electron-Nucleus และ Electron-Electron ส่วน $Z_I e$ นั้นก็คือประจุของนิวเคลียสที่ $I$ ซึ่ง $Z_I$ นั้นก็คือเลขอะตอมของนิวเคลียส เช่น ไฮโดรเจนนั้นก็จะมี $Z_I = 1$ และคาร์บอนก็จะมี $Z_I = 6$ ส่วน $e$ นั้นคือประจุของอิเล็กตรอนซึ่งก็คือ $-e$ นั่นเอง

โดยปกติแล้วเรามักจะใช้ตัวห้อยที่เป็นอักษรภาษาอังกฤษตัวใหญ่ $I, J, K, \ldots$ เพื่อบ่งบอกถึงนิวเคลียส

ขึ้นกับเวลา (Time-Independent Schrödinger Equation) ถ้าหากผมต้องการที่จะใช้คำว่าสมการชโรดิงเงอร์แบบที่ขึ้นกับเวลา (Time-Dependent Schrödinger Equation) ก็จะเขียนใช้คำนี้ตรง ๆ เลย

และใช้ตัวอักษรตัวเล็กสำหรับอิเล็กตรอน แล้วก็จะใช้ $R$ แทนระยะห่างที่วัดจากนิวเคลียสและใช้ $r$ แทนระยะห่างที่วัดจากอิเล็กตรอนอย่างน้อยหนึ่งตัว ซึ่งเราสามารถใช้สมการต่อไปนี้ในการคำนวณ $r_{ij}$ ได้

$$r_{ij} = |\vec{r}_i - \vec{r}_j| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{1.5.3}$$

โดยที่ $r_{iI} = |\vec{r}_i - \vec{R}_I|$ และ $R_{IJ} = |\vec{R}_I - \vec{R}_J|$ นั้นก็มีนิยามแบบเดียวกัน

ถ้าเราเขียนโอเปอร์เรเตอร์พลังงานจลน์โดยใช้หน่วยอะตอม (a.u.) จะได้ดังนี้

$$\hat{T} = \underbrace{-\frac{1}{2}\sum_{I=1}^{N}\frac{1}{m_I}\nabla_I^2}_{\text{nuclei}} \underbrace{-\frac{1}{2}\sum_{i=1}^{n}\nabla_i^2}_{\text{electrons}} \tag{1.5.4}$$

และสำหรับโอเปอร์เรเตอร์พลังงานศักย์

$$\hat{V} = \underbrace{\sum_{I=1,J=I+1}^{N}\frac{Z_I Z_J}{R_I}}_{\text{nucleus-nucleus}} + \underbrace{\sum_{i=1}^{n}\sum_{I=1}^{N}\frac{-Z_I}{r_{iI}}}_{\text{Electron-Nucleus}} + \underbrace{\sum_{\substack{i=1,1\\ j=i+1}}^{n}\frac{1}{r_{ij}}}_{\text{Electron-Electron}} \tag{1.5.5}$$

ซึ่งถ้าหากเราเขียน Hamiltonian ของโมเลกุล $\hat{H}^{\text{mol}}$ โดยรวมโอเปอร์เรเตอร์ทั้งสองตัวเข้าด้วยกัน จะได้ดังนี้

$$\hat{H}^{\text{mol}} = \hat{T}_n + \hat{T}_e + \hat{V}_{nn} + \hat{V}_{en} + \hat{V}_{ee} \tag{1.5.6}$$

โดยสองเทอมแรกนั้นก็คือพลังงานจลน์และสามเทอมที่เหลือนั้นก็คือพลังงานศักย์คูลอมบ์

## 1.6 คุณสมบัติพื้นฐานของฟังก์ชันคลื่น

โอเครครับ เราได้ศึกษาโอเปอร์เรเตอร์ Hamiltonian ซึ่งเป็นโอเปอร์เรเตอร์สำหรับพลังงานกันไปแล้ว ในหัวข้อนี้เราจะมาดูรายละเอียดของฟังก์ชันคลื่นกัน โดยผมจะเริ่มการอธิบายคุณสมบัติของฟังก์ชันคลื่นก่อน ซึ่งถือว่าเป็นพื้นฐานสำคัญมาก ๆ ที่ผู้อ่านควรจะต้องเข้าใจก่อนที่จะไปศึกษาฟังก์ชันคลื่นแบบเชิงลึกต่อไป โดยคุณสมบัติของฟังก์ชันคลื่นที่ผมเลือกมาอธิบายนั้นจะเป็นคุณสมบัติที่สำคัญ ๆ เท่านั้นซึ่งจำเป็นและเพียงพอต่อการทำความเข้าใจในบทต่อ ๆ ไป

ก่อนอื่นเลยผมขออ้างการตีความฟังก์ชันคลื่นของบอนส์ (Born's Interpretation) ที่ว่า "$\psi_i^* \psi_i \, \mathrm{d}\tau$ นั้นคือความน่าจะเป็นสำหรับอนุภาคที่อยู่ในสภาวะ $i$ ในอนุภาคที่มีปริมาตรเล็กมาก ๆ $(\mathrm{d}\tau)$ โดยที่ $\psi_i^*$ นั้นแทนคอนจูเกตเชิงซ้อน (Complex Conjugate) ของฟังก์ชันคลื่น $\psi_i$" นั่นหมายความว่าฟังก์ชันคลื่นอาจจะมีส่วนเชิงซ้อนเป็นองค์ประกอบก็ได้ อย่างไรก็ตามความน่าจะเป็น $\psi_i^* \psi_i \, \mathrm{d}\tau$ มีเฉพาะส่วนจริงเป็นองค์ประกอบเท่านั้นซึ่งสอดคล้องกับเงื่อนไขที่ว่าคุณสมบัติของฟังก์ชันคลื่นนั้นจะต้องสามารถสังเกตได้ (Observable)

กำหนดให้ความน่าจะเป็นสำหรับสถานะที่ $i$ ซึ่งเขียนแทนด้วย $\rho_i(\vec{r})$ นั้นถูกทำให้เป็นปกติ (ถูก Normalized แล้ว) เราจะตีความได้ว่าความน่าจะเป็นรวมที่จะพบอนุภาคที่ตำแหน่งไหนก็ตามใน Space นั้นจะมีค่าเท่ากับ 1 ซึ่งเขียนแทนด้วยสมการดังนี้

$$\begin{aligned} \int_{\text{all space}} \rho_i \, \mathrm{d}\tau &= \int_{\text{all space}} \psi_i^* \psi_i \, \mathrm{d}\tau \\ &= 1 \end{aligned} \tag{1.6.1}$$

โดยที่ $\mathrm{d}\tau$ คือปริมาตรในพิกัดคาร์ทีเซียนสำหรับอนุภาคหนึ่งตัวซึ่งมีนิยามแบ่งตามพิกัดอ้างอิง ดังนี้

- พิกัดคาร์ทีเซียน $\mathrm{d}\tau = \mathrm{d}x \, \mathrm{d}y \, \mathrm{d}z$
- พิกัดเชิงขั้วมีนิยามคือ $\mathrm{d}\tau = r^2 \sin(\theta) \mathrm{d}r \, \mathrm{d}\theta \mathrm{d}\varphi$

ซึ่งถ้าหากเราทำอินทิเกรตทั่วทั้งปริมาตรเราสามารถละขอบเขตการอินทิเกรตออกไปได้ ดังนี้

$$\int_{\text{all space}} \ldots \mathrm{d}\tau \equiv \int \ldots \mathrm{d}\tau \tag{1.6.2}$$

นอกจากนี้เรายังพบว่าถ้า $\psi$ นั้นถูก Normalized แล้ว $\Psi(t)$ ก็จะถูก Normalized ด้วย ซึ่งเราก็จะได้ความสัมพันธ์ดังนี้

$$\Psi^*(t)\Psi(t) = \psi^* \psi \tag{1.6.3}$$

สำหรับนิยามถัดมาก็คือโอเปอร์เรเตอร์ $\hat{\Omega}$ ซึ่งมีค่าคาดหวัง (Expectation Value) สำหรับระบบในสถานะ $i$ $(\langle \Omega \rangle_i)$ ดังนี้

$$\langle \Omega \rangle_i \equiv \frac{\int \psi_i^* \hat{\Omega}_i \psi_i \, \mathrm{d}\tau}{\int \psi_i^* \psi_i \, \mathrm{d}\tau} \tag{1.6.4}$$

สำหรับฟังก์ชันคลื่นที่ถูก Normalized แล้วนั้น $(\langle \Omega \rangle_i)$ จะกลายเป็น

$$\langle \Omega \rangle_i = \int \psi_i^* \hat{\Omega}_i \, \mathrm{d}\tau \tag{1.6.5}$$

แล้วก็ถ้าหากว่า $\psi_i$ เป็นฟังก์ชันไอเกนของ $\hat{\Omega}$ เราจะได้ว่า

$$\langle \Omega \rangle_i = \frac{\int \psi_i^* \hat{\Omega} \psi_i \, \mathrm{d}\tau}{\int \psi_i^* \psi_i \, \mathrm{d}\tau} = \frac{\Omega_i \int \psi_i^* \psi_i \, \mathrm{d}\tau}{\int \psi_i^* \psi_i \, \mathrm{d}\tau} = \Omega_i \tag{1.6.6}$$

นั่นหมายความว่า Expectation Value นั้นมีค่าเท่ากับค่าไอเกน (Eigenvalue) หรือ $\Omega_i$ นั่นเอง ดังนั้นเราจึงสามารถเขียนนิยามของพลังงานของระบบในสถานะ $i$ $(E_i)$ ในรูปของ Expectation Value ของ Hamiltonian สำหรับฟังก์ชันคลื่นที่ถูก Normalized แล้วได้ดังนี้

$$E_i = \int \psi_i^* \hat{H} \psi_i \, \mathrm{d}\tau \tag{1.6.7}$$

อย่างไรก็ตามในการพิสูจน์สมการต่าง ๆ ในกลศาสตร์ควอนตัมนั้น ถ้าหากว่าเราต้องมาเขียนนิยามของพลังงานหรือปริมาณอื่น ๆ โดยใช้สมการคณิตศาสตร์ตามด้านบนนั้นก็จะมีความยุ่งยากและเสียเวลา ดังนั้นเพื่อเป็นการทำให้การเขียนนิยามต่าง ๆ นั้นง่ายและกระชับขึ้น Paul Dirac จึงได้เสนอให้ใช้สัญกรณ์ที่เรียกว่า *Dirac bra-c-ket Notation* ดังนี้

$$\left\langle \psi_i | \hat{\Omega} | \psi_j \right\rangle \equiv \langle i | \hat{\Omega} | j \rangle \equiv \int \psi_i^* \hat{\Omega}_j \, \mathrm{d}\tau \tag{1.6.8}$$

โดยที่ $\langle \psi_i |$ หรือ $\langle i |$ นั้นเรียกว่า bra ของฟังก์ชันคลื่นของสถานะที่ $i$ และอีกตัวก็คือ $|\psi_j\rangle$ หรือ $|j\rangle$ นั้นเรียกว่า ket ซึ่งใช้แทนฟังก์ชันคลื่นของสถานะที่ $j$

เนื่องจากว่า Hamiltonian ของโมเลกุลนั้นมีคุณสมบัติที่เป็นเมทริกซ์แบบ Hermitian ดังนั้นเราจึงสามารถใช้คุณสมบัติการเปลี่ยนรูปดังต่อไปนี้ได้

$$\int \psi_i^* \hat{\Omega} \psi_j \, \mathrm{d}\tau = \int \left( \hat{\Omega} \psi_i \right)^* \psi_j \, \mathrm{d}\tau \tag{1.6.9}$$

ซึ่งเราพบว่าค่าไอเกนของมันนั้นเป็นส่วนจริงเท่านั้นและฟังก์ชันไอเกนนั้นเป็น Orthogonal นอกจากนี้ถ้าหากเรามาดูที่พลังงานของระบบ เราจะพบว่าพลังงานนั้นเป็นปริมาณที่สามารถวัดค่าได้ ดังนั้นพลังงานนั้นจะต้องเป็นค่าจริง (Real) เสมอ นี่จึงเป็นการยืนยันได้อีกว่าโอเปอร์เรเตอร์ของพลังงาน (Hamiltonian) นั้นจะต้องเป็น Hermitian

สำหรับ Orthonormal States (สถานะของฟังก์ชันคลื่นที่เป็นทั้ง Orthogonal และ Normalized)

$$\int \psi_i^* \psi_j \, \mathrm{d}\tau \equiv \langle \psi_i | \psi_j \rangle \equiv \langle i | j \rangle = \delta_{ij} \tag{1.6.10}$$

โดยที่ $\delta_{ij}$ นั้นคือ Kroenecker Delta Function ซึ่งจะมีค่าเท่ากับ 1 เมื่อ $i = j$ และเท่ากับ 0 เมื่อ $i \neq j$

ถ้าผู้อ่านต้องการศึกษาละเอียดมากกว่านี้ผมแนะนำหนังสือ Molecular Quantum Mechanics แต่งโดย Peter W. Atkins และ Ronald S. Friedman[1]

## 1.7 การประมาณของบอร์น-ออพเพนไฮเมอร์

การแก้สมการชโรดิงเงอร์นั้นมีความซับซ้อนมาก ดังนั้นนักฟิสิกส์และนักเคมีเชิงทฤษฎีจึงได้พยายามพัฒนาทฤษฎีเสริมต่าง ๆ เพื่อมาช่วยในการหาคำตอบ หนึ่งในเทคนิคที่สำคัญมากในการจัดการกับฟังก์ชันคลื่นของโมเลกุล (ระบบที่มีอิเล็กตรอนและนิวเคลียสหลาย ๆ ตัวอยู่ด้วยกัน) นั้นก็คือ **การประมาณของบอร์น-ออพเพนไฮเมอร์ (Born-Oppenheimer Approximation)** ซึ่งเป็นการประมาณที่เขียนฟังก์ชันคลื่นของโมเลกุล $\psi\left(\vec{R}_{1\ldots N}, \vec{r}_{1\ldots n}\right)$ ให้อยู่ในรูปของผลคูณระหว่างฟังก์ชันคลื่นของอิเล็กตรอน $\left(\psi^{\mathrm{el}}\right)$ และฟังก์ชันคลื่นของนิวเคลียส $\left(\psi^{\mathrm{nuc}}\right)$ ได้ พูดง่าย ๆ คือเราสามารถแยกชิ้นส่วนของฟังก์ชันคลื่น (Separation) ออกจากกันได้ ดังนี้

$$\psi\left(\vec{R}_{1\ldots N}, \vec{r}_{1\ldots n}\right) \approx \psi^{\mathrm{el}}\left(\vec{r}_{1\ldots n}; \vec{R}_{1\ldots N}\right) \psi^{\mathrm{nuc}}\left(\vec{R}_{1\ldots N}\right) \tag{1.7.1}$$

โดยที่ $\psi^{\mathrm{el}}$ คือฟังก์ชันพิกัดเชิงอิเล็กทรอนิกส์ $\vec{r}_{1\ldots n}$ ซึ่งขึ้นอยู่กับพิกัดของนิวเคลียสด้วย $\hat{R}_{1\ldots N}$ โดย Hamiltonian $\hat{H}$ ที่สอดคล้องกันนั้นมีสมการดังต่อไปนี้

$$\begin{aligned} \hat{H}^{\mathrm{el}} &= -\frac{1}{2}\sum_{i=1}^{n} v_i^2 - \sum_{i=1}^{n}\sum_{I=1}^{N} \frac{z_i}{r_{iI}} + \sum_{\substack{i=1 \\ j=i+1}}^{n} \frac{1}{r_{ij}} + \sum_{\substack{I=1 \\ J=I+1}}^{N} \frac{z_I z_J}{R_{IJ}} \\ &= \hat{T}_e + \hat{V}_{en} + \hat{V}_{ee} + \hat{V}_{nn} \end{aligned} \tag{1.7.2}$$

ซึ่งจะเห็นได้ว่าสมการด้านบนนั้นจะไม่มีเทอมโอเปอร์เรเตอร์พลังงานจลน์ของนิวเคลียส ซึ่งจะต่างจากกรณีของ Hamiltonian ก่อนหน้านี้ (สมการที่ (1.5.6)) และเทอมสุดท้ายของสมการที่ (1.7.2) ซึ่งก็คือพลังงานศักย์ระหว่างนิวเคลียสนั้นจะเป็นค่าคงที่เนื่องจากว่าพิกัดตำแหน่งของนิวเคลียสนั้นจะถูกมองว่าเป็นพารามิเตอร์และไม่ใช่ตัวแปรในฟังก์ชันคลื่นเชิงอิเล็กทรอนิกส์ $\psi^{\mathrm{el}}$ ดังนั้นสมการชโรดิงเงอร์สำหรับสถานะเชิงอิเล็กทรอนิกส์ที่ $i$ จึงมีหน้าตาดังนี้

[1] https://global.oup.com/academic/product/molecular-quantum-mechanics-9780199541423

$$\hat{H}^{\text{el}}\psi_i^{\text{el}} = \epsilon_i^{\text{el}}\psi_i^{\text{el}} \tag{1.7.3}$$

ถ้าหากว่าเราทำการแก้สมการ (1.7.3) สำหรับโมเลกุลเดียวกันแต่ว่ามีโครงสร้าง (Molecular Geometries หรือ $\vec{R}_{1\ldots N}$) ที่แตกต่างกันหลาย ๆ โครงสร้างไปเรื่อย ๆ เราจะสามารถพลอตพื้นผิวพลังงานศักย์ (Potential Energy Surface) สำหรับสถานะ $i$ $\left(\epsilon_0^{\text{el}}\vec{R}_{1\ldots N}\right)$ ที่เป็นสถานะพื้น (Ground State) ได้ดังนี้

$$V\left(\bar{R}_{1\ldots N}\right) = \epsilon_0^{\text{el}}\left(\hat{R}_{1\ldots N}\right) \tag{1.7.4}$$

คราวนี้เราลองมาดูกรณีที่เราสนใจเฉพาะนิวเคลียสกันบ้าง เราสามารถกำหนด Hamiltonian สำหรับนิวเคลียสดังนี้

$$H^{\text{nuc}} = -\sum_{I=1}^{N}\frac{1}{2m_I}V_I^2 + V\left(\vec{R}_{1\ldots N}\right) \tag{1.7.5}$$

และสมการชโรดิงเงอร์ของนิวเคลียสนั้นคือ

$$\hat{H}^{\text{nuc}}\psi_k^{\text{nuc}} = \epsilon_k^{\text{nuc}}\psi_k^{\text{nuc}} \tag{1.7.6}$$

โดยสรุปแล้วถ้าหากว่าเรานำ Born-Oppenheimer Approximation มาใช้ เราจะสามารถแบ่งเคมีควอนตัมออกได้เป็น 2 ปัญหาหลัก ๆ ดังนี้

1. ปัญหาเชิงอิเล็กทรอนิกส์ที่เราจะต้องแก้สมการชโรดิงเงอร์สำหรับ Molecular Geometry ที่ต้องการศึกษา
2. ปัญหาเชิงนิวเคลียสซึ่งเป็นการคำนวณหา Potential Energy Surface โดยการแก้สมการชโรดิงเงอร์เชิงอิเล็กทรอนิกส์สำหรับหลาย ๆ Molecular Geometries

## 1.8 ออร์บิทัลเชิงอะตอม

### 1.8.1 อะตอมที่มีอิเล็กตรอน 1 ตัว

การที่เราจะหาวิธีในการแก้สมการชโรดิงเงอร์นั้นก็ควรที่จะเริ่มต้นศึกษาจากระบบที่ง่าย ๆ ก่อนซึ่งระบบที่ง่ายที่สุดนั้นก็คืออะตอมที่มีอิเล็กตรอนเพียงแค่ 1 ตัวเท่านั้น (One-Electron Atom) ซึ่งตำแหน่งของนิวเคลียสนั้นไม่ถูกนำมาพิจารณาในการแก้สมการเพราะว่าเราใช้การประมาณของ Born-Oppenheimer

ซึ่งผู้อ่านเพิ่งได้ศึกษาไปในหัวข้อที่แล้ว โดย Hamiltonian สำหรับอิเล็กตรอนที่มีอันตรกิริยากับนิวเคลียสในหน่วย Atomic Units (a.u.) นั้นมีหน้าตาดังต่อไปนี้

$$\hat{H}^{\text{el}} = -\frac{1}{2}\nabla^2 - \frac{Z}{r} \tag{1.8.1}$$

โดยที่ $Z$ คือประจุของนิวเคลียสและ $r$ คือระยะห่างระหว่างอิเล็กตรอนและนิวเคลียส เราพบว่าโอเปอร์เรเตอร์ Hamiltonian นี้ประกอบไปด้วยเทอมโอเปอร์เรเตอร์พลังงานจลน์และพลังงานศักย์คูลอมบ์ซึ่งพอเราพิจารณาสมการนี้แล้วนั้นมีความเรียบง่ายมากกว่า โดยผลเฉลยของสมการชโรดิงเงอร์ในระบบพิกัดเชิงขั้วเมื่อใช้ Hamiltonian Operator ตัวนี้คือ

$$\psi_{nlm_l}(r,\theta,\varphi) = R_{nl}(r)Y_{lm_l}(\theta,\varphi) \tag{1.8.2}$$

โดยที่ $R_{nl}(r)$ คือฟังก์ชันรัศมี (Radial Function) และ $Y_{lm_l}(\theta,\varphi)$ คือฟังก์ชันฮาร์โมนิกทรงกลม (Spherical Harmonics) ซึ่งผลเฉลยของทั้งฟังก์ชันทั้งสองอันนี้อยู่กับเลขควอนตัม 3 ตัวคือเลขควอนตัมหลัก $n$ 1 ตัว และเลขควอนตัมเชิงมุม $l$ และ $m_l$ อีกสองตัวซึ่งมีเงื่อนไขความสัมพันธ์ของค่าของเลขควอนตัมดังนี้

$$\begin{aligned} &n = 1, 2, 3 \ldots \\ &l = 0, 1, 2 \ldots, n-1 \\ &m = 0, \pm 1, \pm 2, \ldots \pm l \end{aligned}$$

ออร์บิทัลเชิงอะตอมนั้นถูกนำมาใช้ในการสร้างเซตของฟังก์ชัน Orthogonal ดังนี้

$$\int \psi^*_{nlm_l}(r,\theta,\varphi)\psi_{n'l'm'_l}(r,\theta,\varphi)\mathrm{d}\tau = \delta_{nn'}\delta_{ll'}\delta_{m_lm'_l} \tag{1.8.3}$$

สำหรับอิเล็กตรอนแต่ตรอนแต่ละตัวนั้นสามารถมีได้ 2 สปินซึ่งจะแทนด้วยเลขควอนตัมสปิน $m_s$

$$m_s = \pm\frac{1}{2} \tag{1.8.4}$$

ในออร์บิทัลเชิงอะตอมแต่ละอันนั้นสามารถที่จะบรรจุอิเล็กตรอนได้เพียง 2 ตัวเท่านั้นโดยจะต้องมีสปินตรงข้ามกันตามหลักของเพาลี (Pauli Principle) นั้นคืออิเล็กตรอนแต่ละตัวนั้นจะมีชุดเลขควอนตัมที่เฉพาะและห้ามซ้ำกัน ($n$, $l$, $m_l$, และ $m_s$)

ตัวอย่างของการจัดเรียงอิเล็กตรอนสำหรับอะตอมโดยใช้หลักเอาฟเบา (Aufbau Principle) มีดังนี้

- He $1s^2$

- Ne $1s^2 2s^2 2p^5$
- Cl $1s^2 2s^2 2p^6 3s^2 2p^3$ หรือ Ne $3s^2 2p^5$

### 1.8.2 อะตอมที่มีอิเล็กตรอน 2 ตัว

ในหัวข้อนี้เราจะมาศึกษาระบบที่ซับซ้อนเพิ่มขึ้นมาอีกหน่อยนึงนั่นก็คืออะตอมที่มีอิเล็กตรอน 2 ตัว ซึ่งเรายังคงใช้หลักการเดิมในการวิเคราะห์ Hamiltonian นั่นก็คือเริ่มด้วยผลรวมของโอเปอร์เรเตอร์ของพลังงานของอันตรกิริยาระหว่างอิเล็กตรอนกับนิวเคลียส ดังต่อไปนี้

$$\hat{H}^{\text{el}} = \underbrace{-\frac{1}{2}\nabla_i^2 - \frac{Z}{r_i}}_{\hat{H}_i} \underbrace{-\frac{1}{2}\nabla_j^2 - \frac{Z}{r_j}}_{\hat{H}_j} + \frac{1}{r_{ij}} \tag{1.8.5}$$

โดยที่ $i$ กับ $j$ คือดัชนีหรือ Index ของอิเล็กตรอนซึ่งมีอันตรกิริยา (Interaction) กับนิวเคลียส 1 ตัว สรุปก็คือว่า Hamiltonian ที่แสดงตามสมการด้านบนนี้ประกอบไปด้วย 5 เทอม ดังนี้

- พลังงานจลน์ของอิเล็กตรอนแต่ละตัว (เทอมที่ 1 และ 3)
- พลังงานคูลอมบ์ระหว่างอิเล็กตรอนแต่ละตัวกับนิวเคลียส (เทอมที่ 2 และ 4)
- พลังงานคูลอมบ์ระหว่างอิเล็กตรอน (เทอมที่ 5)

ถ้าสมมติว่าเราตัดเทอมสุดท้ายที่เป็นแรงผลักระหว่างอิเล็กตรอน-อิเล็กตรอน (Electron Repulsion) ออกไป เราจะได้ Hamiltonian ดังต่อไปนี้

$$\hat{H}^{\text{el}} = \hat{H}_i + \hat{H}_j \tag{1.8.6}$$

ซึ่งถ้าเรานำ Hamiltonian ตามสมการ (1.8.6) ไปใช้ในสมการชโรดิงเงอร์สำหรับออร์บิทัล (One-Electron Wavefunction) หรือ $\phi_i(\vec{r}_i)$ เราจะได้สมการชโรดิงเงอร์สำหรับอะตอมที่มีอิเล็กตรอน 2 ตัว ดังต่อไปนี้

$$\left(\hat{H}_i + \hat{H}_j\right) \phi_i(\vec{r}_i)\phi_j(\vec{r}_j) = (\epsilon_i + \epsilon_j)\, \phi_i(\vec{r}_i)\phi_j(\vec{r}_j) \tag{1.8.7}$$

โดยที่ $\epsilon_i$ คือพลังงานของออร์บิทัล แล้วก็เนื่องจากว่าอิเล็กตรอนนั้นคือเฟอร์มิออน (Fermion) ดังนั้นอิเล็กตรอนทุกตัวนั้นจึงมีคุณสมบัติเหมือนกันหมดและไม่สามารถแยกอิเล็กตรอนแต่ละตัวออกจากกันได้หรือภาษาอังกฤษก็คือ Indistinguishable (หมายความว่าจริง ๆ แล้วไม่มีอิเล็กตรอนตัวที่ 1, 2, หรือ 3) และฟังก์ชันคลื่นนั้นก็มีคุณสมบัติปฏิสมมาตร (Anti-Symmetry) เมื่อเราเทียบกับการสลับอิเล็กตรอน

ถ้าเราใส่อิเล็กตรอนทั้ง 2 ตัวเข้าไปในออร์บิทัล $1s$ เราจะกำหนดให้ $1s^2$ เป็นการแทนถึง Occupation ของออร์บิทัลซึ่งก็คือมีอิเล็กตรอนบรรจุอยู่ 2 ตัว ดังนั้นเราจะสามารถเขียนฟังก์ชันคลื่นได้ดังนี้

$$\psi(i,j) = 1s(i)1s(j) \tag{1.8.8}$$

โดยที่เราจะใช้ Notation $\vec{r}_i \equiv i$ และ $\vec{r}_j \equiv j$ อย่างไรก็ตามฟังก์ชันคลื่นในสมการ (1.8.8) นั้นแสดงถึงอิเล็กตรอน 2 ตัวที่ไม่สามารถแยกความแตกต่างกันได้ (Indistinguishable) แต่ว่าฟังก์ชันคลื่นนั้นไม่มีสมบัติ Anti-Symmetric แล้วทำไมถึงเป็นเช่นนี้กันหล่ะ? คำตอบก็คือฟังก์ชันคลื่นที่เราใช้ในการอธิบายออร์บิทัลที่ใส่อิเล็กตรอนอยู่นั้นมันขึ้นอยู่กับเพียงแค่พิกัดเชิงพื้นที่ (Spatial Coordinates) เท่านั้น ซึ่งฟังก์ชันคลื่นที่ถูกต้องนั้นควรจะต้องขึ้นอยู่กับสปิน (Spin) ของอิเล็กตรอนด้วยซึ่งถ้าเรารวมผลของสปินเข้าไปก็จะทำให้ฟังก์ชันคลื่นรวมนั้นมีคุณสมบัติ Anti-Symmetry

คราวนี้เราจะกำหนดให้ออร์บิทัลเชิงสปินนั้นคือ $\chi_i(\vec{r}_i, s_i)$ ซึ่งเราสามารถเขียนออร์บิทัลกระจายได้โดยเป็นผลคูณระหว่างฟังก์ชันเชิงพื้นที่และฟังก์ชันเชิงสปิน

$$\chi_i(\vec{r}_i, s_i) \equiv \phi_i(\vec{r}_i)\,\sigma_i(s_i) \tag{1.8.9}$$

โดยที่ $\sigma_i$ สามารถที่จะเป็น $\alpha$ สำหรับสปินขึ้น $m_s = \frac{1}{2}$ หรือจะเป็น $\beta$ สำหรับสปินลง $m_s = -\frac{1}{2}$ ก็ได้ ซึ่งถ้าเรานำมาเขียนรวมทั้งเราจะได้ฟังก์ชันคลื่นใหม่ที่รวมสปินเข้าไปด้วย ดังนี้

$$\psi(i,j) = 1s(i)\alpha(i) \times 1s(j)\beta(j) \tag{1.8.10}$$

แล้วเราก็จะทำการใช้เทคนิคการรวมเชิงเส้นหรือ Linear Combinations ของ Spin-Functions ในการทำให้ฟังก์ชันคลื่นตามสมการที่ (1.8.10) นั้นมีความ Anti-Symmetric ดังนี้

$$\begin{array}{ll} \frac{1}{\sqrt{2}}(\alpha(i)\beta(j) + \alpha(j)\beta(i)) & \text{Symmetric Spin-Function} \\ \frac{1}{\sqrt{2}}(\alpha(i)\beta(j) - \alpha(j)\beta(i)) & \text{Anti-Symmetric Spin-Function} \end{array} \tag{1.8.11}$$

โดยเราจะทำการตั้งสมมติฐานเพิ่มด้วยว่า Spin-Functions นั้นเป็น Orthonormal เช่น

$$\langle \alpha(i) | \beta(j) \rangle = \delta_{\alpha\beta}\delta_{ij} \tag{1.8.12}$$

ซึ่งมี $1/\sqrt{2}$ เป็น Normalization Factor และฟังก์ชันคลื่นอันใหม่สำหรับอิเล็กตรอน 2 ตัวที่มีคุณสมบัติ Anti-Symmetric ก็จะมีหน้าตาเป็นดังนี้

$$\psi(i,j) = 1s(i)1s(j) \times \frac{1}{\sqrt{2}}(\alpha(i)\beta(j) - \alpha(j)\beta(i)) \tag{1.8.13}$$

### 1.8.3 อะตอมที่มีอิเล็กตรอน $n$ ตัว

สำหรับระบบที่มีอิเล็กตรอน $n$ ตัวนั้นเราสามารถเขียนฟังก์ชันคลื่นที่อยู่ในรูปของผลรวมเชิงเส้นได้โดยใช้ Determinant ของเมทริกซ์จตุรัสขนาด $n \times n$ โดยเราจะเรียก Determinant นี้ว่า Slater Determinant

$$\psi = \frac{1}{\sqrt{n!}} \begin{vmatrix} \chi_1(1) & \chi_2(1) & \cdots & \chi_n(1) \\ \chi_1(2) & \chi_2(2) & \cdots & \chi_n(2) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_1(n) & \chi_2(n) & \cdots & \chi_n(n) \end{vmatrix} \tag{1.8.14}$$

ซึ่งจะมีคุณสมบัติ Anti-Symmetric รวมอยู่ในนั้นด้วยเพราะว่าใช้ Spin-Orbital ถ้าหากอยากรู้ว่าหน้าตาของฟังก์ชันคลื่นของระบบที่มีอิเล็กตรอนหลาย ๆ ตัว เช่น 10 ตัว เป็นอย่างไรก็ลองเขียน Slater Determinant ขนาด $10 \times 10$ แล้วลองหา Determinant ดูแล้วจะพบว่าจะมีเทอมที่ถูกกระจายออกมาทั้งหมด 20 เทอม

## 1.9 ออร์บิทัลเชิงโมเลกุล

ในหัวข้อนี้ถือว่าเป็นอีกหนึ่งหัวข้อที่สำคัญมาก ๆ ในการศึกษากลศาสตร์ควอนตัมเชิงโมเลกุลนั่นก็คือออร์บิทัลเชิงโมเลกุล (Molecular Orbitals) ซึ่งเราจะใช้ความรู้เกี่ยวกับออร์บิทัลเชิงอะตอมและฟังก์ชันคลื่นที่สร้างขึ้นจาก Spin-Orbital ที่เราเพิ่งได้ศึกษาไปนั้นมาใช้ในหัวข้อนี้ด้วย อย่างไรก็ตาม ความรู้คณิตศาสตร์ที่ใช้ในหัวข้อนี้ก็ยังคงเป็นพีชคณิตเชิงเส้นทั่วไปไม่ได้ซับซ้อนอะไรมาก

เริ่มต้นเลยก็คือนักวิทยาศาสตร์นั้นได้เสนอแบบจำลองหรือโมเดลที่ใช้ในการอธิบาย Molecular Orbitals $(\varphi_i)$ นั่นก็คือการใช้ผลรวมเชิงเส้น (Linear Combination) อีกเช่นเดิม ซึ่ง Molecular Orbitals นั้นก็คือผลรวมเชิงเส้นของ Atomic Orbitals $(\phi_j)$ ทั้งหมด $m$ ออร์บิทัล ดังนี้

$$\varphi_i = \sum_{j=1}^{m} c_{ij}\phi_j \tag{1.9.1}$$

โดยสมการด้านบนนี้มีชื่อเรียกตรงตัวเลยก็คือ Linear Combination of Atomic Orbitals (LCAO) Approximation มี $c_{ij}$ เป็นสัมประสิทธิ์ของแต่ละออร์บิทัล ตัวอย่างเช่น โมเลกุลไฮโดรเจน $H_2$ เราสามารถเขียน Molecular Orbital หรือ MO ได้ในรูปของผลรวมเชิงเส้นของ Atomic Orbital หรือ AO ของออร์บิทัล $1s$

ของไฮโดรเจนอะตอมแรก $(1s_A)$ และ $1s$ ของไฮโดรเจนอะตอมที่สอง $(1s_B)$ ซึ่ง MO ที่เกิดขึ้นนั้นก็คือพันธะซิกม่า $\sigma_i$ นั่นเอง ดังนี้

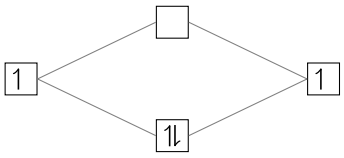
$$\sigma_i = c_{iA}1s_A + c_{iB}1s_B \tag{1.9.2}$$

ถ้าหากสมมติว่าตำแหน่งของนิวเคลียสของอะตอมไฮโดรเจนทั้ง 2 อะตอมนั้นอยู่ที่ $(x, 0, 0)$ และ $(-x, 0, 0)$ ในระบบพิกัดฉาก 3 มิติ ความน่าจะเป็นที่เราจะพบอิเล็กตรอน ณ ตำแหน่ง $x$ และ $-x$ นั้นจะเท่ากันนั่นก็เพราะว่าโมเลกุลไฮโดรเจนนั้นมีความสมมาตร ดังนั้นเรามีสมมติฐานเริ่มต้นว่าฟังก์ชันคลื่นนั้นจะต้องมีคุณสมบัติดังต่อไปนี้

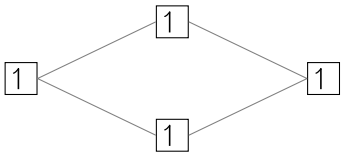
$$\psi^2(x) = \psi^2(-x) \tag{1.9.3}$$

สมการนี้มีคำตอบได้ 2 แบบนั่นก็คือแบบที่ฟังก์ชันคลื่นนั้นยังคงความ Symmetric ไว้กับแบบที่มีความ Anti-Symmetric ดังนี้

$$\psi(x) = \psi(-x) \quad \text{และ} \quad \psi(x) = -\psi(-x) \tag{1.9.4}$$




ภาพ 1.2 Ground State




ภาพ 1.3 Triplet State

ดังนั้น $1s_A$ และ $1s_B$ คือฟังก์ชันที่เหมือนกัน (Identical Functions) ซึ่งก็คือ AO ที่มีจุดศูนย์กลางอยู่ที่ $x$ และ $-x$ ตามลำดับ นอกจากนี้แล้วสัมประสิทธิ์ของออร์บิทัลยังสอดคล้องกันอีกด้วยด้วย

$$c_{Ai} = c_{Bi} \quad \text{และ} \quad c_{Ai} = -c_{Bi} \tag{1.9.5}$$

ถ้า AO นั้นเป็น Orthonormal (มีคุณสมบัติที่เป็น Orthogonal และมีความ Normality) เราจะได้ว่า Orthonormal MOs นั้นจะมีสมการดังต่อไปนี้

$$\sigma_g = \frac{1}{\sqrt{2}}(1s_A + 1s_B) \quad \text{และ} \quad \sigma_u = \frac{1}{\sqrt{2}}(1s_A - 1s_B) \tag{1.9.6}$$

โดยที่ $\sigma_g$ นั้นคือ $\sigma$-orbital ที่สร้างพันธะ (Bonding MO) โดย $g$ ย่อมาจากภาษาเยอรมันคำว่า gerade ที่แปลว่าคู่ และ $\sigma_u$ นั้นคือ $\sigma$-orbital ที่ต้านพันธะ (Antibonding MO) โดย $u$ ย่อมาจากภาษาเยอรมันคำว่า ungerade ที่แปลว่าคี่ สำหรับประเภทของ MO นั้นเราจะแบ่งออกได้เป็น 3 ประเภท ดังนี้

1. ออร์บิทัลแบบสร้างพันธะ (Bonding Orbitals) เป็นบริเวณที่มีความหนาแน่นของอิเล็กตรอนสูงระหว่างนิวเคลียสซึ่งอิเล็กตรอนในออร์บิทัลเหล่านี้จะยึดเหนี่ยวนิวเคลียสของอะตอมที่เกิดพันธะเข้าด้วยกัน

2. ออร์บิทัลแบบต้านพันธะ (Antibonding Orbitals) เป็นอิเล็กตรอนที่อยู่หลังนิวเคลียสและมีแนวโน้มที่จะดึงนิวเคลียสของอะตอมที่สร้างพันธะออกจากกันหรือทำให้ความแข็งแรงของพันธะอ่อนลงนั่นเอง

3. ออร์บิทัลแบบไม่สร้างพันธะ (Non-bonding Orbitals) อิเล็กตรอนที่ยังคงอยู่ในออร์บิทัลเชิงอะตอมของอะตอมคู่ร่วมพันธะ โดยอิเล็กตรอนในออร์บิทัลชนิดนี้จะไม่มีอันตรกิริยากับส่วนอื่นๆของโมเลกุลและไม่มีผลต่อความแข็งแรงของพันธะด้วย

โดยทั่วไปแล้วฟังก์ชันคลื่นของโมเลกุลนั้นจะถูกเขียนให้อยู่ในรูปของ Slater Determinant ที่มีสมาชิกในเมทริกซ์เป็น MO ตามที่เราได้ไปศึกษาไป สำหรับโมเลกุลที่มีอิเล็กตรอน $n$ ตัวนั้นจะมี MO ทั้งหมด $\frac{n}{2}$ อันที่จะมีอิเล็กตรอนบรรจุอยู่ซึ่ง MO เหล่านี้จะไม่ว่างหรือ Occupied นั่นเอง ส่วน MO ที่เหลือนั้นจะมีที่ว่างในการใส่อิเล็กตรอนซึ่งเราก็จะเรียกว่า Unoccupied โดยตัวอย่างด้านล่างก็คือเป็นฟังก์ชันคลื่นของโมเลกุลไฮโดรเจนที่สถานะพื้น (Ground State) ซึ่งเขียนแทนด้วย Slater Determinant ที่มี Occupied MOs เป็นสมาชิก สำหรับ Notation $\psi_0$ นั้นสามารถตีความเลข 0 ได้ว่าเป็นสถานะพื้นหรือสถานะที่ต่ำที่สุด

$$\begin{aligned}\psi_0 =& \frac{1}{\sqrt{2}}\begin{vmatrix} \chi_1(1) & \chi_2(1) \\ \chi_1(2) & \chi_2(2) \end{vmatrix} \\ =& \frac{1}{\sqrt{2}}(\chi_1(1)\chi_2(2) - \chi_1(2)\chi_2(1))\end{aligned} \tag{1.9.7}$$

โดยที่ $\chi_1(j) = \sigma_g(j)\alpha(j)$ และ $\chi_2(j) = \sigma_g(j)\beta(j)$ และฟังก์ชันคลื่นนั้นถูก Normalized แล้ว ถ้าหากเราพิจารณา Triplet State ของโมเลกุลไฮโดรเจนเราจะได้ว่าฟังก์ชันคลื่นนั้น $\psi_1$ นั้นจะกลายเป็น

$$\begin{aligned}\psi_1 =& \frac{1}{\sqrt{2}} \begin{vmatrix} \chi_1(1) & \chi_3(1) \\ \chi_1(2) & \chi_3(2) \end{vmatrix} \\ =& \frac{1}{\sqrt{2}} \begin{vmatrix} \sigma_g(1)\alpha(1) & \sigma_u(1)\alpha(1) \\ \sigma_g(2)\alpha(2) & \sigma_u(2)\alpha(2) \end{vmatrix}\end{aligned} \tag{1.9.8}$$

โดยที่เรากำหนดให้ $\chi_3(j) = \sigma_u(j)\alpha(j)$

### 1.9.1 พลังงานของโมเลกุลไฮโดรเจน

เมื่อเราสามารถสร้างฟังก์ชันคลื่นที่ใช้ในการอธิบายโมเลกุลไฮโดรเจนได้แล้ว ลำดับต่อไปก็คือการกำหนดและสร้าง Electronic Hamiltonian สำหรับการคำนวณพลังงานของโมเลกุลไฮโดรเจนซึ่งเราเขียน Hamiltonian ให้อยู่ในรูปของผลรวมของพลังงานที่เกิดจากอันตริกิริยาระหว่างอนุภาคได้ดังนี้

$$\hat{H}^{\text{el}} = -\frac{1}{2}\nabla_1^2 - \frac{1}{2}\nabla_2^2 - \frac{Z_A}{r_{1A}} - \frac{Z_A}{r_{2A}} - \frac{Z_B}{r_{1B}} - \frac{Z_B}{r_{2B}} + \frac{1}{r_{12}} + \frac{Z_A Z_B}{R_{AB}} \tag{1.9.9}$$

ถ้าเราทำการจัดรูปนิดหน่อยโดยการจัดให้เทอมที่อ้างอิงอิเล็กตรอนตัวเดียวกันนั้นมาอยู่ด้วยกัน ดังนี้

$$\hat{H}^{\text{el}} = -\frac{1}{2}\nabla_1^2 - \frac{Z_A}{r_{1A}} - \frac{Z_B}{r_{1B}} - \frac{1}{2}\nabla_2^2 - \frac{Z_A}{r_{2A}} - \frac{Z_B}{r_{2B}} + \frac{1}{r_{12}} + \frac{Z_A Z_B}{R_{AB}} \tag{1.9.10}$$

เราสามารถเขียนใหม่ได้เป็น

$$\hat{H}^{\text{el}} = \hat{H}_1 + \hat{H}_2 + \frac{1}{r_{12}} + \frac{Z_A Z_B}{R_{AB}} \tag{1.9.11}$$

โดยที่เทอมที่ขึ้นอยู่กับอิเล็กตรอนเพียง 1 ตัวนั้นเราจะเรียกว่า One-Electron Term ($\hat{H}_i$) สำหรับอิเล็กตรอนตัวที่ $i$ นั้นเขียนได้ดังนี้

$$\hat{H}_i = -\frac{1}{2}\nabla_i^2 - \frac{Z_A}{r_{iA}} - \frac{Z_B}{r_{iB}} \tag{1.9.12}$$

จากสมการ (1.9.9) นั้นเราจะได้ว่าพลังงานของโมเลกุลนั้นก็จะมีหน้าตาที่คล้ายกันซึ่งก็เทียบเคียงกับเทอมแต่ละเทอมของ Hamiltonian นั่นเอง ดังนี้

$$E_0 = E_0(1) + E_0(2) + E_0(1,2) + \frac{Z_A Z_B}{R_{AB}} \tag{1.9.13}$$

โดยที่เทอมสุดท้ายของทางด้านขวามือนั้นคือพลังงานคูลอมบ์ระหว่างนิวเคลียส ส่วนเทอมที่เป็น Contribution ของ One-Electron สำหรับอิเล็กตรอนตัวที่ 1 นั้นเราจะใช้ Expectation Value ดังนี้

$$\begin{aligned} E_0(1) &= \left\langle \psi_0 \left| \hat{H}_1 \right| \psi_0 \right\rangle \\ &= \frac{1}{2}\left( \left\langle \chi_1(1) \left| \hat{H}_1 \right| \chi_1(1) \right\rangle + \left\langle \chi_2(1) \left| \hat{H}_1 \right| \chi_2(1) \right\rangle \right) \\ &= \left\langle \sigma_g(1) \left| \hat{H}_1 \right| \sigma_g(1) \right\rangle \end{aligned} \tag{1.9.14}$$

และก็เหมือนกันกับพลังงานของอิเล็กตรอนตัวที่ 2 $(E_0(2))$

$$E_0(2) = \left\langle \sigma_g(2) \left| \hat{H}_2 \right| \sigma_g(2) \right\rangle \tag{1.9.15}$$

ส่วนเทอมที่ Contribution นั้นมาจากอิเล็กตรอน 2 ตัวนั้น (Two-Electron Term), $E_0(1,2)$, ก็จะกลายเป็น

$$E_0(1,2) = \left\langle \sigma_g(1)\sigma_g(2) \left| \frac{1}{r_{12}} \right| \sigma_g(1)\sigma_g(2) \right\rangle \tag{1.9.16}$$

ซึ่งเราสามารถเขียนใหม่ได้เป็น

$$\begin{aligned} E_0(1,2) &= \int \sigma_g^*(1)\sigma_g^*(2)\frac{1}{r_{12}}\sigma_g(1)\sigma_g(2)\mathrm{d}\tau \\ &= \int \rho(1)\frac{1}{r_{12}}\rho(2)\mathrm{d}\tau \end{aligned} \tag{1.9.17}$$

โดยที่ $\rho(j) = \sigma^*(j)\sigma(j)$ นั้นคือการตีความตาม Born's Interpretation ที่ว่าความหนาแน่นอิเล็กตรอนนั้นสัมพันธ์กับ MO นอกจากนี้แล้วเรายังสามารถตีความ $E(1,2)$ ว่าเป็นอันตริกิริยาแบบคูลอมบ์ระหว่างอิเล็กตรอนซึ่งมักจะเขียนแทนด้วย $J_{ij}$ ดังนั้นเราจึงเขียนสมการนี้ใหม่ได้เป็น

$$E_0 = H_1 + H_2 + J_{12} + \frac{Z_A Z_B}{R_{AB}} \tag{1.9.18}$$

โดยที่เราใช้ Notation $H_j \equiv E_0(j)$ สำหรับ One-Electron Term ซึ่งพลังงานของเทอมนี้จะเหมือนกับพลังงานของโมเลกุลที่สถานะพื้น ดังนี้

$$E_1(j) = \frac{1}{2}\left(\left\langle \sigma_g(j) \left| \hat{H}_j \right| \sigma_g(j) \right\rangle + \left\langle \sigma_u(j) \left| \hat{H}_j \right| \sigma_u(j) \right\rangle\right) \tag{1.9.19}$$

แต่ว่าพลังงานของ Two-Electron Term นั้นจะมีความซับซ้อนกว่ามาก ดังนี้

$$\begin{aligned} E_1(1,2) = & \underbrace{\frac{1}{2}\Bigg( \left\langle \sigma_g(1)\sigma_u(2) \left| \frac{1}{r_{12}} \right| \sigma_g(1)\sigma_u(2) \right\rangle \\ & \quad + \left\langle \sigma_u(1)\sigma_g(2) \left| \frac{1}{r_{12}} \right| \sigma_u(1)\sigma_g(2) \right\rangle \Bigg)}_{J_{12}} \\ & - \underbrace{\left\langle \sigma_g(1)\sigma_u(2) \left| \frac{1}{r_{12}} \right| \sigma_g(2)\sigma_u(1) \right\rangle}_{K_{12}} \end{aligned} \tag{1.9.20}$$

เรามาทวนสมการที่ (1.9.20) กันอีกรอบนะครับ เทอมแรกทางด้านขวาของสมการนั้นคือพลังงานคูลอมบ์ซึ่งจะเขียนแทนด้วย $J_{ij}$ ส่วนเทอมที่สองนั้นคือพลังงานแลกเปลี่ยน (Exchange Integral) เขียนแทนด้วย $K_{ij}$ โดยพลังงานของสำหรับ Triplet State ของโมเลกุลไฮโดรเจนนั้นจะกลายเป็น

$$E_1 = H_1 + H_2 + J_{12} - K_{12} + \frac{Z_A Z_B}{R_{AB}} \tag{1.9.21}$$

เนื่องจากว่า Exchange Integral $(K_{ij})$ นั้นมีเครื่องหมายเป็นบวก หมายความว่าพลังงานสำหรับ Triplet State นั้นจะมีค่าต่ำกว่าพลังงานของโมเลกุลในสถานะกระตุ้นแบบ Singlet State ซึ่งตีความได้ว่าอิเล็กตรอนของโมเลกุลนั้นจะมีถูก Delocalized มากกว่าเมื่อโมเลกุลอยู่ในสถานะ Triplet State

### 1.9.2 พลังงานของ Slater Determinant

ในหัวข้อนี้เราจะมาดูรายละเอียดของพลังงานของ Slater Determinant กัน เราจะเริ่มต้นด้วยการเขียน Slater Determinant (จากสมการที่ (1.8.14)) ใหม่โดยใช้ Dirac Notation ดังนี้

$$|\psi\rangle = \hat{A} \,|\, \chi_1(1)\chi_2(2)\ldots\chi_n(n)\rangle \tag{1.9.22}$$

โดยที่ $\hat{A}$ คือโอเปอร์เรเตอร์ที่ทำให้เป็นปฏิสมมาตร (Anti-symmetrizing Operator) ซึ่งเป็นองค์ประกอบ

สำคัญที่ทำให้ Slater Determinat นั้นถูกต้องโดยการกระทำบนผลคูณของ Spin-Orbitals สมการทางคณิตศาสตร์ของ $\hat{A}$ มีดังนี้

$$\begin{aligned}\hat{A} &= \frac{1}{\sqrt{n!}}\sum_{p=0}^{n-1}(-1)^p\hat{P}^{(p)} \\ &= \frac{1}{\sqrt{n!}}\left(\hat{1} - \sum_{i=1}^{n}\sum_{j=i+1}^{n}\hat{P}_{ij}^{(1)} + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\sum_{k=j+1}^{n}\hat{P}_{ijk}^{(2)} - \dots\right)\end{aligned} \tag{1.9.23}$$

โดยที่ $\hat{1}$ คือ Identity Operator, $\hat{P}_{ij}^{(1)}$ คือ Permutation Operator สำหรับเรียงสับเปลี่ยนพิกัดของอิเล็กตรอนสองตัว $i$ กับ $j$ ซึ่งมีสมการดังต่อไปนี้

$$\hat{P}_{ij}^{(1)}\left|\chi_i(i)\chi_j(j)\right\rangle = \left|\chi_j(i)\chi_i(j)\right\rangle \tag{1.9.24}$$

ในทำนองเดียวกันกับระบบโมเลกุลที่มีอิเล็กตรอนมากกว่า 2 ตัว เช่น ถ้าระบบมีอิเล็กตรอน 3 ตัว เราจะได้ว่า Permutation Operator $\hat{P}_{ijk}^{(2)}$ นั้นจะให้การเรียงสลับพิกัดของอิเล็กตรอนทั้ง 3 ตัวดังสมการต่อไปนี้

$$\hat{P}_{ijk}^{(2)}\left|\chi_i(i)\chi_j(j)\chi_k(k)\right\rangle = \left|\chi_k(i)\chi_i(j)\chi_j(k)\right\rangle + \left|\chi_j(i)\chi_k(j)\chi_i(k)\right\rangle \tag{1.9.25}$$

โดยธรรมเนียมแล้ว (หนังสือเคมีควอนตัมเกือบทั้งหมด) นั้นมักจะทำการเรียงลำดับออร์บิทัลในการเขียนผลคูณของออร์บิทัลในสมการของ Permutation Operator (สมการที่ (1.9.24) และ (1.9.25)) โดยการใช้เลเบลของพิกัดของอิเล็กตรอน

นอกจากนี้แล้ว $\hat{A}$ นั้น Commute กับ $\hat{H}$ ได้ด้วย ดังนี้

$$[\hat{A},\hat{H}] = \hat{A}\hat{H} - \hat{H}\hat{A} = 0 \tag{1.9.26}$$

แล้วก็

$$\hat{A}\hat{A} = \sqrt{n!}\hat{A} \tag{1.9.27}$$

ผู้อ่านอาจจะลองไปพิสูจน์สมการที่ (1.9.26) และ (1.9.27) ก็ได้

คราวนี้เราจะมาลองเขียนสมการ Electronic Hamiltonian (สมการที่ (1.7.2)) ใหม่โดยการใช้สมการที่ (1.9.11) เข้ามาช่วย ดังนี้

$$\begin{aligned}\hat{H}^{\text{el}} &= \hat{T}_{\text{e}} + \hat{V}_{\text{en}} + \hat{V}_{\text{ee}} + \hat{V}_{\text{nn}} \\ &= \sum_{i=1}^{n} \hat{h}(i) + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \hat{g}(i,j) + \hat{V}_{nn}\end{aligned} \tag{1.9.28}$$

โดยที่เทอม One-Electron Term $(\hat{h}(i))$ นั้นคือ Motion ของอิเล็กตรอน $i$ ในสนามศักย์ของนิวเคลียสทุก ๆ ตัวแล้วก็รวม $\hat{T}_e$ และ $\hat{V}_{ne}$ เข้าไปด้วย ส่วนเทอม $\hat{g}(i,j)$ นั้นก็คือ Two-Electron ที่รวมพลังงานแรงผลักคูลอมบ์ระหว่างอิเล็กตรอน (Electron-Electron Coulomb Repulsion หรือ $\hat{V}_{ee}$)

โดยสรุปแล้วพลังงานของ Slater Determinant ตามสมการที่ (1.9.22) มีหน้าตาดังนี้

$$\begin{aligned}E_0 &= \left\langle \psi \middle| \hat{H}^{\text{el}} \middle| \psi \right\rangle \\ &= \left\langle \hat{A}\chi_1(1)\chi_2(2)\dots\chi_n(n) \middle| \hat{H}^{\text{el}} \middle| \hat{A}\chi_1(1)\chi_2(2)\dots\chi_n(n) \right\rangle \\ &= \sqrt{n!} \left\langle \chi_1(1)\chi_2(2)\dots\chi_n(n) \middle| \hat{H}^{\text{el}} \middle| \hat{A}\chi_1(1)\chi_2(2)\dots\chi_n(n) \right\rangle \\ &= \sum_{p=0}^{n-1} (-1)^p \left\langle \chi_1(1)\chi_2(2)\dots\chi_n(n) \middle| \hat{H}^{\text{el}} \middle| \hat{A}^{(p)}\chi_1(1)\chi_2(2)\dots\chi_n(n) \right\rangle\end{aligned} \tag{1.9.29}$$

ส่วนเทอมสุดท้ายที่ผมยังไม่ได้ลงรายละเอียดก็คือโอเปอร์เตอร์ของพลังงานคูลอมบ์ระหว่างนิวเคลียส (Nucleus-nucleus Coulomb Operator หรือ $\hat{V}_{nn}$) ซึ่งจะขึ้นอยู่กับพิกัดของนิวเคลียส ดังนี้

$$\langle\psi|\hat{V}_{nn}|\psi\rangle = V_{nn}\langle\psi \mid \psi\rangle = V_{nn} \tag{1.9.30}$$

โดยที่ $\psi$ นั้นถูก Normalized แล้วและ $V_{nn}$ นั้นก็ถูกลดรูปให้เหลือเป็นแค่พลังงานคูลอมบ์แบบกลศาสตร์ดั้งเดิม (Classical Coulomb Interaction Energy) ตามที่แสดงในสมการ (1.9.13) สำหรับโมเลกุลไฮโดรเจน เนื่องจากว่าเราสร้าง Orthonormal Set มาจาก Spin-Orbitals ดังนั้นจะมีแค่ Identity Operator $(\hat{1})$ ใน Antisymmetrizing Operator $(\hat{A})$ ตามสมการที่ (1.9.23) เท่านั้นที่จะส่งผลหรือมี Contribution ต่อพลังงานของ One-Electron Operator $(\hat{h}(i))$ เช่น

$$\begin{aligned}&\langle\ \chi_1(1)\chi_2(2)\dots\chi_n(n)|\hat{h}(1)|\chi_1(1)\chi_2(2)\dots\chi_n(n)\ \rangle \\ &\qquad = \langle\chi_1(1)|\hat{h}(1)|\chi_1(1)\rangle\ \langle\chi_2(2)\mid\chi_2(2)\rangle\dots\langle\chi_n(n)\mid\chi_n(n)\rangle \\ &\qquad = \langle\chi_1(1)|\hat{h}(1)|\chi_1(1)\rangle \\ &\qquad = h_1\end{aligned} \tag{1.9.31}$$

สำหรับ One-Electron Operator นั้นพลังงานทุกเทอมที่รวมผลของ Permutation จะมีค่าเท่ากับ 0 ดังนี้

$$\begin{aligned}
&\langle\, \chi_1(1)\chi_2(2)\dots\chi_n(n)|\hat{h}(1)|\hat{P}_{12}^{(1)}\chi_1(1)\chi_2(2)\dots\chi_n(n)\,\rangle \\
&\qquad = \langle\chi_1(1)|\hat{h}(1)|\chi_2(1)\rangle\,\langle\chi_2(2)\mid\chi_1(2)\rangle\dots\langle\chi_n(n)\mid\chi_n(n)\rangle \\
&\qquad = 0
\end{aligned} \tag{1.9.32}$$

โดยที่เทอมที่ 2 ของทางด้านขวาของสมการที่มีเทอมการอินทริเกรตผ่านพิกัดของอิเล็กตรอนตัวที่ 2 นั้นมีค่าเท่ากับ 0 ก็เพราะว่าสมบัติ Orthogonality ของ Spin-Orbitals อันที่ 1 กับ 2 ซึ่งด้วยเหตุผลเดียวกันนี้จึงทำให้มีแค่ Identity Operator $(\hat{1})$ และโอเปอร์เรเตอร์ Two-Electron Permutation $(\hat{P}_{ij}^{(1)})$ ตามสมการที่ (1.9.23) เท่านั้นที่มี Contribution ต่อ Two-Electron Operator $(g(i,j))$ ซึ่งท้ายที่สุดแล้วเทอมสำหรับ Identity Operator ของอิเล็กตรอนตัวที่ 1 กับ 2 จะกลายเป็น

$$\begin{aligned}
&\langle\chi_1(1)\chi_2(2)\chi_3(3)\dots\chi_n(n)|\hat{g}(1,2)|\chi_1(1)\chi_2(2)\chi_3(3)\dots\chi_n(n)\,\rangle \\
&\quad = \langle\chi_1(1)\chi_2(2)|\hat{g}(1,2)|\chi_1(1)\chi_2(2)\rangle\,\langle\chi_3(3)\mid\chi_3(3)\rangle\dots\langle\chi_n(n)\mid\chi_n(n)\rangle \\
&\quad = \langle\chi_1(1)\chi_2(2)|\hat{g}(1,2)|\chi_1(1)\chi_2(2)\rangle \\
&\quad = J_{12}
\end{aligned} \tag{1.9.33}$$

ซึ่งเทอมนี้จริง ๆ แล้วก็คืออินทิกรัลคูลอมบ์ (Coulomb Integral) ซึ่งจะสอดคล้องกับพลังงานของ One-Electron ที่สถานะพื้น (สมการที่ (1.9.17)) สำหรับโมเลกุลไฮโดรเจน ส่วนเทอมที่ 2 สำหรับ $\hat{P}_{12}^{(1)}$ นั้นก็จะกลายเป็น

$$\begin{aligned}
&\langle\chi_1(1)\chi_2(2)\chi_3(3)\dots\chi_n(n)|\hat{g}(1,2)|\hat{P}_{12}^{(1)}\chi_1(1)\chi_2(2)\chi_3(3)\dots\chi_n(n)\,\rangle \\
&\quad = \langle\chi_1(1)\chi_2(2)|\hat{g}(1,2)|\chi_2(1)\chi_1(2)\rangle\,\langle\chi_3(3)|\chi_3(3)\rangle\dots\langle\chi_n(n)|\chi_n(n)\rangle \\
&\quad = \langle\chi_1(1)\chi_2(2)|\hat{g}(1,2)|\chi_2(1)\chi_1(2)\rangle \\
&\quad = K_{12}
\end{aligned} \tag{1.9.34}$$

โดยที่ $K_{12}$ คือ Exchange Integral ซึ่งก็จะสอดคล้องกับสมการพลังงานของ Two-Electron ที่สถานะกระตุ้นแบบ Triplet State (สมการที่ (1.9.20)) โดยการรวม Slater Determinant กับ Orthonormal Orbitals เข้าด้วยกันเพื่อเป็นการจัดรูปหรือลดรูปสมการพลังงานของโมเลกุลให้อยู่ในรูปของผลรวมของ One-Electron และ Two-Electron Integrals นั้นมีชื่อเรียกว่าหลักการ Slater-Condon โดยในสมการที่ (1.9.33) และ (1.9.34) เราได้ทำการใส่อิเล็กตรอนตัวที่ 1 กับ 2 เข้าไปในออร์บิทัลที่ 1 และ 2 ตามลำดับ อย่างไรก็ตามเมื่อเราทำการอินทิเกรตโดยใช้พิกัดของอิเล็กตรอนนั้นอิเล็กตรอนจะมี Label เป็นอะไรก็ได้เพราะว่าอิเล็กตรอนทุกตัวนั้นเหมือนกันหมด ดังนั้นเราจึงไม่ต้องใส่ Label ให้กับอิเล็กตรอนและเขียน Coulomb Integral กับ Exchange Integral ใหม่ได้ดังนี้

$$J_{12} = \langle \chi_1\chi_2|\hat{g}|\chi_1\chi_2\rangle \quad \text{และ} \quad K_{12} = \langle \chi_1\chi_2|\hat{g}|\chi_2\chi_1\rangle \tag{1.9.35}$$

ซึ่งตัวสมการจะมีความเรียบง่ายมากขึ้น เมื่อเราทำการเปลี่ยนลำดับของออร์บิทัลใหม่เราจะสามารถเขียนสมการพลังงานของ Slater Determinant จากเดิมที่เรามีในสมการ (1.9.29) ได้ใหม่เป็นดังนี้

$$E_0 = \sum_{i=1}^{n} h_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} (J_{ij} - K_{ij}) + V_{nn} \tag{1.9.36}$$

โดยที่เครื่องหมายลบสำหรับ Exchange Integral นั้นมาจากแฟคเตอร์ $(-1)^p$ ในสมการที่ (1.9.29) นอกจากนี้เรายังพบว่าเทอมที่เป็น Self-interaction ระหว่างอิเล็กตรอนกับตัวมันเองนั้น $(J_{ii})$ จะหักล้างกับเทอม $K_{ii}$ พอดี (ดูตามสมการที่ (1.9.33) และ (1.9.34)) ดังนั้นเราจึงสามารถเขียนสมการที่ (1.9.36) ใหม่ได้เป็น

$$E_0 = \sum_{i=1}^{n} h_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} (J_{ij} - K_{ij}) + V_{nn} \tag{1.9.37}$$

ซึ่งสิ่งที่ต่างไปจากเดิมก็คือจำนวนเทอมของ Coulomb Integral และ Exchange Integral ซึ่งจะเหลืออยู่เพียงแค่ครึ่งหนึ่งเท่านั้นและดัชนีของเครื่องหมาย Summation อันที่ 2 จะเปลี่ยนจาก $j = i + 1$ เป็น $j = 1$ อีกด้วย โดยสำหรับกรณีเฉพาะที่ระบบนั้นเป็นแบบ Closed-shell ก็คือเราใส่อิเล็กตรอน 2 ตัวที่มีสปินตรงข้ามกันเข้าไปใน Spin Orbital (อิเล็กตรอนทุกตัวนั้นมีคู่หมด) เราจะได้ว่า

$$\chi_1(1) = \varphi_1(1)\alpha(1) \quad \text{และ} \quad \chi_2(2) = \varphi_1(2)\beta(2) \tag{1.9.38}$$

ถ้าเราทำการคำนวณ Exchange Integral สมการที่ (1.9.34) สำหรับอิเล็กตรอนแต่ละคู่ในแต่ละออร์บิทัลนั้นเราจะได้ว่า

$$\begin{aligned} K_{12} &= \langle \chi_1(1)\chi_2(2)|\hat{g}(1,2)|\chi_1(2)\chi_2(1)\rangle \\ &= \langle \varphi_1(1)\varphi_2(2)|\hat{g}(1,2)|\varphi_1(2)\varphi_2(1)\rangle \, \langle \alpha(1) \mid \beta(1)\rangle\langle \alpha(2) \mid \beta(2)\rangle \\ &= 0 \end{aligned} \tag{1.9.39}$$

เหตุผลที่ Integral นี้เท่ากับ 0 ก็เพราะว่าสมบัติ Orthogonality ของ Spin Functions (ดูสมการที่ (1.8.12)) นั่นเอง ดังนั้นจำนวนเทอมครึ่งหนึ่งของ Exchange Integral ในสมการที่ (1.9.37) ซึ่งจะทำให้เราได้สมการพลังงานของ Slater Determinant ของโมเลกุลที่มีความซับซ้อนน้อยกว่าสมการพลังงานที่เราได้กำหนดไว้ในตอนต้น ดังนี้

$$E_0 = 2\sum_{i=1}^{n/2} h_i + \sum_{i=1}^{n/2}\sum_{j=1}^{n/2} (2J_{ij} - K_{ij}) \tag{1.9.40}$$

โดยที่เราจะกระจายเทอม Summation ตามจำนวนของออร์บิทัลที่มีอิเล็กตรอนบรรจุอยู่ 2 ตัว (Doubly Occupied Orbitals)

## 1.10 หลักการผันแปร

หลักการผันแปร (Variation Theory) เป็นวิธีที่สำคัญมาก ๆ ในวิชาเคมีควอนตัมและถูกนำมาประยุกต์ใช้กับทฤษฎีอื่น ๆ ในวิชา Electronic Structure เยอะมาก ๆ ซึ่งเป็นหลักการที่ช่วยให้เราสามารถประมาณ (Approximate) คำตอบหรือผลเฉลยของสมการชโรดิงเงอร์ได้ เริ่มต้นเลยเรากำหนดให้ $\psi_i$ เป็นฟังก์ชันคลื่นที่แท้จริงของโมเลกุลและ $\tilde{\psi}_i$ เป็นฟังก์ชันคลื่นที่ถูกประมาณขึ้นมา (อาจจะเรียกว่าเป็นฟังก์ชันคลื่นปลอม ๆ ที่เราสร้างขึ้นมาก็ได้หรือภาษาอังกฤษก็คือ Approximate Trial Function) ส่วนพลังงานของของฟังก์ชันคลื่นจริงกับฟังก์ชันคลื่นของปลอมที่เป็นการประมาณนี้จะแทนด้วย $E_i$ และ $\tilde{E}_i$ ตามลำดับ โดยพลังงานของ Trial Wavefunction นี้สามารถเขียนให้อยู่ในรูปของ Expectation Value ได้ดังนี้

$$\tilde{E}_i = \frac{\langle \tilde{\psi}_i | \hat{H} | \tilde{\psi}_i \rangle}{\langle \tilde{\psi}_i | \tilde{\psi}_i \rangle} \tag{1.10.1}$$

ซึ่งมีชื่อเรียกว่า Rayleigh Ratio

Variation Theorem นั้นกล่าวไว้ว่า “พลังงานของฟังก์ชันคลื่นที่ได้มาจากการประมาณนั้นจะไม่มีวันที่จะน้อยกว่าพลังงานของฟังก์ชันที่แท้จริงได้” นั่นหมายความว่าถ้าเราสามารถฟังก์ชันคลื่นประมาณที่ดีที่สุดเลยเท่าที่จะทำได้ พลังงานก้อนนี้ก็จะต้องมีค่าได้น้อยที่สุดคือเท่ากับพลังงานจริงของโมเลกุล โดยเราสามารถเขียนเป็นฟังก์ชันคณิตศาสตร์ได้ดังนี้

$$\tilde{E}_0 \geq E_0 \qquad \text{สำหรับทุก } \tilde{\psi}_i \tag{1.10.2}$$

ผลที่ตามมาก็คือว่าพลังงานที่คำนวณออกมาได้นั้นจะเป็นเสมือนมาตรวัดที่คอยบอกเราว่า Trial Wavefunction นั้นดีแค่ไหน นั่นหมายความว่าเราจะต้องค้นหา Trial Wavefunction ที่ให้พลังงานที่น้อยที่สุดนั่นเอง โดยกรณีศึกษาที่สำคัญมาก ๆ อันหนึ่งของ Variational Principle ก็คือการเขียน Trial Wavefunction ให้อยู่ในรูปการกระจายของเซตฟังก์ชัน $\phi_p$ ทั้งหมด $m$ ฟังก์ชัน ดังนี้

$$\tilde{\psi}_0 = \sum_{p=1}^{m} c_p \phi_p \tag{1.10.3}$$

โดยที่ $c_p$ นั้นคือสัมประสิทธิ์ที่เราจะต้องคำนวณหาออกมา นอกจากนี้แล้ว Rayleigh Ratio ตามสมการที่ (1.10.1) ก็จะกลายเป็น

$$\tilde{E}_0 = \frac{\sum_{p,q=1}^{m} c_p c_q H_{pq}}{\sum_{p,q=1}^{m} c_p c_q S_{pq}} \tag{1.10.4}$$

โดยที่เราใช้ Notation ตามนี้ $H_{pq} = \left\langle \phi_p | \hat{H} | \phi_q \right\rangle$ และ $S_{pq} = \langle \phi_p \mid \phi_q \rangle$ ตามลำดับ เมื่อเรานำ Variation Theorem ตามสมการที่ (1.10.2) มาประยุกต์ใช้กับ Trial Wavefunction อันนี้แล้วเราจะได้เงื่อนไขดังต่อไปนี้

$$\frac{\partial \tilde{E}_0}{\partial c_r} = 0 \quad \forall r \tag{1.10.5}$$

โดยผลลัพธ์ที่เราได้ออกมานั้นก็คือ Secular Equation ดังนี้

$$\sum_{p=1}^{m} c_p \left( H_{pr} - \tilde{E}_0 S_{pr} \right) = 0 \quad \forall r \tag{1.10.6}$$

ซึ่งสมการข้างต้นนี้ก็จะเป็นจริงและสามารถแก้หาผลเฉลยได้ถ้า Secular Determinant เท่ากับ 0 ดังนี้

$$|\boldsymbol{H} - \tilde{E}_0 \boldsymbol{S}| = 0 \tag{1.10.7}$$

โดยที่ $H_{pr}$ และ $S_{pr}$ คือ Matrix Element ของ $\boldsymbol{H}$ และ $\boldsymbol{S}$ ตามลำดับ วิธีการนี้มีชื่อเรียกว่า Rayleigh-Ritz Method

## 1.11 การประมาณของฮาร์ทรี-ฟ็อค

จุดเริ่มต้นของการประมาณของฮาร์ทรี-ฟ็อค (Hartree-Fock Approximation) ก็คือพลังงานสำหรับ Slater Determinant ที่เราเพิ่งศึกษาไปในหัวข้อก่อนหน้านี้ ซึ่งถ้าหากเขียนสมการของพลังงานดังกล่าวโดยใช้ Notation แบบกระชับ ๆ จะได้ดังนี้

$$\begin{aligned} E_0 &= \sum_{i=1}^{n} h_i + \frac{1}{2}\sum_{i,j=1}^{n}\left(J_{ij} - K_{ij}\right) + V_{nn} \\ &= \sum_{i=1}^{n} \left\langle \chi_i | \hat{h} | \chi_i \right\rangle \\ &\quad + \frac{1}{2}\sum_{i,j=1}^{n}\left(\langle \chi_i\chi_j | \hat{g} | \chi_i\chi_j\rangle - \langle \chi_i\chi_j | \hat{g} | \chi_j\chi_i\rangle\right) \\ &\quad + V_{nn} \end{aligned} \tag{1.11.1}$$

Slater Determinant ก็คือฟังก์ชันคลื่นที่ถูกประมาณขึ้นมาซึ่งเราสามารถหาพลังงานที่ต่ำที่สุดได้โดยการใช้ Variational Principle โดยการปรับออร์บิทัล (Orbital Optimization) อย่างไรก็ตามเราจะต้องไม่ลืมว่าออร์บิทัลนั้นเป็นตัวกำหนด Orthonormal Set และเงื่อนไขนี้ก็เป็นจริงตามหลักการ Minimization โดยใช้การ Lagrangian Multipliers ซึ่ง Lagrangian, $\tilde{E}_0$, นั้นมีหน้าตาดังนี้

$$\begin{aligned} \tilde{E}_0 &= E_0 - \sum_{i,j=1}^{n} \lambda_{ij}\left(\langle \chi_i \mid \chi_j\rangle - \delta_{ij}\right) \\ &= E_0 - \sum_{i,j=1}^{n} \lambda_{ij}\left(S_{ij} - \delta_{ij}\right) \end{aligned} \tag{1.11.2}$$

โดยที่เรามี Lagrangian Multiplier $\lambda_{ij}$ แต่ละอันสำหรับแต่ละคู่ออร์บิทัลและ $\delta_{ij}$ ก็คือ Kroenecker Delta Function ซึ่งบ่งบอกถึงสมบัติของการเป็น Orthonormality (Orthogonal + Normal) ของออร์บิทัล นอกจากนี้แล้วเราจะสังเกตเห็นในสมการข้างต้นด้วยว่ามีเทอมที่เป็นการ Overlap กันระหว่างออร์บิทัล ซึ่งเราเรียกเทอมนี้ว่า Overlap Matrix $S_{ij}$ โดยมีนิยามดังนี้

$$S_{ij} = \langle \chi_i \mid \chi_j\rangle \tag{1.11.3}$$

เนื่องจากว่าการผันแปรของ Lagrangian นั้นมีค่าน้อยมาก $(\delta\tilde{E}_0)$ เราจะได้ว่าการเปลี่ยนแปลงน้อย ๆ ของ Lagrangian นี้มีสมการคือ

$$\delta\tilde{E}_0 = \delta E_0 - \sum_{i,j=1}^{n} \lambda_{ij}\left(\langle \delta\chi_i \mid \chi_j\rangle + \langle \chi_i \mid \delta\chi_j\rangle\right) \tag{1.11.4}$$

โดยที่การเปลี่ยนแปลงเพียงน้อย ๆ ของพลังงาน $E_0$ ในสมการ (1.11.1) นั้นจะกลายเป็น

$$\begin{aligned}\delta E_0 = &\sum_{i=1}^{n} \left\langle \delta\chi_i|\hat{h}|\chi_i \right\rangle \\ &+ \left\langle \chi_i|\hat{h}|\delta\chi_i \right\rangle \\ &+ \frac{1}{2}\sum_{i,j=1}^{n} \big( \langle \delta\chi_i\chi_j|\hat{g}|\chi_i\chi_j\rangle + \langle \chi_i\delta\chi_j|\hat{g}|\chi_i\chi_j\rangle \\ &\qquad + \langle \chi_i\chi_j|\hat{g}|\delta\chi_i\chi_j\rangle + \langle \chi_i\chi_j|\hat{g}|\chi_i\delta\chi_j\rangle \big) \\ &\qquad - \big(\langle \delta\chi_i\chi_j|\hat{g}|\chi_j\chi_i\rangle + \langle \chi_i\delta\chi_j|\hat{g}|\chi_j\chi_j\rangle \\ &\qquad + \langle \chi_i\chi_j|\hat{g}|\delta\chi_j\chi_i\rangle + \langle \chi_i\chi_j|\hat{g}|\chi_j\delta\chi_i\rangle \big)\end{aligned} \tag{1.11.5}$$

ถ้าเราพิจารณาเทอมที่ 3 ของสมการที่ (1.11.5) ให้ดี ๆ จะพบว่าเทอม Integral ทั้ง 8 เทอมของผลรวมนั้นสามารถจัดรูปให้ง่ายกว่านี้ได้ โดยถ้าหากว่าเราใช้คุณสมบัติการสลับที่จะพบว่าเราสามารถรวม Integral เข้าด้วยกันได้นั่นก็เพราะว่า Index $i$ และ $j$ นั้นจริง ๆ แล้วเป็น Dummy Index ซึ่งสามารถสลับกันได้ ดังนั้นเราจึงสมการที่กระชับขึ้น ดังนี้

$$\begin{aligned}\delta E_0 = &\sum_{i=1}^{n} \left\langle \delta\chi_i|\hat{h}|\chi_i \right\rangle \\ &+ \left\langle \chi_i|\hat{h}|\delta\chi_i \right\rangle \\ &+ \sum_{i,j=1}^{n} \langle \delta\chi_i\chi_j|\hat{g}|\chi_i\chi_j\rangle + \langle \chi_i\chi_j|\hat{g}|\delta\chi_i\chi_j\rangle \\ &\qquad - \langle \delta\chi_i\chi_j|\hat{g}|\chi_j\chi_i\rangle - \langle \chi_i\chi_j|\hat{g}|\delta\chi_j\chi_i\rangle\end{aligned} \tag{1.11.6}$$

ในขั้นตอนสุดท้ายนี้เราจะทำการ Introduce โอเปอร์เรเตอร์เพิ่มเติมอีก 2 ตัวนั่นก็คือ Coulomb Operator

$$\hat{J}_j\,|\chi_i\rangle = \langle \chi_j|\hat{g}|\chi_j\rangle\,|\chi_i\rangle \tag{1.11.7}$$

และ Exchange operator

$$\hat{K}_j\,|\chi_i\rangle = \langle \chi_j|\hat{g}|\chi_i\rangle\,|\chi_j\rangle \tag{1.11.8}$$

โดยที่โอเปอร์เรเตอร์ตัวนี้สามารถแลกเปลี่ยน (*exchange*) ออร์บิทัลที่ต้องการที่จะกระทำได้ ดังนั้นเราจึงได้ว่าสมการ (1.11.6) นั้นจะกลายเป็น

$$\begin{aligned}\delta E_0 = &\sum_{i=1}^{n} \left\langle \delta\chi_i | \hat{h} | \chi_i \right\rangle \\ &+ \left\langle \chi_i | \hat{h} | \delta\chi_i \right\rangle \\ &+ \sum_{i,j=1}^{n} \left\langle \delta\chi_i \left| \hat{J}_j - \hat{K}_j \right| \chi_i \right\rangle + \left\langle \chi_i \left| \hat{J}_j - \hat{K}_j \right| \delta\chi_i \right\rangle\end{aligned} \tag{1.11.9}$$

ลำดับต่อไปคือเราจะทำการกำหนด Fock Operator $\hat{f}$ ดังนี้

$$\hat{f} = \hat{h} + \sum_{j=1}^{n} \left( \hat{J}_j - \hat{K}_j \right) \tag{1.11.10}$$

ซึ่งเราจะได้ว่า Hartree-Fock Hamiltonian $\hat{H}_{\mathrm{HF}}$ นั้นก็คือผลรวมของ Fock Operator ของแต่ละออร์บิทัล ดังนี้

$$\hat{H}_{\mathrm{HF}} = \sum_{i=1}^{n} \hat{f}_i \tag{1.11.11}$$

โดยที่ $i$ บ่งบอกว่าเรามีอิเล็กตรอนแต่ละตัวนั้นมี Fock Operator เป็นของตัวเอง เราจึงได้ว่า

$$\delta E_0 = \sum_{i=1}^{n} \left\langle \delta\chi_i \left| \hat{f}_i \right| \chi_i \right\rangle + \left\langle \chi_i \left| \hat{f}_i \right| \delta\chi_i \right\rangle \tag{1.11.12}$$

ซึ่งเราจะได้ว่าผลรวมของ Lagrangian นั้นกลายเป็น

$$\begin{aligned}\delta \tilde{E}_0 = &\sum_{i=1}^{n} \left\langle \delta\chi_i \left| \hat{f}_i \right| \chi_i \right\rangle + \left\langle \chi_i \left| \hat{f}_i \right| \delta\chi_i \right\rangle \\ &- \sum_{i,j=1}^{n} \lambda_{ij} \left( \langle \delta\chi_i | \chi_j \rangle + \langle \chi_i | \delta\chi_j \rangle \right)\end{aligned} \tag{1.11.13}$$

โดยเราจะสันนิษฐานว่าไม่ว่าจะเป็นการเปลี่ยนแปลงเพียงน้อย ๆ ของ $\langle \delta\chi_i |$ หรือ $| \delta\chi_i \rangle$ นั้นสอดคล้องกับหลักการผันแปร (Variational Principle) เช่น $\delta \tilde{E}_0 = 0$ เราจะได้ว่ามีความสัมพันธ์ 2 อันที่เป็นจริงและเกิดขึ้นได้พร้อมกันนั่นคือ

$$\sum_{i=1}^{n}\langle\delta\chi_i|\hat{f}_i|\chi_i\rangle - \sum_{i,j=1}^{n}\lambda_{ij}\langle\delta\chi_i|\chi_j\rangle = 0 \tag{1.11.14}$$

และ

$$\sum_{i=1}^{n}\langle\chi_i|\hat{f}_i|\delta\chi_i\rangle - \sum_{i,j=1}^{n}\lambda_{ij}\langle\chi_i|\delta\chi_j\rangle = 0 \tag{1.11.15}$$

ถ้าหากว่าเราใช้คุณสมบัติดังต่อไปนี้

$$\left\langle\delta\chi_i\left|\hat{f}_i\right|\chi_i\right\rangle = \left\langle\chi_i\left|\hat{f}_i\right|\delta\chi_i\right\rangle^* \tag{1.11.16}$$

และทำการลบสมการที่ (1.11.14) ออกจากคอนจูเกตเชิงซ้อน (Complex Conjugate) ของสมการที่ (1.11.15) เราจะได้ว่า

$$\sum_{i,j=1}^{n}\left(\lambda_{ij}-\lambda_{ji}^*\right)\langle\delta\chi_i|\chi_j\rangle = 0 \tag{1.11.17}$$

ซึ่งเงื่อนไขข้างบนนั้นจะเป็นจริงถ้า $\lambda_{ij}$ นั้นคือสมาชิกของ Hermitian Matrix.

ขั้นตอนต่อไปก็คือเราจะทำการปรับสมการที่ (1.11.14) ใหม่ให้อยู่ในรูปของเซตของปัญหาค่าไอเกน (Eigenvalue Problems) แทนที่จะอยู่ในรูปของค่าคาดหวัง (Expectation Value)

$$\hat{f}_i|\chi_i\rangle = \sum_{j=1}^{n}\lambda_{ij}|\chi_j\rangle \tag{1.11.18}$$

ตอนนี้เราสามารถใช้ Unitary Transformation เพื่อช่วยในการสร้างออร์บิทัลซึ่งจะได้ว่า $\lambda_{ij}$ นั้นกลายเป็น Diagonal Matrix ดังนี้ $(\epsilon_i = \lambda_{ij})$

$$\hat{f}_i|\chi_i\rangle = \epsilon_i|\chi_j\rangle \tag{1.11.19}$$

โดยออร์บิทัลที่เราสร้างหรือกำหนดขึ้นมานี้เป็นออร์บิทัลแบบเฉพาะซึ่งเราจะเรียกออร์บิทัลนี้ว่า Canonical Orbital และ $\epsilon_i$ ก็คือพลังงานของออร์บิทัล นอกจากนี้แล้วสมการที่ (1.11.19) นั้นจริง ๆ แล้วก็คือสมการ

Hartree-Fock แล้วก็เป็นเซตของสมการEigenvalue Equation หลาย ๆ อันผสมกันเนื่องจากว่า Fock Operator นั้นขึ้นอยู่กับออร์บิทัลทุก ๆ อันผ่าน Coulomb Operator กับ Exchange Operator ตามลำดับ

ในการแก้สมการ Hartree-Fock นั้นเราจะใช้เทคนิคที่เรียกว่า Self-Consistent Field (SCF) ซึ่งเป็นการแก้สมการแบบวนซ้ำเทียบกับตัวเอง โดย SCF นั้นถูกเอามาใช้กับออร์บิทัลเริ่มต้นที่เราจะต้องเดาขึ้นมาก่อนเพื่อนำไปใช้ในการสร้างหรือเดา Fock Operator ต่อไป ซึ่งต่อจากนั้นก็จะเป็นการแก้สมการที่ (1.11.19) เพื่อใช้ในการปรับ (Update) Fock Operator อันใหม่ กระบวนการทั้งหมดนี้จะถูกทำซ้ำไปเรื่อย ๆ จนกว่าจะลู่เข้าภายใต้เงื่อนไขที่เรากำหนด

คราวนี้เรามาดูรายละเอียดของ Hartree-Fock ก็คือว่าถ้าเราสามารถเขียน Slater Determinant ได้ดังนี้

$$\varphi_t = \sum_{I=1}^{N} \frac{\mathrm{d}Z_i}{\mathrm{d}R_{It}} - \sum_{i=1}^{n} \langle \chi_i | \frac{1}{r_{it}} | \chi_i \rangle \tag{1.11.20}$$

โดยที่เทอมที่สองของทางด้านขวาของสมการนั้นคืออินทิกรัลตัวเดียวกับที่ Coulomb Operator มี

ท้ายที่สุดแล้วพลังงานของออร์บิทัล $i$ $(\epsilon_i)$ สามารถคำนวณได้จาก Expectation Value ดังนี้

$$\begin{aligned} \epsilon_i &= \left\langle \chi_i \left| \hat{f}_i \right| \chi_i \right\rangle \\ &\approx h_i + \sum_{j=1}^{n} (J_{ij} - K_{ij}) \end{aligned} \tag{1.11.21}$$

แทนที่เราจะทำการอินทิเกรต Fock Operator $\hat{f}_i$ เราก็ทำการแทนค่าเทอมนี้กลับเข้าไปในสมการ Two-Electron Integrals ที่เรามีอยู่ก่อนหน้านี้ ซึ่งก็จะได้ว่าพลังงานสำหรับ Slater Determinant (สมการที่ (1.11.1)) นั้นสามารถหาได้จากการประมาณ Hartree-Fock ดังนี้

$$E_0 = \sum_{i=1}^{n} \epsilon_i - \frac{1}{2} \sum_{i,j=1}^{n} (J_{ij} - K_{ij}) + V_{nn} \tag{1.11.22}$$

และจากสมการข้างบนนี้ไม่ได้มีเพียงแค่เทอมพลังงานรวมที่มาจากออร์บิทัลเท่านั้น แต่ยังมีการรวมพลังงานระหว่างนิวเคลียส-นิวเคลียส $V_{nn}$ เข้าไปด้วย

## 1.12 เบซิสเซท

"เบซิสเซท (Basis Set) สำคัญยังไง ทำไมเราต้องกำหนด Basis Set ก่อนการรันการคำนวณเคมีควอนตัมทุกครั้ง?" ผมเริ่มต้นหัวข้อด้วยคำถามนี้ก็เพราะว่าผู้อ่านหลายคนน่าจะให้ความสนใจ ใครที่เรียนวิชาเคมีเชิงฟิสิกส์ขั้นสูงโดยเฉพาะหัวข้อโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) หรือกำลังทำงานวิจัยทางด้านนี้อยู่น่าจะต้องเคยมีประสบการณ์ในการคำนวณเคมีควอนตัมสำหรับการศึกษาคุณสมบัติของโมเลกุลโดยการใช้วิธีทางควอนตัมกันมาบ้างแล้ว ปกติแล้วเราจะต้องทำการกำหนด Basis Set ที่เราจะใช้สำหรับอะตอมแต่ละตัวซึ่งโดยทั่วไปเราก็มักจะเลือก Basis Set เพียงแค่ 1 อันสำหรับทั้งโมเลกุล เช่น 6-31G(d) หรือ cc-pVTZ แล้ว Basis Set สำคัญยังไงและส่งผลต่อความถูกต้องของผลที่ได้จากการคำนวณมากน้อยแค่ไหน เราจะมาหาคำตอบกันในหัวข้อนี้

ต้องเท้าความความรู้ที่เราเคยเรียนกันจากวิชากลศาสตร์ควอนตัมเชิงโมเลกุล (Molecular Quantum Mechanics) ก่อนว่านักวิทยาศาสตร์นั้นต้องการที่จะหาฟังก์ชันทางคณิตศาสตร์ที่ไม่ซับซ้อนเพื่อมาอธิบายออร์บิทัล (ออร์บิทัลในที่นี้คือออร์บิทัลเชิงสปิน ซึ่งเป็นออร์บิทัลที่รวมผลของสปินของอิเล็กตรอนเข้าไปด้วย) ซึ่งฟังก์ชันที่เราเลือกมานั้นจะต้องสามารถช่วยให้เราแก้สมการ Hartree-Fock ได้อย่างสะดวกด้วย ซึ่งสุดท้ายแล้วนักวิทยาศาสตร์นั้นก็ใช้ออร์บิทัลเชิงโมเลกุลหรือ Molecular Orbitals (MOs) ในการอธิบายโมเลกุล โดย MOs นี้สามารถถูกเขียนให้อยู่ในรูปของผลรวมเชิงเส้นของออร์บิทัลเชิงอะตอมหรือ Atomic Orbitals (AOs) ได้หรือที่เรียกว่าวิธี Linear Combination of Atomic Orbitals (LCAO) ซึ่งก็มาจากแนวคิดที่ว่าอะตอมหลาย ๆ อะตอมรวมกันได้เป็นโมเลกุล โดยออร์บิทัลเชิงสปิน $(\chi_i)$ นั้นสามารถถูกเขียนให้อยู่ในรูปการกระจายด้วยฟังก์ชันเบสิส (Basis Function, $\phi_p$) ทั้งหมด $m$ ฟังก์ชันได้ดังนี้

$$|\chi_i\rangle = \sum_{p=1}^{m} c_{ip} \, |\phi_p\rangle \tag{1.12.1}$$

ถ้าผู้อ่านไปอ่านหนังสือบางเล่มแล้วพบว่ามีการใช้ตัวแปรที่ต่างกันออกไป เช่น ตามสมการด้านล่างนี้

$$|\psi_i\rangle = \sum_{p=1}^{m} c_{ip} \, |\varphi_p\rangle \qquad \text{หรือ} \qquad |\psi_i\rangle = \sum_{j=1} c_{ij} \, |\varphi_j\rangle \tag{1.12.2}$$

ก็ไม่ต้องตกใจไปเพราะว่าสมการที่ (1.12.2) นั้นก็คือสมการเดียวกันกับสมการที่ (1.12.1) นั่นเอง

อย่างไรก็ตามในการเขียนสมการคณิตศาสตร์เพื่ออธิบายออร์บิทัลนั้นถึงแม้ว่าหนังสือหลาย ๆ เล่มจะใช้ตัวอักษรต่างกันแต่โดยทั่วไปแล้วมักจะเขียนด้วยตัวอักษรกรีก ตัวอย่างเช่น เราใช้ psi $(\psi)$ ในการแทน MOs และใช้ phi $(\varphi)$ ในการแทน Basis Function ซึ่งเราสามารถเขียน MOs ได้ด้วยวิธี LCAO ซึ่งเป็นผลรวมของผลคูณระหว่างสัมประสิทธิ์ $c$ กับ Basis Function สำหรับแต่ละ MOs ในโมเลกุล จริง ๆ แล้ว $c$ นั้นมีชื่อเต็ม ๆ ว่า "สัมประสิทธิ์การกระจายของออร์บิทัลเชิงโมเลกุล" หรือ Molecular Orbital Expression Coefficients หรือเราจะเรียกสั้น ๆ ว่า MO Coefficients ก็ได้ นอกจากนี้แล้วในทางทฤษฎีนั้น Basis Function จะถูก

กำหนดให้มีตำแหน่งอยู่ที่จุดศูนย์กลางของอะตอม (Atom-centered Basis Function) อย่างไรก็ตามเราไม่มีกฎตายตัวว่า Basis Function นั้นจะต้องอยู่จุดศูนย์กลางของอะตอมเสมอไปถ้าหากเราสามารถหาฟังก์ชันที่อธิบายรูปร่างของออร์บิทัลได้อย่างเหมาะสม

เมื่อเรานำสมการของการกระจาย Basis Function เข้าไปแทนในสมการ Hartree-Fock (สมการที่ (1.11.19)) เราจะได้ว่า

$$\hat{f}_i \sum_{p=1}^{m} c_{ip} \left|\phi_p\right\rangle = \epsilon_i \sum_{j=1}^{m} c_{ip} \left|\phi_p\right\rangle \qquad \forall i \tag{1.12.3}$$

ในหัวข้อต่อไปเราจะมาดูรายละเอียดของ Atom-centered Basis Function กันครับ

### 1.12.1 การกระจายของเบซิสเซท

เราเริ่มต้นหัวข้อนี้ด้วยสัญกรณ์เมทริกซ์ต่อไปนี้

$$\phi = (\phi_1, \phi_2, \ldots \phi_m) \tag{1.12.4}$$

$$\boldsymbol{c}_i = \begin{pmatrix} c_{1i} \\ c_{2i} \\ \vdots \\ c_{mi} \end{pmatrix} \quad \text{และ} \quad \boldsymbol{C} = \begin{pmatrix} c_{11} & c_{12} & \ldots & c_{1n} \\ c_{21} & c_{22} & \ldots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \ldots & c_{mn} \end{pmatrix} \tag{1.12.5}$$

ซึ่งจะทำให้เราสามารถเขียนความสัมพันธ์ดังต่อไปนี้ได้

$$\chi_i = \phi \cdot \mathrm{c}_i \qquad \text{และ} \qquad \chi = \phi \cdot \mathrm{C} \tag{1.12.6}$$

โดยที่ $\cdot$ หมายถึงการคูณแบบ Dot Product และทางด้านซ้ายกับด้านขวาคือการคูณกับเวกเตอร์และเมทริกซ์ตามลำดับ แล้วเราก็กำหนด Fock Matrix ดังต่อไปด้วยนี้

$$\boldsymbol{F} = \begin{pmatrix} \langle\phi_1|\hat{f}|\phi_1\rangle & \langle\phi_1|\hat{f}|\phi_2\rangle & \ldots & \langle\phi_1|\hat{f}|\phi_m\rangle \\ \langle\phi_2|\hat{f}|\phi_1\rangle & \langle\phi_2|\hat{f}|\phi_2\rangle & \ldots & \langle\phi_2|\hat{f}|\phi_m\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle\phi_m|\hat{f}|\phi_1\rangle & \langle\phi_m|\hat{f}|\phi_2\rangle & \ldots & \langle\phi_m|\hat{f}|\phi_m\rangle \end{pmatrix} \tag{1.12.7}$$

และกำหนด Overlap Matrix ดังต่อไปนี้

$$\boldsymbol{S} = \begin{pmatrix} \langle \phi_1 \mid \phi_1 \rangle & \langle \phi_1 \mid \phi_2 \rangle & \cdots & \langle \phi_1 \mid \phi_m \rangle \\ \langle \phi_2 \mid \phi_1 \rangle & \langle \phi_2 \mid \phi_2 \rangle & \cdots & \langle \phi_2 \mid \phi_m \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi_m \mid \phi_1 \rangle & \langle \phi_m \mid \phi_2 \rangle & \cdots & \langle \phi_m \mid \phi_m \rangle \end{pmatrix} \tag{1.12.8}$$

โดยที่ $\langle \ldots | \ldots | \ldots \rangle$ และ $\langle \ldots | \ldots \rangle$ คือสมาชิกของเมทริกซ์หรือ Matrix Element

เมื่อเรานำ Variation Principle และวิธี Rayleigh-Ritz ที่ได้ศึกษาไปแล้วเราจะได้ว่าสมการที่เรียกว่า Roothaan-Hall ดังต่อไปนี้

$$\boldsymbol{F}\boldsymbol{c}_i = \epsilon_i \boldsymbol{S}\boldsymbol{c}_i \quad \text{หรือ} \quad \boldsymbol{F}\boldsymbol{C} = \boldsymbol{S}\boldsymbol{C}\boldsymbol{\epsilon} \tag{1.12.9}$$

โดยที่ $\boldsymbol{\epsilon}$ คือเมทริกซ์แนวทแยง (Diagonal Matrix) ซึ่งมี $\epsilon_i$ เป็นพลังงานออร์บิทัล (Orbital Energies) และสมการที่ (1.12.9) นั้นก็คือสมการปัญหาไอเกน (Eigenvalue Problem) ซึ่งเราสามารถหา Eigenvalues และ Eigenvectors ของ Fock Matrix ออกมาได้ซึ่งก็คือพลังงานของออร์บิทัลนั่นเอง ส่วน Overlap Matrix นั้นจริง ๆ แล้วเราจะมองว่าเป็นตัวเทียบกับ Fock Matrix ก็ได้เพราะว่า Overlap Matrix นั้นจะถูกลดรูปเหลือเป็นเพียงแค่ Unity Matrix $\mathbf{1}$

ในทำนองเดียวกันกับการแก้สมการ Hartree-Fock นั้นเราสามารถใช้วิธีการวนซ้ำ Self-Consistent Field (SCF) ในการแก้สมการ Roothaan-Hall เนื่องจากว่า Fock Matrix นั้นขึ้นอยู่กับ Eigenvectors (สัมประสิทธิ์ของออร์บิทัล)

### 1.12.2 เมทริกซ์ความหนาแน่น

ในหัวข้อนี้เราจะมาศึกษาสิ่งที่เรียกว่า *เมทริกซ์ความหนาแน่น* หรือ *Density Matrix* กันครับ ซึ่ง Density Matrix นี้สำคัญมาก ๆ ในเคมีควอนตัมเพราะว่าถูกนำมาใช้เยอะมาก ๆ ในหลาย ๆ สมการ คำถามหลาย ๆ ข้อ เช่น “Density Matrix คืออะไร?”, “ทำไมเราต้องมีสิ่งนี้ด้วย?”, “ประโยชน์หรือความสำคัญของ Density Matrix คืออะไร?” เราจะมาหาคำตอบกันในบทนี้ครับ

ผมขอเริ่มด้วย Fock Matrix $F_{pq}$ ซึ่งมีสมการดังต่อไปนี้ (เราเพิ่งศึกษากันไปในหัวข้อที่แล้ว)

$$
\begin{aligned}
F_{pq} &= \left\langle \phi_p | \hat{f} | \phi_q \right\rangle \\
&= \left\langle \phi_p | \hat{h} | \phi_q \right\rangle + \sum_{j=1}^{n} \left\langle \phi_p \left| \hat{J}_j - \hat{K}_j \right| \phi_q \right\rangle \\
&= \left\langle \phi_p | \hat{h} | \phi_q \right\rangle + \sum_{j=1}^{n} \langle \phi_p \chi_j | \hat{g} | \phi_q \chi_j \rangle - \langle \phi_p \chi_j | \hat{g} | \chi_j \phi_q \rangle \\
&= \left\langle \phi_p | \hat{h} | \phi_q \right\rangle + \sum_{j=1}^{n} \sum_{r,s=1}^{m} c_{jr} c_{js} \left( \langle \phi_p \phi_r | \hat{g} | \phi_q \phi_s \rangle - \langle \phi_p \phi_r | \hat{g} | \phi_s \phi_q \rangle \right) \\
&= \left\langle \phi_p | \hat{h} | \phi_q \right\rangle + \sum_{r,s=1}^{m} D_{rs} \left( \langle \phi_p \phi_r | \hat{g} | \phi_q \phi_s \rangle - \langle \phi_p \phi_r | \hat{g} | \phi_s \phi_q \rangle \right)
\end{aligned}
\tag{1.12.10}
$$

ไอเดียก็คือเมื่อเราทำการเขียนออร์บิทัลให้อยู่ในรูปของผลคูณของสัมประสิทธิ์ (Coefficients) กับ Basis Function แล้วเราก็ทำการจัดรูปสมการให้ Coefficients นั้นมาคูณกัน ซึ่งเราจะทำการกำหนดให้ Density Matrix นั้นคือเมทริกซ์ที่เป็นผลคูณระหว่างสัมประสิทธิ์ของออร์บิทัล (Coefficients) นั่นเอง ดังนี้

$$
D_{rs} = \sum_{j=1}^{n} c_{jr} c_{js} \tag{1.12.11}
$$

นอกจากนี้เรายังสามารถกำหนด Notation เพิ่มเติมสำหรับ Two-Electron Integrals ได้อีกด้วย ดังนี้

$$
G_{prqs} = \langle \phi_p \phi_r | \hat{g} | \phi_q \phi_s \rangle - \langle \phi_p \phi_r | \hat{g} | \phi_s \phi_q \rangle \tag{1.12.12}
$$

ซึ่งทำให้เราสามารถเขียนสมการ Fock Matrix ที่เรียบง่ายกว่าเดิมได้ดังนี้

$$
F_{pq} = h_{pq} + \sum_{r,s=1}^{m} D_{rs} G_{prqs} \tag{1.12.13}
$$

ไม่เพียงแค่ Fock Matrix กับ Two-Electron Integrals เท่านั้นที่เราสามารถเขียนสมการใหม่ให้อยู่ในรูปที่มี Density Matrix ได้ แต่ยังมีปริมาณอื่น ๆ อีก เช่น พลังงานที่ได้จาก Slater Determinant ก็สามารถเขียนให้มีเทอม Density Matrix ได้ด้วย โดยเราเริ่มต้นจากสมการพลังงานก่อน

$$
E_0 = \sum_{i=1}^{n} \left\langle \chi_i | \hat{h} | \chi_i \right\rangle + \frac{1}{2} \sum_{i,j=1}^{n} \left( \langle \chi_i \chi_j | \hat{g} | \chi_i \chi_j \rangle - \langle \chi_i \chi_j | \hat{g} | \chi_j \chi_i \rangle \right) + V_{nn} \tag{1.12.14}
$$

แล้วก็ทำการใช้ LCAO แล้วก็เขียนให้ Coefficients นั้นมาคูณกัน

$$\begin{aligned} E_0 =& \sum_{i=1}^{n}\sum_{p,q=1}^{m} c_{ip}c_{iq}\left\langle \phi_p|\hat{h}|\phi_q \right\rangle \\ &+ \frac{1}{2}\sum_{i,j=1}^{n}\sum_{p,q,r,s=1}^{m} c_{ip}c_{iq}c_{jr}c_{js}\left(\langle\phi_p\phi_r|\hat{g}|\phi_q\phi_s\rangle - \langle\phi_p\phi_r|\hat{g}|\phi_s\phi_q\rangle\right) \end{aligned} \tag{1.12.15}$$

แล้วก็ตามด้วยการแทนเทอมต่าง ๆ ด้วย Density Matrix และ Two-Electron Integrals

$$E_0 = \sum_{p,q=1}^{m} D_{pq}h_{pq} + \frac{1}{2}\sum_{p,q,r,s=1}^{m} D_{pq}D_{rs}G_{prqs} + V_{nn} \tag{1.12.16}$$

ความสำคัญของ Density Matrix นั้นก็คือความสามารถในการอธิบายสถานะควอนตัมของระบบของเรา โดยเราอาจจะเปรียบเทียบง่าย ๆ ก็ได้ว่า Density Matrix นั้นเป็นเสมือนตัวแทนของ Wavefunction แต่ข้อดีอย่างหนึ่งที่ Density Matrix มีนั้นก็คือข้อมูลที่สมาชิกแนวทแยงของ Density Matrix นั้นเก็บซ่อนไว้นั่นก็คือ Diagonal Elements $(p = q)$ ซึ่งบ่งบอกถึงโอกาสที่ออร์บิทัลนั้นจะอยู่ในสถานะควอนตัมหนึ่ง ๆ หรือที่เราเรียกกันว่า Population ส่วน Off-diagonal Elements $(p \neq q)$ นั้นจะบ่งบอกถึง Coherence ของระบบ

นอกจากนี้แล้วก็ยังมีคุณสมบัติอื่น ๆ ที่น่าสนใจของ Density Matrix เช่น Density Matrix นั้นเป็นตัวกำหนดความหนาแน่นประจุ (Charge Density) ของระบบ และ Density Matrix นั้นไม่ขึ้นกับ Orbitals

### 1.12.3 Basis Set สำหรับการคำนวณโครงสร้างเชิงอิเล็กทรอนิกส์

คราวนี้เราจะมาดูรายละเอียดของ Basis Function โดยผมขอยกตัวอย่างของ Basis Set ที่ได้รับความนิยมมาก ๆ อันหนึ่งนั่นก็คือ 6-31G(d) ซึ่งหลาย ๆ คนมักจะใช้กันตอนที่เตรียม Input File สำหรับรันการคำนวณ เราจะมาดูรายละเอียดประเภทของฟังก์ชันที่เป็นหน้าตาของ Basis Function กันก่อน ในช่วงยุคเริ่มต้นของการพัฒนาวิธีสำหรับการคำนวณ Electronic Structure นั้นได้มีการพัฒนาสิ่งที่เรียกว่า Slater Type Orbitals (STOs) ขึ้นมา ซึ่ง STOs นี้เป็นฟังก์ชันที่ถูกสร้างขึ้นมาจากการนำฟังก์ชัน 2 ฟังก์ชันมารวมกันนั่นคือฟังก์ชันของส่วนเป็นเชิงรัศมี (Radial Part) กับฟังก์ชันของส่วนที่เป็นเชิงมุม (Angular Part) ที่อธิบายรัศมีหรือขนาดของออร์บิทัลและอธิบายรูปร่างของออร์บิทัลตามลำดับ สมการของ STOs คือ

$$R(r) = Nr^{n-1}e^{-\zeta r} \tag{1.12.17}$$

เมื่ออ่านมาถึงจุดนี้แล้วผู้อ่านก็น่าจะเข้าใจได้ทันทีเลยว่า STOs นั้นก็คือฟังก์ชันเริ่มต้นที่ถูกนำมาใช้ในการอธิบายออร์บิทัลหรือฟังก์ชันคลื่น (Wavefunction) ของอิเล็กตรอนที่อยู่ในอะตอมนั้น ๆ ขึ้นมา ถ้าหากเราพลอต STOs Function ให้เป็นฟังก์ชันกับรัศมีแล้วเราจะได้ฟังก์ชันที่มันจะมีความราบเรียบ (Smooth) ตามค่ารัศมีที่เพิ่มขึ้น อย่างไรก็ตามการใช้ STOs นั้นมีข้อจำกัดหรือข้อด้อยสำหรับการนำไปใช้ในการคำนวณก็คือเทอมที่เป็น Two-Electron Integral หรือ Electron Repulsion Integral (ERI) ที่ถูกอินทิเกรตโดยใช้ STOs นั้นคำนวณได้ยากมาก ๆ ดังนั้นในช่วงเวลาต่อมาจึงได้มีการพัฒนาฟังก์ชันที่เหมือนกับว่าคล้าย ๆ กับ STOs ขึ้นมาแต่สามารถนำไปใช้ได้ในกรณีที่หลากหลายกว่า (General) นั่นก็คือ Gaussian Type Orbitals (GTOs) ซึ่งก็ตามชื่อเลยนั่นคือฟังก์ชันที่ใช้เป็น Gaussian Function โดยมีสมการคือ

$$G_{nlm}(r,\theta,\psi) = N_n \underbrace{r^{n-1}e^{-\alpha r^2}}_{\text{radial part}} \underbrace{Y_l^m(\theta,\psi)}_{\text{angular part}} \tag{1.12.18}$$

ความแตกต่างระหว่าง STOs กับ GTOs ก็คือเทอมที่เป็นดีกรีหรือกำลังของฟังก์ชัน Exponential ใน GTOs นั้นเราจะมีการนำรัศมีมายกกำลัง $(r^2)$ แต่ว่าใน STOs นั้นรัศมีจะเป็นแค่กำลังหนึ่งเท่านั้น GTOs นั้นมีประโยชน์มาก ๆ ในการคำนวณเพราะว่าเราสามารถคำนวณ ERI ได้ง่ายกว่า STO มาก ผู้อ่านที่สนใจรายละเอียดของทฤษฎีสามารถอ่านบทความวิชาการของ S.F. Boys ที่ตีพิมพ์งานวิจัยในปี 1950 หรือประมาณ 70 ปีที่แล้วได้ ผมขอสรุปอย่างนี้ครับว่าความแตกต่างระหว่าง STOs กับ GTOs นั้นก็คือลักษณะพฤติกรรมของตัวฟังก์ชันที่ $r = 0$ (ที่จุดศูนย์กลางของอะตอม) กับที่ $r = \inf$ (Infinity) หรือที่ใกลจากนิวเคลียสมาก ๆ โดยที่ STOs นั้นจะมี Cusp หรือจุดที่เป็นการเปลี่ยนหรือ Transition ระหว่าง States ที่ตำแหน่ง $r = 0$ ในขณะที่ GTOs นั้นจะมีความไม่ถูกต้องที่ตำแหน่ง $r = 0$ นอกจากนี้คือลักษณะของฟังก์ชัน GTO จะมีค่าที่ลดลงเร็วกว่า STO มากโดยเฉพาะตำแหน่งที่อิเล็กตรอนนั้นอยู่ห่างจากนิวเคลียสแบบไกล ๆ $(r = \inf)$ นอกจากนี้แล้วยังมีฟังก์ชันแบบพิเศษอีกอันนึงที่เรียกว่า Contracted Gaussian Type Orbitals ด้วยซึ่งเป็นการปรับปรุงให้ GTOs สามารถอธิบายพฤติกรรมของอิเล็กตรอนสำหรับออร์บิทัลเชิงอะตอมได้ดียิ่งขึ้น

กลับมาที่คำถามของเรานั่นก็คือ Basis Set สำคัญยังไง คำตอบคือ Basis Set นั้นจะประกอบไปด้วยข้อมูลที่เราจะนำมาใช้ในการสร้าง MOs นั่นเอง โดยที่ในไฟล์หนึ่งไฟล์นั้นจะมีข้อมู, เช่น ประเภทของออร์บิทัล (Orbital Types), จำนวนของ Primitive Gaussian, Scale Factor, Orbital Exponent และที่สำคัญคือ Coefficients ที่จะถูกนำมาใช้ในการสร้าง Wavefunction เริ่มต้นนั่นเอง โดยฟอร์แมทของ Basis Set ในไฟล์นั้นมีดังนี้

```
atomic symbol
Shell_type, No. of primitive Gaussians, Scale_factor
Orbital exponent, Contraction coefficient
  [repeat x times]
```

โดยที่ $x$ คือจำนวนของ Primitive Gaussian ตัวอย่างเช่น Basis Set “STO-3d” ของอะตอมคาร์บอนนั้นมีดังนี้

```
C 0
S 3 1.00
.7161683735D+02 .1543289673D+00
.1304509632D+02 .5353281423D+00
.3530512160D+01 .4446345422D+00
SP 3 1.00
.2941249355D+01 -.9996722919D-01 .1559162750D+00
.6834830964D+00 .3995128261D+00 .6076837186D+00
.2222899159D+00 .7001154689D+00 .3919573931D+00
```

เราจะมาดูทีละแถวกันครับ

- แถวแรกคือระบุว่าเป็นออร์บิทัล 1s ของอะตอมคาร์บอนซึ่งสามารถเขียนได้ด้วยผลรวมของ Primitive Gaussian 3 อันโดยมีตัวคูณปรับขนาด (Scale Factor) คือ 1

- แถวที่ 2-4 คือเป็น Orbital exponent และ coefficients ตามลำดับ

ดังนั้นสำหรับออร์บิทัล 1s ของอะตอมคาร์บอนนั้นจะมาสามารถเขียนได้เป็นผลรวมของเทอม STO Functions 3 ฟังก์ชันรวมกันนั่นเอง สำหรับออร์บิทัลอื่น ๆ ของอะตอมคาร์บอนนั้นก็ทำแบบเดียวกันแต่ว่าจะมีเทคนิคบางอย่างมาช่วยให้การคำนวณนั้นทำได้เร็วขึ้น เช่น ออร์บิทัล 2s กับ 2p นั้นจะใช้ Orbital Exponent ค่าเดียวกันแต่ว่าจะใช้ Contraction Coefficients ที่ต่างกัน คราวนี้เราลองมาทำแบบฝึกหัดสั้น ๆ ในการนับจำนวนของ Basis Functions ที่เราต้องการนำมาใช้สำหรับโมเลกุล Methanol ($CH_4O$) กัน เริ่มต้นเลยสำหรับ AOs 1s, 2s, และ 2p นั้นเราจะใช้ Gaussian 3 ฟังก์ชัน ดังนั้นเราจะมี Basis Function 5 อันสำหรับคาร์บอนแต่ละอะตอมและสำหรับอะตอมออกซิเจนด้วย และจะมีแค่ Basis Function 1 ฟังก์ชันสำหรับอะตอมไฮโดรเจนแต่ละตัว ดังนั้นรวมทั้งหมดเราจะมี 14 Basis Functions ซึ่งก็เท่ากับ 14 MO-coefficients สำหรับการทำ SCF Calculation ในแต่ละรอบนั่นเอง คราวนี้ Basis Function ทั้ง 14 อันนั้นจะมี Gaussian Primitive อีก 3 อันย่อย ดังนั้นจำนวน Primitives ทั้งหมดของโมเลกุล $CH_4O$ จึงเท่ากับ $14 = 42$

สำหรับการคำนวณจำนวน Basis Function และ Gaussian Primitive นั้นมีรายละเอียดอีกเยอะพอสมควร ขึ้นอยู่กับว่าใช้ Basis Set แบบไหน เพราะว่า Basis Set นั้นมีหลายประเภท เช่น Split-valence, Double Zeta, Polarization, หรือ Diffuse Functions นอกจากนี้การเลือกใช้ Basis Set นั้นก็ขึ้นอยู่กับประเภทของโมเลกุลรวมถึงสิ่งที่ต้องการคำนวณด้วย

## 1.13 พลังงานสหสัมพันธ์ของอิเล็กตรอน

พลังงานสหสัมพันธ์ของอิเล็กตรอน (Electron Correlation Energy) เป็นเทอมพลังงานอีกเทอมนึงที่สำคัญมาก ๆ ซึ่งเป็นเทอมที่อธิบายถึงอันตรกิริยาระหว่างอิเล็กตรอนซึ่งในทฤษฎี HF นั้นไม่มีเทอมนี้ดังนั้น

ทำให้ค่าพลังงานที่ได้จากการคำนวณนั้นไม่ถูกต้องดังนั้นจึงได้มีการพัฒนาทฤษฎีการคำนวณเพื่อเพิ่มความถูกต้องให้กับวิธี HF ซึ่งเพิ่มหรือรวม Correlation Energy เข้าไปด้วยโดยเราเรียกวิธีเหล่านั้นว่าวิธี Post-HF

กำหนดให้ Correlation Energy ของโมเลกุลที่อยู่ในสถานะ $i$ นั้นมีสมการดังต่อไปนี้

$$E_i^{\text{corr}} = E_i^{\text{exact}} - E_i^{\text{HF}} \tag{1.13.1}$$

โดยเทอม $E_i^{\text{exact}}$ คือพลังงานที่แท้จริงของโมเลกุลซึ่งเป็นผลเฉลยของสมการชโรดิงเงอร์ซึ่งก็ไม่มีใครรู้ว่ามีค่าเท่าไหร่ (สำหรับระบบที่มีอิเล็กตรอนตั้งแต่ 2 ตัวขึ้นไป) เนื่องจากว่า Hamiltonian ของโมเลกุลนั้นมีคุณสมบัติที่เหมือนกันกับ Fock Operator ตามสมการที่ (1.11.10) ดังนั้น “Exact Wavefunction” $\psi_i^{\text{exact}}$ นั้นจึงสามารถให้อยู่ในรูปของผลรวมของผลเฉลยของวิธี HF สำหรับแต่ละ State $\psi_\mu^{\text{HF}}$ ได้ดังนี้

$$\psi_i^{\text{exact}} = \sum_\mu C_{i\mu} \psi_\mu^{\text{HF}} \tag{1.13.2}$$

โดยที่ $C_{i\mu}$ คือสัมประสิทธิ์ของการกระจาย (Expansion Coefficients) ที่เราจะต้องคำนวณนั่นเอง ในทฤษฎี Molecular Orbital นั้นเราจะโฟกัสไปที่ Hartree-Fock State $\left(\psi_\mu^{\text{HF}}\right)$ และการหา $C_{i\mu}$ ที่สอดคล้องกันนั่นเอง โดยวิธีที่ผู้อ่านจะได้ศึกษาในหัวข้อนี้จะอ้างอิงกับวิธี Restricted Hartree-Fock เช่น Closed-Shell System (ระบบที่มีอิเล็กตรอนเป็นเลขคู่และออร์บิทัลแต่ละอันนั้นมีอิเล็กตรอนบรรจุอยู่ 2 ตัว) สำหรับ Hartree-Fock State นั้นจริง ๆ แล้วก็คือ State ของโมเลกุลที่เป็นไปได้ทั้งหมดนั่นเองซึ่งก็จะแตกต่างกันไปตาม Configuration ของอิเล็กตรอน กรณีที่เป็นสถานะพื้นนั้นโมเลกุลจะมี State ได้เพียงแบบเดียวซึ่งจะมีพลังงานที่ต่ำที่สุดด้วยและเมื่ออิเล็กตรอนถูกกระตุ้นให้กระโดดขึ้นไปอยู่ในออร์บิทัลที่สูงขึ้นก็จะนับเป็น State ใหม่

## 1.14 ทฤษฎีฟังก์ชันคลื่น

### 1.14.1 วิธี Configuration Interaction

ทฤษฎีอันหนึ่งที่ได้รับความนิยมและถูกพัฒนาและใช้มาอย่างยาวนานแล้วก็คือ Configuration Interaction (CI) ซึ่งถ้าจะให้ผมแปลเป็นภาษาไทยก็น่าจะแปลได้เป็น “วิธีปฏิสัมพันธ์ขององค์ประกอบของอิเล็กตรอน” ซึ่งผมเชื่อว่าผู้อ่านหลายคนก็คงจะงงกันแน่ ดังนั้นผมจะขอเรียกวิธีนี้ด้วยชื่อภาษาอังกฤษแทนเพราะว่าคำศัพท์เชิงเทคนิคหลาย ๆ คำนั้นถ้าเราเรียกโดยใช้ภาษาอังกฤษจะเข้าใจได้ง่ายกว่า โอเคครับแล้ววิธี CI คืออะไรกันแน่ ผมจะพยายามอธิบายตามที่ผมเข้าใจครับ

คำว่า Configuration นั้นคือเป็นการอธิบายว่า Wavefunction นั้นสามารถถูกเขียนให้อยู่ในรูปของผลรวมเชิงเส้นของ Slater Determinants หลาย ๆ อันได้ ส่วนคำว่า Interactions จะหมายถึงปฏิสัมพันธ์หรือ

อันตรกิริยาระหว่างการจัดเรียงอิเล็กตรอนในรูปแบบที่แตกต่างกันไป พูดง่าย ๆ ก็คือถ้าเรานำการจัดเรียงอิเล็กตรอนที่เป็นไปได้แต่ละแบบมารวมกันก็จะเกิด Interaction ขึ้นนั่นเอง โดยในทางเคมีควอนตัมนั้นการจัดเรียงอิเล็กตรอนหรือ (Electron Configuration) นั้นคือ State ของ Wavefunction นั่นเอง คำถามถัดมาคือแล้ววิธี CI นั้นอธิบาย Electron Correlation ให้กับวิธี HF ได้ยังไง คำตอบก็คือวิธี CI นั้นก็ใช้หลักการเดียวกันกับวิธี HF นั่นก็คือใช้ Variational Wavefunction ที่เป็นผลรวมเชิงเส้นของ Configuration State Functions (CSFs) ที่ถูกสร้างขึ้นมาจาก Spin Orbitals โดยเราสามารถเขียน Wavefunction ของวิธี CI ให้อยู่ในรูปของผลรวมเชิงเส้นของ CSFs หลาย ๆ อันรวมกันได้ ดังนี้

$$\begin{aligned}\psi_0^{\mathrm{CI}} =& C_0\psi_0^{\mathrm{HF}} + \sum_{\mu} C_\mu \psi_\mu^{(1)} + \sum_{\mu} C_\mu \psi_\mu^{(2)} + \dots \\ =& C_0\psi_0^{\mathrm{HF}} + \sum_{i}\sum_{a} C_i^a \psi_i^a + \sum_{\substack{i,\\ j>i}}\sum_{\substack{a,\\ b>a}} C_{ij}^{ab}\psi_{ij}^{ab} + \dots\end{aligned} \tag{1.14.1}$$

โดยที่ $\psi_i^{\mathrm{HF}}$ คือ State ที่เป็น Ground State ของวิธี HF แบบปกติ, $\psi_\mu^{(1)}$ คือ State ที่มีการกระตุ้นอิเล็กตรอน 1 ตัวหรือ Single Excitation ของ HF Wavefunction, และ $\psi_\mu^{(2)}$ คือ State ที่มีการกระตุ้นอิเล็กตรอน 2 ตัวไหนก็ได้ในออร์บิทัลหรือ Double Excitation ของ HF Wavefunction และในบรรทัดที่สองของสมการ (1.14.1) นั้น $i$ และ $j$ หมายถึง Occupied Orbitals แล้วก็ $a$ และ $b$ นั้นหมายถึง vVirtual Orbitals (Orbitals ที่ไม่มีอิเล็กตรอนหรือจะเรียกว่า Unoccupied Orbitals ก็ได้เช่นกัน) ตามลำดับ ดังนั้นถ้าเราเจอการเขียน Wavefunction ของวิธี CI ด้วย $\psi_i^a$ และ $\psi_{ij}^{ab}$ นั่นก็คือว่าเรากำลังสนใจ Singly Excited State กับ Doubly Excited State โดยที่มี HF Ground State เป็นสถานะอ้างอิง (Reference State) นั่นเอง ซึ่งโดยทั่วไปแล้วเราจะทำการตัด (Truncate) เทอมที่สูงกว่า Doubly Excited State แล้วเราก็จะได้ว่ามีแค่ Singly กับ Doubly เท่านั้น ดังนั้นเราจึงเรียกวิธีนี้ว่า CISD (CI ที่มีแค่ Singles กับ Doubles Excitations) ส่วน Coefficients $C_\mu$ นั้นจะถูกคำนวณได้โดยการใช้ Variation Method

### Brillouin's Theorem

ทฤษฎีบทอันหนึ่งที่ใช้ในการอธิบายวิธี CI ก็คือทฤษฎีบทของบริลลูอิน (Brillouin's Theorem) โดยผมขอเริ่มต้นอธิบายด้วยสมการของ Energy Contribution ซึ่งเกิดจากการเชื่อมโยง (Coupling) กันระหว่าง HF State กับ Single Excitations ทั้งหมดที่เป็นไปได้ ดังนี้

$$\begin{aligned}&\sum_i\sum_a \left\langle \psi_0^{\mathrm{HF}} \middle| \hat{H}^{\mathrm{el}} \middle| \psi_i^a \right\rangle \\ &\quad= \sum_i\sum_a \left( \left\langle \chi_i | \hat{h} | \chi_a \right\rangle + \frac{1}{2}\sum_j \left( \langle \chi_i\chi_j | \hat{g} | \chi_a\chi_j \rangle - \langle \chi_i\chi_j | \hat{g} | \chi_j\chi_a \rangle \right) \right) \\ &\quad= \sum_i\sum_a \left\langle \chi_i | \hat{f} | \chi_a \right\rangle = \sum_i\sum_a \epsilon_i \langle \chi_i \mid \chi_a \rangle = \sum_i\sum_a \epsilon_i \delta_{ia} = 0\end{aligned} \tag{1.14.2}$$

โดยความสัมพันธ์ตามสมการที่ (1.14.2) นั้นเป็น Brillouin's Theorem ซึ่งอธิบายความสัมพันธ์ระหว่าง Fock Operator, Coulomb Operator, และ Exchange Operator ซึ่งผู้อ่านได้ศึกษาไปในบทที่แล้ว

**Full CI**

ลำดับต่อไปคือทฤษฎีที่เป็นการเพิ่มความถูกต้องให้กับวิธี CI นั่นก็คือทฤษฎี Full Configuration Interaction (Full CI) โดยคำว่า Full ในที่นี้แปลว่า "ทั้งหมด" หมายความว่า Full CI นั้นจะรวมรูปแบบของการกระตุ้น (Excitations) ของอิเล็กตรอนที่เป็นไปได้ทั้งหมดเข้าไว้ด้วยกัน โดยจำนวนของรูปแบบที่เป็นไปได้ทั้งหมดนั้นกำหนดให้เขียนแทนด้วย $N_{\mathrm{FCI}}$ สามารถคำนวณได้ก็คือว่าถ้าเรามีอิเล็กตรอน $n$ ตัวแล้วเราจะทำการใส่อิเล็กตรอนทั้งหมดนี้เข้าไปในออร์บิทัลซึ่งมีจำนวน $m$ ออร์บิทัล ได้กี่วิธี (โดยมีเงื่อนไขว่าเราสามารถใส่อิเล็กตรอนได้มากที่สุดต่อออร์บิทัลคือ 2 ตัว) ซึ่งก็คือเป็นการใช้ Combinatorics ตามสมการดังต่อไปนี้

$$N_{\mathrm{FCI}} = \frac{(2m)!}{n!\,(2m-n)!} \tag{1.14.3}$$

โดยที่ Scaling ของจำนวนที่เป็นไปได้ในการจัด Excitations นั้นจะแปลผันตาม Factorial ของขนาดของปัญหาซึ่งก็คือจำนวนของอิเล็กตรอนและจำนวนออร์บิทัลที่เรามี นั่นจึงทำให้วิธี Full CI นั้นมีความสิ้นเปลืองสูงมากจึงทำให้วิธีนี้เหมาะกับระบบโมเลกุลขนาดเล็ก ๆ เท่านั้น สรุปก็คือว่ายิ่งเราเลือกใช้ Basis Set ที่มีขนาดใหญ่ จำนวนของ Basis Function ก็จะสูงตามไปด้วยและทำให้ Full CI นั้นสิ้นเปลืองมาก ๆ เพราะว่าจำนวนออร์บิทัลนั้นจะเท่ากับจำนวน Basis Functions เสมอซึ่งเป็นผลมาจากการทำ Diagonalization ของ Eigenvalue Problem นั่นเอง อย่างไรก็ตามวิธี Full CI นั้นให้คำตอบหรือผลการคำนวณแบบที่เป็น Exact Solution ซึ่งมีความถูกต้องสูงมาก ๆ เมื่อเทียบกับวิธีอื่นที่พิจารณาหรือรวม Electron Correlation เข้าไปด้วย

## 1.14.2 ทฤษฎี Møllor-Plesset Perturbation

ในบทนี้เราจะมาดูรายละเอียดของทฤษฎีอีกอันหนึ่งซึ่งหลาย ๆ คนรู้จักกันดีและได้รับความนิยมสูงมาก ๆ นั่นก็คือ Møllor-Plesset Perturbation Theory ผมขอเริ่มต้นด้วยการอธิบายก่อนว่าวิธี Møllor-Plesset Perturbation นั้นมี Hamiltonian Operator ที่รวม Electron Correlation เข้าไปได้ยังไงและจะแสดงว่าเห็นว่าพลังงานสุดท้ายที่ได้ออกมานั้นมีความแตกต่างจากพลังงานที่ได้จากการคำนวณด้วยวิธี HF

สำหรับวิธี Møllor-Plesset (MP) Perturbation นั้นเราจะมีสมมติฐานเริ่มต้นว่า $\hat{H}_0$ เป็น Operator ของวิธี HF ดังนั้น

$$\begin{aligned}\hat{H}_{\mathrm{HF}} &= \sum_{i=1}^{n} \hat{f}_i \\ &= \sum_{i=1}^{n} \left( \hat{h}_i + \sum_{j=1}^{n} \left( \hat{J}_j - \hat{K}_j \right) \right)\end{aligned} \tag{1.14.4}$$

โดยที่เทอม Coulomb Operator $\left(\hat{J}_j\right)$ และ Exchange Operator $\left(\hat{K}_j\right)$ นั้นจะถูกนำมาคิดรวม 2 ครั้ง เพราะว่าเรามีอิเล็กตรอน 2 ตัวสำหรับแต่ละคู่ $(i$ และ $j)$ ซึ่งก็จะสอดคล้องกับพลังงานของ HF นั่นเอง

คราวนี้มามีการนำ Perturbation Operator $\left(\hat{H}_1\right)$ เข้ามาใช้ซึ่งเราจะได้สมการดังต่อไปนี้

$$\begin{aligned}\hat{H}_1 &= \hat{H}^{\mathrm{el}} - \hat{H}_{\mathrm{HF}} \\ &= \hat{V}_{ee} - \sum_{i,j=1}^{n} \left( \hat{J}_j - \hat{K}_j \right)\end{aligned} \tag{1.14.5}$$

โดยที่ $\hat{H}^{\mathrm{el}}$ คือ Molecular Hamiltonian สิ่งที่น่าสนใจเกี่ยวกับ $\hat{H}_1$ ก็คือว่าเทอมที่ 2 ของทางด้านขวาของสมการที่ (1.14.5) นั้นมีค่าประมาณเป็นสองเท่าของเทอมแรกและตามหลักการ Perturbation นั้น $\hat{H}_1$ ควรจะต้องมีค่าน้อยที่สุดเท่าที่จะเป็นไปได้ ดังนั้นเราจึงได้ว่าพลังงานของระบบแบบที่ยังไม่มี Perturbation หรือ Zeroth-Order Energy $\left(\varepsilon_0^{(0)}\right)$ นั้นจะกลายเป็น

$$\varepsilon_0^{(0)} = \sum_{i=1}^{n} \epsilon_i \tag{1.14.6}$$

ถ้าเราเขียนสมการพลังงานด้านบนนี้ให้อยู่ในรูปของพลังงานคูลอมบ์ (Coulomb) และพลังงานแลกเปลี่ยน (Exchange) เราจะได้ว่าพลังงานที่เราจะใส่เข้าไปเพื่อทำให้พลังงานของ HF นั้นถูกต้องมากขึ้นหรือที่เรียกว่า Correction Energy แบบลำดับที่ 1 $\left(\varepsilon_0^{(1)}\right)$ ซึ่งได้จากการใช้ Perturbation Operator $\left(\hat{H}_1\right)$ มีหน้าตาดังต่อไปนี้

$$\varepsilon_0^{(1)} = -\frac{1}{2} \sum_{i,j=1}^{n} \left( J_{ij} - K_{ij} \right) \tag{1.14.7}$$

โดยที่ว่าถ้าหากเราทำการอินทิเกรตเทอมแรกของสมการที่ (1.14.5) จะหักล้างพอดีกับครึ่งหนึ่งของการอินทิเกรตเทอมที่สอง ดังนั้นพลังงาน HF $\left(E_0\right)$ สามารถเขียนใหม่ได้เป็น

$$E_0 = \varepsilon_0^{(0)} + \varepsilon_0^{(1)} + V_{nn} \tag{1.14.8}$$

โดยที่เราทำการเพิ่มพลังงานที่เกิดจากการผลักกันระหว่างนิวเคลียสเข้าไปได้ด้วย นอกจากนี้เราจะพบว่าพลังงาน Electron Correlation นั้นจะถูกรวมอยู่ในเทอม Second-Order Contribution ของพลังงานที่ได้จากการคำนวณด้วยวิธี MP เนื่องจากว่าผลรวมของ Zeroth-Order Contribution กับ First-Order Correction นั้นมีค่าเท่ากับพลังงาน HF สรุปสั้น ๆ ก็คือถ้าหากเราทำ Correction โดยใช้ First-Order MP สิ่งที่เราจะได้ออกมาก็คือพลังงาน HF นั่นเองดังนั้นเราจึงมักจะทำการ Correction ด้วย Second-Order หรือลำดับที่สูงกว่า เป็นต้น

สำหรับการคำนวณหาพลังงาน Second-Order Correction ของวิธี MP นั้นจะค่อนข้างซับซ้อนนิดหน่อยแต่ผมสรุปสั้น ๆ ดังนี้ก็คือว่าเราเริ่มด้วยสมการ Second-Order Perturbation

$$\varepsilon_0^{(2)} = \sum_{j=1}^{\infty} \frac{\left\langle \psi_0^{(0)} \middle| \hat{H}_1 \middle| \psi_j^{(0)} \right\rangle \left\langle \psi_j^{(0)} \middle| \hat{H}_1 \middle| \psi_0^{(0)} \right\rangle}{\varepsilon_0^{(0)} - \varepsilon_j^{(0)}} \tag{1.14.9}$$

ตามทฤษฎี CI นั้นเราสามารถเขียน Excited States Wavefunction $\left(\psi_j^{(0)}\right)$ ให้อยู่ในรูปของผลรวมของ Single Excitations $\left(\psi_i^a\right)$, Double Excitations $\left(\psi_{ij}^{ab}\right)$, และเทอมที่สูงกว่าได้ และตามหลักการ Slater-Condon นั้น Excitation ที่เป็นเทอมสูง ๆ นั้นจะมี Contribution ต่อสมการที่ (1.14.9) ที่น้อยมากเมื่อเทียบกับเทอม Double Excitation นอกจากนี้สำหรับ Single Excitations นั้นเราสามารถทฤษฎีบท Brillouin ได้อีกด้วยซึ่งจะทำให้เทอมบางเทอมนั้นมีค่าเท่ากับ 0 (ไม่มี Contribution) ดังนี้

$$\begin{aligned}\sum_i \sum_a \langle \psi_0^{\mathrm{HF}} | \hat{H}^{\mathrm{el}} - \sum_j \hat{f}_j | \psi_{ij}^{ab} \rangle &= \sum_i \sum_a \underbrace{\langle \psi_0^{\mathrm{HF}} | \hat{H}^{\mathrm{el}} | \psi_{ij}^{ab} \rangle}_{=0} \\ &\quad - \left( \sum_j \epsilon_j \right) \underbrace{\langle \psi_0^{\mathrm{HF}} | \psi_i^a \rangle}_{=0}\end{aligned} \tag{1.14.10}$$

พูดง่าย ๆ ก็คือ Single Excitations นั้นไม่มีผลหรือไม่มี Contribution ต่อพลังงานของ Second-Order Møllor-Plesset Perturbation (MP2) เลยและเรายังพบอีกว่าท้ายที่สุดแล้วเทอมที่เป็น Double Excitations นั้นจะมีหน้าตาสมการดังต่อไปนี้

$$\begin{aligned}\sum_{\substack{i \\ j>i}} \sum_{\substack{a \\ b>a}} \langle \psi_0^{\mathrm{HF}} | \hat{H}^{\mathrm{el}} - \sum_k \hat{f}_k | \psi_{ij}^{ab} \rangle &= \sum_{\substack{i \\ j>i}} \sum_{\substack{a \\ b>a}} \left\langle \psi_0^{\mathrm{HF}} \middle| \hat{V}_{ee} \middle| \psi_{ij}^{ab} \right\rangle \\ &= \sum_{\substack{i \\ j>i}} \sum_{\substack{a \\ b>a}} \langle \chi_i \chi_j | \hat{g} | \chi_a \chi_b \rangle - \langle \chi_i \chi_j | \hat{g} | \chi_b \chi_a \rangle\end{aligned} \tag{1.14.11}$$

ซึ่งจะทำให้เราสามารถเขียนสมการพลังงานของ MP2 ออกมาได้ดังนี้

$$\epsilon_0^{(2)} = \sum_{\substack{i \\ j>i}} \sum_{\substack{a \\ b>a}} \frac{\left|\langle \chi_i \chi_j | \hat{g} | \chi_a \chi_b \rangle - \langle \chi_i \chi_j | \hat{g} | \chi_b \chi_a \rangle\right|^2}{(\epsilon_i - \epsilon_a) + (\epsilon_j - \epsilon_b)} \tag{1.14.12}$$

โดยที่ในตัวส่วนของ Fraction ด้านบนนั้นคือพลังงานกระตุ้น (Excitation Energy) สำหรับ Wavefunction ที่ไม่ถูกรบกวน (Unperturbed) ซึ่งเท่ากับผลต่างของพลังงานของออร์บิทัล

### 1.14.3 ทฤษฎี Coupled Cluster

คราวนี้ก็มาถึงหัวข้อที่ถือได้ว่าเป็นอีกหนึ่งพระเอกของเคมีควอนตัมนั่นก็คือทฤษฎี Coupled Cluster (CC) โดยผมขออ้างอิงประโยคต่อไปนี้จาก Wikkipedia

> The method was initially developed by Fritz Coester and Hermann Kümmel in the 1950s for studying nuclear-physics phenomena, but became more frequently used when in 1966 Jiri Cizek (and later together with Josef Paldus) reformulated the method for electron correlation in atoms and molecules. It is now one of the most prevalent methods in quantum chemistry that includes electronic correlation.

จะเห็นได้ว่าทฤษฎี CC นั้นถูกพัฒนามานานมากกว่า 70 ปีแล้ว ซึ่งทฤษฎีนี้ถูกนำมาใช้กับโจทย์เคมีควอนตัมเพื่อใช้ในการหาคำตอบของสมการชโรดิงเงอร์ที่สอดคล้องกับฟังก์ชันคลื่นที่สามารถอธิบาย Electron Correlation ได้อย่างถูกต้อง

ในหัวข้อของทฤษฎี CI นั้นเราได้ดูรายละเอียดไปแล้วว่าถ้าหากเราสามารถสร้างฟังก์ชันคลื่นให้เป็นฟังก์ชันที่เกิดจากฟังก์ชันเล็ก ๆ หลาย ๆ ฟังก์ชันที่แยกออกจากกันและยังมีคุณสมบัติของเอกลักษณ์การคูณ (Multiplicatively Separable) สำหรับระบบที่มี Fragment ที่ไม่มีอันตรกิริยาต่อกัน (Noninteracting Fragments) ตัวอย่างเช่นระบบที่ประกอบไปด้วย Fragments 2 อันคือ Fragment A กับ Fragment B นั้น ฟังก์ชันคลื่นของ Fragments ระบบนี้ควรจะต้องมีหน้าตาเป็นแบบนี้

$$\left|\Psi^{AB}\right\rangle = \left|\Psi^A \Psi^B\right\rangle \tag{1.14.13}$$

นี่จึงเป็นที่มาของการพัฒนาทฤษฎีที่ชื่อว่า Coupled Cluster (CC) Theory ซึ่งแก้ปัญหานี้โดยการใช้ฟังก์ชันคลื่นแบบเลขชี้กำลัง (Exponential Wavefunction) ซึ่งมีฟอร์มดังต่อไปนี้

$$|\Psi_{\mathrm{CC}}\rangle = e^{\hat{T}} |\Phi_0\rangle = \left(1 + \hat{T} + \frac{1}{2!}\hat{T}^2 + \frac{1}{3!}\hat{T}^3 + \ldots\right) |\Phi_0\rangle \tag{1.14.14}$$

โดยที่โอเปอร์เรเตอร์ $\hat{T}$ นั้นถูกกำหนดด้วยโอเปอร์เรเตอร์ $\hat{C}$ ซึ่งเป็นโอเปอร์เรเตอร์ที่ใช้ในวิธี CI ซึ่งเป็นผลรวมเชิงเส้น (Linear Combination) ของโอเปอร์เรเตอร์ของอันดับที่หนึ่ง, อันดับสอง, ไปเรื่อย ๆ จนถึงอันดับที่ $n$

$$\hat{T} = \hat{T}_1 + \hat{T}_2 + \ldots + \hat{T}_n \tag{1.14.15}$$

โดยที่โอเปอร์เรเตอร์อันนี้จะเป็นการกำหนดกับการกระตุ้น (Excitation) $k$-body Excitation Operator $\hat{T}_k$ ซึ่งถูกกำหนดด้วยสมการดังต่อไปนี้

$$\hat{T}_k = \frac{1}{(k!)^2} \sum_{ij\cdots}^{\text{occ}} \sum_{ab\cdots}^{\text{vir}} t_{ij\ldots}^{ab\cdots} \underbrace{\hat{a}_a^\dagger \hat{a}_b^\dagger \cdots \hat{a}_j \hat{a}_i}_{k\text{-fold excitation}} \tag{1.14.16}$$

พารามิเตอร์ $t_{ij\ldots}^{ab\cdots}$ นั้นมีชื่อเรียกว่า Coupled Cluster Amplitudes และเป็นตัวแปรหลัก (Central Variables) ในทฤษฎี CC

เพื่อให้เห็นภาพมากขึ้นว่าทำไมฟังก์ชันคลื่นของ CC นั้นถึงมีเอกลักษณ์การแยกกันแบบการคูณได้ เราลองมายกตัวอย่างด้วยระบบโมเลกุล 2 โมเลกุล ซึ่งเรามีสมการฟังก์ชันคลื่นของ CC สำหรับระบบทั้งสองอันแยกกัน ดังนี้

$$\left|\Psi_{\text{CC}}^A\right\rangle = e^{\hat{T}^A} \left|\Phi_0^A\right\rangle, \tag{1.14.17}$$

$$\left|\Psi_{\text{CC}}^B\right\rangle = e^{\hat{T}^B} \left|\Phi_0^B\right\rangle \tag{1.14.18}$$

ถ้าหากว่าระบบ A กับระบบ B นั้นรวมเป็นระบบเดียวกันก็คือมีอันตรกิริยาต่อกัน $(A \cdots B)$ เราจะสามารถเขียนฟังก์ชันคลื่นที่ Coupled กันนี้ได้โดยการเขียนผลคูณของฟังก์ชันคลื่น (Product Wavefunction) ดังนี้

$$\left|\Psi_{\text{CC}}^A \Psi_{\text{CC}}^B\right\rangle = e^{\hat{T}^A} e^{\hat{T}^B} \left|\Phi_0^A \Phi_0^B\right\rangle = e^{\hat{T}^A + \hat{T}^B} \left|\Phi_0^A \Phi_0^B\right\rangle = \left|\Psi_{\text{CC}}^{AB}\right\rangle \tag{1.14.19}$$

ซึ่งเราสามารถทำแบบนี้ได้เพราะว่า $\exp\left(\hat{T}^A\right)\exp\left(\hat{T}^B\right) = \exp\left(\hat{T}^A + \hat{T}^B\right)$ แต่ว่าจะต้องโน๊ตไว้ด้วยนะครับว่าในกรณีปกตินั้น สำหรับโอเปอร์เรเตอร์ $\hat{X}$ และ $\hat{Y}$ เราจะได้ว่า

$$\exp\left(\hat{X}\right)\exp\left(\hat{Y}\right) = \exp\left(\hat{X} + \hat{Y}\right) \tag{1.14.20}$$

ซึ่งสมการด้านบนนั้นจะเป็นจริงก็ต่อเมื่อ $\hat{X}$ และ $\hat{Y}$ นั้น Commute กัน นั่นก็คือ $[\hat{X}, \hat{Y}] = 0$ เนื่องจากว่า Cluster Operator นั้นจะสอดคล้องกับการแทนที่ของ Occupied Orbitals กับ Virtual Orbitals ดังนั้นโอเปอร์เรเตอร์ $\hat{X}$ และ $\hat{Y}$ นั้นจึง Commute กันเสมอ นั่นจึงสรุปได้ว่า $[\hat{T}^A, \hat{T}^B] = 0$

โอเคครับ เมื่อเรากำหนดฟังก์ชันคลื่นของ CC รวมถึงโอเปอร์เรเตอร์ได้แล้ว ลำดับต่อไปเราจะนำฟังก์ชันคลื่น Exponential Wavefunction อันนี้ไปใช้ โดยเราจะนำเข้าไปใส่ในสมการชโรดิงเงอร์ซึ่งจะทำให้เราคำนวณพลังงานและแอมพลิจูดได้ดังนี้

$$\hat{H}e^{\hat{T}} |\Phi_0\rangle = Ee^{\hat{T}} |\Phi_0\rangle \tag{1.14.21}$$

แล้วถ้าเราทำการคูณ $\exp\left(-\hat{T}\right)$ ทางด้านซ้ายทั้งสองข้างของสมการ ดังนี้

$$e^{-\hat{T}}\hat{H}e^{\hat{T}} |\Phi_0\rangle = Ee^{-\hat{T}}e^{\hat{T}} |\Phi_0\rangle = E |\Phi_0\rangle \tag{1.14.22}$$

เราจะได้ว่าพลังงานของระบบนั้นจะสามารถหาได้จากการนำด้านซ้ายของสมการที่ (1.14.22) มาทำการ Projection ลงไปบน Bra ของฟังก์ชันคลื่น $\langle\Phi_0|$ ดังนี้

$$E = \langle\Phi_0| e^{-\hat{T}}\hat{H}e^{\hat{T}} |\Phi_0\rangle \tag{1.14.23}$$

ในขณะที่แอมพลจูดของ CC นั้นสามารถหาได้จากการทำ Projection ลงไปบน Excited Determinants $\left\langle\Phi_{ij\cdots}^{ab\cdots}\right|$ ดังนี้

$$0 = \left\langle\Phi_{ij\cdots}^{ab\cdots}\right| e^{-\hat{T}}\hat{H}e^{\hat{T}} |\Phi_0\rangle \tag{1.14.24}$$

คำถามคือ *แล้วเราจะคำนวณเทอม $e^{-\hat{T}}\hat{H}e^{\hat{T}}$ อย่างไร* จริง ๆ แล้วคำตอบก็คือเราสามารถใช้สมการที่มีชื่อว่า Baker–Campbell–Hausdorff (BCH) เข้ามาช่วยได้ ซึ่งสรุปแบบสั้น ๆ ก็คือว่าสมการ BCH นั้นบอกไว้ว่าเราสามารถทำการกระจาย (Express) เทอม $e^{-\hat{T}}\hat{H}e^{\hat{T}}$ ได้โดยการเขียนให้อยู่ในรูปอนุกรมของ Commutators ดังนี้

$$\begin{aligned} e^{-\hat{T}}\hat{H}e^{\hat{T}} =& \hat{H} + [\hat{H}, \hat{T}] + \frac{1}{2!}[[\hat{H}, \hat{T}], \hat{T}] + \frac{1}{3!}[[[\hat{H}, \hat{T}], \hat{T}], \hat{T}] + \frac{1}{4!}[[[[\hat{H}, \hat{T}], \hat{T}], \hat{T}], \hat{T}] + \dots \\ =& \hat{H} + \sum_{k=1}^{\infty} \underbrace{[\dots[[\hat{H}, \hat{T}], \hat{T}]\dots]}_{k \text{ nested commutators}} \end{aligned} \tag{1.14.25}$$

โดยอนุกรมด้านบนนี้เป็นอนุกรมอนันต์ (Infinite Series) แต่เนื่องจากว่า Components ของโอเปอร์เรเตอร์ $\hat{T}$ ทั้งหมดนั้น Commute กัน เราจึงได้ว่าอนุกรมของ BCH **จะทำการจัดเทอมที่อยู่หลังจาก Four-fold Commutator Term ออกไป** แล้วก็หลักการอันนี้ก็เป็นจริงกับทุกระบบและไม่ขึ้นกับจำนวนอิเล็กตรอนของระบบด้วย แล้วก็หลักการอันนี้สำคัญมาก ๆ เพราะว่าเราสามารถหาเทอมทุกเทอมในสมการ CC โดยที่เราไม่จำเป็นที่จะต้องทำการประมาณค่าของ $e^{-\hat{T}}\hat{H}e^{\hat{T}}$

เพื่อทำให้วิธี CC นั้นสามารถใช้งานได้ง่ายขึ้นและประมาณค่าพลังงานได้ดีเทียบกับ FCI นั้น เราสามารถทำการตัด (Truncate) โอเปอร์เรเตอร์ $\hat{T}$ ให้สั้นลงได้ (เป็นผลรวมของโอเปอร์เรเตอร์ย่อยแค่ไม่กี่เทอม) โดยการทำแบบนี้นั้นก็คือเป็นการใช้ Approximation อย่างหนึ่งสำหรับฟังก์ชันคลื่นของ CC ซึ่ง Approximation ที่ง่ายที่สุดนั้นก็คือใช้โอเปอร์เรเตอร์ 2 ตัว หรือที่เราเรียกว่า Cluster Cluster with Doubles (CCD) ซึ่งใช้การประมาณ $\hat{T} \approx \hat{T}_2$ โดยสมการชโรดิงเงอร์ที่ใช้ฟังก์ชันคลื่น CCD นั้นจะมีดังต่อไปนี้

$$\hat{H}e^{\hat{T}_2}\left|\Phi_0\right\rangle = E_{\mathrm{CCD}}e^{\hat{T}_2}\left|\Phi_0\right\rangle \tag{1.14.26}$$

ซึ่งใช้การทำ Projection ลงไปบนฟังก์ชันคลื่น $\Phi_0$ ซึ่งจะให้ผลลัพธ์ดังต่อไปนี้

$$\left\langle\Phi_0\right|\hat{H}e^{\hat{T}_2}\left|\Phi_0\right\rangle = E_{\mathrm{CCD}}\left\langle\Phi_0\right|e^{\hat{T}_2}\left|\Phi_0\right\rangle \tag{1.14.27}$$

โดยที่สมาชิกเมทริกซ์ (Matrix Element) ทางด้านซ้ายของสมการด้านบนนี้นั้นมีหน้าตาดังนี้

$$\begin{aligned}\left\langle\Phi_0\right|e^{\hat{T}_2}\left|\Phi_0\right\rangle &= \left\langle\Phi_0\right|\left(1+\hat{T}_2+\ldots\right)\left|\Phi_0\right\rangle \\ &= \left\langle\Phi_0|\Phi_0\right\rangle + \underbrace{\frac{1}{4}\sum_{ij}^{\mathrm{occ}}\sum_{ab}^{\mathrm{vir}}\left\langle\Phi_0\middle|\Phi_{ij}^{ab}\right\rangle + \ldots}_{=0} = 1\end{aligned} \tag{1.14.28}$$

เนื่องจากว่า Determinants ทุกอันนั้นมีความเป็น Orthogonality ต่อกัน จึงทำให้ Contribution ทุกอันในสมการด้านบนเท่ากับ 0 ยกเว้นแค่อันแรกที่ไม่เท่ากับ 0

ในทำนองเดียวกัน ทางด้านขวาของสมการที่ (1.14.27) ก็จะกลายเป็น

$$\begin{aligned}\langle\Phi_0|\,\hat{H}e^{\hat{T}_2}\,|\Phi_0\rangle &= \langle\Phi_0|\,\hat{H}\left(1+\hat{T}_2+\ldots\right)|\Phi_0\rangle \\ &= \langle\Phi_0|\hat{H}|\Phi_0\rangle + \frac{1}{4}\sum_{ij}^{\text{occ}}\sum_{ab}^{\text{vir}}\left\langle\Phi_{ij}^{ab}\middle|\hat{H}\middle|\Phi_0\right\rangle \\ &= E_{\text{HF}} + \frac{1}{4}\sum_{ij}^{\text{occ}}\sum_{ab}^{\text{vir}}\langle ij||ab\rangle\, t_{ij}^{ab}\end{aligned} \tag{1.14.29}$$

คราวนี้เราจะมาลองวิเคราะห์ฟังก์ชันคลื่นของ CCD กันครับเพื่อที่ว่าเราจะได้เข้าใจโครงสร้างของฟังก์ชันคลื่นอันนี้มากขึ้น เริ่มต้นด้วยการลองกระจาย Exponential ซึ่งเราจะได้ว่า

$$|\Psi_{\text{CCD}}\rangle = e^{\hat{T}_2}\,|\Phi_0\rangle = |\Phi_0\rangle + \hat{T}_2\,|\Phi_0\rangle + \underbrace{\frac{1}{2!}\hat{T}_2^2\,|\Phi_0\rangle + \frac{1}{3!}\hat{T}_2^3\,|\Phi_0\rangle + \ldots}_{\text{unlinked quadruples, hextuples, ...}} \tag{1.14.30}$$

เทอมแรกในฟังก์ชันคลื่น $\Psi_{\text{CCD}}$ นั้นคือ Hartree–Fock Determinant แล้วก็ตามด้วยเทอม Double Substitutions ซึ่งถูกสร้างด้วยโอเปอร์เรเตอร์ $\hat{T}_2$ อย่างไรก็ตาม เราจะพบด้วยว่า Contribution ทั้งหมดที่เกี่ยวข้องกับ Quadruple Substitution หรือที่สูงกว่านั้นเกิดมาจากคลัสเตอร์โอเปอร์เรเตอร์ที่มีอันดับกำลังสูง ๆ ($\hat{T}_2^2$ เป็นต้น) ซึ่งเทอมทั้งหมดนี้นั้นสร้างหรือทำให้เกิดการกระตุ้นที่ไม่เกี่ยวข้องกัน (Unlinked Excitations) ซึ่งจะไม่มีอยู่ในวิธี Configuration Interaction

นอกจากนี้ความสิ้นเปลืองของวิธี CCD นั้นแปรผันตรงกับ $O^2V^4$ โดยที่ $O$ คือจำนวนของ Occupied Orbitals และ $V$ คือจำนวนของ Virtual orbitals และเนื่องจากว่าการที่เราเติมเทอม Single Excitation เข้าไปในไม่ทำให้ Computational Scaling ของ CC นั้นเปลี่ยน ดังนั้นวิธี CC ที่รวม Singles และ Doubles Excitations เข้าไปนั้น (เรียกว่า CCSD Approximation) สิ้นเปลืองพอ ๆ กันดับวิธี CCD จึงทำให้วิธี CCSD นั้นเหมาะสมกว่าวิธี CCD ในแง่ของการนำมาใช้งานจริง เหตุผลอีกข้อนั้นก็คือเทอม Singles นั้นจำเป็นด้วย

พิจารณาฟังก์ชันคลื่นของ CCSD ดังนี้

$$|\Psi_{\text{CCSD}}\rangle = e^{\hat{T}_1+\hat{T}_2}\,|\Phi_0\rangle = e^{\hat{T}_2}e^{\hat{T}_1}\,|\Phi_0\rangle \tag{1.14.31}$$

โดยที่เราทำตามเงื่อนไขที่ว่า $\hat{T}_1$ นั้น Commute กับ $\hat{T}_2$ เพื่อที่เราจะสามารถทำการปรับสมการข้างบนนี้ได้

นอกจากนี้เรามี **Thouless' Theorem** ที่บอกไว้ว่าเหตุผลที่เราจำเป็นต้องรวมเทอม Single Excitation $\left(\exp\left(\hat{T}_1\right)\right)$ เข้าไปใน Determinant ของฟังก์ชันคลื่น $\Phi_0$ นั้นก็เพื่อทำให้เรามี Determinant อันใหม่อีกอันซึ่งยังไม่ถูก Normalized (Non-normalized Determinent) ดังนี้

$$e^{\hat{T}_1} |\Phi_0\rangle = C \left|\tilde{\Phi}_0\right\rangle \tag{1.14.32}$$

โดยที่ $C$ คือค่าคงที่ของการการทำ Normalization ดังนั้นโอเปอร์เรเตอร์ $\hat{T}_1$ ในวิธี CC นั้นจึงอาจจะถูกตีความได้ว่าเป็นการสร้าง Slater Determinant ในอีกแบบหนึ่งที่ต่างไปจาก Slater Determinant Reference

ส่วนพลังงานของ CC นั้นสามารถเขียนให้อยู่ในรูปทั่วไปได้ดังนี้

$$\begin{aligned}\langle\Phi_0| \hat{H}e^{\hat{T}} |\Phi_0\rangle &= \langle\Phi_0| \hat{H} \left(1 + \hat{T}_1 + \hat{T}_2 + \frac{1}{2}\hat{T}_1^2 + \ldots\right) |\Phi_0\rangle \\ &= E_{\mathrm{HF}} + \sum_{i}^{\mathrm{occ}}\sum_{a}^{\mathrm{vir}} f_{ia}t_i^a + \frac{1}{4}\sum_{ij}^{\mathrm{occ}}\sum_{ab}^{\mathrm{vir}} \langle ij||ab\rangle\, t_{ij}^{ab} + \frac{1}{2}\sum_{ij}^{\mathrm{occ}}\sum_{ab}^{\mathrm{vir}} \langle ij||ab\rangle\, t_i^a t_j^b\end{aligned} \tag{1.14.33}$$

อย่างไรก็ตาม ถึงแม้ว่า CCSD นั้นจะให้ผลการคำนวณพลังงานของระบบที่ถูกต้องประมาณหนึ่ง แต่ก็ยังไม่มากพอที่จะถูกต้องและแม่นยำในระดับ Chemical Accuracy (ยังมีส่วนต่างอยู่เยอะมาก) ซึ่งถูกกำหนดให้มีค่าเท่ากับ 1 kcal/mol หรือน้อยกว่านั้น เพื่อทำให้ความแม่นยำของวิธี CCSD นั้นเพิ่มมากขึ้น จึงได้มีการนำเสนอวิธี CCSDT ซึ่งเป็นการเติม Triples เข้าไป โดยเราจะได้ฟังก์ชันคลื่นของ CCSDT ดังนี้

$$|\Psi_{\mathrm{CCSDT}}\rangle = e^{\hat{T}_1+\hat{T}_2+\hat{T}_3} |\Phi_0\rangle \tag{1.14.34}$$

แต่ว่าปัญหาที่ตามมาก็คือว่าวิธี CCSDT นั้นมีความสิ้นเปลืองในการคำนวณสูงมาก ๆ ซึ่งแปรผันตรงตาม $O^3V^5$ ดังนั้นในทางปฏิบัติจึงได้มีการพัฒนาและใช้วิธีการที่เรียกว่า CCSD(T) ซึ่ง (T) นั้นคือการเติมเทอม Triples ผ่านวิธี Perturbation ซึ่งทำให้ CCSD(T) นั้นได้รับความนิยมอย่างมากในการนำมาใช้คำนวณพลังงานเพื่อเป็นค่าอ้างอิงเทียบกับวิธีอื่น ๆ เพราะว่าให้ค่าความคลาดเคลื่อนที่น้อยกว่า 1 kcal/mol โดยวิธี CCSD(T) ได้รับการยอมรับว่าเป็น Gold Standard หรือวิธีมาตรฐานของเคมีควอนตัม สำหรับ CCSD(T) นั้นเป็นการคำนวณ Second-Order Triples ($\hat{T}_3^{(2)}$) เพื่อนำมาใช้ในการหาเทอม Fouth-Order Perturbative Corrections สำหรับการคำนวณพลังงาน [$E^{(4)}$] ที่เกี่ยวข้องกับ Singles, Doubles, และ Triples ซึ่งถูกเพิ่มเข้าไปในค่าพลังงานของวิธี CCSD ดังนี้

$$E_{\mathrm{CCSD(T)}} = E_{\mathrm{CCSD}} + E_T^{(4)} + E_{ST}^{(4)} + E_{DT}^{(4)} \tag{1.14.35}$$

ความสิ้นเปลืองของการคำนวณพลังงาน $E_{\mathrm{CCSD(T)}}$ นั้นแปรผันตรงกับ $O^3V^4$ และวิธีนี้ก็ให้ผลการคำนวณที่ถูกต้องในระดับเดียวกันกับวิธี CCSDT (หรือบางครั้งก็ให้ผลการคำนวณที่ถูกต้องกว่าด้วย)

ในปัจจุบันนั้นเนื่องจากว่าโมเลกุลที่มีขนาดเล็ก (จำนวนอะตอมไม่เกิน 20 อะตอม) นั้นถูกศึกษาแบบละเอียดจนแทยจะทุกโมเลกุลแล้ว (ยกเว้นโมเลกุลบางประเภทที่นักวิจัยทางทฤษฎีเอาไว้ทดสอบทฤษฎีใหม่ ๆ ที่พัฒนาขึ้นมา) แล้วก็โมเลกุลที่นักเคมีในปัจจุบันสนใจนั้นก็เป็นโมเลกุลใหม่ที่โดยส่วนใหญ่แล้วก็มีขนาดใหญ่ด้วย ทำให้การใช้วิธี CC กับโมเลกุลเหล่านี้นั้นยากขึ้นไปอีก ทำให้ผู้ใช้งาน (Users) ส่วนใหญใช้วิธีอื่น ๆ ที่ง่ายกว่า ถึงแม้จะให้ผลการคำนวณที่ไม่ได้แม่นยำแบบสุด ๆ ก็ตาม เช่น วิธี DFT

ไม่ใช่แค่ที่ประเทศไทยอย่างเดียวที่คนไม่ค่อยใช้วิธี Post-HF แบบขั้นสูงกัน แต่ที่ต่างประเทศนั้นก็มีสัดส่วนของคนที่ทำงานวิจัยทางด้านเคมีที่ใช้วิธี Post-HF น้อยเหมือนกันเมื่อเทียบกับวิธีการอื่นที่ประหยัดการคำนวณมากกว่า ซึ่งกลุ่มวิจัยส่วนใหญ่ในต่างประเทศที่ยังทำงานวิจัยทางด้าน Post-HF ต่างก็เป็นนักพัฒนาทฤษฎีหรือพยายามทำให้วิธี Post-HF นั้นสามารถนำมาใช้จริงกับโมเลกุลที่มีขนาดใหญ่ได้ง่ายและสะดวกมากขึ้น

## 1.15 ทฤษฎีฟังก์ชันนอลความหนาแน่น

และแล้วในที่สุดผมก็พาผู้อ่านมาถึงหัวข้อที่อาจจะเรียกได้ว่าสำคัญมาก ๆ ในเคมีควอนตัมยุคใหม่นั่นก็คือทฤษฎีฟังก์ชันนอลความหนาแน่น (Density Functional Theory หรือ DFT) ซึ่ง DFT เป็นทฤษฎีที่เปรียบเสมือนเป็นอีกทางเลือกหนึ่งของทฤษฎีโครงสร้างเชิงอิเล็กทรอนิกส์เพราะว่าหลักการหรือไอเดียของ DFT นั้นคือพยายามหลีกเลี่ยง Schrödinger Equation พูดง่าย ๆ ก็คือเราจะไม่ได้ใช้ Wavefunction ในการอธิบายระบบแต่จะเปลี่ยนมาใช้ความหนาแน่นของอิเล็กตรอนหรือ Electron Density แทนนั่นเอง

DFT นั้นจะใช้หลักการขั้นตอนของ Kohn-Sham หรือที่เรียกสั้น ๆ ว่า KS-DFT ซึ่งได้ถูกนำเสนอในปี ค.ศ. 1965 และก็ได้กลายมาเป็นหนึ่งในเครื่องมือหลักทางเคมีควอนตัม สาเหตุที่ทำให้ KS-DFT นั้นประสบความสำเร็จในแง่ของการนำไปใช้จริงในการศึกษาระบบโมเลกุลแบบต่าง ๆ นั้นก็คือ KS-DFT ได้รวม Electron Correlation เข้าไปด้วยทำให้ KS-DFT นั้นให้ผลการคำนวณที่ถูกต้องเมื่อเทียบกับผลการทดลองและยังมีความสิ้นเปลืองของการคำนวณที่น้อยพอ ๆ กับวิธี HF อีกด้วย[1]

ถ้าอยากที่จะเข้าใจ DFT นั้นจะต้องทำความเข้าใจทฤษฎีอันแรกที่เป็นจุดเริ่มต้นไอเดียของ DFT ก่อนนั่นก็คือ Hohenberg-Kohn Theorem ซึ่งสรุปใจความได้ว่าพลังงานและคุณสมบัติอื่น ๆ ของระบบที่สถานะพื้นนั้นจะถูกกำหนดหรือขึ้นกับความหนาแน่นของอิเล็กตรอนเพียงแค่แบบเดียวเท่านั้น นั่นหมายความว่าเราสามารถเขียน Hamiltonian ให้อยู่ในรูปของความหนาแน่นของอิเล็กตรอนได้ เราสามารถเขียนสมการคณิตศาสตร์ของฟังก์ชันของพลังงานที่ขึ้นกับความหนาแน่นของอิเล็กตรอน ($E[\rho(\vec{r})]$) ได้ดังนี้

[1]หลายคนยังเข้าใจผิดว่าทฤษฎี DFT ได้รับรางวัลโนเบลเพราะว่าเป็นทฤษฎีให้ผลการคำนวณเคมีควอนตัมที่แม่นยำมาก ๆ ซึ่งจริง ๆ แล้วไม่ใช่เลย เหตุผลที่แท้จริงก็คือว่าเป็นเพราะแนวคิด (Concept) ของทฤษฎีที่ใช้ Electron Density แทนฟังก์ชันคลื่นตรง ๆ ต่างหากที่ทำให้การคำนวณเคมีควอนตัมนั้นใช้เวลาคำนวณที่ไม่นานมากเมื่อเทียบกับวิธีอื่น ๆ ซึ่งทำให้วิธี DFT นั้นถูกนำมาใช้อย่างแพร่หลายในหลาย ๆ สาขา สามารถนำไปใช้งานได้จริงกับระบบโมเลกุลตั้งแต่ขนาดเล็กไปจนถึงขนาดใหญ่ แต่ว่าในปัจจุบันนี้ก็มีปัญหาหลายอย่างที่ทำให้ทฤษฎี DFT นั้นยังไม่สามารถอธิบายได้ ทำให้ DFT นั้นยังต้องได้รับการปรับปรุงและพัฒนาต่อไป

$$E[\rho(\vec{r})] = \int V_{\text{ext}}(\vec{r})\rho(\vec{r})\mathrm{d}\vec{r} + F[\rho(\vec{r})] \tag{1.15.1}$$

โดยที่ $V_{\text{ext}}(\vec{r})$ คือศักย์ไฟฟ้าสถิตย์จากภายนอก (External Electrostatic Potential) ซึ่งมาจากนิวเคลียสของระบบโมเลกุลและ $F[\rho(\vec{r})]$ คือฟังก์ชันของพลังงานที่เราไม่มีรู้หน้าตา (Unknown Energy Functional) ซึ่งเทอมนี้ก็รวมพลังงานจลน์ (Kinetic Energy) ของอิเล็กตรอนและพลังงานอัตรกิริยาระหว่างอิเล็กตรอน (Electron-Electron Interaction) เข้าไปด้วย

นอกจากนี้ถ้าหากเราทำการอินทิเกรตหรือรวมความหนาแน่นของอิเล็กตรอนทั้งหมดเข้าด้วยเราจะได้ผลลัพธ์เท่ากับจำนวนของอิเล็กตรอน $n$ ดังนี้

$$\int \rho(\vec{r})\mathrm{d}\vec{r} = n \tag{1.15.2}$$

และถ้าหากเรานำ Variational Principle เข้ามาใช้ด้วยเราจะได้ว่า Energy Functional $(E[\rho(\vec{r})])$ นั้นจะถูกทำให้มีค่าต่ำที่สุด (Minimization) เมื่อเทียบกับความหนาแน่นของอิเล็กตรอนและมีเงื่อนไขว่าจำนวนของอิเล็กตรอนนั้นจะต้องเท่าเดิมเสมอ ดังนี้

$$\frac{\delta}{\delta\rho(\vec{r})}\left(E[\rho(\vec{r})] - \mu\int \rho(\vec{r})\mathrm{d}\vec{r}\right) = 0 \tag{1.15.3}$$

โดยที่ $\delta$ คืออนุพันธ์เชิงฟังก์ชัน (“Functional Derivative”) และ $\mu$ คือ Lagrangian Multiplier สำหรับเงื่อนไขที่เรากำหนดไว้ในสมการที่ (1.15.2) ดังนั้นเราจึงสามารถเขียน Derivative ใหม่ได้เป็น

$$\left(\frac{\delta E[\rho(\vec{r})]}{\delta\rho(\vec{r})}\right)_{V_{\text{ext}}} = \mu \tag{1.15.4}$$

ซึ่งเราอาจจะพิจารณาได้ว่า DFT นั้นเทียบเท่าหรือเหมือนกันกับ Schrödinger Equation

### 1.15.1 ขั้นตอน Kohn-Sham

จริง ๆ แล้วหัวใจสำคัญของ DFT นั้นก็คือ Kohn-Sham Approach ซึ่งใน KS-DFT สิ่งที่เป็นปัญหาตอนนี้ก็คือเทอม Unknown Energy Functional $F[\rho(\vec{r})]$ (ในสมการที่ (1.15.1)) ซึ่งไม่มีใครรู้ว่าหน้าตาเป็นอย่างไร คราวนี้เราจะมาดูกันไปพร้อม ๆ กันว่าเราจะทำยังไงกับเทอมนี้ดี

เริ่มต้นก็คือใน KS-DFT นั้นเราจะแบ่งเทอม $F[\rho(\vec{r})]$ ให้อยู่ในรูปของผลรวมของพลังงานย่อย 3 เทอม ดังนี้

$$F[\rho(\vec{r})] = E_{\text{KE}}[\rho(\vec{r})] + E_{\text{H}}[\rho(\vec{r})] + E_{\text{XC}}[\rho(\vec{r})] \tag{1.15.5}$$

ดังนั้นสมการที่ (1.15.1) จึงกลายเป็น

$$E[\rho(\vec{r})] = \int V_{\text{ext}}(\vec{r})\rho(\vec{r})\mathrm{d}\vec{r} + E_{\text{KE}}[\rho(\vec{r})] + E_{\text{H}}[\rho(\vec{r})] + E_{\text{XC}}[\rho(\vec{r})] \tag{1.15.6}$$

โดยที่ทั้งสามเทอมที่เพิ่มขึ้นมานั้นมีคำอธิบายดังนี้

- $E_{\text{KE}}[\rho(\vec{r})]$ คือพลังงานจลน์สำหรับอิเล็กตรอนแก๊ส (Ideal Gas) ที่มีความหนาแน่นของอิเล็กตรอนที่ถูกต้องก็คือเทียบเท่ากับความหนาแน่นของอิเล็กตรอนของระบบจริง ๆ
- $E_{\text{H}}[\rho(\vec{r})]$ คือพลังงาน Hartree ซึ่งจะเป็นพลังงานที่เกี่ยวข้องกับพลังงานไฟฟ้าสถิตย์ (Classical Electrostatics) และพลังงานคูลอมบ์ในวิธี Hartree-Fock
- $E_{\text{XC}}[\rho(\vec{r})]$ คือฟังก์ชันของ Exchange-Correlation ที่เป็นเทอมที่อธิบายพลังงานของอันตริกิริยาระหว่างอิเล็กตรอน

ประเด็นก็คือว่าอย่างแรกเลยคือเราต้องการสมการคณิตศาสตร์ที่สามารถใช้ในการอธิบายความหนาแน่นของอิเล็กตรอนก่อนและตัวเลือกที่ดีที่สุดใน Kohn-Shan Approach ก็คือใช้ออร์บิทัล $(\varphi(\vec{r}_i))$ นั่นเอง

$$\rho(\vec{r}) = \sum_{i=1}^{n} |\varphi(\vec{r}_i)|^2 \tag{1.15.7}$$

โดย $\rho(\vec{r})$ เป็น Representation ที่ขึ้นกับตัวแปรเพียงแค่ 3 ตัวเท่านั้น นอกจากนี้แล้วเรายังสามารถ Basis Sets แบบเดียวกันกับที่เราใช้ใน Wavefunction-Based Theory ได้อีกด้วย และสำหรับเทอม $V_{\text{XC}}$ นั้นเราสามารถนิยามได้โดยใช้ Derivative ของ Energy Functional ได้ดังนี้

$$V_{\text{XC}} = \left(\frac{\delta E_{\text{XC}}[\rho(\vec{r})]}{\delta\rho(\vec{r})}\right)_{V_{\text{ext}}} \tag{1.15.8}$$

โดยที่ $V_{XC}$ คือฟังก์ชันนอลของ Exchange-Correlation Functional ซึ่งพอเรานำทุกเทอมมารวมกันแล้วเราจะได้ว่า Hamiltonian Operator ของ KS-DFT นั้นจะสามารถนำไปใช้ในการหาพลังงานของระบบได้จาก Kohn-Sham Orbitals ดังนี้

$$(V_{\text{KE}} + V_{\text{ext}} + V_{\text{H}} + V_{\text{XC}})\,\varphi_i = \varepsilon_i\varphi_i \tag{1.15.9}$$

แล้วถ้าหากว่าผู้อ่านยังจำกันได้ว่า Fock Operator นั้นมีสมการดังต่อไปนี้

$$\hat{f}_i = \hat{h}_i + \sum_{j=1}^{n} \left( \hat{J}_j - \hat{K}_j \right) \tag{1.15.10}$$

เราจะสามารถเทียบเคียงเทอมแต่ละเทอมใน Fock Operator กับ Kohn-Sham Hamiltonian Operator ได้ดังนี้

- $\hat{h}_i$ คือเทอม One-Electron ซึ่งในที่นี้ก็คือสอดคล้องกับเทอม $V_{\mathrm{KE}}$ และ $V_{\mathrm{ext}}$ นั่นเอง
- $\hat{J}_j$ นั้นก็คือเทอม Coulomb ซึ่งก็สอดคล้องกับ $V_{\mathrm{H}}$
- สุดท้ายคือเทอม Exchange $\hat{K}_j$ ซึ่งจะถูกแทนที่ด้วยฟังก์ชันนอล Exchange-Correlation ของ Kohn-Sham Appraoch $V_{\mathrm{XC}}$ นั่นเอง

ดังนั้นเมื่อเราพิจารณาเทอมต่าง ๆ ใน Kohn-Sham Approach แล้วเราอาจจะสรุปได้ว่าจริง ๆ แล้ว Kohn-Sham Hamiltonian Operator นั้นก็คือ Fock operator ที่ถูกดัดแปลงมานั่นเอง ดังนี้

$$\hat{f}_i^{\mathrm{DFT}} = V_{\mathrm{KE}} + V_{\mathrm{ext}} + V_{\mathrm{H}} + V_{\mathrm{XC}} \tag{1.15.11}$$

โดยที่ได้รวมผลของ Correction สำหรับ Electron Correlation เข้าไปด้วยโดยผ่านเทอม Exchange-Correlation Functional $V_{\mathrm{XC}}$ นั่นเอง อย่างไรก็ตามสมการหรือหน้าตาของ $V_{\mathrm{XC}}$ นั้นไม่มีใครรู้ (จนถึงทุกวันนี้) ดังนั้นจึงได้มีนักเคมีและนักฟิสิกส์พัฒนา Functional $V_{\mathrm{XC}}$ ออกมาเยอะมาก ๆ ให้เราได้เลือกใช้กัน โดย Functionals เหล่านี้หลาย ๆ ตัวก็ถูกพัฒนาขึ้นมาเพื่อวัตถุประสงค์บางอย่างในการคำนวณคุณสมบัติบางประการของโมเลกุลโดยเฉพาะ แล้วก็มีอีกหลาย Functionals ที่ถูกพัฒนาขึ้นมาโดยใช้ข้อมูลจากการทดลอง (Empirical Pameters) เข้ามาช่วยในการเพิ่มความถูกต้อง ดังนั้นการเลือกใช้ Functional ที่เหมาะสมกับระบบโมเลกุลที่เราต้องการศึกษานั้นจึงเป็นสิ่งที่สำคัญมาก ๆ เพราะว่าถ้าเราเลือกใช้ Functional ที่ไม่ดีอาจจะทำให้ได้ผลการคำนวณที่ไม่ถูกต้องได้

## 1.16 ฟังก์ชันนอล

เทอมที่สำคัญที่สุดของ DFT ก็คือ (Unknown) Kohn-Sham Potential นั่นคือฟังก์ชันนอล (Functional) ซึ่งเป็นสิ่งที่อธิบายพฤติกรรมของอิเล็กตรอน โดยเราเรียกฟังก์ชันนอลนี้ว่า Exchange-Correlation (XC) Functional

### 1.16.1 ฟังก์ชันนอลแบบ Local

วิธีที่ง่ายมาก ๆ ในการประมาณค่าหรือหน้าตาของ Exchange-Correlation Function คือการเขียนฟังก์ชันนอลในรูปของอินทิกรัลของความหนาแน่น (Density) กับพลังงานที่ขึ้นอยู่กับความหนาแน่นนั้น ๆ ณ ตำแหน่งใดตำแหน่งหนึ่งในโมเลกุล (Local-Density) ซึ่งมีวิธีการนี้มีชื่อเรียกว่า Local-Density Approximation (LDA) อ้างอิงกับโมเดลของ Uniform Electron Gas (UEG) โดยวิธีนี้ถูกเสนอโดย Kohn และ Sham ในปี ค.ศ. 1965 ดังนี้

$$E_{xc}^{LDA}[\rho] = \int d\mathbf{r}\rho(\mathbf{r})\epsilon_{xc}^{\text{unif}}(\rho(\mathbf{r})) \tag{1.16.1}$$

โดยที่พลังงาน Exchange ต่อหนึ่งหน่วยอนุภาคของ UEG นั้นสามารถคำนวณได้จาก

$$\epsilon_x^{\text{unif}} = c_x\rho^{1/3} \tag{1.16.2}$$

ส่วนพลังงาน Correlation ต่อหนึ่งห่วยอนุภาคนั้นจะมีความซับซ้อนมากกว่า ซึ่งได้จากการปรับฟังก์ชันโดยการ Fit พารามิเตอร์กับข้อมูลของ Quantum Monte Carlo (QMC)

### 1.16.2 ฟังก์ชันนอลแบบ Semi-Local

#### Generalised Gradient Approximation

Generalised Gradient Approximation (GGA) เป็น Semi-Local Functional แบบแรกที่ถูกพัฒนาขึ้นโดยมีสมการทั่วไปดังต่อไปนี้

$$E_{ex}^{GGA}[\rho] = \int d\mathbf{r}F[\rho, \nabla\rho] \tag{1.16.3}$$

สาเหตุที่ฟังก์ชันนอล GGA นั้นมีชื่อเรียกอีกอย่างว่า Semi-Local Approximation ก็เพราะว่าตัวฟังก์ชันนอลนั้นใช้อินทิกรัลของ $\mathbf{r}$ โดยการใช้ “Semilocal Information” ผ่านเทอม $\nabla\rho$ ซึ่งโดยปกติแล้ว GGA นั้นก็สามารถเขียนได้ในแบบที่เป็น Gradient ได้เช่นกัน ดังนี้

$$s(\mathbf{r}) = \frac{|\nabla\rho(\mathbf{r})|}{\rho^{4/3}(\mathbf{r})} \tag{1.16.4}$$

ตัวอย่างของฟังก์ชันนอล GGA เช่น B88 (Exchange), LYP (Correlation), PW91 (Exchange-Correlation), PBE (Exchange-Correlation)

**Meta Generalised Gradient Approximation**

Meta Generalised Gradient Approximation (Meta GGA) ถูกพัฒนาต่อจากฟังก์ชันนอล GGA โดยมีสมการทั่วไปดังต่อไปนี้

$$E_{ex}^{GGA}[\rho] = \int d\mathbf{r} F\left[\rho, \nabla\rho, \nabla^2\rho, \tau\right] \tag{1.16.5}$$

โดยที่ $\tau(\mathbf{r})$ คือ Non-Interacting Positive Kinetic Energy Density

$$\tau(\mathbf{r}) = \frac{1}{2}\sum_{i}^{n} |\nabla\phi_i(\mathbf{r})|^2 \tag{1.16.6}$$

ซึ่งเราใช้เทอม $\tau(\mathbf{r})$ ในการอธิบาย Curvature ของ Exchange Hold (ปรับให้เหมาะสมมากขึ้น) ตัวอย่างของฟังก์ชันนอล Meta GGA เช่น TPSS และ SCAN

## 1.16.3 ฟังก์ชันนอลแบบ Hybrid

หนึ่งในปัญหาหลักของฟังก์ชันนอล GGA ก็คือความคลาดเคลื่อนที่เกิดจากอันตรกิริยาระหว่างอิเล็กตรอนกับตัวมันเอง (Self-Interaction Error) ซึ่งทำให้ผลที่คำนวณด้วย GGA นั้นมีความคลาดเคลื่อนสูง เช่น Electron Density นั้นก็มีค่าที่ไม่ถูกต้องเพราะว่าอิเล็กตรอนนั้น Delocalized มากไป และยังมีปัญหาที่เกี่ยวข้องกับการที่ Charged Fragment ในโมเลกุลนั้นไม่สัมพันธ์กันอีก และยังมีปริมาณอื่น ๆ อีกที่คำนวณออกมาแล้วยังไม่ถูกต้อง เช่น Reaction Barriers รวมถึงระบบโมเลกุลบางประเภทที่ GGA นั้นไม่ยังไม่สามารถ Treat ได้ เช่น Radials และ Excited States

จุดเริ่มต้นของ Hybrid Functional นั้นมาจากการที่ Becke ได้เสนอวิธีการสร้างฟังก์ชันนอลแบบใหม่โดยการนำ Hartree-Fock Exchange มาผสมกับ GGA Functionals ในฟังก์ชันนอลที่ชื่อว่า Three-Parameter Hybrid (3H) Approximation (ในปี ค.ศ. 1993) แล้วก็ได้เสนอฟังก์ชันนอลอีกอันที่ง่ายกว่าคือ One-Parameter Hybrid (1H) Approximation (ในปี ค.ศ. 1996) ตามสมการต่อไปนี้

$$E_{xc}^{3H} = aE_x^{HF} + bE_x^{GGA} + (1-a-b)E_x^{LDA} + cE_c^{GGA} + (1-c)E_c^{LDA} \tag{1.16.7}$$

$$E_{xc}^{1H} = aE_x^{HF} + (1-a)E_x^{DFA} + E_c^{DFA} \tag{1.16.8}$$

โดย $E_x^{HF}[\Phi]$ คือ Hartree-Fock Exchange Energy ที่ถูกคำนวณด้วย Occupied Kohn-Sham Orbitals ซึ่ง Orbitals ที่เอามาใช้นั้นจะถูกคำนวณด้วย Non-Local Exchange Potential $v_x^{HF}\left(\mathbf{r},\mathbf{r}'\right)$ แทนที่จะใช้ Local Exchange Potential (เราอาจจะเรียกเทคนิคแบบนี้ว่าเป็นวิธี KS ก็ได้ แต่ว่าวิธีนี้จะมีความ General กว่า) โดยตัวอย่าง 2 อันที่ถือได้ว่าเป็นฟังก์ชันนอลที่โด่งดังและได้รับการใช้งานมาอย่างยาวนานก็คือฟังก์ชันนอล B3LYP และ PBE0 ซึ่งมีหน้าตาดังนี้

$$E_{xc}^{B3LYP} = 0.20E_x^{HF} + 0.72E_x^{B88} + 0.08E_x^{LDA} + 0.81E_c^{LYP} + 0.19E_c^{VWN} \quad (1.16.9)$$

$$E_{xc}^{PBE0} = 0.25E_x^{HF} + 0.75E_x^{PBE} + E_c^{PBE} \quad (1.16.10)$$

วิธีการสร้างฟังก์ชันนอลแบบ Hybrid นั้นยากตรงที่เราจะหา Coefficients หรือค่าน้ำหนักของเทอม Exchange และ Correlation แต่ละอันได้ยังไง วิธีการทั่วไปคือเราจะทำการปรับค่าพารามิเตอร์ (Parameters) โดยการ Fit สมการของฟังก์ชันนอลเข้ากับค่าอ้างอิงของ Molecular Properties ตัวอย่างอื่นของ Hybrid Functionals เช่น B97 ซึ่งมีพารามิเตอร์ 13 ตัวและ M06 ซึ่งมีพารามิเตอร์ 36 ตัว

| MAE | | HF | MP2 | BLYP | B3LYP |
|---|---|---|---|---|---|
| $r_e$/ Å: | 12 first row diatomics | 0.024 | 0.011 | 0.012 | 0.004 |
| $r_e$/ Å: | 12 second row diatomics | 0.016 | 0.017 | 0.024 | 0.006 |
| $v_0/\mathrm{cm}^{-1}$ : | 122 molecules | 50 | 63 | 45 | 34 |
| $D_0/\mathrm{kcal/mol}$ : | 44 molecules | 86 | 15 | 8 | 5 |
| $\Delta E^{\dagger}/\mathrm{kcal/mol}$ | Diels-Alder (reverse) | -8 | | -16 | -9 |

ตาราง 1.2 ค่าความคลาดเคลื่อนเฉลี่ยสัมบูรณ์ (Mean Absolute Error หรือ MAE)

ตารางที่ 1.2 แสดงค่าความคลาดเคลื่อนเฉลี่ยสัมบูรณ์ (Mean Absolute Error) ของความถูกต้องในการคำนวณปริมาณต่าง ๆ ที่เป็นคุณสมบัติเชิงกายภาพและเชิงเคมีของโมเลกุลแต่ละระบบด้วยวิธีทางเคมีควอนตัมที่แตกต่างกัน (HF = Hartree-Fock, MP2 = Second-Order Møllor-Plesset Perturbation (MP2), BLYP = Becke-Lee-Yang-Parr (BLYP), B3LYP = Becke, 3-parameter, Lee-Yang-Parr)

นอกจากนี้แล้วยังมีฟังก์ชันนอลประเภทใหม่ที่ถูกพัฒนาขึ้นมาอีก เช่น Double-Hybrid Approximation ซึ่งเสนอโดย Grimme ในปี ค.ศ. 2006 และ Range-Separated Hybrid Functionals เช่น Long-Range Correction (LC) ที่เสนอขึ้นในปี ค.ศ. 1996 และ 2001 ตัวอย่างของฟังก์ชันนอลประเภทนี้ที่ได้รับความนิยมก็คือ CAM-B3LYP และ $\omega$B97X

**เจาะลึก B3LYP**

B3LYP เป็นฟังก์ชันนอลในตำนานที่คนทำงานวิจัยโดยใช้วิธี DFT นั้นคุ้นเคยกันดี แล้วมีที่มาแบบละเอียดยังไง? ผมได้ลองค้นคว้าแล้วก็พบว่าบทความงานวิจัยแรกที่ได้เสนอ B3LYP ไว้อย่างครบถ้วนนั้นก็

คือ “Ab Initio Calculation of Vibrational Absorption and Circular Dichroism Spectra Using Density Functional Force Fields”[7] ซึ่งเป็นงานวิจัยที่ทำโดยนักวิจัย 4 คนคือ P. J. Stephens, F. J. Devlin, C. F. Chabalowski, และ M. J. Frisch โดยสมการของ B3LYP นั้นก็มีหน้าตาตามสมการที่ 2 ในบทความดังกล่าว ซึ่งก็เหมือนกับฟอร์มของสมการ Functional Exchange-Correlation ทั่วไปที่ Becke ได้เสนอไว้

จริง ๆ แล้วในเปเปอร์นี้เขาได้นำส่วนผสม 3 อย่างจากงานวิจัย 3 ชิ้นมารวมกัน นั่นคือ

1. B3 (เทอมที่ 3 ในสมการ)
2. LYP (เทอมที่ 4 ในสมการ)
3. VWN (เทอมที่ 5 ในสมการ) - จริง ๆ เทอมนี้คือ Correlation ของ Local Spin Density Approximation (LSDA) ส่วนเทอมแรกในสมการที่เป็นเทอม Local Exchange มาตรฐานทั่วไป แล้วก็เทอมที่ 2 นั้นเป็น HF exchange ธรรมดา

ดังนั้นถ้าเราเขียนบทความงานวิจัยหรือเอกสารทางวิชาการใด ๆ ที่ต้องมีการอ้างอิงบทความงานวิจัยของ B3LYP ก็ควรจะอ้างอิง 4 บทความดังต่อไปนี้

1. A.D. Becke, *J. Chem. Phys.* 98 (1993) 5648-5652
2. C. Lee, W. Yang, R.G. Parr, *Phys. Rev. B* 37 (1988) 785-789
3. S.H. Vosko, L. Wilk, M. Nusair, Can. *J. Phys.* 58 (1980) 1200-1211
4. P.J. Stephens, F.J. Devlin, C.F. Chabalowski, M.J. Frisch, *J. Phys. Chem.* 98 (1994) 11623-11627

ส่วนการ Implement ฟังก์ชันนอล B3LYP นั้นสามารถดูได้ที่ Source Code ของ libxc ซึ่งเป็นไลบรารี่ที่รวบรวมฟังก์ชันนอลของ DFT ไว้ให้นักวิจัยคนอื่น ๆ เอาไปใช้ในโปรแกรมของตัวเอง จะได้ไม่ต้องเสียเวลามาเขียนโค้ดสำหรับ Implement ฟังก์ชันนอลแต่ละอันใหม่[1] ซึ่งก็จะมี Implementation แตกต่างกันไปขึ้นอยู่กับว่าใช้เทอม VWN แบบเวอร์ชันไหน ตัวอย่างเช่นในรูปที่อยู่ในคอมเมนต์นั้นคือฟังก์ชันของ B3LYP3 ซึ่งเป็นการใช้ VWN 3 โดยก็จะไปเรียกใช้งานฟังก์ชันอื่น ๆ อีก เช่น `xc_hyb_init_hybrid`

## 1.17 แบบฝึกหัด

1. จงแสดงว่า Hamiltonian Operator นั้นมี Eigenvalues เป็นส่วนจริง (Real) และมี Eigenfunctions เป็น Orthogonal

2. จงแสดงว่า Normalization Factor ของ Slater Determinant สำหรับระบบที่มีอิเล็กตรอน $n$ ตัว นั้นมีค่าเท่ากับ $1/\sqrt{n!}$

3. เราสามารถแก้สมการ Schrödinger เพื่อหาผลเฉลยแบบแม่นตรง (Exact Solution) สำหรับระบบที่มีโปรตอน 2 ตัวและมีอิเล็กตรอน 1 ตัวได้หรือไม่ เพราะอะไร

[1] https://gitlab.com/libxc/libxc/-/blob/master/src/hyb_gga_xc_b3lyp.c

# บทที่ 2

# พลวัตเชิงโมเลกุลแบบดั้งเดิม

## 2.1 การประยุกต์ใช้ Molecular Dynamics

ก่อนที่เราจะไปศึกษาวิธีการจำลองระบบโมเลกุลที่มีความซับซ้อนนั้นเราควรเริ่มต้นด้วยการศึกษาวิธีอย่างง่ายก่อนนั่นก็คือ Molecular Dynamics (MD) (จริง ๆ จะว่าไปแล้ววิธีนี้ก็ไม่ได้ง่ายนะครับ ถ้าลงรายละเอียดลึก ๆ แล้วก็มีความซับซ้อนมากพอสมควร ซึ่งในปัจจุบันนั้นก็มีการทำวิจัยที่เกี่ยวข้องกับการพัฒนาเทคนิคของวิธี MD อย่างต่อเนื่อง) เราใช้เทคนิคการจำลองทางคอมพิวเตอร์ในการทำความเข้าใจคุณสมบัติของระบบที่ประกอบไปด้วยโมเลกุล ๆ โมเลกุลในเชิงโครงสร้างและอันตรกิริยาในระดับจุลภาคหรือหน่วยย่อย ซึ่งหน่วยย่อยในที่นี้ก็คือโมเลกุลนั่นเอง โดยเราสามารถแบ่งวิธีการจำลองออกได้เป็น 2 วิธีคือ Molecular Dynamics (MD) กับ Monte Carlo (MC) และหนังสือเล่มนี้จะเน้นไปที่ MD ซึ่งเป็นวิธีที่สามารถนำมาใช้ในการศึกษาคุณสมบัติเชิงพลวัตได้ เช่น สัมประสิทธิ์การเคลื่อนที่ (Transport Coefficients), การตอบสนองต่อการรบกวนแบบที่ขึ้นกับเวลา (Time-dependt Response to Perturbation), และสเปกตรัม

ตัวอย่างของคุณสมบัติและปรากฏการณ์ของโมเลกุลหรือสสารที่เราสามารถใช้การจำลอง MD เพื่อศึกษาได้นั้นมีดังต่อไปนี้

**เคมี (Chemistry)**

- อันตรกิริยาภายในและภายนอกโมเลกุล (Intra- and Intermolecular Interactions)
- ปฏิกิริยาเคมี (Chemical Reactions)
- การเปลี่ยนเฟส (Phase Transitions)
- การคำนวณพลังงานอิสระ (Free Energy Calculations)

**วัสดุศาสตร์ (Materials Science)**

- เทอร์โมไดนามิกส์ที่สภาวะสมดุล (Equilibrium Thermodynamics)
- การเปลี่ยนเฟส (Phase Transitions)
- Properties of Lattice Defects
- Nucleation and Surface Growth
- กระบวนการความร้อนและความ (Heat/Pressure Processing)
- Ion Implantation
- Properties of Nanostructures

**ชีวเชิงฟิสิกส์และชีวเคมี (Biophysics and Biochemistry)**

- การพับของโปรตีน (Protein Folding)
- การทำนายโครงสร้างของโปรตีน (Protein Structure Prediction)
- การเข้ากันได้เชิงชีวะ (Biocombatibility) เช่น Cell Wall Penetration หรือ Chemical Processes
- การจำลองการจับกันของโมเลกุล (Molecular Docking)

**การแพทย์ (Medicine)**

- การออกแบบโมเลกุลยา (Drug Design)
- การค้นหาโมเลกุลยา (Drug Discovery)

## 2.2 ประวัติศาสตร์ของ Molecular Dynamics

Molecular Dynamics หรือ MD นั้นเป็นสาขาหนึ่งของเคมีทฤษฎีที่มีการค้นคว้าและวิจัยมาอย่างยาวนาน ผมสรุปไทม์ไลน์เรียงตามเหตุการณ์ที่เกิดขึ้นในวงวิชาการตั้งแต่อดีตในช่วงยุคแรก ๆ ของการพัฒนาวิธี MD จนถึงปัจจุบันตามนี้ครับ

- 1953: Nicholas Metropolis และคณะได้ตีพิมพ์บทความวิจัยเรื่อง “Equation of State Calculations by Fast Computing Machines”[8] โดยบทความนี้เป็นเสมือนจุดเริ่มต้นของไอเดีย MD เลยก็ว่าได้ โดยเป็นครั้งแรกที่ได้มีการประยุกต์ใช้เทคนิค Monte Carlo เพื่อแก้สมการที่อธิบายคุณสมบัติเชิงกายภาพของระบบที่ประกอบไปด้วยโมเลกุลที่มีอันตรกิริยาต่อกัน โดยขั้นแรกคือสร้างเซตของตัวเลขสุ่ม (Random Number) เพื่อใช้เป็นตัวแทนของ Conformational Space แล้วก็ใช้ค่าของพลังงานเป็นตัวระบุความน่าจะเป็นของสถานะของระบบที่ศึกษา
- 1956: Berni J. Alder และ Thomas E. Wainwright ได้ตีพิมพ์บทความเรื่อง “Phase Transition for a Hard Sphere System”[9] ซึ่งถือได้ว่าเป็นงานวิจัยที่เป็นจุดเริ่มต้นของ MD เลยก็ว่าได้

- 1958: เป็นครั้งแรกที่นักวิทยาศาสตร์ค้นพบโครงสร้างสามมิติของโปรตีนได้โดยใช้เทคนิค X-ray โดยเผยแพร่ในบทความ “A Three-Dimensional Model of the Myoglobin Molecule Obtained by X-Ray Analysis”[10]
- 1964: บทความวิจัยเรื่อง “Correlations in the Motion of Atoms in Liquid Argon”[11] โดย Aneesur Rahman ซึ่งเป็นผู้ที่ใช้ MD ในการคำนวณระบบของ Liquid Argon ซึ่งระบบที่ศึกษาตอนนั้นมี Argon ทั้งหมด 864 อะตอม โดยคำนวณด้วยซุปเปอร์คอมพิวเตอร์ CDC 3600 โดยใช้ Lennard-Jones Potential นอกจากนี้ Aneesur Rahman ได้รับการยอบรับว่าเป็นบิดาแห่งพลวัตเชิงโมเลกุลอีกด้วย (The Father of Molecular Dynamics)
- 1971: Aneesur Rahman และ Frank H. Stillinger ได้ตีพิมพ์บทความเรื่อง “Molecular Dynamics Study of Liquid Water”[12] ซึ่งเป็นใช้ MD ในการจำลองระบบโมเลกุลน้ำที่มีจำนวนโมเลกุลคือ 216 โมเลกุล
- 1975: Michael Levitt และ Arich Warshel ได้เผยแพร่บทความวิจัยเรื่อง “Computer Simulation of Protein Folding”[13] ซึ่งเป็นครั้งแรกที่มีการนำเทคนิค MD มาใช้ในการจำลองการพับของโปรตีน โดยเป็นการศึกษาการพับของ Bovine Pancreatic Trypsin Inhibitor (BPTI) จากโครงสร้างที่เป็นแบบสายเปิด
- 1979: David A. Case และ Martin Karplus ได้จำลองโปรตีนที่มีลิแกนด์เป็นโมเลกุลที่เข้าไปจับกับโปรแกรมด้วยเป็นครั้งแรก โดยได้ตีพิมพ์งานวิจัยเรื่อง “Dynamics of ligand binding to heme protein”[14]
- 1980s: ในช่วงต้น ๆ ทศวรรษ 1980 นั้นเป็นช่วงที่มีการศึกษาชีวโมเลกุลด้วยการจำลอง MD เป็นจำนวนมาก รวมไปถึงมีการคำนวณ Free Energy ด้วย
- 1985: Roberto Car Michele Parrinello ได้พัฒนาเทคนิค Car-Parrinello Molecular Dynamics (CPMD) ซึ่งเสนอในบทความเรื่อง “Unified Approach for Molecular Dynamics and Density-Functional Theory”[15] โดยเป็นการนำเทคนิค Density Functional Theory มารวมกับ Born-Oppenheimer Molecular Dynamics
- 1988: Michael Levitt และ Ruth Sharon ได้คำนวณระบบของโปรตีนที่มีโมเลกุลน้ำเป็นตัวทำละลายและนำเสนอในบทความเรื่อง “Accurate Simulation of Protein Dynamics in Solution”[16]
- 1990s: ในช่วงต้น ๆ ทศวรรษ 1990 นั้นก็ได้มีการพัฒนาศักย์ (Potential) ที่ใช้ในวิธี MD รวมถึงเทคนิคการเพิ่มประสิทธิภาพในการสุ่ม (Enhanced Sampling) อย่างต่อเนื่อง

## 2.3 สนามแรง

สนามแรง (Force Field) เป็นสมการคณิตศาสตร์ที่อธิบายพื้นผิวพลังงานศักย์ของโมเลกุลได้โดย Force Field นั้นจะเป็นผลรวมของเทอมพลังงานต่าง ๆ ที่อ้างอิงอยู่กับอะตอมและมีพารามิเตอร์ที่สอดคล้องกับโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลซึ่งเทอมพลังงานแต่ละเทอมนั้นจะมีการตีความทางกายภาพ (Physi-

cal Interpretation) แตกต่างกันไป ไอเดียเริ่มต้นของ Force Field นั้นก็คือในการจำลอง MD นั้นเราจำเป็นจะต้องคำนวณแรง (Force) ระหว่างอะตอมแต่ละคู่ของทุกอะตอมในระบบของเราซึ่งอาจจะมีมากถึงหลักพันหรือหลักหมื่นอะตอมเลยทีเดียว โดยทั่วไปแล้ว Time-step ที่เรามักจะใช้ในการจำลอง MD นั้นคือ 1 fs ถ้าหากเราต้องการรัน MD เป็นระยะเวลา 100 ns เราจำเป็นจะต้องคำนวณแรงระหว่างอะตอมทั้งหมดประมาณ $10^7$-$10^8$ ครั้งเลยทีเดียว สำหรับ Force Field มาตรฐานของพื้นผิวพลังงานศักย์ของโมเลกุล (Potential Energy Surface หรือ PES) นั้นมีหน้าตาประมาณนี้

$$V\left(R^{3N}\right) = \underbrace{\sum_{i}^{\text{bonds}} \frac{k_i}{2}(l_i - l_{i,0})^2}_{\text{Bond Stretches}} + \underbrace{\sum_{i}^{\text{angles}} \frac{k_i}{2}(\theta_i - \theta_{i,0})^2}_{\text{Angle Bends}} + \underbrace{\sum_{i}^{\text{torsions}} \frac{V_i}{2}(1 + \cos(n_i\omega_i - \gamma_i))}_{\text{Torsional Motion}} + \underbrace{\sum_{i,j>i} \underbrace{4\epsilon_{i,j}\left(\left(\frac{\sigma_{ij}}{R_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{R_{ij}}\right)^{6}\right)}_{\text{Lennard-Jones Term}} + \underbrace{\frac{q_i q_j}{4\pi\epsilon_0 R_{ij}}}_{\text{Coulomb Term}}}_{\text{Intermolecular Interactions}} \tag{2.3.1}$$

โดยที่พลังงานแต่ละเทอมนั้นมีตัวแปรที่เกี่ยวข้องกับลักษณะเชิงเรขาคณิตของโมเลกุล เช่น ความยาวพันธะ $l_i$, มุมพันธะ $\theta_i$, มุมบิด $\omega_i$, และระยะห่างระหว่างอะตอม $R_{ij}$ นอกจากนี้พลังงานแต่ละเทอมนั้นยังมีพารามิเตอร์ที่ขึ้นอยู่กับชนิดของอะตอมด้วย เช่น เทอมที่เป็นพลังงานสำหรับการยืดหดของพันธะ (Bond Strething) นั้นจะมีค่าคงที่แรง (Force Constant) $k_i$ และค่าความพันธะที่สภาวะสมดุล (Equilibrium Bond Length) $l_{i,0}$

1. 3 เทอมแรกคือ Bonded เป็นพลังงานที่มาจากภายในของโมเลกุลเอง

2. เทอมที่ 4 คือ Non-bonded ที่อธิบาย Electrostatic Interaction มีชื่อเรียกว่า Coulomb Interaction

3. เทอมที่ 5 คือ Non-bonded ที่อธิบายพลังงาน Non-electrostatic ที่เกิดจาก Dipole-dipole Interaction (เช่น London Force ที่อธิบาย Interaction ระหว่าง Non-polar Molecules เป็นต้น) มีชื่อเรียกว่า Lennard-Jones Potential (หรือเรียกสั้น ๆ ว่า LJ Potential หรือ 12-6 Potential)

ตอนนี้ผู้อ่านคงกำลังคิดว่าพารามิเตอร์ชุดนี้นั้นจะต้องมีค่าเพียงแค่ค่าเดียวสำหรับอะตอมที่เป็นธาตุเดียวกันแต่ว่าในความเป็นจริงกลับไม่ใช่อย่างนั้นเพราะว่าถึงแม้ว่าจะเป็นธาตุชนิดเดียวกันแต่ก็ขึ้นอยู่กับสภาพวาดล้อม (Environment) รอบ ๆ ธาตุนั้นด้วย ยกตัวอย่างให้เข้าใจง่ายคือสมมติว่าเรามีอะตอมคาร์บอนในหมู่เมทิล (Mehtyl Group, $-CH_3$) และกับอะตอมคาร์บอนในหมู่คาร์บอนิล (Carbonyl Group, C=O) นั้นจะมี Characteristics ต่างกันดังนั้นจึงทำให้มีคุณสมบัติที่แตกต่างกัน เช่น ประจุของอะตอม ดังนั้นสำหรับ Force Field ที่ดีนั้นควรจะต้องมีชุดเซตของพารามิเตอร์ที่แตกต่างกันไปสำหรับโมเลกุลหรือระบบที่ต้องการ

ศึกษา เช่น โปรตีน, โลหะออกไซด์, พอลิเมอร์, หรือคริสตัลไอออนิค

**สำหรับ Quantum calculation** ในการศึกษาคุณสมบัติของโมเลกุล (Molecular properties) นั้น Properties หลาย ๆ ตัวนั้นเป็นญาติกับพลังงาน ก็คือถ้าเรารู้พลังงานของโมเลกุล เราก็จะสามารถคำนวณ Properties อื่น ๆ ตามมาได้ (ในรูปของอนุพันธ์เทียบกับ Perturbation อะไรก็ว่าไป)

**สำหรับ Molecular dynamics** สำหรับ Molecular Dynamics: เราใช้ Force Field ในการคำนวณหาพลังงานของโมเลกุล เพื่อนำพลังงานมาคำนวณหาแรง (Force) ที่กระทำต่ออะตอมแต่ละตัว

ในการสร้าง Force Field สักอันหนึ่งขึ้นมานั้นเราจะต้องมีการกำหนดว่าเราจะมีการใช้เทอมพลังงานอะไรบ้างสำหรับ Force Field ของเราแล้วก็รวมถึงว่าเราจะมีวิธีการในการคำนวณหาค่าของพารามิเตอร์ใน Force Field สำหรับธาตุแต่ละธาตุอย่างไร ในยุคแรก ๆ ของเคมีเชิงคำนวณนั้นนักวิจัยมักจะใช้ผลการทดลองนั้นนำมาเทียบหาค่าของพารามิเตอร์ของ Force Field (Parameter Fitting) ซึ่ง Force Field ประเภทนี้จะมีชื่อเรียกว่า *Empirical* Force Field

การนำ Force Field ไปใช้ในการจำลอง Molecular Dynamics นั้นมีขั้นตอนคร่าว ๆ ดังนี้

1. นำแรงต่ออะตอมมาคำนวณหาความเร่ง (Acceleration) แล้วนำไปเข้าสมการ Equation of Motions เพื่อคำนวณหาความเร็ว (Velocity) และการกระจัด (Displacement) ที่เปลี่ยนไป

2. นำการกระจัดที่เปลี่ยนไปมาทำการอัพเดทตำแหน่งของอะตอม/โมเลกุล เรียกวิธีการนี้ว่า Propagation

3. เมื่อเราทำแบบนี้ไปเรื่อย ๆ เราจะได้ Dynamic ของระบบที่สามารถที่จะ Represent คุณสมบัติ Microscopic ของระบบจริง ๆ ได้

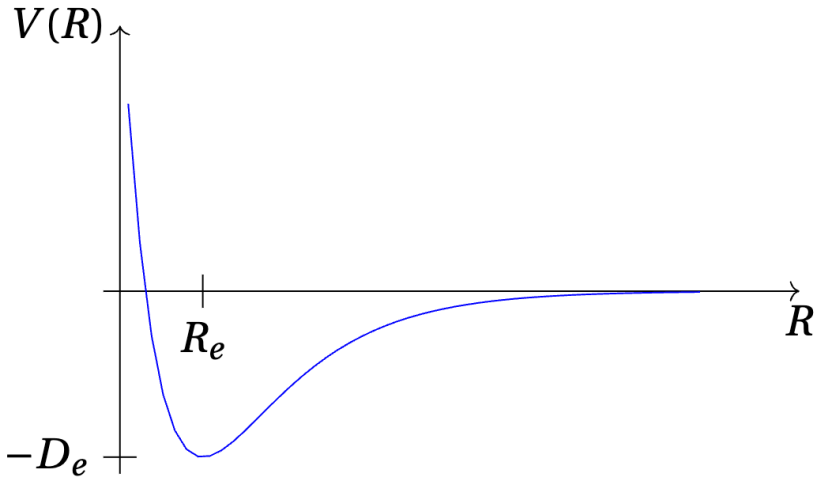
โดยขนาดของระบบที่เราใช้ในการจำลองจะสอดคล้องกับ Time-scale ของการรัน Dynamic Simulation ในการศึกษา Properties ที่แตกต่างกันออกไป นอกจากนี้ยังมีวิธีการคำนวณ Force field ที่ซับซ้อนกว่านี้อีกมากมาย ขึ้นอยู่กับวิธีการ accuracy ที่ต้องการ สรุปคือ Force field นั้นสำคัญมาก ๆ เพราะเป็นจุดเริ่มต้นของการศึกษา Properties อื่น ๆ อีกมากมายของโมเลกุล ดังนั้น *เลือก Force Field ไม่ดี = ชีวิตพัง*

## 2.4 สนามแรงสำหรับพันธะโควาเลนท์

ในหัวข้อนี้เราจะมาดูรายละเอียดของ Force Field ที่สำคัญมากที่สุดอันหนึ่งนั่นก็คือ Force Field ที่ใช้ในการอธิบายพันธะโควาเลนท์ (Covalent Bonding) ซึ่งประกอบไปด้วย การยืดหดของพันธะ (Bond Stretching), การงอของพันธะ (Angle Bending) และการเคลื่อนแบบบิด (Torsional Motion)

### 2.4.1 Bond Stretching

เริ่มต้นด้วยการพิจารณาพลังงานศักย์ $V(R)$ ของโมเลกุลอะตอมคู่ (Diatomic Molecule) ซึ่งเป็นฟังก์ชันของระยะห่างระหว่างอะตอม (Bond Distance) $R$ โดยเราสามารถแสดง (Represent) $V(R)$ อันนี้ได้ด้วยศักย์ของมอส (Morse Potential) ดังต่อไปนี้




ภาพ 2.1 Morse Potential

$$V(R) = D_e \left(e^{-2a(R-R_e)} - 2e^{-a(R-R_e)}\right) \tag{2.4.1}$$

โดยที่ $R_e$ คือระยะห่างระหว่างอะตอมที่สภาวะสมดุล เช่น ความยาวพันธะ ณ ตำแหน่งที่พลังงานศักย์ของ Morse Potentail นั้นมีค่าน้อยที่สุด, $D_e$ คือความลึก (Depth) ของพื้นผิวศักย์ (Potentail Surface) ซึ่งก็คือพลังงานการแตกออกหรือการแยกตัว (Dissociation Energy) และ $a$ คือพารามิเตอร์ที่อธิบายความกว้างของบ่อพลังงานศักย์อันนี้ นอกจากนี้ Morse Potential สามารถถูกเขียนได้ด้วยวิธีอื่น ๆ ได้อีกด้วย

หนึ่งในวิธีที่เราจะสามารถใช้ในการ Represent พันธะโควาเลนท์ก็คือการใช้โมเดลการสั่น Harmonic OSciallator แบบคลาสสิค เช่น ถ้าเรามีอะตอม 2 อะตอมที่ถูกยึดเข้าด้วยกันด้วยสปริงที่มี Force Constant $k$ เราสามารถใช้ Taylor Expansion ในการอธิบาย Potential Energy $V(R)$ ได้โดยการใช้ Dunham Expansion Parameter $Q = (\frac{R-R_e}{R_e})$ ดังนี้

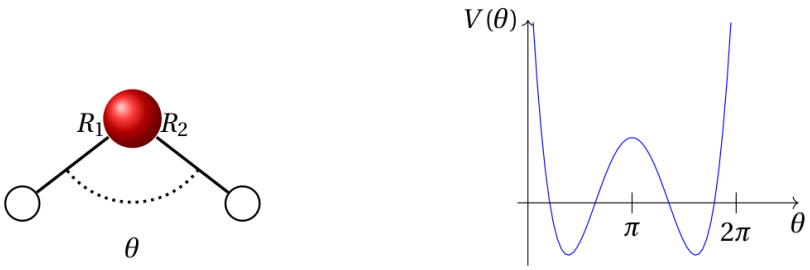
$$V(Q) = \underbrace{V(0)}_{\text{Zero level}} + \underbrace{V'(0)Q}_{V'(0)=0 \text{ in the minimum}} + \underbrace{\frac{1}{2}V''(0)Q^2}_{\text{Harmonic term}} + \underbrace{\frac{1}{6}V^{(3)}(0)Q^3}_{\text{Anharmonicity}} + \underbrace{\frac{1}{24}V^{(4)}(0)Q^4 \ldots,}_{\text{Quartic term}} \tag{2.4.2}$$

โดยที่เราไม่ต้องพิจารณา Zero Level ($V(0)$) ก็ได้ เพราะว่า Potential Energy Surface นั้นสามารถเปลี่ยนระดับพลังงานได้ด้วยค่าพลังงานคงที่ สำหรับเทอมที่เป็นเส้นตรง (Linear Term) ใน $Q$ นั้นมีค่าเท่ากับ 0 เนื่องจากว่า Gradient ของเทอมนี้นั้นเท่ากับ 0 ที่ตำแหน่ง Minimum ส่วนเทอมที่เป็น Quadratic Term กับ Cubic Term ใน $Q$ นั้นคือ Harmonic และ Anharmonic ของพื้นผิวศักย์ตามลำดับ ถ้าหากว่าเราตัดเทอม Anharmonicity แล้วก็เทอมที่มีอันดับสูงกว่านี้ออกไปจาก Tarlor Expansion สิ่งที่เราจะได้นั้นก็คือการสั่นฮาร์โมนิคแบบดั้งเดิม (Classical Harmonic Oscillator) นั่นเอง

$$V(Q) = \frac{k}{2}Q^2 \quad \text{โดยที่} \quad k \equiv V''(0) \tag{2.4.3}$$

Taylor Expansion ของ Morse Potential นั้นเป็นหนึ่งโมเดลของพื้นผิวศักย์ที่สามารถอธิบายสถานะของระบบรอบ ๆ จุดต่ำสุด Minimum ได้ ซึ่งจะอธิบายได้ดีก็ต่อเมื่อเราทำการตัดหรือไม่พิจารณาเทอมที่มีอันดับสูงกว่า Harmonic Term ออกไป อย่างไรก็ตาม โมเดลนี้ก็มีจุดอ่อน ถ้าหากว่าเราดูกรณีที่อะตอมทั้งสองอะตอมนั้นมีระยะห่างกันมาก ๆ ซึ่งก็คือห่างกันอนันต์ $Q \to \infty$ จะทำให้พลังงานศักย์นั้นเข้าใกล้ค่าอนันต์ด้วย $V(Q) \to \infty$ ซึ่งเงื่อนไขอันนี้ทำให้โมเดล Morse Potential นั้นอธิบาย Dissociation ได้ไม่ถูกต้อง แต่ถ้าหากว่าโมเดลอันนี้ถูกนำมาใช้ในการอธิบายระบบที่มีหลาย ๆ โมเลกุลและแต่ละโมเลกุลนั้นไม่ทำปฏิกิริยาต่อกัน พูดง่าย ๆ ก็คือไม่มีการสร้างพันธะ (Non-Reacting) เราจะพบว่าการที่เราทำการตัดเทอมที่มีอันดับสูงกว่า Harmomic ออกไปนั้นจะทำให้มันสามารถอธิบายพื้นผิวศักย์ได้อย่างสมเหตุสมผล

### 2.4.2 Angle Bending




ภาพ 2.2 ซ้าย: โมเลกุลน้ำ, ขวา: Double Minimum Potential สำหรับมุมพันธะของโมเลกุลน้ำ

สำหรับการอธิบาย Angle Bending นั้นผมขอยกตัวอย่างโมเลกุลน้ำ $H_2O$ ซึ่งมีมุมสมดุลคือ $\theta_e$ ระหว่างพันธะ O—H ซึ่งมีความยาวพันธะเท่ากับ 104.5 องศา เมื่อความยาวพันธะยืดออกนั้นเราสามารถใช้การประมาณแบบ Harmonic Oscillator เพื่ออธิบาย Angle Bending ได้ ดังนี้

$$V(\theta) = \frac{k_\theta}{2}(\theta - \theta_e)^2 \tag{2.4.4}$$

ซึ่ง Approximation ด้านบนนี้สามารถนำมาใช้ได้เมื่อมุม $\theta$ นั้นเข้าใกล้กับมุมสมดุล $\theta_e$

ถ้าหากเราลองมาดูกรณีแปลก ๆ เช่น ถ้ามุมพันธะมีค่าเท่ากับ $\pi$ ซึ่งทำให้โมเลกุลน้ำนั้นเป็นเส้นตรง เราจะพบว่าพลังงานศักย์นั้นจะสูงมาก ๆ และในความเป็นจริงนั้นแทบจะเป็นไปได้ยากมาก ๆ ที่โมเลกุลน้ำนั้นจะเป็นเส้นตรง (แต่ก็สามารถทำได้โดยการเพิ่มอุณหภูมิให้สูงมาก ๆ)

### 2.4.3 Dihedral Terms

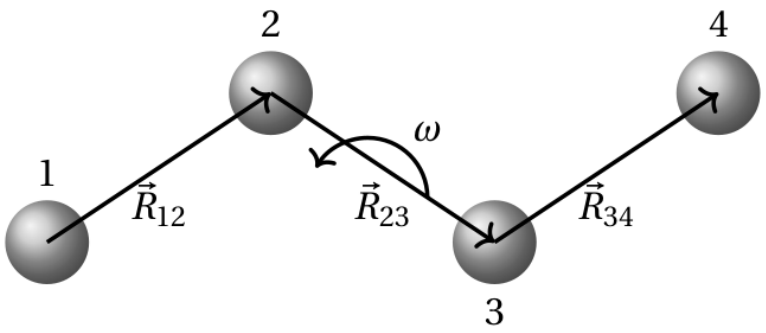
พารามิเตอร์อีกอันหนึ่งที่สำคัญมากในการอธิบายการเปลี่ยนแปลงของมุมพันธะและระนาบในโมเลกุลนั้นก็คือ Dihedral Angle หรือว่ามุมบิด ซึ่งมุมบิดนี้ก็มี Potential Energy เป็นของตัวเองด้วยโดยมีสมการดังต่อไปนี้

$$V(\omega) = \sum_n \frac{V_n}{2} \left(1 + \cos(n\omega - \gamma)\right) \tag{2.4.5}$$

โดยที่ $n$ นั้นคือเลขจำนวนเต็มที่บ่งบอกถึงจำนวนคาบ เช่น $n = 2$ ก็คือมีคาบเท่ากับ 180 องศา หรือ $n = 3$ ก็คือมีคาบเท่ากับ 120 องศานั่นเอง, $V_n$ คือพลังงานศักย์การหมุน (Rotational Energy Barrier) และ $\gamma$ นั้นกำหนดว่ามุมที่มุมบิดนั้นมีค่าเท่ากับ 0 องศา

สมการที่ใช้ในการพลังงานศักย์ของ Dihedral Angle นั้นสามารถเขียนได้โดยการใช้ส่วนจริง (Real Part) ของการแปลงฟูเรียร์ ดังนี้

$$V(\omega) = \sum_n C_n \cos(\omega)^n \tag{2.4.6}$$




ภาพ 2.3 ซ้าย: โมเลกุลน้ำ, ขวา: Double Minimum Potential สำหรับมุมพันธะของโมเลกุลน้ำ

สำหรับโมเลกุลที่ควรจะต้องแบนราบหรือมีความเป็น Planar อยู่แล้วนั้น บางครั้งเราก็อยากที่จะเพิ่ม Constraint เข้าไปให้กับพื้นผิวศักย์ของโมเลกุลเพื่อทำให้โมเลกุลนั้นมีความเป็น Planar ซึ่งวิธีการเพิ่ม Constraint นั้นทำได้หลายวิธี หนึ่งในนั้นก็คือการใส่ Lagrangian Multipliers ในขณะที่เราทำการปรับค่าพลังงานของโมเลกุลให้มีค่าต่ำที่สุดหรือที่เราเรียกว่า Constrained Energy Minimization ซึ่งจะทำให้เราได้โมเลกุลที่มีพลังงานต่ำที่สุดที่สภาวะที่ถูก Constraint อยู่ด้วย จึงทำให้โมเลกุลนั้นถูกบังคับให้แบนราบหรือมีความเป็น Planar ตลอดเวลา นอกจากนี้ยังมีอีกวิธีก็คือการเติมเทอมพลังงานพิเศษเข้าไปอื่ ๆ เลยอีกเทอมนึง ซึ่งเทอมพลังงานพิเศษอันนี้ที่เราเติมเข้าไปนั้นมีชื่อเรียกว่า Energy Restraint หรือ Improper Torsion Term นั่นเอง ซึ่งมีหน้าตาดังนี้

$$V(\omega) = k_\omega(1 - \cos 2\omega) \tag{2.4.7}$$

โดยที่โมเลกุลนั้นจะถูกบังคับหรือถูกตรึงให้มีลักษณะที่ *เกือบ* จะเป็นแบนราบอยู่ตลอดเวลาถ้าหากว่าค่า $k_\omega$ นั้นมีค่ามากพอ

### 2.4.4 Cross Terms

เทอมสุดท้ายที่เรามักจะไม่ค่อยคุนชินกันเท่าไหร่เพราะว่ามักจะไม่ค่อยมีใครพูดถึงแม้แต่ในตำราต่างประเทศหลาย ๆ เล่มนั่นก็คือเทอมที่เรียกว่า Cross Term ซึ่งเทอมนี้จะถูกนำใส่เข้าไปใน Potential Energy Expression เพื่อใช้อธิบายการคู่ควบกัน (Coupling) ระหว่าง Two-bond Stretch หรือ Bond Stretch กับ Angle Bending Term โดยผมขอยกตัวอย่างด้วยโมเลกุลน้ำเหมือนเดิมครับ ถ้าหากเราพิจารณา Intermolecular Motion นั้นเราจะสามารถอธิบาย Motion อันนี้ได้โดยการใช้ Taylor Expansion รอบ ๆ ความยาวพันธะทั้งสองอัน ($R_1$ กับ $R_2$) และมุมพันธะ ($\theta$) ดังนี้

$$
\begin{aligned}
V\left(R_1, R_2, \theta\right) =& V\left(R_{1,0}, R_{2,0}, \theta_0\right) + \left(R_1 - R_{1,0}\right) \left.\frac{\partial V}{\partial R_1}\right|_{R_{1,0},R_{2,0},\theta_0} \\
&+ \left(R_2 - R_{2,0}\right) \left.\frac{\partial V}{\partial R_2}\right|_{R_{1,0},R_{2,0},\theta_0} \\
&+ \left(\theta - \theta_0\right) \left.\frac{\partial V}{\partial \theta}\right|_{R_{1,0},R_{2,0},\theta_0} + \frac{1}{2}\left(R_1 - R_{1,0}\right)^2 \left.\frac{\partial^2 V}{\partial R_1^2}\right|_{R_{1,0},R_{2,0},\theta_0} \\
&+ \frac{1}{2}\left(R_2 - R_{2,0}\right)^2 \left.\frac{\partial^2 V}{\partial R_2^2}\right|_{R_{1,0},R_{2,0},\theta_0} + \frac{1}{2}\left(\theta - \theta_0\right)^2 \left.\frac{\partial^2 V}{\partial \theta^2}\right|_{R_{1,0},R_{2,0},\theta_0} \\
&+ \left(R_1 - R_{1,0}\right)\left(R_2 - R_{2,0}\right) \left.\frac{\partial^2 V}{\partial R_1 \partial R_2}\right|_{R_{1,0},R_{2,0},\theta_0} \\
&+ \left(R_1 - R_{1,0}\right)\left(\theta - \theta_0\right) \left.\frac{\partial^2 V}{\partial R_1 \partial \theta}\right|_{R_{1,0},R_{2,0},\theta_0} \\
&+ \left(R_2 - R_{2,0}\right)\left(\theta - \theta_0\right) \left.\frac{\partial^2 V}{\partial R_2 \partial \theta}\right|_{R_{1,0},R_{2,0},\theta_0} + \ldots
\end{aligned} \tag{2.4.8}
$$

โดยที่ 3 เทอมสุดท้ายนั้นคือ Coupling Terms อย่างไรก็ตาม โดยปกติแล้วเราไม่จำเป็นต้องใส่เทอม Coupling ทั้งหมดที่เรามี ซึ่งเทอม Coupling บางเทอมก็สำคัญ บางเทอมก็ไม่สำคัญซึ่งขึ้นอยู่กับโมเลกุล สำหรับโมเลกุลน้ำนั้นเทอม Coupling ที่สำคัญและควรจะต้องมีก็คือ Coupling Term ที่มาจาก Stretch Bond เพื่อที่ว่าจะสามารถอธิบาย Symmetric Stretch Coordinate ได้ ซึ่งมีสมการดังนี้

$$
Q_1 - Q_{1,0} = \left(R_1 - R_{1,0}\right) + \left(R_2 - R_{2,0}\right) \tag{2.4.9}
$$

และสามารถ Antisymmetric Stretch Coordinate ได้เช่นกัน ซึ่งมีสมการดังนี้

$$
Q_2 - Q_{2,0} = \left(R_1 - R_{1,0}\right) - \left(R_2 - R_{2,0}\right) \tag{2.4.10}
$$

แล้วก็ในการรวม Crossing Term เข้าไปในสมการพลังงานศักย์เพื่อใช้อธิบายพื้นผิวศักย์ของโมเลกุลน้ำนั้น เราจะพบว่าพารามิเตอร์ที่ขึ้นอยู่กับชนิดของอะตอมนั้น (เช่น $R_{1,0}$, $R_{2,0}$ และ $\theta_0$) ไม่ได้เกี่ยวข้องหรือสอดคล้องกับ Equilibrium Geometry เลย

### 2.4.5 สรุป Bonding

โดยสรุปแล้วเราเพิ่งได้ศึกษาเทอมของพลังงานต่าง ๆ ที่เรานำมาใช้ในการสร้าง Force Field เพื่อใช้ในการอธิบาย Covalent Bond ซึ่งประกอบไปด้วยเทอมดังต่อไปนี้ Bond Stretching, Angle Bending, Tor-

sional Motion แล้วก็มีการรวมเทอมพิเศษเข้าไปด้วยซึ่งก็คือ Coupling Term และนอกจากนี้ยังมีเทอมอื่น ๆ อีกที่ผมไม่ได้พูดถึง เช่น เทอมพิเศษที่อธิบาย Hyperconjugation ซึ่งเป็นปรากฏการณ์ที่ $\pi$-Conjugation นั้นส่งผลต่อการยืดหกของพันธะอย่างไรในโมเลกุล โดยเราสามารถแบ่งประเภทของเทอมเหล่านี้ออกเป็นคลาสได้ดังนี้

- Class I: มีเพียงแค่ Harmonic Terms เท่านั้น ไม่มีการเติม Coupling Terms เข้าไป
- Class II: มี Anharmonic Terms และ Cross Terms
- Class III: มีเทอมพื้นฐานทั้งหมดและมีการเพิ่มเทอมพิเศษ เช่น Huperconjugation เข้าไปด้วย

## 2.5 อันตรกิริยาระหว่างโมเลกุล

ในหัวข้อที่ 2.4 เราได้ศึกษาอันตรกิริยาที่อยู่ภายในโมเลกุลไปแล้วนั่นก็คือพันธะโควาเลนท์ซึ่งเป็นอันกิริยาที่เกิดขึ้นระหว่างอะตอม ในหัวข้อนี้ผู้อ่านจะได้ศึกษาอันตรกิริยาที่เกิดขึ้นระหว่างโมเลกุล เช่น อันตรกิริยาแบบอ่อน (Weak Interaction) ซึ่ง “อ่อน” ในที่นี้คือเทียบกับพันธะโควาเลนท์ ตัวอย่างเช่น Dispersion Interaction ที่เกิดขึ้นใน Liquid Argon, Hydrogen Bonding ที่เกิดขึ้นใน Liquid Water, หรือ Ion-Ion Interaction ที่เกิดขึ้นในสารละลายอิเล็กโทรไลต์ นอกจากนี้แล้วยังมีอันตรกิริยาอื่น ๆ ที่เกิดขึ้นระหว่างโมเลกุลที่เราจะต้องพิจารณาด้วย เช่น อันตรกิริยาแบบไกล (Long-Range Interaction) ที่สามารถเกิดขึ้นได้ภายในโมเลกุลเดียวกันสำหรับโมเลกุลที่มีขนาดใหญ่มาก ๆ ซึ่งพลังงานที่เกิดขึ้นจาก Long-Range Interaction นั้นก็เป็นอีกเทอมที่สำคัญมาก ๆ ที่ทำให้การจำลองโมเลกุลนั้นมีความถูกต้องมากขึ้น

ในหัวข้อนี้เราจะมาโฟกัสพลังงานทั้งหมด 4 เทอมที่สำคัญซึ่งเป็นเทอมพลังงานที่เป็นตัวแทนของ Intermolecular Interaction ได้เป็นอย่างดีครับ

1. พลังงานไฟฟ้าสถิตย์ (Electrostatic Energy): เป็นเทอมพลังงานที่อธิบายอันตรกิริยาระหว่างไอออนหรือโมเลกุลที่มีความมีขั้ว
2. พลังงานเหนี่ยวนำ (Induction Energy): เป็นเทอมพลังงานที่อธิบายถึงการเปลี่ยนแปลงของความหนาแน่นของอิเล็กตรอนภายในโมเลกุลที่เกิดจากการถูก Polarized ด้วยสนามไฟฟ้าจากโมเลกุลรอบ ๆ ซึ่งส่งผลให้เกิดการเหนี่ยวนำ Electric Moment เช่น Induced Dipole Moment
3. พลังงานผลักแบบใกล้ (Short-Range Repulsion Energy): เป็นเทอมพลังงานที่มาจากอันตรกิริยาแบบผลักระหว่างอิเล็กตรอนภายซึ่งถูกอธิบายด้วย Pauli Exclusion Principle
4. พลังงานแพร่กระจาย (Dispersion Energy): เป็นเทอมพลังงานที่อธิบายการ Correlation ของการเคลื่อนที่ของอิเล็กตรอน

### 2.5.1 Electrostatic Energy

พลังงานที่เกี่ยวข้องกับ Intermolecular Interaction อันแรกที่เราจะมาศึกษากันนั้นคือพลังงานไฟฟ้าสถิตย์ (Electrostatic Energy) ซึ่งถือว่าเป็นพลังงานพื้นฐานที่สุดเลยก็ว่าได้ โดยพลังงานไฟฟ้าสถิตย์นั้นเกิดขึ้นมาจากอันตรกิริยาทางไฟฟ้าระหว่างอะตอมที่มีประจุ (Charge) ภายในโมเลกุล ดังนั้นผมจะเริ่มด้วยการอธิบายเรื่องของประจุก่อนเพราะว่าแรงทางไฟฟ้านั้นเกี่ยวข้องโดยตรงกับประจุ ในทางเคมีควอนตัมนั้น เรากำหนดการกระจายตัวของประจุภายในโมเลกุลด้วยประจุของนิวเคลียส (Nuclear Charges) $\{Z_I\}$ (โดยที่ $I = 1, 2, \ldots, N$ และ $N$ คือจำนวนของอะตอม) และความหนาแน่นของอิเล็กตรอน $\rho(\vec{r})$ ซึ่งมีความสัมพันธ์กันดังนี้

$$\int \rho(\vec{r})\mathrm{d}\tau = n \tag{2.5.1}$$

โดยที่ $n$ คือจำนวนของอิเล็กตรอนของโมเลกุล สำหรับ Force Field นั้น วิธีที่ง่ายและตรงไปตรงมาที่สุดที่ใช้ในการแสดงถึงการกระจายตัวของประจุภายในโมเลกุลนั้นคือใช้เซตของประจุเชิงอะตอม (Atomic Charges) $\{q_I, I = 1, 2, \ldots, N\}$ กับกฎของคูลอมป์ (Coulomb's Law) สำหรับการอธิบายอันตรกิริยาระหว่างประจุเชิงอะตอม ดังนี้

$$V = \sum_{I=1}^{N} \sum_{J=I+1}^{N} \frac{q_I q_J}{4\pi\varepsilon_0 R_{IJ}}, \tag{2.5.2}$$

โดยที่ $R_{IJ}$ คือระยะห่างระหว่างอะตอม $I$ กับอะตอม $J$ สำหรับการคำนวณของประจุเชิงอะตอมนั้นง่ายมาก โดยเราก็แค่ทำการนำความหนาแน่นของอิเล็กตรอนของอะตอมที่เราสนใจในโมเลกุลมารวมกับประจุของนิวเคลียสของอะตอมนั้น แต่ปัญหาก็คือว่าเราไม่รู้ว่าเราจะทำการแบ่งโมเลกุล (ซึ่งถูกอธิบายความหนาแน่นของอิเล็กตรอน) ออกเป็นชิ้น ๆ อย่างไรเพื่อทำการกำหนดขอบเขตในการคำนวณการประจายของประจุ

เนื่องจากว่าเราไม่มีคำจำกัดความที่แน่นอนสำหรับประจุเชิงอะตอมเนื่องจากว่าประจุนั้นไม่ใช่ปริมาณที่สามารถวัดได้แม้แต่ในทางทดลอง (Non-Observable) ดังนั้นในปัจจุบันนี้เรามีทฤษฎีเป็นสิบ ๆ ร้อย ๆ ทฤษฎีเลยที่ถูกพัฒนาขึ้นมาเพื่อนิยามและคำนวณประจุเชิงอะตอม

ผมขอเริ่มต้นด้วยเหตุผลที่ว่าประจุเชิงอะตอมนั้นควรที่จะต้อง Reproduce ค่าของโมเมนต์เชิงไฟฟ้าของโมเลกุลได้ (Molecular Electric Moments) ดังนั้นเราจึงอ้างได้ว่าโมเลกุลนั้นมีประจุรวมทั้งเป็น

$$q^{\mathrm{mol}} = \sum_{I=1}^{N} q_I \tag{2.5.3}$$

สำหรับโมเลกุลที่เป็นกลางหรือประจุเท่ากับศูนย์นั้น $(q^{\mathrm{mol}} = 0)$ ถึงแม้ว่าค่าความคลาดเคลื่อนที่มาจากผลต่างระหว่างของค่าประจุที่เบี่ยงเบนออกจากศูนย์นั้นจะมีน้อย แต่ว่ามันจะทำให้เกิดค่าความคลาดเคลื่อนของค่าพลังงานเชิงไฟฟ้าสถิตย์ที่เยอะมาก ๆ ได้เช่นกัน สาเหตุก็เพราะว่าค่าระยะห่าง $(1/R)$ นั้นขึ้นอยู่กับอันตรกิริยาระหว่างประจุนั่นเอง (Charge-Charge Interactions) สำหรับโมเลกุลที่มีขนาดเล็กนั้นเราสามารถคำนวณค่าไดโพลโมเมนต์เชิงโมเลกุล (Molecular Dipole Moment) ได้ดังนี้

$$\mu_{\alpha}^{\mathrm{mol}} = \sum_{I=1}^{N} q_I R_{I,\alpha}, \tag{2.5.4}$$

และคำนวณควอนรูโพลโมเมนต์เชิงโมเลกุล (Molecular Quadrupole Moment)

$$\Theta_{\alpha\beta}^{\mathrm{mol}} = \sum_{I=1}^{N} q_I \left( \frac{3}{2} R_{I,\alpha} R_{I,\beta} - \frac{1}{2} R_{I,\gamma} R_{I,\gamma} \delta_{\alpha\beta} \right) \tag{2.5.5}$$

ซึ่งโมเมนต์ทั้งสองอันนี้ก็สอดคล้องกับค่าประจุเชิงอะตอมนั่นเอง $q_I$

สำหรับการใช้ประจุเชิงอะตอมมาอธิบายการกระจายของประจุภายในโมเลกุลนั้น เรายังมีเทคนิคอื่น ๆ อีกเยอะเลยที่ช่วยทำให้โมเดลโมเดลประจุเชิงอะตอมนั้นมีความถูกต้องและแม่นยำมากยิ่งขึ้น หนึ่งในวิธีที่ง่ายและตรงไปตรงมาก็คือการเพิ่มเทอมไดโพลโมเมนต์เชิงอะตอม (Atomic Dipole Moment, $\mu_{I,\alpha}$) และควอดรูโพลโมเมนต์เชิงอะตอม (Atomic Quadrupole Moment, $Q_{I,\alpha\beta}$) เข้าไป ซึ่งเราจะได้ว่าโมเมนต์เชิงโมเลกุลที่ได้จากการเติมโมเมนต์เชิงอะตอมเข้าไปนั้น มีดังนี้

**Molecular Dipole Moment**

$$\mu^{\mathtt{mol}_{\alpha}} = \sum_{I=1}^{N} q_I R_{I,\alpha} + \mu_{I,alpha} \tag{2.5.6}$$

**Molecular Quadrupole Moment**

$$Q^{\mathtt{mol}_{\alpha\beta}} = \sum_{I=1}^{N} q_I R_{I,\alpha} R_{I,\beta} + q_{I,\alpha} R_{I,\beta} + R_{I,\alpha} \mu_{I,\beta} + Q_{I,\alpha\beta} \tag{2.5.7}$$

ซึ่งเราสามารถนำสมการที่ (2.5.7) มาใช้ในการคำนวณหา Quadrupole Moment ได้โดยใช้สมการดังต่อไปนี้

$$\Theta_{\alpha\beta} = \frac{3}{2} Q_{\alpha\beta} - \frac{1}{2} Q_{\gamma\gamma} \delta_{\alpha\beta} \tag{2.5.8}$$

นอกจากนี้ยังมีกรณีพิเศษอีกบางกรณีที่เราจำเป็นที่จะต้องนำมาพิจารณาเพิ่มเติมเพื่อทำให้โมเดลของเราอธิบายคุณสมบัติทางเคมีของโมเลกุลได้ถูกต้องขึ้น เช่น แนวคิดของการใช้ประจุพิเศษ (Extra Charge) ซึ่งเราจะจำลองว่าได้วางประจุอันนี้ไว้ด้านนอกของอะตอมซึ่งมีชื่อเรียกว่า Virtual Charge ซึ่งถูกนำมาใช้ในการอธิบายอิเล็กตรอนคู่โดดเดี่ยวของอะตอม จริง ๆ แล้วเราสามารถพบกรณีพิเศษแบบนี้ได้แม้แต่ในโมเลกุลเล็ก ๆ หรือระบบง่าย ๆ เช่น โมเลกุลน้ำหรือระบบที่มีอิเล็กตรอนที่เกี่ยวข้องกับพันธะ $\pi$

ถ้าอ่านมาถึงตรงนี้แล้วอย่าเพิ่งสับสนนะครับว่าเราสามารถใช้ได้แค่ประจุเชิงอะตอมเพียงอย่างเดียวตามที่บอกไว้ก่อนหน้านี้คือเรามีโมเดลหลายอันที่สามารถนำมาใช้ในการพัฒนา Force Field เพื่อให้ครอบคลุมอันตรกิริยาเชิงไฟฟ้าสถิตย์ แต่ว่าการใช้ประจุเชิงอะตอมนั้นเป็นวิธีที่ได้รับความนิยมมากที่สุดเพราะว่ามีความทั่วไป (General) มากกว่าวิธีอื่น ๆ และสามารถนำไปใช้ได้กับระบบหลาย ๆ อัน (Systematic Approach) ได้อย่างตรงไปตรงมา

คำถามถัดมาคือ “แล้วเราจะคำนวณประจุเชิงอะตอมได้ยังไง?” คำตอบก็คือเราสามารถใช้เทคนิคทางเคมีควอนตัมได้แต่ว่าเทคนิคนั้นมีเป็นสิบ ๆ ทฤษฎีเลยที่ถูกเสนอขึ้นมาเพื่อใช้ในการคำนวณประจุเชิงอะตอม วิธีอันหนึ่งที่ถึงแม้ว่าจะโบราณมาก ๆ แล้วก็ตามแต่ก็ยังได้รับความนิยมมาจนถึงปัจจุบันก็คือนิยามของประจุของมุลลิเกนหรือประจุมุลลิเกน (Mulliken Charge) ซึ่งได้รับการเสนอมาตั้งแต่ปี ค.ศ. 1955, แล้วก็มีทฤษฎีประจุของเฮิร์ชเฟลด์ (Hirshfeld Charge) ซึ่งถูกเสนอในปี ค.ศ. 1977 ซึ่งถูกนำมาใช้อย่างมากในการสร้างพารามิเตอร์ที่ใช้ใน Force Field โดยนำมารวมกันกับค่าจากการทดลอง สำหรับโมเลกุลขนาดเล็กนั้น เราสามารถคำนวณพารามิเตอร์ของประจุเชิงอะตอมได้โดยการพิสูจน์จากไดโพลโมเมนต์เชิงโมเลกุลและควอดรูโพลโมเมนต์ แต่ว่าถ้าเป็นกรณีของโมเลกุลที่มีขนาดใหญ่นั้น เราไม่สามารถทำได้เพราะว่าไดโพลโมเมนต์ของโมเลกุลขนาดใหญ่นั้นเกิดขึ้นมาจากผลรวมของ Contribution หลาย ๆ อันของไฟฟ้าสถิตย์ของอะตอมแต่ละตัวในโมเลกุลซึ่งมันมีความเฉพาะเจาะจงมากเกินไป (มีความ Local มากเกินไป)

### Electronegativity Equalization Model

ในหัวข้อย่อยอันนี้เราจะมาดูตัวอย่างของโมเดลที่สามารถคำนวณประจุเชิงอะตอทในโมเลกุลได้ โมเดลนั้นก็คือ Electronegativity Equalization Model (EEM) ซึ่งจะใช้หลักการที่ว่าประจุของอะตอมแต่ละตัวในโมเลกุลนั้นสามารถที่จะถูกอธิบายได้ด้วยค่า Atomic Electronegativity ($\xi_I$) และค่า Atomic Chemical Hardness ($\eta_I$) ถ้าหากว่าค่า Electronegativity ของอะตอม 2 อันนั้นแตกต่างกัน ประจุจะไหล (Flow) จากอะตอมอันแรกไปอะตอมอันที่สองจนกว่าค่า Molecular Electronegativity นั้นจะมีค่าเฉลี่ยที่เท่า ๆ กันในทุก ๆ ตำแหน่งของโมเลกุล[17,18] ซึ่งค่า Molecular Electronegativity ที่ว่านี้ก็คือ Chemical Potential นั่นเอง นอกจากนี้แล้วยังมีปริมาณอีกหนึ่งตัวที่ทำให้อะตอมนั้นมีประจุก็คืองาน (Work) ซึ่งสามารถหาได้จากค่า Chemical Hardness หรือ Capacitance ของอะตอม ยิ่งไปกว่านั้นทั้ง Electronegativity และ Chemical Hardness นั้นต่างก็เป็น Concept หลักที่เรานำมาใช้ใน Density Functional Theory อีกด้วย[19,20]

สำหรับการคำนวณประจุเชิงอะตอมด้วยวิธี EEM นั้น ผมขอเริ่มต้นด้วยสมการของพลังงานของโมเลกุลที่มี $N$ อะตอม, $V$, ดังนี้

$$V = \sum_{I}^{N} \xi_I q_I + \frac{1}{2}\sum_{I}^{N} \eta_I q_I^2 + \frac{1}{2}\sum_{I,J\neq I}^{N} q_I T_{IJ}^{(0)} q_J + \mu\left(q^{\mathrm{mol}} - \sum_{I}^{N} q_I\right) \tag{2.5.9}$$

โดยที่เทอมที่ 3 นั้นเป็น Coulomb Interaction ระหว่างประจุเชิงอะตอม 2 อัน ($q_I$ และ $q_J$) ส่วนเทอมสุดท้ายนั้นเป็นเทอมที่เป็นตัวบังคับให้ประจุรวมของโมเลกุล ($q^{\mathrm{mol}}$) นั้นมีค่าเท่าเดิมเสมอ แล้วก็ $\mu$ นั้นเป็น Lagrangian Multiplier

ในการหาประจุเชิงอะตอมนั้นสามารถทำได้โดยการปรับค่าพลังงานของโมเลกุลให้ต่ำที่สุดซึ่งก็คือการหาอนุพันธ์ของพลังงานเทียบกับประจุเชิงอะตอมและ Lagrangian Multiplier ซึ่งจะทำให้เราได้ว่า

$$\frac{\partial V}{\partial q_K} = 0 = \xi_K + \eta_K q_K + \sum_{L\neq K}^{N} T_{KL}^{(0)} q_L - \mu \tag{2.5.10}$$

และ

$$\frac{\partial V}{\partial \mu} = 0 \tag{2.5.11}$$

$$= q^{\mathrm{mol}} - \sum_{I}^{N} q_I \tag{2.5.12}$$

โดยที่เทอมสุดท้ายนั้นก็คือเทอมบังคับ (Constraint Term) นั่นเอง ในการหาประจุเชิงอะตอมของอะตอมแต่ละตัวนั้นสามารถหาได้จากการสร้างเมทริกซ์สำหรับการแก้สมการเชิงเส้นทั้งหมด $N+1$ ซึ่งสมการเชิงเส้นทั้งหมดนี้นั้นมีความพัวพันหรือ Coupled กันอยู่และเราสามารถแก้ได้โดยใช้เทคนิคการประมาณเชิงตัวเลขแบบมาตรฐานทั่วไป

$$\begin{pmatrix} \eta_1 & T_{12}^{(0)} & \dots & T_{1N}^{(0)} & 1 \\ T_{21}^{(0)} & \eta_2 & \dots & T_{2N}^{(0)} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ T_{N1}^{(0)} & T_{N2}^{(0)} & \dots & \eta_N & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \\ \mu \end{pmatrix} = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \\ q^{\mathrm{mol}} \end{pmatrix} \tag{2.5.13}$$

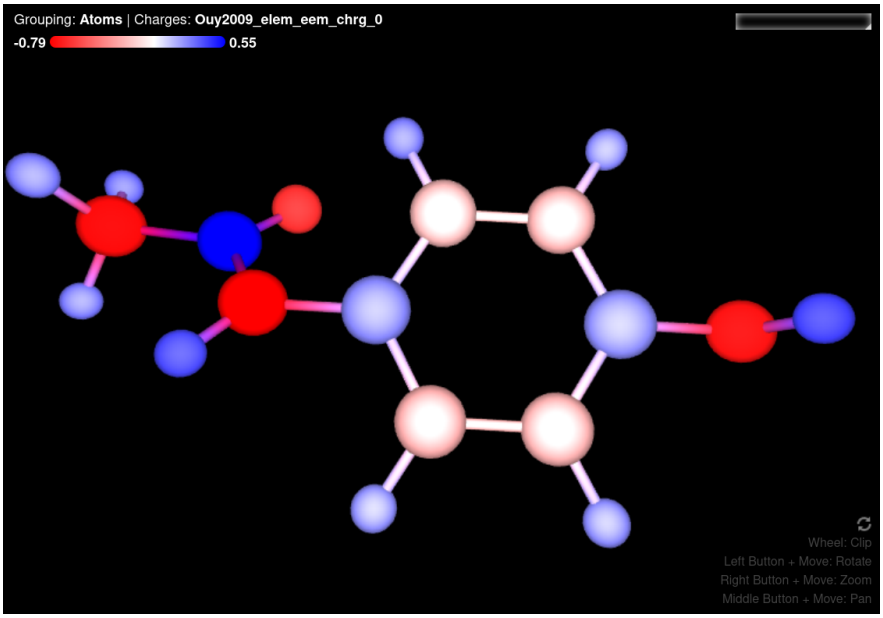
ซึ่งถ้าหากเราทำการย้ายเมทริกซ์ด้านซ้ายสุดไปอยู่ทางด้านขวาของสมการ (ก็คือ Inverse ของเมทริกซ์) เราจะได้เวกเตอร์ที่ประกอบไปด้วยสมาชิกที่เป็นค่าประจุเชิงอะตอมของแต่ละอะตอม ดังนี้

$$\begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \\ \mu \end{pmatrix} = \begin{pmatrix} \eta_1 & T_{12}^{(0)} & \dots & T_{1N}^{(0)} & 1 \\ T_{21}^{(0)} & \eta_2 & \dots & T_{2N}^{(0)} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ T_{N1}^{(0)} & T_{N2}^{(0)} & \dots & \eta_N & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \\ q^{\text{mol}} \end{pmatrix} \tag{2.5.14}$$

สรุปก็คือในการคำนวณประจุเชิงอะตอมนั้น เราจำเป็นที่จะต้องแก้สมการที่ (2.5.14) เพื่อที่จะหาค่า $\xi_I$ และ $\eta_I$ ซึ่งทั้งสองค่านี้นั้นขึ้นอยู่กับชนิดของอะตอม ซึ่งวิธีการหาค่าพารามิเตอร์ของทั้งสองค่าสำหรับแต่ละอะตอมแต่ละชนิดนั้นเรียกว่า Parameterization ซึ่งสามารถทำได้หลายวิธีด้วยกัน เช่น ใช้ Delft Molecular Mechanics (DMM) Force Field ซึ่งถูกพัฒนาขึ้นมาเพื่อทำการปรับค่าพารามิเตอร์ของสารประกอบอินทรีย์หรือ Hydrocarbon โดยการใช้ข้อมูลจากผลการทดลอง

วิธี EEM มีประโยชน์มากโดยเฉพาะการนำมาใช้ในทางเคมีอินทรีย์เพื่อใช้ในการบอกถึงโอกาสของการเกิดปฏิกิริยา Electrophilic และ Nucleophilic Attack ที่เกิดขึ้นในโมเลกุลเนื่องจากว่าปฏิกิริยาเคมีทั้งสองประเภทนี้ขึ้นอยู่กับความแตกต่างเชิงอิเล็กทรอนิกส์ของค่าศักย์เชิงไฟฟ้าสถิตย์ภายในโมเลกุล

ถ้าหากว่าอยากลองใช้ EEM เพื่อคำนวณประจุเชิงอะตอมสามารถใช้งานได้ที่เว็บไซต์ https://webchem.ncbr.muni.cz/Platform/ChargeCalculator




ภาพ 2.4 ตัวอย่างการแสดงผลประจุเชิงอะตอมของโมเลกุล Paracetamol ซึ่งคำนวณด้วยวิธี EEM

**Hydrogen Bonding**

พันธะไฮโดรเจน (Hydrogen Bonding) นั้นเป็นหนึ่งในพันธะที่อธิบายได้ยากมาก ๆ ในทางทฤษฎี ซึ่งทำให้การสร้างโมเดลที่สามารถอธิบาย Hydrogen Bond นั้นมีความท้าทายไปด้วยโดยเฉพาะการทำให้ Force Field นั้นนำไปใช้ได้กับระบบที่มี Hydrogen Bond แบบที่เข้มมาก ๆ

ตัวอย่างอันหนึ่งของการโมเดล Hydrogen Bonding ใน Force Field นั้นก็คือ Force Field ที่ชื่อว่า YETI[21] ซึ่งมีการเพิ่มเทอมพิเศษเข้าไป ดังนี้

$$V_{\text{YETI}} = \left( \frac{A}{R_{H\cdots O}^{12}} - \frac{C}{R_{H\cdots O}^{10}} \right) \cos^2\theta \cos^4\omega \tag{2.5.15}$$

โดยที่ $A$ กับ $C$ นั้นเป็นพารามิเตอร์เฉพาะสำหรับโมเลกุลที่ต้องการศึกษา ซึ่งในที่นี้ก็คือโมเลกุลน้ำแบบคู่ (Water Dimer) นอกจากนี้แล้วจะสังเกตได้ว่าค่าพลังงานนั้นจะขึ้นอยู่กับมุม $\theta$ กับมุม $\omega$ อีกด้วย ซึ่งอันนี้เป็นความตั้งใจของผู้พัฒนา Force Field ที่ต้องการให้โมเดลนี้สามารถอธิบาย Hydrogen Bond ที่ขึ้นอยู่กับทิศทาง (Orientation) ของโมเลกุล

## 2.5.2 Induction Energy

พลังงานที่เกี่ยวข้องกับ Intermolecular Interaction อันที่สองก็คือพลังงานเหนี่ยวนำ (Induction Energy) ซึ่งเป็นพลังงานที่เกิดขึ้นมาจากการโพลาไรซ์ของโมเลกุล (Polarization) ซึ่งผู้อ่านอาจจะสงสัยว่าแล้ว Polarization นั้นคืออะไรกันแน่? คำตอบก็คือ Polarization นั้นเกิดขึ้นเมื่อเราทำการใส่สนามไฟฟ้า (Electric Field) เข้าไปให้กับโมเลกุลซึ่งเป็นการรบกวนโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลแบบหนึ่ง เมื่อโมเลกุลนั้นถูกรบกวนด้วยสนามไฟฟ้า สิ่งที่เกิดขึ้นคือกลุ่มก้อยของอิเล็กตรอนซึ่งมีประจุลบรอบ ๆ นิวเคลียสซึ่งมีประจุบวกภายในอะตอมนั้นมีการเปลี่ยนทิศทางไปในทิศตรงข้ามกับสนามไฟฟ้า และทำให้เกิดการแบ่งของประจุภายในอะตอมซึ่งทำให้ด้านหนึ่งของอะตอมนั้นมีความเป็นบวกและอีกด้านหนึ่งนั้นมีความเป็นลบ ทำให้โมเลกุลนั้นมีความเป็นขั้วทางไฟฟ้าเกิดขึ้นมาและทำให้เกิดอันตรกิริยาทางไฟฟ้าสถิตย์กับโมเลกุลอื่นรอบ ๆ ได้ ดังนั้นเราจึงเรียกปรากฏการณ์แบบนี้ว่า Induction Interaction นั่นเอง ซึ่งในทางเคมีเชิงคำนวณนั้นเราจำเป็นที่จะต้องใส่เทอมของพลังงานเหนี่ยวนำนี้เข้าไปใน Force Field ด้วยเพื่อเพิ่มความแม่นยำให้กับโมเดล

เราใช้ Polarizability $(\alpha_{\alpha\beta})$ ในการอธิบายความสามารถในการเกิด Polarization ของโมเลกุลภายใต้สนามไฟฟ้า $(E_\beta)$ โดยที่ $(\alpha_{\alpha\beta})$ นั้นเป็นการตอบสนองเชิงเส้น (Linear Response) ต่อสนามไฟฟ้า ซึ่งมีนิยามดังต่อไปนี้

$$\mu_\alpha^{\text{ind}} = \alpha_{\alpha\beta} E_\beta \tag{2.5.16}$$

โดยที่ $\mu_{\alpha}^{\text{ind}}$ คือไดโพลโมเมนต์ที่ถูกเหนี่ยวนำ (Induced Dipole Moment) เราสามารถทำการ Generalize สมการที่ (2.5.16) ให้อ้างอิงกับการเกิดโพลาไรเซชั่นของอะตอม (Atomic Polarizability) $(\alpha_{I,\alpha\beta})$ ซึ่งจะทำให้เรามีสมการของ Induced Dipole Moment สำหรับกรณีปกติ ดังนี้

$$\mu_{I,\alpha}^{\text{ind}} = \alpha_{I,\alpha\beta} E_{I,\beta}^{\text{tot}} \tag{2.5.17}$$

โดยที่ $\mu_{I,\alpha}^{\text{ind}}$ คือ Atomic Induced Dipole Moment และ $E_{I,\beta}^{\text{tot}}$ คือผลรวมของสนามไฟฟ้าทั้งหมดที่กระทำต่ออะตอม $I$ โดยสนามไฟฟ้าทั้งหมดนั้นก็จะมีการรวมสนามไฟฟ้าจากภายนอกที่เราใส่เข้าไปและสนามไฟฟ้าที่มาจากประจุเชิงอะตอมรอบ ๆ โมเลกุลด้วย เป็นต้น

สำหรับหน้าตาของพลังงานเหนี่ยวนำที่เราจะนำเข้าไปใส่ใน Force Field ของเรานั้นจะมีการรวมพลังงาน 3 อันเข้าไว้ด้วยกันคือ ไฟฟ้าสถิตย์ (Electrostatic Energy), พลังงานที่เกิดขึ้นจากตัวของอะตอมเอง (Self-Energy), และพลังงานที่เกิดจากอันตรกิริยาระหว่างไดโพล-ไดโพล (Dipole-Dipole Interaction Energy) ซึ่งพลังงานเหนี่ยวนำ $(V_{\text{ind}})$ มีสมการดังต่อไปนี้[22]

$$V_{\text{ind}} = -\frac{1}{2} \sum_{I,J \neq I}^{N} \mu_{I,\alpha}^{\text{ind}} T_{IJ,\alpha\beta}^{(2)} \mu_{J,\beta}^{\text{ind}} + \sum_{I}^{N} V_{I,\text{ self}} - \sum_{I}^{N} \mu_{I,\alpha}^{\text{ind}} E_{I,\alpha}^{\text{ext}} \tag{2.5.18}$$

โดยที่ $\mu_{I,\alpha}^{\text{ind}}$ คือ Induced Dipole Moment ของอนุภาค $I$, $T_{IJ,\alpha\beta}^{(2)}$ คือเทนเซอร์ที่อธิบายอันตรกิริยา (ขออนุญาตไม่ลงรายละเอียดครับ), และ $V_{I,\text{ self}}$ คือพลังงานอนุภาคเอง (Self-Energy) ซึ่งมีสมการดังนี้

$$V_{I,\text{ self}} = \frac{1}{2} \left(\alpha_{I,\alpha\beta}\right)^{-1} \mu_{I,\alpha}^{\text{ind}} \mu_{I,\beta}^{\text{ind}} \tag{2.5.19}$$

ระบบโมเลกุลที่ถูกโพลาไรซ์นั้นจะมีการเปลี่ยนแปลงของพลังงานเหนี่ยวที่ลดลงซึ่งก็คือการ Minimization นั่นเอง ดังนั้นเราจึงกำหนดเงื่อนไขขึ้นมาได้ดังนี้

$$\frac{\partial V_{\text{ind}}}{\partial \mu_{K,\gamma}^{\text{ind}}} = 0 = -\left( \sum_{J \neq K}^{N} T_{KJ,\gamma\beta}^{(2)} \mu_{J,\beta}^{\text{ind}} \right) + \left(\alpha_{K,\beta\gamma}\right)^{-1} \mu_{K,\beta}^{\text{ind}} - E_{K,\gamma}^{\text{ext}} \tag{2.5.20}$$

ซึ่งมีความหมายก็คือโมเลกุลนั้นจะไม่มีการเปลี่ยนแปลงของพลังงานเหนี่ยวนำเมื่อเทียบกับไดโพลโมเมนต์เหนี่ยวนำเมื่อมีการเกิด Polarization ซึ่งจะทำให้เราได้ว่า

$$\mu_{K,\beta}^{\text{ind}} = \alpha_{K,\beta\gamma} \left( E_{K,\gamma}^{\text{ext}} + \sum_{J \neq K}^{N} T_{KJ,\gamma\beta}^{(2)} \mu_{J,\beta}^{\text{ind}} \right) \tag{2.5.21}$$

ไดโพลโมเมนต์เหนี่ยวนำเชิงอะตอม (Atomic Induced Dipole Moment) นั้นสามารถหาได้จากการ Coupling กันของสมการทั้งหมด $3N$ สมการหรือที่เรียกว่า (Coupled Equations) ถ้าหากว่าเรารวมผลของ Polarizability เข้าไปใน Force Field เราจะได้ว่าการคำนวณไดโพลโมเมนต์เหนี่ยวนำนั้นจะใช้เวลาในการคำนวณนานมาก ๆ เมื่อเทียบกับเทอมอื่นของพลังงานเหนี่ยวนำ เพราะว่าไดโพลโมเมนต์เหนี่ยวนำนั้นเป็นเทอมของปัญหาแบบ Many-Body หรือปัญหาที่ขึ้นกับจำนวนของอนุภาคทุกอนุภาคในระบบ

นอกจากนี้เรายังสามารถทำสมการที่ (2.5.20) ให้มีหน้าตาที่ง่ายขึ้นได้โดยการใช้สมการที่ (2.5.21) ดังนี้

$$\begin{aligned} V_{\text{ind}} &= -\frac{1}{2}\sum_{I,J\neq I}^{N}\mu_{I,\alpha}^{\text{ind}}T_{IJ,\alpha\beta}^{(2)}\mu_{J,\beta}^{\text{ind}} + \frac{1}{2}\sum_{I}^{N}\mu_{I,\alpha}^{\text{ind}}\left(E_{I,\alpha}^{\text{ext}} + \sum_{J\neq I}^{N}T_{IJ,\alpha\beta}^{(2)}\mu_{J,\beta}^{\text{ind}}\right) - \sum_{I}^{N}\mu_{I,\alpha}^{\text{ind}}E_{I,\alpha}^{\text{ext}} \\ &= -\frac{1}{2}\sum_{I}^{N}\mu_{I,\alpha}^{\text{ind}}E_{I,\alpha}^{\text{ext}} \end{aligned} \tag{2.5.22}$$

ซึ่งจะเห็นว่าเทอม Self-Energy นั้นจะทำให้เทอมอันตรกิริยา Induced Dipole-Induced Dipole กับเทอม Indiced Dipole และเทอมสนามไฟฟ้านั้นหายไปครึ่งหนหนึ่ง แล้วถ้าหากว่าเราทำใส่แทนสมการที่ (2.5.21) เข้าไป เราจะได้สมการที่สามารถนำไปใช้ในการคำนวณพลังงานเหนี่ยวนำสำหรับ Force Field ดังนี้

$$V_{\text{ind}} = -\frac{1}{2}\sum_{I}^{N}\alpha_{I,\alpha\beta}\left(E_{I,\beta}^{\text{ext}} + \sum_{J\neq I}^{N}T_{IJ,\beta\gamma}^{(2)}\mu_{J,\gamma}^{\text{ind}}\right)E_{I,\alpha}^{\text{ext}} \tag{2.5.23}$$

### 2.5.3 Dispersion และ Short-Range Repulsion

ในหัวข้อนี้เราจะมาศึกษาอันตรกิริยาระหว่างโมเลกุลอีกแบบหนึ่งซึ่งไม่แตกต่างจากอันตรกิริยา 2 หัวข้อก่อนหน้านี้ที่เพิ่งได้ศึกษาไป ยกตัวอย่างเช่น ถ้าเราสนใจโมเลกุลอาร์กอนที่มีสถานะเป็นของเหลว (Liquid Argon) และไม่มีการใส่สนามไฟฟ้าภายนอกเข้าไปให้กับระบบโมเลกุลอันนี้ สิ่งที่เกิดขึ้นคือจะไม่มีอันตรกิริยา Electronic และ Polarization เกิดขึ้นมา แต่ว่าเรายังมีพลังงานระหว่างโมเลกุลอยู่อีก ซึ่งก็คือพลังงานการแพร่กระจาย (Dispersion Energy) และพลังงานที่เกิดจากแรงผลักแบบพิสัยใกล้ (Short-Range Repulsion Energy) ถ้าพร้อมแล้วก็ลุยกันเลยครับ

#### Dispersion Energy

พลังงานการแพร่กระจาย (Dispersion Energy) เกิดจากอันตรกิริยาที่มาจากการ Correlation กันของการเคลื่อนที่ของอิเล็กตรอนในโมเลกุล โดยเรามีสมการที่ชื่อว่า London Equation ที่ถูกพัฒนาขึ้นมาเพื่อใช้

ในการคำนวณ Dispersion Energy โดยพิสูจน์มาจากทฤษฎี Second-Order Perturbation Theory[1] ซึ่ง London Energy มีสมการดังต่อไปนี้[23,24,25]

$$V_{\text{disp}} = -\frac{C_6}{R^6} \tag{2.5.24}$$

โดยที่ $C_6$ คือพารามิเตอร์ที่มีค่าเป็นบวกเสมอซึ่งสมเหตุสมผลกับการที่ Dispersion นั้นจะต้องมีค่าเป็นลบเสมอ (เป็นแรงดึงดูด) นอกจากนี้จะเห็นได้ว่าพลังงาน Dispersion นั้นจะแปรผกผันกับ $R^{-6}$ ตีความง่าย ๆ ก็คือ Dispersion นั้นเป็นอันตรกิริยาหรือแรงที่อ่อนมาก ๆ ยิ่งอะตอมหรือโมเลกุลอยู่ห่างกันมากเท่าไหร่ค่า Dispersion Force นั้นก็จะลดลงเยอะมาก และจริง ๆ แล้วเทอมที่แสดงในสมการที่ (2.5.24) นั้นเป็นเพียงแค่เทอม ๆ หนึ่งจากอนุกรมผลรวมของ Range-Separated Interaction ซึ่งเราสามารถเขียนพลังงาน Dispersion ให้มีความสมบูรณ์มากขึ้นได้โดยการรวมเทอมอันดับสูง ๆ จาก Perturbation Expansion ได้ดังนี้

$$V_{\text{disp}} = -\frac{C_6}{R^6} - \frac{-C_8}{R^8} - \frac{-C_{10}}{R^{10}} + \dots \tag{2.5.25}$$

อนุกรมด้านบนนี้จริง ๆ แล้วก็คือ Taylor Expansion ของ $\frac{1}{R}$ ซึ่งปกติแล้วเรามักจะทำการตัดอนุกรมด้านบนให้เหลือแค่เทอม $R^{-6}$ เท่านั้นเพื่อความง่ายต่อการคำนวณเพราะว่าเทอมสูง ๆ นั้นมีค่าน้อยมากนั่นเอง

### Repulsion Energy

พลังงานการผลักนั้นมาจากหลักการการกีดกันของเพาลี (Pauli Exclusion Principle) ซึ่งมีใจความว่าอิเล็กตรอนนั้นไม่สามารถมีสถานะทางควอนตัมที่เหมือนกันพร้อม ๆ กันได้ ตัวอย่างเช่น ถ้าหากว่าเรามีอะตอมของอาร์กอน 2 อะตอมอยู่ใกล้ ๆ กัน (เป็นระบบ Closed Shell) อาร์กอนทั้ง 2 อะตอมนี้จะผลักกันและสอดคล้องกับการที่ทำให้ Pauli Exclusion Principle นั้นยังเป็นจริงอยู่ โดยพลังงานที่เกิดขึ้นจากการผลักกันระหว่างอะตอมนั้นมักจะถูกพิจารณาหรือศึกษาโดยการแบ่งออกเป็น 2 กรณี ก็คือแรงผลักที่เกิดขึ้นในพิสัยใกล้ (Short-Range) และแรงผลักที่เกิดขึ้นในพิสัยไกล (Long-Range) โดยเราสามารถคำนวณหา Repulsion Energy ได้จากการใช้อันตรกิริยาระหว่างโมเลกุลทั้ง 3 อันก่อนหน้านี้ที่เราเพิ่งได้ศึกษาไปก็คือ Electrostatic, Induction และ Dispersion Forces มาใช้ในการสร้าง Approximation สำหรับ Repulsion Energy ซึ่งมีสมการดังนี้

$$V_{\text{Repulsion}} = V_{\text{Interaction}} - V_{\text{Electrostatic}} - V_{\text{Induction}} V_{\text{Dispersion}} \tag{2.5.26}$$

[1]London Dispersion มีชื่อเรียกเต็ม ๆ ว่า London Dispersion Force และยังมีชื่อเรียกอื่นอีก เช่น London Forces, Instantaneous Dipole-Induced Dipole Forces, Fluctuating Induced Dipole Bonds หรืออาจจะเรียกว่า van der Waals (vdW) Force ก็ได้เพราะว่า London Force นั้นจัดว่าเป็น vdW แบบหนึ่ง (ซึ่งจริง ๆ แล้วก็ไม่ถูกซะทีเดียว)

จะเห็นได้ว่าสมการด้านบนนั้นมีการรวมความคลาดเคลื่อนของอันตรกิริยาแบบ Short-Range และ Long-Range เข้าไปด้วยในรูปของเทอมพลังงานแต่ละอันที่อันดับสูง ๆ เช่น Dispersion Energy ดังนั้นพลังงานการผลักหรือ Repulsion Energy ($V_{\text{Repulsion}}$) นั้นจึงสามารถถูกปรับพารามิเตอร์ (Parameterization) ได้จากการพลังงานอันตรกิริยา (Interaction Energy) ของระบบที่มีหลายโมเลกุล (เช่น Dimer หรือ Cluster) ด้วยวิธีการเคมีควอนตัมแล้วก็ทำการนำค่าพลังงานของเทอมอื่น ๆ มาหักลบออกไป

### Lennard-Jones Potential

โมเดลอีกอันหนึ่งที่ถูกพัฒนาขึ้นมาเพื่อใช้ในการอธิบาย Attraction-Repulsion Interaction ระหว่างอะตอมหรือโมเลกุลก็คือโมเดลพลังงานศักย์ของเลนนาร์ด-โจนส์ (Lennard-Jones หรือ LJ) ซึ่งมีแนวคิดเริ่มต้นมาจากการรวมเทอม Repulsion Energy ที่พิสัยใกล้กับเทอมแรงดึงดูด London Dispersion Energy ที่พิสัยไกลเข้าไว้ด้วยกัน ดังสมการต่อไปนี้

$$V_{\text{LJ}} = 4\epsilon\left(\left(\frac{\sigma}{R}\right)^{12} - \left(\frac{\sigma}{R}\right)^{6}\right) = \frac{a}{R^{12}} - \frac{C_6}{R^6} \tag{2.5.27}$$

จะเห็นว่าเรามีเทอม $R^{-12}$ อยู่ในสมการ ซึ่งเทอมนี้เป็นเทอมที่อธิบาย Repulsion ซึ่งมีเลขยกกำลังเป็นสองเท่าของเลขยกกำลังของ $R^{-6}$ ซึ่งมาจากการที่ Lennard-Jones นั้นแก้สมการมาจากระบบของ Statistical Mechanical Model

สำหรับ Force Field ที่ใช้กับระบบที่มีมากกว่า 1 โมเลกุลนั้นเราสามารถคำนวณ Lennard-Jones Potential ทั้งหมดได้โดยการใช้ Pair-Wise Additive ดังนี้

$$\sum_{I=1}^{N_A}\sum_{J=1}^{N_B} V_{\text{LJ}} = 4\epsilon_{IJ}\left(\left(\frac{\sigma_{IJ}}{R_{IJ}}\right)^{12} - \left(\frac{\sigma_{IJ}}{R_{IJ}}\right)^{6}\right) \tag{2.5.28}$$

โดยที่ $N_A$ และ $N_B$ คือจำนวนอะตอมของโมเลกุล $A$ กับโมเลกุล $B$ ส่วนพารามิเตอร์ $\epsilon_{IJ}$ กับ $\sigma_{IJ}$ นั้นเป็นพารามิเตอร์ที่ขึ้นกับอันตรกิริยาระหว่างอะตอม $I$ กับอะตอม $J$ ซึ่งจะมีพารามิเตอร์เยอะมาก ๆ ถ้าหากว่าโมเลกุลนั้นมีขนาดใหญ่ ซึ่งวิธีที่เราสามารถใช้ในการลดจำนวนพารามิเตอร์ทั้ง 2 ตัวนี้ได้ก็คือการใช้กฎการผสมของ Lorentz-Berthlot ดังนี้

$$\sigma_{IJ} = \frac{1}{2}(\sigma_{II} + \sigma_{JJ}) \tag{2.5.29}$$

และ

$$\epsilon_{IJ} = \sqrt{\epsilon_{II}\epsilon_{JJ}} \tag{2.5.30}$$

## 2.6 สมการของการเคลื่อนที่

หัวใจสำคัญของ MD Simulations นั้นก็คืออันตรกิริยาระหว่างโมเลกุลนั่นก็คือ “แรง (Force)” โดยมีสมการสำคัญ 2 สมการที่ถือได้ว่าเป็นสมการหลักของ MD เลยก็ว่าได้ ดังนี้

$$m_i \ddot{\boldsymbol{r}}_i = \boldsymbol{f}_i \tag{2.6.1}$$

$$\boldsymbol{f}_i = -\nabla_i V(\boldsymbol{r}) \tag{2.6.2}$$

สมการด้านบนนี้คือสมการการเคลื่อนที่ของนิวตัน (Newtonian Equation of Motion) สำหรับอะตอม $i$ โดยที่เป้าหมายของเรานั้นก็คือการคำนวณแรง $\boldsymbol{f}$ ที่กระทำต่ออะตอมซึ่งสามารถคำนวณได้จากพลังงานศักย์ $V(\boldsymbol{r})$ นั่นเอง ส่วนเวกเตอร์ $\boldsymbol{r}$ นั้นก็คือพิกัดคาร์ทีเซียนของตำแหน่งของอะตอม (นิวเคลียส) ทั้งหมดทุกอะตอมในโมเลกุลซึ่งเป็นพิกัดแบบ 3 มิติ

$$\boldsymbol{r} = (\underbrace{r_{1,x}, r_{1,y}, r_{1,z}}_{\text{อะตอมตัวที่ 1}}, \ldots, \underbrace{r_{N,x}, r_{N,y}, r_{N,z}}_{\text{อะตอมตัวที่ } N}) \tag{2.6.3}$$

โดยในการจำลอง MD นั้นจะเป็นการแก้สมการที่ (2.6.1) และ (2.6.2) พร้อม ๆ กันไปเป็นสเต็ป ๆ ตลอดช่วงระยะเวลาที่ทำการจำลอง โดยระยะห่างระหว่างสเต็ปนั้นเรียกว่า Time Step $(\Delta t)$

## 2.7 ข้อจำกัดของ MD

วิธี MD นั้นก็เหมือนกับวิธีการจำลองทางคอมพิวเตอร์อื่น ๆ ที่มีข้อจำกัดทั้งในเชิงตัวโมเดลของวิธีเองกับในเชิงทรัพยากรที่ใช้ในการคำนวณ โดยข้อจำกัดของ MD สามารถแบ่งออกได้เป็น 4 ข้อหลัก ๆ ดังนี้

**1. Time Scale** สเกลเวลาหรือ Time Scale คือสเกลที่บอกถึงระดับของช่วงเวลาที่ใช้ในการอธิบายปรากฏการณ์หรือพฤติกรรมของโมเลกุลหรือระบบที่เราต้องการศึกษา เช่น การสั่นของพันธะโมเลกุลนั้นจะมี Time Scale ในระดับ Femtosecond ดังนั้น Time Scale ที่เหมาะสมสำหรับการกำหนด Time Step นั่นจึงอยู่ที่ประมาณ 1 fs เพราะว่าถ้าหากเรากำหนด Time Step ที่กว้างหรือช้ากว่านี้เช่น 10 fs เราก็จะไม่สามารถติดตามการสั่นของโมเลกุลได้เพราะว่าช่วงระยะเวลาที่ใช้ในการขยับหรือเปลี่ยนตำแหน่งของโครสร้างของโมเลกุลนั้นมากกว่าการสั่นของโมเลกุลหลายเท่า

สำหรับการจำลองเหตุการณ์หรือ Event ในการจำลอง MD นั้นเราควรจะต้องทราบถึงระยะเวลาที่เร็วที่สุดที่เหตุการณ์นั้นสามารถเกิดขึ้นได้ก่อน เช่น การพับของโปรตีน (Protein Folding) นั้นจะใช้เวลาประมาณ 1 วินาที ดังนั้นถ้าหากเรากำนดให้ Time Step = 1 fs เราจะต้องทำการจำลอง MD ประมาณ $10^{15}$ สเต็ปถึงจะสามารถจำลองการพับของโปรตีนได้ อย่างไรก็ตามในความเป็นจริงนั้นปรากฏการณ์ต่าง ๆ ของโมเลกุลที่เกิดขึ้นนั้นมักจะเกิดขึ้นในช่วงเวลาระดับ Microsecond $(\mu s)$

**2. Length Scale** สเกลขนาดหรือ Length Scale คือสเกลที่บ่งบอกถึงขนาดของระบบที่ถูกจำลอง ซึ่ง Length Scale นี้จะแบ่งตามขนาดของระบบที่ใช้ในการศึกษา ถ้าหากเราต้องการที่จะศึกษาคุณสมบัติของระบบที่มีขนาดใหญ่ Length Scale ก็จะต้องสอดคล้องกับระบบด้วย เช่น การจำลองโครงข่ายพอลิเมอร์ (Polymer) เพื่อให้มีความเหมาะสมและมีขนาดใหญ่ของระบบที่ใหญ่มากพอที่จะเป็นตัวแทนของระบบพอลิเมอร์ในธรรมชาติจริง ๆ

**3. ความแม่นยำของแรงที่คำนวณได้** หัวใจสำคัญของ MD นั้นก็คือการคำนวณแรงที่เป็นอันตรกิริยาระหว่างอะตอมในโมเลกุล ถ้าหากเราใช้วิธีการคำนวณแรงที่มีความแม่นยำสูงก็จะทำให้เราได้แรงที่มีความถูกต้องมาก แต่วิธีการที่มีความแม่นยำสูงนั้นมักจะต้องแลกมาด้วยการคำนวณที่สิ้นเปลือง ดังนั้นเรามักจะทำการ Trade-off หรือชั่งน้ำหนักระหว่างการเลือกวิธีในการคำนวณแรงและความสิ้นเปลืองของวิธีนั้น ๆ เพราะอย่าลืมว่าเราต้องคำนวณแรงทุก ๆ สเต็ปของการจำลอง MD

## 2.8 ขั้นตอนการจำลอง MD

การจำลอง MD นั้นโดยปกติแล้วประกอบไปด้วยขั้นตอนดังต่อไปนี้

### 2.8.1 เลือกโมเดล

การเลือกโมเดลแบบจำลองเชิงอะตอม (Atomistic Simulation Model) เพื่อมาจำลองระบบโมเลกุลของเรานั้นเป็นสิ่งที่สำคัญมากที่สุด เพราะนั่นหมายถึงการเลือกประเภทของการจำลองที่จะต้องสอดคล้องกับขนาดของระบบของเราและสิ่งที่เราต้องการจำลอง ซึ่งปรากฏการณ์ต่าง ๆ ทางเคมีหรือทางฟิสิกส์ที่เราต้องการจำลองนั้นมีช่วงเวลาหรือ Time Scale ที่เกิดขึ้นแตกต่างกันไป ถ้าหากว่าเรามีระบบโมเลกุลที่มีขนาดใหญ่มาก ๆ และต้องการรัน Simulation เป็นระยะเวลานานเพื่อศึกษาปรากฏการณ์ที่ใช้ระยะเวลานานในการที่จะจำลองขึ้นมา (หรือเราอาจจะรัน Simulation ไปเรื่อน ๆ จนกว่าจะเจอสิ่งที่เราต้องการ) วิธีที่เราใช้ในการจำลองนั้นก็ควรที่จะไม่สิ้นเปลืองมากเกินไป ในทางตรงกันข้าม ถ้าหากว่าเราต้องการศึกษากระบวนการการเปลี่ยนแปลงทางเคมีที่เกิดขึ้นในช่วงระยะเวลาสั้น ๆ และเกิดขึ้นได้เร็ว (มี Time Scale ที่อยู่ในช่วงน้อย ๆ) เราก็สามารถที่จะใช้วิธีการจำลองที่มีความอลังการได้ ซึ่งก็จะให้ผลการคำนวณที่ถูกต้องและแม่นยำกว่าวิธี

การจำลองที่หยาบกว่า

ประเภทของวิธีหรือโมเดลแบบจำลอง Atomistic Simulations ที่สำคัญ ๆ นั้นแบ่งได้ดังนี้

Molecular Dynamics (MD)

- เป็นการจำลองการเคลื่อนที่ของระบบอะตอมหรือโมเลกุล

Monte Carlo (MC)

- MC คือการจำลองคอมพิวเตอร์แบบไหนก็ได้ที่มีการใช้เลขสุ่ม (Random Number)
- แม้แต่การคำนวณ MD ส่วนใหญ่นั้นก็ใช้ Random Number แต่ว่าก็ยังไม่ถือว่าเป็นวิธี MC ซะทีเดียว
- ตัวอย่างของการจำลองแบบ MC เช่น
    1. Metropolis MC (MMC): เป็นการจำลอง Thermodynamic Ensemble, Energy Minimization โดยการใช้ Simulated Annealing
    2. Kinetic MC (KMC): เป็นการจำลอง Activated Processes เช่น กระบวนการ Diffusion

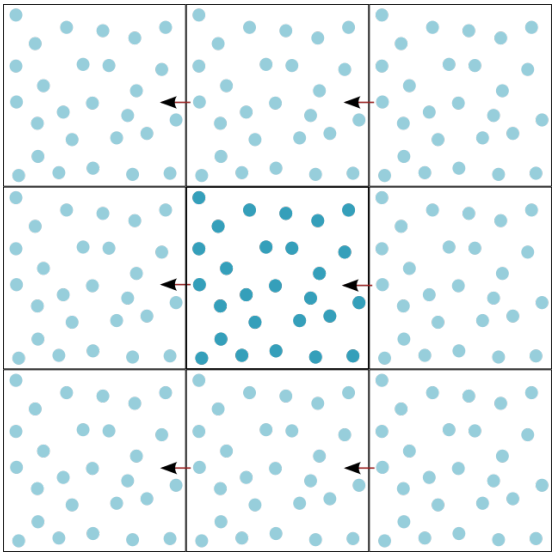
## 2.8.2 เตรียมโครงสร้างเริ่มต้น

เราสามารถเตรียมระบบที่เราต้องการจำลองเพื่อศึกษาคุณสมบัติต่าง ๆ ของระบบนั้นได้โดยการใช้โปรแกรม Molecular Editors ที่สามารถให้เราแก้ไข จัดการ ปรับเปลี่ยน อะตอมหรือโมเลกุลของระบบของเราได้ เราสามารถใช้โปรแกรม VMD[1] ในการแสดงระบบโมเลกุลของเราซึ่งประเภทของไฟล์ข้อมูล Cartesian Coordinates ของระบบของเรานั้น (ไฟล์ xyz) ก็ได้รับความนิยมมากที่สุดเพราะว่ามีความกระชับและสามารถนำไฟล์ไปเปิดด้วยโปรแกรมอื่น ๆ ได้ด้วย และสะดวกต่อการแก้ไขและจัดการ

สิ่งที่สำคัญมากอีกอย่างหนึ่งในการจำลอง MD นั้นก็คือ Periodic Boundary Condition (PBC) ซึ่งเป็นตัวที่ช่วยเพิ่มความเป็นกลุ่มก้อน (Bulk) ให้กับระบบของเรามากขึ้น โดยเฉพาะระบบที่เป็น Solvent หรือ Crystal ซึ่งการใช้ PBC นั้นจะเป็นการรักษาและทำให้คุณสมบัติเชิงความร้อน (Thermodynamic Properties) เช่น อุณหภูมิ, ความดัน, ความหนาแน่น นั้นมีค่าที่ถูกต้องและสมเหตุสมผลตลอดการจำลอง

ภาพที่ 2.5 แสดง Periodic Box ของ Unit Cell ในการจำลอง MD โดยสิ่งที่ PBC นั้นก็คือการจำลอง Unit Cell ขึ้นมาหลาย ๆ อันแล้วไปวางรอบ ๆ Unit Cell ที่เป็น Reference ในทุกทิศทางทั้ง x, y, และ z (ในกรณี 3 มิติ) ถ้าหากว่าอะตอมหรือโมเลกุลใน Unit Cell ที่เป็น Reference ขยับออกไปจาก Cell ทางด้านหนึ่ง ก็จะไปโผลใน Unit Cell อันที่อยู่ติดกันในอีกทางด้านหนึ่ง พูดง่าย ๆ คือ PBC นั้นช่วยลด Artifact ระหว่างพื้นผิวของ Unit Cell แต่ละอันที่เกิดขึ้นจากอันตรกิริยาของระบบโดดเดี่ยว (Isolated System) กับสุญญากาศนั่นเอง

[1]VMD เป็นโปรแกรมที่สามารถดาวน์โหลดมาใช้งานได้ฟรี สามารถแสดงผล (Display), เล่นภาพเคลื่อนไหว (Animate), วิเคราะห์โครงสร้างของโมเลกุลขนาดใหญ่ได้, และยังสามารถ Render ภาพได้ด้วย ดาวน์โหลดได้ที่ https://www.ks.uiuc.edu/Research/vmd/



ภาพ 2.5 Periodic Boundary Condition (PBC)

### 2.8.3 สรุป Key Steps และแนวคิดของการจำลอง MD

1. การเตรียมระบบที่ต้องการจำลอง
   - เริ่มต้นด้วยการหา Cartesian Coordinates ของโครงสร้างโมเลกุลทางเคมีที่ต้องการศึกษา
2. รันการจำลอง Energy Minimization เพื่อทำการ Relax โครงสร้างของโมเลกุล
3. รันการจำลอง Equilibration ด้วยวิธี NVT และ/หรือ NPT
   - ใช้ NVT Ensemble เพื่อควบคุมอุณหภูมิ
   - ตามด้วยการใช้ NPT Ensemble เพื่อควบคุมทั้งอุณหภูมิและความดัน
4. รันการจำลอง MD
5. ศึกษา ทดสอบและประเมินการคำนวณโดยพิจารณาจากพารามิเตอร์ต่อไปนี้
   - Ensemble Settings (NVT และ NPT)
   - อุณหภูทิและความดัน
   - Integrator
   - Cutoff Distance
   - Constraints และElectrostatic Treatments
6. แสดงผลและวิเคราะห์ผล
   - ใช้โปรแกรม เช่น VMD สำหรับการแสดงโครงสร้างของโมเลกุล (Trajectory) และตรวจสอบว่าเราได้ผลการคำนวณที่เหมาะสม

## 2.9 การวิเคราะห์ผลการจำลอง MD

ตัวอย่างโปรแกรมสำหรับการวิเคราะห์เชิงโครงสร้าง (Structural Analysis) ซึ่งผู้อ่านสามารถดาวน์โหลดมาทดลองใช้ได้ฟรี ซึ่งจริง ๆ โปรแกรมเหล่านี้ก็ถูกนำมาใช้จริงในการทำงานวิจัยทางเคมีเชิงคำนวณ

● Visualization Tools:
VMD (Visual Molecular Dynamics): VMD สามารถแสดงผลและวิเคราะห์ข้อมูลเชิงโครงสร้างของโมเลกุลได้ และยังช่วยในเรื่องของการทำ Identification สำหรับปรับแก้ไขโครงสร้างบางส่วนของโมเลกุล

● Molecular Editing Software:
PyMOL: PyMOL มาพร้อมกับเครื่องที่หลากหลายที่ช่วยให้เราแก้ไขและจัดการโมเลกุลได้ โดยผู้ใช้งานสามารถเลือก ลบ แก้ไข โครงสร้างบางส่วนได้

● Topological Analysis:
GROMACS Utilities: นอกจาก GROMACS จะเป็นโปรแกรมที่สามารถรันการจำลอง MD ได้อย่างมีประสิทธิภาพมาก ๆ โปรแกรมหนึ่งแล้ว GROMACS ยังมาพร้อมกับฟีเจอร์ที่เยอะมาก ๆ อีกด้วยโดยเฉพาะสำหรับการทำ Topology Analysis

● Graphical User Interface (GUI) Tools:
UCSF Chimera: Chimera เป็นโปรแกรมสำหรับการแสดงโมเลกุลอีกโปรแกรมหนึ่งที่ผมชอบส่วนตัว ยังมีเครื่องคือที่สามารถใช้ในการวิเคราะห์โมเลกุล (Analysis Tool) ที่ใช้งานได้ง่าย

● Scripting Languages:
Python and BioPython: เป็นโปรแกรมที่ผู้ใช้งานสามารถใช้ Scripting Languages ในการควบคุมโปรแกรมเพื่อทำตามคำสั่งที่ต้องการ นอกจากนี้แล้ว BioPython ยังมีโมดูลสำหรับการจัดวางและแก้ไขโครงสร้างโมเลกุลอีกด้วย

● Force Field Preparation Tools:
CHARMM-GUI, AMBERTools: เป็นเครื่องมือสำหรับการเตรียมระบบโมเลกุลเพื่อเอาไปจำลองด้วยวิธี MD ต่อไป

● Hydrogen Bond Analysis Tools:
HBondAnalyzer (เป็นหนึ่งในโปรแกรมของ GROMACS): ช่วยให้เราวิเคราะห์พันธะไฮโดรเจนได้ซึ่งทำให้เราเข้าใจอันตรกิริยาระหว่างโมเลกุล

## 2.10 ศึกษาเพิ่มเติมเกี่ยวกับงานวิจัยทางด้าน MD

- M. P. Allen, D. Tildesley: Computer Simulation of Liquids (Oxford University Press, Oxford, 1989)
  - หนังสือที่เขียนได้ดีมาก หลายคนอ่านเล่มนี้และเรียกได้ว่าเป็นสุดยอดหนังสือของ Molecular

Simulation เลย

  - ฝนหนังสือมีอธิบายกลศาสตร์เชิงสถิติ (Statistical Mechanics) ด้วย

- D. Frenkel, B. Smit: Understanding Molecular Simulation: From Algorithms to Applications, 2nd Edition (Academic Press, 2001)
  - หนังสือที่อธิบายกลศาสตร์เชิงสถิติที่มีการประยุกต์สำหรับการศึกษาระบบโมเลกุล
- R. Phillips: Crystals, Defects and Microstructure: Modeling Across Scales (Cambridge University Press, 2001)
  - เป็นหนังสือที่รวบรวมวิธีเชิงคำนวณสำหรับการทำวิจัยทางด้านการจำลองคอมพิวเตอร์ของวัสดุ
- A. R. Leach: Molecular Modelling: Principles and Applications, 2nd Edition (Prentice Hall, 2001)
  - หนังสือที่อธิบายวิธีการจำลองคอมพิวเตอร์ของระบบโมเลกุลได้ดีมาก ๆ ทำความเข้าใจได้ง่าย นอกจากนี้ยังมีพูดถึง Interaction Model ด้วย ทั้งแบบ Classical และแบบ Quantum
  - ในหนังสือเน้นไปที่ Molecular Mechanics และ Force Field

## 2.11 แบบฝึกหัด

1. พิสูจน์สมการการเคลื่อนที่ (Equation of Motion) ของ Molecular Dynamics

2. เขียนโปรแกรม Molecular Dynamics ที่สามารถจำลองการเคลื่อนที่ของระบบโมเลกุลน้ำ (Water Cluster) แบบ 2 มิติ โดยจะใช้ Interaction Potentail โมเดลไหนก็ได้

# บทที่ 3

# พลวัตเชิงโมเลกุลแบบแอบ อินิชิโอ

## 3.1 ทำไมต้อง *Ab Initio* Molecular Dynamics

เทคนิคการจำลองแบบ Molecular Dynamics (MD) แบบดั้งเดิมหรือ Classical MD นั้นจะใช้ Potential ที่ได้มาจากการใช้ข้อมูลเชิงการทดลอง (Empirical Data) หรือจากการคำนวณ Electronic Structure และหัวใจสำคัญของ MD นั้นก็คือสมการที่ใช้ในการอธิบายอันตรกิริยาระหว่างอะตอม (Interatomic Interactions) โดยอันตรกิริยาที่เกิดขึ้นทั้งหมดนั้นเราสามารถแบ่งออกได้เป็นหลาย ๆ ส่วน คือ

- Two-Body Contribution
- Three-Body Contribution
- Many-Body Contribution
- Long-Range Interaction
- Short-Range Interaction
- เทอมอื่น ๆ

จุดเริ่มต้นของเทคนิคการจำลองทางเคมีคอมพิวเตอร์แบบใหม่ที่เกิดขึ้นมาจากการนำวิธี MD และ Electronic Structure มารวมกันนั้นเรียกว่า *ab initio* Molecular Dynamics (AIMD) ซึ่งอาจจะมีชื่อเรียกอื่น ๆ ที่เราอาจจะคุ้นเคยกันมาบ้าง เช่น

- Car-Parrinello Molecular Dynamics
- Hellmann-Feynman Molecular Dynamics
- First Principles Molecular Dynamics
- Quantum Chemical Molecular Dynamics

- On-The-Fly Molecular Dynamics
- Direct Molecular Dynamics
- Potential-Free Molecular Dynamics
- Quantum Molecular Dynamics

แต่ไอเดียพื้นฐานที่เป็นหัวใจสำคัญของวิธีการคำนวณแบบ AIMD ทุกวิธีนั้นก็คือการคำนวณแรง (Force) ที่กระทำระหว่างอะตอมโดยการใช้วิธี Electronic Structure ในแต่ละ Step ของการคำนวณ MD

การประยุกต์ใช้วิธี AIMD นั้นกว้างขวางมาก ๆ โดยเฉพาะในด้านวัสดุศาสตร์และเคมี จะเห็นได้จากจำนวนงานบทความงานวิจัยเกี่ยวกับ AIMD ที่ได้รับการตีพิมพ์เพิ่มมากขึ้นเรื่อย ๆ ทุกปี ซึ่งจุดเริ่มต้นนั้นก็มาจากเปเปอร์ของ Car และ Parrinello ที่ตีพิมพ์ในปี 1985 นั่นคือ “Unified Approach for Molecular Dynamics and Density-Functional Theory” ที่ทำให้งานวิจัยทางด้านนี้นั้นได้รับความสนใจ

อย่างไรก็ตาม ถึงแม้ว่าวิธี AIMD นั้นจะทำให้การคำนวณ MD นั้นมีความแม่นยำเพิ่มมากขึ้น แต่ว่าราคาที่นักคำนวณจะต้องจ่ายก็คือความสิ้นเปลืองในการคำนวณ (Computational Cost) ในการนำ MD ไปผสมรวมกับวิธี *ab initio* นั่นก็คือความสัมพันธ์ระหว่าง Length และ Relaxation Time ที่เราสามารถรันการคำนวณด้วยแบบจำลอง AIMD นั้นสั้นมาก ๆ เมื่อเทียบกับวิธี MD ทั่วไป (สำหรับระบบโมเลกุลเดียวกัน) ถึงแม้ว่าข้อเสียของวิธี AIMD นั้นคือใช้เวลาในการคำนวณที่นานกว่า MD เยอะมาก ๆ แต่ว่าเราก็อย่าลืมไปว่าวิธี AIMD นั้นมีข้อดีอีกหลายข้อเลยที่เราจะไม่พูดถึงก็ไม่ได้ ข้อดีอย่างแรกก็คือวิธี AIMD สามารถให้ผลการคำนวณที่สอดคล้องกับ Physical Picture จริง ๆ ของระบบที่เราจำลอง ข้อดีอีกอย่างก็คือวิธี AIMD นั้นสามารถช่วยให้เราสามารถจำลองปรากฏการณ์ของระบบโมเลกุลที่ไม่สามารถเกิดขึ้นได้ในการจำลองด้วยวิธี MD

จริง ๆ แล้วก่อนที่จะมีการพัฒนาวิธี AIMD ขึ้นมานั้น ในอดีตก็มีวิธีที่คล้าย ๆ กันเรียกว่า “Classical Trajectory Calculation” ซึ่งมีจุดเริ่มต้นคือคำนวณระบบ Gas Phase ด้วยวิธี MD เพื่อศึกษา “Global” Potential Energy Surface (PES) หรือพื้นผิวพลังงานศักย์ แล้วก็ตามด้วยการคำนวณ Dynamical Evolution ของระบบโดยการใช้ Classical Mechanics หรือ Quantum Mechanics หรือ Semi/Quasiclassical Approximations ซึ่งในกรณีที่ใช้วิธี Classical Mechanics ในการอธิบาย Dynamics ของระบบโมเลกุลนั้นมีอุปสรรคก็คือขนาดของระบบ กล่าวคือ ยิ่งระบบมีขนาดใหญ่ การใช้วิธี Classical Mechanics นั้นก็จะยิ่งทำได้ยาก (สิ้นเปลืองการคำนวณ) นั่นก็เพราะว่าระบบที่มี $N$ อะตอมนั้นก็จะมีจำนวนดีกรีของความอิสระ (Degree of Freedom) เท่ากับ $3N-6$ ที่จะเป็นตัวกำหนดขนาดของ PES แล้วถ้าหากว่าเราใช้จำนวน Discretization Points เช่น 10 Points ต่อ Coordinate นั่นคือเรามีจำนวนการคำนวณ Electronic Structure ที่จะต้องคำนวณเท่ากับ $10^{3N-6}$ เพื่อที่ทำการ Mapping เพื่อให้ได้ Global PES ของระบบโมเลกุลของเราออกมา ดังนั้นความสิ้นเปลืองของวิธีแรกนั้นเท่ากับ $10^N$ ซึ่งเพิ่มตามขนาดของระบบ ซึ่งเราเรียกปัญหานี้ว่า “Dimensionality Bottleneck”

คำถามคือ “ถ้าหากอยากจะรู้ว่า AIMD สิ้นเปลืองแค่ไหน เราจะต้องคำนึงถึงอะไรบ้าง?” ในการตอบคำถามนี้ผมขอเริ่มด้วยการยกตัวอย่างต่อไปนี้ สมมติว่าเรามี Trajectory ของการคำนวณ MD ที่มีจำนวนทั้งหมด $10^M$ Steps (ก็คือมีทั้งหมด $10^M$ Configurations) นั่นคือจะต้องมีการคำนวณ Electronic Structure ทั้งหมด $10^M$ ครั้ง ถ้าหากว่ามีจำนวน Independent Trajectory ทั้งหมด $10^n$ Trajectories ที่

จำเป็นที่จะต้องคำนวณเพื่อทำการเฉลี่ย Initial Conditions ดังนั้นจึงมีการคำนวณ AIMD ทั้งหมด $10^{M+n}$ การคำนวณที่จะต้องทำการรัน ลำดับสุดท้าย ถ้าหากว่าเราจะต้องทำการคำนวณ Single-Point Electronic Structure เพื่อคำนวณ Global PES และแต่ละการคำนวณของ AIMD นั้นใช้เวลา CPU Time เท่ากัน จากข้อมูลทั้งหมดที่เราอ้างขึ้นมาตามสถานการณ์ความเป็นจริงนั้น เราจะสรุปได้ว่าการใช้ AIMD ในการคำนวณ Global PES นั้นจะมีความสิ้นเปลืองอยู่ที่ประมาณ $10^{3N-6-M-n}$ ประเด็นสำคัญก็คือว่า สำหรับระบบที่มี $M$ และ $n$ คงที่และไม่ขึ้นกับ $N$ นั้น การคำนวณ AIMD จะมี Advantage เป็นแบบ “On-The-Fly” ซึ่งจะมีความสิ้นเปลืองของวิธีคือ $10^N$ เพิ่มขึ้นตามขนาดของระบบ

อย่างไรก็ตาม ความสิ้นเปลืองของวิธี AIMD ที่ประมาณ $10^N$ นั้นก็ยังเยอะอยู่ดี ดังนั้นจึงได้มีการพัฒนาเทคนิคต่าง ๆ ขึ้นมาเพื่อใช้ในการลดจำนวน Degrees of Freedom แต่ว่าเทคนิคเหล่านั้นก็เป็นการใช้ Approximations (การประมาณ) เสียส่วนใหญ่ นั่นจึงทำให้ความถูกต้องของ AIMD นั้นลดลงด้วย

ในบทนี้ผมจะพาผู้อ่านทุกท่านไปทำความรู้จักกับวิธี AIMD ซึ่งเป็นภาพรวมกว้าง ๆ โดยเราจะเริ่มต้นกันด้วยสมการ Schrödinger แล้วก็จะมีการพูดถึง Classical MD, Ehrenfest MD, Born-Oppenheimer, และ Car-Parrinello MD ซึ่งเป็นวิธีที่ได้มาจากวิธี Time-Dependent Mean-Field Approach ซึ่งได้หลังจากการที่เราทำการแยก Degrees of Freedom ของนิวเคลียสกับอิเล็กตรอนออกจากกัน นอกจากนี้ยังมีอีกหนึ่งเรื่องสำคัญที่ผู้อ่านจะได้ศึกษานั่นคือการคำนวณแรง (Force) ของวิธีต่าง ๆ ด้วย ซึ่งหนึ่งในวิธีที่ถูกนำมาใช้ในการคำนวณ Force ที่ได้รับความนิยมเป็นอย่างมากนั่นคือ Density Functional Theory (DFT) ดังนั้นเราจึงสามารถเรียกวิธีที่ใช้ในการจำลอง AIMD ที่มีความถูกต้องของการคำนวณ Electronic Structure ในแต่ละ Step ของ MD ได้ว่า Density Functional Theory-based Molecular Dynamics หรือ DFT-MD

## 3.2 จาก MD สู่ *Ab Initio* MD

เรามาเริ่มกันที่สมการ Time-Dependent Schrödinger Equation ซึ่งเราต่างก็ทราบกันดีอยู่แล้วว่าเป็นสมการที่ฟังก์ชันคลื่น (Wavefunction) นั้นเป็นฟังก์ชันที่ขึ้นกับตำแหน่งของอนุภาคที่เราสนใจและเวลา โดยอนุภาคที่เราสนใจในที่นี้ก็คืออิเล็กตรอนและนิวเคลียส

$$i\hbar\frac{\partial}{\partial t}\Phi\left(\mathbf{r}_i,\mathbf{R}_I;t\right)=H\Phi\left(\mathbf{r}_i,\mathbf{R}_I;t\right) \tag{3.2.1}$$

ซึ่งมี Position Representation ที่เชื่อมโยงกับ Standard Hamiltonian ซึ่งมีนิยามดังต่อไปนี้ (มี 5 เทอมรวมเข้าด้วยกัน)

$$\begin{aligned} H &= -\sum_I \frac{\hbar^2}{2M_I}\nabla_I^2 - \sum_i \frac{\hbar^2}{2m_\mathrm{e}}\nabla_i^2 + \sum_{i<j}\frac{\mathrm{e}^2}{|\mathbf{r}_i-\mathbf{r}_j|} - \sum_{I,i}\frac{\mathrm{e}^2 Z_I}{|\mathbf{R}_I-\mathbf{r}_i|} + \sum_{I<J}\frac{\mathrm{e}^2 Z_I Z_J}{|\mathbf{R}_I-\mathbf{R}_J|} \\ &= -\sum_I \frac{\hbar^2}{2M_I}\nabla_I^2 - \sum_i \frac{\hbar^2}{2m_\mathrm{e}}\nabla_i^2 + V_{\mathrm{n-e}}\left(\mathbf{r}_i,\mathbf{R}_I\right) \\ &= -\sum_I \frac{\hbar^2}{2M_I}\nabla_I^2 + H_\mathrm{e}\left(\mathbf{r}_i,\mathbf{R}_I\right) \end{aligned} \tag{3.2.2}$$

สำหรับระดับของความอิสระ (Degrees of Freedom) ของอิเล็กทรอนิกส์ $\mathbf{r}_i$ กับของนิวเคลียร์ $\mathbf{R}_I$ แล้ว เราก็จะใช้ Atomic Units (a.u.) เพื่อช่วยให้สมการต่าง ๆ ของเรานั้นดูง่ายและชัดเจนมากขึ้น ดังนั้นเราจะสนใจเทอมที่เป็นอันตรกิริยาระหว่าง อิเล็กตรอน-อิเล็กตรอน (Electron-Electron), อิเล็กตรอน-นิวเคลียร์ (Electron-Nuclear), และนิวเคลียร์-นิวเคลียร์แบบคูลอมป์ (Nuclear-Nuclear Coulomb) เป็นพิเศษ

เป้าหมายของหัวข้อนี้ก็คือการพิสูจน์ที่มาของ Classical Molecular Dynamics โดยเริ่มจากสมการคลื่นของ Schrödinger ซึ่งถึงตรงนี้แล้ว เราจะทำการแยกฟังก์ชันคลื่นรวมของระบบโมเลกุลของเราออกเป็น 2 พาร์ท $\Phi\left(\mathbf{r}_i,\mathbf{R}_I;t\right)$ นั่นก็คือพาร์ทที่ขึ้นกับ Nuclear Coordinates และพาร์ทที่ขึ้นกับ Electronic Coordinates ซึ่งจะสามารถเขียนให้อยู่ในรูปที่ง่ายที่สุดได้โดยใช้ ผลคูณระหว่าง Ansatz ดังนี้

$$\Phi\left(\mathbf{r}_i,\mathbf{R}_I;t\right) \approx \Psi\left(\mathbf{r}_i;t\right)\chi\left(\mathbf{R}_I;t\right)\exp\left[\frac{i}{\hbar}\int_{t_0}^{t} dt'\tilde{E}_\mathrm{e}\left(t'\right)\right] \tag{3.2.3}$$

โดยที่ Nuclear Wavefunction และ Electronic Wavefunction นั้นถูกแยกออกจากกันอย่างสิ้นเชิงและถูก Normalized ในแต่ละ Time Step ด้วย ดังนี้ $\langle\chi;t \mid \chi;t\rangle = 1$ และ $\langle\Psi;t \mid \Psi;t\rangle = 1$ ตามลำดับ นอกจากนี้แล้วเรายังมีการกำหนดพารามิเตอร์อีกตัวหนึ่งขึ้นมานั่นคือ Phase Factor ดังนี้

$$\tilde{E}_\mathrm{e} = \int d\mathbf{r}d\mathbf{R}\Psi^\star\left(\mathbf{r}_i;t\right)\chi^\star\left(\mathbf{R}_I;t\right)H_\mathrm{e}\Psi\left(\mathbf{r}_i;t\right)\chi\left(\mathbf{R}_I;t\right) \tag{3.2.4}$$

เพื่อที่จะทำให้สมการสุดท้ายที่เราได้ออกมานั้นมีหน้าตาที่ดูสั้นและกระชับ ดังนี้ $\int d\mathbf{r}d\mathbf{R}$ ซึ่งเป็นการคำนวณ Integration ทั่วทั้งหมดโมเลกุล $i = 1,\ldots$ และ $I = 1,\ldots$ สำหรับตัวแปร $\mathbf{r}_i$ และ $\mathbf{R}_I$ ตามลำดับ

นอกจากนี้แล้วเราต้องทราบกันไว้ด้วยว่า Product Ansatz (ที่ไม่รวม Phase Factor) ตามด้านบนนั้นมีความแตกต่างจาก Ansatz ของ Born-Oppenheimer ซึ่งเป็นการแยกพาร์ทที่คำนวณได้เร็วกว่าและพาร์ทที่คำนวณได้ช้ากว่าออกจากกัน ดังนี้

$$\Phi_{\mathrm{BO}}\left(\mathbf{r}_i, \mathbf{R}_I; t\right) = \sum_{k=0}^{\infty} \tilde{\Psi}_k\left(\mathbf{r}_i, \mathbf{R}_I\right) \tilde{\chi}_k\left(\mathbf{R}_I; t\right) \tag{3.2.5}$$

ถ้าหากว่าเรานำสมการ Separation Ansatz ที่ (3.2.3) แทนเข้าไปในสมการที่ (3.2.1) และสมการที่ (3.2.2) (หลังจากที่เราทำการคูณทางด้านซ้ายของสมการด้วย $\langle\Psi|$ และ $\langle\chi|$ และทำให้สอดคล้องตามเงื่อนไขของกฎอนุรักษ์พลังงาน $d\langle H\rangle/dt \equiv 0$ แล้ว) เราจะได้ความสัมพันธ์ต่อไปนี้

$$i\hbar\frac{\partial\Psi}{\partial t} = -\sum_i \frac{\hbar^2}{2m_{\mathrm{e}}}\nabla_i^2\Psi + \left\{\int d\mathbf{R}\chi^\star\left(\mathbf{R}_I; t\right) V_{\mathrm{n-e}}\left(\mathbf{r}_i, \mathbf{R}_I\right) \chi\left(\mathbf{R}_I; t\right)\right\}\Psi \tag{3.2.6}$$

$$i\hbar\frac{\partial\chi}{\partial t} = -\sum_I \frac{\hbar^2}{2M_I}\nabla_I^2\chi + \left\{\int d\mathbf{r}\Psi^\star\left(\mathbf{r}_i; t\right) H_{\mathrm{e}}\left(\mathbf{r}_i, \mathbf{R}_I\right) \Psi\left(\mathbf{r}_i; t\right)\right\}\chi \tag{3.2.7}$$

ซึ่งเซตของสมการที่พัวพันกัน (Coupled Equations) ตามด้านบนนี้เป็นตัวกำหนด Time-Dependent Self-Consistent Field (TDSCF) ที่ได้มีการเสนอไว้เมื่อนานมาแล้วโดย Paul Dirac ในช่วงปี ค.ศ. 1930

## 3.3 มาเจาะลึก Classical Molecular Dynamics

ขั้นตอนต่อไปในการพิสูจน์ Classical Molecular Dynamics ก็คือการประมาณและกำหนดให้นิวเคลียอของอะตอมนั้นเป็นอนุภาคจุด (Point Particle) ซึ่งเราสามารถทำได้โดยการใช้พิสูจน์ Classical Mechanics ออกมาจาก Quantum Mechanics โดยเราจะเริ่มด้วยการเขียนฟังก์ชันคลื่น (Wavefunction) ใหม่ ดังนี้

$$\chi\left(\mathbf{R}_I; t\right) = A\left(\mathbf{R}_I; t\right)\exp\left[iS\left(\mathbf{R}_I; t\right)/\hbar\right] \tag{3.3.1}$$

โดยเราเขียนในเทอมของ Amplitude Factor $A$ และ Phase $S$ ซึ่งแฟคเตอร์ทั้งสองแฟคเตอร์นี้จะถูกพิจารณาเฉพาะค่าจริง (Real) เท่านั้น $(A > 0)$ และหลังจากที่เราทำการแปลง Nuclear Wavefunction (สมการที่ (3.2.7)) และทำการแยก Real Part กับ Imaginary Part ออกจากกัน เราจะได้ TDSCF สำหรับนิวเคลียสดังต่อไปนี้

$$\frac{\partial S}{\partial t} + \sum_I \frac{1}{2M_I}\left(\nabla_I S\right)^2 + \int d\mathbf{r}\Psi^\star H_{\mathrm{e}}\Psi = \hbar^2\sum_I \frac{1}{2M_I}\frac{\nabla_I^2 A}{A} \tag{3.3.2}$$

$$\frac{\partial A}{\partial t} + \sum_I \frac{1}{M_I}\left(\nabla_I A\right)\left(\nabla_I S\right) + \sum_I \frac{1}{2M_I}A\left(\nabla_I^2 S\right) = 0 \tag{3.3.3}$$

ซึ่งเป็นการเขียนสมการเดิมโดยการใช้ตัวแปรใหม่สองอันก็คือ $A$ และ $S$ และเราเรียกสมการที่ (3.3.2) และ (3.3.3) นี้ว่า "Quantum Fluid Dynamical Representation" ซึ่งเราสามารถสมการทั้งสองอันนี้ไปใช้ในการแก้ Time-Dependent Schrödinger Equation นอกจากนี้เรายังสามารถเขียน $A$ ใหม่ได้ให้อยู่ในรูปของ Continuity Equation ได้โดยการใช้คุณสมบัติ Identification ของ Nuclear Density $|\chi|^2 \equiv A^2$ ซึ่งสามารถคำนวณมาได้จากสมการที่ (3.3.1) โดยสมการ Continuity Equation นี้จะไม่ขึ้นกับ $\hbar$ และยังทำให้มีการอนุรักษ์ Particle Probability $|\chi|^2$ อีกด้วย

เมื่อเรามีการใช้ Transformation ของโมเมนตัมดังต่อไปนี้มาช่วย

$$\mathbf{P}_I \equiv \nabla_I S \tag{3.3.4}$$

เราจะสามารถเขียนสมการการเคลื่อนที่ของนิวตัน $\dot{\mathbf{P}}_I = -\nabla_I V(\mathbf{R}_I)$ ได้ดังต่อไปนี้

$$\frac{d\mathbf{P}_I}{dt} = -\nabla_I \int d\mathbf{r} \Psi^\star H_\mathrm{e} \Psi \tag{3.3.5}$$

$$M_I \ddot{\mathbf{R}}_I(t) = -\nabla_I \int d\mathbf{r} \Psi^\star H_\mathrm{e} \Psi \tag{3.3.6}$$

$$= -\nabla_I V_\mathrm{e}^\mathrm{E}(\mathbf{R}_I(t)) \tag{3.3.7}$$

ดังนั้น นิวเคลียสนั้นจะเคลื่อนที่ไปตามหลักการของ Classical Mechanics ท่ามกลาง Effect Potentail $V_\mathrm{e}^\mathrm{E}$ ซึ่งมาจากอิเล็กตรอน โดย Potential ดังกล่าวนี้เป็นฟังก์ชันของ Nuclear Positions ณ เวลา $t$ ซึ่งได้มาจากการเฉลี่ย $H_\mathrm{e}$ จาก Degrees of Freedom ทั้งหมด เช่น คำนวณค่า Quantum Expectation Value $\langle \Psi | H_\mathrm{e} | \Psi \rangle$ ในขณะที่เราบังคับให้ตำแหน่งของนิวเคลียสนั้นถูกตรึง (Fixed) อยู่กับที่นั่นคือ $\mathbf{R}_I(t)$

อย่างไรก็ตาม เรายังคงมีฟังก์ชันคลื่นของนิวเคลียส (Nuclear Wavefunction) เหลืออยู่ในสมการ TDSCF สำหรับ Degrees of Freedom ของอิเล็กตรอนและเทอมนี้ควรจะต้องถูกแทนที่ด้วยตำแหน่งของนิวเคลียสเพื่อที่ว่าสมการ TDSCF นั้นจะขึ้นอยู่กับนิวเคลียสเพียงอย่างเดียว ในกรณีนี้เราจะทำการแทนที่ Nuclear Density $|\chi(\mathbf{R}_I; t)|^2$ ในสมการก่อนหน้านี้โดยมีเงื่อนไขของลิมิตว่า $\hbar \to 0$ โดยแทนที่ด้วยผลคูณของฟังก์ชันเดลต้า (Delta Functions) $\prod_I \delta(\mathbf{R}_I - \mathbf{R}_I(t))$ ที่มีตำแหน่งจุดกึ่งกลางของฟังก์ชันอยู่ที่ตำแหน่งของนิวเคลียส ณ ขณะใดขณะหนึ่ง (Instantaneous Positions) $\mathbf{R}_I(t)$ ตามสมการที่ (3.3.6) ซึ่งเราจะได้สมการดังต่อไปนี้ (สำหรับ Position Operator)

$$\int d\mathbf{R}^\star(\mathbf{R}_I; t)\, \mathbf{R}_I \chi(\mathbf{R}_I; t) \xrightarrow{\hbar \to 0} \mathbf{R}_I(t) \tag{3.3.8}$$

โดย Classical Limit อันนี้จะนำไปสู่สมการ Time-Dependent Wave Equation สำหรับอธิบายอิเล็กตรอนดังต่อไปนี้

$$i\hbar\frac{\partial\Psi}{\partial t} = -\sum_i \frac{\hbar^2}{2m_{\mathrm{e}}}\nabla_i^2\Psi + V_{\mathrm{n-e}}\left(\mathbf{r}_i, \mathbf{R}_I(t)\right)\Psi \tag{3.3.9}$$

$$= H_{\mathrm{e}}\left(\mathbf{r}_i, \mathbf{R}_I(t)\right)\Psi\left(\mathbf{r}_i, \mathbf{R}_I; t\right) \tag{3.3.10}$$

ซึ่งสมการด้านบนนี้มีความซับซ้อนเพราะว่าเป็นสมการที่มันขึ้นอยู่กับตัวของมันเอง (Self-Consistently) นอกจากนี้แล้ว $H_{\mathrm{e}}$ และ $\Psi$ นั้นจะขึ้นต่อกันแบบเชิงพารามิเตอร์หรือเรียกว่าอิงพารามิเตอร์ก็ได้ (Parametrically) ตามตำแหน่งของนิวเคลียส $\mathbf{R}_I(t)$ ที่เวลา $t$ โดยผ่าน $V_{\mathrm{n-e}}\left(\mathbf{r}_i, \mathbf{R}_I(t)\right)$ นั่นหมายความว่าทั้ง Classical และ Quantum Degress of Freedom นั้นได้ถูกรวมเข้าด้วยกันแล้ว

สำหรับวิธีที่เกี่ยวข้องกับการแก้สมการที่ (3.3.6) และ (3.3.10) นั้นมีชื่อเรียกว่า “Ehrenfest Molecular Dynamics” เพื่อเป็นเกียรติให้กับ Ehrenfest ผู้ที่เป็นคนแรกที่สามารถหาวิธีแก้ปัญหาสำหรับคำถามที่ว่า “เราจะสามารถพิสูจน์ Newtonian Classical Dynamics จากสมการคลื่นของ Schrödinger ได้อย่างไร?” ซึ่งนำไปสู่การพัฒนาวิธีการแบบผสม (Hybrid) นั่นก็คือมีเพียงแค่นิวเคลียสเท่านั้นที่ถูกบังคับให้มีพฤติกรรมที่ทำตัวคล้ายกับอนุภาค Classical Particles ในขณะที่อิเล็กตรอนนั้นยังถูกอธิบายด้วยวิธีทางควอนตัม

ตามที่ได้กล่าวไว้ในตอนต้นของหัวข้อนี้แล้วว่าวิธี MD นั้นมีปัญหาอย่างหนึ่งที่หลีกเลี่ยงไม่ได้ก็คือ Dimensionality Bottleneck ซึ่งจะเพิ่มมากขึ้นตาม Degrees of Freedom ของนิวเคลียส (จำนวนอะตอม) ซึ่งหนึ่งในวิธีที่เป็นทางออกของปัญหานี้ก็คือการทำการประมาณ Global PES ซึ่งมีสมการคือ

$$V_{\mathrm{e}}^{\mathrm{E}} \approx V_{\mathrm{e}}^{\mathrm{approx}}\left(\mathbf{R}_I\right) = \sum_{I=1}^{N} v_1\left(\mathbf{R}_I\right) + \sum_{I<J}^{N} v_2\left(\mathbf{R}_I, \mathbf{R}_J\right) + \sum_{I<J<K}^{N} v_3\left(\mathbf{R}_I, \mathbf{R}_J, \mathbf{R}_K\right) + \cdots \tag{3.3.11}$$

โดยเขียนในรูปของการกระจายของเทอมที่เกิดจาก Contribution แบบ Many-Body (Truncated Expansion) และเราจะทำการแทน Degrees of Freedom เชิงอิเล็กทรอนิกส์ด้วย Interaction Potentials $(v_n)$ ซึ่งจะทำให้เทอมที่เป็น Degrees of Freedom นั้นหายไป ดังนั้นสิ่งที่เราจะได้ออกมาก็คือสมการที่เป็นลูกผสมระหว่างปัญหาทางควอนตัมและแบบคลาสสิกนั้นจะถูกลดรูปให้เหลือเป็นเพียงแค่ปัญหาแบบคลาสสิกเท่านั้น (ปัญหาแบบควอนตัมหายไปแล้ว) ซึ่งจะได้ว่า Classical Molecular Dynamics แบบที่เราต้องการนั้นจะมีหน้าตาคือ

$$M_I\ddot{\mathbf{R}}_I(t) = -\nabla_I V_{\mathrm{e}}^{\mathrm{approx}}\left(\mathbf{R}_I(t)\right) \tag{3.3.12}$$

เราสามารถแบ่งประเภทของ AIMD ได้โดยแบ่งตามวิธีการที่เราใช้ในการรวมการคำนวณโครงสร้างเชิงอิเล็กทรอนิกส์กับการจำลองพลวัตโมเลกุลเข้าด้วยกัน โดยเราสามารถแบ่งออกได้เป็น 3 วิธี ดังนี้

1. พลวัตเชิงโมเลกุลแบบเออเรนเฟสต์ (Ehrenfest Molecular Dynamics)
2. พลวัตเชิงโมเลกุลแบบบอร์น-ออปเพนไฮเมอร์ (Born-Oppenheimer Molecular Dynamics)
3. พลวัตเชิงโมเลกุลแบบคาร์-พาร์ริเนลโล (Car-Parrinello Molecular Dynamics)

## 3.4 พลวัตเชิงโมเลกุลแบบเออเรนเฟสต์

ตามที่ได้อธิบายไว้ในหัวข้อที่ผ่านมาว่าปัญหาอย่างหนึ่งของ MD ก็คือ Dimensionality Bottleneck ซึ่งเราสามารถแก้ปัญหานี้ได้โดยการคำนวณประมาณ Global Potential Energy Surface ตามสมการที่ (3.3.11) หรือทำการลดจำนวนของ Degrees of Freedom นอกจากนี้ ยังมีอีกวิธีหนึ่งที่เราสามารถแก้ปัญหานี้ได้เช่นกันนั่นก็คือทำการแก้สมการ Time-Dependent Self-Consistent Field (TDSCF) โดยการใช้ Classical Nuclei Approxiation เข้ามาช่วย (สมการที่ (3.3.6) และ (3.3.10)) โดยเราสามารถแก้ชุดสมการดังต่อไปพร้อม ๆ กันเพื่อคำนวณแรงของเออเรนเฟสต์ (Ehrenfest Force)

$$M_I\ddot{\mathbf{R}}_I(t) = -\nabla_I \int d\mathbf{r}\Psi^\star H_\mathrm{e}\Psi \tag{3.4.1}$$
$$= -\nabla_I \langle \Psi | H_\mathrm{e} | \Psi \rangle \tag{3.4.2}$$
$$= -\nabla_I \langle H_\mathrm{e} \rangle \tag{3.4.3}$$
$$= -\nabla_I V_\mathrm{e}^\mathrm{E} \tag{3.4.4}$$
$$i\hbar\frac{\partial\Psi}{\partial t} = \left[-\sum_i \frac{\hbar^2}{2m_\mathrm{e}}\nabla_i^2 + V_{\mathrm{n-e}}\left(\mathbf{r}_i, \mathbf{R}_I(t)\right)\right]\Psi \tag{3.4.5}$$
$$= H_\mathrm{e}\Psi \tag{3.4.6}$$

แล้วเราก็พบว่าเราไม่จำเป็นที่จะต้องทราบ Potential Energy Surface (PES) ก็สามารถแก้สมการ Time-Dependent Electronic Schrödinger ได้แบบ “On-The-Fly” (หมายความว่า อยากแก้สมการเมื่อไหร่ก็ได้) ซึ่งทำให้เรานั้นสามารถคำนวณ Ehrenfest Force จาก $\nabla_I \langle H_\mathrm{e} \rangle$ สำหรับแต่ละคอนฟิกูเรชั่น $\mathbf{R}_I(t)$ ของโมเลกุลที่ได้มาจาก Trajectory ที่จำลองมาจากวิธี Molecular Dynamics ได้ด้วย (แรงชนิดนี้มีชื่อเรียกอีกชื่อว่า “Hellmann-Feynman Forces”

นอกจากนี้แล้ว ยังมีสมการของ Equations of Motion ที่อยู่ในรูปของ Adiabatic Basis และ Time-Dependent Expansion Coefficients อีกด้วย ซึ่งมีหน้าตาดังนี้

$$M_I\ddot{\mathbf{R}}_I(t) = -\sum_k |c_k(t)|^2 \nabla_I E_k - \sum_{k,l} c_k^\star c_l \left(E_k - E_l\right) \mathbf{d}_I^{kl} \tag{3.4.7}$$

$$i\hbar\dot{c}_k(t) = c_k(t)E_k - i\hbar \sum_{I,l} c_l(t)\dot{\mathbf{R}}_I \mathbf{d}_I^{kl} \tag{3.4.8}$$

ซึ่งมี Coupling Term ดังนี้

$$\mathbf{d}_I^{kl}\left(\mathbf{R}_I(t)\right) = \int d\mathbf{r}\Psi_k^\star \nabla_I \Psi_l \tag{3.4.9}$$

แล้วก็มี Property $\mathbf{d}_I^{kk} \equiv \mathbf{0}$

ดังนั้น เราอาจจะมองว่าวิธีการของ Ehrenfest นั้นก็คือเป็นการรวม Non-Adiabatic Transitions ระหว่าง Electronic States ที่แตกต่างกัน นั่นคือ $\Psi_k$ และ $\Psi_l$ โดยรวมเข้ามาไว้ใน Framework ของ Mean-Field (TDSCF) Approximation นั่นเอง นอกจากนี้แล้วเรายังสามารถกำหนดเงื่อนไข (Restriction) สำหรับ Electronic State แต่ละอันได้อีกด้วย (ซึ่งกรณีส่วนใหญ่นั้นจะเป็นการอธิบาย $\Psi_0$ ที่สภาวะพื้น) ซึ่งก็จะทำให้เราได้สมการที่เป็นสำหรับกรณีพิเศษของสมการที่ (3.4.4) และ (3.4.6) ดังนี้

$$M_I\ddot{\mathbf{R}}_I(t) = -\nabla_I \left\langle \Psi_0 \left| H_\mathrm{e} \right| \Psi_0 \right\rangle \tag{3.4.10}$$

$$i\hbar\frac{\partial \Psi_0}{\partial t} = H_\mathrm{e}\Psi_0 \tag{3.4.11}$$

โดยที่ $H_\mathrm{e}$ คือ Time-Dependent Hamiltonian ของ Nuclear Coordinates $(\mathbf{R}_I(t))$

Ehrenfest Molecular Dynamics (MD) นั้นถือได้ว่าเป็นวิธี “On-The-Fly” MD ที่เก่าที่สุดและถูกใช้ในการจำลองระบบที่มีจำนวน Degrees of Freedom น้อย ๆ ดังนั้นเราจึงมักไม่ค่อยเห็นงานวิจัยที่นำ Ehrenfest MD มาใช้ในการศึกษาระบบที่มี Degrees of Freedom เยอะ ๆ เช่น Condensed Matter

## 3.5 พลวัตเชิงโมเลกุลแบบบอร์น-ออปเพนไฮเมอร์

อีกหนึ่งวิธีที่เราสามารถใช้ในการรวมการคำนวณ Electronic Structure เข้ากับ Molecular Dynamics ได้นั้นก็คือการแก้สมการของ Electronic Structure ตรง ๆ (แบบ Static) ในแต่ละ Step ของการจำลอง MD โดยการใช้ตำแหน่งของอะตอมในโมเลกุลซึ่งมีตำแหน่งที่แน่นอน ดังนั้น แทนที่เราจะต้องแก้ปัญหาแบบ Time-Dependent นั้นก็จะกลายเป็นว่าเราแก้ปัญหาแบบ Time-Independent แทน พูด

ง่าย ๆ ก็คือเราคำนวณแรง (Force) ที่กระทำต่ออะตอมแต่ละอะตอมในโมเลกุลในแต่ละ Step เพื่อใช้ในการ Propagation ของคอนฟิกูเรชั่นของโมเลกุลใน Step ต่อ ๆ ไป แน่นอนว่าสมการที่เราจะต้องแก้นั้นก็คือ Time-Independent Schrödinger Equation ดังนั้นเราจึงสรุปได้ว่าวิธีนี้นั้นก็เปรียบเสมือนกับ Time-Dependent Electronic Structure ของการเคลื่อนที่ของโมเลกุล (Nuclear Motion) ซึ่งจะไม่เหมือนกับกรณีของ Ehrenfest MD ซึ่งจะมีความ Intrinsic มากกว่า โดยเราเรียกวิธีการนี้ว่า Born-Oppenheimer Molecular Dynamics (BOMD) นั่นเอง ซึ่งมีสมการหลักดังต่อไปนี้

$$M_I \ddot{\mathbf{R}}_I(t) = -\nabla_I \min_{\Psi_0} \left\{ \langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle \right\} \tag{3.5.1}$$

$$E_0 \Psi_0 = H_\mathrm{e} \Psi_0 \tag{3.5.2}$$

โดยสมการด้านบนนี้สำหรับกรณีสถานะพื้น (Ground State) แล้วก็สิ่งที่ BOMD นั้นแตกต่างจาก Ehrenfest MD อย่างเห็นได้ชัดนั่นก็คือ Nuclear Equation of Motion ซึ่ง BOMD นั้นจะมีเงื่อนไขว่าค่าของ $\langle H_\mathrm{e} \rangle$ นั้นจะต้องมีค่าที่ต่ำที่สุดตามสมการที่ (3.5.1) แต่สำหรับกรณีของ Ehrenfest MD นั้นจะตรงข้ามกันนั่นก็คือฟังก์ชันคลื่น (Wavefunction) ที่เป็นเทอมที่ทำ Minimize $\langle H_\mathrm{e} \rangle$ นั้นยังคงมีค่าเท่าเดิมเมื่อเทียบกับ Motion ของนิวเคลียส (อะตอมในโมเลกุล) ตามสมการที่ (3.4.10)

ลำดับต่อไปก็คือส่วนขยายของสมการการเคลื่อนที่ของ BOMD นั่นก็คือการทำให้ BOMD นั้นสามารถใช้งานได้กับโมเลกุลที่อยู่ในสถานะกระตุ้น (Excited Electronic State) หรือ $\Psi_k$ ซึ่งมีชื่อเรียกแบบเฉพาะอีกชื่อว่า “Adiabatic Molecular Dynamics” (ถึงแม้ว่าจะไม่ควรเรียกก็ตาม)

นอกจากนี้เราพบว่าการที่เรามี Equation of Motion สำหรับ BOMD ที่สามารถใช้ในการอธิบายกรณีพิเศษของ Effective One-Particle Hamiltonian นั้นก็มีประโยชน์อย่างมากเช่นกัน เพราะว่าเราสามารถที่จะใช้ Hartree-Fock Approximation ที่ถูกกำหนดให้เป็น Variational Minimum สำหรับค่าของ Expectation Value ของพลังงาน ($\langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle$) โดยที่เรามีการกำหนดให้ Single Slater Determinant $\Psi_0 = \det \{\psi_i\}$ ซึ่งเปลี่ยนไปตาม Constraint หรือเงื่อนไขที่ว่า One-Particle Orbitals $\psi_i$ นั้นจะต้องมีความเป็น Orthonormal $\langle \psi_i \mid \psi_j \rangle = \delta_{ij}$

สำหรับ Constraint Minimization ของพลังงานของโมเลกุลโดยเทียบกับออร์บิทัลนั้นมีดังต่อไปนี้

$$\min_{\{\psi_i\}} \left\{ \langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle \right\} \Big|_{\{\langle \psi_i | \psi_j \rangle = \delta_{ij}\}} \tag{3.5.3}$$

ซึ่งสามารถเขียนใหม่ให้เป็น Lagrange’s Formalism ได้ดังนี้

$$\mathcal{L} = - \langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle + \sum_{i,j} \Lambda_{ij} \left( \langle \psi_i \mid \psi_j \rangle - \delta_{ij} \right) \tag{3.5.4}$$

โดยที่เรามีเทอม $\Lambda_{ij}$ ซึ่งเป็น Lagrangian Multipliers แล้วก็การแปรผันแบบที่ไม่มีข้อบังคับ (Unconstrained Variation) ของ Lagrangian อันนี้เมื่อเทียบกับออร์บิทัลนั้นมีค่าเป็น

$$\frac{\delta \mathcal{L}}{\delta \psi_i^\star} \stackrel{!}{=} 0 \tag{3.5.5}$$

ซึ่งนำไปสู่สมการ Hartree-Fock ที่เราทราบกันดี นั่นคือ

$$H_{\mathrm{e}}^{\mathrm{HF}} \psi_i = \sum_j \Lambda_{ij} \psi_j \tag{3.5.6}$$

ตามที่เราทราบกันนั่นก็คือ Diagonal Canonical Form $H_{\mathrm{e}}^{\mathrm{HF}} \psi_i = \epsilon_i \psi_i$ นั้นจะถูกคำนวณออกมาหลังจากที่เราทำ Unitary Transformation แล้ว ($H_{\mathrm{e}}^{\mathrm{HF}}$ คือ Effective One-Particle Hamiltonian) สำหรับสมการการเคลื่อนที่ (Equations of Motion) ที่เกี่ยวเนื่องกับสมการที่ (3.5.1) และ (3.5.2) มีดังนี้

$$M_I \ddot{\mathbf{R}}_I(t) = -\nabla_I \min_{\{\psi_i\}} \left\{ \left\langle \Psi_0 \left| H_{\mathrm{e}}^{\mathrm{HF}} \right| \Psi_0 \right\rangle \right\} \tag{3.5.7}$$

$$0 = -H_{\mathrm{e}}^{\mathrm{HF}} \psi_i + \sum_j \Lambda_{ij} \psi_j \tag{3.5.8}$$

สำหรับกรณีของ Hartree-Fock

ถ้าหากว่าเป็นกรณีอื่นที่ไม่ใช่ Hartree-Fock เช่น กรณีของ Hohenberg-Kohn-Sham Density Functional Theory เราก็สามารถใช้สมการที่ (3.5.7) และ (3.5.8) ได้เหมือนกัน แต่สิ่งที่จะไม่เหมือนกันนั่นก็คือ $H_{\mathrm{e}}^{\mathrm{HF}}$ นั้นจะต้องถูกแทนที่ด้วย Kohn-Sham Effective One-Particle Hamiltonian $H_{\mathrm{e}}^{\mathrm{KS}}$ แล้วก็แทนที่เราจะต้องทำ Diagonalization ของ One-Particle Hamiltonian เราก็สามารถที่จะใช้วิธีการที่คล้าย ๆ กันได้ นั่นก็คือการทำ Constraint Minimization ตามสมการที่ (3.5.3) โดยใช้เทคนิค Nonlinear Optimization

สำหรับการนำ BOMD ไปใช้งานนั้นในช่วงแรก ๆ BOMD มักจะถูกนำมาใช้ในการคำนวณร่วมกับวิธี Semiempirical แต่ว่าตั้งแต่ช่วงปี ค.ศ. 2000 เป็นต้นมานั้นก็ได้มีการนำ *ab initio* มา Implement[1] รวมเข้าไปในวิธี BOMD ซึ่งทำให้ได้รับความนิยมเรื่อยมาจนทำให้ BOMD นั้นได้รับความนิยมเป็นอย่างมาก โดยหนึ่งในจุดเปลี่ยนผ่านที่สำคัญของ BOMD ก็คือนำ Density Functional Theory (DFT) เข้ามาผสมกับ BOMD ซึ่งเป็น Framework ที่มีความสิ้นเปลืองที่ไม่สูงมากแต่ว่าให้ผลการคำนวณที่แม่นยำมาก

[1] Implement คือการทำให้สิ่ง ๆ นั้นเกิดขึ้น ถ้าเป็นในบริบทของการพัฒนาโปรแกรมก็คือการเขียนโปรแกรมของทฤษฎีนั้น ๆ

## 3.6 พลวัตเชิงโมเลกุลแบบคาร์-พาร์ริเนลโล

Car กับ Parrinello ได้เสนอวิธีอีกแบบหนึ่งที่ช่วยให้เราสามารถลดความสิ้นเปลืองในการคำนวณ Molecular Dynamics ซึ่งได้รวม Electronic Single State (การเคลื่อนที่ของอิเล็กตรอน) เข้าไว้ด้วย โดยเราเรียกวิธีนี้ว่า Car-Parrinello Molecular Dynamics (CPMD) ซึ่งถือว่าเป็นวิธีที่ 3 ของ *ab initio* Molecular Dynamics

ถ้าสรุปง่าย ๆ ก็คือว่าวิธี CPMD นั้นเป็นการรวมเอาข้อดีของวิธี Ehrenfest MD กับ BOMD เข้าไว้ด้วยกัน ซึ่งในวิธี Ehrenfest MD นั้น Time Scale และ Time Step นั้นจะถูกกำหนดด้วย Dynamics ของอิเล็กตรอน แล้วก็เนื่องจากว่า Motion ของอิเล็กตรอนนั้นเร็วกว่า Motion ของนิวเคลียสเยอะมาก ๆ ทำให้ขนาดของ Time Step ที่มากที่สุดที่จะเป็นไปได้นั้นจะต้องสอดคล้องตาม Motion ของอิเล็กตรอนเพื่อที่เราจะสามารถอินทิเกรตหรือแก้สมการ Equation of Motion ได้ แต่ว่าวิธี BOMD นั้นจะต้องกันข้ามกับวิธี Ehrenfest MD โดยสิ้นเชิงก็คือว่า BOMD นั้นจะไม่มีการพิจารณา Dynamics ของอิเล็กตรอนและไม่นำเอามาคิดรวมด้วย นั่นหมายความว่าในวิธี BOMD นั้นเราจะทำการอินทิเกรตและแก้สมการ Equation of Motion โดยใช้ Time Scale ที่อ้างอิงตาม Motion ของนิวเคลียส อย่างไรก็ตาม นั่นหมายความว่าสำหรับวิธี Ehrenfest MD นั้นเราไม่จำเป็นต้องแก้ปัญหาเชิง Electronic Structure เพราะว่าเราสามารถที่จะทำการ Propagate ฟังก์ชันคลื่นได้โดยการนำ Hamiltonian มากระทำกับฟังก์ชันคลื่นเริ่มต้น แต่ว่าสำหรับวิธี BOMD นั้นเราจะยังคงต้องแก้ปัญหา Electronic Structure โดยการแก้สมการหาคำตอบแบบ Self-Consistently ในแต่ละ Step ของการคำนวณ MD

อย่างไรก็ตาม ถ้าหากว่าเรามานั่งพิจารณาตัวทฤษฎีและอัลกอริทึมอย่างละเอียดแล้ว เราอาจจะพอตั้งข้อสังเกตได้ว่าถ้าหากว่าเราต้องการที่จะพัฒนาวิธีการที่เป็น “The Best of All Worlds Method” แล้วล่ะก็วิธีการนั้นก็ควรที่จะต้องเป็นไปตามเงื่อนไขต่อไปนี้

1. สามารถอินทิเกรตสมการ Equation of Motion ตาม Time Scale ที่ถูกกำหนดด้วย Nuclear Motion (ตำแหน่งของนิวเคลียส)
2. ในขณะเดียวกันนั้นก็ควรที่จะสามารถคำนวณ Dynamics ของอิเล็กตรอนให้ Smooth (Time-Evolution) มากที่สุดเท่าที่จะเป็นไปได้

ซึ่งข้อ 2 นั้นช่วยให้เราสามารถแก้ปัญหา Electronic Structure ได้โดยที่ไม่ต้องคำนวณ Diagonalization หรือทำ Minimization เพื่อนำคำตอบไปใช้ใน Step ต่อไปในระหว่างที่เราทำการคำนวณ MD และนี่จึงทำให้ CPMD นั้นเป็นวิธีที่มีประสิทธิภาพมาก ๆ เพราะว่าสอดคล้องกับทั้งสองข้อ

ไอเดียของวิธี CPMD นั้นเป็นการใช้ Adiabatic Time-Scale Separation (การแบ่งตาม Time-Scale) เพื่อแบ่ง Motion ของนิวเคลียสที่เคลื่อนที่ช้า และอิเล็กตรอนที่เคลื่อนที่เร็วโดยการแปลง Quantum Adiabatic Time-Scale ให้อยู่ในรูปของ Classical Adiabatic Energy-Scale แทน โดยในกรณีของ Classical Mechanics นั้นเราจะคำนวณแรง (Force) ได้จากอนุพันธ์ของ Lagrangian เทียบกับตำแหน่งของนิวเคลียส ซึ่งนี่เป็นการบอกใบ้กับเราอ้อม ๆ ว่าอนุพันธ์เชิงฟังก์ชัน (Functional Derivative) เทียบกับออร์บิทัลนั้นก็คือแรงที่กระทำต่อออร์บิทัลนั่นเอง นอกจากนี้เราจะต้องมีการกำหนด Constraint สำหรับออร์บิทัลด้วย เช่น ออร์บิทัลนั้นก็ควรที่จะมีความเป็น Orthonormality

ในเปเปอร์ต้นฉบับของ Car และ Parrinello นั้นได้มีการเสนอ Lagrangian ไว้ดังนี้

$$\mathcal{L}_{\mathrm{CP}} = \underbrace{\sum_I \frac{1}{2} M_I \dot{\mathbf{R}}_I^2 + \sum_i \frac{1}{2}\mu_i \left\langle \dot{\psi}_i \mid \dot{\psi}_i \right\rangle}_{\text{พลังงานจลน์}} - \underbrace{\langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle}_{\text{พลังงานศักย์}} + \underbrace{\text{Constraints}}_{\text{Orthonormality}} \tag{3.6.1}$$

และเราก็สามารถเขียน Newtonian Equation of Motion ได้จากชุดสมการ Euler-Lagrange ดังนี้

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{R}}_I} = \frac{\partial \mathcal{L}}{\partial \mathbf{R}_I} \tag{3.6.2}$$

$$\frac{d}{dt}\frac{\delta \mathcal{L}}{\delta \dot{\psi}_i^\star} = \frac{\delta \mathcal{L}}{\delta \psi_i^\star} \tag{3.6.3}$$

โดยที่ $\psi_i^\star = \langle \psi_i |$

สมการ Equations of Motion ของ Car-Parrinello นั้นมีหน้าตาดังต่อไปนี้

$$M_I \ddot{\mathbf{R}}_I(t) = -\frac{\partial}{\partial \mathbf{R}_I} \langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle + \frac{\partial}{\partial \mathbf{R}_I} \{ \text{ constraints } \} \tag{3.6.4}$$

$$\mu_i \ddot{\psi}_i(t) = < -\frac{\delta}{\delta \psi_i^\star} \langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle + \frac{\delta}{\delta \psi_i^\star} \{ \text{ constraints } \} \tag{3.6.5}$$

โดยที่ $\mu_i (= \mu)$ คือ “Fictitious Masses” หรือมวลแบบปลอม ๆ ที่เป็นตัวกำหนด Degrees of Freedom ของออร์บิทัล แล้วก็มี Mass Parameter $\mu$ ที่มีหน่วยเป็นพลังงานคูณกับเวลายกกำลังสอง ส่วนเทอมที่เขียนว่าเป็น Constraint ในสมการด้านบนนั้น จริง ๆ แล้วก็คือ Constraint Force ใน Equations of Motion ซึ่งมีหน้าตาประมาณนี้

$$\text{constraints} \;=\; \text{constraints}\; (\{\psi_i\}, \{\mathbf{R}_I\}) \tag{3.6.6}$$

ซึ่งอาจจะเป็นฟังก์ชันของออร์บิทัล $\{\psi_i\}$ และตำแหน่งของนิวเคลียร์ $\{\mathbf{R}_I\}$ ก็ได้

## 3.7 แล้ว Hellmann-Feynman Force ล่ะ?

ส่วนผสมที่สำคัญมาก ๆ ของวิธีทาง Molecular Dynamics ทุกวิธีนั้นก็คือแรง (Force) ซึ่งวิธีการคำนวณแรงที่มีประสิทธิภาพนั้นคือสิ่งที่เราต้องการ โดยเราที่เราสนใจก็คือแรงที่กระทำต่อนิวเคลียสของ

อะตอมแต่ละอะตอมในโมเลกุล โดยเราสามารถคำนวณแรงได้โดยการคำนวณอนุพันธ์ดังต่อไปนี้ด้วยวิธี Numerical Method (การประมาณเชิงตัวเอง)

$$\mathbf{F}_I = -\nabla_I \langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle \tag{3.7.1}$$

โดยเขียนในเทอมของการใช้การประมาณค่าด้วยผลต่างแบบมีขอบเขต (Finite Difference Approximation) ของพลังงานรวม (Total Electronic Energy) ซึ่งการคำนวณแรงตามสมการที่ (3.7.1) นั้นใช้เวลานานและให้ผลที่ไม่ถูกต้อง คำถามที่ตามมาคือ “จะเกิดอะไรขึ้นถ้าหากว่าเราสามารถคำนวณ Gradient ของพลังงานรวม (แรง) ได้ด้วยวิธีเชิงวิเคราะห์ (Analytically)” โดยนอกเหนือจากอนุพันธ์ของ Hamiltonian แล้ว

$$\nabla_I \langle \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle = \langle \Psi_0 | \nabla_I H_\mathrm{e} | \Psi_0 \rangle \tag{3.7.2}$$
$$+ \langle \nabla_I \Psi_0 | H_\mathrm{e} | \Psi_0 \rangle + \langle \Psi_0 | H_\mathrm{e} | \nabla_I \Psi_0 \rangle \tag{3.7.3}$$

ก็ยังมี Contribution ที่เกิดจาก Variation ของฟังก์ชันคลื่น (Wavefunction) $\sim \nabla_I \Psi_0$ ด้วย ซึ่งเราค่อนข้างโชคดีเพราะว่า Contribution ที่ว่านั้นจะหายไป (Vanish) ถ้าหากว่าฟังก์ชันคลื่นนั้นเป็น Eigenfunction ของ Hamiltonian

$$\mathbf{F}_I^\mathrm{HFT} = -\langle \Psi_0 | \nabla_I H_\mathrm{e} | \Psi_0 \rangle \tag{3.7.4}$$

ซึ่งเราเรียกแรงตามสมการที่ (3.7.4) นี้ว่าแรงของ Hellmann-Feynman Theorem (HFT) ซึ่งเป็นแรงที่เราสามารถคำนวณได้จาก Variational Favefunctions เช่น Hartree-Fock Wavefunction โดยมีข้อแม้ว่าฟังก์ชันคลื่นอันนั้นจะต้องเกิดขึ้นจากการใช้ Complete Basis Sets (CBS) เท่านั้น ถ้าหากว่าไม่ใช่ CBS เราจะต้องทำการเพิ่มเทอมพิเศษเข้าไป

เราลองมาดูตัวอย่างที่เป็นการใช้ Slater Determinant $\Psi_0 = \det\{\psi_i\}$ ของออร์บิทัล $\psi_i$ ซึ่งสามารถเขียนได้ตามสมการดังต่อไปนี้

$$\psi_i = \sum_\nu c_{i\nu} f_\nu (\mathbf{r}; \{\mathbf{R}_I\}) \tag{3.7.5}$$

ซึ่งเขียนในรูปของผลรวมเชิงเส้นของฟังก์ชันเบสิส (Linear Combination of Basis Functions) $\{f_\nu\}$ ซึ่งจะถูกใช้ร่วมกับ Effective One-Particle Hamiltonian เช่น Hamiltonian ในวิธี Hartree-Fock หรือวิธี Kohn-Sham โดยที่ Basis Functions นั้นอาจจะขึ้นอยู่กับตำแหน่งของนิวเคลียสเพียงอย่างเดียว ในขณะที่ Expansion Coefficients จะเป็นตัวที่กำหนดความแตกต่างที่บอกว่า Basis Functions แต่ละฟังก์ชันนั้นต่างกันอย่างไร นั่นหมายความว่าเราจะมีแรงสองแบบที่สามารถรวมกันได้ ดังนี้

$$\nabla_I \psi_i = \sum_\nu (\nabla_I c_{i\nu}) f_\nu (\mathbf{r}; \{\mathbf{R}_I\}) + \sum_\nu c_{i\nu} (\nabla_I f_\nu (\mathbf{r}; \{\mathbf{R}_I\})) \tag{3.7.6}$$

ซึ่งจะเป็นแรงที่นอกเหนือจากแรง Hellmann-Feynman

ถ้าหากว่าเราใช้การกระจายแบบเชิงเส้น (Linear Expansion) ตามสมการที่ (3.7.5) เราจะสามารถเขียนแรงที่มี Contribution ที่มาจาก Nuclear Gradient ของฟังก์ชันคลื่นได้ โดยเราจะแบ่งออกได้เป็น 2 เทอม

**แรงที่ได้จาก Nuclear Gradient ของฟังก์ชันคลื่น เทอมที่ 1**

เทอมแรกก็คือ Incomplete-Basis-Set Correction (IBS) ซึ่งเทอมนี้จะมีประโยชน์ในทฤษฎี Solid State และแรงชนิดนี้ก็ยังสอดคล้องกับ Wavefunction Force อีกด้วย (มีชื่อเรียกอีกชื่อว่า Pulay Force) ซึ่งสมการของแรง IBS นี้จะประกอบไปด้วย Nuclear Gradient ของ Basis Functions และ Effective One-Particle Hamiltonian ดังนี้

$$\mathbf{F}_I^{\mathrm{IBS}} = -\sum_{i\nu\mu} \left( \langle \nabla_I f_\nu \left| H_{\mathrm{e}}^{\mathrm{NSC}} - \epsilon_i \right| f_\mu \rangle + \langle f_\nu \left| H_{\mathrm{e}}^{\mathrm{NSC}} - \epsilon_i \right| \nabla f_\mu \rangle \right) \tag{3.7.7}$$

**แรงที่ได้จาก Nuclear Gradient ของฟังก์ชันคลื่น เทอมที่ 2**

สำหรับแรงที่เกิดมาจาก Nuclear Gradient ของฟังก์ชันคลื่นอีกแรงมีชื่อว่า Non-Self-Consistency Correction (NSC) ซึ่งมีสมการดังนี้

$$\mathbf{F}_I^{\mathrm{NSC}} = -\int d\mathbf{r} \left( \nabla_I n \right) \left( V^{\mathrm{SCF}} - V^{\mathrm{NSC}} \right) \tag{3.7.8}$$

อธิบายง่าย ๆ ก็คือแรง NSC นี้จะถูกกำหนดด้วยผลต่างระหว่าง Self-Consistent (“Exact”) Potential หรือ สนาม (Field) $V^{\mathrm{SCF}}$ และ Non-Self-Consistent Counterpart $V^{\mathrm{NSC}}$ ซึ่งก็จะเกี่ยวเนื่องกับ $H_{\mathrm{e}}^{\mathrm{NSC}}; n(\mathbf{r})$ ซึ่งก็คือความหนาแน่นประจุ (Charge Density) นั่นเอง

โดยสรุปแล้ว ผลรวมของแรงทั้งหมดที่จำเป็นสำหรับการจำลอง [ab initio] Molecular Dynamics นั้นมีดังนี้

$$\mathbf{F}_I = \mathbf{F}_I^{\mathrm{HFT}} + \mathbf{F}_I^{\mathrm{IBS}} + \mathbf{F}_I^{\mathrm{NSC}} \tag{3.7.9}$$

ถ้าหากว่าเราสนใจกรณีที่เราจะต้องคำนวณแรงตามสมการด้านบนนี้ด้วยวิธี Self-Consistency (ซึ่งเราไม่มีทางที่จะใช้ Numerical Method ได้อย่างแน่นอน) เราพบว่าแรง $\mathbf{F}_I^{\mathrm{NSC}}$ นั้นจะหายไปและ $H_{\mathrm{e}}^{\mathrm{SCF}}$ ก็จะถูกคำนวณเพื่อใช้ในการคำนวณแรง $\mathbf{F}_I^{\mathrm{IBS}}$ ต่อไป

## 3.8 การสุ่มตัวอย่างแบบมีประสิทธิภาพ

การสุ่มตัวอย่างแบบมีประสิทธิภาพ (Enhanced Sampling) เป็นเทคนิคที่ถูกพัฒนาขึ้นมาเพื่อแก้ปัญหาของการจำลอง MD แบบดั้งเดิมที่ใช้เวลานานมาก (Time Consuming) ในการศึกษาระบบเคมีสักระบบหนึ่ง นั่นจึงทำให้วิธีการจำลอง MD นั้นสามารถจำลองระบบได้ใน Time Scale ที่ระยะสั้นมาก ๆ เนื่องจากว่าเราจำเป็นที่จะต้องใช้ Time Step ที่สั้น (ได้เพียงแค่ไม่กี่ Femtoseconds เท่านั้นเอง) ดังนั้นจึงเป็นการยากที่จะใช้วิธี MD ในการจำลองปรากฏการณ์ทางเคมีของระบบที่เกิดขึ้นในช่วงระยะเวลา Time Scale ที่นานกว่า เช่น ในระดับ Millisecond

ในการเพิ่มประสิทธิภาพหรือความเร็วในการจำลองของ MD เพื่อให้เราสามารถมองเห็นปรากฏการณ์บางอย่างของระบบที่เราศึกษาได้ง่ายขึ้น เราสามารถทำได้โดยการใช้เทคนิคที่สามารถค้นหา Metastable State ของระบบของเราใน Energy Landscape ได้ง่ายขึ้น ซึ่งสิ่งที่เป็นตัวคั่นระหว่าง Metastable State เหล่านี้ก็คือพลังงานจลน์นั่นเอง ดังนั้นเราสามารถก้าวข้าม Barrier อันนี้ไปได้โดยการปรับเปลี่ยนระบบของเราโดยผ่านตัวแปรบางอย่าง ซึ่งตัวแปรหรือพารามิเตอร์ที่ว่านั้นก็คือตัวที่กำหนด Probability ในการ Identify ระบบของเรานั่นเอง สำหรับเทคนิค Enhanced Sampling ที่เราต้องกำหนด Parameters ที่สอดคล้องกับระบบของเราขึ้นมานั้นจะเรียกว่า Parameters-based ซึ่งมีอยู่ 2 เทคนิคหลัก ๆ ที่ได้รับความนิยม คือ Umbrella Sampling (US) และ Metadynamics โดย US นั้นเทคนิคที่จะมีการใส่ Bias แบบคงที่เข้าไปให้กับระบบ ส่วน Metadynamics นั้นจะเป็นการใส่ Bias ที่ไม่คงที่ (ขึ้นกับเวลาหรือ Time-Dependent) ให้กับระบบแทน

สำหรับในหัวข้อนี้ผมจะขออธิบายเพียงแค่เทคนิคเมตาไดนามิกส์ (Metadynamics)

### 3.8.1 เทคนิคเมตาไดนามิกส์

เมตาไดนามิกส์ (Metadynamics) เป็นหนึ่งในวิธีการ Enhanced Sampling ซึ่งอาศัยแนวคิดของการเติมพลังงานเข้าไปให้กับระบบเรื่อย ๆ (History-Dependent Potential) ซึ่งพลังงานที่ใส่เข้าไปนั้นจะอยู่ในรูปของฟังก์ชัน Gaussian และเราจะใส่พลังงาน Bias Potential เข้าไปเรื่อย ๆ ในระหว่างการจำลองด้วยวิธี MD โดยสมการของ Metadynamics Potential (Bias) นั้นจะเขียนโดยให้อยู่ในรูปของผลรวมของ Gaussian Function ดังนี้

$$V_{\text{bias}}\left(\vec{\boldsymbol{\xi}}, t\right) = \sum_{k\tau<t} W(k\tau) \exp\left(-\sum_{i=1}^{d} \frac{(\boldsymbol{\xi}_i - \boldsymbol{\xi}_i^{(0)}(k\tau))^2}{2\sigma_i^2}\right) \tag{3.8.1}$$

โดยที่ $\boldsymbol{\xi} = \{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots\}$ คือ Collective Variables (CV) ซึ่งเป็นพารามิเตอร์ที่ผู้ใช้งาน Metadynamics นั้นจะต้องกำหนดเอง โดย $\boldsymbol{\xi}$ นั้นจะเป็นตัวกำหนด Sampling Direction และ Reaction Coordinates ที่ Bias Potential นั้นจะถูกใส่เข้าไป

เมื่อเราทำการจำลอง MD พร้อมกับการ Sampling ด้วยวิธี Metadynamics เราจะได้สมการ Equation of Motion ที่มีการรวมแรงจาก MD และแรงจาก Metadynamics เข้าด้วยกัน ดังนี้

$$M_i \frac{d^2\mathbf{r}_i}{dt^2} = \mathbf{F}_{\text{MD},i}(\mathbf{r}_i) - \mathbf{F}_{\text{bias},i}(\mathbf{r}_i) \tag{3.8.2}$$

$$= -\frac{\partial V(\mathbf{R})}{\partial \mathbf{r}_i} - \frac{\partial V_{\text{bias}}(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{\xi}}{\partial \mathbf{r}_i} \cdots \tag{3.8.3}$$

โดยที่ $M_i$ คือมวลของอะตอม $i$, $\mathbf{r}_i$ คือตำแหน่งของอะตอม, และ $t$ คือเวลา

การกำหนด CV สำหรับการรันการคำนวณ Metadynamics นั้นสำคัญมาก ๆ โดย CV ที่เรากำหนดนั้นจะต้องสอดคล้องกับปรากฏการณ์ทางเคมีที่เราสนใจ และ CV นั้นจะต้องเป็นฟังก์ชันที่สามารถแยกสถานะเริ่มต้นและสถานะสิ้นสุดของโมเลกุลได้อย่างชัดเจน หรือที่เรียกว่า Metastable State ตัวอย่างเช่น ถ้าหากว่าเราอยากจะจำลอง MD ของปฏิกิริยาการสร้างพันธะระหว่างโมเลกุล 2 โมเลกุล สถานะตั้งต้น (Reactant State) กับสถานะผลิตภัณฑ์ (Product State) ก็คือ Metastable State ที่เราสนใจ ซึ่งพารามิเตอร์หรือฟังก์ชันที่เหมาะสมที่จะถูกนำมาใช้เป็น CV ในกรณีนี้ก็คือความยาวพันธะ (Bond Distance) ระหว่างอะตอมของทั้ง 2 โมเลกุลนั่นเอง

ตัวเลือกของ CV นั้นมีหลากหลายฟังก์ชัน โดยพารามิเตอร์ที่เราจะสามารถใช้ได้นั้นจะต้องเป็นฟังก์ชันที่ขึ้นกับตำแหน่งของอะตอมด้วย

**ความยาวพันธะ (Bond Distance)**:

$$\boldsymbol{\xi}_{dist} = |\mathbf{r}_{G_1} - r_{G_2}| \tag{3.8.4}$$

**มุมพันธะ (Bond Angle)**:

$$\begin{aligned}
\mathbf{r}_a &= \mathbf{r}_{G_1} - \mathbf{r}_{G_2} \\
\mathbf{r}_b &= \mathbf{r}_{G_1} - \mathbf{r}_{G_3} \\
\mathbf{r}_c &= \mathbf{r}_{G_2} - \mathbf{r}_{G_3} \\
\boldsymbol{\xi}_{angle} &= \cos^{-1}\left(\frac{\mathbf{r}_a^2 + \mathbf{r}_c^2 - \mathbf{r}_b^2}{2|\mathbf{r}_a||\mathbf{r}_c|}\right)
\end{aligned}$$

**มุมบิด (Torsion Angle)**:

$$\begin{aligned}
\mathbf{r}_a &= \mathbf{r}_{G_4} - \mathbf{r}_{G_3} \\
\mathbf{r}_b &= \mathbf{r}_{G_3} - \mathbf{r}_{G_3} \\
\mathbf{r}_c &= \mathbf{r}_{G_2} - \mathbf{r}_{G_2} \\
\mathbf{r}_d &= \mathbf{r}_{G_1} - \mathbf{r}_{G_2} \\
\boldsymbol{\xi}_{torsion} &= \cos^{-1}\left(\frac{(\mathbf{r}_a \times \mathbf{r}_b) \cdot (\mathbf{r}_c \times \mathbf{r}_d)}{|\mathbf{r}_a \times \mathbf{r}_b)||(\mathbf{r}_c \times \mathbf{r}_d|}\right)
\end{aligned}$$

Adjacency matrix (Smooth Function):

$$\xi_A = \frac{1 - \left(\frac{|\mathbf{r}_i - \mathbf{r}_j|}{r_0}\right)^n}{1 - \left(\frac{|\mathbf{r}_i - \mathbf{r}_j|}{r_0}\right)^m} \tag{3.8.5}$$

สำหรับโปรแกรมคำนวณ Metadynamics นั้นผู้อ่านสามารถศึกษาได้จากโค้ดด้านล่าง (ภาษา Python) โดยเป็นโค้ดของ Metadynamics อย่างง่าย (กรณี 1 มิติ)

```
import numpy as np


def energy(h, w, xyz, x, nbump):
    """
    The sum of Gaussian with height h and
    width w at positions xyz sampled at x.
    Use distance matrices to maintain
    rotational invariance.

    Args :
    h : height
    w: width
    xyz : bumps x N x 3 tensor
    x : (n X 3) tensor representing the point
        at which the energy is sampled
    nbump : the number of bumps
    """

    xshp = np.shape(x)
    nx = xshp[0]
    Nzxyz = np.slice(xyz, [0, 0, 0], [nbump, nx, 3])
    Ds = distances(Nzxyz)
    Dx = distances(x)
    w2 = np.square(w)
    rij = Ds - np.tile(np.reshape(Dx, [1, nx, nx]), [nbump, 1,
    1])
    ToExp = np.einsum("ijk,-ijk>i", rij, rij)
    ToSum = -1.0 * h * np.exp(-0.5 * ToExp / w2)
    return -1.0 * np.reduce_sum(ToSum, axis=0)


def force(energy, x):
```

```
33      return np.gradient(energy, x)
34
35
36  def distances(r):
37      """
38      Calculat edistance matrices
39      """
40      rm = np.einsum("ijk,ijk->ij", r, r)
41      rshp = np.shape(rm)
42      rmt = np.tile(rm, [1, rshp[1]])
43      rmt = np.reshape(rmy, [rshp[0], rshp[1], rshp[1]])
44      rmtt = np.transpose(rmp, perm=[0, 2, 1])
45      D = rmt - 2 * np.einsum("ihk,ilk->ijl", r, r) + rmtt +
       np.cast(1e-28, np.float64)
46      return np.sqrt(D)
```

## 3.9 แบบฝึกหัด

1. เขียนโปรแกรมสำหรับจำลองระบบอะตอมเดี่ยวด้วยวิธี Born-Oppenheimer Molecular Dynamics อย่างง่าย

2. เขียนโปรแกรม Metadynamics สำหรับจำลองระบบโมเลกุลแบบ 1 มิติด้วยภาษา C++

3. เขียนโปรแกรม Metadynamics สำหรับจำลองระบบโมเลกุลแบบ 2 มิติ (ใช้ภาษาอะไรก็ได้และสามารถประยุกต์จากโปรแกรมตัวอย่างได้)

# บทที่ 4

# การพัฒนาซอฟต์แวร์สำหรับเคมีเชิงคำนวณ

## 4.1 การเขียนโปรแกรมทางเคมีเชิงคำนวณ

ถ้าหากผู้อ่านอยากจะศึกษาการเขียนโปรแกรมทางเคมีเชิงคำนวณจะเริ่มยังไงดี เช่น ต้องการเขียนโปรแกรม Density Functional Theory (DFT) หรือ Implement วิธีโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) ผมขอให้ความเห็นอย่างนี้ครับว่าการจะที่เขียนโปรแกรมทางเคมีคำนวณขึ้นมาสักโปรแกรมหนึ่งนั้นใช้เวลามากพอสมควรเพราะว่ามีรายละเอียดที่ซับซ้อนมาก (เวลาที่ใช้ในการเขียนนั้นขึ้นอยู่กับว่าเขียนคนเดียวหรือช่วยกันเขียนหลายคน) ดังนั้นผมแนะนำว่าสำหรับผู้ที่เพิ่งเริ่มต้นการเขียนโปรแกรมทางวิทยาศาสตร์ควรศึกษาจากโปรแกรมมาตรฐานที่ได้รับความนิยมอยู่แล้ว ผมไม่ได้บอกว่าห้ามเขียนโปรแกรมใหม่เองแบบเริ่มจากศูนย์หรือ From Scratch แต่ถ้าหากว่าเราเริ่มต้นเรียนรู้จากโปรแกรมที่ได้รับความนิยมและใช้งานกันอย่างแพร่หลายอยู่แล้วก็มีข้อดีดังนี้

- ประหยัดเวลา ไม่ต้องมานั่งศึกษาหรือเขียนโค้ดใหม่เองทั้งหมด
- ได้เรียนรู้วิธีการเขียนโค้ดที่มีประสิทธิภาพจากนักพัฒนาคนอื่น ๆ
- เป็นการต่อยอดและพัฒนาโปรแกรมนั้น ๆ ให้ดีขึ้นไปอีกเพราะเราไม่จำเป็นต้องมา “Reinvent the Wheel”[1]
- เป็นการสร้างเครือข่ายนักวิจัยและความร่วมมือทางวิชาการในระดับนานาชาติ

อย่างไรก็ตามถ้าหากใครอยากจะเริ่มเขียนโปรแกรมเองนั้น (ไม่จำเป็นต้องเป็น DFT อย่างเดียว แต่รวมถึงวิธีการจำลองทางคอมพิวเตอร์อื่น ๆ ด้วย เช่น Molecular Dynamics หรือ Monte Carlo) ก็มีข้อดีหลายข้อเหมือนกัน ดังนี้

[1]คือการพยายามทำสิ่งที่คนอื่นได้ทำเอาไว้ดีอยู่แล้วใหม่ตั้งแต่ต้น ซึ่งเป็นอะไรที่เสียเวลาและเสียทรัพยากรณ์ โดยไม่คุ้มค่าเอาเสียเลย

- ได้ทำความเข้าใจการเขียนโปรแกรมอ้างอิงตามสมการทาง Electronic Structure
- ฝึกทักษะการเขียนโปรแกรมสำหรับการคำนวณทางวิทยาศาสตร์และได้เรียนรู้เทคนิคการประมาณค่าเชิงตัวเลข
- ได้ออกแบบโปรแกรมเองและ Implement วิธีใหม่ ๆ ที่โปรแกรมอื่นไม่มี
- ต่อยอดเป็นโปรแกรมในรูปแบบเชิงพานิชย์ได้เพราะว่ามีโปรแกรมทางเคมีคำนวณหลาย ๆ โปรแกรมที่ขาย License

ประเด็นหรือคำถามสำคัญคือ “ถ้าหากอยากจะเริ่มศึกษาโค้ดของวิธีการคำนวณทางเคมีควอนตัม เช่น โปรแกรม Density Functional Theory (DFT) ดี ๆ สักตัวนึงจะเริ่มจากไหนดี?” ความเห็นของผมคือแนะนำให้ศึกษาโปรแกรม PySCF โดยมีเหตุผลดังต่อไปนี้

- โปรแกรมมีประสิทธิภาพสูง ทำงานได้เร็วและให้ผลการคำนวณที่ถูกต้องและแม่นยำและยังคำนวณได้หลากหลายวิธี
- มีผู้ใช้งานเยอะเนื่องจากว่าโปรแกรม PySCF นั้นสามารถติดตั้งและใช้งานได้ง่าย เตรียมไฟล์ Input ได้ไม่ยุ่งยาก
- PySCF เขียนด้วย Python เกือบทั้งหมด (87% เขียนด้วย Python, 12% เป็นภาษา C ก็คือพวกไลบรารี่ต่าง ๆ ที่เอามาคำนวณในส่วนที่ Python อาจจะคำนวณได้ช้า) ดังนั้นจึงง่ายต่อการทำความเข้าใจ
- มีทีมพัฒนาที่ใหญ่และแข็งแกร่ง ได้รับการสนับสนุนฟีเจอร์และแก้ไข Bug อย่างต่อเนื่อง

จากข้อ 1 ถ้าหากเราต้องการ Implement วิธีหรือเทคนิคใหม่ ๆ เข้าไปใน PySCF ก็ทำได้ง่ายเพราะว่าเขียนด้วยภาษา Python นอกจากนี้โปรแกรมยังสามารถทำงานด้วย GPU ได้ด้วย (มี Plugin พิเศษชื่อว่า gpu2pyscf) ตัวโค้ดถูกเขียนและได้รับการปรับปรุงมาเป็นอย่างดี (Well-Written) มีการวางโครงสร้างของโปรแกรมที่เรียบร้อย แบ่ง Methods ต่าง ๆ ออกเป็น Module ที่ชัดเจนและมีการจัดวาง Function ที่เหมาะสม สำหรับผู้อ่านที่สนใจโปรแกรม PySCF ก็ไปดูได้ที่ https://github.com/pyscf/pyscf

เมื่อเราเลือกโปรแกรมได้แล้ว ขั้นตอนต่อไปก็คือพยายามทำความเข้าใจทฤษฎีของหัวข้องานวิจัยที่เราต้องการศึกษา พยายามหาว่าเราสามารถพัฒนาวิธีนั้น ๆ ได้อย่างไรเพื่อที่จะปรับปรุงให้มีความถูกต้องในการคำนวณมากขึ้นหรือหากรณีที่ทฤษฎีนั้นยังไม่ครอบคลุม ขั้นตอนต่อไปคือหาวิธีการแก้ไขปัญหาหรือ Solution สำหรับการปรับปรุงทฤษฎีนั้นแล้วเขียนออกมาเป็นสมการทางคณิตศาสตร์ที่เราจะนำไป Implement ได้ ขั้นตอนต่อไปก็คือการวางแผนการเขียนโปรแกรมซึ่งสามารถทำได้ด้วยการเขียนโค้ดเทียมหรือ Pseudo Code ก่อนที่เราจะ Implement จริง ๆ โดยเราจะต้องคิดเกี่ยวกับการวางโครงสร้าง (Structure) ของโปรแกรม เช่น แบ่งโปรแกรมออกเป็นโปรแกรมย่อย ๆ หลายส่วน เช่น แบ่งเป็น Modules, Functions, Classes, หรือ Types โดยเราควรจะต้องคำนึงถึงการพัฒนาโปรแกรมต่อไปในอนาคตด้วยว่าโปรแกรมของเรานั้นสามารถที่จะรองรับฟีเจอร์ใหม่ ๆ ที่นักพัฒนาคนอื่น ๆ จะเข้ามาช่วยพัฒนาเพิ่มเติมได้

หลังจากที่เรา Implement เข้าไปในโปรแกรมเสร็จเรียบร้อยแล้วเราควรจะต้องมีการตรวจสอบการทำงานของโปรแกรมหรือฟังก์ชันต่าง ๆ อย่างสม่ำเสมอเพื่อตรวจสอบค่าที่ได้จากคำนวณว่ามีความถูกต้องและ

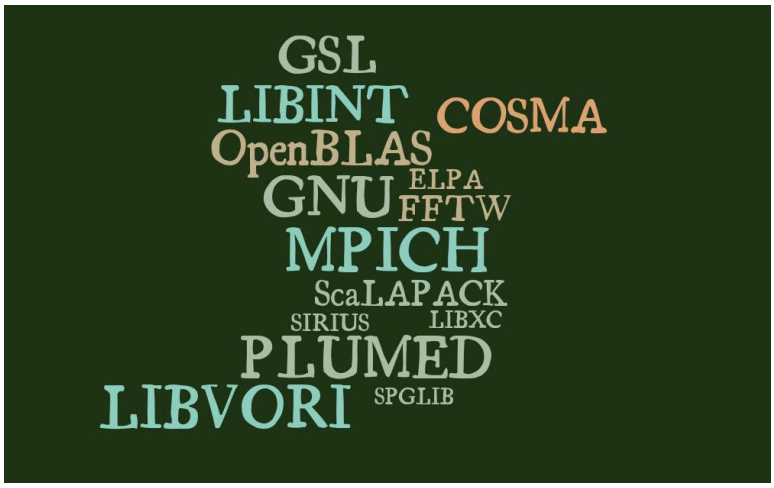
มีความสมเหตุสมผลมากน้อยแค่ไหน เมื่อได้ค่าการคำนวณที่ถูกต้องแล้วขั้นตอนสุดท้ายก็คือการปรับปรุงหรือทำความสะอาดโค้ดให้มีประสิทธิภาพและอ่านได้ง่ายขึ้น ในขั้นตอนนี้เราสามารถเรียนรู้ได้จากการศึกษาโค้ดที่นักพัฒนาคนอื่นเขียนไว้ก็ได้ว่าเขาเขียนอย่างไร ใช้วิธีหรือเทคนิคอะไรที่ทำให้โค้ดรันได้เร็วและมีประสิทธิภาพ นอกจากนี้ยังมีสิ่งอื่น ๆ ที่เราควรจะต้องทำด้วย เช่น เขียน Comment หรือทำเอกสารประกอบการใช้งาน (Documentation) เพื่อที่ว่าตัวเราเองหรือนักพัฒนาคนอื่น ๆ ที่มาอ่านหรือแก้ไขโค้ดของเรานั้นสามารถทำความเข้าใจโค้ดได้ง่ายและไม่ต้องมานั่งศึกษาเองจากศูนย์

## 4.2 การวางโครงสร้างโปรแกรม

การที่เราจะเขียนโปรแกรมคำนวณทางวิทยาศาสตร์นั้นควรที่จะต้องมีการวางแผนให้ดีเพราะว่าเมื่อเราเขียนโค้ดไปเรื่อย ๆ ตัวโปรแกรมของเราก็จะมีขนาดที่ใหญ่ขึ้นและมีความซับซ้อนมากขึ้นด้วย ดังนั้นการวางโครงสร้างของโปรแกรมเพื่อให้รองรับฟังก์ชันหรือฟีเจอร์ใหม่ ๆ ที่อาจจะมีการเขียนโค้ดเพิ่มเข้ามานั้นช่วยให้โปรแกรมนั้นมีความเป็นระเบียบและง่ายต่อการ Maintenance และไม่สร้างความปวดหัวให้กับนักพัฒนาคนอื่น ๆ ที่อาจจะเข้ามาพัฒนาโปรแกรมของเราต่อ (อาจจะสร้างความปวดหัวแต่ก็ไม่เยอะเท่ากับโปรแกรมที่มีการวางโครงสร้างที่ไม่ดี)

ผมขอยกตัวอย่างโปรแกรม CP2K ซึ่งเป็นโปรแกรมที่ผมใช้ในงานทำวิจัย ตัว Source Code ของ CP2K นั้นจะมีโฟลเดอร์ต่าง ๆ เช่น `src/`, `docs/`, `tests/`, หรือ `tools/` ซึ่งโฟลเดอร์เหล่านี้เก็บไฟล์ที่มีหน้าที่แตกต่างกันออกไป แต่โฟลเดอร์ที่น่าจะสำคัญที่สุดก็คือ `src/` ซึ่งเก็บไฟล์โค้ดการทำงานหลักของตัวโปรแกรมเอาไว้ ส่วนโฟลเดอร์อื่น ๆ เช่น `test/` เป็นโฟลเดอร์ที่เก็บไฟล์อินพุต (Input) และเอาต์พุต (Output) ที่ไว้ใช้สำหรับการทดสอบโปรแกรมและเปรียบเทียบกับค่าผลลัพธ์จากการคำนวณอ้างอิง โดยเฉพาะเวลาที่นักพัฒนา (Developers) นั้นแก้ไขตัวโปรแกรมและทำการคอมไพล์โปรแกรมใหม่ก็จะได้มีค่าเปรียบเทียบที่ยืนยันได้ว่าโปรแกรมยังให้ผลการคำนวณที่ถูกต้อง ตัวโฟลเดอร์ `src/` นั้นบางโปรแกรมก็มีขนาดหลายร้อยเมกะไบต์หรือบางโปรแกรมก็มีหลายกิกะไบต์ ขึ้นอยู่กับว่าตัวโปรแกรมนั้นซับซ้อนมากแค่ไหน ซึ่งความซับซ้อนของโปรแกรมนั้นอาจจะวัดได้ง่าย ๆ จากจำนวนของ Features หรือความสามารถของโปรแกรมที่คำนวณได้ (นับจำนวนวิธีที่คำนวณได้ก็ได้) นอกจากนี้เรายังดูได้จากความซับซ้อนของการ Implementation เช่น ถ้าโปรแกรมสามารถทำงานแบบขนาด (Parallel) บน Distributed Cluster ได้นั่นหมายความว่าโค้ดของโปรแกรมนั้นจะต้องมีการถูกปรับ (Optimized) ให้รองรับวิธี OpenMP หรือ Message-Passing Interface (MPI) ซึ่งก็จะซับซ้อนกว่าโค้ดของโปรแกรมทั่วไป นอกจากนี้แล้วยังมีอีกหนึ่งเหตุผลนั่นก็คือโปรแกรมนั้นใช้ Packages หรือ Library อื่นมากน้อยแค่ไหน เพราะว่าในปัจจุบันนั้นการพัฒนาโปรแกรมทางวิทยาศาสตร์โดยเฉพาะเคมีควอนตัมนั้นเราก็มักจะไม่ได้เขียนส่วนประกอบต่าง ๆ ของโปรแกรมเองใหม่ทั้งหมด (หรือที่เรียกว่าเขียนแบบเริ่มจากศูนย์หรือ From Scratch เลย) นั่นก็เพราะว่าแต่ละส่วนหรือองค์ประกอบของการคำนวณนั้นมีความซับซ้อนมาก ดังนั้นจึงมีนักวิจัยที่สร้าง Library สำหรับการคำนวณบางอย่างไว้ให้เราแล้วซึ่งเราก็สามารถหยิบมาใช้ได้เลย การคำนวณบางอย่างที่มีความซับซ้อนนั้น เช่น การคำนวณ Matrix Multiplication หรือการคำนวณ One-Electron Integral และ Two-Electron Integral รวมไปถึง Library เฉพาะทาง เช่น Library ที่มีชุดฟังก์ชันของ Functionals สำหรับการคำนวณ DFT ให้เรานำมาใช้งานได้เลย เรียกได้ว่าทำให้ชีวิตนักเคมีทฤษฎีที่ต้องพัฒนาโปรแกรมนั้นนั้นประหยัดเวลาชีวิตไปได้เยอะมาก

โปรแกรม CP2K ซึ่งเป็นอีกหนึ่งโปรแกรมที่ถูกพัฒนาขึ้นโดยใช้ประโยชน์จาก Library อื่น ๆ มีโครงสร้างตามภาพด้านล่าง




ภาพ 4.1 ไลบรารี่ที่โปรแกรม CP2K ใช้ในการช่วยคำนวณ

ผมจะอธิบาย Library ที่สำคัญ ๆ บางอันที่ CP2K ใช้ ดังนี้

**GNU** แน่นอนว่าเราต้องคอมไพล์ Source Code ดังนั้นเราจะต้องใช้ตัวคอมไพล์ (Compiler) ซึ่ง CP2K เลือกใช้ GNU เป็น Compiler

**OpenBLAS กับ ScaLAPACK** สำหรับ Linear Algebraic Calculation เช่น Matrix-vector, Matrix-matrix Multiplication

**MPICH** อยากจะรันโปรแกรมแบบขนานโดยใช้ MPI ก็ต้องหา Implementation ที่จะมารันโค้ดของเรา ซึ่ง MPICH ก็เป็นหนึ่งใน Implementation ของ MPI ที่ CP2K เลือกใช้

**FFTW** สำหรับทำฟูเรียร์ทรานฟอร์มในการคำนวณ DFT หรือแปลงจาก Real Space ไปเป็น Reciprocal Space สำหรับการคำนวณ Electrostatics โดยใช้ Plane Wave Electron Density

**LIBXC** เป็น Library ที่ให้เราสามารถนำ DFT Functional มาใช้ได้เลยโดยไม่ต้อง Implement เอง

สรุปก็คือจะเห็นได้ว่าการเขียนโค้ดของโปรแกรมเคมีควอนตัมนั้นมีความซับซ้อนมากดังนั้นเรามีสองทางเลือกคือ

1. ใช้ Library ที่มีอยู่แล้วสำหรับการทำงานเฉพาะจุด
2. เขียนโค้ดทั้งหมดเองเลย

แน่นอนว่าถ้าเราเลือกวิธีแรกก็จะประหยัดเวลาไปได้เยอะมากและเวลาที่เราคอมไพล์โปรแกรมก็ขอแค่ Link กับ Library ต่าง ๆ ก็รันโปรแกรมได้แล้วและทำให้ขนาดของตัวโปรแกรมของเรา (ขนาดของ Binary Files) นั้นมีขนาดไม่ใหญ่มากเกินไปด้วย (เช่นหลักสิบ-ร้อยเมกะไบต์)

อย่างไรก็ตามโปรแกรมสำหรับจำลองระบบโมเลกุลหลาย ๆ โปรแกรมก็ไม่ได้ใช้ Library เหล่านี้และเลือกใช้วิธีที่ 2 ก็คือการเขียนโค้ดสำหรับการคำนวณส่วนต่าง ๆ เองเลยเนื่องด้วยเหตุหลายข้อ เช่น การเขียนโค้ดทั้งหมดภายใน Framework โปรแกรมเดียวกันนั้นจะทำให้โค้ดมีประสิทธิภาพและทำงานร่วมกันได้ดี (Compatibility), ง่ายต่อการดูแลรักษาและปรับปรุงโค้ดเพราะว่า Developers นั้นรู้และเข้าใจการทำงานของโค้ดทั้งหมด, ถึงแม้ว่าโปรแกรมที่เขียนโค้ดทั้งหมดเองเมื่อถูกคอมไพล์แล้วจะได้ออกมาเป็น Binary File ที่มีขนาดนั้นใหญ่มาก ๆ เช่น หลายกิกะไบต์ แต่ว่าก็มีความคล่องตัวในการใช้งานเพราะว่าไม่ต้องติดตั้ง Library อื่น ๆ เพิ่มเติม

นอกจากนี้แล้วการที่ใช้ Library หลาย ๆ ตัวแบบนี้ก็มีจุดอ่อนบางข้อที่เราควรจะต้องรู้ไว้นั่นก็คือการเข้ากันได้ (Compatibility) ระหว่าง Library หรือเวอร์ชันซึ่งก็อาจจะทำให้เราปวดหัวได้ถ้าหากว่า Library บางตัวมีการอัพเดทเวอร์ชันใหม่แล้ว Conflict กับ Library ตัวอื่น

## 4.3 ทักษะและเครื่องมือสำหรับการเขียนโปรแกรมคำนวณทางวิทยาศาสตร์

**ภาษาคอมพิวเตอร์**

- ภาษาสคริปต์ (Scripting Language): Bash, Python
- ภาษาระดับล่างที่เป็น Object-Oriented Programming: C++, Fortran
- ภาษาเชิงสัญลักษณ์ (Symbolic Programming): Mathematica, SymPy

**โปรแกรมสำหรับการเขียนโค้ด**

- Vi/Vim, Nano
- VS Code, Atom, Eclipse, Sublime, Notepad++

**พื้นฐานการเขียนโปรแกรม**

- ชนิดของตัวแปร (Variable Types)
- ตัวดำเนินการหรือโอเปอร์เรเตอร์ (Operator)
- Control Statements เช่น For, Do, If-Else, Case
- ฟังก์ชัน (Function)

- Vairable Scope และ Reference Types
- คลาส (Class) และวัตถุ (Objects)

**ภาษาคอมพิวเตอร์ระดับล่าง**

ภาษา C

- Function, Pointer, Storage Class
- Enum, Struct, Union
- Preprocessor
- Operator, Memory Management, Array
- การจัดการไฟล์ (File Handling)

**ภาษาคอมพิวเตอร์ระดับสูง**

ภาษา Python

- Pip และ Conda: ตัวช่วยจัดการไลบรารี่ของ Python
- NumPy: จัดการและคำนวณ Array (เวกเตอร์, เมทริกซ์)
- Numba: JIT Compiler สำหรับ NumPy
- Jax: ทำ Autograd (Gradient Comptuation) สำหรับ NumPy Array
- SciPy: ไลบรารี่ที่รวมรวบฟังก์ชันทางคณิตศาสตร์และวิทยาศาสตร์
- Scikit-learn: ไลบรารี่สำหรับทำสถิติ, Optimization, และ Curve Fitting รวมถึง Machine Learning
- Matplotlib, Plotly: พลอตกราฟ
- Theano: คำนวณเชิงตัวเลข (Numerical Computation)
- SCOOP: โมดูลสำหรับการทำโปรแกรมแบบขนาน (Parallel Programming)
- NetworkX: ไลบรารี่สำหรับ Graph

ภาษา C++

- Type of variable: ‘signed‘, ‘unsigned‘, ‘long‘, ‘double‘ และอื่น ๆ
- Loops, Conditional Statement

- Standard libraries: ‘vector‘, ‘rand‘
- เข้าใจ header (‘.hpp‘) และ Source File (`.cpp` or `.cc`)
- Preprocessor (`#if`, `#ifdef`, `#ifndef`, `#define`, etc.)
- Function, Class, Struct, Template
- Declaration
- Namespace, Const, Attribute, Pointer, Pass by Reference, ‘static_assert‘
- Initialization
- Casting, Lambda Expression, Encapsulation, File Handling, Exception Handling

ภาษา Fortran

- เรียนรู้ภาษา Fortran ที่เป็น Modern Fortran ตั้งแต่เวอร์ชัน 2003 เป็นต้นไป
- โมดูล (Module), โปรแกรมย่อยหรือซับรูทีน (Subroutine), ฟังก์ชัน (Function)
- Array ทั้งแบบที่ปรับเปลี่ยนได้ (Allocatable) และแบบหลายมิติ (Multidimentional)
- Operator Overloading, Flow control
- Derived Type
- Callback
- การเขียนโปรแกรมเชื่อมโยงกับภาษาอื่น เช่น Python หรือ C++
- การใช้ GNU Library ในการคอมไพล์
- การจัดการหน่วยความจำ (Memory Allocation): Stack, Heap, Global Memory

ไลบรารี่สำหรับการคำนวณทางคณิตศาสตร์

- BLAS (OpenBLAS)
- LAPACK: สำหรับคำนวณ Linear Algebra
- ScaLAPACK: เป็น LAPACK สำหรับซุปเปอร์คอมพิวเตอร์
- Intel MKL (Intel oneAPI)
- FFTW: สำหรับคำนวณ Discrete Fourier Transform ในหนึ่งมิติหรือมากกว่าหนึ่งมิติก็ได้
- Eigen: สำหรับคำนวณ Linear Algebra

- Boost: ไลบรารี่ที่รวบรวมฟังก์ชันต่าง ๆ สำหรับช่วยเขียนโปรแกรมภาษา C++ เช่น regex, serialization

เครื่องมือช่วยการพัฒนาซอฟต์แวร์

- การทำ Code Optimization
- การทำ Benchmarking และ Scaling
- ความซับซ้อนเชิงการคำนวณ (Computational Complexity)
- Static และ Dynamic Libraries
- คอมไพเลอร์ Compiling (g++, gcc) และ Linking (ld)
- เครื่องมือสำหรับช่วยการคอมไพล์โค้ด: autoconf, configure, make, cmake, automake
- เครื่องมือสำหรับช่วยการ Debug: gdb สำหรับการ Debug ทั่วไปและ Valgrind สำหรับการวิเคราะห์ Memory Leak
- การทำ Source Code Control: Git (GitHub และ GitLab)
- การทำ Documentation: Sphinx (สำหรับ markdown และ reStructuredText), Doxygen

ไลบรารี่สำหรับเคมีควอนตัม

- libxc: ไลบรารี่ที่รวบรวม XC Function สำหรับ DFT ซึ่งเราสามารถนำมาใช้งานได้เลย
- libint: ไลบรารี่ที่ช่วยในการคำนวณ Gaussian Integrals
- libcint: ไลบรารี่ที่ช่วยในการคำนวณ GTO Integrals

## 4.4 เขียนโปรแกรมวิเคราะห์ Molecular Geometry (ภาษา C++)

โปรแกรมแรกที่ผู้อ่านจะได้ศึกษาและฝึกเขียนตามก็คือโปรแกรมสำหรับวิเคราะห์เรขาคณิตเชิงโครงสร้าง Structural Geometry ของโมเลกุล โดยเราจะใช้ภาษา C++ กันครับ ซึ่งภาษา C++ นั้นเป็นภาษาที่มีประสิทธิภาพสูงมากและถูกใช้อย่างแพร่หลายในการทำงานวิจัยเคมีควอนตัม (รวมถึงสาขาอื่น ๆ ด้วย) และโปรแกรมสำหรับการวิเคราะห์อันนี้ไม่มีความซับซ้อนมาก

**Step 1: อ่านไฟล์ Coordinates**

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>

using namespace std;

int main()
{
  ifstream input("geom.dat");

  int natom;
  input >> natom;

  int *zval = new int[natom];
  double *x = new double[natom];
  double *y = new double[natom];
  double *z = new double[natom];

  for(int i=0; i < natom; i++)
    input >> zval[i] >> x[i] >> y[i] >> z[i];

  input.close();

  cout << "Number of atoms: " << natom << endl;
  cout << "Input Cartesian coordinates:\n";
  for(int i=0; i < natom; i++)
    printf("%d %20.12f %20.12f %20.12f\n", (int) zval[i], x[i],
     y[i], z[i]);

  delete[] zval;
  delete[] x;  delete[] y;  delete[] z;

  return 0;
}
```

จากตัวอย่างโค้ดด้านบนนั้นเริ่มการทำงานด้วยการอ่านไฟล์ Coordinates ของอะตอมในโมเลกุลซึ่งมีหน่วยคือ bohr โดยผู้อ่านอาจจะลองใช้ตัวอย่างโมเลกุลเล็ก ๆ เช่น Acetaldehyde ซึ่งมี Coordinates ดังนี้

```
7
6  0.000000000000      0.000000000000      0.000000000000
6  0.000000000000      0.000000000000      2.845112131228
```

```
 8  1.899115961744      0.000000000000      4.139062527233
 1 -1.894048308506      0.000000000000      3.747688672216
 1  1.942500819960      0.000000000000     -0.701145981971
 1 -1.007295466862     -1.669971842687     -0.705916966833
 1 -1.007295466862      1.669971842687     -0.705916966833
```

โดยที่เลข 7 ในบรรทัดแรกนั้นคือจำนวนอะตอมและบรรทัดที่เหลือนั้นคือพิกัดตำแหน่ง $x, y, z$ ของอะตอมแต่ละตัว เมื่อโปรแกรมเปิดไฟล์แล้วสิ่งที่จะทำต่อไปก็คือการอ่านข้อมูลแต่ละบรรทัดและเก็บข้อมูลไว้ จริง ๆ แล้วเราสามารถทำให้โค้ดตัวอย่างอันนี้มีความเป็นระเบียบมากขึ้นโดยการใช้ Class เช่น เราสามารถสร้างไฟล์ header ที่ชื่อ `molecule.h` แล้วทำการเรียกใช้งาน Class ในไฟล์โปรแกรม `molecule.cpp` ได้ดังนี้

```cpp
#include "molecule.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>

using namespace std;

int main()
{
  Molecule mol("geom.dat", 0);

  cout << "Number of atoms: " << mol.natom << endl;
  cout << "Input Cartesian coordinates:\n";
  mol.print_geom();

  return 0;
}
```

ทีนี้สิ่งที่ผมอยากให้ผู้อ่านทำก็คือลองฝึกเขียนโค้ดของไฟล์ `molecule.h` โดยดัดแปลงจากโค้ดด้านบนครับ

**Step 2: คำนวณความยาวพันธะ (Bond Lengths)**

คำนวณระยะห่างระหว่างอะตอม (Interatomic Distances) โดยใช้สมการดังต่อไปนี้

$$R_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{4.4.1}$$

โดยที่ $x, y, z$ คือ Cartesian Coordinates และ $i$ กับ $j$ คือเลข Index ของอะตอม

สำหรับการเขียนโปรแกรมเพื่อคำนวณ Distance นั้น เราจะต้องคำนึงถึงการจัดการหน่วยความจำ (Memory Allocation) ด้วยเพื่อให้โปรแกรมนั้นมีประสิทธิภาพมากที่สุดซึ่งนั่นก็จะเกี่ยวข้องกับการเลือก

วิธีในการเก็บข้อมูลของระยะห่างระหว่างอะตอม ในการหาความยาวระหว่างอะตอมของทุก ๆ คู่นั้นสามารถใช้เมทริกซ์มาเก็บข้อมูลได้ซึ่งขนาดของเมทริกซ์จะเป็น $N \times N$ โดยที่ $N$ คือจำนวนอะตอม ในการใช้เมทริกซ์นั้นเราจำเป็นที่จะต้องจัดสรร (Allocate) หน่วยความจำซึ่งทำได้สองวิธีคือ

- 1. ใช้ Static Two-Dimensional Array

```
double R[50][50];
```

- 2. ใช้ Dynamic Allocation โดยการใช้คลาส `Molecule`

```
double **R = new double* [mol.natom];
for(int i=0; i < mol.natom; i++)
  R[i] = new double[mol.natom];
```

แล้วก็อย่าลืมลบหน่วยความจำหลังจากที่คำนวณเสร็จแล้วด้วย ดังนี้

```
for(int i=0; i < mol.natom; i++)
  delete[] R[i];
delete[] R;
```

ขั้นตอนต่อไปก็คือการสร้างเมทริกซ์ขึ้นมาซึ่งสามารถทำได้โดยใช้ For Loop ในการวนไปเรื่อย ๆ ตามเลข Index ของอะตอม

แล้วก็อย่าลืมลบหน่วยความจำหลังจากที่คำนวณเสร็จแล้วด้วย ดังนี้

```
...
#include <cmath>
...

for(int i=0; i < mol.natom; i++) {
  for(int j=0; j < mol.natom; j++) {
      R[i][j] =
        sqrt(

    (mol.geom[i][0]-mol.geom[j][0])*(mol.geom[i][0]-mol.geom[j][0])
        +
    (mol.geom[i][1]-mol.geom[j][1])*(mol.geom[i][1]-mol.geom[j][1])
        +
    (mol.geom[i][2]-mol.geom[j][2])*(mol.geom[i][2]-mol.geom[j][2])
        );
  }
}
```

ลำดับต่อไปคือการแสดงค่าความยาวที่คำนวณได้โดยเราก็จะใช้ For Loop เหมือนเดิม

```
for(int i=0; i < mol.natom; i++)
  for(int j=0; j < i; j++)
    printf("%d %d %8.5f\n", i, j, R[i][j]);
```

ขั้นตอนสุดท้ายคือการทำให้โค้ดนั้นมีความเรียบร้อยและมีประสิทธิภาพมากขึ้น โดยเราสามารถนำโค้ดด้านบนที่เราได้เขียนไว้สร้างเป็นฟังก์ชันใหม่ที่ชื่อว่า `bond()` ในคลาส `Molecule` ได้ดังนี้

```
double Molecule::bond(int a, int b)
{
  return sqrt( (geom[a][0]-geom[b][0])*(geom[a][0]-geom[b][0])
              + (geom[a][1]-geom[b][1])*(geom[a][1]-geom[b][1])
              + (geom[a][2]-geom[b][2])*(geom[a][2]-geom[b][2])
    );
}
```

แล้วก็โปรแกรมของเราก็จะมีหน้าตาเป็นแบบนี้

```
#include "molecule.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <cmath>

using namespace std;

int main()
{
  Molecule mol("geom.dat", 0);

  cout << "Number of atoms: " << mol.natom << endl;
  cout << "Input Cartesian coordinates:\n";
  mol.print_geom();

  cout << "\nInteratomic distances (bohr):\n";
  for(int i=0; i < mol.natom; i++)
    for(int j=0; j < i; j++)
      printf("%d %d %8.5f\n", i, j, mol.bond(i,j));

  return 0;
}
```

ซึ่งก็จะมีเอาต์พุตดังต่อไปนี้

```
Number of atoms: 7
Input Cartesian coordinates:
6     0.000000000000     0.000000000000     0.000000000000
6     0.000000000000     0.000000000000     2.845112131228
8     1.899115961744     0.000000000000     4.139062527233
1    -1.894048308506     0.000000000000     3.747688672216
1     1.942500819960     0.000000000000    -0.701145981971
1    -1.007295466862    -1.669971842687    -0.705916966833
1    -1.007295466862     1.669971842687    -0.705916966833

Interatomic distances (bohr):
1 0  2.84511
2 0  4.55395
2 1  2.29803
3 0  4.19912
3 1  2.09811
3 2  3.81330
4 0  2.06517
4 1  4.04342
4 2  4.84040
4 3  5.87463
5 0  2.07407
5 1  4.05133
5 2  5.89151
5 3  4.83836
5 4  3.38971
6 0  2.07407
6 1  4.05133
6 2  5.89151
6 3  4.83836
6 4  3.38971
6 5  3.33994
```

โดยที่ `bond()` คือ Member Function ของ `Molecule` ซึ่งเราสามารถเรียกใช้ข้อมูลของ Geometry (Coordinates) ได้ผ่าน `geom` โดยไม่จำเป็นต้องเรียกใช้ผ่าน `mol.geom`

### Step 3: คำนวณมุมพันธะ (Bond Angles)

เราสามารถคำนวณมุมพันธะทุกมุมที่เป็นไปได้ทั้งหมดของโมเลกุล (ระหว่างอะตอม $i, j, k$) $\phi_{ijk}$ ได้

ด้วยสมการดังต่อไปนี้

$$\cos\phi_{ijk} = \tilde{e}_{ji} \cdot \tilde{e}_{jk} \tag{4.4.2}$$

โดยที่ $e_{ij}$ คือเวกเตอร์หนึ่งหน่วย (Unit Vector) ระหว่างอะตอมซึ่งสามารถคำนวณได้ด้วยสมการดังต่อไปนี้

$$e_{ij}^{x} = \frac{-(x_i - x_j)}{R_{ij}}$$
$$e_{ij}^{y} = \frac{-(y_i - y_j)}{R_{ij}}$$
$$e_{ij}^{z} = \frac{-(z_i - z_j)}{R_{ij}}$$

โดยเราจะสร้างฟังก์ชันสำหรับคำนวณมุมพันธะซึ่งก็คือ Inverse ของฟังก์ชัน Cosine ของสมการที่ (4.4.2) แล้วเพิ่มฟังก์ชันนี้เข้าไปในไฟล์ `molecule.cpp` ดังนี้

```
// Returns the value of the unit vector between atoms a and b
// in the cart direction (cart=0=x, cart=1=y, cart=2=z)
double Molecule::unit(int cart, int a, int b)
{
  return -(geom[a][cart]-geom[b][cart])/bond(a,b);
}

// Returns the angle between atoms a, b, and c in radians
double Molecule::angle(int a, int b, int c)
{
  return acos(unit(0,b,a) * unit(0,b,c) + unit(1,b,a) *
   unit(1,b,c) + unit(2,b,a) * unit(2,b,c));
}
```

นอกจากนี้เราจะต้องเพิ่ม Declaration ของ Member Function เข้าไปในไฟล์ `molecule.h` ด้วย ดังนี้

```
#include <string>

using namespace std;

class Molecule
{
  public:
```

```
    int natom;
    int charge;
    int *zvals;
    double **geom;
    string point_group;

    void print_geom();
    void rotate(double phi);
    void translate(double x, double y, double z);
    double bond(int atom1, int atom2);
    double angle(int atom1, int atom2, int atom3);
    double torsion(int atom1, int atom2, int atom3, int atom4);
    double unit(int cart, int atom1, int atom2);

    Molecule(const char *filename, int q);
    ~Molecule();
};
```

แล้วโปรแกรมสำหรับคำนวณมุมพันธะที่สมบูรณ์นั้นจะมีหน้าตาแบบนี้

```
#include "molecule.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <cmath>

using namespace std;

int main()
{
  Molecule mol("geom.dat", 0);

  cout << "Number of atoms: " << mol.natom << endl;
  cout << "Input Cartesian coordinates:\n";
  mol.print_geom();

  cout << "\nBond angles:\n";
  for(int i=0; i < mol.natom; i++) {
    for(int j=0; j < i; j++) {
      for(int k=0; k < j; k++) {
        if(mol.bond(i,j) < 4.0 && mol.bond(j,k) < 4.0)
          printf("%2d-%2d-%2d %10.6f\n", i, j, k,
```

```
      mol.angle(i,j,k)*(180.0/acos(-1.0)));
24       }
25     }
26   }
27
28   return 0;
29 }
```

โดยที่เราใช้ `acos(-1.0)` ในการประมาณค่า $\pi$

เมื่อเรารันโปรแกรมด้านบนจะได้เอาต์พุตแบบนี้

```
1  Number of atoms: 7
2  Input Cartesian coordinates:
3  6     0.000000000000     0.000000000000     0.000000000000
4  6     0.000000000000     0.000000000000     2.845112131228
5  8     1.899115961744     0.000000000000     4.139062527233
6  1    -1.894048308506     0.000000000000     3.747688672216
7  1     1.942500819960     0.000000000000    -0.701145981971
8  1    -1.007295466862    -1.669971842687    -0.705916966833
9  1    -1.007295466862     1.669971842687    -0.705916966833
10
11 Bond angles:
12   2- 1- 0 124.268308
13   3- 1- 0 115.479341
14   3- 2- 1  28.377448
15   5- 4- 0  35.109529
16   6- 4- 0  35.109529
17   6- 5- 0  36.373677
18   6- 5- 4  60.484476
```

**Step 4: คำนวณมุมบิด (Torsion หรือ Dihedral Angles)**

พารามิเตอร์ลำดับต่อไปที่เราจะคำนวณก็คือมุมบิด (Torsion Angle) สำหรับมุมบิดของอะตอม 4 อะตอมใด ๆ ในโมเลกุล $i, j, k, l$ นั้นมีสมการในการคำนวณคือ

$$\cos \tau_{ijkl} = \frac{(\tilde{e}_{ij} \times \tilde{e}_{jk}) \cdot (\tilde{e}_{jk} \times \tilde{e}_{kl})}{\sin \theta_{ijk} \sin \theta_{jkl}} \tag{4.4.3}$$

สิ่งที่ต้องระวังในการคำนวณ Torsion Angle ก็คือเครื่องหมายของมุมซึ่งจะเป็นบวกหรือลบนั้นก็คือขึ้นอยู่กับว่าเวกเตอร์นั้นมีทิศทางไปทางไหนเมื่อเทียบกับระนาบ

- มุมบิดของอะตอม $i-j-k-l$ เป็นบวกเมื่อเวกเตอร์ตามแนวอะตอม $k-l$ นั้นวางตัวไปทางด้านขวาของระนาบที่สร้างจากอะตอม $i-j-k$ เมื่อมองจากทิศทางของเวกเตอร์ $j-k$

- มุมบิดของอะตอม $i-j-k-l$ เป็นลบเมื่อเวกเตอร์ตามแนวอะตอม $k-l$ นั้นวางตัวไปทางด้านซ้ายของระนาบที่สร้างจากอะตอม $i-j-k$ เมื่อมองจากทิศทางของเวกเตอร์ $j-k$

โปรแกรมสำหรับคำนวณ Torsion Angle นั้นมีดังนี้ เริ่มต้นด้วยการสร้าง Member Function ของคลาส `Molecule`

```
// Computes the angle between planes a-b-c and b-c-d
double Molecule::torsion(int a, int b, int c, int d)
{
  double eabc_x = (unit(1,b,a)*unit(2,b,c) -
   unit(2,b,a)*unit(1,b,c));
  double eabc_y = (unit(2,b,a)*unit(0,b,c) -
   unit(0,b,a)*unit(2,b,c));
  double eabc_z = (unit(0,b,a)*unit(1,b,c) -
   unit(1,b,a)*unit(0,b,c));

  double ebcd_x = (unit(1,c,b)*unit(2,c,d) -
   unit(2,c,b)*unit(1,c,d));
  double ebcd_y = (unit(2,c,b)*unit(0,c,d) -
   unit(0,c,b)*unit(2,c,d));
  double ebcd_z = (unit(0,c,b)*unit(1,c,d) -
   unit(1,c,b)*unit(0,c,d));

  double exx = eabc_x * ebcd_x;
  double eyy = eabc_y * ebcd_y;
  double ezz = eabc_z * ebcd_z;

  double tau = (exx + eyy + ezz)/(sin(angle(a,b,c)) *
   sin(angle(b,c,d)));

  if(tau < -1.0) tau = acos(-1.0);
  else if(tau > 1.0) tau = acos(1.0);
  else tau = acos(tau);

  // Compute the sign of the torsion
  double cross_x = eabc_y * ebcd_z - eabc_z * ebcd_y;
  double cross_y = eabc_z * ebcd_x - eabc_x * ebcd_z;
  double cross_z = eabc_x * ebcd_y - eabc_y * ebcd_x;
  double norm = cross_x*cross_x + cross_y*cross_y +
   cross_z*cross_z;
```

```
  cross_x /= norm;
  cross_y /= norm;
  cross_z /= norm;
  double sign = 1.0;
  double dot =
   cross_x*unit(0,b,c)+cross_y*unit(1,b,c)+cross_z*unit(2,b,c);
  if(dot < 0.0) sign = -1.0;

  return tau*sign;
}
```

แล้วเราก็เรียกใช้ฟังก์ชันใหม่ที่เราเพิ่งสร้างไว้ในโปรแกรมหลักของเราได้ดังนี้

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <cmath>
#include "molecule.h"

using namespace std;

int main()
{
  Molecule mol("geom.dat", 0);

  cout << "Number of atoms: " << mol.natom << endl;
  cout << "Input Cartesian coordinates:\n";
  mol.print_geom();

  cout << "\nTorsional angles:\n";
  for(int i=0; i < mol.natom; i++) {
    for(int j=0; j < i; j++) {
      for(int k=0; k < j; k++) {
        for(int l=0; l < k; l++) {
          if(mol.bond(i,j) < 4.0 && mol.bond(j,k) < 4.0 &&
   mol.bond(k,l) < 4.0)
            printf("%2d-%2d-%2d-%2d %10.6f\n", i, j, k, l,
   mol.torsion(i,j,k,l)*(180.0/acos(-1.0)));
        }
      }
    }
  }
```

```
29
30   return 0;
31 }
```

เมื่อเรารันโปรแกรมด้านบนเราจะได้เอาต์พุตดังนี้

```
Number of atoms: 7
Input Cartesian coordinates:
6       0.000000000000       0.000000000000       0.000000000000
6       0.000000000000       0.000000000000       2.845112131228
8       1.899115961744       0.000000000000       4.139062527233
1      -1.894048308506       0.000000000000       3.747688672216
1       1.942500819960       0.000000000000      -0.701145981971
1      -1.007295466862      -1.669971842687      -0.705916966833
1      -1.007295466862       1.669971842687      -0.705916966833

Torsional angles:
  3- 2- 1- 0 180.000000
  6- 5- 4- 0  36.366799
```

### Step 5: คำนวณจุดศูนย์กลางมวล มุมบิด (Center of Mass)

แบบฝึกหัด: ให้ลองเขียนโปรแกรมคำนวณจุดศูนย์กลางมวลของโมเลกุล โดยสมการที่ใช้ในการคำนวณนั้นสามารถดูได้จาก Wikipedia ครับ

เฉลย

```
#include "molecule.h"
#include "masses.h"

#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <cmath>

using namespace std;

int main()
{
  Molecule mol("geom.dat", 0);

  cout << "Number of atoms: " << mol.natom << endl;
  cout << "Input Cartesian coordinates:\n";
```

```
18  mol.print_geom();
19
20  cout << "Interatomic distances (bohr):\n";
21  for(int i=0; i < mol.natom; i++)
22    for(int j=0; j < i; j++)
23      printf("%d %d %8.5f\n", i, j, mol.bond(i,j));
24
25  cout << "\nBond angles:\n";
26  for(int i=0; i < mol.natom; i++) {
27    for(int j=0; j < i; j++) {
28      for(int k=0; k < j; k++) {
29        if(mol.bond(i,j) < 4.0 && mol.bond(j,k) < 4.0)
30          printf("%2d-%2d-%2d %10.6f\n", i, j, k,
     mol.angle(i,j,k)*(180.0/acos(-1.0)));
31      }
32    }
33  }
34
35  cout << "\nOut-of-Plane angles:\n";
36  for(int i=0; i < mol.natom; i++) {
37    for(int k=0; k < mol.natom; k++) {
38      for(int j=0; j < mol.natom; j++) {
39        for(int l=0; l < j; l++) {
40          if(i!=j && i!=k && i!=l && j!=k && k!=l &&
     mol.bond(i,k) < 4.0 &&
41             mol.bond(k,j) < 4.0 && mol.bond(k,l) < 4.0)
42             printf("%2d-%2d-%2d-%2d %10.6f\n", i, j, k, l,
     mol.oop(i,j,k,l)*(180.0/acos(-1.0)));
43        }
44      }
45    }
46  }
47
48  cout << "\nTorsional angles:\n\n";
49  for(int i=0; i < mol.natom; i++) {
50    for(int j=0; j < i; j++) {
51      for(int k=0; k < j; k++) {
52        for(int l=0; l < k; l++) {
53          if(mol.bond(i,j) < 4.0 && mol.bond(j,k) < 4.0 &&
     mol.bond(k,l) < 4.0)
54            printf("%2d-%2d-%2d-%2d %10.6f\n", i, j, k, l,
     mol.torsion(i,j,k,l)*(180.0/acos(-1.0)));
55        }
```

```
56        }
57      }
58    }
59
60    /* find the center of mass (COM) */
61    double M = 0.0;
62    for(int i=0; i < mol.atom; i++) M += an2masses[(int)
       mol.zvals[i]];
63
64    double xcm=0.0;
65    double ycm=0.0;
66    double zcm=0.0;
67    double mi;
68    for(int i=0; i < mol.atom; i++) {
69      mi = an2masses[(int) mol.zvals[i]];
70      xcm += mi * mol.geom[i][0];
71      ycm += mi * mol.geom[i][1];
72      zcm += mi * mol.geom[i][2];
73    }
74    xcm /= M;
75    ycm /= M;
76    zcm /= M;
77    printf("\nMolecular center of mass: %12.8f %12.8f %12.8f\n",
       xcm, ycm, zcm);
78
79    mol.translate(-xcm, -ycm, -zcm);
80
81    return 0;
82  }
```

เมื่อเรารันโปรแกรมด้านบนเราจะได้เอาต์พุตดังนี้

```
1  Number of atoms: 7
2  Input Cartesian coordinates:
3  6       0.000000000000       0.000000000000       0.000000000000
4  6       0.000000000000       0.000000000000       2.845112131228
5  8       1.899115961744       0.000000000000       4.139062527233
6  1      -1.894048308506       0.000000000000       3.747688672216
7  1       1.942500819960       0.000000000000      -0.701145981971
8  1      -1.007295466862      -1.669971842687      -0.705916966833
9  1      -1.007295466862       1.669971842687      -0.705916966833
10 Interatomic distances (bohr):
11 1 0  2.84511
```

```
 2 0  4.55395
 2 1  2.29803
 3 0  4.19912
 3 1  2.09811
 3 2  3.81330
 4 0  2.06517
 4 1  4.04342
 4 2  4.84040
 4 3  5.87463
 5 0  2.07407
 5 1  4.05133
 5 2  5.89151
 5 3  4.83836
 5 4  3.38971
 6 0  2.07407
 6 1  4.05133
 6 2  5.89151
 6 3  4.83836
 6 4  3.38971
 6 5  3.33994

Bond angles:
   0- 1- 2 124.268308
   0- 1- 3 115.479341
   0- 4- 5  35.109529
   0- 4- 6  35.109529
   0- 5- 6  36.373677
   1- 2- 3  28.377448
   4- 5- 6  60.484476

Out-of-plane angles:
   0- 3- 1- 2  -0.000000
   0- 6- 4- 5  19.939726
   0- 6- 5- 4 -19.850523
   0- 5- 6- 4  19.850523
   1- 5- 0- 4  53.678778
   1- 6- 0- 4 -53.678778
   1- 6- 0- 5  54.977064
   2- 3- 1- 0   0.000000
   3- 2- 1- 0  -0.000000
   4- 5- 0- 1 -53.651534
   4- 6- 0- 1  53.651534
   4- 6- 0- 5 -54.869992
```

```
  4- 6- 5- 0  29.885677
  4- 5- 6- 0 -29.885677
  5- 4- 0- 1  53.626323
  5- 6- 0- 1 -56.277112
  5- 6- 0- 4  56.194621
  5- 6- 4- 0 -30.558964
  5- 4- 6- 0  31.064344
  6- 4- 0- 1 -53.626323
  6- 5- 0- 1  56.277112
  6- 5- 0- 4 -56.194621
  6- 5- 4- 0  30.558964
  6- 4- 5- 0 -31.064344

Torsional angles:

  3- 2- 1- 0 180.000000
  6- 5- 4- 0  36.366799

Molecular center of mass:   0.64494926   0.00000000   2.31663792
```

## 4.5 เขียนโปรแกรม Hartree-Fock SCF (ภาษา Python)

### 4.5.1 ทำความเข้าใจขั้นตอน SCF กันก่อน

โปรแกรมเคมีควอนตัมทุกโปรแกรมจะต้องมีส่วนหนึ่งของโปรแกรมที่เป็นโค้ดสำหรับแก้สมการอันหนึ่ง ซึ่งขาดไม่ได้เลยนั่นก็คือ Roothaan-Hall Equation โดยใช้วิธี Self-Consistent Field ซึ่งเป็นสมการที่เราใช้ในการคำนวณหาพลังงานของระบบที่เราสนใจ

สมการ Roothaan-Hall นั้นจะเรียกว่าเป็นสมการ HF ที่แปลงร่างมาแล้วก็ได้ สาเหตุที่เราต้องทำการแปลงร่าง HF นั้นก็เพราะว่าเราเขียน Wavefunction ให้อยู่ในรูปที่มี Basis Function นั่นเอง (Basis Function คือสิ่งที่เราใช้อธิบายออร์บิทัลเชิงอะตอม) โดยเราจะมีสมการ Roothaan-Hall ทั้งหมด $m$ สมการ โดยที่ $m$ คือจำนวนของ Basis Function (เรามีสมการ HF 1 สมการต่อ 1 MO และการที่เรามี $m$ Basis Function นั้นก็จะสร้างทั้งหมด $m$ MOs ด้วย), $F$ คือ Fock Matrix ซึ่งก็จะได้จาก Density Matrix แล้ว Density Matrix ก็คือ Matrix ที่มีสมาชิกเป็นผลคูณระหว่าง Coefficients ของ MO ซึ่งก็จะได้มาจากการประมาณค่าด้วยวิธีเริ่มต้น (Initial Guess) แบบต่าง ๆ เช่นใช้ Hückel Method, ส่วน $S$ นั้นก็คือ Overlap Matrix ซึ่งก็จะเป็น Matrix ที่บอกว่า Orbitals นั้นสัมพันธ์กันมากน้อยแค่ไหน, และ $\epsilon$ ของแต่ละสมการนั้นก็คือค่าพลังงานของแต่ละ MO นั้นเองซึ่งเป็นสิ่งที่เราต้องการแก้สมการหามันออกมา ทีนี้เราสามารถยุบรวมสมการ HF สำหรับแต่ละ Basis Function เข้าด้วยกันได้ ซึ่งจะได้ออกมาเป็นตามสมการที่สั้นและกระชับขึ้นนั่นก็คือสมการ Roothaan-Hall นั่นเอง ดังนี้ (เขียนในรูปของเมทริกซ์)

$$\boldsymbol{FC} = \boldsymbol{SC\varepsilon} \tag{4.5.1}$$

โดยมี $F$, $C$, และ $S$ แทน Fock Matrix, MO Coefficients Matrix, และ Overlap Matrix ตามลำดับ โดย Matrix ทั้งสามอันนี้มีขนาดคือ $m \times m$ แล้วก็ $\epsilon$ นั้นก็มีขนาด $m \times m$ ด้วย (มีเฉพาะสมาชิกในแนวทแยงเท่านั้นที่มีค่าไม่เท่ากับ 0) โดย $C$ มีหน้าตาดังต่อไปนี้

$$\boldsymbol{C} = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,m} \\ c_{2,1} & c_{2,2} & \dots & c_{2,m} \\ \vdots & \vdots & & \vdots \\ c_{m,1} & c_{m,2} & \dots & c_{m,m} \end{pmatrix} \tag{4.5.2}$$

และ $\epsilon$

$$\boldsymbol{E} = \begin{pmatrix} \epsilon_1 & 0 & \dots & 0 \\ 0 & \epsilon_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \epsilon_m \end{pmatrix} \tag{4.5.3}$$

ซึ่งแนวคิดของการแก้ปัญหา SCF นั้นก็คือเราพยายามที่จะหาค่าของ $\boldsymbol{C}$ ที่จะทำให้มีพลังงานของระบบนั้นมีค่าน้อยที่สุดเท่าที่จะเป็นไปได้ (Energy Minimization) แต่ว่าเราดันมี $\boldsymbol{C}$ อยู่ทั้งสองข้างของสมการ ดังนั้นเราจึงต้องทำการแก้สมการด้านบนโดยการใช้การวนซ้ำ (Iteration) จนกว่าจะได้ $\boldsymbol{C}$ ของทั้งสองข้างของสมการที่เท่ากัน (หรือเกือบจะเท่ากัน) ซึ่งจะทำให้ได้ค่าพลังงาน $\epsilon$ ที่ต่ำที่สุดด้วย

ปัญหาถัดมาก็คือว่าสมการด้านบนนั้นมันเป็น Eigenvalue Problem ในฟอร์มที่ยังแก้ไม่ได้ (แก้ได้แต่ว่าจะได้คำตอบที่ไม่ถูกต้อง) ซึ่งถ้าหากเราสามารถเปลี่ยนจาก $\boldsymbol{FC} = \boldsymbol{SC\epsilon}$ ให้เป็น $\boldsymbol{FC} = \boldsymbol{C\epsilon}$ ได้ ก็จะทำให้เราสามารถแก้ Eigenvalue Problem ได้อย่างมีประสิทธิภาพและได้คำตอบที่ถูกต้อง ดังนั้นเราจึงมีเทคนิคในการแปลงโดยการทำ Orthogonalization ซึ่งถ้าสรุปสั้น ๆ ก็คือว่าเราสามารถแปลงออกมาได้เป็น

$$\boldsymbol{F'C'} = \boldsymbol{C'E} \tag{4.5.4}$$

โดยที่

$$\boldsymbol{F'} = \boldsymbol{S}^{-1/2}\boldsymbol{F}\boldsymbol{S}^{-1/2} \tag{4.5.5}$$

$$\boldsymbol{C'} = \boldsymbol{S}^{-1/2}\boldsymbol{C} \tag{4.5.6}$$

ตามวิชา Linear Algebra ถ้าเราต้องการหา Matrix ของ $\epsilon$ เราสามารถใช้วิธี Diagonalization กับสมการ

Roothaan-Hall ได้เพราะว่าสมการนี้มันเป็น Eigenfunction ซึ่งกระบวนการที่จะใช้ในการแก้นั้นก็คือตามขั้นตอนต่อไปนี้

**กระบวนการ SCF แบบคร่าว ๆ ตามทฤษฎี**

1. เรากำหนดตำแหน่งของอะตอมในโมเลกุล, ประจุ, และทำการเลือก Basis Set

2. เริ่มคำนวณพลังงานจลน์และพลังงานศักย์ รวมไปถึง Overlap Integral

3. คำนวณ Orthogonalizing Matrix โดยใช้ Overlap Matrix (Overlap Matrix ที่ถูกสร้างมาจาก Overlap Integral ที่ได้จากขั้นที่ 2 อีกที)

4. คำนวณ Fock Matrix อันแรกเลยโดยใช้พลังงานจลน์และพลังงานศักย์แล้วก็ใช้ Initial Guess ที่ได้จาก Coefficients ของ Basis Set ที่กำหนดไว้ตั้งแต่ขั้นตอนที่ 1

5. ใช้ Orthogonalizing Matrix เพื่อแปลง (Tranform) Fock Matrix ให้เป็น Matrix อันใหม่ (เรียกว่าเป็น $F'$ ก็แล้วกัน) ที่มันขึ้นอยู่กับ Orthogonal Set ของฟังก์ชันที่ได้มาจาก Basis Set ที่ถูกเซนเตอร์กับอะตอม (Atom-centered Basis Set)

6. ทำการทำเมทริกซ์แนวทแยง (Diagonalize) Fock Matrix เพื่อหา Coefficient Matrix ที่ถูกแปลงมาแล้ว (เรียกว่า $C'$) ก็ได้แล้วก็หาพลังงานของแต่ละ MO ได้

7. เปรียบเทียบ $C'$ และพลังงานกับค่าที่ได้จากรอบก่อนหน้านี้ ถ้าผลต่างยังไม่น้อยกว่า Cutoff ที่กำหนดไว้ก็ให้นำ Fock Matrix ที่ได้มาล่าสุดไปใช้ใหม่อีกครั้งหนึ่งในขั้นที่ 4 ทำวนไปเรื่อย ๆ จนได้คำตอบที่แม่นยำ

สำหรับการกำหนด Basis Set นั้นก็คือการเลือกชุดค่าสัมประสิทธิ์ของออร์บิทัล (Wavefunction) ของธาตุแต่ละธาตุ จริง ๆ แล้ว Basis Set นั้นก็คือไฟล์ Text ที่มีรูปแบบ (Format) ที่แตกต่างกันไปตามโปรแกรมที่ใช้ เราสามารถดาวน์โหลดไฟล์ Basis Set ได้ที่เว็บไซต์ https://www.basissetexchange.org/ ตัวอย่างเช่น Basis Set: aug-cc-PV7Z ของอะตอมคาร์บอน https://www.basissetexchange.org/basis/aug-cc-pv7z/format/nwchem/?version=0&elements=6

## 4.5.2 มาเขียนโปรแกรมคำนวณ SCF กันเถอะ

ในส่วนนี้ผู้อ่านจะได้ศึกษาการเขียนโปรแกรมคำนวณ SCF ด้วยภาษา Python ซึ่งโปรแกรมด้านล่างนี้เป็นโปรแกรมที่ดัดแปลงมาจากโปรแกรมคำนวณ SCF จากหนังสือ Modern Quantum Chemsitry: Introduction to Advanced Electronic Structure Theory เขียนโดย Attila Szabo และ Neil S. Ostlund[1] ซึ่งเป็นโปรแกรมที่คำนวณพลังงานของโมเลกุล $HeH^+$ ซึ่งมี Electronic Configuration ที่เหมือนกันกับ $H_2$

[1]ซอร์สโค้ดของโปรแกรมต้นฉบับเขียนด้วยภาษา Fortran สามารถดูไฟล์ได้ที่ http://www.ccl.net/cca/software/SOURCES/FORTRAN/szabo/

แต่จะมีความแตกต่างกันตรงที่ $HeH^+$ นั้นมีประจุ +2 อยู่ที่อะตอม He ทำให้โมเลกุลนี้ไม่มีความสมมาตร จึงต้องทำการคำนวณด้วยวิธีการวนซ้ำ

**Step 1: สร้างออร์บิทัล**

ขั้นตอนแรกสุดเลยในการเขียนโปรแกรม SCF ก็คือการสร้างออร์บิทัล ทำไมเราต้องสร้างออร์บิทัลก่อนล่ะ? ก็เพราะว่าเราออร์บิทัลนั้นจริง ๆ แล้วก็คือ “ฟังก์ชันคลื่นของอิเล็กตรอน 1 ตัว” นั่นเอง ซึ่งเราจะกำหนดให้มีเซทของออร์บิทัล 1 เซทที่ใช้ในการอธิบายโมเลกุลทั้งโมเลกุล โดยเราจะใช้ Atomic Orbitals ที่มีจุดศูนย์กลางอยู่ที่อะตอม และเรานำ Atomic Orbitals แต่ละอันมารวมกันให้เป็นฟังก์ชันคลื่นหรือ Delocalized Orbital ได้โดยการใช้ Linear Combination of Atomic Orbitals (LCAO) ดังนี้

$$\psi_i(\boldsymbol{r}_i) = \sum_{\mu}^{K} c_\mu \phi_{\mu I}(\boldsymbol{r}_i) \tag{4.5.7}$$

โดยที่ $K$ คือจำนวน Basis Functions, $i = 1, 2, \ldots, K$, และ $c$ คือพารามิเตอร์ที่เราจะต้องทำการปรับเพื่อลดค่าพลังงานของระบบโมเลกุล

เนื่องจากว่าวิธีการ Hartree-Fock นั้นใช้หลักการ Variation ซึ่งเป็นการหาฟังก์ชันคลื่นของระบบที่มีค่าพลังงานที่ไม่ว่ายังไงก็จะต้องมีค่าสูงกว่าพลังงานจริงของระบบ ข้อดีของ Variational Method ก็คือเราสามารถปรับพารามิเตอร์อะไรก็ได้อย่างเป็นระบบเพื่อให้ได้ฟังก์ชันคลื่นที่มีความถูกต้องมากขึ้นและให้ค่าพลังงานที่ลู่เข้าหรือ Converged

สำหรับ Atomic Orbitals ที่เราจะใช้นั้นมีชื่อเรียกว่า Slater Type Orbitals (STO) สำหรับออร์บิทัล $1s$ นั้นมีสมการดังต่อไปนี้

$$\phi^{Slater}(\boldsymbol{r}) = \left(\zeta^3/\pi\right)^{1/2} e^{-\zeta \boldsymbol{r}} \tag{4.5.8}$$

เราลองมาพลอต STO อันนี้กันเพื่อดูว่ามีหน้าตาเป็นยังไง เริ่มต้นโดยการอิมพอร์ตไลบารี่ที่เราต้องการก่อน

```
%matplotlib inline
import math
import numpy as np
import scipy.special as sp
import matplotlib.pyplot as plt
```

สร้างฟังก์ชันสำหรับพลอต STO

```
x = np.linspace(-5,5,num=1000)
r = abs(x)
zeta = 1.0
```

```
psi_STO = (zeta**3/np.pi)**(0.5)*np.exp(-zeta*r)

plt.figure(figsize=(4,3))
plt.plot(x,psi_STO)
```

Slater Type Orbitals (STO) นั้นจริง ๆ แล้วก็คือฟังก์ชันที่เป็นผลเฉลยของสมการชโรดิงเงอร์สำหรับไฮโดรเจนอะตอมนั่นเอง (อิเล็กตรอน 1 ตัว) แต่ว่าการคำนวณอินทิกรัลของ STO นั้นสิ้นเปลืองมาก วิธีหนึ่งที่เราใช้ในการประมาณฟังก์ชัน STO นั้นก็คือการใช้ผลรวมของ Contracted Gaussian Functions (CGF) ซึ่งการที่เราใช้ CGF แทนนั้นช่วยทำให้เราประหยัดเวลาในการคำนวณมากขึ้นเพราะว่าการคำนวณอินทิกรัลของผลคูณระหว่าง Gaussian Functions 2 อันนั้นค่อนข้างง่ายพอสมควร สมการต่อไปนี้คือ Gaussian Function (CGF) ของออร์บิทัล $1s$

$$\phi^{GF}(\alpha) = (2\alpha/\pi)^{3/4} exp(-\alpha r^2) \tag{4.5.9}$$

และผลรวมของ CGF คือ

$$\phi^{CGF}(\boldsymbol{r}) = \sum_n d_n \phi_n^{GF}(\alpha) \tag{4.5.10}$$

ซึ่งเราใช้ Gaussian Functions 3 อันในการหา Approximation ของ STO นั่นเอง ดังนี้

$$\phi_{STO-3G}^{CGF}(\boldsymbol{r}) = \sum_n^3 d_n \phi_n^{GF}(\alpha) \tag{4.5.11}$$

```
# Coeff is the d_n variable in the equation above
Coeff = np.array(
    [[1.00000, 0.0000000, 0.000000], [0.678914, 0.430129,
   0.000000], [0.444635, 0.535328, 0.154329]]
)

# Expon is the alpha variable in the equation above
Expon = np.array(
    [[0.270950, 0.000000, 0.000000], [0.151623, 0.851819,
   0.000000], [0.109818, 0.405771, 2.227660]]
)

psi_CGF_STO1G = Coeff[0, 0] * (2 * Expon[0, 0] / np.pi) **
   (0.75) * np.exp(-Expon[0, 0] * r**2)
psi_CGF_STO2G = (
    Coeff[1, 0] * (2 * Expon[1, 0] / np.pi) ** (0.75) *
   np.exp(-Expon[1, 0] * r**2)
```

```
    + Coeff[1, 1] * (2 * Expon[1, 1] / np.pi) ** (0.75) *
    np.exp(-Expon[1, 1] * r**2)
    + Coeff[1, 2] * (2 * Expon[1, 2] / np.pi) ** (0.75) *
    np.exp(-Expon[1, 2] * r**2)
)
psi_CGF_STO3G = (
    Coeff[2, 0] * (2 * Expon[2, 0] / np.pi) ** (0.75) *
    np.exp(-Expon[2, 0] * r**2)
    + Coeff[2, 1] * (2 * Expon[2, 1] / np.pi) ** (0.75) *
    np.exp(-Expon[2, 1] * r**2)
    + Coeff[2, 2] * (2 * Expon[2, 2] / np.pi) ** (0.75) *
    np.exp(-Expon[2, 2] * r**2)
)

# Plot the three functions
plt.figure(figsize=(5, 3))
plt.title("Approximations to a STO with CGF")
plt.plot(x, psi_STO, label="STO")
plt.plot(x, psi_CGF_STO1G, label="STO-1G")
plt.legend()
plt.figure(figsize=(5, 3))
plt.plot(x, psi_STO, label="STO")
plt.plot(x, psi_CGF_STO2G, label="STO-2G")
plt.legend()
plt.figure(figsize=(5, 3))
plt.plot(x, psi_STO, label="STO")
plt.plot(x, psi_CGF_STO3G, label="STO-3G")
plt.legend()
```

จะเห็นได้ว่ายิ่งจำนวนของ GF เพิ่มมากขึ้นเท่าไหร่ ฟังก์ชันนั้นก็จะมีความเข้าใกล้ความเป็น STO มากขึ้นเท่านั้น โดยตำแหน่งที่ $x = 0$ นั้นจะเห็นว่าเป็นตำแหน่งที่ Approximation นั้นแย่ที่สุด โดยเราจะเรียกบริเวณที่เป็นความคลาดเคลื่อนของ CGF จาก STO นี้ว่า Cusp

**Step 2: เตรียมส่วนผสมสำหรับการคำนวณ SCF**

- Step 2.1: กำหนดตัวแปรที่จะใช้ในการเก็บข้อมูลทางควอนตัมของโมเลกุล

```
# Make all variables 'global variables' accessible through out
    the whole program
global H, S, X, XT, TT, G, C, P, Oldp, F, Fprime, Cprime, E

H = np.zeros([2, 2])
S = np.zeros([2, 2])
```

```
X = np.zeros([2, 2])
XT = np.zeros([2, 2])
TT = np.zeros([2, 2, 2, 2])
G = np.zeros([2, 2])
C = np.zeros([2, 2])

P = np.zeros([2, 2])
Oldp = np.zeros([2, 2])
F = np.zeros([2, 2])
Fprime = np.zeros([2, 2])
Cprime = np.zeros([2, 2])
E = np.zeros([2, 2])

Energy = 0.0
Delta = 0.0

# Values below taken from 'Modern Quantum Chemistry' book
# Appendix B: Two-Electron Self-Consistent-Field Program
IOP = 2
N = 3
R = 1.4632
Zeta1 = 2.0925
Zeta2 = 1.24
Za = 2.0
Zb = 1.0
```

- Step 2.2: สร้างฟังก์ชันสำหรับคำนวณ Integrals ต่าง ๆ

การคำนวณ Integrals นั้นจะทำได้ง่ายขึ้นเยอะมาก ๆ เลยถ้าหากว่าเราใช้ฟังก์ชัน Gaussian ในการอธิบายออร์บิทัล ผมแนะนำให้ผู้อ่านศึกษา Appendix ของหนังสือ Modern Quantum Chemistry ครับซึ่งจะช่วยให้เข้าใจได้มากขึ้น

```
def F0(t):
  """
  F function for 1s orbital
  """
  if t < 1e-6:
      return 1.0 - t / 3.0
  else:
      return 0.5 * (np.pi / t) ** 0.5 * sp.erf(t**0.5)
```

ฟังก์ชันสำหรับคำนวณ Overlap Matrix

```
def S_int(A, B, Rab2):
  """
  Calculates the overlap between two gaussian functions
  """
  return (np.pi / (A + B)) ** 1.5 * np.exp(-A * B * Rab2 / (A +
   B))
```

ฟังก์ชันสำหรับคำนวณ Kinetic Energy Integrals

```
def T_int(A, B, Rab2):
  """
  Calculates the kinetic energy integrals for un-normalised
   primitives
  """
  return (
      A
      * B
      / (A + B)
      * (3.0 - 2.0 * A * B * Rab2 / (A + B))
      * (np.pi / (A + B)) ** 1.5
      * np.exp(-A * B * Rab2 / (A + B))
  )
```

ฟังก์ชันสำหรับคำนวณ Nuclear Attraction Integrals

```
def V_int(A, B, Rab2, Rcp2, Zc):
  """
  Calculates the un-normalised nuclear attraction integrals
  """
  V = 2.0 * np.pi / (A + B) * F0((A + B) * Rcp2) * np.exp(-A * B
   * Rab2 / (A + B))

  return -V * Zc
```

ฟังก์ชันสำหรับคำนวณความคลาดเคลื่อน

```
def erf(t):
  """
  Approximation for the error function
  """
  P = 0.3275911
  A = [0.254829592, -0.284496736, 1.421413741, -1.453152027,
   1.061405429]
  T = 1.0 / (1 + P * t)
```

```
    Tn = T
    Poly = A[0] * Tn

    for i in range(1, 5):
        Tn = Tn * T
        Poly = Poly * A[i] * Tn

    return 1.0 - Poly * np.exp(-t * t)
```

ฟังก์ชันสำหรับคำนวณ Two-Electron Integrals

```
def TwoE(A, B, C, D, Rab2, Rcd2, Rpq2):
    """
    Calculate Two-Electron integrals
    A, B, C, D are the exponents alpha, beta, etc.
    Rab2 equals squared distance between center A and center B
    """
    return (
        2.0
        * (np.pi**2.5)
        / ((A + B) * (C + D) * np.sqrt(A + B + C + D))
        * F0((A + B) * (C + D) * Rpq2 / (A + B + C + D))
        * np.exp(-A * B * Rab2 / (A + B) - C * D * Rcd2 / (C + D))
    )
```

สร้างฟังก์ชันสำหรับเรียกใช้ฟังก์ชันด้านบนที่เราได้เขียนไว้ ประกาศตัวแปรทั้งหมดที่ต้องใช้และทำการรวม Integrals ทั้งหมด (เราจะเรียกใช้ฟังก์ชันนี้ภายหลัง)

```
def Intgrl(N, R, Zeta1, Zeta2, Za, Zb):
    """
    Declares the variables and compiles the integrals
    """

    global S12, T11, T12, T22, V11A, V12A, V22A, V11B, V12B, V22B,
     V1111, V2111, V2121, V2211, V2221, V2222

    S12 = 0.0
    T11 = 0.0
    T12 = 0.0
    T22 = 0.0
    V11A = 0.0
    V12A = 0.0
    V22A = 0.0
    V11B = 0.0
```

```
    V12B = 0.0
    V22B = 0.0
    V1111 = 0.0
    V2111 = 0.0
    V2121 = 0.0
    V2211 = 0.0
    V2221 = 0.0
    V2222 = 0.0

    R2 = R * R

    # The coefficients for the contracted Gaussian functions are
     below
    Coeff = np.array(
        [[1.00000, 0.0000000, 0.000000], [0.678914, 0.430129,
     0.000000], [0.444635, 0.535328, 0.154329]]
    )

    Expon = np.array(
        [[0.270950, 0.000000, 0.000000], [0.151623, 0.851819,
     0.000000], [0.109818, 0.405771, 2.227660]]
    )
    D1 = np.zeros([3])
    A1 = np.zeros([3])
    D2 = np.zeros([3])
    A2 = np.zeros([3])

    # This loop constructs the contracted Gaussian functions
    for i in range(N):
        A1[i] = Expon[N - 1, i] * (Zeta1**2)
        D1[i] = Coeff[N - 1, i] * ((2.0 * A1[i] / np.pi) ** 0.75)
        A2[i] = Expon[N - 1, i] * (Zeta2**2)
        D2[i] = Coeff[N - 1, i] * ((2.0 * A2[i] / np.pi) ** 0.75)

    # Calculate one electron integrals
    # Centre A is first atom centre B is second atom
    # Origin is on second atom
    # V12A - off diagonal nuclear attraction to centre A etc.
    for i in range(N):
        for j in range(N):
            # Rap2 - squared distance between centre A and centre P
            Rap = A2[j] * R / (A1[i] + A2[j])
            Rap2 = Rap**2
```

```
            Rbp2 = (R - Rap) ** 2
            S12 = S12 + S_int(A1[i], A2[j], R2) * D1[i] * D2[j]
            T11 = T11 + T_int(A1[i], A1[j], 0.0) * D1[i] * D1[j]
            T12 = T12 + T_int(A1[i], A2[j], R2) * D1[i] * D2[j]
            T22 = T22 + T_int(A2[i], A2[j], 0.0) * D2[i] * D2[j]
            V11A = V11A + V_int(A1[i], A1[j], 0.0, 0.0, Za) *
    D1[i] * D1[j]
            V12A = V12A + V_int(A1[i], A2[j], R2, Rap2, Za) *
    D1[i] * D2[j]
            V22A = V22A + V_int(A2[i], A2[j], 0.0, R2, Za) * D2[i]
    * D2[j]
            V11B = V11B + V_int(A1[i], A1[j], 0.0, R2, Zb) * D1[i]
    * D1[j]
            V12B = V12B + V_int(A1[i], A2[j], R2, Rbp2, Zb) *
    D1[i] * D2[j]
            V22B = V22B + V_int(A2[i], A2[j], 0.0, 0.0, Zb) *
    D2[i] * D2[j]

    # Calculate two electron integrals

    for i in range(N):
        for j in range(N):
            for k in range(N):
                for l in range(N):
                    Rap = A2[i] * R / (A2[i] + A1[j])
                    Rbp = R - Rap
                    Raq = A2[k] * R / (A2[k] + A1[l])
                    Rbq = R - Raq
                    Rpq = Rap - Raq
                    Rap2 = Rap * Rap
                    Rbp2 = Rbp * Rbp
                    Raq2 = Raq * Raq
                    Rbq2 = Rbq * Rbq
                    Rpq2 = Rpq * Rpq
                    V1111 = (
                        V1111
                        + TwoE(A1[i], A1[j], A1[k], A1[l], 0.0,
    0.0, 0.0) * D1[i] * D1[j] * D1[k] * D1[l]
                    )
                    V2111 = (
                        V2111
                        + TwoE(A2[i], A1[j], A1[k], A1[l], R2,
    0.0, Rap2) * D2[i] * D1[j] * D1[k] * D1[l]
```

```
                    )
                    V2121 = (
                        V2121 + TwoE(A2[i], A1[j], A2[k], A1[l],
    R2, R2, Rpq2) * D2[i] * D1[j] * D2[k] * D1[l]
                    )
                    V2211 = (
                        V2211 + TwoE(A2[i], A2[j], A1[k], A1[l],
    0.0, 0.0, R2) * D2[i] * D2[j] * D1[k] * D1[l]
                    )
                    V2221 = (
                        V2221
                        + TwoE(A2[i], A2[j], A2[k], A1[l], 0.0,
    R2, Rbq2) * D2[i] * D2[j] * D2[k] * D1[l]
                    )
                    V2222 = (
                        V2222
                        + TwoE(A2[i], A2[j], A2[k], A2[l], 0.0,
    0.0, 0.0) * D2[i] * D2[j] * D2[k] * D2[l]
                    )
    return
```

สร้างฟังก์ชันสำหรับนำ Integrals ที่คำนวณได้ใส่เข้าไปใน Array (เราจะเรียกใช้ฟังก์ชันนี้ภายหลัง)

```
def Collect():
    """
    Takes the basic integrals and assembles the relevant matrices,
    that are S, H, X, XT and two electron integrals
    """
    # Form core hamiltonian
    H[0, 0] = T11 + V11A + V11B
    H[0, 1] = T12 + V12A + V12B
    H[1, 0] = H[0, 1]
    H[1, 1] = T22 + V22A + V22B

    # Form overlap matrix
    S[0, 0] = 1.0
    S[0, 1] = S12
    S[1, 0] = S12
    S[1, 1] = 1.0

    # This is S^-1/2
    X[0, 0] = 1.0 / np.sqrt(2.0 * (1.0 + S12))
    X[1, 0] = X[0, 0]
```

```
21    X[0, 1] = 1.0 / np.sqrt(2.0 * (1.0 - S12))
22    X[1, 1] = -X[0, 1]
23
24    # This is the coulomb and exchange term (aa|bb) and (ab|ba)
25    TT[0, 0, 0, 0] = V1111
26    TT[1, 0, 0, 0] = V2111
27    TT[0, 1, 0, 0] = V2111
28    TT[0, 0, 1, 0] = V2111
29    TT[0, 0, 0, 1] = V2111
30    TT[1, 0, 1, 0] = V2121
31    TT[0, 1, 1, 0] = V2121
32    TT[1, 0, 0, 1] = V2121
33    TT[0, 1, 0, 1] = V2121
34    TT[1, 1, 0, 0] = V2211
35    TT[0, 0, 1, 1] = V2211
36    TT[1, 1, 1, 0] = V2221
37    TT[1, 1, 0, 1] = V2221
38    TT[1, 0, 1, 1] = V2221
39    TT[0, 1, 1, 1] = V2221
40    TT[1, 1, 1, 1] = V2222
```

### Step 3: สร้างตัวคำนวณ SCF Calculator

สร้างฟังก์ชันสำหรับการทำ Diagonalization ของ Fock Matrix ซึ่งเราจะได้ Eigenvectors และ Eigenvalues ออกมา ซึ่งก็คือ Array C กับ Array E ตามลำดับ

```
1  def Diag(Fprime, Cprime, E):
2    """
3    Diagonalises F to give eigenvectors in C and eigenvalues in E
4    theta is the angle describing the solution
5    """
6    # Angle for heteronuclear diatonic
7    Theta = 0.5 * math.atan(2.0 * Fprime[0, 1] / (Fprime[0, 0] -
       Fprime[1, 1]))
8    # print('Theta', Theta)
9
10   Cprime[0, 0] = np.cos(Theta)
11   Cprime[1, 0] = np.sin(Theta)
12   Cprime[0, 1] = np.sin(Theta)
13   Cprime[1, 1] = -np.cos(Theta)
14
15   E[0, 0] = (
16       Fprime[0, 0] * np.cos(Theta) ** 2
```

```
        + Fprime[1, 1] * np.sin(Theta) ** 2
        + Fprime[0, 1] * np.sin(2.0 * Theta)
    )
    E[1, 1] = (
        Fprime[1, 1] * np.cos(Theta) ** 2
        + Fprime[0, 0] * np.sin(Theta) ** 2
        - Fprime[0, 1] * np.sin(2.0 * Theta)
    )

    if E[1, 1] <= E[0, 0]:
        Temp = E[1, 1]
        E[1, 1] = E[0, 0]
        E[0, 0] = Temp
        Temp = Cprime[0, 1]
        Cprime[0, 1] = Cprime[0, 0]
        Cprime[0, 0] = Temp
        Temp = Cprime[1, 1]
        Cprime[1, 1] = Cprime[1, 0]
        Cprime[1, 0] = Temp
    return
```

ฟังก์ชันสุดท้ายคือฟังก์ชันที่จะเป็นการรันการคำนวณ SCF

```
def SCF(R, Za, Zb, G):
    """
    Performs the SCF iterations
    """
    Crit = 1e-11  # Convergence critera
    Maxit = 250  # Maximum number of iterations
    Iter = 0

    #--------
    # STEP 1. Guess an initial density matrix
    #--------
    # Use core hamiltonian for initial guess of F (P = 0)
    P = np.zeros([2, 2])

    Energy = 0.0

    while Iter < Maxit:
        Iter += 1

        #--------
```

```
21        # STEP 2. Calculate the Fock matrix
22        #--------
23        # Form two electron part of Fock matrix from P
24        # This is the two electron contribution in the equations
      above
25        G = np.zeros([2, 2])
26        for i in range(2):
27            for j in range(2):
28                for k in range(2):
29                    for l in range(2):
30                        G[i, j] = G[i, j] + P[k, l] * (TT[i, j, k,
      l] - 0.5 * TT[i, l, k, j])
31
32        # Add core hamiltonian H^CORE to get Fock matrix
33        F = H + G
34
35        # Calculate the electronic energy
36        Energy = np.sum(0.5 * P * (H + F))
37
38        #--------
39        # STEP 3. Calculate F'
40        # (remember S^-1/2 is X and S^1/2 is X.T)
41        #--------
42        G = np.matmul(F, X)
43        Fprime = np.matmul(X.T, G)
44
45        #--------
46        # STEP 4. Solve the eigenvalue problem
47        #--------
48        # Diagonalise transformed Fock matrix
49        Diag(Fprime, Cprime, E)
50
51        #--------
52        # STEP 5. Calculate the molecular orbitals coefficients
53        #--------
54        # Transform eigen vectors to get matrix C
55        C = np.matmul(X, Cprime)
56
57        # STEP 6. Calculate the new density matrix from the old P
58        Oldp = np.array(P)
59        P = np.zeros([2, 2])
60
61        # Form new density matrix
```

```
62        for i in range(2):
63            for j in range(2):
64                # Save present density matrix before creating a
    new one
65                for k in range(1):
66                    P[i, j] += 2.0 * C[i, k] * C[j, k]
67
68        #--------
69        # STEP 7. Check to see if the energy has converged
70        #--------
71        Delta = 0.0
72        # Calculate delta the difference between the old and
73        # new density matrix (Old P and new P)
74        Delta = P - Oldp
75        Delta = np.sqrt(np.sum(Delta**2) / 4.0)
76
77        print("Step ", Iter, " Elec Energy", Energy, " Diff",
    Delta)
78
79        # Check for convergence
80        if Delta < Crit:
81            # Add nuclear repulsion to get the total energy
82            Energy_tot = Energy + Za * Zb / R
83            print("")
84            print("Calculation converged with electronic energy:",
    Energy)
85            print("Calculation converged with total energy:",
    Energy_tot)
86            print("Density matrix")
87            print(P)
88            print("Mulliken populations")
89            print(np.matmul(P, S))
90            print("Coeffients")
91            print(C)
92
93            break
```

### Step 4: รันการคำนวณ SCF และตรวจสอบความถูกต้อง

และแล้วเราก็มาถึงขั้นตอนสุดท้ายนั่นก็คือการรันการคำนวณ SCF โดยสรุปอีกครั้งหนึ่งว่าขั้นตอนการคำนวณนั้นมีอัลกอริทึมดังต่อไปนี้

1. ทำการเดา Initial Density Matrix $\boldsymbol{P}$ โดยในตัวอย่างโค้ดของเรานั้นจะใช้ $\boldsymbol{P} = 0$ เพื่อความง่าย

2. นำ $\boldsymbol{P}$ มาสร้าง Fock Matrix $\boldsymbol{F}$
3. คำนวณ $\boldsymbol{F'} = \boldsymbol{S}^{-1/2}\boldsymbol{F}\boldsymbol{S}^{1/2}$
4. แก้สมการ Eigenvalue Problem โดยการนำ Secular Equation $|\boldsymbol{F'} - \boldsymbol{EI}| = 0$ มาทำ Diagonalization ซึ่งเราจะได้ $\boldsymbol{E}$ และ $\boldsymbol{C}$ ออกมา
5. คำนวณ Density Matrix $\boldsymbol{P}$ อันใหม่จาก $\boldsymbol{C}$
6. ตรวจสอบเงื่อนไขว่าพลังงานนั้น Converged แล้วหรือยัง ถ้าหากว่ายังไม่ Converged ให้วนซ้ำขั้นตอนด้านบน

เราเริ่มด้วยการเรียกใช้ฟังก์ชัน `Intgrl` เพื่อคำนวณ One-Electron Intregral และ One-Electron Integral และตามด้วยฟังก์ชัน `Collect` และฟังก์ชัน `SCF` ตามลำดับ

```
# Calculate one and two electron integrals
Intgrl(N, R, Zeta1, Zeta2, Za, Zb)

# Put all integals into array
Collect()

# Perform the SCF calculation
SCF(R, Za, Zb, G)
```

เมื่อรันแล้วจะได้เอาต์พุตของการคำนวณ SCF ของระบบโมเลกุล $HeH^+$ ดังนี้

```
Step  1  Elec Energy 0.0  Diff 0.8828668530136917
Step  2  Elec Energy -4.141862876133925  Diff 0.2791763040686421
Step  3  Elec Energy -4.22649189912899  Diff 0.029661780077229444
Step  4  Elec Energy -4.227522925343759  Diff
    0.0023182848695558057
Step  5  Elec Energy -4.227529268100319  Diff
    0.00017439769686161983
Step  6  Elec Energy -4.227529304014095  Diff
    1.3079512369092073e-05
Step  7  Elec Energy -4.227529304216109  Diff
    9.807146593527476e-07
Step  8  Elec Energy -4.227529304217244  Diff
    7.353368030466615e-08
Step  9  Elec Energy -4.22752930421725  Diff
    5.5135252472270695e-09
Step  10  Elec Energy -4.227529304217251  Diff
    4.1340208808251866e-10
```

```
Step  11  Elec Energy -4.227529304217252  Diff
    3.099665019964731e-11
Step  12  Elec Energy -4.227529304217252  Diff
    2.3244357915420586e-12

Calculation converged with electronic energy: -4.227529304217252
Calculation converged with total energy: -2.86066216370331
Density matrix
[[1.28614168 0.54017322]
  [0.54017322 0.22687011]]
Mulliken populations
[[1.52963579 1.11992783]
  [0.64243955 0.47036421]]
Coeffients
[[ 0.80191698 -0.78226577]
  [ 0.33680121  1.06844537]]
```

จากผลการคำนวณด้านบนจะเห็นได้ว่าพลังงานของโมเลกุล $HeH^+$ ที่คำนวณออกมาได้นั้นมีค่าลดลงเรื่อย ๆ โดยผ่านการคำนวณ SCF ทั้งหมด 12 ครั้ง พลังงานรวมของโมเลกุลที่คำนวณได้คือ -2.86066216370331 Hartree

แหล่งความรู้สำหรับอ่านเพิ่มเติม

1. ถ้าอยากศึกษาทฤษฎี Hartree-Fock รวมถึง Variational Principle และ Basis Sets ผมแนะนำให้อ่านเลคเชอร์โน๊ตของคอร์ส Advanced Computational Chemistry[1]

2. ถ้าอยากศึกษาการพิสูจน์ทฤษฎี Hartree-Fock แบบละเอียด ผมแนะนำให้อ่าน “Derivation of Hartree-Fock Theory” โดย Arvi Rauk[2]

3. ถ้าอยากศึกษาการคำนวณ Molecular Integrals แบบละเอียด ผมแนะนำให้อ่าน “Fundamentals of Molecular Integrals Evaluation” โดย Justin T. Fermann และ Edward F. Valeev

## 4.6 เขียนโปรแกรม Direct Inversion of the Iterative Subspace

### 4.6.1 ไอเดียเริ่มต้นของ DIIS

Direct Inversion of the Iterative Subspace (DIIS) เป็นวิธีที่พัฒนาโดยศาสตราจารย์ Peter Pulay เพื่อเพิ่มความเร็วของการลู่เข้า (Convergence) ของการคำนวณ SCF[26] โดยมีไอเดียเริ่มต้นก็คือการใช้

[1] http://www.chem.helsinki.fi/~manninen/lecture_notes.pdf
[2] https://onlinelibrary.wiley.com/doi/10.1002/0471220418.app0a

Residual (หรือเรียกว่า Error ก็ได้) ซึ่งเป็นผลต่างระหว่าง Trial Vecotrs $\{P^i\}$ ระหว่างรอบ (Iteration)

$$\mathbf{e}^i = \mathbf{p}^{i+1} - \mathbf{p}^i \tag{4.6.1}$$

ซึ่ง DIIS นั้นจะมองว่าคำตอบสุดท้ายที่ได้จากการหาคำตอบด้วยกระบวนการ SCF นั้นสามารถถูก Extrapolate ให้อยู่ในรูปของผลรวมเชิงเส้น (Linear Combination) ของปริภูมิย่อย (Subspace) ของรอบการคำนวณปัจจุบันซึ่งได้มาจากการคำนวณในรอบก่อนหน้า ดังนี้

$$\mathbf{p} = \sum_{i=1}^{m} c_i \mathbf{p}^i \tag{4.6.2}$$

โดยที่เราสามารถคำนวณสัมประสิทธ์ $c$ ได้จากการลดค่าความคลาดเคลื่อนซึ่งได้จาก

$$\mathbf{e} = \sum_{i=1}^{m} c_i \mathbf{e}^i \tag{4.6.3}$$

ซึ่งถ้าหากว่าเราสังเกตดี ๆ แล้วจะพบว่าเราสามารถทำการลดค่าความคลาดเคลื่อน (Error minimization) ได้จาก

$$\mathbf{p} = \sum_{i=1}^{m} c_i \left(\mathbf{p}^f + \mathbf{e}^i\right) \tag{4.6.4}$$

$$= \mathbf{p}^f \sum_{i=1}^{m} c_i + \sum_{i=1}^{m} c_i \mathbf{e}^i \tag{4.6.5}$$

แล้วก็ในกรณีที่ $\mathbf{p} = \mathbf{p}^f$ เราจะได้ว่า

$$\sum_{i=1}^{m} c_i = 1 \qquad \text{และ} \qquad \sum_{i=1}^{m} c_i \mathbf{e}^i = 0 \tag{4.6.6}$$

ทีนี้เราก็มีการกำหนด Lagrangian ขึ้นมาเพื่อใช้ในการลดค่า Norm ของเวกเตอร์ความคลาดเคลื่อน (Error Vector) ด้วย ($\langle \mathbf{e}|\mathbf{e}\rangle = \sum_{i,j=1}^{m} c_i^* c_j \langle e_i|e_j\rangle$) ดังนี้

$$\mathcal{L} = \mathbf{c}^\dagger \mathbf{B} \mathbf{c} - \lambda \left(1 - \sum_{i=1}^{m} c_i\right) \tag{4.6.7}$$

โดยที่ $\mathbf{B}_{ij} = \mathbf{e}_i \cdot \mathbf{e}_j$

$$\frac{\partial \mathcal{L}}{\partial c_k} = 0 \tag{4.6.8}$$

$$= 2\sum_{i=1}^{m} c_i B_{ki} - \lambda \tag{4.6.9}$$

แล้วเราก็ทำการแก้ชุดสมการเชิงเส้นดังต่อไปนี้เพื่อหาสัมประสิทธิ์สำหรับ DIIS Extrapolation

$$\begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \dots & \mathbf{B}_{1m} & -1 \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \dots & \mathbf{B}_{2m} & -1 \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{B}_{m1} & \mathbf{B}_{m2} & \dots & \mathbf{B}_{mm} & -1 \\ -1 & -1 & \dots & -1 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_m \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ -1 \end{pmatrix} \tag{4.6.10}$$

### 4.6.2 รู้จักกับ Commutator-DIIS

Commutator-DIIS หรือ C-DIIS เป็นเทคนิคที่นำ DIIS มาปรับปรุงเพื่อให้มีประสิทธิมากขึ้นโดยมีการปรับเปลี่ยนเงื่อนไขที่ใช้ในการสร้าง Residual หรือ Error[27] นั่นก็คือแทนที่จะใช้ Trial Vector ก็เปลี่ยนมาใช้ Fock Matrix และ Density Matrix แทน เพื่อที่เราจะได้นำไปใช้ในการหาคำตอบของ Hartree-Fock ได้โดยการใช้กระบวนการ SCF โดยเงื่อนไขที่เมทริกซ์ทั้งหมดอันจะต้องมีนั้นก็คือ Commutativity นั่นก็คือ Fock Matrix $\mathbf{F}'$ นั้นจะต้อง Commute กับ $\mathbf{P}'$[1] ซึ่งสามารถเขียนเงื่อนไขของการ Commute ด้วยสมการดังต่อไปนี้[2]

$$[\mathbf{F}', \mathbf{P}'] = \mathbf{F}'\mathbf{P}' - \mathbf{P}'\mathbf{F}' = 0 \tag{4.6.11}$$

ดังนั้น Error ที่เกิดขึ้นระหว่างแต่ละรอบของกระบวนการ SCF คือ

$$\mathbf{e}_i = \mathbf{F}'_i\mathbf{P}'_i - \mathbf{P}'_i\mathbf{F}'_i \tag{4.6.12}$$

ทีนี้เราก็มาพิจารณาสมการ Roothaan, $\mathbf{FC} = \mathbf{SC}\varepsilon$, ซึ่งเราสามารถทำ Transformation หรือการเปลี่ยนรูปเพื่อให้เป็น Eigenvalue Problem (สมการไอเกน) ที่อยู่ในรูปของ MO Basis ได้โดยการใช้เงื่อนไข Orthonormality นั่นคือ $\mathbf{C}^\dagger\mathbf{SC} = \mathbf{1}$ ดังนี้

[1] การ Commute ก็คือ $\mathbf{AB} = \mathbf{BA}$ หรือ $\mathbf{AB} - \mathbf{BA} = 0$

[2] ในหัวข้อนี้ผมใช้ $\mathbf{F}', \mathbf{P}'$ สำหรับเมทริกซ์ที่อยู่ในรูปของ MO Basis และใช้ $\mathbf{F}, \mathbf{P}$ สำหรับเมทริกซ์ที่อยู่ในรูปของ AO Basis

$$\mathbf{F}'\mathbf{C}' = \mathbf{C}'\varepsilon \tag{4.6.13}$$

โดยที่

$$\mathbf{F}' = \mathbf{S}^{-\frac{1}{2}\dagger}\mathbf{F}\,\mathbf{S}^{-\frac{1}{2}} \quad \text{และ} \quad \mathbf{C} = \mathbf{S}^{-\frac{1}{2}}\mathbf{C}' \tag{4.6.14}$$

ซึ่งจะทำให้เราได้ Density Matrix ที่อยู่ในรูปของ AO Basis ดังนี้

$$\mathbf{P} = \mathbf{C}\mathbf{C}^{\dagger} \tag{4.6.15}$$

และในรูปของ MO Basis ดังนี้

$$\mathbf{P}' = \mathbf{C}'\mathbf{C}'^{\dagger} \tag{4.6.16}$$

แล้วเราก็จะได้การแปลง Density Matrix จากรูปของ MO Basis ไปเป็น AO Basis ดังนี้

$$\mathbf{P}' = (\mathbf{S}^{-\frac{1}{2}})^{-1}\mathbf{P}\,(\mathbf{S}^{-\frac{1}{2}\dagger})^{-1} \tag{4.6.17}$$

และเพื่อให้ง่ายกว่านี้ เราจะกำหนดตัวแปรใหม่ขึ้นมาคือ $\mathbf{X} = \mathbf{S}^{-\frac{1}{2}}$ โดยที่ $\mathbf{X}$ นั้นเป็น Hermitian ดังนั้นเราจะได้ว่า

$$\mathbf{F}' = \mathbf{X}\mathbf{F}\mathbf{X} \tag{4.6.18}$$

$$\mathbf{P}' = \mathbf{X}^{-1}\mathbf{P}\mathbf{X}^{-1} \tag{4.6.19}$$

ถ้าหากว่าเราทำการแทนสมการด้านบนนี้เข้าไปใน Commutator ที่อยู่ในรูปของ MO Basis เราจะได้ว่า

$$\mathbf{X}\mathbf{F}\mathbf{P}\mathbf{X}^{-1} - \mathbf{X}^{-1}\mathbf{P}\mathbf{F}\mathbf{X} = 0 \tag{4.6.20}$$

แล้วถ้าเราทำการคูณสมการด้านบนนี้ด้วย $\mathbf{X}^{-1}$ ทั้งทางด้านซ้ายและด้านขวาของทั้งสองข้างของสมการ

$$\mathbf{X}^{-1}\mathbf{X}\mathbf{F}\mathbf{P}\mathbf{X}^{-1}\mathbf{X}^{-1} - \mathbf{X}^{-1}\mathbf{X}^{-1}\mathbf{P}\mathbf{F}\mathbf{X}\mathbf{X}^{-1} = 0 \tag{4.6.21}$$

เราจะได้ Commutator ที่อยู่ในรูปของ AO Basis ออกมาดังนี้

$$\mathbf{F}'\mathbf{P}' - \mathbf{P}'\mathbf{F}' = \mathbf{FPS} - \mathbf{SPF} = 0 \tag{4.6.22}$$

ซึ่งเราก็จะนำมาใช้ในการกำหนด Error ของวิธี DIIS นั่นเอง ดังนี้

$$\mathbf{e}_i = \mathbf{F}_i\mathbf{P}_i\mathbf{S} - \mathbf{SP}_i\mathbf{F}_i \tag{4.6.23}$$

### 4.6.3 อัลกอริทึม DIIS

อัลกอริทึมของ DIIS ที่ถูกนำมาใช้ในโปรแกรมเคมีควอนตัมทั่วไปนั้นมีดังนี้

1. คำนวณ Error Vector โดยการใช้สมการที่ (4.6.23) ในแต่ละรอบของการคำนวณ
2. สร้าง Matrix $\mathbf{B}$ โดยการใช้ Error Vector $(\mathbf{B}_{ij} = \mathbf{e}_i \cdot \mathbf{e}_j)$ และแก้ชุดสมการเชิงเส้นต่อไปนี้

$$\begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} & \dots & \mathbf{B}_{1m} & -1 \\ \mathbf{B}_{21} & \mathbf{B}_{22} & \dots & \mathbf{B}_{2m} & -1 \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{B}_{m1} & \mathbf{B}_{m2} & \dots & \mathbf{B}_{mm} & -1 \\ -1 & -1 & \dots & -1 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_m \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ -1 \end{pmatrix} \tag{4.6.24}$$

3. คำนวณ Fock Matrix อันใหม่ที่ถูก Extrapolate มาแล้วโดยการใช้สมการ Extrapolation ดังนี้

$$\mathbf{F} = \sum_{i=1}^{m} c_i\mathbf{F}_i \tag{4.6.25}$$

4. ทำการวนซ้ำกระบวนการ SCF จนกว่าคำตอบจะลู่เข้า (Converged)

ครานี้ก็ถึงเวลาสนุกแล้ว เรามาดูรายละเอียดการเขียนโค้ดสำหรับคำนวณพลังงานของโมเลกุลน้ำ $H_2O$ ด้วยวิธี SCF และวิธี DIIS กันครับ โดย Basis Set ที่ใช้คือ STO-3G ซึ่งมีจำนวน Basis Functions ทั้งหมด 7 Functions

```
1 import numpy as np
2
3 from math import sqrt, pi
4 from timeit import default_timer as timer
5
6 start_time = timer()
7
8
```

```
9  def readoneint(filename, naos, t):
10     text = open(filename, "r").read()
11     splitfile = text.strip().split("\n")
12     # print splitfile
13     k = 0
14     for i in range(naos):
15         for j in range(i + 1):
16             # print i, j
17             line = splitfile[k]
18             t[i, j] = line.split()[2]
19             t[j, i] = t[i, j]
20             k = k + 1
21     return
22
23
24 def readtwoint(filename, naos, eri):
25     text = open(filename, "r").read()
26     splitfile = text.strip().split("\n")
27     ntwoints = len(splitfile)
28     for k in range(ntwoints):
29         line = splitfile[k]
30         m = int(line.split()[0]) - 1
31         n = int(line.split()[1]) - 1
32         k = int(line.split()[2]) - 1
33         l = int(line.split()[3]) - 1
34         eri[m, n, k, l] = line.split()[4]
35         eri[n, m, k, l] = eri[m, n, k, l]
36         eri[m, n, l, k] = eri[m, n, k, l]
37         eri[n, m, l, k] = eri[m, n, k, l]
38         eri[k, l, m, n] = eri[m, n, k, l]
39         eri[k, l, n, m] = eri[m, n, k, l]
40         eri[l, k, m, n] = eri[m, n, k, l]
41         eri[l, k, n, m] = eri[m, n, k, l]
42     return
43
44
45 # Read one/two electron intergrals
46 def readint(naos, t, s, v, eri):
47     readoneint("STO-3G/t.dat", naos, t)
48     readoneint("STO-3G/s.dat", naos, s)
49     readoneint("STO-3G/v.dat", naos, v)
50     readtwoint("STO-3G/eri.dat", naos, eri)
51     return
```

```
52
53
54 # Builds new (i)th fock matrix from the (i-1)th density matrix
55 def build_new_fock(denmat, naos, hcore):
56     fmat = np.zeros((naos, naos))
57     for m in range(naos):
58         for n in range(naos):
59             fmat[m, n] = hcore[m, n]
60             for k in range(naos):
61                 for l in range(naos):
62                     fmat[m, n] += denmat[k, l] * (
63                         2 * erimat[m, n, k, l] - erimat[m, k, n,
    l]
64                     )
65     return fmat
66
67
68 # Diagonalises (i)th fock matrix and obtain (i)th density matrix,
69 # electronic energy, etc (check line 92)
70 def diagonalize_fock(fmat, smat_half, hcore):
71     fmatp = np.dot(
72         smat_half, np.dot(fmat, smat_half)
73     )  # transforming to canonical AO basis
74
75     epsilon, cmatp = np.linalg.eig(fmatp)  # diagonalising fmatp
76     idx = epsilon.argsort()[::1]  # sorting the eigenvalues
77     epsilon = epsilon[idx]
78     cmatp = cmatp[:, idx]
79     cmat = np.dot(smat_half, cmatp)
80     # only first 5 MOs are occupied in case of water
81     cmat_occ = cmat[:, 0:5]
82
83     denmat = np.dot(
84         cmat_occ, cmat_occ.transpose()
85     )  # building density matrix and calculating electronic
    energy
86     var1 = hcore + fmat
87     eelec = np.trace(np.dot(denmat, var1))
88
89     return denmat, eelec, fmatp, cmat, epsilon
90
91
92 # builds a new error matrix based on (e = FDS-SDF)
```

```
93  def build_error_vector(fmat, denmat, smat):
94      errvec = (
95          np.dot(fmat, np.dot(denmat, smat)) - np.dot(smat,
        np.dot(denmat, fmat))
96      ).reshape(naos * naos, 1)
97      norm_errvec = sqrt(np.dot(errvec.transpose(), errvec)[0, 0])
98      return errvec, norm_errvec
99
100
101 # builds the B-matrix and solve for DIIS coefficient
102 def get_diis_coeff(errvec_subspace):
103     b_mat = np.zeros((len(errvec_subspace) + 1,
        len(errvec_subspace) + 1))
104     b_mat[-1, :] = -1
105     b_mat[:, -1] = -1
106     b_mat[-1, -1] = 0
107
108     rhs = np.zeros((len(errvec_subspace) + 1, 1))
109     rhs[-1, -1] = -1
110
111     for i in range(len(errvec_subspace)):
112         for j in range(i + 1):
113             b_mat[i, j] = np.dot(errvec_subspace[i].transpose(),
        errvec_subspace[j])
114             b_mat[j, i] = b_mat[i, j]
115
116     *diis_coeff, _ = np.linalg.solve(b_mat, rhs)
117
118     return diis_coeff
119
120
121 # Builds an extrapolated fock matrix by a linear combination
122 # (These won't be a part of fmat_subspace)
123 def extrapolated_fock(fmat_subspace, diis_coeff):
124     extrapolated_fmat = np.zeros((naos, naos))
125
126     for i in range(len(fmat_subspace)):
127         extrapolated_fmat += fmat_subspace[i] * diis_coeff[i]
128
129     return extrapolated_fmat
130
131
132 # Defining a class = Subspace for storing error vectors,
```

```
# fock, and density matrices
class Subspace(list):
    def append(self, item):
        list.append(self, item)
        if len(self) > dimSubspace:
            del self[0]


print("=" * 58)
print(f"* Output for SCF energy calculation using DIIS algorithm
   *")
print("=" * 58, "\n\n")


# Bunch of global variables defined; initalizing some arrays

naos = 7  # STO-3G has 7 AOs for H2O

tmat = np.zeros((naos, naos))
smat = np.zeros((naos, naos))
vmat = np.zeros((naos, naos))
erimat = np.zeros((naos, naos, naos, naos))
denmat = np.zeros((naos, naos))

with open("STO-3G/enuc.dat", "r") as f:
    enuc = float(f.read())

readint(naos, tmat, smat, vmat, erimat)

# This will be initial guess for fock matrix
hcore = tmat + vmat

iterations = 12  # maximum number of iterations

scftol = 1e-12  # energy convergence criterion
dentol = 1e-12  # density convergence criterion
errtol = 1e-15  # error vector should be as close to 0

print(f"Molecule = H2O")
print(f"Basis Set = STO-3G")
print(f"Total number of basis functions = {naos}\n\n")

print(f"CONVERGENCE CRITERIA:\n~~~~~~~~~~~~~~~~~~~~~")
```

```
175 print(f"Energy convergence criterion: {scftol}")
176 print(f"Norm of density matrix convergence criterion: {dentol}")
177 print(f"Norm of DIIS-error vector convergence criterion:
       {errtol}\n\n")
178
179
180 # S^(-1/2) is used to diagonalise Fock matrix
181 s_eigval, s_eigvec = np.linalg.eig(smat)
182 s_halfeig = np.zeros((naos, naos))
183
184 for i in range(naos):
185     s_halfeig[i, i] = s_eigval[i] ** (-0.5)
186
187 a = np.dot(s_eigvec, s_halfeig)  # a = L*s^(-1/2)
188 b = s_eigvec.transpose()
189 smat_half = np.dot(a, b)
190
191
192 ####################
193 # DIIS Code begins #
194 ####################
195
196 # Usually, around 6-8 to give a reasonable result
197 dimSubspace = 6
198
199 # Creating instances of the class=Subspace.
200 # The maximum dimension if defined above.
201 errvec_subspace = Subspace()
202 fmat_subspace = Subspace()
203 denmat_subspace = Subspace()
204
205 # Initilizing these variables to check convergence
206 old_energy, old_denmat = 0, np.zeros((naos, naos))
207
208 # Providing the guess fock matrix to begin with
209 # (This will not be a part of fmat_subspace)
210 fmat = np.zeros((naos, naos))
211 for m in range(naos):
212     for n in range(naos):
213         fmat[m, n] = hcore[m, n]
214
215 print(f"SCF ITERATIONS BEGIN:\n~~~~~~~~~~~~~~~~~~~~~")
216
```

```
217 # Begining the DIIS-SCF iterations
218 for i in range(iterations):
219     print(f"Iteration {i+1}:")
220 
221     if i <= 1:
222         denmat = diagonalize_fock(fmat, smat_half, hcore)[0]
223         denmat_subspace.append(denmat)
224 
225         energy = diagonalize_fock(fmat, smat_half, hcore)[1]
226         print(f"    * Electronic energy = {energy} Eh")
227 
228         fmat = build_new_fock(denmat, naos, hcore)
229         fmat_subspace.append(fmat)
230 
231         errvec, norm_errvec = build_error_vector(fmat, denmat,
    smat)
232         errvec_subspace.append(errvec)
233         print(f"    * Norm of DIIS error vector = {norm_errvec}")
234 
235     else:
236         # Start building and solving B-matrix to obatined DIIS
    coefficient
237         # after we have at least 2 error vectors
238         diis_coeff = get_diis_coeff(errvec_subspace)
239 
240         # Building extrapolated fock matrices
241         extrapolated_fmat = extrapolated_fock(fmat_subspace,
    diis_coeff)
242 
243         # Density matrix from extrapolated fock
244         denmat = diagonalize_fock(extrapolated_fmat, smat_half,
    hcore)[0]
245         denmat_subspace.append(denmat)
246 
247         # Next entry in the fock subspace obtained
248         # from the recent density matrix obtained
249         fmat = build_new_fock(denmat, naos, hcore)
250         fmat_subspace.append(fmat)
251 
252         # Electronic energy
253         energy = diagonalize_fock(fmat, smat_half, hcore)[1]
254         print(f"    * Electronic energy = {energy} Eh")
255 
```

```
256         # Error vector calculation
257         errvec, norm_errvec = build_error_vector(fmat, denmat,
    smat)
258         errvec_subspace.append(errvec)
259         print(f"    * Norm of DIIS error vector = {norm_errvec}")
260         # DIIS algorithm ends here ####
261
262     ### Checking for various convergence criteria
263     deltaE = energy - old_energy
264     print(f"    * Energy convergence = {deltaE} Eh")
265
266     denmat_vector, olddenmat_vector = np.reshape(denmat, (naos *
    naos, 1)), np.reshape(
267         old_denmat, (naos * naos, 1)
268     )
269     norm_denmat = denmat_vector - olddenmat_vector
270     denconv = (np.dot(norm_denmat.transpose(), norm_denmat)) **
    0.5
271     print(f"    * Norm of density matrix convergence =
    {denconv[0,0]}\n")
272
273     # Stop the iterations when convergence is reached
274     if abs(deltaE) < scftol and abs(denconv[0, 0]) < dentol:
275         print(f"SCF ENERGY AND DENSITY CONVERGED WITHIN {i+1}
    ITERATIONS!\n\n")
276         MOenergies = diagonalize_fock(fmat, smat_half, hcore)[4]
277         print(f"ORBITAL
    ENERGIES:\n~~~~~~~~~~~~~~~~~\n{MOenergies}\n\n")
278         print(f"FINAL RESULT:\n~~~~~~~~~~~~~")
279         print(f"    * Electronic energy = {energy} Eh")
280         print(f"    * Total energy = {energy+enuc} Eh")
281         print(f"    * HOMO-LUMO gap =
    {MOenergies[5]-MOenergies[4]} Eh")
282         break
283     else:
284         old_denmat = denmat
285         old_energy = energy
286
287 print(f"\n")
288 end_time = timer()
289 print(f"Wall time = {end_time-start_time} seconds\n")
```

โดยมีเอาต์พุตดังนี้

```
============================================================
* Output for SCF energy calculation using DIIS algorithm *
============================================================


Molecule = H2O
Basis Set = STO-3G
Total number of basis functions = 7


CONVERGENCE CRITERIA:
~~~~~~~~~~~~~~~~~~~~
Energy convergence criterion: 1e-12
Norm of density matrix convergence criterion: 1e-12
Norm of DIIS-error vector convergence criterion: 1e-15


SCF ITERATIONS BEGIN:
~~~~~~~~~~~~~~~~~~~~
Iteration 1:
    * Electronic energy = -125.84207743769872 Eh
    * Norm of DIIS error vector = 0.7071326622354689
    * Energy convergence = -125.84207743769872 Eh
    * Norm of density matrix convergence = 2.5502610775639942

Iteration 2:
    * Electronic energy = -78.28658328473979 Eh
    * Norm of DIIS error vector = 0.2916188629562465
    * Energy convergence = 47.55549415295893 Eh
    * Norm of density matrix convergence = 1.8266730844790497

... [Trimmed]

Iteration 11:
    * Electronic energy = -82.94444699000206 Eh
    * Norm of DIIS error vector = 3.77585115992646e-14
    * Energy convergence = 1.8474111129762605e-13 Eh
    * Norm of density matrix convergence = 6.63375456196504e-14

SCF ENERGY AND DENSITY CONVERGED WITHIN 11 ITERATIONS!


```

```
ORBITAL ENERGIES:
~~~~~~~~~~~~~~~~~
[-20.26289162  -1.20969737  -0.54796465  -0.4365272   -0.38758672
   0.47761872   0.58813928]


FINAL RESULT:
~~~~~~~~~~~~~
    * Electronic energy = -82.94444699000206 Eh
    * Total energy = -74.94207992819162 Eh
    * HOMO-LUMO gap = 0.8652054408643053 Eh


Wall time = 0.06017093900300097 seconds
```

โดยส่วนประกอบของโค้ดด้านบนนี้ประกอบไปด้วยฟังก์ชันและคลาสซึ่งมีหน้าที่ดังนี้

- `readoneint` เป็นฟังก์ชันสำหรับอ่านไฟล์ที่เก็บข้อมูล One-Electron Integral
- `readtwoint` เป็นฟังก์ชันสำหรับอ่านไฟล์ที่เก็บข้อมูล Two-Electron Integral
- `readint` เป็นฟังก์ชันสำหรับอ่านข้อมูล Integral ก็คือเรียกใช้ฟังก์ชัน `readoneint` กับ `readtwoint`
- `build_new_fock` เป็นฟังก์ชันสำหรับสร้าง Fock Matrix อันใหม่
- `diagonalize_fock` เป็นฟังก์ชันสำหรับทำ Diagonalization เพื่อแก้คำตอบหาค่าพลังงานของแต่ละออร์บิทัล
- `build_error_vector` เป็นฟังก์ชันสำหรับคำนวณ Error Vector สำหรับวิธี DIIS
- `get_diis_coeff` เป็นฟังก์ชันสำหรับคำนวณสัมประสิทธิ์เพื่อนำไปคำนวณ Extrapolated Fock
- `extrapolated_fock` เป็นฟังก์ชันสำหรับคำนวณ Extrapolated Fock ด้วยวิธี DIIS
- `Subspace` เป็นคลาสสำหรับสร้าง Subspace เพื่อทำการเก็บข้อมูล Error Vector, Fock Matrix, และ Density Matrix

ข้อมูลของ Electron Integrals ของ STO-3G นั้นผมเอามาจาก GitHub https://github.com/CrawfordGroup/ProgrammingProjects/tree/master/Project%2303/input/h2o/STO-3G ซึ่งดาวน์โหลดมาใช้ได้ฟรี แล้วก็ประสิทธิภาพของ DIIS เมื่อเทียบกับวิธี SCF แบบปกตินั้นถือว่าเยอะกว่ามาก ถ้าหากว่าเราคำนวณ

พลังงานของโมเลกุลน้ำด้วย SCF แบบปกติ จะใช้จำนวนรอบประมาณ 38 รอบ แต่เมื่อใช้ DIIS เข้ามาช่วย พบว่าพลังงานนั้นลู่เข้าภายใน 11 รอบเท่านั้น

## 4.7 เขียนโปรแกรม Møller-Plesset Perturbation (ภาษา Python)

ถ้าหากว่าเราสามารถเขียนโปรแกรมสำหรับคำนวณ Hartree-Fock ได้แล้ว การเขียนโปรแกรมคำนวณ Møller-Plesset นั้นก็ทำได้ไม่ยาก เพราะว่าเรามี Transformed Two-Electron Integrals อยู่แล้ว เราก็สามารถนำมาใช้งานต่อได้เลย โดยเฉพาะการคำนวณ Møller-Plesset แบบ Second Order Perturbation หรือเรียกสั้น ๆ ว่า MP2 ยิ่งง่ายเข้าไปใหญ่ เพราะว่าเราไม่ต้องใช้วิธีการวนซ้ำแบบที่เราทำใน Hartree-Fock โดยพลังงานที่เราจะคำนวณด้วย MP2 นั้นก็คือ Correlation Energy ซึ่งเป็นพลังงานที่เกิดการที่อิเล็กตรอนในโมเลกุลนั้นพยายามผลักหรือหลีกเลี่ยงออกจากกัน (Avoid) โดยพลังงานงานของ MP2 สามารถเขียนออกมาได้ในรูปของ Spin Orbital Basis ดังนี้

$$E_{\mathrm{MP2}} = \frac{1}{4} \sum_{ijab} \frac{|\langle ij||ab\rangle|^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} \tag{4.7.1}$$

โดยที่ $i$ กับ $j$ คือดัชนีที่บอกหมายเลขของออร์บิทัลที่มีอิเล็กตรอนบรรจุอยู่ (Occupied Orbitals) และ $a$ กับ $b$ คือดัชนีที่บอกหมายเลขของออร์บิทัลที่ไม่มีอิเล็กตรอน (Unoccupied Orbitals หรือ Virtual Orbitals) ส่วนเทอม Integrals ที่อยู่ด้านบนของของเศษส่วนนั้นก็คือ Double-Bar Integrals ซึ่งเป็นเทอมที่อธิบายพลังงาน Coulomb กับพลังงาน Exchange ระหว่างอิเล็กตรอนแต่ละคู่ซึ่งได้มาจากการแปลง (Transformation) Two-Electron Integrals ให้กลายเป็น Molecular Orbital Basis ส่วนเทอมที่อยู่ด้านล่างนั้นก็คือพลังงานของออร์บิทัลซึ่งได้มาจาก Eigenvalues ของวิธี Hartree-Fock นั่นเอง พูดง่าย ๆ ก็คือส่วนผสมที่ใช้ในการคำนวณ MP2 นั้นได้มาจาก Hartree-Fock หมดเลย

จากเหตุผลด้านบนนั้นจะสรุปได้ว่าวิธี MP2 นั้นเป็นหนึ่งในวิธีที่คำนวณได้ง่ายที่สุดแล้วของวิธี Electronic Structure ทั้งหมด สำหรับการเขียนโค้ด MP2 นั้นจะเห็นได้ว่าเราจะต้องมีการเขียนลูปทั้งหมด 4 ลูป โดยแต่ละลูปจะวนตามจำนวนของออร์บิทัลที่เราสนใจ

เริ่มต้นด้วยการอิมพอร์ตไลบรารี่ที่เราต้องการใช้ก่อน สำหรับโค้ด MP2 เราจะใช้แค่ `numpy` ครับ

```
1 import numpy as np
```

เรามาเขียนฟังก์ชันสำหรับการคำนวณ Compound Index ของ Integrals แบบ Four-Index กันก่อนครับ

```
1 def eint(a, b, c, d):
2     """
3     Return compound index given four indices
4     """
```

```
    if a > b:
        ab = a * (a + 1) / 2 + b
    else:
        ab = b * (b + 1) / 2 + a
    if c > d:
        cd = c * (c + 1) / 2 + d
    else:
        cd = d * (d + 1) / 2 + c
    if ab > cd:
        abcd = ab * (ab + 1) / 2 + cd
    else:
        abcd = cd * (cd + 1) / 2 + ab
    return abcd
```

ต่อด้วยฟังก์ชันสำหรับคำนวณ Integrals

```
def teimo(a, b, c, d):
    """
    Return Value of spatial MO two electron integral
    Example: (12|34) = tei(1,2,3,4)
    """
    return ttmo.get(eint(a, b, c, d), 0.0e0)
```

ลำดับต่อไปคือการกำหนดค่าเริ่มต้นของพลังงานของออร์บิทัล (Orbital Energies) และ Transformed Two-Electron Integrals

```
Nelec = 2  # 2 electrons in HeH+
dim = 2  # two spatial basis functions in STO-3G
E = [-1.52378656, -0.26763148]
ttmo = {
    5.0: 0.94542695583037617,
    12.0: 0.17535895381500544,
    14.0: 0.12682234020148653,
    17.0: 0.59855327701641903,
    19.0: -0.056821143621433257,
    20.0: 0.74715464784363106,
}
```

แล้วเราก็ทำการแปลง Spatial Molecular Orbital ให้เป็น Spin Molecular Orbital

```
# This makes the spin basis double bar integral (physicists'
    notation).
# We double the dimension (each spatial orbital is now two spin
    orbitals)
```

```
3
4  dim = dim * 2
5  ints = np.zeros((dim, dim, dim, dim))
6  for p in range(1, dim + 1):
7      for q in range(1, dim + 1):
8          for r in range(1, dim + 1):
9              for s in range(1, dim + 1):
10                 value1 = (
11                     teimo((p + 1) // 2, (r + 1) // 2, (q + 1) //
       2, (s + 1) // 2)
12                     * (p % 2 == r % 2)
13                     * (q % 2 == s % 2)
14                 )
15                 value2 = (
16                     teimo((p + 1) // 2, (s + 1) // 2, (q + 1) //
       2, (r + 1) // 2)
17                     * (p % 2 == s % 2)
18                     * (q % 2 == r % 2)
19                 )
20                 ints[p - 1, q - 1, r - 1, s - 1] = value1 -
       value2
```

กำหนด Spin Basis Fock Matrix Eigenvalues

```
1  fs = np.zeros((dim))
2  for i in range(0, dim):
3      fs[i] = E[i // 2]
4  # fs = np.diag(fs)
```

เริ่มการคำนวณ MP2 โดยเราจะทำการวนลูป 2 ลูปแรกก่อนจนครบตามจำนวนของ Occupied Spin Orbitals ซึ่งเท่ากับจำนวนอิเล็กตรอน ส่วน 2 ลูปสุดท้ายข้างในเป็นลูปที่จะวนตามจำนวนของ Virtual Orbitals

```
1  EMP2 = 0.0
2  for i in range(0, Nelec):
3      for j in range(0, Nelec):
4          for a in range(Nelec, dim):
5              for b in range(Nelec, dim):
6                  EMP2 += 0.25 * ints[i, j, a, b] * ints[i, j, a,
       b] / (fs[i] + fs[j] - fs[a] - fs[b])
7
8  print("E(MP2) Correlation Energy = ", EMP2, " Hartrees")
```

เมื่อรันโค้ดด้านบนแล้วจะได้ Correlation Energy ออกมาประมาณ -0.00640 Hartree ซึ่งน้อยมากเมื่อเทียบกับ Electronic Energy ของโมเลกุลเดียวกัน ซึ่งถือว่าเป็นเรื่องปกติที่ Correlation Energy นั้นจะมีสัดส่วน

ที่น้อยกว่าพลังงานอื่น ๆ จากงานวิจัยหลาย ๆ งานพบว่า Hartree-Fock นั้นให้ความถูกต้องในการคำนวณพลังงานของโมเลกุลมากถึง 99% ส่วนอีก 1% นั้นก็คือพลังงาน Correlation ที่หายไปนั่นเอง แต่ถ้าหากว่าเรามาดูรายละเอียดจริง ๆ จะพบว่า 1% ที่หายไปนั้นไม่น้อยเลย เพราะถ้าหากว่าโมเลกุลของเรามีพลังงานรวมทั้งหมดอยู่ในหลักสิบหรือหลักร้อย kcal/mol นั่นหมายความว่าความคลาดเคลื่อนของการที่เราไม่พิจารณา Correlation Energy นั้นมีค่าประมาณ 1% ซึ่งเกินค่าความถูกต้องทางเคมี (Chemical Accuracy) ไปเยอะมาก ดังนั้นการรวม Correlation Effect จึงเป็นสิ่งที่สำคัญมากและถือว่าเป็นหนึ่งในหัวข้องานวิจัยที่สำคัญมาก ๆ ของเคมีควอนตัม ในปัจจุบันมีวิธีการต่าง ๆ ที่ถูกพัฒนาขึ้นมาเพื่อใช้ในการคำนวณพลังงานรวมของโมเลกุลโดยรวมผลของ Correlation Effect เข้าไปด้วยเพื่อให้ผลการคำนวณนั้นมีค่าถูกต้องและแม่นยำมากที่สุดเท่าที่จะเป็นไปได้

## 4.8 เขียนโปรแกรม Coupled Cluster (ภาษา Python)

### 4.8.1 มาทำความเข้าใจทฤษฎีกันก่อน

ในหัวข้อนี้เราจะมาดูวิธีการเขียนโค้ด Implement วิธี Coupled Cluster with Singles and Doubles (CCSD) กันครับ โดยเราจะอ้างอิง Formalism จากบทความวิจัยของ John F. Stanton ตีพิมพ์เมื่อปี ค.ศ. 1991[28] โดยเหตุผลที่เลือก Formalism นี้ก็เพราะว่าสามารถ Implement ได้ง่ายนั่นเอง ถ้าพร้อมแล้วก็มาเริ่มกันเลย

สำหรับโมเดล Coupled Cluster (CC) นั้น ถ้าเราสามารถทำการแยกตัวประกอบของเทอม Excitations ต่าง ๆ ได้ เราพบว่าเทอมทุกเทอมที่แยกออกมานั้นจะสามารถหาได้จากการทำให้สั้นลง (Contraction) ของ Cluster Operator ($T$) โดยการใช้เพียงแค่ Integrals แบบ Two-Index หรือ Four-Index เท่านั้น คำถามคือ “แล้ว Integrals อันไหนล่ะที่เราจะสามารถนำมาใช้ได้ แล้ว Integrals อันนั้นจะต้องเป็นแบบ Two-Index หรือ Four-Index ด้วย” คำตอบก็คือ “ขึ้นอยู่กับกรณี”

- ในกรณีของ CC Model ที่เป็นแบบเส้นตรง (Linear) เราจะใช้ Fock Matrix Elements ($f_{pq}$) และ Molecular Orbital (MO) Integrals $\langle pq||rs\rangle$
- ในกรณีของ CC แบบที่ไม่เป็นเส้นตรง (Nonlinear) เช่น CCD หรือ CCSD นั้นเราจะใช้ไม่สามารถใช้ ($f_{pq}$) กับ $\langle pq||rs\rangle$ ได้ตรง ๆ แต่เราจะใช้สิ่งที่เรียกว่า Two-Particle Intermediates แทน ซึ่งมันก็คือการนำ ($f_{pq}$) มา $\langle pq||rs\rangle$ มาดัดแปลง โดย Two-Particle Intermediates ที่ได้จากการการดัดแปลงนั้นคือ $F_{pq}$ และ $W_{pqrs}$ ตามลำดับ

โดยเรากำหนดให้ $i, j, k, \ldots$ แทน Occupied Orbitals, $a, b, c, \ldots$ แทน Unoccupied Orbitals, ส่วน $p, q, r, \ldots$ เป็นดัชนีแบบทั่วไปที่เอาไว้ใช้แทน Occupied หรือ Unoccupied Orbitals ก็ได้

จากเปเปอร์ของ Stanton ตัวโมเดล CCSD นั้นสามารถเขียนให้อยู่ในรูปของการแยกเทอมตาม Spin-

Orbital Form ได้ดังนี้

**Singles Cluster Operator:** $T_1$

$$\begin{aligned} t_i^a D_i^a =& f_{ia} + \sum_e t_i^e F_{ae} - \sum_m t_m^a F_{mi} + \sum_{me} t_{im}^{ae} F_{me} \\ & - \sum_{nf} t_n^f \langle na \| if \rangle - \frac{1}{2} \sum_{mef} t_{im}^{ef} \langle ma \| ef \rangle \\ & - \frac{1}{2} \sum_{mcn} t_{mn}^{ae} \langle nm \| ei \rangle \end{aligned} \tag{4.8.1}$$

**Doubles Cluster Operator:** $T_2$

$$\begin{aligned} t_{ij}^{ab} D_{ij}^{ab} =& \langle ij \| ab \rangle + P_-(ab) \sum_e t_{ij}^{ae} \left( F_{be} - \frac{1}{2} \sum_m t_m^b F_{me} \right) \\ & - P_-(ij) \sum_m t_{im}^{ab} \left( F_{mj} + \frac{1}{2} \sum_e t_j^e F_{me} \right) \\ & + \frac{1}{2} \sum_{mn} \tau_{mn}^{ab} W_{mnij} + \frac{1}{2} \sum_{ef} \tau_{ij}^{ef} W_{abef} \\ & + P_-(ij) P_-(ab) \sum_{me} (t_{im}^{ac} F_{mbej} - t_i^e t_m^a \langle mb \| ej \rangle) \\ & + P_-(ij) \sum_c t_i^e \langle ab \| ej \rangle - P_-(ab) \sum_m t_m^a \langle mb \| ij \rangle \end{aligned} \tag{4.8.2}$$

โดยจะเห็นว่าทั้ง $T_1$ และ $T_2$ นั้นต่างก็มี Two-Particle Intermediates ($F$ และ $W$) เป็นส่วนหนึ่งของสมการ ซึ่งเรามีนิยามของ Intermediates ทั้ง 2 ตัวสำหรับ Contribution จาก Occupied และ Unoccupied Orbitals ที่แตกต่างกันดังนี้

$$F_{ae} = (1 - \delta_{ae}) f_{ae} - \frac{1}{2} \sum_m f_{me} t_m^a + \sum_{mf} t_m^f \langle ma \| fe \rangle - \frac{1}{2} \sum_{mnf} \tilde{\tau}_{mn}^{af} \langle mn \| ef \rangle \tag{4.8.3}$$

$$F_{mi} = (1 - \delta_{mi}) f_{mi} + \frac{1}{2} \sum_e t_i^e f_{me} + \sum_{en} t_n^e \langle mn \| ie \rangle + \frac{1}{2} \sum_{nef} \tilde{\tau}_{in}^{ef} \langle mn \| ef \rangle \tag{4.8.4}$$

$$F_{me} = f_{me} + \sum_{nf} t_n^f \langle mn \| ef \rangle \tag{4.8.5}$$

$$W_{mnij} = \langle mn||ij\rangle + P_{-}(ij)\sum_{e} t_j^e \langle mn||ie\rangle + \frac{1}{4}\sum_{ef} \tau_{ij}^{ef}\langle mn||ef\rangle \tag{4.8.6}$$

$$W_{abef} = \langle ab||ef\rangle - P - (ab)\sum_{m} t_m^b \langle am||ef\rangle \tag{4.8.7}$$

$$\begin{aligned} W_{mbej} =& \langle mb||ej\rangle + \sum_{f} t_j^f \langle mb||ef\rangle + \frac{1}{4}\sum_{mn} \tau_{mn}^{ab}\langle mn||ef\rangle \\ &- \sum_{n} t_n^b \langle mn||ej\rangle - \sum_{nf}\left(\frac{1}{2}t_{jn}^{fb} + t_j^f t_n^b\right)\langle mn||ef\rangle \end{aligned} \tag{4.8.8}$$

โดย Intermediate Operators ทั้ง 2 ตัวนี้ก็ขึ้นกับ Two-Particle Excitation Operators อีกเช่นกัน นั่นก็คือ $\tau$ และ $\tilde{\tau}$ ซึ่งมีนิยามดังต่อไปนี้

$$\tilde{\tau}_{ij}^{ab} = t_{ij}^{ab} + \frac{1}{2}\left(t_i^a t_j^b - t_i^b t_j^a\right) \tag{4.8.9}$$

$$\tau_{ij}^{ab} = t_{ij}^{ab} + t_i^a t_j^b - t_i^b t_j^a \tag{4.8.10}$$

แล้วก็ยังขึ้นอยู่กับ $P_{\pm}(pq)$ Operator ด้วย ซึ่งมีนิยามดังนี้

$$P_{\pm}(pq) = 1 \pm P(pq) \tag{4.8.11}$$

โดยที่ $P(pq)$ นั้นมีคุณสมบัติ Permutation ระหว่างดัชนี $p$ และ $q$, $\delta_{pq}$ คือ Kronecker Delta, และเราก็มี Denominator Arrays $(D)$ ซึ่งมีนิยามคือ

$$D_i^a = f_{ii} - f_{aa} \tag{4.8.12}$$

$$D_{ij}^{ab} = f_{ii} + f_{ij} - f_{aa} - f_{bb} \tag{4.8.13}$$

ก่อนที่เราจะไปดูรายละเอียดการเขียนโค้ดของ CCSD นั้น เรามาวิเคราะห์สมการด้านบนกันก่อนว่าเราควรจะดีไซน์โปรแกรมของเราอย่างไรดี ถ้าหากว่าเราไม่เข้าใจ Workflow ของอัลกอริทึมที่เราต้องการ Implement ให้ละเอียด พอถึงเวลาที่เราต้องเขียนโค้ดจริง ๆ ก็อาจจะสับสนหรืองงได้ และนำไปสู่ผลการคำนวณที่ผิด

จะเห็นได้ว่าจากสมการที่ (4.8.1) - (4.8.8) นั้นจะเป็นการคำนวณที่มีแค่การคูณระหว่างเมทริกซ์ (Matrix-Matrix Product) เท่านั้น ดังนั้นเราจึงควรเลือกใช้ Array สำหรับเป็น Data Structure ของตัวแปรต่าง ๆ ในโค้ด นอกจากนี้แล้วถ้าสังเกตดี ๆ จะเห็นว่าเราเริ่มต้นด้วยสมการของ Singles กับ Doubles Operators ก่อน (สมการที่ (4.8.1) และ (4.8.2)) แล้วก็มีสมการอื่น ๆ ตามมาที่เราจะต้องคำนวณเพราะว่ามันเป็นส่วนประกอบหรือ Component ที่เราจะต้องนำมาใช้ในการคำนวณ Cluster Operator ดังนั้นวิธีการ Implement วิธี CCSD ตาม Formalism ของ Stanton นั้นก็คือใช้เทคนิค Bottom-Up ซึ่งก็คือการเริ่มเขียนฟังก์ชันเพื่อคำนวณหาองค์ประกอบต่าง ๆ ที่เราจำเป็นต้องใช้ก่อน แล้วนำมารวมร่างกันเพื่อนำไปสู่การหาคำตอบสุดท้าย ซึ่งก็คือ $T_1$ และ $T_2$ นั่นเอง แล้วพอเราได้ Cluster Operators แล้วเราก็สามารถนำไปใช้ในการคำนวณหาพลังงาน Correlation ได้ต่อไป (มือธิบายแทรกอยู่ในส่วนที่เขียนโค้ด)

ต้องขอโน๊ตไว้ว่าสำหรับ Fock Matrix Elements นั้น เราจะนำ Hartree-Fock มาใช้เป็นฟังก์ชันอ้างอิง ดังนั้น Diagonal Elements ของ Diagonal Terms นั้นจึงเป็นแค่พลังงานของออร์บิทัลหรือ Eigenvalues

### 4.8.2 มาลงมือเขียนโค้ด CCSD กันเลย

เราเริ่มด้วยการอิมพอร์ตไลบรารี่ที่ต้องใช้

```
1 import numpy as np
```

เขียนฟังก์ชันสำหรับการหา Compound Index (เหมือนกันกับโค้ดของ MP2 ก่อนหน้านี้)

```
1  def eint(a, b, c, d):
2    """
3    Return compound index given four indices
4    """
5    if a > b:
6        ab = a * (a + 1) / 2 + b
7    else:
8        ab = b * (b + 1) / 2 + a
9    if c > d:
10       cd = c * (c + 1) / 2 + d
11   else:
12       cd = d * (d + 1) / 2 + c
13   if ab > cd:
14       abcd = ab * (ab + 1) / 2 + cd
15   else:
16       abcd = cd * (cd + 1) / 2 + ab
17   return abcd
```

เขียนฟังก์ชันสำหรับการคำนวณ Two-Electron Integral ของ Spatial MOs (เหมือนกันกับโค้ดของ MP2 ก่อนหน้านี้)

```
def teimo(a, b, c, d):
  """
  Return Value of spatial MO two electron integral
  Example: (12|34) = tei(1,2,3,4)
  """
  return ttmo.get(eint(a, b, c, d), 0.0e0)
```

กำหนดพารามิเตอร์ต่าง ๆ (คำอธิบายอยู่ในคอมเมนต์ภายในโค้ด)

```
Nelec = 2  # we have 2 electrons in HeH+
dim = 2  # we have two spatial basis functions in STO-3G
E = [-1.52378656, -0.26763148]  # molecular orbital energies
# python dictionary containing Two-Electron repulsion integrals
ttmo = {
    5.0: 0.94542695583037617,
    12.0: 0.17535895381500544,
    14.0: 0.12682234020148653,
    17.0: 0.59855327701641903,
    19.0: -0.056821143621433257,
    20.0: 0.74715464784363106,
}
ENUC = 1.1386276671  # nuclear repulsion energy for HeH+
EN = -3.99300007772  # SCF energy
```

เมื่อเราได้ Spatial MOs แล้ว ลำดับต่อไปคือการเขียนฟังก์ชันสำหรับแปลง Spatial MOs ให้เป็น Spin MOs

```
spinints = np.zeros((dim * 2, dim * 2, dim * 2, dim * 2))

for p in range(1, dim * 2 + 1):
    for q in range(1, dim * 2 + 1):
        for r in range(1, dim * 2 + 1):
            for s in range(1, dim * 2 + 1):
                value1 = (
                    teimo((p + 1) // 2, (r + 1) // 2, (q + 1) //
    2, (s + 1) // 2)
                    * (p % 2 == r % 2)
                    * (q % 2 == s % 2)
                )
                value2 = (
                    teimo((p + 1) // 2, (s + 1) // 2, (q + 1) //
    2, (r + 1) // 2)
```

```
                    * (p % 2 == s % 2)
                    * (q % 2 == r % 2)
                )
                spinints[p - 1, q - 1, r - 1, s - 1] = value1 -
    value2
```

ทำการคำนวณพลังงานของ MOs แล้วใส่ค่าที่คำนวณได้กลับคืนเข้าไปใน Diagonal Array

```
fs = np.zeros((dim * 2))
for i in range(0, dim * 2):
    fs[i] = E[i // 2]
# put MO energies in diagonal array
fs = np.diag(fs)
```

ขั้นตอนต่อจากนี้ไปเราจะมาเขียนฟังก์ชันสำหรับการคำนวณ Operator และ Integrals ต่าง ๆ ที่ได้อธิบายไปก่อนหน้านี้ตามสมการในเปเปอร์ของ Stanton โดยเราจะเริ่มด้วยการสร้าง Array เปล่า ๆ สำหรับเก็บข้อมูลของ Singles $T_1$ และ Doubles $T_2$ Operators ก่อน แล้วก็ตามด้วยการกำหนดค่าเริ่มต้นให้กับ Doubles Operator $T_2$ ซึ่งเราจะใช้ค่า Integrals และพลังงานจาก MP2 ที่เราได้เขียนโค้ดไปก่อนหน้านี้

แล้วก็ตามด้วยการคำนวณ Denominator Arrays ($D_i^a$ กับ $D_{ij}^{ab}$) ตามสมการที่ (4.8.12) และ (4.8.13) ตามลำดับ แล้วก็ตามด้วยการคำนวณ Two-Particle Excitation Operators ($\tilde{\tau}_{ij}^{ab}$ กับ $\tau_{ij}^{ab}$) ตามสมการที่ (4.8.9) และ (4.8.10) ตามลำดับ

```
# twice the dimension of spatial orbital
dim = dim * 2

# Initialize empty T1 (ts) and T2 (td) arrays
ts = np.zeros((dim, dim))
td = np.zeros((dim, dim, dim, dim))

# Initial guess T2 --- from MP2 calculation!
for a in range(Nelec, dim):
    for b in range(Nelec, dim):
        for i in range(0, Nelec):
            for j in range(0, Nelec):
                td[a, b, i, j] += spinints[i, j, a, b] / (fs[i,
    i] + fs[j, j] - fs[a, a] - fs[b, b])

# Make denominator arrays Dai, Dabij
# Stanton eq (12)
Dai = np.zeros((dim, dim))
for a in range(Nelec, dim):
    for i in range(0, Nelec):
```

```
        Dai[a, i] = fs[i, i] - fs[a, a]

# Stanton eq (13)
Dabij = np.zeros((dim, dim, dim, dim))
for a in range(Nelec, dim):
    for b in range(Nelec, dim):
        for i in range(0, Nelec):
            for j in range(0, Nelec):
                Dabij[a, b, i, j] = fs[i, i] + fs[j, j] - fs[a,
    a] - fs[b, b]


# Stanton eq (9)
def taus(a, b, i, j):
    taus = td[a, b, i, j] + 0.5 * (ts[a, i] * ts[b, j] - ts[b,
    i] * ts[a, j])
    return taus


# Stanton eq (10)
def tau(a, b, i, j):
    tau = td[a, b, i, j] + ts[a, i] * ts[b, j] - ts[b, i] *
    ts[a, j]
    return tau
```

เรามาต่อด้วยการเขียนฟังก์ชันสำหรับคำนวณ Two-Particle Intermediates ซึ่งเป็นการ Implement $F_{pq}$ กับ $W_{pqrs}$ สมการที่ (4.8.3), (4.8.4), (4.8.5), (4.8.6), (4.8.7), และ (4.8.8) ตามลำดับ

```
def update_inter(x):
    """
    Update two-particle intermediates
    """
    if x == True:
        # Stanton eq (3)
        Fae = np.zeros((dim, dim))
        for a in range(Nelec, dim):
            for e in range(Nelec, dim):
                Fae[a, e] = (1 - (a == e)) * fs[a, e]
                for m in range(0, Nelec):
                    Fae[a, e] += -0.5 * fs[m, e] * ts[a, m]
                    for f in range(Nelec, dim):
                        Fae[a, e] += ts[f, m] * spinints[m, a,
    f, e]
```

```
                        for n in range(0, Nelec):
                            Fae[a, e] += -0.5 * taus(a, f, m, n)
    * spinints[m, n, e, f]

        # Stanton eq (4)
        Fmi = np.zeros((dim, dim))
        for m in range(0, Nelec):
            for i in range(0, Nelec):
                Fmi[m, i] = (1 - (m == i)) * fs[m, i]
                for e in range(Nelec, dim):
                    Fmi[m, i] += 0.5 * ts[e, i] * fs[m, e]
                    for n in range(0, Nelec):
                        Fmi[m, i] += ts[e, n] * spinints[m, n,
    i, e]
                        for f in range(Nelec, dim):
                            Fmi[m, i] += 0.5 * taus(e, f, i, n)
    * spinints[m, n, e, f]

        # Stanton eq (5)
        Fme = np.zeros((dim, dim))
        for m in range(0, Nelec):
            for e in range(Nelec, dim):
                Fme[m, e] = fs[m, e]
                for n in range(0, Nelec):
                    for f in range(Nelec, dim):
                        Fme[m, e] += ts[f, n] * spinints[m, n,
    e, f]

        # Stanton eq (6)
        Wmnij = np.zeros((dim, dim, dim, dim))
        for m in range(0, Nelec):
            for n in range(0, Nelec):
                for i in range(0, Nelec):
                    for j in range(0, Nelec):
                        Wmnij[m, n, i, j] = spinints[m, n, i, j]
                        for e in range(Nelec, dim):
                            Wmnij[m, n, i, j] += (
                                ts[e, j] * spinints[m, n, i, e]
    - ts[e, i] * spinints[m, n, j, e]
                            )
                            for f in range(Nelec, dim):
                                Wmnij[m, n, i, j] += 0.25 *
    tau(e, f, i, j) * spinints[m, n, e, f]
```

```
52
53          # Stanton eq (7)
54          Wabef = np.zeros((dim, dim, dim, dim))
55          for a in range(Nelec, dim):
56              for b in range(Nelec, dim):
57                  for e in range(Nelec, dim):
58                      for f in range(Nelec, dim):
59                          Wabef[a, b, e, f] = spinints[a, b, e, f]
60                          for m in range(0, Nelec):
61                              Wabef[a, b, e, f] += (
62                                  -ts[b, m] * spinints[a, m, e, f]
    + ts[a, m] * spinints[b, m, e, f]
63                              )
64                              for n in range(0, Nelec):
65                                  Wabef[a, b, e, f] += 0.25 *
    tau(a, b, m, n) * spinints[m, n, e, f]
66
67          # Stanton eq (8)
68          Wmbej = np.zeros((dim, dim, dim, dim))
69          for m in range(0, Nelec):
70              for b in range(Nelec, dim):
71                  for e in range(Nelec, dim):
72                      for j in range(0, Nelec):
73                          Wmbej[m, b, e, j] = spinints[m, b, e, j]
74                          for f in range(Nelec, dim):
75                              Wmbej[m, b, e, j] += ts[f, j] *
    spinints[m, b, e, f]
76                          for n in range(0, Nelec):
77                              Wmbej[m, b, e, j] += -ts[b, n] *
    spinints[m, n, e, j]
78                              for f in range(Nelec, dim):
79                                  Wmbej[m, b, e, j] += (
80                                      -(0.5 * td[f, b, j, n] +
    ts[f, j] * ts[b, n]) * spinints[m, n, e, f]
81                                  )
82
83          return Fae, Fmi, Fme, Wmnij, Wabef, Wmbej
```

```
1  # Stanton eq (1)
2  def makeT1(x, ts, td):
3      if x == True:
4          tsnew = np.zeros((dim, dim))
```

```
        for a in range(Nelec, dim):
            for i in range(0, Nelec):
                tsnew[a, i] = fs[i, a]
                for e in range(Nelec, dim):
                    tsnew[a, i] += ts[e, i] * Fae[a, e]
                for m in range(0, Nelec):
                    tsnew[a, i] += -ts[a, m] * Fmi[m, i]
                    for e in range(Nelec, dim):
                        tsnew[a, i] += td[a, e, i, m] * Fme[m, e]
                        for f in range(Nelec, dim):
                            tsnew[a, i] += -0.5 * td[e, f, i, m]
    * spinints[m, a, e, f]
                        for n in range(0, Nelec):
                            tsnew[a, i] += -0.5 * td[a, e, m, n]
    * spinints[n, m, e, i]
                for n in range(0, Nelec):
                    for f in range(Nelec, dim):
                        tsnew[a, i] += -ts[f, n] * spinints[n,
    a, i, f]
                tsnew[a, i] = tsnew[a, i] / Dai[a, i]
    return tsnew


# Stanton eq (2)
def makeT2(x, ts, td):
    if x == True:
        tdnew = np.zeros((dim, dim, dim, dim))
        for a in range(Nelec, dim):
            for b in range(Nelec, dim):
                for i in range(0, Nelec):
                    for j in range(0, Nelec):
                        tdnew[a, b, i, j] += spinints[i, j, a, b]
                        for e in range(Nelec, dim):
                            tdnew[a, b, i, j] += td[a, e, i, j]
    * Fae[b, e] - td[b, e, i, j] * Fae[a, e]
                            for m in range(0, Nelec):
                                tdnew[a, b, i, j] += (
                                    -0.5 * td[a, e, i, j] *
    ts[b, m] * Fme[m, e]
                                    + 0.5 * td[b, e, i, j] *
    ts[a, m] * Fme[m, e]
                                )
                                continue
```

```
                        for m in range(0, Nelec):
                            tdnew[a, b, i, j] += -td[a, b, i, m]
    * Fmi[m, j] + td[a, b, j, m] * Fmi[m, i]
                            for e in range(Nelec, dim):
                                tdnew[a, b, i, j] += (
                                    -0.5 * td[a, b, i, m] *
    ts[e, j] * Fme[m, e]
                                    + 0.5 * td[a, b, j, m] *
    ts[e, i] * Fme[m, e]
                                )
                                continue
                        for e in range(Nelec, dim):
                            tdnew[a, b, i, j] += (
                                ts[e, i] * spinints[a, b, e, j]
    - ts[e, j] * spinints[a, b, e, i]
                            )
                            for f in range(Nelec, dim):
                                tdnew[a, b, i, j] += 0.5 *
    tau(e, f, i, j) * Wabef[a, b, e, f]
                                continue
                        for m in range(0, Nelec):
                            tdnew[a, b, i, j] += (
                                -ts[a, m] * spinints[m, b, i, j]
    + ts[b, m] * spinints[m, a, i, j]
                            )
                            for e in range(Nelec, dim):
                                tdnew[a, b, i, j] += (
                                    td[a, e, i, m] * Wmbej[m, b,
    e, j]
                                    - ts[e, i] * ts[a, m] *
    spinints[m, b, e, j]
                                )
                                tdnew[a, b, i, j] += (
                                    -td[a, e, j, m] * Wmbej[m,
    b, e, i]
                                    + ts[e, j] * ts[a, m] *
    spinints[m, b, e, i]
                                )
                                tdnew[a, b, i, j] += (
                                    -td[b, e, i, m] * Wmbej[m,
    a, e, j]
                                    + ts[e, i] * ts[b, m] *
    spinints[m, a, e, j]
```

```
                                    )
                                    tdnew[a, b, i, j] += (
                                        td[b, e, j, m] * Wmbej[m, a,
    e, i]
                                        - ts[e, j] * ts[b, m] *
    spinints[m, a, e, i]
                                    )
                                    continue
                                for n in range(0, Nelec):
                                    tdnew[a, b, i, j] += 0.5 *
    tau(a, b, m, n) * Wmnij[m, n, i, j]
                                    continue
                            tdnew[a, b, i, j] = tdnew[a, b, i, j] /
    Dabij[a, b, i, j]
        return tdnew
```

แล้วก็มาถึงการเขียนฟังก์ชันอันสุดท้ายซึ่งก็คือฟังก์ชันที่เราจะใช้ในการคำนวณพลังงานของโมเลกุลด้วยวิธี CCSD โดยการใช้ $T_1$ กับ $T_2$ Operators ที่เราได้เขียนฟังก์ชันคำนวณไว้แล้วตามด้านบน ซึ่งเราจะอ้างอิงตามสมการคำนวณพลังงานใน Chapter ของหนังสือ Reviews in Computational Chemistry, Volume 14[29] ซึ่งเป็นสมการที่ (134) และ (173) ตามลำดับ ดังนี้

$$E_{\mathrm{CCSD}} - E_0 = \sum_{ia} f_{ia} t_i^a + \frac{1}{4} \sum_{ijab} \langle ij \| ab \rangle t_{ij}^{ab} + \frac{1}{2} \sum_{ijab} \langle ij \| ab \rangle t_i^a t_j^b \tag{4.8.14}$$

โดยสมการด้านบนนี้ไม่มีการรวม Contribution จาก Excitation Cluster Operators เทอมสูง ๆ เช่น $T_3$ หรือ $T_4$ เพราะว่า Operators เทอมสูง ๆ เหล่านั้นไม่สามารถนำมาใช้ในการสร้าง Contracted Term กับ Hamiltonian ได้

```
def ccsdenergy():
    """
    Expression from Crawford, Schaefer (2000)
    DOI: 10.1002/9780470125915.ch2
    Equation (134) and (173)
    Computes CCSD energy given T1 and T2
    """
    ECCSD = 0.0
    for i in range(0, Nelec):
        for a in range(Nelec, dim):
            ECCSD += fs[i, a] * ts[a, i]
            for j in range(0, Nelec):
                for b in range(Nelec, dim):
```

```
                    ECCSD += 0.25 * spinints[i, j, a, b] * td[a,
    b, i, j] + 0.5 * spinints[i, j, a, b] * (
                        ts[a, i]
                    ) * (ts[b, j])
    return ECCSD
```

ขั้นตอนสุดท้ายก็คือการรันการคำนวณ CCSD โดยการเรียกใช้ฟังก์ชันหลักด้านบน ดังนี้

```
# MAIN LOOP: CCSD iteration
ECCSD = 0
DECC = 1.0
while DECC > 0.000000001:  # arbitrary convergence criteria
    OLDCC = ECCSD
    Fae, Fmi, Fme, Wmnij, Wabef, Wmbej = update_inter(True)
    tsnew = makeT1(True, ts, td)
    tdnew = makeT2(True, ts, td)
    ts = tsnew
    td = tdnew
    ECCSD = ccsdenergy()
    DECC = abs(ECCSD - OLDCC)
```

แล้วก็ทำการแสดงค่าพลังงาน Correlation กับพลังงานรวมของ CCSD ที่คำนวณได้

```
print("E(corr,CCSD) = ", ECCSD)
print("E(CCSD) = ", ECCSD + ENUC + EN)

# Output
# E(corr,CCSD) =  -0.008225834879316319
# E(CCSD) =  -2.862598245499316
```

ซึ่งจะได้ผลการคำนวณประมาณ -0.0082 กับ -2.8626 Hartree ตามลำดับ

## 4.9 เขียนโปรแกรม Kohn-Sham DFT (ภาษา Python)

### 4.9.1 มาทำความเข้าใจทฤษฎีกันก่อน

Concept ของการคำนวณ Kohn-Sham DFT ก็คือการกำหนดให้อิเล็กตรอนในระบบนั้นไม่มีอันตรกิริยาต่อกัน ซึ่งผู้อ่านสามารถศึกษาทฤษฎีแบบละเอียดได้ในหนังสือ “ปัญญาประดิษฐ์สำหรับเคมีควอนตัม

(Machine Learning for Quantum Chemistry)"[1] สำหรับหัวข้อนี้ เพื่อให้ผู้อ่านเห็นภาพมากขึ้น ผมอยากให้ผู้อ่านจะได้ศึกษาการเขียนโปรแกรมสำหรับการคำนวณพลังงานของระบบหลายอิเล็กตรอนโดยใช้ Kohn-Sham DFT โดยเราจะสนใจกรณีที่เป็น 1 มิติเท่านั้น (อิเล็กตรอนมีการเคลื่อนที่ตามแกน $x$ เพียงอย่างเดียว) เพื่อให้ง่ายต่อการศึกษา

ในการเขียนโค้ดของ Kohn-Sham (KS) DFT นั้นเราจะใช้ Hamiltonian ตามสมการดังต่อไปนี้

$$E_{\text{KS}}[n] = E_{\text{kin,KS}}[n] + E_{\text{Coul}}[n] + E_{\text{Ext}}[n] + \underbrace{E_{\text{XC}}[n] + (E_{\text{kin}}[n] - E_{\text{kin,KS}}[n])}_{\text{นำมารวมกันได้}} \tag{4.9.1}$$

$$= 2\sum_{i=1}^{N_{\text{el}}/2} \int \psi_i^*(\boldsymbol{r}) \left(-\frac{1}{2}\nabla^2\right) + E_{\text{Coul}}[n] + E_{\text{Ext}}[n] + E'_{\text{XC}}[n] \tag{4.9.2}$$

โดยเราสามารถเขียนให้สั้นและกระชับมากขึ้นได้ ดังนี้

$$\hat{H} = -\frac{1}{2}\frac{d^2}{dx^2} + v_{Coul}(x) + v_{LDA}(x) + v_{ext} \tag{4.9.3}$$

โดยทางด้านขวาของสมการ (4.9.3) ประกอบไปด้วยเทอมดังต่อไปนี้

1. พลังงานจลน์ (Kinetic Energy)
2. พลังงานศักย์คูลอมป์ (Coulomb Energy) หรือแรงผลักไฟฟ้าสถิตย์ระหว่างอิเล็กตรอน
3. พลังงานแลกเปลี่ยน (Exchange Energy) ซึ่งเราจะใช้การประมาณค่าความหนาแน่นแบบพื้นที่ (Local Density Approximation)
4. พลังงานภายนอก (External Potential) ซึ่งเราจะใช้ฟังก์ชัน Harmonic Oscillator

หมายเหตุ: เราจะไม่พิจารณา Correlation Energy เนื่องจากว่ามีความซับซ้อนมากเกิน

โดยเราจะใช้ภาษา Python ในการเขียน โดยสิ่งที่เราต้องทำหลัก ๆ มีดังนี้

1. สร้าง Hamiltonian
2. คำนวณฟังก์ชันคลื่นของ Kohn-Sham (KS Wavefunction)
3. คำนวณความหนาแน่น (Density)
4. คำนวณพลังงานอิเล็กทรอนิกส์ (Electronic Energy)

## 4.9.2 มาลงมือเขียนโค้ด DFT กันเลย

### 1. นำเข้าไลบรารี่และสร้างฟังก์ชันที่จำเป็นต้องใช้

[1] ลิงก์หนังสือ: https://rangsimanketkaew.github.io/ml-qm-book

ใช้ไลบรารี่ NumPy สำหรับจัดการกับเมทริกซ์และไลบรารี่ Matplotlib กับ Seaborn สำหรับพลอต

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

ทำการสร้างฟังก์ชันสำหรับการ Integrate ซึ่งก็คือการรวมกันนั่นเอง โดยเราจะนำฟังก์ชันนี้ไปใช้งานต่อในโค้ดด้านล่าง

```
def integral(x, y, axis=0):
    dx = x[1]-x[0]
    return np.sum(y*dx, axis=axis)
```

**2. กำหนดโอเปอเรเตอร์เชิงอนุพันธ์สำหรับการสร้าง Hamiltonian ของพลังงานจลน์**

```
# Define a real-space grid
n_grid = 200
x = np.linspace(-5, 5, n_grid)
y = np.sin(x)

# First derivative
h = x[1]-x[0]
D = -np.eye(n_grid) + np.diagflat(np.ones(n_grid-1),1)
D = D / h

# Second derivative
D2 = D.dot(-D.T)
D2[-1,-1] = D2[0,0]
```

**3. คำนวณพลังงานจลน์**

แก้สมการ Kohn-Sham เฉพาะของพลังงานจลน์โดยการทำ Diagonalization (เป็นขั้นตอนที่กำหนด Computational Complexity ของ DFT นั่นคือ $\mathcal{O}(n^3)$

```
# Solve Kohn-Sham equation
eig_non, psi_non = np.linalg.eigh(-D2/2)
```

**4. คำนวณพลังงานศักย์ภายนอก**

ลำดับต่อไปคือการพิจารณาศักย์ภายนอก (External Potential) ซึ่งเราสามารถใช้ฟังก์ชัน Harmonic Oscillator ง่าย ๆ ได้ ในตัวอย่างนี้ผู้เขียนเลือกใช้ External Potential เป็นฟังก์ชันพหุนาม คือ $v_{ext} = x^2$:

```
# Define external potential with a matrix
X = np.diagflat(x*x)

# Solve Kohn-Sham equation
```

```
5 eig_harm, psi_harm = np.linalg.eigh(-D2/2+X)
```

**5. คำนวณพลังงานแลกเปลี่ยน**

ลำดับต่อไปคือการคำนวณพลังงานแลกเปลี่ยน (Exchange Energy) โดยเราจะพิจารณาฟังก์ชันนอลแลกเปลี่ยน (Exchange Functional) โดยใช้ Local Density Approximation (LDA) ซึ่งมีสมการดังต่อไปนี้ (จริง ๆ แล้ว LDA มี Correlation Functional ด้วยแต่ว่าเราจะไม่สนใจ)

$$E_X^{LDA}[n] = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{1/3}\int n^{4/3}dx \tag{4.9.4}$$

โดยที่ Potential นั้นสามารถคำนวณได้จากอนุพันธ์ของ Exchange Energy เทียบกับความหนาแน่น

$$\begin{aligned} v_X^{LDA}[n] &= \frac{\partial E_X^{LDA}}{\partial n} \\ &= -\left(\frac{3}{\pi}\right)^{1/3} n^{1/3} \end{aligned} \tag{4.9.5}$$

```
1 def get_exchange(nx, x):
2     energy = -3./4.*(3./np.pi)**(1./3.)*integral(x, nx**(4./3.))
3     potential = -(3./np.pi)**(1./3.)*nx**(1./3.)
4     return energy, potential
```

**6. คำนวณพลังงานคูลอมป์**

ลำดับต่อไปคือพลังงานคูลอมป์ซึ่งเป็นพลังงานทางไฟฟ้าสถิตย์ (Electrostatic Energy) หรืออาจจะเรียกเรียกว่าพลังงานฮาร์ทรี Hartree Energy ก็ได้ อย่างไรก็ตาม ตามทฤษฎีนั้นพลังงานคูลอมป์สำหรับนั้นลู่เข้า (Converged) เฉพาะกรณี 3 มิติเท่านั้นซึ่งมีสมการดังต่อไปนี้

$$E_{Coul}^{3D} = \frac{1}{2}\iint \frac{n(r)n(r')}{\sqrt{(r-r')^2}}drdr' \tag{4.9.6}$$

ดังนั้นในกรณี 1 มิติเราจะต้องทำการโกงนิดหน่อยเพื่อทำให้พลังงานนั้นลู่เข้าโดยการปรับสมการ ดังนี้

$$E_{Coul}^{1D} = \frac{1}{2}\iint \frac{n(x)n(x')}{\sqrt{(x-x')^2+\varepsilon}}dxdx' \tag{4.9.7}$$

โดยที่ $\varepsilon$ คือคงที่ที่เป็นบวกที่มีค่าน้อย ๆ ซึ่งทำให้ฟังก์ชันนี้ลู่เข้าได้ง่ายขึ้น

ดังนั้นพลังงานศักย์จึงมีสมการดังต่อไปนี้:

$$v_{Coul} = \int \frac{n(x')}{\sqrt{(x-x')^2+\varepsilon}} dx' \tag{4.9.8}$$

```
def get_coulomb(nx, x, eps=1e-1):
    h = x[1]-x[0]
    energy = np.sum(nx[None,:]*nx[:,None]*h**2 /
   np.sqrt((x[None,:]-x[:,None])**2 + eps)/2)
    potential =
   np.sum(nx[None,:]*h/np.sqrt((x[None,:]-x[:,None])**2+eps),
   axis=-1)
    return energy, potential
```

**7. คำนวณความหนาแน่น**

เนื่องจากว่าเราจะต้องทำการรวม Coulomb Energy และ LDA Exchange โดยที่ทั้งคู่นั้นเป็นฟังก์ชันนอลของความหนาแน่น ดังนั้นเราจึงจำเป็นต้องคำนวณความหนาแน่นของอิเล็กตรอน (Electron Density) โดยเรามีเงื่อนไขของการทำ Normalization ดังนี้

$$\int |\psi|^2 dx = 1 \tag{4.9.9}$$

ซึ่งเราสามารถเขียนความหนาแน่นให้อยู่ในรูปของผลรวมเชิงเส้นของออร์บิทัลยกกำลังสองได้ ดังนี้

$$n(x) = \sum_n f_n |\psi(x)|^2 \tag{4.9.10}$$

โดยที่ $f_n$ คือ Occupation Number (จำนวนอิเล็กตรอนในออร์บิทัลที่ $n$) ซึ่งแต่ละ State นั้นจะมีอิเล็กตรอนที่มีสปินขึ้นและสปินลง โดยใน DFT นั้นเราคำนวณสถานะพื้นของระบบ

กำหนดจำนวนอิเล็กตรอน เช่น 17 ตัว

```
num_electron = 17
```

ทำการคำนวณความหนาแน่น

```
def get_nx(num_electron, psi, x):
    # Normalization
    I = integral(x, psi**2, axis=0)
    normed_psi = psi/np.sqrt(I)[None, :]

```

```
    # Occupation Number
    fn=[2 for _ in range(num_electron//2)]
    if num_electron % 2:
        fn.append(1)

    # Density
    res = np.zeros_like(normed_psi[:,0])
    for ne, psi in zip(fn, normed_psi.T):
        res += ne*(psi**2)

    return res
```

### 8. คำนวณพลังงานอิเล็กทรอนิกส์ของระบบ

เมื่อเราเตรียมองค์ประกอบทุกอย่างพร้อมแล้ว ขั้นตอนต่อไปนี้สำคัญมากเพราะว่าเป็นขั้นตอนสุดท้ายที่เราจะนำฟังก์ชันทั้งหมดที่เราได้เขียนไว้มาแก้สมการ Kohn-Sham (KS) โดยการวนซ้ำเทียบกับตัวเอง (Self-Consistency) มีขั้นตอนดังนี้

1. เริ่มต้นด้วยการกำหนดเมทริกซ์ความหนาแน่นของอิเล็กตรอน (Initialize) ซึ่งเราสามารถใช้ค่าคงที่อะไรก็ได้ (เพื่อให้ง่ายต่อการคำนวณ)
2. คำนวณพลังงานศักย์แลกเปลี่ยน (Exchange) และศักย์คูลอมบ์ (Coulomb Potential)
3. คำนวณ Hamiltonian
4. แก้สมการ KS เพื่อคำนวณหา Wavefunctions และ Eigenvalues (พลังงาน)
5. ตรวจสอบการลู่เข้า ถ้าไม่ลู่เข้า ให้อัพเดทความหนาแน่นและกลับไปที่ขั้นตอนที่ 2

ก่อนอื่นให้สร้างฟังก์ชันสำหรับแสดงผลการคำนวณพลังงานในระหว่างการวนซ้ำ (Iteration)

```
def print_log(i, log):
    print(f"step: {i:<5} energy: {log['energy'][-1]:<10.4f}
    energy_diff: {log['energy_diff'][-1]:.10f}")
```

กำหนดพารามิเตอร์เพิ่มเติม เช่น จำนวนรอบสูงสุดในการวนซ้ำและค่า Cutoff ของความแตกต่างระหว่างพลังงานจากรอบที่ $n$ และรอบที่ $n+1$

```
max_iter = 1000
energy_tolerance = 1e-5
# A dictionary to save energies
log = {"energy":[float("inf")], "energy_diff":[float("inf")]}
```

กำหนดค่าความหนาแน่นเริ่มต้นซึ่งจะถูกมานำใช้เป็นค่าเริ่มต้นในการประมาณค่าหาความหนาแน่นโดยการทำปรับค่าเทียบค่าความหนาแน่นที่ได้จากลูปในรอบก่อนหน้า โดยค่าความหนาแน่นเริ่มต้นนั้นเราจะกำหนดโดยใช้ค่าคงที่อะไรก็ได้ ในตัวอย่างนี้ผู้เขียนใช้ความหนาแน่นเท่ากับ 0 และสิ่งที่เกิดขึ้นภายในลูปนั้นเราจะทำการคำนวณพลังงาน Exchange และพลังงาน Coulomb ก่อนแล้วก็สร้าง Hamiltonian ขึ้นมา

แล้วก็ทำการ Diagonalize Hamiltonian เพื่อให้ได้ Eigenvalue ออกมาซึ่งนั่นก็คือพลังงานของเรานั่นเอน หลังจากนั้นเราจะทำการเก็บค่าพลังงานที่ได้แล้วก็ตรวจสอบว่าส่วนต่างของพลังงานที่ได้จากการวนลูปรอบปัจจุบันที่ $n$ กับรอบที่ $n-1$ นั้นต่ำกว่าค่า Cutoff แล้วหรือยัง ถ้าหากว่ายังก็ให้ทำการอัพเดทค่าความหนาแน่นแล้วทำการคำนวณพลังงานอีกรอบ

```
# Initialize density
nx = np.zeros(n_grid)

for i in range(max_iter):
    ex_energy, ex_potential = get_exchange(nx, x)
    ha_energy, ha_potential = get_coulomb(nx, x)

    # Hamiltonian
    H = -D2/2 + np.diagflat(ex_potential + ha_potential + x*x)

    energy, psi = np.linalg.eigh(H)

    # Collect energy and eenrgy difference
    log["energy"].append(energy[0])
    energy_diff = energy[0] - log["energy"][-2]
    log["energy_diff"].append(energy_diff)
    print_log(i, log)

    # Check if the calculation is converged
    if abs(energy_diff) < energy_tolerance:
        print("Converged!   :)")
        break

    # Update the density
    nx = get_nx(num_electron, psi, x)
else:
    print("Not Converged   :(")
```

เมื่อทำการรันโค้ดด้านบนแล้วจะได้เอาต์พุตดังต่อไปนี้

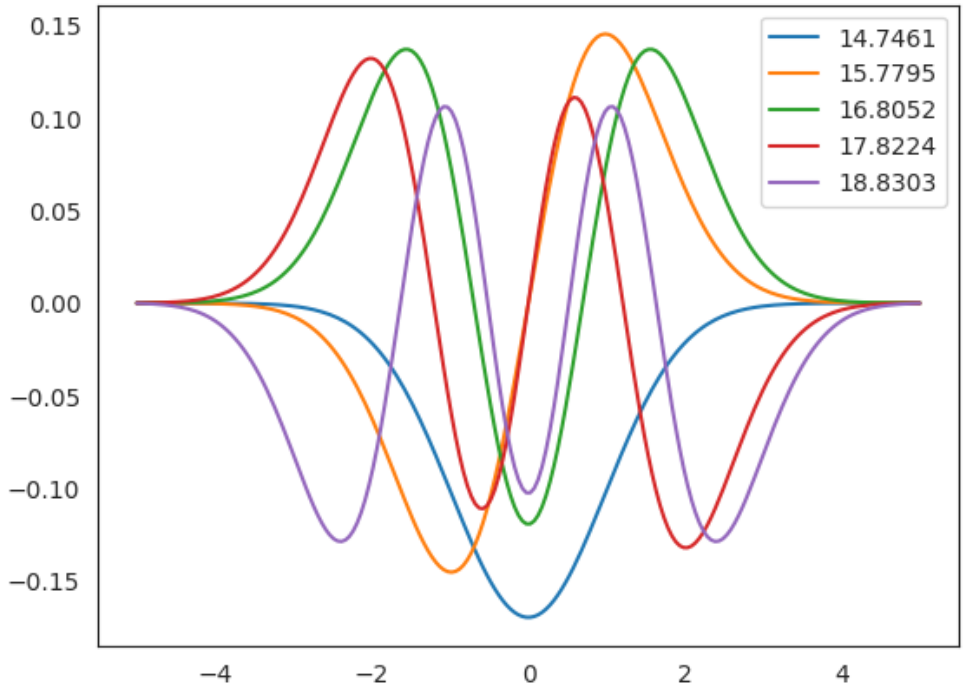
```
step: 0     energy: 0.7069     energy_diff: -inf
step: 1     energy: 16.3625    energy_diff: 15.6555321919
step: 2     energy: 13.8021    energy_diff: -2.5603559494
step: 3     energy: 15.3002    energy_diff: 1.4980525863
step: 4     energy: 14.4119    energy_diff: -0.8882287680
step: 5     energy: 14.9470    energy_diff: 0.5350438262
step: 6     energy: 14.6242    energy_diff: -0.3228271880
step: 7     energy: 14.8201    energy_diff: 0.1959328656
```

```
step: 8     energy: 14.7011    energy_diff: -0.1190355457
step: 9     energy: 14.7735    energy_diff: 0.0724651058
step: 10    energy: 14.7294    energy_diff: -0.0441312736
step: 11    energy: 14.7563    energy_diff: 0.0268946713
step: 12    energy: 14.7399    energy_diff: -0.0163922405
step: 13    energy: 14.7499    energy_diff: 0.0099933983
step: 14    energy: 14.7438    energy_diff: -0.0060926001
step: 15    energy: 14.7475    energy_diff: 0.0037147279
step: 16    energy: 14.7452    energy_diff: -0.0022649307
step: 17    energy: 14.7466    energy_diff: 0.0013810031
step: 18    energy: 14.7458    energy_diff: -0.0008420446
step: 19    energy: 14.7463    energy_diff: 0.0005134280
step: 20    energy: 14.7460    energy_diff: -0.0003130574
step: 21    energy: 14.7462    energy_diff: 0.0001908842
step: 22    energy: 14.7461    energy_diff: -0.0001163900
step: 23    energy: 14.7461    energy_diff: 0.0000709679
step: 24    energy: 14.7461    energy_diff: -0.0000432721
step: 25    energy: 14.7461    energy_diff: 0.0000263849
step: 26    energy: 14.7461    energy_diff: -0.0000160880
step: 27    energy: 14.7461    energy_diff: 0.0000098095
Converged!   :)
```

เมื่อทำการแก้หาค่าพลังงานไปทั้งหมด 27 รอบจะพบว่าพลังงานนั้นลู่เข้า โดยค่าพลังงานสุดท้ายที่ได้คือ 14.7461 และมีค่าความแตกต่างระหว่างพลังงานของรอบที่ 26 กับพลังงานของรอบที่ 27 เท่ากับ 0.0000098095 ซึ่งน้อยกว่า Cutoff ที่กำหนดไว้คือ 0.00001

นอกจากนี้เราสามารถพลอต Wavefunction ซึ่งเป็นฟังก์ชันของ Real-space Grid และระบุพลังงานได้ด้วย ดังนี้

```python
for i in range(5):
    plt.plot(x, psi[:,i], label=f"{energy[i]:.4f}")
    plt.legend(loc=1)
```




ภาพ 4.2 Wavefunction และพลังงานที่ได้จากการคำนวณ Kohn-Sham DFT สำหรับกรณี 1 มิติ

ผู้อ่านที่ต้องการศึกษาโค้ดฉบับสมบูรณ์สามารถดูได้ที่ไฟล์ `6_1D_DFT.ipynb` ใน Code Repository ของหนังสือ “ปัญญาประดิษฐ์สำหรับเคมีควอนตัม (Machine Learning for Quantum Chemistry)” ที่ https://github.com/rangsimanketkaew/ml-qm-book-code

## 4.10 เขียนโปรแกรมคำนวณ Molecular Quantum Integrals

ในเคมีควอนตัมนั้นเราจะต้องปวดหัววุ่นวายกับการคำนวณอินทิกรัล (Integrals) ที่ถือว่าเป็นหนามยอกอกของนักเคมีเชิงฟิสิกส์สายทฤษฎีมาอย่างยาวนาน นั่นก็คือ Molecular Integrals ที่ใช้ Gaussian Basis Functions ซึ่ง Integrals เทอมที่สำคัญ ๆ นั้นก็มีด้วยกันดังนี้

1. Overlap Integrals
2. Kinetic Energy Integrals
3. Nuclear Attraction Integrals
4. Two-Electron Repulsion Integrals

โดยในหัวข้อนี้ผู้อ่านจะได้เรียนรู้ทั้งทฤษฎี, อัลกอริทึม และการเขียนโปรแกรมเพื่อคำนวณเทอม Molecular Integrals ทั้ง 4 เทอม

### 4.10.1 ความรู้ทางคณิตศาสตร์ที่ต้องใช้

เริ่มต้นผมอยากจะให้ผู้อ่านได้ทำความเข้าใจคณิตศาสตร์ที่ควรจะต้องทราบก่อนที่จะไปทำความเข้าใจรายละเอียดของ Integrals โดยผมขอเริ่มด้วย Gaussian Function แบบสามมิติ ดังนี้

$$G_{ijk}(\mathbf{r}, \alpha, \mathbf{A}) = x_A^i y_A^j z_A^k \exp(-\alpha r_A^2) \tag{4.10.1}$$

ซึ่งเป็นฟังก์ชันที่ขึ้นกับพารามิเตอร์ดังต่อไปนี้: Orbital Exponent $\alpha$, Electronic Coordinates $\mathbf{r}$, Origin $\mathbf{A}$, และ

$$\mathbf{r}_A = \mathbf{r} - \mathbf{A} \tag{4.10.2}$$

แล้วก็ $i, j, k$ คือเลขควอนตัมเชิงมุม (Angular Quantum Numbers) เช่น $i = 0$ คือ $s$-type, $i = 1$ คือ $p$-type

เนื่องจากว่า Cartesian Gaussians เป็นฟังก์ชันที่ขึ้นกับทิศทาง , $x, y, z$ ดังนั้นจึงสามารถแยกฟังก์ชันออกจากกันได้ ดังนี้

$$G_{ijk}(\mathbf{r}, \alpha, \mathbf{A}) = G_i(x, \alpha, A_x) G_j(y, \alpha, A_y) G_k(z, \alpha, A_z) \tag{4.10.3}$$

โดยที่ฟังก์ชัน Gaussian แบบหนึ่งมิตินั้นมีสมการคือ

$$G_i(x, \alpha, A_x) = (x - A_x)^i \exp(-\alpha(x - A_x)^2) \tag{4.10.4}$$

คราวนี้เรามาดู Overlap Integral ของฟังก์ชัน Gaussian แบบหนึ่งมิติ จำนวน 2 ฟังก์ชันกันครับ นั่นคือ $a$ and $b$ ดังนี้

$$S_{ab} = \int G_i(x, \alpha, A_x) G_j(x, \beta, B_x) dx \tag{4.10.5}$$

$$= \int K_{AB} x_A^i x_B^j \exp(-p x_P^2) dx \tag{4.10.6}$$

โดยที่เราสามารถใช้คุณสมบัติการคูณของฟังก์ชัน Gaussian ได้ ดังนี้

$$K_{AB} = \exp(-q Q_x^2) \tag{4.10.7}$$

ซึ่ง $q$ และ $Q$ นั้นมีนิยามดังนี้

$$Q_x = A_x - B_x \qquad q = \frac{\alpha\beta}{\alpha+\beta} \tag{4.10.8}$$

$$p = \alpha + \beta \qquad P_x = \frac{1}{p}\left(\alpha A_x + \beta B_x\right) \tag{4.10.9}$$

ถ้าหากว่าเราใช้ฟังก์ชัน Hermite Gaussians เราจะสามารถเขียน $S_{ab}$ ได้ใหม่ดังนี้

$$S_{ab} = \int \sum_{t=0}^{i+j} E_t^{ij} \Lambda_t dx \tag{4.10.10}$$

$$= \sum_{t=0}^{i+j} E_t^{ij} \int \Lambda_t dx \tag{4.10.11}$$

$$= \sum_{t=0}^{i+j} E_t^{ij} \delta_{t0} \sqrt{\frac{\pi}{p}} \tag{4.10.12}$$

$$= E_0^{ij} \sqrt{\frac{\pi}{p}} \tag{4.10.13}$$

จะเห็นว่าเราสามารถลดรูปของ Overlap Matrix $S_{ab}$ ให้ง่ายขึ้นได้โดยการใช้คุณสมบัติของ Summation และ Integral แล้วเราก็ยังสามารถทำให้ Summation หายไปได้โดยการยุบรวม $\sum_{t=0}^{i+j} E_t^{ij}\delta_{t0}$ เข้าด้วยกัน แล้วเราก็มี $E_t^{ij}$ ซึ่งก็คือสัมประสิทธิ์การกระจาย (Expansion Coefficients) ซึ่งเราสามารถคำนวณได้โดยใช้วิธีการวนซ้ำ และ $\Lambda_t$ คือฟัง Hermite Gaussian Overlap ระหว่างฟังก์ชัน Gaussians $a$ กับ $b$ สำหรับการคำนวณหา $E_t^{ij}$ เราสามารถใช้สมการต่อไปนี้

$$E_t^{ij} = \frac{1}{2p} E_{t-1}^{i,j-1} + \frac{qQ_x}{\beta} E_t^{i,j-1} + (t+1) E_{t+1}^{i,j-1} \tag{4.10.14a}$$

$$E_t^{ij} = \frac{1}{2p} E_{t-1}^{i-1,j} - \frac{qQ_x}{\alpha} E_t^{i-1,j} + (t+1) E_{t+1}^{i-1,j} \tag{4.10.14b}$$

โดยสมการที่ (4.10.14a) นั้นช่วยให้เราสามารถลดค่า Index $j$ และสมการที่ (4.10.14b) นั้นช่วยให้เราลดค่าของ Index $i$ ซึ่งทำให้เราได้สมการที่สั้นมากขึ้นนั่นคือสมการที่ (4.10.15a)

$$E_0^{00} = K_{AB} \tag{4.10.15a}$$

$$E_t^{ij} = 0 \quad \text{ถ้า} \quad t < 0, \quad \text{หรือ} \quad t > i + j \tag{4.10.15b}$$

สรุปก็คือค่าของ $E_t^{ij}$ นั้นจะขึ้นอยู่กับค่าของดัชนี $i$, $j$, และ $t$ เช่น ถ้าหากว่า $i = j = t = 0$ เราก็จะได้ Expansion Coefficient ตามสมการที่ (4.10.15b) นั่นเอง

### 4.10.2 อินทิกรัลซ้อนทับ (Overlap Integrals)

อินทริกรัลอันแรกที่ผู้อ่านจะได้ศึกษาก็คือ Overlap Integrals ซึ่งเราเพิ่งดูรายละเอียดการคำนวณหาอินทิกรัลกันไปในหัวข้อที่แล้วโดยการใช้ Expansion Coefficient $E_t^{ij}$ โดยในหัวข้อนี้เราจะเริ่มด้วยการเขียนฟังก์ชัน `E` สำหรับคำนวณ $E_t^{ij}$ ซึ่งนอกจากที่เราจะต้องรู้ค่าของ Angular Momentum $i$ และ $j$ จากฟังก์ชัน Gaussian แล้ว เรายังจะต้องรู้ค่าของระยะห่างระหว่าง Gaussian Function $Q_x$ ด้วย รวมถึงค่าของ Orbital Exponent Coefficients $\alpha$ กับ $\beta$

```python
def E(i, j, t, Qx, a, b):
    """
    Recursive definition of Hermite Gaussian coefficients.

    Returns a float.
    a: orbital exponent on Gaussian 'a' (e.g. alpha in the text)
    b: orbital exponent on Gaussian 'b' (e.g. beta in the text)
    i,j: orbital angular momentum number on Gaussian 'a' and 'b'
    t: number nodes in Hermite (depends on type of integral,
        e.g. always zero for overlap integrals)
    Qx: distance between origins of Gaussian 'a' and 'b'
    """
    p = a + b
    q = a * b / p
    if (t < 0) or (t > (i + j)):
        # out of bounds for t
        return 0.0
    elif i == j == t == 0:
        # base case
        return np.exp(-q * Qx * Qx)  # K_AB
    elif j == 0:
        # decrement index i
        return (
            (1 / (2 * p)) * E(i - 1, j, t - 1, Qx, a, b)
            - (q * Qx / a) * E(i - 1, j, t, Qx, a, b)
```

```
            + (t + 1) * E(i - 1, j, t + 1, Qx, a, b)
        )
    else:
        # decrement index j
        return (
            (1 / (2 * p)) * E(i, j - 1, t - 1, Qx, a, b)
            + (q * Qx / b) * E(i, j - 1, t, Qx, a, b)
            + (t + 1) * E(i, j - 1, t + 1, Qx, a, b)
        )
```

จะเห็นว่าสำหรับ Overlap แบบหนึ่งมิติ (1D) ระหว่าง Gaussian Function 2 อันนั้น เราก็แค่ทำการคำนวณ $E_0^{ij}$ แล้วก็คูณด้วย $\sqrt{\frac{\pi}{p}}$ แล้วก็ถ้าเป็นแบบสามมิติ (3D) เราก็แค่นำ Overlap แบบ 1D สำหรับ $x, y, z$ มาคูณเข้าด้วยกัน ดังนั้น เราสามารถเขียนโค้ดสำหรับการคำนวณ 3D Overlap ได้ดังนี้

```
import numpy as np


def overlap(a, lmn1, A, b, lmn2, B):
    """
    Evaluates overlap integral between two Gaussians

    Returns a float.
    a: orbital exponent on Gaussian 'a' (e.g. alpha in the text)
    b: orbital exponent on Gaussian 'b' (e.g. beta in the text)
    lmn1: int tuple containing orbital angular momentum (e.g.
    (1,0,0))
          for Gaussian 'a'
    lmn2: int tuple containing orbital angular momentum for
    Gaussian 'b'
    A: list containing origin of Gaussian 'a', e.g. [1.0, 2.0,
    0.0]
    B: list containing origin of Gaussian 'b'
    """
    l1, m1, n1 = lmn1  # shell angular momentum on Gaussian 'a'
    l2, m2, n2 = lmn2  # shell angular momentum on Gaussian 'b'
    S1 = E(l1, l2, 0, A[0] - B[0], a, b)  # X
    S2 = E(m1, m2, 0, A[1] - B[1], a, b)  # Y
    S3 = E(n1, n2, 0, A[2] - B[2], a, b)  # Z

    return S1 * S2 * S3 * np.power(np.pi / (a + b), 1.5)
```

เราใช้ไลบรารี่ NumPy เพื่อที่ว่าเราสามารถใช้ค่าคงที่ของ $\pi$ และเลขกำลังแบบเศษส่วน $(3/2)$ ได้

นั่นเอง แล้วก็การเขียนฟังก์ชันด้านบน `overlap` และ `E` ทั้งสองอันนั้นก็เพียงพอแล้วต่อการคำนวณหา Overlap Integrals ระหว่าง Gaussian Function แบบที่เป็น Primitive แต่ประเด็นก็คือว่า Basis Functions ส่วนใหญ่ในการคำนวณทางควอนตัมนั้นเป็นแบบ Contracted Basis Functions นั่นก็คือเป็น Basis Function ที่เกิดจากการรวมเอา Gaussian Primitive Function หลาย ๆ อันมารวมกัน ซึ่งจริง ๆ ถ้าหากว่าเราจะต้องหา Overlap Integrals ระหว่าง Contracted Gaussian Function นั้นก็ทำได้ไม่ยาก โดยสามารถดูได้ในฟังก์ชัน `S(a,b)` ดังนี้

```
def S(a, b):
    """
    Evaluates overlap between two contracted Gaussians

    Returns float.
    Arguments:
    a: contracted Gaussian 'a', BasisFunction object
    b: contracted Gaussian 'b', BasisFunction object
    """
    s = 0.0
    for ia, ca in enumerate(a.coefs):
        for ib, cb in enumerate(b.coefs):
            s += (
                a.norm[ia]
                * b.norm[ib]
                * ca
                * cb
                * overlap(a.exps[ia], a.shell, a.origin,
    b.exps[ib], b.shell, b.origin)
            )

    return s
```

เมื่อเราได้ฟังก์ชัน `S(a,b)` แล้ว สิ่งที่เราจะทำต่อไปก็คือเราจะมาลองทดสอบคำนวณค่าของ $S$ กันครับ โดยเราจะลองคำนวณหา $S$ ของ Basis Function 2 อันที่เหมือนกัน ซึ่งคำตอบที่เราจะต้องได้ออกมานั้นแน่นอนว่าจะต้องเท่ากับ 1 เพราะว่าฟังก์ชันที่เหมือนกันก็จะ Overlap กันได้พอดี โดยเราจะต้องมาเขียนโค้ดสำหรับสร้าง Basis Function กันก่อน โดยเราจะใช้ `class` สำหรับ `BasisFunction` แล้วเราก็จะใช้ `class` ดังกล่าวในการสร้าง Object ที่จะมีข้อมูลของ Basis Function เป็นจำนวนหลาย ๆ อัน โดยโค้ดที่เราจะเขียนนั้นจะอ้างอิงตามสมการต่อไปนี้[1]

เริ่มด้วย Self-Overlap Integral คือ

[1] ดูรายละเอียดการพิสูจน์ได้ที่ Fundamentals of Molecular Integrals Evaluation โดย Justin T. Fermann และ Edward F. Valeev[30]

$$\int \phi^*(\mathbf{r})\phi(\mathbf{r})d\mathbf{r} = \frac{N^2\pi^{3/2}(2l-1)!!\,(2m-1)!!\,(2n-1)!!}{2^{l+m+n}} \sum_{i,j}^{n} \frac{a_i a_j}{(\alpha_i+\alpha_j)^{l+m+n+3/2}} \tag{4.10.16}$$

เนื่องจากว่า $l+m+n = L$ (โมเมนตัมเชิงมุมของแต่ละชั้น (Shell)) เราจะสามารถปรับสมการเพื่อคำนวณหา Normalization Factor $N$ ได้ดังนี้

$$\int \phi^*\phi = \frac{N^2\pi^{3/2}(2l-1)!!\,(2m-1)!!\,(2n-1)!!}{2^{L}} \sum_{i,j}^{n} \frac{a_i a_j}{(\alpha_i+\alpha_j)^{L+3/2}} \tag{4.10.17}$$

$$= 1 \tag{4.10.18}$$

$$N = \left[\frac{\pi^{3/2}(2l-1)!!\,(2m-1)!!\,(2n-1)!!}{2^{L}} \sum_{i,j}^{n} \frac{a_i a_j}{(\alpha_i+\alpha_j)^{L+3/2}}\right]^{-1/2} \tag{4.10.19}$$

แล้วเราก็จะสามารถเขียนโค้ดเพื่อ Implement สมการที่ ได้ดังนี้

```python
from scipy.special import factorial2 as fact2


class BasisFunction(object):
    """
    A class that contains all our basis function data

    Attributes:
    origin: array/list containing the coordinates of the Gaussian origin
    shell: tuple of angular momentum
    exps: list of primitive Gaussian exponents
    coefs: list of primitive Gaussian coefficients
    norm: list of normalization factors for Gaussian primitives
    """

    def __init__(self, origin=[0.0, 0.0, 0.0], shell=(0, 0, 0), exps=[], coefs=[]):
        self.origin = np.asarray(origin)
        self.shell = shell
```

```
        self.exps = exps
        self.coefs = coefs
        self.norm = None
        self.normalize()

    def normalize(self):
        """
        Routine to normalize the basis functions,
        in case they do not integrate to unity.
        """
        l, m, n = self.shell
        L = l + m + n
        # self.norm is a list of length equal to number primitives
        # normalize primitives first (PGBFs)
        self.norm = np.sqrt(
            np.power(2, 2 * (l + m + n) + 1.5)
            * np.power(self.exps, l + m + n + 1.5)
            / fact2(2 * l - 1)
            / fact2(2 * m - 1)
            / fact2(2 * n - 1)
            / np.power(np.pi, 1.5)
        )

        # now normalize the contracted basis functions (CGBFs)
        # Eq. 1.44 of Valeev integral whitepaper
        prefactor = (
            np.power(np.pi, 1.5)
            * fact2(2 * l - 1)
            * fact2(2 * m - 1)
            * fact2(2 * n - 1)
            / np.power(2.0, L)
        )

        N = 0.0
        num_exps = len(self.exps)
        for ia in range(num_exps):
            for ib in range(num_exps):
                N += (
                    self.norm[ia]
                    * self.norm[ib]
                    * self.coefs[ia]
                    * self.coefs[ib]
```

```
                    / np.power(self.exps[ia] + self.exps[ib], L
    + 1.5)
                )

        N *= prefactor
        N = np.power(N, -0.5)
        for ia in range(num_exps):
            self.coefs[ia] *= N
```

ถ้าหากว่าเรามี Basis Function STO-3G ของออร์บิทัล $1s$ ของไฮโดรเจนที่มีจุดกำเนิดอยู่ที่ $(1.0, 2.0, 3.0)$ เราจะสามารถสร้าง Basis Function ได้ดังนี้

```
myOrigin = [1.0, 2.0, 3.0]
myShell = (0, 0, 0)  # p-orbitals would be (1,0,0) or (0,1,0) or
    (0,0,1), etc.
myExps = [3.42525091, 0.62391373, 0.16885540]
myCoefs = [0.15432897, 0.53532814, 0.44463454]
a = BasisFunction(origin=myOrigin, shell=myShell, exps=myExps,
    coefs=myCoefs)
```

โดยข้อมูลของ Basis Function ด้านบนนั้นสามารถดูได้จาก STO-3G ของ EMSL Basis Set Exchange Library ดังนี้

```
!  STO-3G  EMSL  Basis Set Exchange Library
! Elements                             References
! --------                             ----------
!  H - Ne: W.J. Hehre, R.F. Stewart and J.A. Pople, J. Chem.
    Phys. 2657 (1969).

****
H     0
S   3   1.00
      3.42525091           0.15432897
      0.62391373           0.53532814
      0.16885540           0.44463454
****
```

ดังนั้น $S(a, a) = 1.0$ เนื่องจากว่าการ Overlap ของ Basis Function กับตัวมันเองนั้นมีค่าเท่ากับ 1 เพราะว่าเราได้มีการใช้ Normalized Factor $N$ ด้วยนั่นเอง

### 4.10.3 อินทิกรัลพลังงานจลน์ (Kinetic Energy Integrals)

เรามาต่อกันที่อินทิกรัลอันที่สองนั่นก็คือ อินทิกรัลพลังงานจลน์ (Kinetic Energy Integrals) ซึ่งสามารถเขียนได้ในรูปของอินทิกรัลซ้อนทับ (Overlap Integral) ดังนี้[1]

$$T_{ab} = -\frac{1}{2}\left[P_{ij}^2 S_{kl} S_{mn} + S_{ij} P_{kl}^2 S_{mn} + S_{ij} S_{kl} P_{mn}^2\right] \tag{4.10.20}$$

where

$$D_{ij}^2 = j(j-1)S_{i,j-2} - 2\beta(2j+1)S_{ij} + 4\beta^2 S_{i,j+2} \tag{4.10.21}$$

สำหรับ Primitive Function แบบ 3D เราสามารถสร้างฟังก์ชันสำหรับคำนวณ Kinetic Integral ได้คล้าย ๆ กันกับฟังก์ชันของ Overlap Integral ดังนี้

```
1  def kinetic(a, lmn1, A, b, lmn2, B):
2      """
3      Evaluates kinetic energy integral between two Gaussians
4
5      Returns a float.
6      a: orbital exponent on Gaussian 'a' (e.g. alpha in the text)
7      b: orbital exponent on Gaussian 'b' (e.g. beta in the text)
8      lmn1: int tuple containing orbital angular momentum (e.g.
       (1,0,0))
9            for Gaussian 'a'
10     lmn2: int tuple containing orbital angular momentum for
       Gaussian 'b'
11     A: list containing origin of Gaussian 'a', e.g. [1.0, 2.0,
       0.0]
12     B: list containing origin of Gaussian 'b'
13     """
14     l1, m1, n1 = lmn1
15     l2, m2, n2 = lmn2
16     term0 = (
17         b * (2 * (l2 + m2 + n2) + 3) * overlap(a, (l1, m1, n1),
       A, b, (l2, m2, n2), B)
18     )
19     term1 = (
```

[1]นั่นจึงเป็นเหตุผลที่ว่าทำไมเราถึงต้องดูรายละเอียดของ Overlap Integral เป็นอันดับแรก เพราะว่าเป็นอินทิกรัลสำคัญที่เรานำไปใช้ในการคำนวณอินทิกรัลหรือเทอมอื่น ๆ ในวิชาโครงสร้างเชิงอิเล็กทรอนิกส์

```
        -2
        * np.power(b, 2)
        * (
            overlap(a, (l1, m1, n1), A, b, (l2 + 2, m2, n2), B)
            + overlap(a, (l1, m1, n1), A, b, (l2, m2 + 2, n2), B)
            + overlap(a, (l1, m1, n1), A, b, (l2, m2, n2 + 2), B)
        )
    )
    term2 = -0.5 * (
        l2 * (l2 - 1) * overlap(a, (l1, m1, n1), A, b, (l2 - 2, m2, n2), B)
        + m2 * (m2 - 1) * overlap(a, (l1, m1, n1), A, b, (l2, m2 - 2, n2), B)
        + n2 * (n2 - 1) * overlap(a, (l1, m1, n1), A, b, (l2, m2, n2 - 2), B)
    )
    return term0 + term1 + term2
```

และสำหรับ Contracted Gaussian Functions เราสามารถเขียนฟังก์ชัน `T(a,b)` ได้ดังนี้

```
def T(a, b):
    """
    Evaluates kinetic energy between two contracted Gaussians

    Returns float.
    Arguments:
    a: contracted Gaussian 'a', BasisFunction object
    b: contracted Gaussian 'b', BasisFunction object
    """
    t = 0.0
    for ia, ca in enumerate(a.coefs):
        for ib, cb in enumerate(b.coefs):
            t += (
                a.norm[ia]
                * b.norm[ib]
                * ca
                * cb
                * kinetic(a.exps[ia], a.shell, a.origin, b.exps[ib], b.shell, b.origin)
            )
    return t
```

### 4.10.4 อินทิกรัลแรงดึงดูดเชิงนิวเคลียร์ (Nuclear Attraction Integrals)

ในหัวข้อนี้เราจะมาดูรายละเอียดของอินทิกรัลอีกอันหนึ่งซึ่งเป็น One-Body Integral นั่นคืออินทิกรัลแรงดึงดูดเชิงนิวเคลียร์ (Nuclear Attraction Integrals) โดยอินทิกรัลอันนี้จะแตกต่างจาก Overlap Integral และ Kinetic Integral ตรงที่เรามีการใช้โอเปอร์เรเตอร์ $1/r_C$ ซึ่งก็คือคูลอมป์ นั่นหมายความว่าเราไม่สามารถทำการแยกตัวประกอบ (Factorization) อินทิกรัลของเราให้อยู่ในรูปของพิกัดคาร์ทีเซียน (Cartesian Components) หรือ $x, y, z$ ได้นั่นเอง ดังนั้นในการคำนวณ Nuclear Attraction Integral เราจำเป็นต้องใช้ตัวช่วยนั่นก็คือ Hermite Coulomb Integral $(R^n_{tuv}(p, \mathbf{P}, \mathbf{C}))$ ซึ่งเป็นตัวแทนของอันตรกิริยาแบบคูลอมป์ (Coulomb Interaction) ที่เกิดขึ้นระหว่างการกระจายตัวของประจุแบบ Gaussian ในโมเลกุลซึ่งมีจุดศูนย์กลางคือ $\mathbf{P}$ ส่วนนิวเคลียสนั้นมีจุดศูนย์กลางอยู่ที่ $\mathbf{C}$ โดย Hermite Coulomb Integral นั้นมีองค์ประกอบคือ $E^{ij}_t$ ซึ่งสามารถหาได้ด้วยวิธีการวนซ้ำตามชุดสมการดังต่อไปนี้

$$R^n_{t+1,u,v} = tR^{n+1}_{t-1,u,v} + X_{PC}R^{n+1}_{t,u,v} \tag{4.10.22}$$
$$R^n_{t,u+1,v} = uR^{n+1}_{t,u-1,v} + Y_{PC}R^{n+1}_{t,u,v} \tag{4.10.23}$$
$$R^n_{t,u,v+1} = vR^{n+1}_{t,u,v-1} + Z_{PC}R^{n+1}_{t,u,v} \tag{4.10.24}$$
$$R^n_{0,0,0} = (-2p)^n F_n(pR^2_{PC}) \tag{4.10.25}$$

โดยที่ $F_n(T)$ คือฟังก์ชันของบอยส์ (Boys Function)

$$F_n(T) = \int_0^1 \exp(-Tx^2)x^{2n}dx \tag{4.10.26}$$

ซึ่ง Boys Function นั้นเป็นกรณีเฉพาะแบบพิเศษของ Kummer Confluent Hypergeometric Function ${}_1F_1(a, b, x)$ ดังนี้[1]

$$F_n(T) = \frac{{}_1F_1(n + \frac{1}{2}, n + \frac{3}{2}, -T)}{2n + 1} \tag{4.10.27}$$

โดยที่เราสามารถ Implement สมการด้านบนนี้ได้ง่าย ๆ โดยการใช้ฟังก์ชันของไลบรารี่ `SciPy` สำหรับ ${}_1F_1$ ซึ่งอยู่ในโมดูล `scipy.special`

```
1 def R(t, u, v, n, p, PCx, PCy, PCz, RPC):
2     """
3     Returns the Coulomb auxiliary Hermite integrals
```

[1] https://en.wikipedia.org/wiki/Confluent_hypergeometric_function

```
    Returns a float.
    Arguments:
    t,u,v: order of Coulomb Hermite derivative in x,y,z
           (see defs in Helgaker and Taylor)
    n: order of Boys function
    PCx,y,z: Cartesian vector distance between Gaussian
             composite center P and nuclear center C
    RPC: Distance between P and C
    """
    T = p * RPC * RPC
    val = 0.0
    if t == u == v == 0:
        val += np.power(-2 * p, n) * boys(n, T)
    elif t == u == 0:
        if v > 1:
            val += (v - 1) * R(t, u, v - 2, n + 1, p, PCx, PCy,
    PCz, RPC)
        val += PCz * R(t, u, v - 1, n + 1, p, PCx, PCy, PCz, RPC)
    elif t == 0:
        if u > 1:
            val += (u - 1) * R(t, u - 2, v, n + 1, p, PCx, PCy,
    PCz, RPC)
        val += PCy * R(t, u - 1, v, n + 1, p, PCx, PCy, PCz, RPC)
    else:
        if t > 1:
            val += (t - 1) * R(t - 2, u, v, n + 1, p, PCx, PCy,
    PCz, RPC)
        val += PCx * R(t - 1, u, v, n + 1, p, PCx, PCy, PCz, RPC)
    return val
```

และเราสามารถทำการกำหนด Boys Function โดยการเขียนฟังก์ชัน `boys(n,T)` ได้ดังนี้

```
from scipy.special import hyp1f1

def boys(n, T):
    return hyp1f1(n + 0.5, n + 1.5, -T) / (2.0 * n + 1.0)
```

แล้วเราก็สามารถคำนวณ $\mathbf{P}$ ได้โดยการใช้ Gaussian Product Center Rule ดังนี้

$$P = \frac{\alpha A + \beta B}{\alpha + \beta} \tag{4.10.28}$$

ซึ่งสามารถเขียนออกมาเป็นโค้ดได้ง่าย ๆ ดังนี้

```
def gaussian_product_center(a, A, b, B):
    return (a * A + b * B) / (a + b)
```

เมื่อเราคำนวณ Coulomb Auxiliary Hermite Integrals $R^n_{tuv}$ ออกมาได้แล้ว ขั้นตอนต่อไปคือเราสามารถคำนวณ Nuclear Attraction Integral โดยเทียบกับนิวเคลียสที่มีจุดศูนย์กลางอยู่ที่ $\mathbf{C}$, $V_{ab}(C)$ ได้โดยการใช้สมการต่อไปนี้

$$V_{ab}(C) = \frac{2\pi}{p} \sum_{t,u,v}^{\substack{i+j+1,\\ k+l+1,\\ m+n+1}} E^{ij}_t E^{kl}_u E^{mn}_v R^0_{tuv}(p, \mathbf{P}, \mathbf{C}) \tag{4.10.29}$$

ซึ่งเราก็จะสามารถ Implement สมการด้านบนนี้ให้เป็นโค้ดตามฟังก์ชัน `nuclear_attraction` ได้ดังนี้

```
def nuclear_attraction(a, lmn1, A, b, lmn2, B, C):
    """
    Evaluates kinetic energy integral between two Gaussians

    Returns a float.
    a: orbital exponent on Gaussian 'a' (e.g. alpha in the text)
    b: orbital exponent on Gaussian 'b' (e.g. beta in the text)
    lmn1: int tuple containing orbital angular momentum (e.g.
    (1,0,0))
          for Gaussian 'a'
    lmn2: int tuple containing orbital angular momentum for
    Gaussian 'b'
    A: list containing origin of Gaussian 'a', e.g. [1.0, 2.0,
    0.0]
    B: list containing origin of Gaussian 'b'
    C: list containing origin of nuclear center 'C'
    """
    l1, m1, n1 = lmn1
    l2, m2, n2 = lmn2
    p = a + b
    P = gaussian_product_center(a, A, b, B)  # Gaussian
    composite center
    RPC = np.linalg.norm(P - C)

    val = 0.0
```

```
    for t in range(l1 + l2 + 1):
        for u in range(m1 + m2 + 1):
            for v in range(n1 + n2 + 1):
                val += (
                    E(l1, l2, t, A[0] - B[0], a, b)
                    * E(m1, m2, u, A[1] - B[1], a, b)
                    * E(n1, n2, v, A[2] - B[2], a, b)
                    * R(t, u, v, 0, p, P[0] - C[0], P[1] - C[1],
P[2] - C[2], RPC)
                )
    val *= 2 * np.pi / p
    return val
```

และเราก็สามารถใช้ Contracted Gaussians ในการคำนวณอินทิกรัลของเราได้ตามที่แสดงในโค้ดดังต่อไปนี้

```
def V(a, b, C):
    """
    Evaluates overlap between two contracted Gaussians

    Returns float.
    Arguments:
    a: contracted Gaussian 'a', BasisFunction object
    b: contracted Gaussian 'b', BasisFunction object
    C: center of nucleus
    """
    v = 0.0
    for ia, ca in enumerate(a.coefs):
        for ib, cb in enumerate(b.coefs):
            v += (
                a.norm[ia]
                * b.norm[ib]
                * ca
                * cb
                * nuclear_attraction(
                    a.exps[ia], a.shell, a.origin, b.exps[ib],
b.shell, b.origin, C
                )
            )
    return v
```

ผู้อ่านจะต้องเข้าใจด้วยว่าอินทิกรัลที่เราคำนวณออกมาได้นั้นเป็นแค่ Nuclear Attraction เท่านั้น ซึ่งเรายังมี Nuclear Repulsion Integrals ที่เป็น Contribution มาจากอะตอมที่มีจุดศูนย์กลางอยู่ที่ $\mathbf{C}$ อีกด้วย ถ้าหากว่าเราต้องการคำนวณ Nuclear Attraction Contribution ทั้งหมดของทั้งโมเลกุล เราจะต้องทำการ

รวม Integral ทั้งหมดทุกเทอมของทุกอะตอมเข้าด้วยแล้วก็ทำตามปรับสเกลของแต่ละ Integral ตามประจุย่อย (Nuclear Charge) ของอะตอมนั้น ๆ

### 4.10.5 อินทิกรัลแรงผลักระหว่างอิเล็กตรอน (Two-Electron Repulsion Integrals)

เราได้ศึกษารายละเอียดของ One-Body Integrals ไปแล้ว ซึ่งเป็นอินทิกรัลพื้นฐานที่ผู้อ่านควรจะต้องทราบหากต้องการเขียนโปรแกรมคำนวณพลังงานของโมเลกุลด้วยวิธี Hartree-Fock ในหัวข้อนี้เราจะมาดูรายละเอียดของอินทิกรัลอันสุดท้ายซึ่งเป็นอินทิกรัลที่สำคัญมากและมีความซับซ้อนมากเช่นกัน นั่นคืออินทิกรัลของแรงผลักระหว่างอิเล็กตรอน (Electron-Electron Repulsion Integrals) ซึ่งเป็นอินทิกรัลแบบ Two-Body

ในการคำนวณหาอินทิกรัลอันนี้นั้น เราจะยังคงมีการใช้ Hermite Integrals ซึ่งเราจะต้องทำการหาผลรวมดังต่อไปนี้

$$g_{abcd} = \frac{2\pi^{5/2}}{pq\sqrt{p+q}} \sum_{\substack{t,u,v \\ i+j+1, \\ k+l+1, \\ m+n+1}} E_t^{ij} E_u^{kl} E_v^{mn} \sum_{\substack{\tau,\nu,\phi \\ i'+j'+1, \\ k'+l'+1, \\ m'+n'+1}} E_\tau^{i'j'} E_\nu^{k'l'} E_\phi^{m'n'} (-1)^{\tau+\nu+\phi} R^0_{t+\tau,u+\nu,v+\phi}(p, q, \mathbf{P}, \mathbf{Q}) \tag{4.10.30}$$

ซึ่งจะเห็นได้ว่าสมการด้านบนนี้น่ากลัวมาก มีความซับซ้อนมาก อย่างไรก็ตาม เนื่องจากว่าเราทราบว่า $p = \alpha + \beta$ เราจะกำหนดให้ $q = \gamma + \delta$ เพื่อที่ว่าเราจะได้มี Gaussian Function Exponents ของ $a$, $b$, $c$ และ $d$ ซึ่งเราสามารถเขียนสมการใหม่อีกสมการที่มีหน้าตาคล้าย ๆ กับ Nuclear Attraction Integrals ได้ดังนี้

```
1  def electron_repulsion(a, lmn1, A, b, lmn2, B, c, lmn3, C, d,
       lmn4, D):
2      """
3      Evaluates kinetic energy integral between two Gaussians
4
5      Returns a float.
6      a,b,c,d: orbital exponent on Gaussian 'a','b','c','d'
7      lmn1,lmn2
8      lmn3,lmn4: int tuple containing orbital angular momentum
9                 for Gaussian 'a','b','c','d', respectively
10     A,B,C,D: list containing origin of Gaussian 'a','b','c','d'
11     """
12     l1, m1, n1 = lmn1
```

```
    l2, m2, n2 = lmn2
    l3, m3, n3 = lmn3
    l4, m4, n4 = lmn4
    p = a + b  # composite exponent for P (from Gaussians 'a' and 'b')
    q = c + d  # composite exponent for Q (from Gaussians 'c' and 'd')
    alpha = p * q / (p + q)
    P = gaussian_product_center(a, A, b, B)  # A and B composite center
    Q = gaussian_product_center(c, C, d, D)  # C and D composite center
    RPQ = np.linalg.norm(P - Q)

    val = 0.0
    for t in range(l1 + l2 + 1):
        for u in range(m1 + m2 + 1):
            for v in range(n1 + n2 + 1):
                for tau in range(l3 + l4 + 1):
                    for nu in range(m3 + m4 + 1):
                        for phi in range(n3 + n4 + 1):
                            val += (
                                E(l1, l2, t, A[0] - B[0], a, b)
                                * E(m1, m2, u, A[1] - B[1], a, b)
                                * E(n1, n2, v, A[2] - B[2], a, b)
                                * E(l3, l4, tau, C[0] - D[0], c, d)
                                * E(m3, m4, nu, C[1] - D[1], c, d)
                                * E(n3, n4, phi, C[2] - D[2], c, d)
                                * np.power(-1, tau + nu + phi)
                                * R(
                                    t + tau,
                                    u + nu,
                                    v + phi,
                                    0,
                                    alpha,
                                    P[0] - Q[0],
                                    P[1] - Q[1],
                                    P[2] - Q[2],
                                    RPQ,
                                )
```

```
                                )

    val *= 2 * np.power(np.pi, 2.5) / (p * q * np.sqrt(p + q))
    return val
```

และเพื่อให้มีความสมบูรณ์มากกว่านี้ เราจะสามารถเขียนฟังก์ชันเพื่อคำนวณหา Electron-Electron Repulsion Integral สำหรับ Contracted Gaussians ได้ดังนี้

```
def ERI(a, b, c, d):
    """
    Evaluates overlap between two contracted Gaussians

    Returns float.
    Arguments:
    a: contracted Gaussian 'a', BasisFunction object
    b: contracted Gaussian 'b', BasisFunction object
    c: contracted Gaussian 'b', BasisFunction object
    d: contracted Gaussian 'b', BasisFunction object
    """
    eri = 0.0
    for ja, ca in enumerate(a.coefs):
        for jb, cb in enumerate(b.coefs):
            for jc, cc in enumerate(c.coefs):
                for jd, cd in enumerate(d.coefs):
                    eri += (
                        a.norm[ja]
                        * b.norm[jb]
                        * c.norm[jc]
                        * d.norm[jd]
                        * ca
                        * cb
                        * cc
                        * cd
                        * electron_repulsion(
                            a.exps[ja],
                            a.shell,
                            a.origin,
                            b.exps[jb],
                            b.shell,
                            b.origin,
                            c.exps[jc],
                            c.shell,
                            c.origin,
```

```
                                d.exps[jd],
                                d.shell,
                                d.origin,
                            )
                        )
    return eri
```

ทั้งหมดที่เราเพิ่งดูรายละเอียดกันไปนั้นคือสมการและการเขียนโค้ดของอินทิกรัลทั้งหมดที่จำเป็นต้องใช้ในการเขียนโปรแกรมเคมีควอนตัมด้วยวิธี Hartree-Fock นั่นเองครับ

### 4.10.6 การพิจารณาประสิทธิภาพเชิงการคำนวณ (Computational Efficiency)

จากที่เราได้ศึกษา Integrals แบบต่าง ๆ มาแล้วทั้งสมการคณิตศาสตร์และการเขียนโปรแกรม ผู้อ่านจะพบว่าโค้ดที่ผมเอามาให้ดูเป็นตัวอย่างนั้นสามารถทำงานได้ปกติ ได้ผลการคำนวณที่ถูกต้อง แต่ประเด็นสำคัญอีกอย่างหนึ่งก็คือเรื่องของประสิทธิภาพของโปรแกรมหรือโค้ด แน่นอนว่าโค้ดของเรารันได้ แต่ว่าจะโค้ดจะทำงานได้ช้าหรือเร็วนั้นก็ขึ้นอยู่กับว่าเราเขียนโค้ดอย่างไร นอกจากนี้ยังขึ้นกับภาษาคอมพิวเตอร์ที่เราใช้ในการเขียนด้วย โดยภาษาที่ผมเลือกมาใช้ในการเขียนโค้ดในหนังสือเล่มนี้นั้นส่วนใหญ่เป็นภาษา Python เพราะว่าเขียนได้ง่าย ผู้อ่านสามารถศึกษาตามได้ไม่ยากเมื่อเทียบกับภาษาอื่น ๆ

ถ้าหากว่าเราเขียนโค้ดออกมาแล้วนำไปใช้เลยโดยที่ไม่มีพิจารณาประสิทธิภาพเชิงการคำนวณก่อนนั้น โค้ดของเราก็อาจจะไม่สามารถทำงานได้เต็มที่บนคอมพิวเตอร์ที่มีประสิทธิภาพสูง เช่น Supercomputer ดังนั้นเราจะมาดูกันว่าเราจะสามารถปรับปรุงโค้ดของเราได้อย่างไรบ้างเพื่อที่ว่าจะสามารถนำไปใช้ได้จริง

ปัจจัยแรกเลยคือภาษาคอมพิวเตอร์ที่เรานำมาใช้ในการเขียนโค้ด จริง ๆ แล้วถ้าหากว่าเราอยากจะเขียนโค้ดแบบจริงจังเพื่อให้ทำงานได้อย่างมีประสิทธิภาพ เราไม่ควรเขียนโปรแกรมโดยการใช้เพียงแค่ภาษา Python เพราะว่าภาษา Python นั้นจะทำงานได้ช้ากว่าภาษาคอมพิวเตอร์อื่นที่เป็นแบบ Low-Level เหตุผลที่โค้ดของเราด้านบนที่ถูกด้วยเขียน Python ทำงานได้ช้านั้นก็เพราะว่าเรามีการวนลูปโดยการใช้ `for` แบบหลาย ๆ ชั้น (Nested Loop) ซึ่งเป็นตัวการที่ทำให้โค้ดของเราช้า ดังนั้นจึงจะเป็นการดีกว่าถ้าหากว่าเราเขียนบางส่วนของโค้ด Python ของเราด้วยเทคนิคต่าง ๆ โดยหนึ่งในนั้นก็คือการเขียนโค้ดใหม่ด้วย Cython (http://www.cython.org) ซึ่งจะช่วยให้โค้ดของเราทำงานได้เร็วขึ้นหลายสิบเท่าเลย ปัญหาอีกอย่างหนึ่งของ Python ก็คือมีการใช้ Global Interpreter Lock (GIL) ซึ่งทำให้เราไม่สามารถรันโค้ดของเราแบบขนาน (Parallel) ได้ ซึ่งถ้าหากว่าเราสามารถทำการแบ่ง Routine ในโค้ดการคำนวณ Integral ของเราออกเป็นหลาย ๆ อันและทำการรัน Routines เหล่านั้นด้วย CPUs หลาย ๆ อันได้ก็จะทำให้โค้ดเราทำงานได้เร็วขึ้นเยอะมาก โดยหนึ่งในวิธีการเขียนโค้ดแบบ Parallel นั้นก็คือการใช้วิธี OpenMP หรือใช้โมดูลของ Cython นั่นคือ `cython.parallel` โดยผู้อ่านสามารถนำเทคนิคเหล่านี้ไปใช้ในการทำให้โค้ดการคำนวณ Integrals ของเราได้ เช่น หลีกเลี่ยงการคำนวณแบบวนซ้ำหลาย ๆ รอบ (Explicit Recursion) ในฟังก์ชัน `E` และ `R`

นอกจากนี้ผู้อ่านยังสามารถใช้เทคนิคอื่น ๆ ได้อีก เช่น ใช้ Permutational Symmetry ของอินทิกรัล

เช่น เราสามารถทำการคำนวณ Two-Electron Repulsion Integral ได้เร็วขึ้นประมาณ 8 เท่าเพียงแค่นำคุณสมบัติของความสมมาตรของอินทิกรัลมาใช้ แล้วเราก็สามารถหลักเลี่ยงการคำนวณ Integrals หลาย ๆ อันที่มีค่าเท่ากับศูนย์ได้เพื่อเป็นการประหยัดเวลาในการคำนวณ

## 4.11 การเพิ่มประสิทธิภาพการแปลง Two-Electron Integral

อัลกอริทึมแบบที่มีประสิทธิภาพในการสเกล (Scaling Performance) ที่ต่ำนั้นทำให้การคำนวณทางเคมีควอนตัมนั้นเสียเวลาและเปลืองทรัพยากร เช่น พลังงาน ตัวอย่างหนึ่งที่เห็นได้ชัดนั่นก็คือการแปลงอินทิกรัล (Integral Transformation) ของเทอม Two-Electron Integral ในโปรแกรมทาง Electronic Structure ซึ่งถ้าหากว่าอัลกอริทึมที่เราใช้นั้นไม่มีประสิทธิภาพ การ Transformation นั้นก็จะมีการสเกลมากถึง $O(N^8)$ ทำให้การคำนวณของโมเลกุลที่มีขนาดใหญ่นั้นใช้เวลานานมากหรือแทบจะคำนวณไม่ได้เลย ดังนั้นจึงได้มีการปรับปรุงให้การ Transformation นั้นทำได้ง่ายขึ้นโดยเราสามารถลดการสเกลลงมาได้อยู่ในระดับ $O(N^5)$ เลยทีเดียว

ถ้าหากว่าเราคำนวณ Hartree-Fock (HF) เราจะต้องมีการนำเมทริกซ์ของ Molecular Orbital Coefficients หลาย ๆ อันมาคูณกัน ซึ่ง Coefficients เหล่านี้เป็น Eigenvectors ที่สอดคล้องกับ HF Hamiltonian (Fock Matrix) นั่นเอง ส่วนค่า Eigenvalues นั้นก็เป็นพลังงานของออร์บิทัล โดยเทคนิคที่เราสามารถนำมาใช้ในการลดความซับซ้อนของการคำนวณ Two-Electron Integral ลงได้นั้นก็คือเทคนิค Orbital Transformation ซึ่งเป็นการแปลง Two-Electron Integral จาก Atomic Orbital Basis (ซึ่งก็คือ Basis Functions ที่เรานำมาใช้ในการสร้าง Atomic Orbital) ให้กลายเป็น Molecular Orbital Basis แทน แล้วเราก็ทำการ Project อินทิกรัลที่คำนวณได้ไปบนฟังก์ชันคลื่นของโมเลกุล ซึ่งวิธีที่เราทำเริ่มต้นด้วย

$$(pq|rs) = \sum_{\mu}\sum_{\nu}\sum_{\lambda}\sum_{\sigma} C_{\mu}^{p} C_{\nu}^{q} C_{\lambda}^{r} C_{\sigma}^{s} (\mu\nu|\lambda\sigma) \tag{4.11.1}$$

ซึ่งเขียนเป็นโค้ดด้วยภาษา Python โดยการใช้ `for` ได้คร่าว ๆ ดังนี้

```
for p in range(0,dim):
  for q in range(0,dim):
    for r in range(0,dim):
      for s in range(0,dim):
        for mu in range(0,dim):
          for nu in range(0,dim):
            for lam in range(0,dim):
              for sig in range(0,dim):
                TxInt[p,q,r,s] +=
    C[p,mu]*C[q,nu]*C[r,lam]*C[s,sig]*UnTxInt[mu,nu,lam,sig]
```

จะเห็นได้ว่าถ้าเรารันโค้ดด้านบนนี้จะใช้เวลานานมาก ๆ เพราะว่าเรามีการวนซ้ำ `for` ถึง 8 ชั้นด้วยกัน โดยการสเกลนั้นจะแปรผันตรงตามจำนวนของ Basis Functions ที่เราใช้ยกกำลังด้วย 8 ซึ่งเยอะมาก ๆ

ถ้าหากว่าเราสังเกตสมการด้านบนนี้ดี ๆ จะพบว่า Coefficients แต่ละตัวนั้นไม่ขึ้นต่อกัน เช่น $C_\mu^p$ ก็ไม่ขึ้นกับ $C_\nu^q$ ดังนั้นเราจึงสามารถเขียนสมการใหม่ได้เป็น

$$(pq|rs) = \sum_\mu C_\mu^p [\sum_\nu C_\nu^q [\sum_\lambda C_\lambda^r [\sum_\sigma C_\sigma^s (\mu\nu|\lambda\sigma)]]] \tag{4.11.2}$$

หรือถ้าอยากให้เห็นภาพชัด ๆ และเข้าใจมากกว่านี้ก็สามารถเขียนโดยอินทิกรัลแต่ละอันแยกจากกันได้ ดังนี้

$$(\mu\nu|\lambda s) = \sum_\sigma C_\sigma^s (\mu\nu|\lambda\sigma) \tag{4.11.3}$$

$$(\mu\nu|rs) = \sum_\lambda C_\lambda^r (\mu\nu|\lambda s) \tag{4.11.4}$$

$$(\mu q|rs) = \sum_\nu C_\nu^q (\mu\nu|rs) \tag{4.11.5}$$

$$(pq|rs) = \sum_\mu C_\mu^p (\mu q|rs) \tag{4.11.6}$$

นั่นก็คือเราทำการแปลง Quarter Transformation ทั้งหมด 4 อัน โดยเราจะทำการบันทึก Transformation ที่คำนวณเสร็จแล้วเก็บไว้ใช้สำหรับการคำนวณ Transformation อันต่อไป ซึ่งท้ายที่สุดแล้วเราจะสามารถเขียนโค้ดของการแปลงดังกล่าวได้โดยการใช้ Loop เพียงแค่ 5 Loops เท่านั้น ซึ่งนั่นก็คือเป็นการลดสเกลลงเหลือแค่ $N^5$ ดังนี้

```
for p in range(0,dim):
  for mu in range(0,dim):
    temp[p,:,:,:] += C[p,mu]*UnTXInt[mu,:,:,:]
  for q in range(0,dim):
    for nu in range(0,dim):
      temp2[p,q,:,:] += C[q,nu]*temp[p,nu,:,:]
    for r in range(0,dim):
      for lam in range(0,dim):
        temp3[p,q,r,:] += C[r,lam]*temp2[p,q,lam,:]
      for s in range(0,dim):
        for sig in range(0,dim):
          TxInt[p,q,r,s] += C[s,sig]*temp3[p,q,r,sig]
```

โดยโค้ดเวอร์ชั่นที่สองนี้เขียนได้ดีกว่าเวอร์ชั่นแรกเยอะมาก นอกจากนี้จะสังเกตได้ว่าเราได้มีการใช้เมทริกซ์ `temp` เพื่อใช้ในการเก็บผลลัพธ์ที่ได้ระหว่างการคำนวณ Quarter Transformations แล้วก็ Transformation นั้นก็มีการใช้ Slice Notation ใน Python/Numpy ด้วยซึ่งทำให้เราสามารถทำ Transformation ได้ครบทุก Dimension ของเมทริกซ์ (แทนที่จะใช้ Indices)

ด้านล่างคือโค้ดแบบสมบูรณ์ของการทำ Integral Transformation โดยมีการบันทึกเวลาว่าทั้งสองวิธีนั้นใช้เวลานานแค่ไหนแล้วก็มาเปรียบเทียบกัน

```
import numpy as np
import time

# Dimension of arrays ... e.g number of basis functions
dim = 5
# For our first dumb O[N^8] method
MO1 = np.zeros((dim, dim, dim, dim))
# For our smarter O[N^5] method
MO2 = np.zeros((dim, dim, dim, dim))

INT = np.random.randint(
    9, size=(dim, dim, dim, dim)
)  # Our toy "two electron integrals"
# Toy "wavefunction coefficients"
C = np.random.randint(9, size=(dim, dim))

# First method: N^8

t0 = time.time()
for i in range(0, dim):
    for j in range(0, dim):
        for k in range(0, dim):
            for l in range(0, dim):
                for m in range(0, dim):
                    for n in range(0, dim):
                        for o in range(0, dim):
                            for p in range(0, dim):
                                MO1[i, j, k, l] += (
                                    C[i, m]
                                    * C[j, n]
                                    * C[k, o]
                                    * C[l, p]
                                    * INT[m, n, o, p]
                                )
t1 = time.time()
```

```
36
37 # Second method: N^5
38
39 t2 = time.time()
40 temp = np.zeros((dim, dim, dim, dim))
41 temp2 = np.zeros((dim, dim, dim, dim))
42 temp3 = np.zeros((dim, dim, dim, dim))
43 for i in range(0, dim):
44     for m in range(0, dim):
45         temp[i, :, :, :] += C[i, m] * INT[m, :, :, :]
46     for j in range(0, dim):
47         for n in range(0, dim):
48             temp2[i, j, :, :] += C[j, n] * temp[i, n, :, :]
49         for k in range(0, dim):
50             for o in range(0, dim):
51                 temp3[i, j, k, :] += C[k, o] * temp2[i, j, o, :]
52             for l in range(0, dim):
53                 for p in range(0, dim):
54                     MO2[i, j, k, l] += C[l, p] * temp3[i, j, k,
    p]
55 t3 = time.time()
56
57 # Set up random index to check correctness.
58 i = np.random.randint(dim)
59 j = np.random.randint(dim)
60 k = np.random.randint(dim)
61 l = np.random.randint(dim)
62
63 print(MO1[i, j, k, l])
64 print(MO2[i, j, k, l])
65 print("Time 1: ", t1 - t0)
66 print("Time 2: ", t3 - t2)
```

โดยได้เอาต์พุตออกมาดังนี้

```
1 1904496.0
2 1904496.0
3 Time 1:  0.2975327968597412
4 Time 2:  0.0035715103149414062
```

เมื่อเรารันโค้ดแล้วจะพบว่าระยะเวลาที่วิธีที่หนึ่งใช้นั้นจะนานกว่าวิธีที่สองมาก (ผมกำหนดจำนวน Basis Functions = 5) โดยใช้เวลาคือประมาณ 0.298 และ 0.004 วินาที ตามลำดับ ดังนั้นจึงสรุปได้ว่าวิธีที่สองนั้นมีประสิทธิภาพมากกว่าวิธีแรกในเชิงของการคำนวณอย่างมาก นอกจากนี้เรายังสามารถ Parallel อัล

กอริทึมของวิธีที่สองเพื่อให้ทำงานได้เร็วมากยิ่งขึ้นกว่านี้ได้อีกด้วย

## 4.12 คำแนะนำสำหรับการคำนวณเคมีควอนตัม

คนที่ทำวิจัยด้านเคมีควอนตัมก็จะมีเทคนิคหรือแนวทางในการทำการคำนวณเคมีควอนตัมของตนเองขึ้นอยู่กับหลายปัจจัย ตัวอย่างเช่น เรียนจากคอร์สของมหาวิทยาลัย, อ่านจากหนังสือหรือตำราต่างประเทศที่ต่างกัน, ได้รับคำชี้แนะจากการทำวิจัยจากหลาย ๆ คน รวมถึงประสบการณ์หรือความเคยชินกับโปรแกรมเคมีควอนตัมที่ตนเองถนัด อย่างไรก็ตาม ผมคิดว่าเราทุกคนควรจะต้องมีหลักการทางทฤษฎีที่ถูกต้องที่ควรจะต้องปฏิบัติตามกัน ซึ่งก็คือการเข้าใจทฤษฎีทางเคมีควอนตัมและโครงสร้างเชิงอิเล็กทรอนิกส์ ในหัวข้อนี้ผู้อ่านได้จะศึกษาแนวทางสำหรับการคำนวณเคมีควอนตัม

**1. การเตรียม Input File โครงสร้างของโมเลกุลและการเลือก Electronic State** ผมขอเริ่มด้วยสิ่งที่เป็นพื้นฐานที่สุดนั่นก็คือการเตรียมไฟล์อินพุต (Input) ซึ่งปกติแล้วเรามักจะทราบกันดีว่าเราสามารถแสดง (Represent) โครงสร้างของโมเลกุลได้ด้วยรูปแบบ (Format) 2 อัน นั่นคือ Cartesian (xyz) Format กับ Z-Matrix Format โดยใน xyz Format นั้นเราจะใส่ลิสต์รายละเอียดของอะตอมในโมเลกุลนั่นคือเลขอะตอมและพิกัด Cartesian Coordinates ของอะตอมแต่ละตัว ส่วนใน Z-Matrix Format นั้นจะเป็นการกำหนดโครงสร้างหรือ Geometry โดยการใช้พิกัดภายใน (Internal Coordinates) เช่น ความยาวพันธะ มุมพันธะ และมุมบิด โดยโปรแกรมควอนตัมเคมีส่วนใหญ่นั้นทำการกำหนดรูปแบบเริ่มต้นของโครงสร้างโมเลกุลเป็นแบบพิกัด Cartesian รวมถึงพารามิเตอร์อื่น ๆ เช่น กำหนดหน่วยเป็น Angstroms แต่ว่าโปรแกรมแต่ละโปรแกรมนั้นก็มักจะมีคีเวิร์ดให้สลับไปใช้หน่วยอื่น เช่น bohr พารามิเตอร์ที่สำคัญอีก 2 อันก็คือ Charge กับ Spin Multiplicity ของ Electronic State ที่เราสนใจที่จะศึกษา โดยประจุนั้นเป็นการระบุถึงจำนวนของอิเล็กตรอน ส่วน Spin Multiplicity หาได้จากจำนวนรูปแบบที่อิเล็กตรอนสามารถจัดคู่ได้ซึ่งเขียนแทนด้วย $|\Psi\rangle$ สำหรับแต่ละ Configuration นั่นเอง ตัวอย่างเช่น โมเลกุลมีเทน $CH_4$ เรากำหนดประจุเป็น 0 ส่วน Spin Multiplicity นั้นเป็น 1 เพราะว่าจำนวน Alpha Electron กับ Beta Electron นั้นเท่ากันใน Singlet State ดังนั้นจึงมี Configuration ได้แบบเดียว

**2. การเลือก Basis Set** ลำดับต่อไปคือเราจะต้องเลือก Basis Set ที่ต้องการใช้ อันนี้ก็เป็นปัญหาโลกแตกอีกอันหนึ่งในทางเคมีควอนตัม แต่ละคนก็จะมี Basis Set ที่ตัวเองชอบหรือจะต้องใช้ Basis Set ที่โดนบังคับให้ใช้ อย่างไรก็ตาม ผมขำแบ่งคำแนะนำเบื้องต้นในการเลือก Basis Set ออกเป็น 2 กลุ่มหลัก ๆ คือ Basis Set สำหรับการคำนวณด้วย DFT Method และด้วย Wavefunction-Based Method

1. สำหรับการคำนวณ Density Functional Theory (DFT) และ Hartree-Fock (HF) นั้นเรามักจะมีตัวเลือกทั่ว ๆ ไป เช่น 6-31G(d), 6-31G(d,p) แต่ถ้าเป็นไปได้ก็ไปใช้ Basis Set อื่นที่ใหม่กว่านี้ดีกว่า (มีบทความวิชาการรีวิวที่ให้เหตุผลว่าทำไมเราถึงไม่ควรใช้ Pople's Basis ตัวเล็ก ๆ) ถ้าเป็นไปได้ให้ลองใช้ Basis Set จากสำนัก Karlsruhe เช่น def2-SVP, def2-TZVP, def2-QZVP ถ้าหากว่า

เราต้องมีการคำนวณเยอะมาก ๆ ผมแนะนำตัว def2-TZVP เพราะว่าจะให้ผลการคำนวณทั้งหมดที่ Converge ดีกว่า

2. สำหรับ Wavefunction-Based Method เช่น MP2, CCSD, CCSD(T) นั่น แนะนำให้ใช้จากสำนักของ Dunning ที่เป็นตัว Correlation Consistent เช่น cc-pVXZ ตัวอย่างคือ cc-pVDZ, cc-pVTZ เป็นต้น ซึ่ง Basis Set ของตระกูลนี้มักจะให้ผลการคำนวณที่ Converge หรือว่าเข้าใกล้ Full Basis Set Limit นั่นเอง ถ้าต้องการคำนวณที่แม่นยำและถูกต้องมาก ๆ ก็ใช้ตัวใหญ่ ๆ ไปเลย เช่น cc-pV5Z

3. ถ้าต้องการศึกษาระบบโมเลกุลที่เกี่ยวข้องกับ Rydberg States, Long-Range Interaction, หรือประจุลบ (Anions) ควรจะต้องใช้ Basis Set ที่มี Diffuse Function ใส่เข้าไปด้วย เช่น เรามักจะเห็นว่ามีสัญลักษณ์เครื่องหมาย + หรือตัว D อยู่ใน Basis Set ถ้าเป็นของตระกูล Correlation-Consistent ก็จะเติมคำว่า “aug” เข้าไปข้างหน้า

4. Basis Set ของ Dunning นั้นถูกออกแบบมาให้เหมาะกับการ Correlate แค่อิเล็กตรอนที่อยู่วงนอกเท่านั้น (Valence Electrons) ซึ่งในการใช้งานจริง ๆ นั้นเราควรจะต้อง treat อิเล็กตรอนข้างใน (Core Electrons) แบบ Frozen หรือเราควรจะต้องใช้ Core-Valence Set เช่น cc-pwCVDZ

**3. การคำนวณที่ Ground State** ในการเริ่มโปรเจ็คใหม่นั้นเราอย่าเพิ่งรีบร้อนใช้วิธีที่อลังการงานสร้างเกินไปเพราะอาจจะสิ้นเปลืองการคำนวณได้ง่าย ๆ ดังนั้นควรเริ่มด้วยวิธีเบา ๆ เร็ว ๆ ในช่วงวันแข่งวิ่งจริง เช่น รันด้วยวิธี DFT แล้วก็ใช้ Basis Set กลาง ๆ เช่น def-SVP แล้วก็ใช้ Functional พื้นฐาน เช่น B3LYP (ปัญหาโลกแตกอีกอันหนึ่งคือเลือกใช้ Functional ตามความชอบ แล้วค่อยมาแตรียมอีกครั้งตอนคำนวณจริง ๆ ที่ต้องการผลการคำนวณที่ถูกต้องไปตีพิมพ์หรือนำเสนอต่อไป) ซึ่งวิธี DFT นั้นมี Computational Demand ที่ต่ำ $\mathcal{O}(N^4)$ เอง โดย $N$ คือจำนวนอิเล็กตรอนหรือ Basis Functions เมื่อเราพอมีผลการคำนวณระดับหนึ่งแล้วเราอาจจะใช้วิธีการที่สูงหรือซับซ้อนขึ้น ซึ่งจะให้ผลการคำนวณที่ถูกต้องขึ้นแต่ก็แลกมาด้วยการคำนวณที่สิ้นเปลืองกว่า DFT เช่น MP2, MP3 เพราะว่ามันสเกลตาม $\mathcal{O}(N^5)$ อย่างไรก็ตามต้องอย่าลืมว่าวิธี MPn มันไม่ได้ให้ผลที่ Converge หมายความว่ายิ่งเราเพิ่ม Order ของ Perturbation ไม่ได้หมายความว่าความถูกต้องนั้นมันจะมีมากขึ้นเรื่อย ๆ ซึ่งถ้าอยากใช้วิธี MPn ที่ Order สูง ๆ ให้ไปใช้วิธีอื่นที่แฟนซีมากกว่านี้ดีกว่า เช่น Coupled Cluster (CC) Method สำหรับ CC นั้นก็ใช้เริ่มด้วย CCSD ก่อนก็ได้ แต่ว่าก็ไม่ได้ให้ผลการคำนวณที่ถูกต้องมากนักถ้าเทียบกับ CCSD(T) ซึ่งสิ้นเปลืองกว่าหน่อยนึงแต่ว่าให้ผลการคำนวณที่ถูกต้องกว่ามาก แต่ขึ้นชื่อว่า CC มันย่อมสิ้นเปลืองอยู่แล้ว ซึ่งวิธี CC นั้นมีสเกลถึง $\mathcal{O}(N^6)$

**4. การคำนวณที่ Excited State** ถ้าอยากจะคำนวณ Electronic Structure ของโมเลกุลที่ Excited State ก็ให้เริ่มด้วย Time-Dependent DFT (TDDFT) ก็ได้ ถึงแม้ว่าวิธี TDDFT ไม่สามารถให้ผลการคำนวณที่ถูกต้องได้ในหลาย ๆ กรณีก็ตาม แต่ว่าก็ไม่ได้ให้ผลการคำนวณทั่ว ๆ ไปของ Excited State ที่แย่มากนัก นอกจากนี้วิธี TDDFT นั้นจริง ๆ แล้วก็ไม่ได้สิ้นเปลืองมากนัก เพราะว่าตัว Formalism นั้นแบบเดียวกันกับวิธี DFT เลยคือการสร้าง Density Matrix แล้วก็ทำ Diagonalization สำหรับการใช้วิธี TDDFT นั้นเราก็ควรจะใช้ Functional พิเศษที่ถูกดีไซน์มาเพื่อ Excited State ด้วย เพราะว่า Functional พวกนี้มันจะสามารถอธิบายเหตุการณ์บางอย่างได้ เช่น Rydberg Effect หรือการกระตุ้นของ Charge/Electron

Transfer ซึ่งจำเป็นมาก ๆ โดยเฉพาะการคำนวณพวก Spectra ต่าง ๆ สำหรับวิธี Wavefunction-Based Method นั้นก็ให้เริ่มด้วยวิธี Configuration Interaction ที่ใส่ Single Excitation เข้าไป เรียกว่า CIS ซึ่งเป็นวิธีที่ทำการรวม Slater Determinant ของ Singly Excited นั่นเองซึ่งได้มาจากการคำนวณที่มีการกระตุ้นอิเล็กตรอนหนึ่งตัวขึ้นไปใน Virtual Orbital[1] ถึงแม้ว่า CIS จะไม่แม่นยำแต่ก็ไม่สิ้นเปลืองเลย ส่วนถ้าอยากจะใช้วิธี Coupled Cluster สำหรับศึกษา Excited State นั้นจะต้องใช้สิ่งที่เรียกว่า Equation-of-Motion เข้ามาช่วย ซึ่งก็คือวิธี EOM-CC หรือถ้าอีกวิธีก็คือใช้ Linear Response เรียกว่า LR-CC โดยวิธี EOM-CC นั้นโคตรสิ้นเปลืองแต่ว่าให้ผลที่ถูกต้องมาก นอกจากนี้ยังมีวิธีอื่นอีก เช่น Algebraic Diagrammatic Construction (ADC) หรือ CC2, CC3 ซึ่งเป็น Approximation แบบหนึ่งของ EOM-CC ซึ่งจะสิ้นเปลืองน้อยกว่า

**5. การแก้ปัญหาที่เกิดขึ้นในการคำนวณเคมีควอนตัม** คราวนี้เราลองมาดูปัญหายอดฮิตที่เรามักจะเจอกันรวมถึงวิธีหรือคำแนะนำในการแก้ปัญหา ซึ่งปัญหาหลักที่ผมจะเน้นนั้นก็คือการที่ Self-Consistent Field (SCF) Calculation นั้นไม่ Converge วิธีการแก้ที่อาจจะลองเอาไปใช้คือ

1. เพิ่มจำนวนรอบของการรัน SCF
2. เปลี่ยน Initial Orbital Guess เริ่มต้นที่เราเอามาใช้ในการสร้างฟังก์ชันคลื่น
3. เปลี่ยน Basis Set ซึ่งให้ลองใช้ Basis Set ที่มีขนาดเล็กลงมาแล้วก็ค่อยเอาไปใช้เป็น Orbital Guess สำหรับการคำนวณที่ใช้ Basis Set ใหญ่กว่าได้
4. ใช้ Trick อื่น ๆ เช่น ปรับ Level Shift ซึ่งเป็นการปรับพลังงานของ Virtual Orbitals ซึ่งมีประโยชน์มากสำหรับระบบโมเลกุลบางอันที่ซับซ้อน เช่น Biradical หรือ Triplet State

**6. ทำ Stability Analysis หลังการรันทุกครั้ง** หลายคนนั้นหลังจากที่รันการคำนวณเสร็จแล้วก็มักจะดีใจและนำค่าต่าง ๆ จาก Output File ที่ได้จากการคำนวณมาใช้ ซึ่งการทำแบบนี้นั้นจริง ๆ แล้วถือว่าเสี่ยงมาก ถึงแม้ว่าระบบโมเลกุลที่เราคำนวณนั้นจะเล็กก็ตาม ที่ผมบอกว่าเสี่ยงนั้นหมายความว่าผลการคำนวณที่เราได้ออกมานั้นอาจจะผิดได้ ยกตัวอย่างเช่นโมเลกุล $N_2$ นั้นบางครั้งให้ผลการคำนวณที่ไม่สมเหตุสมผลมาก ๆ โดยปัญหาที่ทราบกันดีนั่นก็คือพลังงานของ LUMO ต่ำกว่าของ HOMO ซึ่งเป็นไปไม่ได้เลย เหตุผลก็คือ SCF นั้นมัน Converge เข้าไปหา Electronic State ที่ผิดเพราะว่าเกิดจาก Guess เริ่มต้นที่ใช้ในการคำนวณ SCF นั้นผิด โดยเกี่ยวข้องกับ Occupation ของออร์บิทัลที่มาจาก Symmetry หรือสมมาตรของโมเลกุลนั่นเอง โดย $N_2$ นั้นมี Irreducible Representation ทั้งหมด 8 อัน ($D_{2h}$) ซึ่ง SCF นั้นจะต้องเดาว่ามีจำนวน Irreducible Representation เท่าไหร่ที่จะต้องนำมาอิเล็กตรอนเข้ามาใส่ ถ้าหากว่าเดาผิดก็จะทำให้ SCF นั้นผิดและทำให้เลข Occupation Number นั้นผิดไปด้วย วิธีการเช็คง่าย ๆ คือให้ดูที่ Occupation ของ Doubly Occupied Orbitals โดยโปรแกรมแต่ละโปรแกรมก็จะมี Keyword ที่เราสามารถใช้ในการเช็ค Stability ของฟังก์ชันคลื่นที่ต่างกันไป

[1]Virtual Orbital ก็คือ Unoccupied Orbital หรือออร์บิทัลที่ไม่มีอิเล็กตรอนอยู่ในสถานะพื้น

**7. การปรับโครงสร้างโมเลกุล (Geometry Optimization)** การปรับโครงสร้างโมเลกุล (Geometry Optimization) นั้นเป็นสิ่งที่ทุกคนนั้นจะต้องมีประสบการณ์ทำกันมาแล้วทั้งนั้น คำแนะนำสำหรับการรัน Geometry Optimization มีดังนี้ครับ

1. การรัน Geometry Optimization นั้นใช้เวลานานกว่าจะ Converge เพราะว่าโครงสร้างนั้นจะต้องประมาณค่าของ Hessian Matrix แทนที่จะคำนวณหา Hessian ที่ถูกต้อง ดังนั้นเราควรเริ่มด้วยการใช้ Full Hessian Matrix เพื่อทำให้ Converge เร็วขึ้น
2. โมเลกุลบางระบบปรับโครงสร้างได้ยากมาก ๆ การที่เราใช้ Coordinate System ที่ซับซ้อนก็อาจจะช่วยทำให้การรันสามารถ Converge ได้
3. เมื่อสามารถหา Stationary Point ได้แล้ว ให้ทำการคำนวณ Frequency Analysis เพื่อตรวจสอบว่าโครงสร้างที่เราได้นั้นเป็น Local Minimum หรือ Transition State กันแน่
4. เราไม่ควรเริ่มการปรับโครงสร้างด้วยการใช้โครงสร้างที่มี Symmetry สูง ๆ เช่น $D_{2h}$ เพราะว่าโดยหลักการแล้วตัว Optimizer นั้นไม่สามารถปรับโครงสร้างโมเลกุลจาก High symmetry ไปยังโครงสร้างที่มี Low symmetry ได้ ดังนั้นเราควรเริ่มด้วยการไม่ใช้ Symmetry และปล่อยให้ Optimizer นั้นมันเช็คเองว่าโครงสร้างอันนั้นมีสมมาตรหรือเปล่า

## 4.13 เรื่องที่หลายคนเข้าใจผิดเกี่ยวกับโปรแกรมเคมีควอนตัม

โปรแกรมเคมีควอนตัมแต่ละเจ้าก็จะมีการใช้ทฤษฎีที่เอามาใช้ในการคำนวณพลังงานของโมเลกุลที่แตกต่างกัน ก็แล้วแต่ว่าอยากจะศึกษาโมเลกุลแบบไหน แต่ละค่ายก็จะเคลมว่าของตัวเองดีอย่างนั้นดีอย่างนี้

แต่ละโปรแกรมมี Framework, Theory, Method ที่ใช้ในการคำนวณหรือจำลองโมเลกุลที่แตกต่างกันไป ขึ้นกับว่าใช้สูตรไหนในการอธิบาย โดยทั่วไปแล้วโปรแกรมเคมีควอนตัมจะมีการใช้วิธีดังต่อไปนี้ในการอธิบาย Electron Density

1. ฟังก์ชันคลื่น (Wavefunction) กรณีที่พูดถึง Wavefunction-Based Method เช่น MPn
2. ความหนาแน่น (Density) กรณีที่พูดถึง Density-Based Method เช่น DFT

เพื่อนำมาใช้ในการคำนวณพลังงานรวมและคุณสมบัติอื่น ๆ ที่เป็น Derivatives ของพลังงานออกมา

บางโปรแกรมก็ใช้สูตรที่ง่าย บางโปรแกรมก็สูตรที่สลับซับซ้อน บางโปรแกรมก็คัดลอกหรือลอกไอเดียของโปรแกรมอื่นมาแล้วก็นำมาดัดแปลง ตัดแต่ง พัฒนาให้ดีมากขึ้น (เรียกว่าเป็นแรงบันดาลใจก็ได้) โปรแกรมยอดฮิตอย่าง Gaussian[1] ถือว่าเป็นต้นแบบของโปรแกรมเคมีควอนตัมเกือบทุกโปรแกรมที่เรารู้จักกันใน

[1] https://gaussian.com/

ปัจจุบัน มีข้อดีเยอะ แต่ในขณะเดียวกันก็มีข้อเสียหลายอย่างและมีความสามารถจำกัด ไม่เหมาะสำหรับการคำนวณระบบโมเลกุลบางประเภทหรือการศึกษาคุณสมบัติของวัสดุโมเลกุลในหลาย ๆ สถานะ

สิ่งที่ผมอยากจะเพิ่มเติมก็คืออยากจะให้ผู้อ่านเข้าใจประเด็นหลาย ๆ อย่างที่อาจจะเคยเข้าใจผิด ๆ ดังนี้

- ไม่มีทฤษฎีควอนตัมอันไหนที่ดีและแม่นยำที่สุดสำหรับทุกระบบ (General)
- ไม่มีโปรแกรมควอนตัมโปรแกรมไหนที่ดีที่สุดโลก ทั้งโลกนี้และโลกหน้า
- มีแต่ทฤษฎีและโปรแกรมที่เหมาะสำหรับระบบโมเลกุลแค่บางประเภทที่มันถูกออกแบบมาเพื่อให้คำนวณแล้วได้ผลที่แม่นยำและถูกต้องและมีประสิทธิภาพ
- โปรแกรมเสียเงินใช่ว่าจะดีเสมอไป และโปรแกรมฟรีใช่ว่าจะไม่ดี การเลือกใช้โปรแกรมฟรีก็มีความเสี่ยงหลาย ๆ อย่าง
- ถ้าผู้อ่านอยากใช้โปรแกรมไหนก็ให้ใช้โปรแกรมนั้นเลย แต่ควรจะต้องเข้าใจด้วยว่าโปรแกรมนั้นเหมาะสำหรับคำนวณอะไร

## 4.14 แบบฝึกหัด

1. เขียนโปรแกรมคำนวณ Restricted Hartree-Fock (RHF) ด้วยภาษา Python
2. ใช้โค้ด RHF คำนวณ Ionisation Energy ของ Helium และพลังงานและออร์บิทัลของ Beryllium
3. (เสริม) ปรับแก้โค้ด RHF ให้สามารถคำนวณ Unrestricted Hartree-Fock (UHF) ได้
4. ใช้โค้ด UHF คำนวณ Singlet-Triplet Splitting ของ Helium และ Ionisation Energy ของ Lithium
5. อธิบายและเขียนโปรแกรมสำหรับคำนวณ Density Matrix ของ Density Functional Theory

# บทที่ 5

# การคำนวณวิทยาศาสตร์สมรรถนะสูง

## 5.1 ทำไมต้อง High-Performance Computing ?

High-Performance Computing (HPC)คือเทคโนโลยีที่ใช้คลัสเตอร์คอมพิวเตอร์ (เครือข่ายของคอมพิวเตอร์หลาย ๆ เครื่องที่เชื่อมต่อและทำงานร่วมกัน) ในการประมวลผลหรือคำนวณข้อมูลขนาดใหญ่และแก้ปัญหาทางคณิตศาสตร์ที่ซับซ้อนมาก ๆ ให้เสร็จในระยะเวลาที่เร็วที่สุด โดยทั่วไปแล้ว HPC ทำงานหรือประมวลผลได้เร็วกว่าคอมพิวเตอร์ธรรมดาทั่วไปที่เราใช้ทำงานกันปกติหลายล้านเท่า[1]

### 5.1.1 อัลกอริทึมสำหรับ Parallel Computing

ในปัจจุบันเราสามารถแบ่งอัลกอริทึมสำหรับการคำนวณ Parallel Computing ออกได้เป็น 2 อัลกอริทึม คือ Open Multi-Processing (OpenMP) กับ Message Passing Interface (MPI) ซึ่งทั้งสองวิธีนี้มีความแตกต่างกันดังนี้

**OpenMP** เป็นวิธีที่เหมาะสำหรับการคำนวณกับหน่วยประมวลที่มีความเชื่อมโยงกันแบบแน่น (Tightly Coupled Multiprocessing) เช่น เครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลหลาย ๆ ตัวอยู่ภายในเครื่องเดียวกันและมีการแชร์หน่วยความจำกัน (Shared Memory) โดยจะเป็นการใช้จำนวน Threads หลาย ๆ ตัวมาใช้ในการวนลูปแบบขนานซึ่งเรียกว่า Fine Grain Parallelism (Loop Level) สำหรับการเขียนโค้ดเพื่อให้รันได้แบบขนานโดยการใช้ OpenMP สามารถทำได้ไม่ยาก เมื่อเทียบกับ วิธี MPI

[1]อ่านเพิ่มเติมเกี่ยวกับ HPC ได้ที่ https://www.ibm.com/topics/hpc

**MPI** เป็นวิธีที่ตรงข้ามกับ OpenMP นั่นคือเหมาะสำหรับการคำนวณกับหน่วยประมวลผลที่มีความเชื่อมโยงกันแบบหลวม (Loosely Coupled Multiprocessing) เช่น คลัสเตอร์หรือซุปเปอร์คอมพิวเตอร์ที่ประกอบไปด้วยคอมพิวเตอร์หลาย ๆ เครื่องที่เชื่อมต่อกันผ่านเครือข่ายภายใน (Local Network) วิธี MPI มักถูกนำมาใช้สำหรับการทำ Coarse-Grain Parallelism (Domain Decomposition) การเขียนโค้ดเพื่อให้รันได้แบบขนานโดยการใช้ MPI นั้นค่อนข้างมีความซับซ้อนมากกว่าการใช้วิธี OpenMP ขึ้นอยู่กับความเร็วของโค้ดที่เราต้องการให้เพิ่มขึ้น รวมไปถึงการออกแบบอัลกอริทึมสำหรับการแบ่ง Workload ไปยัง Compute Node ต่าง ๆ และการ Communication ระหว่าง Node ด้วย

### 5.1.2 การเลือกใช้อัลกอริทึม Parallelism

ผมเชื่อว่าตอนนี้ผู้อ่านอาจจะเกิดคำถามว่าแล้วเราจะใช้ OpenMP หรือ MPI ในสถานการณ์ไหน ผมมีตัวอย่างมาให้ดู 3 กรณี จะได้เห็นภาพกันง่าย ๆ ครับ

**ตัวอย่างที่ 1**

> เราต้องการทำ Parallelization เพราะว่าหน่วยความจำของเครื่องที่ใช้รันโค้ดของเรานั้นไม่เพียงพอ เช่น เรามีการคำนวณที่มีความซับซ้อนมาก แล้วก็ขนาดของข้อมูลที่ใช้นั้นมีปริมาณเยอะมากซึ่งไม่สามารถ Fit กับขนาดของหน่วยความจำที่มี

ในกรณีนี้เราควรใช้อัลกอริทึม MPI แล้วก็ควรจะเริ่มด้วยการใช้แค่ 1 MPI Process ต่อ 1 Compute Node ซึ่งจะเป็นการใช้หน่วยความจำให้ได้มากที่สุด

**ตัวอย่างที่ 2**

> เรามีข้อมูลที่มีปริมาณน้อยมากแล้วเราก็ต้องการแค่เพิ่มความเร็วของโค้ดที่กินทรัพยากรสิ้นเปลืองมาก ๆ ในการคำนวณ แล้วก็เราไม่อยากเสียเวลาในการทำ Parallelization

ในกรณีนี้เราควรใช้อัลกอริทึม OpenMP เพราะว่าการเขียนโค้ด OpenMP นั้นทำได้ไม่ยากมาก เพียงแค่เขียนโค้ดเพิ่ม Statement เข้าไปแค่เพียงกี่บรรทัด (ให้ครอบ Loop ที่เราต้องการทำ Parallel)

**ตัวอย่างที่ 3**

> เราต้องการเพิ่มหน่วยความจำเป็น (ใช้ Compute Node หลาย ๆ ตัว) แล้วก็ในขณะเดียวกันเราต้องการเพิ่มความเร็วในการคำนวณให้ได้มากที่สุดเท่าที่จะเป็นไปได้ด้วย เช่น อยากให้โค้ดของเรารันด้วย Core หลาย ๆ Core ต่อ Compute Node

ในกรณีนี้ค่อนข้างซับซ้อนกว่าตัวอย่างสองอันแรกสักหน่อย จากประสบการณ์ของผมพบว่าในกรณีข้างต้นนี้ Hardware มีบทบาทอย่างมากต่อประสิทธิภาพของ Parallelization ถ้าหากว่าเครื่อง Compute Node ของเรามีจำนวน Core ที่ไม่เยอะ (เช่น 4-8 Cores) ประสิทธิภาพของการคำนวณจะขึ้นอยู่กับ OpenMP (นั่นก็เพราะว่าโค้ดจะต้องมีการเริ่มต้น OpenMP Threads ก่อน) มากกว่าที่จะขึ้นกับ MPI (เพราะว่าการ Communication ระหว่าง Processors ที่มีการแชร์หน่วยความจำร่วมกันนั้นไม่จำเป็นต้องใช้ MPI)

แต่ถ้าหากว่าเครื่อง Compute Node ของเรามีจำนวน Cores ที่เยอะ (เช่น 16 Cores ขึ้นไป) ในกรณีนี้เราควรใช้ Hybrid Algorithm ซึ่งเป็นการใช้ MPI และ OpenMP เพื่อมาทำ Parallelization แบบพร้อมกัน แต่ว่าการเขียนโค้ดให้รันได้ด้วย MPI และ OpenMP นั้นมีความยากมาก

## 5.2 ทักษะและเครื่องมือสำหรับการเขียนโปรแกรมสำหรับ HPC

**ทักษะที่จำเป็นในการทำความเข้าใจ HPC**

องค์ประกอบของ HPC

- สถาปัตยกรรม (Architecture)
- การจัดการหน่วยความจำ (Memory Management)
- Kernels, Threading, Multithreading
- Block

อัลกอริทึมสำหรับการเขียนโค้ดแบบขนาน

- Parallel Computing (SPMD)
- Shared Memory: OpenMP
- Distributed Memory: MPI
- Implementations: OpenMPI, Intel MPI, MVAPICH

คอมไพเลอร์ของ Intel

- OpenMP Compiler: `icc`, `ifort`
- MPI Compiler: `mpicc`, `mpiicc` (สำหรับ Intel C Compiler), `mpicxx` (สำหรับ C++), `mpiifort` (สำหรับ Fortran)

**ทักษะสำหรับการเขียนโค้ดสำหรับ GPU**

มีความเข้าใจภาษา CUDA (Operation)

1. สามารถระบุ (Declare) และจัดสรร (Allocate) หน่วยความจำของ Host และ Device ได้

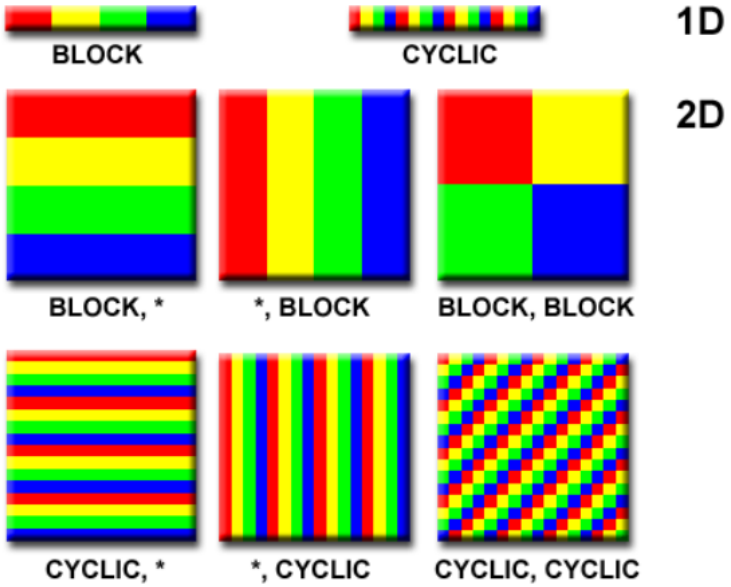
2. สามารถเริ่มต้น (Initialize) การสร้างข้อมูลของ Host ได้
3. ถ่ายโอนข้อมูลจาก Host ไปยัง Device
4. สามารถสั่งให้ Kernels หลาย ๆ ตัวทำงานได้
5. ถ่ายโอนผลการคำนวณจาก Device กลับไปยัง Host

## 5.3 การทำให้เกิดเมทริกซ์รูปทแยง (Matrix Diagonalization)

ในหัวข้อนี้ผู้อ่านจะได้ศึกษาการทำให้เกิดเมทริกซ์รูปทแยงหรือ Matrix Diagonalization (ผมขอเรียกสั้น ๆ ว่า MatDiag เพื่อความสะดวก) ในการคำนวณแบบขนาน (Parallel Computing) สำหรับการคำนวณทางเคมีควอนตัม โดย MatDiag คือการดำเนินการ (Operation) ทางพีชคณิตเชิงเส้นแบบหนึ่งซึ่งถูกใช้อย่างแพร่หลายโดยเฉพาะในงานวิจัยด้านการคำนวณทางวิทยาศาสตร์ แน่นอนว่าโปรแกรมทางเคมีควอนตัมนั้นก็ใช้ MatDiag เยอะมาก ๆ ซึ่งมีความซับซ้อนเชิงการคำนวณอยู่ที่ $O(n^3)$ ทำให้เกิดปัญหาคอขวดและทำให้การคำนวณของระบบที่มีขนาดใหญ่นั้นช้ามาก ๆ เพื่อแก้ปัญหาดังกล่าวจึงได้มีการพัฒนาเทคนิคและไลบรารี่ที่จะเข้ามาช่วยเราในการทำ MatDiag ได้แบบขนาดหรือ Parallel ซึ่งช่วยให้การทำ MatDiag เร็วขึ้นถึง 50% เลยทีเดียว เริ่มต้นเรามีเมทริกซ์ที่มีขนาดใหญ่และสมาชิกส่วนใหญ่ของเมทริกซ์นั้นมีค่าไม่เท่ากับ 0 (Nonzero Elements) ซึ่งเราจะเรียกเมทริกซ์ประเภทนี้ว่าเมทริกซ์แบบเต็ม (Full Matrix) หรือเมทริกซ์แบบแน่น (Dense Matrix) ก็ได้ โดยเราสามารถแทน Dense Matrix ได้โดยใช้รูปแบบที่เรียกว่า Block Cyclic ในการทำการคำนวณแบบขนานด้วยวิธี Message-Passing Interface (MPI) ซึ่งเป็นการกำหนดการกระจาย Dense Matrix ไปยังหน่วยประมวลผล (Processor) แต่ละตัวของเครื่องคำนวณ (Compute Node) ในคลัสเตอร์คอมพิวเตอร์

ให้ดูที่ภาพ 5.1 ก่อนซึ่งเป็นการเปรียบเทียบการกระจายแบบ “Cyclic” และแบบ “Block” สำหรับเวกเตอร์ 1 มิติและเมทริกซ์ 2 มิติ (ภาพด้านล่างผมคัดลอกมาจาก Tutorial “Introduction to Parallel” ของ Blaise Barney แห่งสถาบัน LLNL) โดยสีแต่ละสีของแต่ละช่องนั้นจะเป็นการบ่งบอกถึง Processor ที่ต่างกัน และแต่ละ Segment นั้นจะบ่งบอกถึงสัดส่วน (Portion) ของ Dense Matrix ที่ถูกกำหนดและแบ่งเข้าไปใน Local Memory ของแต่ละ Processor สำหรับการแยกออกเป็นส่วน ๆ แบบหมุนวน (Cyclic Decomposition) ของเมทริกซ์นั้นสามารถทำได้คือเราจะทำการ Distribute แต่ละแถวหรือแต่ละคอลัมน์ไปยัง Processor ที่แตกต่างกัน (เราอาจจะแบ่งเป็นทีละคู่ก็ได้ เช่น แบ่งทุก ๆ 2 แถว) ในทางตรงข้ามนั้น วิธีการแบ่งแบบ Block Representation นั้นจะเป็นการแยกเมทริกซ์ออกเป็นเมทริกซ์ย่อย ๆ จำนวน $N$ เมทริกซ์ (เรียกว่า Submatrices ก็ได้) โดยไม่สนใจว่าขนาดของแต่ละ Block นั้นจะต้องมีขนาดที่เท่ากัน ซึ่งแต่ละ Submatrix นั้นจะถูกส่งต่อไปยัง Processor แต่ละตัว สรุปคือการแบ่งแบบ Cyclic นั้นเป็นการนำการแบ่งแบบ Block มาทำซ้ำ ๆ กันไปแบบละเอียดกว่า ซึ่งจะทำให้เราได้ Block ที่มากกว่า

แล้วข้อดีหรือข้อเสียของทั้งสองวิธีนี้คืออะไร? เราจะเห็นได้ว่า Cyclic Distribution นั้นจะเหมาะกว่าการกระจายเมทริกซ์แบบเท่า ๆ กัน (Evenly) แต่ว่าจะมีการจัดการเมทริกซ์ที่ทำได้แย่กว่าเพราะว่าจะต้องมีการสื่อสาร (Communication) ระหว่าง Processor และระหว่าง Compute Node ในการส่งต่อข้อมูลของ Matrix Element ที่ถูกคำนวณด้วย Processor ที่อยู่ใกล้กันซึ่งในทางตรงกันข้ามนั้นการ Communication




ภาพ 5.1 เปรียบเทียบ Cyclic Format และ Block Format สำหรับการแสดงเวกเตอร์และเมทริกซ์ในการประมวลผลแบบขนาน (Parallel) ที่มีการใช้หน่วยประมวลผลหลายอัน (Multiple Processors) โดยแต่ละสีคือหน่วยประมวลผลที่แตกต่างกัน

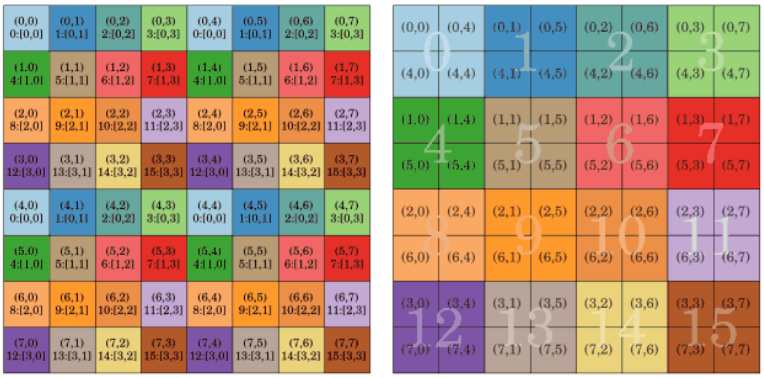
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1, 2 | 1, 3 | 1, 4 | 1, 5 | 1, 6 | 1, 7 | 1, 8 | 1, 9 | 1, 10 |
| 2, 2 | 2, 3 | 2, 4 | 2, 5 | 2, 6 | 2, 7 | 2, 8 | 2, 9 | 2, 10 |
| 3, 2 | 3, 3 | 3, 4 | 3, 5 | 3, 6 | 3, 7 | 3, 8 | 3, 9 | 3, 10 |
| 4, 2 | 4, 3 | 4, 4 | 4, 5 | 4, 6 | 4, 7 | 4, 8 | 4, 9 | 4, 10 |
| 5, 2 | 5, 3 | 5, 4 | 5, 5 | 5, 6 | 5, 7 | 5, 8 | 5, 9 | 5, 10 |
| 6, 2 | 6, 3 | 6, 4 | 6, 5 | 6, 6 | 6, 7 | 6, 8 | 6, 9 | 6, 10 |
| 7, 2 | 7, 3 | 7, 4 | 7, 5 | 7, 6 | 7, 7 | 7, 8 | 7, 9 | 7, 10 |

ภาพ 5.2 Block Cyclic แบบ 2 มิติ: Global View

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1, 2 | 1, 3 | 1, 8 | 1, 9 | 1, 4 | 1, 5 | 1, 10 | 1, 6 | 1, 7 |
| 2, 2 | 2, 3 | 2, 8 | 2, 9 | 2, 4 | 2, 5 | 2, 10 | 2, 6 | 2, 7 |
| 5, 2 | 5, 3 | 5, 8 | 5, 9 | 5, 4 | 5, 5 | 5, 10 | 5, 6 | 5, 7 |
| 6, 2 | 6, 3 | 6, 8 | 6, 9 | 6, 4 | 6, 5 | 6, 10 | 6, 6 | 6, 7 |
| 3, 2 | 3, 3 | 3, 8 | 3, 9 | 3, 4 | 3, 5 | 3, 10 | 3, 6 | 3, 7 |
| 4, 2 | 4, 3 | 4, 8 | 4, 9 | 4, 4 | 4, 5 | 4, 10 | 4, 6 | 4, 7 |
| 7, 2 | 7, 3 | 7, 8 | 7, 9 | 7, 4 | 7, 5 | 7, 10 | 7, 6 | 7, 7 |

ภาพ 5.3 Block Cyclic แบบ 2 มิติ: Local View

ใน Block Representation Matrix นั้นจะทำได้ดีกว่าเพราะว่ามันมีการแบ่งแบบต่อเนื่องบนหน่วยความจำ แต่ว่าถ้าหากว่า Matrix ของเราเป็นแบบ Sparse Matrix หรือเมทริกซ์ที่มีสมาชิกส่วนใหญ่เป็น 0 นั้นก็อาจจะเกิดปัญหาเช่น Load Balancing ได้



ภาพ 5.4 Square Virtual Matrix กับ Block Cyclic Distribution ด้านซ้ายคือ Global View ของการแบ่ง Array ตาม Block ด้านขวาคือ Local View ที่ข้อมูลนั้นถูกเก็บอยู่ในแต่ละหน่วยประมวลผลใน P-Array ส่วนตำแหน่งของ Process (Block Coordinates ในวงเล็บ) นั้นสอดคล้องกับ Linear Rank (Watermark)

เพื่อรวมข้อดีของทั้งสองวิธีไว้จึงได้มีการแบ่งแบบ Block Cyclic Distribution ซึ่งก็คือเป็นการแยกเมทริกซ์ออกเป็น Block เล็ก ๆ แล้วก็ทำการกระจาย Blocks เหล่านี้แบบหมุนวน (Cyclically) ไปยัง Processors ทุกตัว โดยให้ดูตัวอย่างของ Block Cyclic Distribution ตามภาพที่ 5.2 และ 5.3 สำหรับวิธีการแบ่งที่มีประสิทธิภาพที่สุดนั้นก็คือจำนวนของ Block ที่ถูกแบ่งออกมานั้นนั้นจะต้องเท่ากับจำนวนของ Processors (หมายความว่ามีมิติเท่ากัน $N_{\text{row}} \times N_{\text{col}}$) โดย $N$ คือจำนวน Processors ซึ่งตามทั้งสองภาพนั้นเราจะเห็นได้ว่า Block ที่มีสีเหมือนกันนั้นคือถูกคำนวณบน Processor เดียวกัน (ดู Local View) และขนาดของแต่ละ Block ควรจะต้องเท่ากันด้วย ส่วนภาพที่ 5.4 นั้นแถมให้ครับ ซึ่งก็เป็นอีกตัวอย่างของการแยกเมทริกซ์ออกเป็นส่วน ๆ (Decomposition) แบบ Block Cyclic Distribution

## 5.4 การวัดประสิทธิภาพไลบรารี่สำหรับ Matrix Diagonalization

เราจะมาศึกษาการวัดประสิทธิภาพของไลบรารี่ ScaLAPACK กับ ELPA ซึ่งทั้งสองตัวนี้เป็นไลบรารี่สำหรับการทำ MatDiag แบบขนานซึ่งได้รับความนิยมในการนำมาใช้ในการเขียนโปรแกรมที่รันบนซุปเปอร์คอมพิวเตอร์ เช่น โปรแกรมสำหรับการทำงานวิจัยด้านวิทยาศาสตร์ โดยเราได้ศึกษากันไปแล้วว่าถ้าหากเรามี Dense Matrix $A$ ที่ถูกกระจายหรือแบ่งไปคำนวณบน Processors แต่ละตัวด้วยวิธี MPI โดยการใช้ Block Cyclic Distribution สิ่งที่เรามักจะทำกันต่อจากนั้นก็คือการ Diagonalize ซึ่งเราต้องพยายามทำให้มัน Efficient ที่สุดโดยการทำ Diagonalization นั้นคือการหาคำตอบของปัญหาค่าไอเกน (Eigenvalue Problem) ดังต่อไปนี้

$$AX = XL \tag{5.4.1}$$

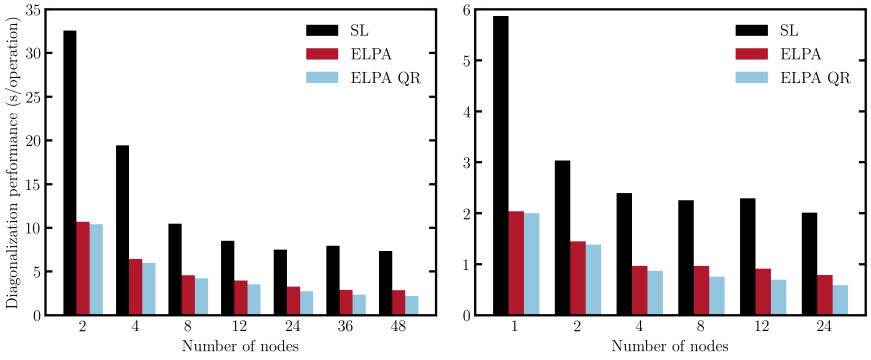
โดยที่ $X$ คือเมทริกซ์ที่บรรจุ Eigenvector ของ $A$ ไว้ ส่วน $L$ นั้นคือเมทริกซ์ที่บรรจุค่า Eigenvalue ซึ่งเมทริกซ์ $L$ นี้เองที่เราจะต้องมาทำการหาเพราะมันเป็น Diagonal เมทริกซ์จริง ๆ ของ $A$

ในไลบรารี่ ScaLAPACK นั้นมีอัลกอริทึม 3 ตัวที่เราสามารถนำมาใช้ในการทำ Diagonalize Matrix ที่เป็น Real Symmetry Matrix (เมทริกซ์ที่มีความสมมาตรและมีเพียงแค่ค่าจริงเป็นสมาชิกเท่านั้น) นั่นคือ `p?syevd`, `p?syevx`, และ `p?syevr` โดยที่ `?` นั้นแทนด้วยค่าที่บ่งบอกถึงประเภทของข้อมูลในเมทริกซ์ เช่น ถ้าแทน `?` ด้วย `d` จะหมายถึงเมทริกซ์นั้นเก็บข้อมูลประเภท Double Precision หรือความเที่ยงแบบสองเท่าซึ่งเป็นประเภทของข้อมูลแบบหนึ่งของ Floating Point หรือจำนวนจุดลอยตัว (ผมแนะนำว่าให้เรียกโดยการใช้คำศัพท์ภาษาอังกฤษไปเลย เพราะว่าคำแปลภาษาไทยนั้นอาจจะเข้าใจได้ยากกว่า) โดยทั้งสามอัลกอริทึมนี้ก็จะใช้วิธีที่แตกต่างกันไป เช่น syevd จะใช้วิธีแบ่งแยกและเอาชนะ (Divide and Conquer) ซึ่งผลการทดสอบที่แสดงด้านล่างนั้นได้มาจากการใช้ `p?syevd` นั่นเองครับ โดยสรุปสั้น ๆ คือวิธี Divide and Conquer นั้นจะเริ่มด้วยการทำการลดรูปเมทริกซ์ (Reduction) $A$ ให้เป็น Tridiagonal Matrix ก่อนโดยใช้การแปลง Householder หลังจากนั้นก็ทำการหา Tridiagonal Eigenvalue ด้วยอัลกอริทึม Divide and Conquer แล้วก็ทำการแปลงย้อนกลับไปให้ได้เป็น Eigenvector ออกมา

ตามที่ผมได้อธิบาย `p?syevd` ของไลบรารี่ ScaLAPACK ไปแล้วนั้น ลำดับต่อไปคือไลบรารี่ยอดฮิตอีกตัวที่ได้รับความนิยมในการนำมาใช้ในโปรแกรมทางเคมีควอนตัมหลายตัวด้วยกัน เช่น NWChem และ CP2K นั่นก็คือไลบรารี่ ELPA จริง ๆ แล้ว ELPA นั้นเอาอัลกอริทึมใน ScaLAPACK มาปรับปรุงอีกทีนึงเพื่อให้มีประสิทธิภาพมากขึ้น โดยจะใช้เทคนิค Direction Transformation ของ Tridiagonal Form ซึ่งก็มีความซับซ้อนพอสมควร เอาเป็นว่าทั้ง ScaLAPACK กับ ELPA ก็สามารถนำมาใช้ได้ทั้งคู่ แล้วก็ Interface นั้นมีความคล้ายคลึงกันมาก สิ่งที่แตกต่างอีกอย่างหนึ่งก็คือ ELPA นั้นมีความ General มากกว่าตรงที่สามารถใช้กับ Fortran Kernel ได้บนหลากหลายสถาปัตยกรรมมากกว่า เช่น ถ้า Kernel ของ CPU เป็นแบบใหม่ ๆ เช่น AVX, AVX2, หรือ AVX-512 นั้น ELPA ก็จะรองรับ คราวนี้เรามาดูการทำ Benchmark หรือการวัดประสิทธิภาพของไลบรารี่ทั้งคู่นี้กัน ปกติแล้วการทำ MatDiag นั้นจะขึ้นอยู่กับปัจจัยหลายตัว โดยหลัก ๆ แล้วมีดังนี้

1. ขนาดของ Matrix
2. โครงสร้างของ Matrix
3. จำนวน Processors ของเครื่องที่ใช้ในการรันหรือคำนวณ Diagonalization
4. ขนาดของ MPI Block Size สำหรับ Matrix
5. อัลกอริทึมที่ใช้ในการทำ Diagonalization

สำหรับตัวอย่างที่เราจะมารันทดสอบ Benchmark กันนั้นก็คือเมทริกซ์จตุรัสขนาด $5888 \times 5888$ กับขนาด $13034 \times 13034$ ตามลำดับ โดยใช้โปรแกรม CP2K (โปรแกรมทางเคมีควอนตัม) ซึ่งจริง ๆ แล้วก็คือระบบที่เป็นโมเลกุลน้ำ (Water Cluster) 128 โมเลกุลนั่นเอง สำหรับเครื่องซุปเปอร์คอมพิวเตอร์ที่ใช้ในการทดสอบนั้นคือ Cray XC40 ซึ่งมีสเปคคือแต่ละโหนดนั้นจะมี 12-core Intel Xeon E5-2690v3 Processors ทั้งหมด 2 ตัว และมี Memory คือ 64 GB (DDR4) สำหรับผลการทดสอบนั้นก็ดูได้ตามภาพที่




ภาพ 5.5 Local View

5.5 ได้เลย แกน y คือประสิทธิภาพที่ได้ส่วนแกน $x$ นั้นคือจำนวนของ Node ของ Cray XC40 โดย SL คือ ScaLAPACK, ส่วน QR นั้นคือเทคนิค Decomposition แบบหนึ่งที่เราเอาเข้ามาช่วยในการเพิ่มประสิทธิภาพการทำ Diagonalization ของ ELPA นั่นเอง ภาพด้านซ้ายคือระบบที่เมทริกซ์ขนาดใหญ่ ส่วนภาพด้านขวาเมทริกซ์ขนาดเล็ก สรุปคือจากการทดสอบนั้นก็คือ ELPA ชนะขาดลอยในการทำ Diagonalization แบบขนานด้วย MPI ซึ่ง ELPA ทำประสิทธิภาพได้ดีกว่า SL ประมาณ 60-80% เลยทีเดียว

## 5.5 การประยุกต์ใช้ Matrix Diagonalization

หนึ่งในการประยุกต์ใช้ Matrix Diagonalization ในโปรแกรมเคมีเชิงคำนวณก็คือการคำนวณพลังงานของออร์บิทัล (Orbital Energies) ซึ่งเป็นเทอมที่สำคัญมาก ๆ ในทางเคมีเพราะว่าเป็นตัวที่เราจะนำมาใช้ในการศึกษาโมเลกุล โดยหลาย ๆ คนที่เคยวิชาเคมีอินทรีย์เชิงฟิสิกส์มานั้นก็น่าจะเคยผ่านการใช้ Hückel Model Theory ในการคำนวณหาพลังงานของออร์บิทัลของโมเลกุลเคมีอินทรีย์ (สารประกอบไฮโดรคาร์บอน) แบบง่าย ๆ กันมาแล้ว เช่น โมเลกุลเบนซีน ซึ่งวิธีที่เราจะใช้ในการคำนวณหา Orbital Energy นั้นเราจะต้องทำการกำหนดฟังก์ชันคลื่นที่ใช้อธิบาย MO สำหรับ $\pi$ Electron ขึ้นมาก่อน ซึ่งเราสามารถใช้ผลรวมเชิงเส้นของ Atomic Orbitals ได้ เช่น สำหรับโมเลกุลเบนซีน เขียนได้ดังนี้

$$\phi_n = \sum_{i=1}^{6} C_i \chi_i \tag{5.5.1}$$

โดยที่ $C_i$ คือ Molecular Orbital Coefficients และ $\chi$ คือ Basis Function คราวนี้เราสามารถใช้สมการ Eigenfunction

$$HC = \epsilon C \tag{5.5.2}$$

ในการหา $\epsilon$ ซึ่งเป็นพลังงานของแต่ละออร์บิทัลได้ โดยที่ H คืออินทิกรัลของ Basis Function โดยหน้าตา H,

C และ $\epsilon$ นั้นจริง ๆ แล้วก็คือ Square Matrix ดี ๆ นี่เอง โดยสามารถดูตัวอย่างของทั้งสามเมทริกซ์นี้สำหรับกรณีโมเลกุลเบนซีนได้ตามภาพที่ 1 โดย $\alpha$ กับ $\beta$ นั้นก็คือ Coefficient ของแต่ AO แต่ละอันนั่นเอง เช่น อะตอมคาร์บอนตัวที่ 1 นั้นก็จะมี Interaction กับคาร์บอนที่ 2 กับ 6 ซึ่งการแก้สมการ Secular Equation นี้เราสามารถทำ Diagonalization ได้ โดยจัดรูปสมการเป็น[1]

$$(H - \epsilon)C = 0 \tag{5.5.3}$$

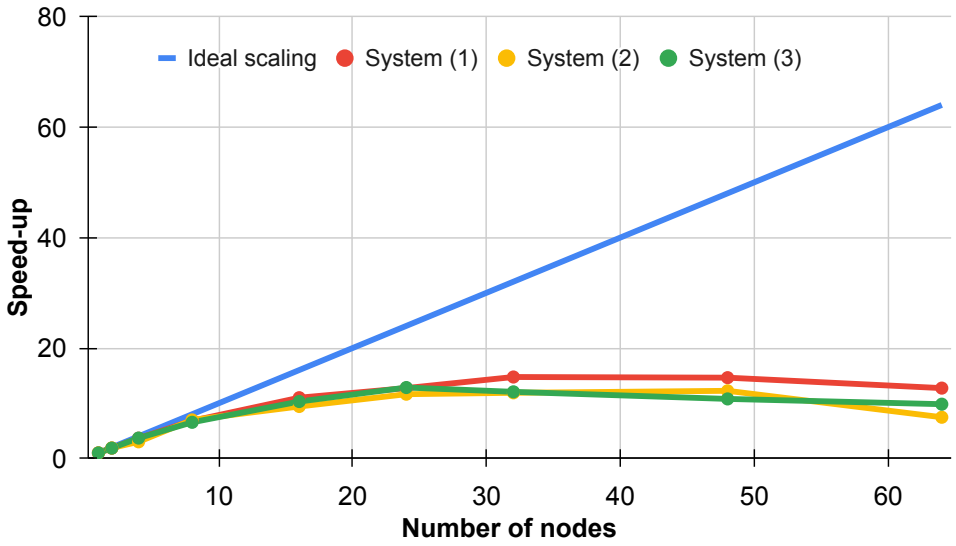
หนึ่งในวิธีที่หลายคนมักจะนำมาใช้ในการทำ Diagonalization นั้นก็คือ Jacobi Method แต่ว่าวิธีนี้มีจุดอ่อนคือมันไม่ได้ทำการแยกตัวประกอบของ Secular Equation ออกเป็นเทอม ๆ จึงทำให้เราไม่มี Main-diagonal Block แล้ว Main-diagonal Block คืออะไร? ทำไมถึงสำคัญ? ประเด็นก็คือการที่เราทำ Diagonalization นั้นมันจะมีวิธีบางอย่างที่สามารถจัดการเมทริกซ์ให้อยู่ในรูปที่เกิดจากการประกอบกันระหว่าง Diagonal Matrix หลาย ๆ อันได้และสมาชิกของเมทริกซ์ที่อยู่นอก Main-diagonal Block นั้นจะต้องเป็น 0 ด้วย เช่นให้ดูตามภาพที่ 2 ถ้าใครยังไม่เข้าใจให้ดูภาพที่ 3 จะได้เห็นภาพของ Diagonal Block ชัดขึ้น ซึ่งกลับมาที่ Jacobi Method ที่ไม่ได้ทำการแยก Block Diagonal Matrix ออกมาให้เรา ซึ่งนี่เป็นสาเหตุที่ทำให้ Symmetry ของโมเลกุลนั้นหายไประหว่างการทำ Diagonalization และทำให้ออร์บิทัลที่เป็นแบบ Degenerate (ออร์บิทัลที่มีพลังงานเท่ากัน) นั้นหายไปด้วย ซึ่งในความเป็นจริงโมเลกุลเบนซีนจะต้องมีบางออร์บิทัลที่มีพลังงานเท่ากัน ตามภาพที่ 4 ดังนั้นสิ่งที่เราต้องการคือ Diagonalization Method ที่สามารถให้ Main-diagonal Block ที่มีให้ Degenerate MOs อย่างไรก็ตามในการคำนวณทางเคมีควอนตัมนั้นถ้าหากเราใช้ทฤษฎีอื่น ๆ นั้นก็จะมีการนิยามและคำนวณหาพลังงาน MO (Eigenvalues) ที่แตกต่างกันไปและซับซ้อนมากขึ้น แต่หลัก ๆ แล้วก็จะต้องมีการทำ Diagonalization อยู่ดีครับ

## 5.6 การวัดประสิทธิภาพโปรแกรม *Ab Initio* Molecular Dynamics

ในหัวข้อนี้เราจะมาดูรายละเอียดขั้นตอนการประเมินประสิทธิภาพ (ความเร็ว) ของโปรแกรม Ab initito Molecular Dynamics (AIMD) กันครับ ในปัจจุบันนั้นมีโปรแกรมที่สามารถรันการคำนวณ AIMD หลายโปรแกรมมาก ๆ โดยโปรแกรมที่ได้รับความนิยม เช่น CPMD, Quantum Espresso, CP2K, NWChem, VASP โดยส่วนตัวของผมเองนั้นก็ได้มีโอกาสวัดประสิทธิภาพ (Benchmarking) ความเร็วโปรแกรมที่กลุ่มวิจัยที่ผมมาศึกษาต่อนั้นใช้ ก็คือโปรแกรม CP2K ว่าทำงานได้เร็วและมี Speed-Up Scaling มากน้อยแค่ไหนบน Distributed System ของ Supercomputer

แน่นอนว่าขั้นตอนเริ่มต้นนั้นก็คือการเตรียมโครงสร้างของระบบโมเลกุลที่ต้องการนำมาใช้ในการทดสอบ แล้วก็รัน Simulation โดยการเปลี่ยนจำนวนของ Compute Nodes โดยเริ่มจาก 1 Node และเพิ่มจำนวนเป็น 2, 4, 8, 16, 24, 32, 48, 64 Node เป็นต้น ซึ่งตามทฤษฎีแล้วนั้นความเร็วของ Software ที่ได้ควรจะต้องเร็วขึ้นตามจำนวนเท่าของการเพิ่มจำนวน Compute Node

[1]ต้องทำความเข้าใจกันก่อนว่าทฤษฎี Hückel Model นั้นจะ Treat หรือสนใจเฉพาะ MO ของ $\pi$ Electron สำหรับโมเลกุล




ภาพ 5.6 Speed-Up Scaling ของโปรแกรม CP2K ตามจำนวน Compute Nodes ของซุปเปอร์คอมพิวเตอร์ CSCS Piz Daint

แต่ในความจริงนั้นโปรแกรมของเราไม่ได้ทำงานเร็วตามทฤษฎีและ Speed-Up Scaling ก็ไม่ได้เป็น Linear Scaling หรอกครับ มันมีปัญหาเยอะและมีหลายเหตุผลที่ทำให้เกิดคอขวด (Bottleneck) โดย Scaling ที่ได้ก็ตามกราฟในรูปที่ 5.6 จะเห็นว่าพอเราเพิ่มจำนวน Compute Node จาก 24 → 32 แล้ว Efficiency (คำนวณจาก Speedup หารด้วยจำนวน Node) เริ่มลดลง

คำถามคือสิ่งที่ผมทำนี้มันถูกต้อง 100% ไหม จริง ๆ แล้วไม่ถูก 100% เพราะว่าการที่เรารัน MD Simulation หลาย ๆ ครั้งถึงแม้ว่าจะใช้ Input ไฟล์เดียวกันและเหมือนกันทั้งหมดนั้น สิ่งที่เกิดขึ้นคือผลการคำนวณที่ได้จะไม่มีทางที่เหมือนกัน นั่นก็เพราะว่าอัลกอริทึมของโปรแกรม MD (เรียกได้ว่าเกือบทุกโปรแกรมเลย) จะมีการสุ่ม (Random) พารามิเตอร์บางตัวขึ้นมาก่อนใน Step แรกสุดของการคำนวณ ซึ่งพารามิเตอร์นั้นก็คือความเร็วของอะตอมแต่ละตัวในโมเลกุลซึ่งจะถูกนำมาใช้ในการแก้สมการ Newton แบบคลาสสิคสำหรับการคำนวณหาแรง (Force) เพื่อใช้ในการอัพเดทตำแหน่งของอะตอมแต่ละตัวใน Step ต่อไป ซึ่งเราเรียกเท่ ๆ ว่าการ Propagation (ใน CP2K เราใช้ DFT-MD ซึ่งจะใช้ควอนตัมในการคำนวณพลังงานและแรงของอะตอมแต่ละตัว ซึ่งเรามักจะ “เคลม” และเชื่อว่าให้ผลที่แม่นยำและถูกต้องกว่าใช้ Force Field)

ถึงแม้ว่าเราจะรัน Simulation ด้วยไฟล์ Input เดียวกัน 2 รอบ ผลการคำนวณนั้นจะไม่เหมือนกัน เช่น พลังงานของระบบในแต่ละ MD Step นั้นถ้าเทียบกันแล้วจะแตกต่างกันอย่างสิ้นเชิง และแน่นอนว่าระยะเวลาจริงที่ใช้ในการคำนวณ (Simulation Time) ของแต่ละ MD Step นั้นก็ต่างกันด้วย เหตุผลก็ตามที่ได้บอกไปคือ Initial Parameter นั้นไม่เหมือนกัน จึงทำให้การรันสองทั้งนั้นให้ผลที่ไม่เหมือนกัน

ถ้าถามว่าผิดไหมที่โปรแกรม MD ส่วนใหญ่นั้นทำการสุ่มค่าความเร็วเริ่มต้นของอะตอมแต่ละตัว คำตอบคือไม่ผิด เพราะว่าท้ายที่สุดแล้วเราสนใจระบบกรณีที่มีความเป็น Ergodicity แบบสมบูรณ์แล้ว เมื่อเรา

ที่เป็นแบบ $\pi$-conjugated เท่านั้น

รัน Simulation ไปเรื่อย ๆ จนระบบเข้าสู่สภาวะสมดุล (Equilibrium) เราสามารถอ้างได้ว่า Configuration แต่ละตัวนั้นสามารถที่จะ Represent คุณสมบัติแบบ Microscopic ได้

```
DO i = 1, natoms
  atomic_kind => part(i)%atomic_kind
  CALL get_atomic_kind(atomic_kind=atomic_kind, mass=mass)
  part(i)%v(1) = 0.0_dp
  part(i)%v(2) = 0.0_dp
  part(i)%v(3) = 0.0_dp
  IF (mass .NE. 0.0) THEN
     SELECT CASE (is_fixed(i))
     CASE (use_perd_x)
        part(i)%v(2) = globenv%gaussian_rng_stream%next()/SQRT(mass)
        part(i)%v(3) = globenv%gaussian_rng_stream%next()/SQRT(mass)
     CASE (use_perd_y)
        part(i)%v(1) = globenv%gaussian_rng_stream%next()/SQRT(mass)
        part(i)%v(3) = globenv%gaussian_rng_stream%next()/SQRT(mass)
     CASE (use_perd_z)
        part(i)%v(1) = globenv%gaussian_rng_stream%next()/SQRT(mass)
        part(i)%v(2) = globenv%gaussian_rng_stream%next()/SQRT(mass)
     CASE (use_perd_xy)
        part(i)%v(3) = globenv%gaussian_rng_stream%next()/SQRT(mass)
     CASE (use_perd_xz)
        part(i)%v(2) = globenv%gaussian_rng_stream%next()/SQRT(mass)
     CASE (use_perd_yz)
        part(i)%v(1) = globenv%gaussian_rng_stream%next()/SQRT(mass)
     CASE (use_perd_none)
        part(i)%v(1) = globenv%gaussian_rng_stream%next()/SQRT(mass)
        part(i)%v(2) = globenv%gaussian_rng_stream%next()/SQRT(mass)
        part(i)%v(3) = globenv%gaussian_rng_stream%next()/SQRT(mass)
     END SELECT
  END IF
END DO
```

โค้ดด้านบนคือโค้ดของ CP2K ที่ทำการกำหนด (Assign) ความเร็วให้อะตอมแต่ละตัวโดยใช้ Maxwell-Boltzmann Distribution (เรียกอีกชื่อว่า Maxwellian Distribution) ซึ่งมีเบื้องหลังคือถูก Derived มาจาก Gaussian Distribution โดยที่มีการกำหนดค่า Random สำหรับ Variance จาก 0 ไปถึง 1

```
globenv%gaussian_rng_stream = rng_stream_type( &
                              name="Global Gaussian random numbers", &
                              distribution_type=GAUSSIAN, &
                              seed=initial_seed, &
                              extended_precision=.TRUE.)
```

การกำหนดความเร็วของอะตอมแบบสุ่มนั้นจะต้องมีการกำหนด Seed สำหรับการ Random ด้วยโดยดูได้ตามโค้ดด้านบน

```
SUBROUTINE normalize_velocities(simpar, part, force_env, md_env, is_fixed)
  TYPE(simpar_type), POINTER                         :: simpar
```

```
3    TYPE(particle_type), DIMENSION(:), POINTER          :: part
4    TYPE(force_env_type), POINTER                       :: force_env
5    TYPE(md_environment_type), POINTER                  :: md_env
6    INTEGER, DIMENSION(:), INTENT(INOUT)                :: is_fixed
7
8    REAL(KIND=dp)                                       :: ekin
9    REAL(KIND=dp), DIMENSION(3)                         :: rcom, vang, vcom
10   TYPE(cell_type), POINTER                            :: cell
11
12   NULLIFY (cell)
13
14   ! Subtract the vcom
15   CALL compute_vcom(part, is_fixed, vcom)
16   CALL subtract_vcom(part, is_fixed, vcom)
17   ! If requested and the system is not periodic, subtract the angular
       velocity
18   CALL force_env_get(force_env, cell=cell)
19   IF (SUM(cell%perd(1:3)) == 0 .AND. simpar%angvel_zero) THEN
20     CALL compute_rcom(part, is_fixed, rcom)
21     CALL compute_vang(part, is_fixed, rcom, vang)
22     CALL subtract_vang(part, is_fixed, rcom, vang)
23   END IF
24   ! Rescale the velocities
25   IF (simpar%do_thermal_region) THEN
26     CALL rescale_vel_region(part, md_env, simpar)
27   ELSE
28     ekin = compute_ekin(part)
29     CALL rescale_vel(part, simpar, ekin)
30   END IF
31 END SUBROUTINE normalize_velocities
```

เมื่อเรากำหนดหรือคำนวณความเร็วของอะตอมได้แล้ว เรามีวิธีการปรับให้ความเร็วนั้นสอดคล้องกับอุณหภูมิของระบบที่เราต้องการโดยดูได้ตามโค้ดด้านบน (ดูโค้ดของโปรแกรม CP2K ในพาร์ทที่เป็น MD Motion ได้ที่ https://github.com/cp2k/cp2k/tree/master/src/motion)

สรุป ถ้าหากเราอยากจะแก้ปัญหาเรื่องการไม่เท่ากันของความเร็ว เราสามารถทำได้ 2 วิธี (จริง ๆ มีวิธีอื่นอีก) คือ

1. รัน MD Simulation ก่อน 1 ครั้ง แล้วนำความเร็วสุดท้ายที่ได้มาใช้เป็นความเร็วเริ่มต้นสำหรับการทำ Benchmark
2. กำหนด Seed ในการสุ่มของการสร้าง Uniformly Distributed Random Number เพื่อที่ว่าเราจะได้ Gaussian Distribution ที่เหมือนกันทุกประการ และได้ Maxwellian Velocity ที่เหมือนกันด้วย

ถ้าหากผู้อ่านอยากศึกษาเพิ่มเติม ผมแนะนำหนังสือที่น่าสนใจตามนี้ครับ

- Understanding Molecular Simulation. Berend Smit & Daan Frenkel
- Computer Simulation of Liquids. Michael P. Allen & Dominic J. Tildesley หรืออ่านวิกิพี

เดียที่ https://en.wikipedia.org/wiki/Maxwell%E2%80%93Boltzmann_distribution#Distribution_for_the_velocity_vector

## 5.7 เทคนิคการเขียนโปรแกรม *Ab Initio* Molecular Dynamics

### 5.7.1 การทำซ้ำผลการคำนวณ

ถ้าผมอยากจะทำซ้ำ (Reproduce) ผลการคำนวณของ Molecular Dynamics (MD) ให้ได้เหมือนเดิมแบบเป๊ะ ๆ สามารถทำได้แต่ว่าทำได้ยากมาก เหตุผลก็คือมีพารามิเตอร์ (Parameter) หรือปัจจัย (Factor) ต่าง ๆ มากมายที่เราจะต้องพิจารณาแล้วก็ควบคุมให้เหมือนกันและเท่ากันเสมอ ในหัวข้อนี้ผู้อ่านจะได้ศึกษาปัจจัยต่าง ๆ ที่ส่งผลต่อความคลาดเคลื่อนของการคำนวณ MD แบบคร่าว ๆ กันครับ

Molecular Dynamics คือการจำลองทางคอมพิวเตอร์ของระบบโมเลกุลเพื่อศึกษาพฤติกรรมเชิงจลนศาสตร์ของโมเลกุล ถ้าหากว่าเรามี Trajectory ของการคำนวณระบบโมเลกุลน้ำด้วยวิธี *ab initio* Molecular Dynamics[1] แล้วเราต้องการที่จะ Reproduce หรือทำซ้ำผลการคำนวณเพื่อให้ได้ Trajectory ที่เหมือนกันแบบ 100 % (Configuration ของโมเลกุลทุก ๆ Configuration เหมือนกันหมด ละแต่ละ Configuration มีพิกัดของอะตอมหรือ Coordinates ที่เหมือนกัน) ก็สามารถทำได้ แต่ทำได้ยากมาก เหตุผลก็เพราะว่าในการคำนวณ MD นั้น มันมี Error หรือ Noise (ความคลาดเคลื่อนหรือสิ่งรบกวน) ที่เกิดขึ้นนั้นมาจากทั้งฮาร์ดแวร์ (Hardware) ซึ่งก็คือเครื่องคอมพิวเตอร์ที่เราใช้ในการรัน MD และซอฟต์แวร์ (Software) หรือโปรแกรม MD ที่เราใช้ ดังนั้นถ้าหากว่าเราสามารถควบคุมพารามิเตอร์ต่าง ๆ เหล่านี้ได้ เราก็จะสามารถเขียนโปรแกรม MD ที่สามารถนำมาใช้ในการรันการคำนวณที่ให้ผลเหมือนกันได้ทุกครั้ง

สมมติว่าผมมีเพื่อนที่ทำงานวิจัยโดยใช้วิธี MD ในการคำนวณโมเลกุลอะไรก็ได้สักโมเลกุลนึง แล้วถ้าหากว่าผมอยากที่จะเขียนโปรแกรม MD แล้วก็รันเพื่อให้ได้ผลการคำนวณออกมาเหมือนกันเป๊ะ ๆ สิ่งที่ผมต้องจะต้องนึกถึงมีดังต่อไปนี้

1. กำหนดตัวแปรโดยใช้ Fixed Point Number (ห้ามใช้ Floating Point) (เพราะว่ามีความ Deterministic)
2. ต้องไม่ใช้ Thermostat หรือ Barostat เลย
3. รันโค้ดด้วย CPU แค่ 1 Core เท่านั้น (ถ้าอยากจะรันด้วย Multi-CPU Cores ก็ต้องกำหนด Processor Affinity)
4. Seed สำหรับ PRNG จะต้องเท่ากันเสมอ
5. ใช้ Compiler ตัวเดียวกันและเวอร์ชั่นเดียวกันในการคอมไพล์โค้ด
6. ใช้ Operating System ที่เหมือนกัน
    - ใช้ Library เหมือนกัน

[1] https://www.youtube.com/watch?v=BeUCOsGC_eM

- ใช้ Kernel เดียวกัน
- ใช้ Bit เหมือนกัน (32 หรือ 64 bit)

7. ใช้โมเดล CPU เดียวกัน แม้แต่ผู้ผลิตก็ต้องเป็นค่ายเดียวกัน

## 5.8 การเขียนโปรแกรมเคมีควอนตัมบน GPU

### 5.8.1 GPU เข้ามามีบทบาทต่อเคมีควอนตัมได้อย่างไร

เป็นระยะเวลามากกว่า 10 ปีแล้วที่ Graphical Processor Units (GPUs) นั้นเข้ามามีบทบาทในชีวิตประจำวันของเรา โดยเฉพาะในวงการไอที เทคโนโลยี เกมส์ กราฟฟิค รวมไปถึงวิทยาศาสตร์ด้วย โดย GPU นั้นเป็นหนึ่งในผลผลิตจากการพัฒนาอุปกรณ์คอมพิวเตอร์เพื่อนำมาช่วยในการประมวลผลด้านกราฟฟิคโดยเฉพาะการนำไปใช้เป็นเครื่องมือในการเพิ่มความเร็วในการสร้างและประมวลข้อมูลที่เป็นรูปภาพในปริมาณมาก ๆ ตัวอย่างเช่น มีการนำ GPU จากค่าย NVIDIA ซึ่งก็คือ NVIDIA DGX A100 มาใช้งานร่วมกับ CPU จากค่าย AMD นั่นก็คือ AMD Rome 7742 CPU โดยใช้ GPU A100 ทั้งหมด 8 อันและใช้ CPU ทั้งหมด 2 อัน ซึ่งทำให้สามารถประมวลผลแบบ Double Precision ได้โดยมี FLOP Rate อยู่ที่ 78 TFLOPS

ในวงการวิจัยเคมีควอนตัมนั้นก็มีการนำ GPU เข้ามาเพื่อช่วยในการเพิ่มความเร็วในการคำนวณทางควอนตัม เราสามารถใช้ GPU ในการคำนวณ Semi-Empirical, Hartree-Fock (HF), Density Functional Theory (DFT), Post-HF, Energy Gradient Calculations เป็นต้น ถ้าหากว่าเราไล่ดูตามไทม์ไลน์ของการพัฒนาวิธีการทางโครงสร้างเชิงอิเล็กโทรนิกส์เพื่อให้สามารถไปรันได้บน GPU นั้น บทความงานวิจัยแรกสุดเลยที่มีการศึกษาการคำนวณ Electron Repulsion Integral สำหรับออร์บิทัล s กับออร์บิทัล p นั้นคือเมื่อปี 2008 โดย Koji Yasuda ซึ่งได้เสนออัลกอริทึมสำหรับการคำนวณ Coulomb Integral สำหรับการคำนวณ *ab initio* DFT บน NVIDIA GeForce 8800 GTX โดยใช้โปรแกรม Gaussian 03 และโมเลกุลที่ใช้ในการทดสอบก็คือ Taxol กับ Valinomycin[31] แล้วหลังจากนั้นก็มีงานวิจัยที่พัฒนาต่อยอดที่เกี่ยวกับการ GPU ตามมาอีกเยอะมาก[32] สำหรับลำดับเหตุการณ์ในงานวิจัยที่นำ GPU เข้ามาใช้นั้นมีดังนี้

- Yasuda เสนออัลกอริทึมสำหรับการคำนวณ ERI สำหรับออร์บิทัล s และ p[31]
- Yasuda และ Maruoka เสนอวิธีคำนวณ ERI สำหรับ Angular Moment Basis Function แบบอันดับสูง (High-Order)[33]
- Ufimtsev และ Martínez ได้ศึกษาวิธีที่แตกต่างกัน 3 วิธีที่ช่วยในการจับคู่ (Mapping) Integrals ให้เข้ากันกับ GPU Threads[34] ในการคำนวณ Direct Self-Consistent Field (Direct SCF)[35]
- Titov และทีมวิจัย ใช้ระบบการคำนวณคอมพิวเตอร์แบบพีชคณิต (Computer Algebra System) ในการสร้าง Kernels สำหรับการคำนวณ ERI ที่รวมออร์บิทัล d เข้าไปได้ด้วย[36]
- Leuhr และทีมวิจัย ได้เสนอ Dynamic Precision Scheme ในการปรับและควบคุมค่าความถูกต้อง

ในการประมวลผลแบบ Single Precision Operation ของ GPU[37]

- Johnson และทีมวิจัย ได้ทำการ Implement การคำนวณ ERI แบบทั้ง Multi-Node และ Multi-GPU ซึ่งทำให้สามารถคำนวณ Integral ได้บน GPU หลาย ๆ ตัวหรือบน Compute Node หลาย ๆ เครื่อง[38]

- Asadchev และทีมวิจัย ได้ Implement Uncontracted Rys Quadrature Algorithm สำหรับการคำนวณ ERI บน GPU ซึ่งสามารถรวมออร์บิทัล g เข้าไปได้ด้วย[39]

- Asadchev และ Gordon ได้เสนอวิธีสำหรับการคำนวณ HF ด้วย CPU และ GPU แบบ Hybrid Multi-Thread[40]

- Barca และทีมวิจัย ได้ออกแบบอัลกอริทึมแบบใหม่สำหรับการสร้าง Fock Matrix บน GPU[41] และปรับปรุงการเพิ่มประสิทธิภาพในการ Digestion ของ ERI[42]

- Kussmann และ Ochsenfeld พัฒนาอัลกอริทึมสำหรับการคำนวณ Linear-Scaling Matrix Evaluation โดยการใช้เทคนิค Schwarz Integral Estimates[43]

- Miao และ Merz ได้พัฒนาวิธีที่สามารถเพิ่มความเร็วในการคำนวณ ERI ซึ่งรวมออร์บิทัล s, p, d[44] และ g เข้าไปได้ด้วยโดยใช้เทคนิค Recursive Relation[45]

- Rák และ Cserey พัฒนาอัลกอริทึม BRUSH สำหรับคำนวณ ERI บน GPU[46]

- Tornai และทีมวิจัย ปรับปรุงอัลกอริทึม BRUSH โดยทดสอบกับ Compilers เพื่อให้สามารถคำนวณ ERI ที่รวมออร์บิทัล g เข้าไปได้[47]

- Fernandes และทีมวิจัย ได้พัฒนาไลบรารี่ Quantum Supercharge[48]

- Tian และทีมวิจัย ได้ทำการปรับปรุงการคำนวณ ERI บน GPU[49]

ยิ่งไปกว่านั้น ในช่วงเดือนพฤษภาคมของปี ค.ศ. 2022 ทางสถาบันวิจัย Oak Ridge Leadership Computing Facility ในรัฐ Tennessee ประเทศสหรัฐอเมริกาก็ได้เปิดตัวซุปเปอร์คอมพิวเตอร์สมรรถนะสูงที่ชื่อ Frontier[1] ซึ่งเป็นซุปเปอร์คอมพิวเตอร์ที่เร็วที่สุดในโลกและมีความเร็วในระดับ Exascale เครื่องแรกของโลก โดยมี Speed สูงสุดคือ 1.194 exaFLOPS (Rmax) และ 1.67982 exaFLOPS (Rpeak) โดย Rmax คือความเร็วสูงสุดที่วัดได้และ Rpeak คือความเร็วสูงสุดที่เป็นไปได้ทางทฤษฎี[2]

[1]มีชื่อเต็มคือ Hewlett Packard Enterprise Frontier หรือ OLCF-5

[2]ข้อมูลจากเว็บไซต์ https://www.top500.org/lists/top500/2023/06/

### 5.8.2 อัลกอริทึมสำหรับการคำนวณ Self-Consisten Field แบบผสมบน CPU และ GPU

ในหัวข้อนี้ผู้อ่านจะได้ศึกษาความก้าวหน้าเกี่ยวกับงานวิจัยที่พัฒนาวิธีในการคำนวณ Self-Consistent Field (SCF) เพื่อให้สามารถรันได้บนทั้ง CPU และ GPU แบบพร้อม ๆ กัน ซึ่งช่วยลดระยะเวลาในการคำนวณของวิธี HF ไปได้เยอะมาก ๆ ซึ่งการที่เราสามารถคำนวณ HF ได้เร็วขนาดนั้น ก็ทำให้การคำนวณวิธีอื่น ๆ ที่ใช้ HF เป็น Single Reference นั้นเร็วขึ้นตามไปด้วย เช่น วิธี Post-HF ต่าง ๆ

เริ่มต้นเลยต้องเข้าใจก่อนว่าในการคำนวณทางเคมีควอนตัมที่ต้องมีการใช้วิธี HF เพื่อมาสร้าง Single Wavefunction Reference นั้นจะมีส่วนที่สิ้นเปลืองการคำนวณมากที่สุดอยู่ด้วยกัน 2 ส่วน นั่นคือ

1. การคำนวณอินทิกรัลของการผลักระหว่างอิเล็กตรอน (Electron Repulsion Integral หรือ ERI)

2. การสร้าง Fock Matrix ขึ้นมาจาก ERI และ Density Matrix

ในการคำนวณ HF นั้น เริ่มต้นเราจะต้องสร้าง Fock Matrix ขึ้นมาก่อน ซึ่งเมทริกซ์อันนี้เป็นแบบ 2 มิติ จำนวนสมาชิกของเมทริกซ์เท่ากับจำนวน Basis Functions คูณกัน โดย Fock Matrix คำนวณได้จากการนำ Hamiltonian Matrix มารวมกันกับ Density Matrix คูณกับ ERI โดยสมการของการสร้าง Fock Matrix $(F_{\mu v})$ มีดังนี้

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\lambda\sigma} D_{\lambda\sigma} \left[ (\mu\nu \mid \lambda\sigma) - \frac{1}{2}(\mu\lambda \mid \nu\sigma) \right] \tag{5.8.1}$$

โดยที่ $H_{\mu v}$ และ $D_{\mu v}$ นั้นคือสมาชิกของ Core Hamiltonian Matrix และ Density Matrix ตามลำดับ แล้วก็ $(\mu v \mid \lambda\sigma)$ นั้นคือ Notation ที่เราใช้เพื่อแทนอินทิกรัล ERI ซึ่งมีสมการดังต่อไป

$$(\mu\nu \mid \lambda\sigma) = \iint \varphi_\mu(\boldsymbol{r}_1)\, \varphi_\nu(\boldsymbol{r}_1)\, \frac{1}{r_{12}} \varphi_\lambda(\boldsymbol{r}_2)\, \varphi_\sigma(\boldsymbol{r}_2)\, \mathrm{d}\boldsymbol{r}_1\, \mathrm{d}\boldsymbol{r}_2 \tag{5.8.2}$$

โดยเราใช้เบสิสเซทที่เป็นแบบ Contracted Gaussian Basis Functions $\varphi(r)$ ซึ่งมีเลขดัชนี $\mu, v, \lambda,$ และ $\sigma$ ที่แสดงคือ Basis Function แต่ละอัน โดยทั่วไปแล้ว ในบริบทของการคำนวณ ERI นั้นเราสามารถมองหรือตีความ $(\mu v \mid \text{and} \mid \lambda\sigma)$ ว่าเป็น bra กับ ket ก็ได้

สำหรับ Gaussian Basis Function นั้นเรามีสมการดังต่อไปนี้

$$\varphi(\boldsymbol{r}) = \sum_{k=1}^{K} C_k \phi_k(\boldsymbol{a}, \boldsymbol{r}, \boldsymbol{A}, \alpha_k) \tag{5.8.3}$$

และ

$$\begin{aligned}\phi_k\left(\boldsymbol{a},\boldsymbol{r},\boldsymbol{A},\alpha_k\right) =& N_k\left(x-A_x\right)^{a_x}\left(y-A_y\right)^{a_y}\left(z-A_z\right)^{a_z} \\ & \times \exp\left(-\alpha_k|\boldsymbol{r}-\boldsymbol{A}|^2\right)\end{aligned} \tag{5.8.4}$$

ซึ่งสมการด้านบนนั้นจริง ๆ แล้วก็เป็นแค่การเขียนให้ Contracted Gaussian Functions $\varphi(r)$ นั้นอยู่ในรูปของผลรวมเชิงเส้นของ Primitive Gaussian Functions $\phi_k\left(\boldsymbol{a},\boldsymbol{r},\boldsymbol{A}\right.$, และ $\left.\alpha_k\right)$ นั่นเอง ซึ่ง Primitive Gaussian Functions ทั้งหมดนั้นจะถูกกำหนดให้มีตำแหน่งจุดศูนย์กลางอยู่บนอะตอมที่ $\boldsymbol{A}=(A_x,A_y,A_z)$ พร้อมกับมีเลข Orbital Exponent $\alpha_k$ และโมเมนตัมเชิงมุม (Angular Momentum) คือ $\boldsymbol{a}=(a_x,a_y,a_z)$ ส่วน $K$ กับ $C_k$ นั้นจะเป็นตัวที่บ่งบอกถึงอันดับและสัมประสิทธิ์ของการ Contraction ส่วนเลข Angular Momentum ของ Basis Function $\varphi(r)$ นั้นมีนิยามคือ $I_a=a_x+a_y+a_z$

ในส่วนของ Contracted ERI นั้นสามารถเขียนได้ง่ายกว่านี้โดยการใช้ Summation โดยลูปตามจำนวนของ Primitive Gaussian Functions ซึ่งเราสามารถ Represented แทนได้ด้วย Notation ง่าย ๆ คือ $[ab\mid cd]$

$$(\mu\nu\mid\lambda\sigma)=\sum_{i=1}^{K_a}\sum_{j=1}^{K_b}\sum_{k=1}^{K_c}\sum_{l=1}^{K_d}C_{ai}C_{bj}C_{ck}C_{dl}[ab\mid cd] \tag{5.8.5}$$

และ

$$[ab\mid cd]=\iint\phi_a\left(\boldsymbol{r}_1\right)\phi_b\left(\boldsymbol{r}_1\right)\frac{1}{r_{12}}\phi_c\left(\boldsymbol{r}_2\right)\phi_d\left(\boldsymbol{r}_2\right)\mathrm{d}\boldsymbol{r}_1\,\mathrm{d}\boldsymbol{r}_2 \tag{5.8.6}$$

เนื่องจากว่าการคำนวณ Direct SCF นั้นจะเกี่ยวข้องกับการคำนวณระหว่าง Basis Functions หลายพัน Functions จึงทำให้การคำนวณ ERI นั้นเป็นส่วนที่กินเวลานานที่สุดของกระบวนนานทั้งหมด

เนื่องจากว่า ERI นั้นเป็นอินทิกรัล เราจึงสามารถใช้คุณสมบัติความสมมาตรเชิงการสลับที่ของอินทิกรัลเพื่อช่วยลดความสิ้นเปลืองในการคำนวณได้สำหรับการคำนวณแบบขนานบน CPU ดังนี้

$$\begin{aligned}(\mu\nu\mid\lambda\sigma)=(\mu\nu\mid\sigma\lambda)=(\nu\mu\mid\lambda\sigma)=(\nu\mu\mid\sigma\lambda) \\ =(\lambda\sigma\mid\mu\nu)=(\sigma\lambda\mid\mu\nu)=(\lambda\sigma\mid\nu\mu)=(\sigma\lambda\mid\nu\mu)\end{aligned} \tag{5.8.7}$$

ซึ่งคุณสมบัติดังกล่าวนี้มีชื่อเรียกว่า Eight-Fold Integral Permutational Symmetry ซึ่งจะทำให้เรามี Fock Matrix ที่แตกต่างกันทั้งหมด 6 อัน ดังนี้

$$
\begin{aligned}
F_{\mu\nu} &= F_{\mu\nu} + 4D_{\lambda\sigma}(\mu\nu \mid \lambda\sigma) \\
F_{\lambda\sigma} &= F_{\lambda\sigma} + 4D_{\mu\nu}(\mu\nu \mid \lambda\sigma) \\
F_{\mu\lambda} &= F_{\mu\lambda} - D_{\nu\sigma}(\mu\nu \mid \lambda\sigma) \\
F_{\nu\sigma} &= F_{\nu\sigma} - D_{\mu\lambda}(\mu\nu \mid \lambda\sigma) \\
F_{\mu\sigma} &= F_{\mu\sigma} - D_{\nu\lambda}(\mu\nu \mid \lambda\sigma) \\
F_{\nu\lambda} &= F_{\nu\lambda} - D_{\mu\sigma}(\mu\nu \mid \lambda\sigma)
\end{aligned}
\tag{5.8.8}
$$

ถึงแม้ว่าในปัจจุบันเราจะสามารถใช้เทคนิค Eight-Fold Symmetry มาช่วยในการคำนวณ Fock Matrix บน GPU ได้แล้ว แต่ว่าเรามีปัญหา 2 อย่างที่ยังต้องแก้ให้ได้คือ

1. ยังมี Conflict ที่เกิดขึ้นระหว่างการ Memory Access อยู่ในขั้นตอนที่เราต้องทำการรวม Fock Matrix

2. เราไม่สามารถเข้าถึง Global Memory เราในขณะที่เรากำลังเขียนหรืออ่านตัว Density Matrix กับ Fock Matrix ที่อยู่ใน Storage เดียวกัน

สำหรับปัญหาอันแรกที่เกี่ยวกับ Memory Access นั้น ได้มีงานวิจัยของ Ufimtsev และ Martínez ที่เสนอให้คำนวณ Coulomb Matrix กับ Exchange Matrix แบบแยกกันบน GPU โดยการป้องกันปัญหา Memory Access ในการคำนวณ Coulomb Matrix นั้น เราสามารถใช้ Thread Block แต่ละอันเพื่อคำนวณ Single Element ของ Primitive Coulomb Matrix ได้ นอกจากนี้แล้วเราจะไม่ใช้ Integral Symmetry ที่เกิดระหว่าง bra กับ ket แต่ว่าเราจะใช้เฉพาะ Symmetry ภายใน bra หรือ ket แยกกันเท่านั้น ซึ่งทำให้เกิดการลดรูปจาก Eight-Fold Integral Symmetry เหลือเป็น Four-Fold Integral Symmetry ในการคำนวณ Coulomb Matrix ส่วนการคำนวณ Exchange Matrix นั้นก็ทำคล้าย ๆ กัน แต่ว่าเราจะใช้ Two-Fold Symmetry ระหว่าง bra กับ ket แทนเพื่อป้องกันปัญหาการสื่อสารระหว่าง Block (Inter-Block Communication)

ที่กล่าวมาด้านบนนั้นเป็น Algorithm ที่เราสามารถ Implement ลงไปบน CPU ได้โดยการแบ่ง Task ออกเป็น Batch ย่อย ๆ คราวนี้เราลองมาดู Algorithm สำหรับกรณีของ GPU ซึ่งจะแบ่ง Task ออกเป็นตามจำนวน Kernel แทน ตัวอย่างเช่น การคำนวณ Integrals $(sp \mid sd)$ บน CPU นั้นจะสอดคล้องกับการคำนวณ Coulomb Integral $(sp \mid sd)$ และ $(sd \mid sp)$ แล้วก็สอดคล้องกับการคำนวณ Exchange Integrals $(sp \mid sd)$, $(sp \mid ds)$, $(ps \mid sd)$ และ $(ps \mid ds)$ บน GPU

สำหรับข้อดีของการใช้ Hybrid Algorithm ก็คือ

1. เราสามารถแบ่ง Tasks เพื่อให้ไปคำนวณบน CPU กับ GPU แบบแยกกันได้ซึ่งทำให้การจัดการ Tasks นั้นมีประสิทธิภาพมากขึ้น

2. การคำนวณ ERI ที่เกิดขึ้นบน CPU กับ GPU นั้นแยกกันอย่างสิ้นเชิง ทำให้ไม่มีปัญหาจากการ Communication ทำให้ Operation นั้นเป็นแบบ Asynchronous แบบสมบูรณ์แบบ

3. วิธีการนี้สามารถนำไปใช้ในการคำนวณ ERI ได้สำหรับ Angular Momentum ทุกอัน

## 5.9 การวัดประสิทธิภาพ(ซุปเปอร์)คอมพิวเตอร์

ผู้อ่านเคยสงสัยกันไหมครับว่าเราวัดประสิทธิภาพการคำนวณของคอมพิวเตอร์กันยังไง จริง ๆ แล้วไม่จำเป็นต้องเป็นคอมพิวเตอร์ที่เราใช้ทำงานกันเท่านั้น แต่ขอแค่เป็นอุปกรณ์อิเล็กทรอนิกส์อะไรก็ได้ที่ใช้ Processor ในการประมวลผล เช่น โทรศัพท์หรือแล็ปท็อปของเรา ซึ่งการวัดประสิทธิภาพหรือ Performance ของคอมพิวเตอร์หรือคลัสเตอร์ของเรานั้นต้องวัดออกมาเป็นตัวเลขเพื่อที่เราจะสามารถนำตัวเลขมาเปรียบเทียบประสิทธิภาพหรือ “ความแรง” ของคอมพิวเตอร์ได้ โดยสิ่งที่เราวัดออกมานั้นจะใช้ค่าเทียบเคียงที่เป็นการดำเนินการจุดทศนิยมต่อหนึ่งหน่วยวินาทีหรือ Floating Point Operations Per Second (FLOPS) ซึ่งค่า FLOPS นี้เป็นการวัดประสิทธิภาพมาตรฐานที่ใช้โดยรายการซูเปอร์คอมพิวเตอร์ TOP500[1] โดยเป็นเว็นไซต์จัดอันดับซุปเปอร์คอมพิวเตอร์ตามการดำเนินการ 64 บิต (รูปแบบจุดทศนิยมแบบ Double-Precision) ต่อวินาทีโดยใช้ Library ที่ชื่อว่า HPL (High Performance LINPACK) ซึ่งพัฒนาโดยกลุ่มวิจัยของ Jack Dongarra ซึ่งดำรงตำแหน่งศาสตราจารย์ที่ University of Tennessee, Knoxville โดยในแต่ละปี TOP500 จะทำการจัดอันดับทั้งหมด 2 ครั้ง

สำหรับค่า FLOPS ก็มีการแบ่งตามเลข Prefix ที่มีความต่างยกกำลังระดับ 1000 เท่า ตัวอย่างเช่น

- GigaFLOPS = $10^9$ Operations Per Second
- TeraFLOPS = $10^{12}$ Operations Per Second
- PetaFLOPS = $10^{15}$ Operations Per Second
- ExaFLOPS = $10^{18}$ Operations Per Second

โดยค่า FLOPS สามารถคำนวณได้โดยใช้สมการต่อไปนี้ (สำหรับค่า GigaFLOPS)

$$\texttt{Performance in GFlops} = A \times B \times C \times D \tag{5.9.1}$$

โดยตัวแปรแต่ละตัวคือ

- A = CPU Speed in GHz หรือความเร็วของ CPU ในหน่วย GHz
- B = Number of CPU Cores หรือจำนวน CPU cores
- C = CPU Instruction Per Cycle หรือค่า Instruction ของ CPU ต่อรอบ
- D = Number of CPUs Per Node หรือจำนวน CPU ต่อหนึ่งหน่วยประมวลผล

[1] https://www.top500.org

ตัวอย่างด้านล่างคือการใช้ HPL ในการคำนวณค่า GFlops ของคอมพิวเตอร์ของผู้เขียนที่ใช้หน่วยประมวลผล Intel Xeon โดยเครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบครั้งนี้มี Specfification ตามนี้

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Stepping:              2
CPU MHz:               2400.221
BogoMIPS:              4799.29
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              15360K
NUMA node0 CPU(s): 0,2,4,6,8,10,12,14,16,18,20,22
NUMA node1 CPU(s): 1,3,5,7,9,11,13,15,17,19,21,23
```

โดยหน่วยประมวล (computing node) เครื่องนี้มี 24 CPUs และมีหน่วยความจำ 32 GB

โดยผมสามารถวัดประสิทธิภาพของคอมพิวเตอร์เครื่องนี้ได้โดยการวัดค่า FLOPS โดยใช้ไลบรารี่ HPL โดยด้านล่างแสดงตัวอย่างผลลัพธ์ (Output) ที่ได้[1]

```
CPU frequency:    3.199 GHz
Number of CPUs: 2
Number of cores: 12
Number of threads: 12
Parameters are set to:
Number of tests: 15
Number of equations to solve (problem size) : 1000  2000  5000  10000 15000
    18000 20000 22000 25000 26000 27000 30000 35000 40000 45000
Leading dimension of array                  : 1000  2000  5008  10000 15000
    18008 20016 22008 25000 26000 27000 30000 35000 40000 45000
Number of trials to run                     : 4     2     2     2     2
    2     2     2     2     2     1     1     1     1     1
Data alignment value (in Kbytes)            : 4     4     4     4     4
    4     4     4     4     4     4     1     1     1     1

Maximum memory requested that can be used=16200901024, at the size=45000

```

[1]ผมได้ละขั้นตอนการใช้งาน HPL และการคำนวณแบบละเอียดไป ถ้าหากผู้อ่านมีคำถามก็สามารถติดต่อผมได้ครับ

```
14 ================== Timing linear equation system solver ==================
15 Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
16 1000   1000   4      0.024      27.2993  9.394430e-13 3.203742e-02   pass
17 1000   1000   4      0.006      107.6577 9.394430e-13 3.203742e-02   pass…
18
19 content skipped…
20
21 35000  35000  1      99.141     288.3358 1.275258e-09 3.701880e-02   pass
22 40000  40000  1      146.370    291.5210 1.516881e-09 3.373595e-02   pass
23 45000  45000  1      213.964    283.9451 2.008430e-09 3.533621e-02   pass
24
25 Performance Summary (GFlops)
26 Size   LDA    Align.  Average  Maximal
27 1000   1000   4       86.9113  107.6577…
28
29 content skipped…
30
31 40000  40000  1       291.5210 291.5210
32 45000  45000  1       283.9451 283.9451
```

จาก Output ของ HPL ด้านบนบอกอะไรเราบ้าง จริง ๆ ก็บอกหลายอย่าง แต่อยากผมให้ดูในส่วนท้ายสุดที่มีการบอกค่า GigaFLOPS ของเครื่อง Intel Xeon Node ของผมว่ามีประสิทธิภาพอยู่ที่ 283.945 GFlop/s สรุปง่าย ๆ คือยิ่งมีค่า FLOPS เยอะยิ่งดี

## 5.10 ศึกษาเพิ่มเติมเกี่ยวกับการคำนวณแบบขนาน

เว็บไซต์และหนังสือที่ผมแนะนำให้ผู้ที่สนใจศึกษาเพิ่มเติม

1. Introduction to Parallel Computing Tutorial https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
2. C++ Concurrency in Action: Practical Multithreading. Anthony Williams (2012)
3. The Art of Multiprocessor Programming. Maurice Herlihy (2012)
4. Parallel Computing: Theory and Practice. Umut A. Acar (2016)

ไลบรารี่สำหรับการประมวลผลแบบขนาน

1. CAF: An Open Source Implementation of the Actor Model in C++
2. CGraph: A cross-platform DAG framework based on C++17
3. Chapel: A Programming Language for Productive Parallel Computing on Large-scale Systems
4. Charm++: A Parallel Programming Framework

5. Cilk Plus: C/C++ Extension for Data and Task Parallelism
6. Taskflow: A Modern C++ Parallel Task Programming Library
7. FastFlow: High-performance Parallel Patterns in C++
8. Galois: A C++ Library to Ease Parallel Programming with Irregular Parallelism
9. Heteroflow: Concurrent CPU-GPU Task Programming using Modern C++
10. HPX: A C++ Standard Library for Concurrency and Parallelism
11. Intel TBB: Threading Building Blocks
12. Kokkos: A C++ Programming Model for Writing Performance Portable Applications on HPC platforms
13. MPICH: High-Performance Portable MPI
14. MPL: A message passing library
15. OmpSs: A task based programming model
16. OpenMP: Multi-platform Shared-memory Parallel Programming in C/C++ and Fortran
17. OpenMPI: A High Performance Message Passing Library
18. RaftLib: A C++ Library for Enabling Stream and Dataflow Parallel Computation
19. STAPL: Standard Template Adaptive Parallel Programming Library in C++
20. STLab: High-level Constructs for Implementing Multicore Algorithms with Minimized Contention
21. Transwarp: A Header-only C++ Library for Task Concurrency
22. UPC++: A C++ library that supports Partitioned Global Address Space (PGAS) programming
23. Workflow: C++ Parallel Computing and Asynchronous Networking Engine

## 5.11 แบบฝึกหัด

1. เขียนโปรแกรมคำนวณพลังงานของโมเลกุล 5 โมเลกุล (โมเลกุลอะไรก็ได้) ที่สามารถทำงานได้บน Cluster Computer (ประมวลผลแบบขนานหรือ Parallel Calculation)

2. เขียนโปรแกรม Hartree-Fock ที่สามารถทำงานได้แบบขนาน(Parallel) ด้วยวิธี OpenMP

3. (เสริม) เขียนโปรแกรม Hartree-Fock ที่สามารถทำงานได้แบบขนาน(Parallel) ด้วยวิธี MPI

# ภาคผนวก

# ภาคผนวก A

# เทคนิคทางโครงสร้างเชิงอิเล็กทรอนิกส์

## 1 Static Correlation กับ Dynamic Correlation

การที่เราจะเรียนกลศาสตร์ควอนตัมให้เข้าใจและไปคุยกับคนอื่นรู้เรื่องได้นั้น เราจะต้องรู้ความหมายหรือนิยามของคำศัพท์ทางเทคนิค (Technical Terms) กันก่อน สมมติว่ามีคนสองคนกำลังพูดถึงสิ่งเดียวกัน แต่ตีความสิ่งนั้นกันคนละความหมายก็จบข่าวใช่ไหมครับ ดังนั้นการเข้าใจ Terminology ในทางกลศาสตร์ควอนตัม (เคมีเชิงฟิสิกส์) โดยเฉพาะ Electronic Structure นั้นจึงสำคัญมาก ๆ

ทำไมคำสองคำนี้จึงสำคัญ? จริง ๆ แล้วทฤษฎีพิเศษทางเคมีควอนตัมนั้นมักจะเกี่ยวข้องกับ Correlation ของอิเล็กตรอน เช่น Density Matrix Functional Theory (DMFT) ซึ่งผมคิดว่าเป็นทฤษฎีที่กำลังจะเข้ามาเปลี่ยนวงการเคมีควอนตัมเลยเพราะมันแก้ปัญหาหลาย ๆ อย่างของ Density Functional Theory (DFT) ได้ ดังนั้นในการทำความเข้าใจทฤษฎีเหล่านั้น เราก็ควรที่จะต้องเข้าใจความหมายของคำว่า Correlation กันก่อน

ผมขอเริ่มที่คำว่า Correlation ก่อน ถ้าแปลเป็นภาษาไทยเราจะเรียกว่า “สหสัมพันธ์” ซึ่ง สห คือ “พร้อม ๆ กัน” ส่วน “สัมพันธ์” ก็คือ “ความสัมพันธ์เกี่ยวเนื่องกัน” แล้วอะไรล่ะที่มันเกี่ยวเนื่องเชื่อมโยงกัน? คำตอบก็คือ *อิเล็กตรอน* เพราะในทางกลศาสตร์ควอนตัมที่เน้นทางด้านเคมีนั้นเราติดปัญหาอยู่อย่างเดียวคือการที่จะอธิบายระบบที่มีอิเล็กตรอนหลายตัวนั้นมันทำได้ยาก (มีบทพิสูจน์ออกมาแล้วว่าทำไมไม่ได้เลยในกรณีที่เรายังใช้ทฤษฎี Schrödinger อยู่) ซึ่งนิยามทางคณิตศาสตร์ของ Correlation ก็คือความน่าจะเป็น (Probability) ของการที่เราจะเจออิเล็กตรอนตัวที่ 1 ที่ตำแหน่ง $a$ กับอิเล็กตรอนตัวที่ 2 ที่ตำแหน่ง $b$ ซึ่งเราควรที่จะสามารถคำนวณหา Probability ของสถานการณ์นี้ได้อย่างง่าย ๆ โดยการนำ Probability ของการพบอิเล็กตรอนทั้งสองตัวนี้มาคูณกัน แต่ว่าจริง ๆ แล้วมันไม่ได้ง่ายขนาดนั้น

แนวคิดของคำว่า Correlation ก็คืออันตรกิริยา (Interaction) ระหว่างอิเล็กตรอน ซึ่งต้องเป็นแรงผลักแบบเกิดขึ้น ณ ขณะใดขณะหนึ่งแบบทันที (Instantaneous) ด้วย โดยเราเรียกแรงผลักชนิดนี้ว่า “Dynamic Correlation” นั่นเอง ซึ่งการนิยามคำนี้นั้นเริ่มต้นมาจากการศึกษาการสลายหรือแตกออกของพันธะเคมีใน

โมเลกุล (Bond Dissociation) ถ้าเราค่อย ๆ ดึงอะตอม 2 อะตอมที่มีพันธะเคมีกันอยู่ให้ห่างออกจากกัน เราจะพบว่าอิเล็กตรอนก็จะอยู่ห่างกันมากขึ้น ทำให้แรงผลักลดลง จึงทำให้พลังงานสหสัมพันธ์หรือ Correlation Energy ลดลงตามไปด้วย แล้วที่เราใช้คำว่า Dynamic เพราะมันคือผลที่เกิดจากการเคลื่อนที่ของอิเล็กตรอน (Electron Motion) นั่นเอง

แต่ว่าเรากลับพบว่ามันมีหลาย ๆ กรณีที่มันตรงข้ามกับสิ่งที่ผมเพิ่งอธิบายไปเมื่อกี้นี้ ซึ่งเราพบว่าในกรณีแปลก ๆ พวกนั้นค่าพลังงาน Correlation Energy มันกลับเพิ่มขึ้น คำถามคือ เป็นไปได้ไง? สมมติฐานที่เป็นไปได้ก็คือว่า แสดงว่ามันต้องมี Correlation แบบอื่นที่นอกเหนือจาก Dynamic Correlation หลบซ่อนอยู่แน่ ๆ ซึ่งเราเรียก Correlation แบบนั้นว่า “Static Correlation” นั่นเอง

สุดท้ายแล้วนักเคมีทฤษฎีก็ค้นพบว่าสาเหตุที่มันเป็นแบบนี้เพราะว่ามันมีสิ่งที่เรียกว่า (Near-)degenerate Configuration เพิ่มขึ้นมาซึ่งมันส่งผลหรือ Contribute ต่อพฤติกรรมของฟังก์ชันคลื่นในระหว่างที่พันธะเคมีแตกออกแบบเยอะมาก ๆ เราเลยเรียกระบบพวกนี้ว่า “(Strongly) Statically Correlated System” นี่จึงเป็นสาเหตุที่ทำให้วิธีการคำนวณ เช่น Hartree-Fock ที่ใช้ Single Slater Determinant นั้นใช้งานไม่ได้หรือ Fail นั่นเอง

คำว่า Near-degenerate State ถ้าเราแปลตรงตัวเลยก็คือระดับพลังงานของออร์บิทัลที่อิเล็กตรอนมันอยู่หรือถูกกระตุ้นไปให้ไปอยู่นั้นมันใกล้กันมาก ๆ ซึ่งเราจะพบเหตุการณ์แบบนี้ได้เช่นกรณีที่เราสนใจการกระตุ้นอิเล็กตรอนหลาย ๆ ตัว (หลาย ๆ Configuration) ซึ่งก็จะมีเทคนิคที่แตกต่างกันไปในการจัดการ (Treat) กับคอนฟิกุเรชั่น (Configuration) ของ Excited Electrons พวกนี้ เช่นอาจจะ Treat พร้อมกันหมดทุกกันด้วยวิธี CASSCF หรือทำการตัดหรือแยกกัน treat ด้วยวิธี CCS, CCSD เป็นต้น ซึ่งผมไม่ได้ลงรายละเอียดในหนังสือเล่มนี้

สรุปสั้น ๆ อีกครั้งคือ Static Correlation นั้นมาจากการอธิบายสถานการณ์ที่การที่ฟังก์ชันคลื่นของ Hartree-Fock (HF) ที่เราใช้เป็น Reference Wavefunction นั้นไม่ Fail หรือล้มเหลวในการคำนวณสิ่งต่าง ๆ นั่นก็เพราะว่าโมเดล HF นั้นมันใช้แนวคิดที่ว่าอิเล็กตรอนนั้นมี Instantaneous Interaction กับสนามเฉลี่ย (Mean Field) หรือค่าเฉลี่ยของอิเล็กตรอนทั้งหมด แทนที่จะเป็น Instantaneous Interaction ระหว่างอิเล็กตรอนตัวอื่น ๆ แต่ละตัว ซึ่งในความเป็นจริงนั้นมันควรจะต้องเป็นแบบหลัง

ดังนั้น Dynamic Correlation จึงถูกนำมาใช้ในการอธิบายระบบต่าง ๆ แทนเพราะว่ามันทำให้ Hartree-Fock Reference นั้นถูกต้องมากขึ้น แต่ต้องใส่ดอกจันทร์ตัวหนา ๆ เลยว่าให้ผลการคำนวณถูกต้องแบบ Qualitative เท่านั้น (ให้ผลการคำนวณในภาพรวมแบบที่มีแนวโน้มถูกต้อง) แต่ไม่ถูกต้องแบบ Quantitative (ให้ผลการคำนวณที่ผิดคำนวณผิดหรือคลาดเคลื่อน)

ถ้าให้เข้าใจง่ายกว่านี้อีกก็คือ “Correlation” นั้นมันสื่อถึงความห่วยหรือไร้ประสิทธิภาพ (Deficiency) ของวิธี Hartree-Fock ที่ใช้ Single Slater Determinant นั่นเอง โดยปกติแล้วเราสามารถคำนวณหาพลังงาน Correlation Energy ได้ดังนี้

$$E_{corr} = E_{exact} - E_{HF} \tag{1.1}$$

ก็คือการนำค่าพลังงานจริงมาลบออกด้วยค่าพลังงานที่ได้จากวิธี HF จะได้ Correlation Energy ($E_{corr}$) นั่นหมายความว่า Correlation Energy นั้นคือส่วนที่หายไปที่ HF นั้นต้องการเข้ามาเติมเต็ม ซึ่งมันก็มีวิธีต่าง ๆ มากมายที่เราเรียกกันว่า Post-HF นั้นเข้ามาช่วยในการ Correction โดยการรวม Configuration แบบต่าง ๆ ของ Excited States เข้าไปนั่นเอง วิธี Post-HF ก็มีหลายอัน เช่น $n$th-Order Møller-Plesset Perturbation Theory (MPn), Multi-configurational Self-consistent Field (MCSCF), Configuration Interaction (CI), Full CI

แต่เราต้องเข้าใจให้ถูกต้องอีกนะว่าไม่ใช่วิธี Post-HF ทุกวิธีที่สามารถแก้ปัญหา Correlation โดยการใส่เทอม Dynanic Correlation เข้าไปอย่างเดียว ตัวอย่างเช่น วิธี MPn Perturbation นั้นใช้ Dynamic Correlation ในขณะที่วิธีอย่าง MCSCF นั้นใช้ Static Correlation

แล้วคำถามคือทำไมวิธี Post-HF ต่าง ๆ ถึงไม่รวมทั้ง Static Correlation และ Dynamic Correlation เข้าไปพร้อม ๆ กัน คำอธิบายคือ จริง ๆ แล้วมันเป็นไปไม่ได้เลยที่เราจะแยก Static Correlation กับ Dynamic Correlation ออกจากกันนั่นก็เพราะว่า Correlation ทั้งสองอันนี้มีพื้นฐานมาจาก Physical Interaction ที่เหมือนกัน ดังนั้นวิธีการที่ Cover หรือรวม Dynamic Correlation เข้าไปแล้วนั้นก็จะรวม Effect ของ Correlation แบบที่เป็น Non-dynamic Effect ซึ่งก็คือ Static Correlation เข้าไปด้วย และในทำนองเดียวกันกับวิธีที่รวมเฉพาะ Static Correlation เข้าไป ก็จะรวม Dynamic Correlation เข้าไปด้วยโดยปริยายแล้วนั่นเอง ซึ่ง Correlation ทั้งสองอันนี้มันถูกผสมหรือ Mixed กันอยู่ในเทอมสูง ๆ ของ Wavefunction Configuration

หมายเหตุ 1: ตามที่เราศึกษากันมาว่า Hartree-Fock นั้นไม่มี Correlation ผสมอยู่เลย จริง ๆ แล้วก็ไม่ถูกซะทีเดียว เพราะว่า HF นั้นไม่ยอมให้มีอิเล็กตรอน 2 ตัวใด ๆ มี State เหมือนกันได้ ดังนั้น HF จึงมีความเป็น Correlation อยู่นิดหน่อยนั่นเอง (เรียกว่า Fermi Correlation)

หมายเหตุ 2: Single Slater Determinant นั้นเป็น Representation ของฟังก์ชันคลื่นที่ไม่ค่อยดีเท่าไหร่ ไม่เหมาะนำมาใช้อธิบายระบบ Many-electron หรือระบบที่มีอิเล็กตรอนหลายตัว

## 2 Density Matrix Renormalization Group

ในหัวข้อนี้ผมอยากจะให้ผู้อ่านได้รู้จักกับวิธีควอนตัมอีกวิธีหนึ่งที่ตอนนี้ได้รับความสนใจในหมู่นักเคมีทฤษฎีเป็นอย่างมาก นั่นก็คือ Density Matrix Renormalization Group (DMRG)

DMRG เป็นหนึ่งในทฤษฎีที่ถูกพัฒนามาจาก Quantum Renormalization Group Theory โดยเป็นการใช้ Density Matrix Formulation ที่เสนอโดย Steven White ศาสตราจารย์ทางด้านฟิสิกส์ที่ University of California, Irvine ในช่วงปี 1992 แล้วก็ถูกนำมาประยุกต์ใช้กับงานวิจัยทาง Quantum Chemistry ตั้งแต่นั้นเป็นต้นมา

วิธี DMRG นั้นเป็น Vairational-Based Method ซึ่งนำมาใช้ในการคำนวณ Wavefunction ซึ่งถูกเขียนหรือถูก Represented ด้วยสิ่งที่เรียกว่า Matrix Product State (MPS) หรืออีกชื่อคือ Tensor Chain

หรือ Tensor Network นักวิจัยได้นำทฤษฎี DMRG ไปใช้ศึกษาระบบโมเลกุลแบบพิเศษ (Special Case) บางประเภทที่มีความซับซ้อนและไม่สามารถที่จะใช้วิธีควอนตัมทั่วไปในการอธิบายหรือคำนวณได้ เช่น Strongly Correlated System ซึ่งก็คือระบบที่อิเล็กตรอนนั้นมี Correlation ต่อกันสูงมาก ๆ โดยให้นึกถึงโมเลกุลหรือวัสดุจำพวก Conductor-Insulator Material หรือสารประกอบ Transition Metal Oxide เป็นต้น

แม้ว่า DMRG จะถูกพัฒนามานานกว่า 30 ปีแล้ว แต่ก็ยังไม่ได้เป็นที่แพร่หลายมากนักในกลุ่มนักเคมีเชิงคำนวณ ยิ่งถ้าเป็นการประยุกต์ใช้นั้นก็ไม่ต้องพูดถึงเลย เพราะว่าตัวทฤษฎีนั้นเรียกได้ว่ายังอยู่ในขั้นของการพัฒนาเพื่อให้สามารถนำไปใช้งานได้กับระบบทางเคมีจริง ๆ ได้อยู่

ผมคิดผู้อ่านหลาย ๆ คนอาจจะยังไม่เคยได้ยินแม้แต่ชื่อทฤษฎีอันนี้มาก่อน เท่าที่ผมทราบ (อย่างน้อยก็ ณ วันที่ผมเขียนหนังสือเล่มนี้ซึ่งก็คือเดือนกันยายน พ.ศ. 2566) ในประเทศไทยก็ยังไม่มีกลุ่มวิจัยไหนที่นำทฤษฎีนี้มาใช้เลย แต่ก็ไม่ใช่เรื่องแปลกอะไรเพราะแม้แต่ในต่างประเทศก็มีกลุ่มวิจัยแค่ไม่กี่ที่ในโลกเท่านั้นที่ทำวิจัยโดยใช้วิธีนี้นั่นก็เพราะว่าตัวทฤษฎีนั้นมีความยาก ซับซ้อน และสิ้นเปลืองในเชิงการคำนวณพอสมควร

ถ้าสนใจอ่านเปเปอร์เฉพาะทางที่เกี่ยวข้องกับการพัฒนา DMRG สำหรับโจทย์งานวิจัย Electronic Structure ลองอ่านเปเปอร์ของกลุ่มวิจัยของ Professor Garnet Kin-Lic Chan แห่ง California Institute of Technology หรือ Caltech ซึ่งเป็นกลุ่มวิจัยที่พัฒนา Library สำหรับการคำนวณ DMRG (กลุ่มวิจัยเดียวกันกับที่พัฒนาโปรแกรม PySCF) และก็มีกลุ่มวิจัยของ Professor Markus Reiher แห่ง ETH Zürich ที่พัฒนาทฤษฎี DMRG เพื่อใช้ในการศึกษาและแก้ปัญหาโจทย์ทางเคมีเช่นเดียวกัน

## 3 Density Matrix Functional Theory

Density Matrix Functional Theory (DMFT) เป็นทฤษฎีที่นักเคมีเชิงทฤษฎีเชื่อว่าจะเข้ามาพลิกโฉมเปลี่ยนแปลงวงการเคมีควอนตัม โดย DMFT สามารถแก้ปัญหาหลาย ๆ อย่างของ Density Functional Theory (DFT) ได้ โดยในหัวข้อนี้ผู้อ่านจะได้ศึกษา DMFT แบบเบื้องต้นครับ

ขอเท้าความก่อนว่าตัวทฤษฎี DFT นั้นมีปัญหาหลายอย่าง ทำให้ต้องพึ่งพา Approximation ต่าง ๆ มากมายเพื่อเข้ามาช่วยทำให้การคำนวณระบบเคมีแบบต่าง ๆ นั้นถูกต้องหรือที่เราเรียกว่าการทำ Correction ถ้าหากต้องการรายละเอียดที่ครอบคลุมผมแนะนำให้ทุกคนอ่านบทความรีวิวของ Prof. Kieron Burke “Perspective on Density Functional Theory” ซึ่งสรุปไว้ดีมาก ๆ (อ่านได้ฟรี) ลิงก์: https://pubs.aip.org/aip/jcp/article/136/15/150901/941589

**Density Functional Theory** เริ่มด้วยการสรุป DFT คร่าว ๆ ก่อน ตัว DFT ที่เราใช้กันอยู่ในปัจจุบันนั้นเป็น Kohn-Sham (KS) Framework ซึ่งจะใช้อ้างอิงกับระบบที่อิเล็กตรอนนั้นไม่มีอันตรกิริยาต่อกัน หรือที่เราเรียกว่า Non-Interacting System ซึ่งจะมี Electron Density ที่เท่ากันกับของ Interacting System ส่วนพลังงานของระบบที่สถานะพื้นที่ได้จากการคำนวณด้วย DFT จะหามาจาก Electron Density

(ผมเขียนแทนด้วยตัว $p$) โดยมีสมการดังนี้

$$E_{tot}[p] = T_s[p] + E_{ext}[p] + E_H[p] + E_{XC}[p] \tag{3.1}$$

โดยแต่ละเทอมคือ

- $E_{tot}$ คือ Total energy
- $T_s$ คือ Kinetic energy
- $E_{ext}$ คือ External energy
- $E_H$ คือ Coulomb energy
- $E_{XC}$ คือ Exchange-correlation energy

DFT ของ KS Framework ใช้ KS Orbitals ในการนำมาสร้าง KS Wavefunction ซึ่ง KS Orbitals นั้นจะถูกเขียนด้วย KS Determinant แล้วเราก็สามารถเขียน Electron Density ให้อยู่ในรูปของ KS Determinant ได้อีกด้วย เรามาดูรายละเอียดเฉพาะ Kinetic Energy กับ $E_{XC}$ กัน โดยเฉพาะเทอม $E_{XC}$ นั้นจะซับซ้อนกว่าเพื่อนเพราะว่ายังไม่มี Exact Form ที่สามารถคำนวณได้อย่างถูกต้อง 100

อันแรกคือ Kinetic Energy ($T$) ซึ่งพลังงานจลน์นี้เป็นฟังก์ชันที่ขึ้นกับ Electron Density โดยสามารถหาได้จากการใช้ Kinetic Energy Operator ซึ่งก็คือ Laplacian[1]

อันที่สองคือ $E_{XC}$ โดยเทอมนี้นั้นมีสมการดังต่อไปนี้

$$E_{XC}[p] = T[p] - T_s[p] + E_{ee}[p] - E_H[p] \tag{3.2}$$

สาเหตุที่ผมใส่ $[p]$ ในสมการข้างบนนี้ก็เพราะว่าต้องการจะบอกว่าเทอมทุกเทอมในสมการนี้ขึ้นกับ Electron Density ($p$), แล้วก็ $T$ กับ $E_{ee}$ นั้นคือ Kinetic Energy และ Electron-Electron Energy ของระบบ Interacting System, ส่วน $T_s$ กับ $E_H$ นั้นคือ Kinetic Energy และ Coulomb Energy ของระบบ Non-Interacting System จะเห็นได้ว่าในการคำนวณหา $E_{XC}$ นั้นเราจะต้องรู้ Kinetic Energy ของทั้ง Non-Interacting และ Interacting Systems

**Density Matrix Functional Theory** ผมขอเปรียบเทียบ DMFT กับ DFT โดยการเทียบสมการพลังงานให้เห็นกันชัด ๆ ไปเลยว่าทั้งสองทฤษฎีต่างกันยังไง เงื่อนไขแรกที่เราพิจารณานั้นก็คือว่า DMFT นั้นจะอ้างอิงกับระบบแบบ Interacting System (ไม่เหมือนกับ DFT) ส่วนพลังงานรวมหรือ $E_{tot}$ ของ DMFT

[1]Laplacian คือ Divergence ของ Gradient อีกทีหนึ่ง ถ้าหากว่าเราพิจารณากรณีที่ระบบมีการเคลื่อนที่ใน 1 มิติ เราจะได้ว่าจริง ๆ แล้ว Laplacian ก็คือ Hessian หรืออนุพันธ์อันดับที่สองเทียบกับ Displacement นั่นเอง

นั้นสามารถเขียนได้เหมือนกันกับกรณีพลังงานรวมของ DFT นั่นแหละครับ แต่จะต่างกันตรงที่ว่าเทอมทุกเทอมนั้นไม่ได้ขึ้นกับ Electron Density อีกต่อไปแล้ว แต่ว่าเราจะใช้สิ่งที่เรียกว่า One-electron Reduced Density Matrix (ผมใช้แทนด้วยตัว $y$) แทนตัว Density ดังนี้

$$E_{tot}[y] = T[y] + E_{ext}[y] + E_H[y] + E_{XC}[y] \tag{3.3}$$

ก่อนที่จะอธิบายต่อไป ต้องมาทำความเข้าใจ One-electron Reduced Density Matrix (1-RDM) กันสักนิดนึงก่อน จริง ๆ แล้วนั้น 1-RDM ถูกพิสูจน์มาจากกลศาสตร์ควอนตัมซึ่งในทางฟิสิกส์นั้นเราเริ่มด้วยการ Quantum State ของระบบของเราโดยใช้ Pure State กับ Mixed State (Mixed State คือ Pure State มากกว่าหนึ่งอันมารวมกัน) ซึ่ง Density Matrix นั้นก็คือ Matrix ของ Density ที่เกิดจากผลคูณระหว่าง Probability กับฟังก์ชันคลื่นของ Pure State

ซึ่ง 1-RDM นั้นถูกลดรูปมาจาก Two-Electron Reduced Density Matrix (2-RDM) รายละเอียดเพิ่มเติมอ่านได้ที่หน้าที่ 3 ของเอกสาร “An Introduction to Reduced Density Matrix Functional Theory”[1]

เนื่องจากว่า DMFT นั้นถูกพัฒนาขึ้นมาโดยใช้ Interacting System ดังนั้นเราจึงไม่สามารถคำนวณ 1-RDM หรือ $y$ ได้จาก KS Determinant เหมือนกรณี KS DFT ที่เราใช้ Non-Interacting System ซึ่งนี่เป็นสาเหตุที่ทำให้ตัว DMFT นั้นมีความซับซ้อนกว่า DFT เยอะมาก ๆ

เนื่องจากว่าเราใช้ Determinant ของ KS Orbitals ไม่ได้แล้ว เราจึงจำเป็นต้องใช้ฟังก์ชันคลื่นแบบตรง ๆ ไปเลย แล้วทำการ Diagonalization ซึ่งทำให้เราได้ว่า Spectral Representation ของ 1-RDM นั้นหาได้จาก Natural Orbitals กับ Natural Occupation Number

เมื่อเรากำหนด 1-RDM ได้แล้ว ทำให้เราได้ว่า Kinetic Energy ของ DMFT นั้นสามารถเขียนให้อยู่ในรูปของ 1-RDM ได้ และ Kinetic Energy อันใหม่นี้นั้นก็เป็น Kinetic Energy ของระบบ Interacting System นั่นจึงทำให้ $E_{XC}$ ของ DMFT นั้นไม่ขึ้นกับ Kinetic Energy อีกต่อไป แต่จะขึ้นอยู่กับเพียงแค่ Electron-Electron Interaction Energy เท่านั้น ดังนี้

$$E_{XC}[y] = E_{ee}[y] - E_H[y] \tag{3.4}$$

ซึ่งถ้าเทียบกับ $E_{XC}[p]$ ของกรณี DFT นั้นจะมีความง่ายกว่าเยอะ เมื่อเรามีนิยามที่แน่นอนของ $E_{XC}$ สำหรับ DMFT แล้ว เราจึงสามารถพัฒนา Approximation ต่าง ๆ สำหรับ $E_{XC}$ ได้ เหมือนกันกับ DFT ที่เรามี Functionals ต่าง ๆ ให้เลือกใช้นั่นแหละ เพียงแค่ว่า Functionals ที่เรารู้จักกันใน DFT นั้นไม่สามารถนำมาใช้กับ DMFT ได้

ส่วนการหาค่าพลังงานต่ำสุดของระบบโมเลกุลที่สถานะพื้นนั้นก็ทำได้โดยการ Minimize ค่า $E_{tot}[y]$ โดยเทียบกับ 1-RDM $[y]$ แทนที่จะเทียบกับ Electron Density $[p]$ นอกจากนี้เรายังสามารถคำนวณ Elec-

[1]ไฟล์ PDF: https://quantique.u-strasbg.fr/ISTPC/lib/exe/fetch.php?media=istpc2021:lecture_rdmft_pina_romaniello.pdf และวิดีโอ https://www.youtube.com/watch?v=HN3fXcDCytA

tron Density ของระบบจาก DMFT ได้ด้วย โดยการใช้ Occupation Number กับฟังก์ชันคลื่นนั่นเอง

ตัวทฤษฎี DMFT นั้นถึงแม้ว่าในปัจจุบันนั้นจะยังไม่ค่อยได้ถูกใช้งานแพร่หลายในงานวิจัยทั่ว ๆ ไปมากนัก เนื่องจากว่ายังไม่ค่อย General Purpose สักเท่าไหร่ ทำให้ยังไม่ค่อยเป็นที่นิยมเมื่อเทียบกับ DFT แต่ถ้าหากว่าถึงวันที่ DMFT นั้นสามารถนำมาใช้งานได้ง่ายขึ้น นั่นก็อาจจะเรียกได้ว่าเป็นการเปลี่ยนแปลงครั้งยิ่งใหญ่ (Paradigm Shift) เลยก็ว่าได้

# ภาคผนวก B

# ทฤษฎีและโครงสร้างของโปรแกรม CP2K

## 1 Hamiltonian และ Energy Functional

โปรแกรม CP2K เป็นโปรแกรมเคมีควอนตัมที่มีวัตถุประสงค์ในการจำลองระบบโมเลกุลแบบ Periordic (โดยการใช้ Periodic Boundary Condition) โดยระบบโมเลกุลที่เหมาะกับการจำลองประเภทนี้นั้นก็จะเป็นโมเลกุลจำพวก 2D และ 3D เช่น Solid State, Liquid, Material, Crystal รวมถึงระบบโมเลกุลที่มีขนาดใหญ่ หัวใจสำคัญของ CP2K ก็คือการใช้วิธี Gaussian and Plane Wave (GPW) ซึ่งเป็นการใช้ Representation 2 แบบที่แตกต่างกันในการอธิบายความหนาแน่นของอิเล็กตรอน (Electron Density หรือ $n(\boldsymbol{r})$) ซึ่งเป็นอินพุตสำคัญของวิธี Density Functional Theory (DFT)

Representation อันแรกที่ใช้นั้นผู้อ่านทุกคนน่าจะทราบกันเป็นว่าอย่าง นั่นก็คือการใช้ Gaussian Functions โดนเราจะทำการกระจายหรือ Expand Gaussian Functions อันนี้ไปยังอะตอมทุก ๆ อะตอมในโมเลกุลแล้วก็ทำการรวม (Sum) ทุกเทอมเข้าด้วยกัน ซึ่งเขียนเป็นสมการได้ดังนี้

$$n(\boldsymbol{r}) = \sum_{\mu v} P^{\mu v} \varphi_{\mu}(\boldsymbol{r}) \varphi_{v}(\boldsymbol{r}) \tag{1.1}$$

โดยที่ $P^{\mu v}$ คือสมาชิกของ Density Matrix และ $\varphi_{\mu}(\boldsymbol{r}) = \sum_i d_{i\mu} g_i(\boldsymbol{r})$ โดยที่มี Primitive Gaussian Functions คือ $g_i(\boldsymbol{r})$ และมี Contraction Coefficients คือ $d_{i\mu}$

ส่วน Representation อันที่สองนั้นเป็นการใช้ประโยชน์ของ Auxiliary Basis ของ Plane Waves ดังนี้

$$\tilde{n}(\boldsymbol{r}) = \frac{1}{\Omega} \sum_{\boldsymbol{G}} \tilde{n}(\boldsymbol{G}) \exp(\mathrm{i}\boldsymbol{G} \cdot \boldsymbol{r}) \tag{1.2}$$

โดยที่ $\Omega$ คือปริมาตรของ Unit Cell (ขนาดของระบบที่เรากำหนด) และ $\boldsymbol{G}$ คือ Reciprocal Lattice Vectors ส่วน Expansion Coefficients $\tilde{n}(\boldsymbol{G})$ นั้นมีหน้าที่คือทำให้ $\tilde{n}(\boldsymbol{r})$ นั้นมีค่าเท่ากันกับ $n(\boldsymbol{r})$ บน Regular Grid ใน Unit Cell นอกจากนี้แล้วยังมีเทคนิคที่เราสามารถทำการแปลงสลับไปมาระหว่าง $n(\boldsymbol{r})$, $\tilde{n}(\boldsymbol{r})$ และ $\tilde{n}(\boldsymbol{G})$ โดยการใช้เทคนิคการ Mapping และ Fast Fourier Transforms (FFT)

พลังงานรวมของโมเลกุลที่คำนวณด้วยวิธี Kohn-Sham Density Functional Theory (DFT) โดยการใช้ Gaussian Plane Wave (GPW) ในโปรแกรม CP2K นั้นมีสมการดังต่อไปนี้

$$E[n] = E^{\mathrm{T}}[n] + E^{\mathrm{V}}[n] + E^{\mathrm{H}}[n] + E^{\mathrm{XC}}[n] + E^{\mathrm{II}} \tag{1.3}$$

โดยที่สมการ (1.3) นั้นเป็นสูตรของพลังงานรวมที่เกิดจากการนำพลังงานต่าง ๆ มารวมกัน อาจจะเปรียบเทียบสูตรพลังงานกับสูตรการทำอาหารก็ได้ ซึ่งพลังงานแต่ละเทอมนั้นก็คือส่วนผสมที่โปรแกรม CP2K นั้นเลือกใช้ (ตามทฤษฎี) ในทำนองเดียวกัน โปรแกรมเคมีควอนตัมอื่น ๆ นั้นต่างก็มีสูตรที่ใช้ในการคำนวณพลังงานรวมของโมเลกุลเป็นของตัวเอง ขึ้นกับว่าใช้ทฤษฎีไหน สำหรับพลังงานแต่ละเทอมในสมการที่ (1.3) นั้นมี Expression ดังนี้

$$\begin{aligned} E[n] = & \sum_{\mu\nu} P^{\mu\nu} \left\langle \varphi_\mu(\boldsymbol{r}) \right| -\frac{1}{2}\nabla^2 \left| \varphi_\nu(\boldsymbol{r}) \right\rangle \\ & + \sum_{\mu\nu} P^{\mu\nu} \left\langle \varphi_\mu(\boldsymbol{r}) \left| V_{\mathrm{loc}}^{\mathrm{PP}}(r) \right| \varphi_\nu(\boldsymbol{r}) \right\rangle \\ & + \sum_{\mu\nu} P^{\mu\nu} \left\langle \varphi_\mu(\boldsymbol{r}) \left| V_{\mathrm{nl}}^{\mathrm{PP}}(\boldsymbol{r}, \boldsymbol{r}') \right| \varphi_\nu(\boldsymbol{r}') \right\rangle \\ & + 2\pi\Omega \sum_G \frac{\widetilde{n}^*(\boldsymbol{G})\tilde{n}(\boldsymbol{G})}{\boldsymbol{G}^2} \qquad (1.4) \\ & + \int e^{\mathrm{xc}}(\boldsymbol{r})\mathrm{d}\boldsymbol{r} \\ & + \frac{1}{2}\sum_{I\neq J} \frac{Z_I Z_J}{|\boldsymbol{R}_I - \boldsymbol{R}_J|} \qquad (1.5) \end{aligned}$$

# 2 ทำความเข้าใจและวิเคราะห์โปรแกรม CP2K

# 3 การพัฒนาโปรแกรม CP2K

# ภาคผนวก C

# Equation of State สำหรับของเหลว

ในหัวข้อนี้เราจะมาศึกษา MD กันด้วย Virial Theorem กันก่อนซึ่งเป็นหลักการที่ถูกนำมาประยุกต์ใช้ในการคำนวณระบบที่มีอนุภาพ N ตัว (N-body)

ของเหลว (Liquid) เป็นสถานะหนึ่งของสสารที่พลังงานศักย์นั้นมีดีกรีหรือระดับของพลังงานที่เท่ากับพลังงานจลน์ โดยเราสามารถใช้การคำนวณ MD มาคำนวณหา Equation of State ของของเหลวได้ เริ่มต้นสมมติว่าเรามีระบบที่มี $N$ โมเลกุลซึ่งมวลของแต่ละโมเลกุลนั้นมีค่าเท่ากับ $m$ และโมเลกุลทั้งหมดนั้นถูกจำลองอยู่ภายในกล่องทรงลูกบาศก์ที่มีความยาวของแต่ละด้านเท่ากับ $L$ เรากำหนดให้พิกัดเริ่มต้นของโมเลกุลนั้นคือ $\vec{r_i} = (x_i, y_i, z_i)$ และมีความเร็วคือ $\vec{v_i} = (u_i, v_i, w_i)$ และมีแรงคือ $\vec{F_i} = (f_i, g_i, h_i), i = 1, 2, ..., N$

จากกฎข้อที่สองของกฎการเคลื่อนที่ของนิวตันนั้น เรามีความสัมพันธ์คือ

$$\vec{F_i} = m\frac{d^2\vec{r_i}}{dt^2} \tag{0.1}$$

ซึ่งเราสามารถเขียนได้เป็น

$$\begin{aligned} \frac{d}{dt}\left(\frac{d\left(x_i^2\right)}{dt}\right) &= \frac{d}{dt}\left(2x_i\frac{dx_i}{dt}\right) && (0.2)\\ &= 2\frac{dx_i}{dt}\frac{dx_i}{dt} + 2x_i\frac{d^2x_i}{dt^2} && (0.3)\\ &= 2u_i^2 + 2x_i\frac{d^2x_i}{dt^2} && (0.4) \end{aligned}$$

สมการด้านบนนั้นเป็นการบอกใบ้เราว่าแรงที่กระทำต่อระยะกระจัดที่โมเลกุลเคลื่อนที่ไปในแต่ละทิศทางนั้น

สามารถเขียนได้เป็นอนุพันธ์ของฟังก์ชันที่ขึ้นกับความเร็วและความเร่งดังนี้

$$\begin{aligned} f_i x_i &= m x_i \frac{d^2 x_i}{dt^2} && (0.5) \\ &= \frac{1}{2} m \frac{d}{dt}\left(\frac{d\left(x_i^2\right)}{dt}\right) - m u_i^2 && (0.6) \end{aligned}$$

$$\begin{aligned} g_i y_i &= m y_i \frac{d^2 y_i}{dt^2} && (0.7) \\ &= \frac{1}{2} m \frac{d}{dt}\left(\frac{d\left(y_i^2\right)}{dt}\right) - m v_i^2 && (0.8) \end{aligned}$$

$$\begin{aligned} h_i z_i &= m z_i \frac{d^2 z_i}{dt^2} && (0.9) \\ &= \frac{1}{2} m \frac{d}{dt}\left(\frac{d\left(z_i^2\right)}{dt}\right) - m w_i^2 && (0.10) \end{aligned}$$

เมื่อเรารวมสมการด้านบนทั้งสามสมการเข้าด้วยกัน เราจะได้ว่า

$$\frac{1}{2} m \frac{d}{dt}\left\{\frac{d}{dt}\left(x_i^2 + y_i^2 + z_i^2\right)\right\} = \vec{F}_i \cdot \vec{r}_i + m\left|\vec{v}_i\right|^2 \tag{0.11}$$

และเมื่อเราทำการรวมสมการสำหรับทุกโมเลกุลในระบบ $(i = 1, 2, \ldots, N)$ เราจะได้ว่า

$$\frac{1}{2} m \sum_{i=1}^{N} \frac{d}{dt}\left\{\frac{d}{dt} r_i^2\right\} = \sum_{i=1}^{N} \vec{F}_i \cdot \vec{r}_i + m \sum_{i=1}^{N} \left|\vec{v}_i\right|^2 \tag{0.12}$$

ต่อจากนั้นก็เราก็ทำการกำหนดให้กรอบอ้างอิงของระบบของเรานั้นอยู่ที่จุดกึ่งกลางของกล่องทรงลูกบาศน์ ซึ่งจะทำให้เรามีความสัมพันธ์ดังต่อไปนี้

$$r_i^2 \equiv \left|\vec{r}_i\right|^2 = x_i^2 + y_i^2 + z_i^2 \tag{0.13}$$

ซึ่งก็คือระยะห่างจากจุดกำเนิด (จุดศูนย์กลางของกล่อง) ถึงโมเลกุล $i$ ยกกำลังสอง โดยทางด้านซ้ายของสมการที่ (0.12) นั้นจริง ๆ แล้วก็คืออัตราของความเร่งของระยะห่างยกกำลังสองเฉลี่ย (Mean Square

Distance) นั่นเอง ซึ่งปริมาณตัวนี้จะต้องมีค่าเท่ากับ 0 เสมอเพราะว่าของเหลวนั้นอยู่ในระบบปิด (ในที่นี้คือ Fixed Cube) ส่วนเทอมที่ 2 ของทางด้านขวาของสมการที่ (0.12) นั้นมีค่าเป็นสองเท่าของพลังงานจลน์รวมของระบบซึ่งหลายคนน่าจะเคยศึกษามาก่อนแล้วจากวิชาเคมีเชิงฟิสิกส์ในระดับปริญญาตรีว่าโมเลกุลแต่ละโมเลกุลนั้นจะมีค่าเฉลี่ยของพลังงานจลน์ของการเลื่อนตำแหน่ง (Translation) เท่ากับ $\frac{3}{2}k\theta$ โดยที่ $k$ คือค่าคงที่โบลท์ซมันน์ (Boltzmann Constant) และ $\theta$ คืออุณหภูมิสัมบูรณ์ ดังนั้นเทอมที่สองของด้านขวาของสมการนี้จะต้องมีค่าเฉลี่ยเท่ากับ $3Nk\theta$

สำหรับ $\sum_{i=1}^{N} \vec{F_i} \cdot \vec{r_i}$ ซึ่งเป็นเทอมแรกของทางด้านขวาของสมการที่ (0.12) นั้นมีชื่อเรียกว่า *Virial of Clausius* และมีมิติ (หน่วย) ที่เหมือนกันกับทอร์ก (Torque) โดย Virial นั้นสามารถหาได้จากค่าคาดหวัง (Expectation Value) ซึ่งเป็นค่าเฉลี่ย ดังนี้

$$-PV + \left\langle \sum_{i \neq j} F(r_{ij}) r_{ij} \right\rangle \tag{0.14}$$

คราวนี้เราจะมาทำการพิสูจน์สมการของ Virial ของสมการที่ (0.14) โดยเราจะพิจารณาที่ละเทอม โดยเทอมแรกนั้นคือแรงที่เกิดขึ้นจากขอบ (Boundary) ของระบบหรือด้านข้างของกล่องลูกบาศน์นั่นเอง ส่วนเทอมที่สองคือแรงที่กระทำต่อโมเลกุลซึ่งเป็นแรงระหว่างโมเลกุล

**1) แรงที่เกิดขึ้นที่ผนังของกล่อง** เริ่มต้นนั้นเราสามารถคำนวณหาแรงที่เกิดขึ้นบนผนัง ณ ตำแหน่ง $x = \frac{L}{2}$ ได้เท่ากับ $PL^2$ โดยที่ $P$ คือความดันที่กระทำต่อผนังและ $L^2$ คือพื้นที่ของผนัง ณ ตำแหน่ง $x = \frac{L}{2}$ แล้วเราก็ใช้กฎข้อที่ 3 ของกฎการเคลื่อนที่ของนิวตันเราจะได้ว่าผลรวมของแรงทั้งหมดที่กระทำต่อโมเลกุลที่อยู่ใกล้ ๆ ผนังนั้นจะมีค่าเท่ากับ $-PL^2$ ซึ่งเราจะได้ว่า Contribution ที่เกี่ยวข้องกับ Virial นั้นคือ

$$-\frac{1}{2}PL^3 \tag{0.15}$$

เมื่อเราพิจารณาผนังทั้ง 6 ด้านของลูกบาศก์นั้นเราจะต้องทำการรวม Virial ของทั้ง 6 ด้านเข้าด้วยกัน พูดง่าย ๆ คือเราคูณสมการที่ (0.15) ด้วย 6 ซึ่งจะทำให้เราได้สมการ Virial ที่สมบูรณ์มากขึ้น ดังนี้

$$-3PL^3 = -3PV \tag{0.16}$$

โดยที่ $V = L^3$ คือปริมาตรของลูกบาศก์

**2) แรงระหว่างโมเลกุล** เรากำหนดให้ $x$ เป็นองค์ประกอบเชิงเวกเตอร์ของแรงที่กระทำต่อโมเลกุล $i$ ที่เกิดขึ้นจากโมเลกุล $j$ โดยเขียนแทนด้วย $f(r_{ij})$ ซึ่งเราจะมีความสัมพันธ์ดังนี้

$$f_i = \sum_{j=1 j\neq i}^{N} f\left(r_{ij}\right) \tag{0.17}$$

ในทำนองเดียวกันเราก็จะมีแรงที่เหมือนกันกับกรณีข้างต้นด้วย ดังต่อไปนี้

$$g_i = \sum_{j=1 j\neq i}^{N} g\left(r_{ij}\right) \tag{0.18}$$

$$h_i = \sum_{j=1 j\neq i}^{N} h\left(r_{ij}\right) \tag{0.19}$$

ดังนั้นส่วนที่เป็นแรงที่เกิดขึ้นจากอันตรกิริยาระหว่างโมเลกุลของ Virial นั้นจึงสามารถอธิบายได้ด้วยสมการดังต่อไปนี้

$$\sum_{i=1}^{N}\left[\left\{\sum_{j=1 j\neq i}^{N} f(r_{ij})\right\} x_i + \left\{\sum_{j=1 j\neq i}^{N} g(r_{ij})\right\} y_i + \left\{\sum_{j=1 j\neq i}^{N} h(r_{ij})\right\} z_i\right] \tag{0.20}$$

ลำดับไปต่อเราเราสามารถคำนวณหา Contribution ที่เกิดขึ้นกับสมการด้านบนได้โดยมีความสัมพันธ์เพิ่มเติมคือ

$$\begin{aligned} f\left(r_{ij}\right) x_i + f\left(r_{ji}\right) x_j &= f\left(r_{ij}\right) x_i - f\left(r_{ij}\right) x_j & (0.21)\\ &= f\left(r_{ij}\right)\left(x_i - x_j\right) & (0.22)\end{aligned}$$

$$g\left(r_{ij}\right) y_j + g\left(r_{ji}\right) y_j = g\left(r_{ij}\right)\left(y_i - y_j\right) \tag{0.23}$$

$$h\left(r_{ij}\right) z_i + h\left(r_{ji}\right) z_j = h\left(r_{ij}\right)\left(z_i - z_j\right) \tag{0.24}$$

เราสามารถใช้สมการข้างต้นนี้ได้เพราะว่าแรงที่โมเลกุล $j$ กระทำต่อโมเลกุล $i$ นั้นมีขนาดที่เท่ากับแรงที่โมเลกุล $i$ กระทำต่อโมเลกุล $j$ แต่ว่ามีขนาดตรงข้ามกัน

ถ้าหากเรากำหนดให้ $\left(f\left(r_{ij}\right), g\left(r_{ij}\right), h\left(r_{ij}\right)\right)$ มีทิศทางที่อยู่ในแนวเดียวกันกับเส้นตรงระหว่าง $\vec{r_i}$ ถึง $\vec{r_j}$ เช่น

$$\vec{F}(r_{ij}) \cdot (\vec{r_j} - \vec{r_i}) = F\left(r_{ij}\right) r_{ij} \tag{0.25}$$

โดยที่ $F\left(r_{ij}\right) = \left|\vec{F}(r_{ij})\right|$ และ $r_{ij} = |\vec{r_j} - \vec{r_i}|$ เราจะได้ว่า Virial นั้นจริง ๆ แล้วมีค่าเท่ากับสมการดังต่อไปนี้

$$-3PV + \left\langle \sum_{i \neq j} F\left(r_{ij}\right) r_{ij} \right\rangle \tag{0.26}$$

ซึ่งเราสามารถเขียนใหม่ได้เป็น

$$3PV = 3NK\theta + \left\langle \sum_{i \neq j} F\left(r_{ij}\right) r_{ij} \right\rangle \tag{0.27}$$

# บรรณานุกรม


[1] Attila Szabo and Neil S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Reprint Edition. Mineola, N.Y: Dover Publications, July 1996 (cit. on p. xi).

[2] Peter W. Atkins and Ronald S. Friedman. *Molecular Quantum Mechanics*. 5th edition. Oxford ; New York: Oxford University Press, Dec. 2010 (cit. on p. xi).

[3] Robert G. Parr and Yang Weitao. *Density-Functional Theory of Atoms and Molecules*. New York, NY: Oxford University Press, May 1994 (cit. on p. xi).

[4] Michael P. Allen and Dominic J. Tildesley. *Computer Simulation of Liquids*. 2nd edition. Oxford, United Kingdom: Oxford University Press, Aug. 2017 (cit. on p. xi).

[5] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. 2nd edition. San Diego: Academic Press, Nov. 2001 (cit. on p. xi).

[6] Dominik Marx and Jürg Hutter. *Ab Initio Molecular Dynamics: Basic Theory and Advanced Methods*. 1st edition. Cambridge ; New York: Cambridge University Press, May 2009 (cit. on p. xi).

[7] P. J. Stephens et al. "Ab Initio Calculation of Vibrational Absorption and Circular Dichroism Spectra Using Density Functional Force Fields". In: *The Journal of Physical Chemistry* 98.45 (Nov. 1994), pp. 11623–11627 (cit. on p. 68).

[8] Nicholas Metropolis et al. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (June 1953), pp. 1087–1092 (cit. on p. 70).

[9] B. J. Alder and T. E. Wainwright. "Phase Transition for a Hard Sphere System". In: *The Journal of Chemical Physics* 27.5 (Nov. 1957), pp. 1208–1209 (cit. on p. 70).

[10] J. C. Kendrew et al. "A Three-Dimensional Model of the Myoglobin Molecule Obtained by X-Ray Analysis". In: *Nature* 181.4610 (4610 Mar. 1958), pp. 662–666 (cit. on p. 71).

[11] A. Rahman. "Correlations in the Motion of Atoms in Liquid Argon". In: *Physical Review* 136 (2A Oct. 19, 1964), A405–A411 (cit. on p. 71).

[12] Aneesur Rahman and Frank H. Stillinger. "Molecular Dynamics Study of Liquid Water". In: *The Journal of Chemical Physics* 55.7 (Oct. 1971), pp. 3336–3359 (cit. on p. 71).

[13] Michael Levitt and Arieh Warshel. "Computer Simulation of Protein Folding". In: *Nature* 253.5494 (5494 Feb. 1975), pp. 694–698 (cit. on p. 71).

[14] D. A. Case and M. Karplus. “Dynamics of Ligand Binding to Heme Proteins”. In: *Journal of Molecular Biology* 132.3 (Aug. 15, 1979), pp. 343–368 (cit. on p. 71).

[15] R. Car and M. Parrinello. “Unified Approach for Molecular Dynamics and Density-Functional Theory”. In: *Physical Review Letters* 55.22 (Nov. 25, 1985), pp. 2471–2474 (cit. on p. 71).

[16] M Levitt and R Sharon. “Accurate Simulation of Protein Dynamics in Solution.” In: *Proceedings of the National Academy of Sciences* 85.20 (Oct. 1988), pp. 7557–7561 (cit. on p. 71).

[17] Wilfried J. Mortier, Swapan K. Ghosh, and S. Shankar. “Electronegativity-Equalization Method for the Calculation of Atomic Charges in Molecules”. In: *Journal of the American Chemical Society* 108.15 (July 1986), pp. 4315–4320 (cit. on p. 82).

[18] Crina-Maria Ionescu et al. “Rapid Calculation of Accurate Atomic Charges for Proteins via the Electronegativity Equalization Method”. In: *Journal of Chemical Information and Modeling* 53.10 (Oct. 2013), pp. 2548–2558 (cit. on p. 82).

[19] Robert G. Parr and Yang Weitao. *Density-Functional Theory of Atoms and Molecules*. New York, NY: Oxford University Press, May 1994 (cit. on p. 82).

[20] Zhongzhi Yang, Changsheng Wang, and Aoqing Tang. “Molecular Electronegativity in Density Functional Theory (VI)”. In: *Science in China Series B: Chemistry* 41.3 (June 1998), pp. 331–336 (cit. on p. 82).

[21] Angelo Vedani. “YETI: An Interactive Molecular Mechanics Program for Small-Molecule Protein Complexes”. In: *Journal of Computational Chemistry* 9.3 (1988), pp. 269–280 (cit. on p. 85).

[22] Franz J. Vesely. “N-Particle Dynamics of Polarizable Stockmayer-type Molecules”. In: *Journal of Computational Physics* 24.4 (Aug. 1977), pp. 361–371 (cit. on p. 86).

[23] R. Eisenschitz and F. London. “Über das Verhältnis der van der Waalsschen Kräfte zu den homöopolaren Bindungskräften”. In: *Zeitschrift für Physik* 60.7 (July 1930), pp. 491–527 (cit. on p. 88).

[24] F. London. “Zur Theorie und Systematik der Molekularkräfte”. In: *Zeitschrift für Physik* 63.3 (Mar. 1930), pp. 245–279 (cit. on p. 88).

[25] F. London. “The General Theory of Molecular Forces”. In: *Transactions of the Faraday Society* 33.0 (Jan. 1937), 8b–26 (cit. on p. 88).

[26] Péter Pulay. “Convergence Acceleration of Iterative Sequences. the Case of Scf Iteration”. In: *Chemical Physics Letters* 73.2 (July 1980), pp. 393–398 (cit. on p. 154).

[27] P. Pulay. “Improved SCF Convergence Acceleration”. In: *Journal of Computational Chemistry* 3.4 (1982), pp. 556–560 (cit. on p. 156).

[28] John F. Stanton et al. “A Direct Product Decomposition Approach for Symmetry Exploitation in Many-body Methods. I. Energy Calculations”. In: *The Journal of Chemical Physics* 94.6 (Mar. 1991), pp. 4334–4345 (cit. on p. 171).

[29] T. Daniel Crawford and Henry F. Schaefer III. “An Introduction to Coupled Cluster Theory for Computational Chemists”. In: *Reviews in Computational Chemistry*. John Wiley & Sons, Ltd, 2000, pp. 33–136 (cit. on p. 182).

[30] Justin T. Fermann and Edward F. Valeev. *Fundamentals of Molecular Integrals Evaluation*. July 2020. arXiv: 2007.12057 `[physics, physics:quant-ph]` (cit. on p. 196).

[31] Koji Yasuda. “Two-Electron Integral Evaluation on the Graphics Processor Unit”. In: *Journal of Computational Chemistry* 29.3 (2008), pp. 334–342 (cit. on p. 232).

[32] Ji Qi, Yingfeng Zhang, and Minghui Yang. “A Hybrid CPU/GPU Method for Hartree–Fock Self-Consistent-Field Calculation”. In: *The Journal of Chemical Physics* 159.10 (Sept. 2023), p. 104101 (cit. on p. 232).

[33] Koji Yasuda and Hironori Maruoka. “Efficient Calculation of Two-Electron Integrals for High Angular Basis Functions”. In: *International Journal of Quantum Chemistry* 114.9 (2014), pp. 543–552 (cit. on p. 232).

[34] Ivan S. Ufimtsev and Todd J. Martínez. “Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation”. In: *Journal of Chemical Theory and Computation* 4.2 (Feb. 2008), pp. 222–231 (cit. on p. 232).

[35] Ivan S. Ufimtsev and Todd J. Martinez. “Quantum Chemistry on Graphical Processing Units. 2. Direct Self-Consistent-Field Implementation”. In: *Journal of Chemical Theory and Computation* 5.4 (Apr. 2009), pp. 1004–1015 (cit. on p. 232).

[36] Alexey V. Titov et al. “Generating Efficient Quantum Chemistry Codes for Novel Architectures”. In: *Journal of Chemical Theory and Computation* 9.1 (Jan. 2013), pp. 213–221 (cit. on p. 232).

[37] Nathan Luehr, Ivan S. Ufimtsev, and Todd J. Martínez. “Dynamic Precision for Electron Repulsion Integral Evaluation on Graphical Processing Units (GPUs)”. In: *Journal of Chemical Theory and Computation* 7.4 (Apr. 2011), pp. 949–954 (cit. on p. 233).

[38] K. Grace Johnson et al. “Multinode Multi-GPU Two-Electron Integrals: Code Generation Using the Regent Language”. In: *Journal of Chemical Theory and Computation* 18.11 (Nov. 2022), pp. 6522–6536 (cit. on p. 233).

[39] Andrey Asadchev et al. “Uncontracted Rys Quadrature Implementation of up to G Functions on Graphical Processing Units”. In: *Journal of Chemical Theory and Computation* 6.3 (Mar. 2010), pp. 696–704 (cit. on p. 233).

[40] Andrey Asadchev and Mark S. Gordon. “New Multithreaded Hybrid CPU/GPU Approach to Hartree–Fock”. In: *Journal of Chemical Theory and Computation* 8.11 (Nov. 2012), pp. 4166–4176 (cit. on p. 233).

[41] Giuseppe M. J. Barca et al. “High-Performance, Graphics Processing Unit-Accelerated Fock Build Algorithm”. In: *Journal of Chemical Theory and Computation* 16.12 (Dec. 2020), pp. 7232–7238 (cit. on p. 233).

[42] Giuseppe M. J. Barca et al. “Faster Self-Consistent Field (SCF) Calculations on GPU Clusters”. In: *Journal of Chemical Theory and Computation* 17.12 (Dec. 2021), pp. 7486–7503 (cit. on p. 233).

[43] Jörg Kussmann and Christian Ochsenfeld. “Pre-Selective Screening for Matrix Elements in Linear-Scaling Exact Exchange Calculations”. In: *The Journal of Chemical Physics* 138.13 (Apr. 2013), p. 134114 (cit. on p. 233).

[44] Yipu Miao and Kenneth M. Jr. Merz. “Acceleration of Electron Repulsion Integral Evaluation on Graphics Processing Units via Use of Recurrence Relations”. In: *Journal of Chemical Theory and Computation* 9.2 (Feb. 2013), pp. 965–976 (cit. on p. 233).

[45] Yipu Miao and Kenneth M. Jr. Merz. “Acceleration of High Angular Momentum Electron Repulsion Integrals and Integral Derivatives on Graphics Processing Units”. In: *Journal of Chemical Theory and Computation* 11.4 (Apr. 2015), pp. 1449–1462 (cit. on p. 233).

[46] Ádám Rák and György Cserey. “The BRUSH Algorithm for Two-Electron Integrals on GPU”. In: *Chemical Physics Letters* 622 (Feb. 2015), pp. 92–98 (cit. on p. 233).

[47] Gábor János Tornai et al. “Calculation of Quantum Chemical Two-Electron Integrals by Applying Compiler Technology on GPU”. In: *Journal of Chemical Theory and Computation* 15.10 (Oct. 2019), pp. 5319–5331 (cit. on p. 233).

[48] Kyle D. Fernandes, C. Alicia Renison, and Kevin J. Naidoo. “Quantum Supercharger Library: Hyper-parallelism of the Hartree–Fock Method”. In: *Journal of Computational Chemistry* 36.18 (2015), pp. 1399–1409 (cit. on p. 233).

[49] Yingqi Tian et al. “Optimizing Two-Electron Repulsion Integral Calculations with McMurchie–Davidson Method on Graphic Processing Unit”. In: *The Journal of Chemical Physics* 155.3 (July 2021) (cit. on p. 233).

# ดรรชนีภาษาไทย


การจำลองเชิงตัวเลข, 1
การทำให้เกิดเมทริกซ์รูปทแยง, 222
การประมาณของบอร์น-ออพเพนไฮเมอร์, 21
การประมาณของฮาร์ทรี-ฟ็อค, 37
การผลัก, 88
การสุ่มตัวอย่างแบบมีประสิทธิภาพ, 111
การแพร่กระจาย, 87
ทฤษฎีฟังก์ชันคลื่น, 50
ทฤษฎีฟังก์ชันนอลความหนาแน่น, 61
    การคำนวณพลังงานของระบบอิเล็กตรอน Kohn-Sham, 183
พลังงานระหว่างโมเลกุล, 79
    พลังงานผลัก, 88
    พลังงานศักย์ของเลนนาร์ด-โจนส์, 89
    พลังงานเหนี่ยวนำ, 85
    พลังงานแพร่กระจาย, 87
    พลังงานไฟฟ้าสถิตย์, 80
พลังงานสหสัมพันธ์ของอิเล็กตรอน, 49
พันธะไฮโดรเจน, 85
พื้นผิวพลังงานศักย์, 22
วิธีเชิงวิเคราะห์, 1
สนามแรง, 71
    พันธะโควาเลนท์, 73
สมการของการเคลื่อนที่, 90
สมการชโรดิงเงอร์, 15
หน่วยอะตอม, 14
หลักการผันแปร, 36
หลักเอาฟเบา, 23
ออร์บิทัลเชิงอะตอม, 22
ออร์บิทัลเชิงโมเลกุล, 26
อันตรกิริยาระหว่างโมเลกุล, 79
    อันตรกิริยาแบบอ่อน, 79
อันตรกิริยาแบบไฟฟ้าสถิตย์, 80
อินทิกรัลซ้อนทับ, 194
อินทิกรัลพลังงานจลน์, 200
อินทิกรัลแรงดึงดูดเชิงนิวเคลียร์, 202
อินทิกรัลแรงผลักระหว่างอิเล็กตรอน, 206
เงื่อนไขเริ่มต้น, 1
เทคนิคเชิงคอมพิวเตอร์, 1
เบซิสเซท, 43
เมตาไดนามิกส์, 111
เมทริกซ์ความหนาแน่น, 45
แฮมิลโทเนียนเชิงโมเลกุล, 17
โพลาไรเซชั่นเชิงอิเล็กทรอนิกส์, 85

# ดรรชนีภาษาอังกฤษ


Analytical Method, 1
Angle Bending, 75
Atomic Orbitals, 22
Atomic Units, 14
Aufbau Principle, 23

Basis Sets, 43
Bond Stretching, 74
Born-Oppenheimer Approximation, 21

Computer Simulation, 1
Configuration Interaction, 50
Correlation Energy, 49
Coupled Cluster Theory, 55
Cross Terms, 77

Density Functional Theory, 61
    Kohn-Sham Energy Calculation, 183
Density Matrices, 45
Density Matrix Functional Theory, 245
Density Matrix Renormalization Group, 244
Dihedral Terms, 76
DIIS, 154
    Commutator-DIIS, 156
Direct Inversion of the Iterative Subspace, 154
Dispersion, 87
Dynamic Correlation, 242

Electronegativity Equalization Model, 82
Electronic Polarization, 85
Electrostatic Interactions, 80
Enhanced Sampling, 111
Equations of Motion, 90

Force Fields, 71
    Covalent Bonding, 73

Hamiltonian
    Molecular Hamiltonian, 17
Hartree-Fock Approximation, 37
Hydrogen Bonding, 85

Initial Conditions, 1
Intermolecular Energy, 79
    Dispersion Energy, 87
    Electrostatic Energy, 80
    Induction Energy, 85
    Lennard-Jones Potential, 89
    Repulsion Energy, 88
Intermolecular Interactions, 79
    Weak Interaction, 79

Kinetic Energy Integrals, 200
Kohn-Sham Approach, 62

Møllor-Plesset Perturbation, 52
Matrix Diagonalization, 222
Metadynamics, 111
Molecular Orbitals, 26

Nuclear Attraction Integrals, 202
Numerical Modeling, 1

Overlap Integrals, 194

Potential Energy Surface, 22

Repulsion, 88

Schrödinger Equation, 15
Static Correlation, 242

Two-Electron Repulsion Integrals, 206
Variational Principle, 36
Wavefunction-Based Theory, 50

# ประวัติผู้เขียน

รังสิมันต์ เกษแก้ว สำเร็จการศึกษาปริญญาตรี (พ.ศ. 2559) และปริญญาโท (พ.ศ. 2562) สาขาเคมี จากภาควิชาเคมี คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ ปัจจุบันกำลังศึกษาปริญญาเอกสาขาเคมีทฤษฎีที่ภาควิชาเคมี มหาวิทยาลัยแห่งซูริค ประเทศสวิตเซอร์แลนด์ หัวข้องานวิจัยที่สนใจ ได้แก่ เคมีควอนตัม เมตาไดนามิกส์ การถ่ายโอนอิเล็กตรอน ปัญญาประดิษฐ์ และการพัฒนาซอฟต์แวร์เคมีเชิงคำนวณ

ประสบการณ์การทำงาน

- พ.ศ. 2563 ที่ปรึกษาบริษัท New Equilibrium Biosciences, Boston, MA
- พ.ศ. 2564 ที่ปรึกษาบริษัท ติ๊งกิ้ง แมชชีนส์ จำกัด (Thinking Machines)
- พ.ศ. 2564 คณะกรรมการจัดการแข่งขันปัญญาประดิษฐ์สำหรับเคมีแห่งประเทศไทย (TMLCC)
- พ.ศ. 2564 คณะกรรมการจัดงาน PyCon Thailand และ PyCon APAC 2021
- พ.ศ. 2565 นักเขียนบทความบริษัท คลาวด์ เอชเอ็ม จำกัด (Cloud HM)

ผู้ก่อตั้งเพจและกลุ่ม Facebook

- วิทย์ตามิน
- Computational Chemistry and Machine Learning Thailand
- Thai Computational Science Students

ดูบทความและผลงานเพิ่มเติมของผู้เขียนได้ที่ https://rangsimanketkaew.github.io