การเขียนโปรแกรมกระบวนการไดอะกอนอลสำหรับเมทริกซ์ซ้อนเกยด้วยขั้นตอนวิธีเฮาส์โฮเดอร์

Implementation of diagonalization for overlap matrix via householder algorithm

โดย

นายณภัทร สิทธิมนต์ชัย

รายงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร

ปริญญาวิทยาศาสตรบัณฑิต

ภาควิชาเคมี คณะวิทยาศาสตร์

จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2557

เรื่อง การเขียนโปรแกรมกระบวนการไดอะกอนอลสำหรับเมทริกซ์ซ้อนเกยด้วยขั้นตอนวิธีเฮาส์โฮเดอร์

โดย นายณภัทร สิทธิมนต์ชัย

ได้รับอนุมัติให้เป็นส่วนหนึ่งของการศึกษา

ตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต ภาควิชาเคมี

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

คณะกรรมการสอบโครงการ

..... ประธานกรรมการ

(รองศาสตราจารย์ ดร.วุฒิชัย พาราสุข)

..... อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร.วิวัฒน์ วชิรวงศ์กวิน)

..... กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.บุษยรัตน์ ธรรมพัฒนกิจ)

รายงานฉบับนี้ได้รับความเห็นชอบและอนุมัติโดยหัวหนาภาควิชาเคมี

.....

(รองศาสตราจารย์ ดร.วุฒิชัย พาราสุข)

หัวหนาภาควิชาเคมี

วัน 21 เดือน พ.ค. พ.ศ. 2558

คุณภาพของการเขียนรายงานเล่มนี้อยู่ในระดับ ☐ ดีมาก ☐ ดี ☑ พอใช้

ชื่อโครงการ การเขียนโปรแกรมกระบวนการไดอะกอนอลสำหรับเมทริกซ์ซ้อนเกยด้วยขั้นตอนวิธีเฮาส์โฮเดอร์

ชื่อนิสิตในโครงการ นายณภัทร สิทธิมนต์ชัย รหัสประจำตัว 5433084023

อาจารย์ที่ปรึกษา รองศาสตราจารย์ ดร.วิวัฒน์ วชิรวงศ์กวิน

ภาควิชา เคมี คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2557

บทคัดย่อ

การคำนวณค่าไอเกนและไอเกนเวกเตอร์จากเมทริกซ์ซ้อนเกยของสมการชโรดิงเงอร์ทางเคมีควอนตัม เมทริกซ์ซ้อนเกยมีขนาดใหญ่และมีความซับซ้อนซึ่งขึ้นอยู่กับขนาดของระบบโมเลกุลและเบสิซเซตที่ศึกษา การแก้ปัญหาจึงต้องอาศัยกระบวณการไดอะโกนอไลเซชันที่มีประสิทธิภาพ เพื่อจะได้ค่าไอเกนและเวกเตอร์ไอเกนที่แม่นยำและถูกต้อง วิธีเฮาส์โฮเดอร์เป็นวิธีจัดการ เมทริกซ์ซ้อนเกยได้อย่างมีประสิทธิภาพและมีความแม่นยำในการคำนวณ ดังนั้น ผู้วิจัยจึงนำเฮาส์โฮเดอร์มาพัฒนาในรูปแบบการเขียนโปรแกรมด้วยภาษาซี ทั้งนี้ผลที่ได้จากการคำนวณจะนำไปเปรียบเทียบกับผลที่ได้จากโปรแกรม GAMESS ซึ่งเป็นโปรแกรมมาตรฐานทั่วไปที่นิยมใช้ ผลของโปรแกรมที่ทางผู้วิจัยได้พัฒนาเมื่อเปรียบเทียบผลที่ได้จากโปรแกรม GAMESS พบว่า ค่าไอเกนและไอเกนเวกเตอร์ที่ได้จากโปรแกรมที่พัฒนามีค่าไม่เท่ากันกับค่าจากโปรแกรม GAMESS เนื่องจาก โปรแกรม GAMESS ใช้วิธีการหาไอเกนเวกเตอร์จากไตรไดอะกอนอลที่ต่างจากโปรแกรมของผู้วิจัยได้สร้างขึ้น ซึ่งโปรแกรมยังอยู่ในระหว่างการพัฒนาคาดว่าเมื่อพัฒนาเสร็จสมบูรณ์ค่าที่ได้จะเทียบเท่ากับโปรแกรม GAMESS

คำสำคัญ: เมทริกซ์ซ้อนเกย, เฮาส์โฮเดอร์

Title Implementation of diagonalization for overlap matrix via householder algorithm

Student name Mr. Napat Sitthimonchai ID 5433084023

Advisor Assoc. Prof. Dr. Viwat Vchirawongkwin

Department of Chemistry, Faculty of Science, Chulalongkorn University, Academic year 2014

Abstract

Calculations of eigenvalues and eigenvectors are evaluated through overlap matrix from Schrödinger equation in quantum chemistry. The overlap matrix is complexity and large matrix depending on the stage of molecule and the selected basis set. This problem can be solved effectively by using a diagonalization method. The householder is a method that it is able to diagonalize the overlap matrix, which shows a high potential and accuracy for calculation. Thus, we selected the householder method, developed by C language. The results of program are eigenvalues and eigenvectors compared with those values obtained from the GAMESS package. The developed program and GAMESS have given the differences of eigenvalues and eigenvectors, according to the developed program has used different methods to calculate the results. The program is under the development stage at the moment. If the program succeeds, the results will be comparable with the GAMESS package.

Keyword: Overlap matrix, Householder

Acknowledgement

This senior project was successfully finished with the tremendous support from our senior project advisor, Assoc. Prof. Dr. Viwat Vchirawongkwin who always gives me his advice encouragement for my academic study.

Secondly, I would like to special thank Assoc. Prof. Dr. Vudhichai Parasuk and Assist. Prof. Dr. Boosayarat Tomapatanaget, who acted as the senior project exam committee members.

Thirdly, I heartily thank our internal project guide, Mr. Chokchai Pornpiganon, for his guidance and suggestions during this project work.

Finally, I would like to thank Department of Chemistry, Faculty of Science, Chulalongkorn University for the support of the infrastructure for the calculations.

Contents

List of Figures

List of Table

Chapter 1

Introduction

Molecules contain several atoms and generate the various molecular shapes. A basis set is a set of functions, which are combined in linear combinations to create the molecular orbitals. The geometry and basis set are explained with mathematics by the overlap matrix. If molecular shape changed, the elements of overlap matrix[1] are also modified.

Schrödinger equation[1] explains the moving of electrons in the molecular shape, but the is equation cannot be calculated in computer. Thus, we will use Roothaan-Hall [2] equation to solve on the computer program.

The standard procedure to solve the molecular problem in computational chemistry is the transformation the molecular space into the orthogonal space. A number of methods is diagonalization, e.g., Jacobi, householder[3], QR, LR and further. The diagonalization of each method shows an equality only eigenvalues.

Householder method was developed for a high effectiveness of diagonalization. This method shows the accuracy than other methods, because this one is unlimited size of matrix for calculation. The method can solve an overlap matrix having a large size. Moreover, householder method was developed into the effective method, which is the popular one at this moment. According to, the eigenvectors from householder method can increase the accuracy value.

The overlap matrix explains a molecular form in each element of the matrix. It is able to calculate the bonding, energy and further. The overlap matrix is a symmetric matrix, which is a square matrix [n × n]. It is a complicated problem for calculation. As a result of this problem, it needs a computer program to solve. Generally, the program is compiled through General Atomic and Molecular Electronic Structure System (GAMESS)[5]. GAMESS was a program for *ab initio* molecular quantum chemistry. The program can compute the method of approximation for the determination of the wavefunction and the energy, but the program is coded with FORTRAN language.

We are developing the program for computational chemistry, namely Molecular Orbital calculation with CUDA (MOCCA)[6] . MOCCA have been developed with C/C++ languageand utilize the parallel feature of the graphics processing unit (GPU). The process diagonalization of MOCCA use Jacobi for this time. So, my program of project work is wrote using C language and is prepared to implement on GPU in the future.

We are interested in the developing the program and calculation methods based on the householder effectively with C language. The method will be applied to MOCCA, when the program is successful.

Chapter 2

Theories and Programing

2.1 Electronic structure calculation

Electronic structure describes the motion of electrons in atoms or molecules. A molecular shape determines the molecular orbitals (MOs) or wavefunction, contracted from the atomic orbitals (AOs).

Schrödinger proposed the equation (eq. (2.1))

$$\hat{\mathrm{H}}\psi = \mathrm{E}\psi, \tag{2.1}$$

where $\hat{\mathrm{H}}$ is Hamiltonian operator, ψ is wavefunction, and E is the energy (eigenvalue) for the system. However, the equation is unable to calculate energy system directly. Thus, the equation needs to transformation for solving.

$$E = \frac{\int \psi \hat{\mathrm{H}} \psi dv}{\int \psi^2 dv}. \tag{2.2}$$

The variable, *dv*, indicates an integration with respect to the spatial coordinates (x, y, z, in Cartesian coordinate system), integrated over all of space is implied.

For example, two s orbitals (ϕ_1 and ϕ_2) of hydrogen molecule (H_2) are approximated by the linear combination of atomic orbital (LCAO) to be the molecular orbitals,

$$\psi = c_1\phi_1 + c_2\phi_2, \tag{2.3}$$

where c_1 and c_2 are coefficients, and ϕ_1 and ϕ_2 are basis set on each atom.

Eq. (2.3) takes the value, ψ, into eq. (2.2).

$$E = \frac{\int (c_1\phi_1 + c_2\phi_2)\hat{\mathrm{H}}(c_1\phi_1 + c_2\phi_2)dv}{\int (c_1\phi_1 + c_2\phi_2)^2 dv} \tag{2.4}$$

Eq. (2.4) is multiplied and changed variable into eq. (2.5)

$$E = \frac{c_1^2 H_{11} + 2c_1c_2H_{12} + c_2^2 H_{22}}{c_1^2 S_{11} + 2c_1c_2S_{12} + c_2^2 S_{22}}, \tag{2.5}$$

where

$$\int \phi_1 \hat{\mathrm{H}} \phi_1 dv = H_{11},$$

$$\int \phi_1 \hat{\mathrm{H}} \phi_2 dv = H_{12} = \int \phi_2 \hat{\mathrm{H}} \phi_1 dv = H_{21},$$

$$\int \phi_2 \hat{\mathrm{H}} \phi_2 dv = H_{22},$$

$$\int \phi_1^2 dv = S_{11},$$

$$\int \phi_1 \phi_2 dv = S_{12} = \int \phi_2 \phi_1 dv = S_{21},$$

$$\int \phi_2^2 dv = S_{22},$$

H_{ij} are not operators, but they are the integrals involving $\hat{\mathrm{H}}$ and basis functions. Eq. (2.5) is transformed to Roothaan-Hall equation (eq. (2.6)) that is a simple equation for solving.

$$\mathrm{HC} = \mathrm{SC}\varepsilon, \tag{2.6}$$

where the four matrix

$$\mathrm{H} = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix},$$

$$\mathrm{C} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix},$$

$$\mathrm{S} = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix},$$

$$\varepsilon = \begin{pmatrix} \varepsilon_1 & 0 \\ 0 & \varepsilon_2 \end{pmatrix}.$$

The H matrix is an energy-element matrix, called the Fock matrix. The coefficient matrix C contains elements of the weighting factors (c_{ij}), determining the extension of each basis function ϕ within each atomic orbital on an atom that contributes in each MO. The S matrix is the overlap matrix, whose elements are the overlap integral S_{ij} measuring of how well pair of basis function, atomic orbitals, overlap. Perfect overlap between the identical function on the same atom corresponds to $S_{ii} = 1$, while zero overlap between different functions on the same atom or well-separated functions on different atoms corresponds to $S_{ij} = 0$. The diagonal ε matrix is an energy-levels matrix, whose diagonal elements are the MO energy level, corresponding to the MOs.

The overlap matrix is a square matrix containing a matrix size of $\mathrm{n} \times \mathrm{n}$, where n is the number of atomic orbitals modulate by basis functions ,For example, the overlap matrix is fig. 2.1.

$$\begin{bmatrix} 1 & 0.241137 & 0 & 0 & 0 \\ 0.241137 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{5\times5}$$

Figure 2.1 Example form of overlap matrix for Li^+ with STO-3G basis set.

Overlap matrix can use to calculate the energy, bonding and further, which are showed on the eigenvalues and eigenvectors from the diagonalization matrix. But, eq. (2.6) is unable to calculate directly, it needs the transformation process or orthogonalization to eq. (2.10) form.

Eq. (2.7) to eq.(2.10) show the Löwdin transformation procedure for eq. (2.6) into eq. (2.10). The definition of a matrix C' is

$$C' = S^{\frac{1}{2}}C \quad \text{i.e. } C = S^{-\frac{1}{2}}C'. \tag{2.7}$$

Substituting $C = S^{-\frac{1}{2}}C'$ from eq. (2.7) into eq. (2.6), getting

$$S^{-\frac{1}{2}}HS^{-\frac{1}{2}}C' = S^{-\frac{1}{2}}SS^{-\frac{1}{2}}C'\varepsilon. \tag{2.8}$$

Let

$$S^{-\frac{1}{2}}HS^{-\frac{1}{2}} = H' \tag{2.9}$$

and note that $S^{-\frac{1}{2}}SS^{-\frac{1}{2}} = S^{-\frac{1}{2}}S^{-\frac{1}{2}} = 1$. Finally, the eq. (2.6) is transformed into the orthogonal space as

$$H'C' = C'\varepsilon\,, \tag{2.10}$$

where H' is the Fock matrix in orthogonal dimension, C' is the matrix of coefficients c', and ε is eigenvalue matrix.

The orthogonalizing matrix $S^{-\frac{1}{2}}$ is calculated from S. The symmetric orthogonalization treats all the wave functions on an equal footing.

The diagonalization in eq. (2.10) is called orthogonalization, since the result is to make the basis functions orthogonal. The orthogonaolization is important for solving Roothaan-Hall equations so the diagonalization is the solution of process.

2.2 Diagonalization

Diagonalization is a procedure of matrix transformation from one space to orthogonal space. All of vectors in orthogonal space arrange in perpendicular dimension. We want the solution of the Schrödinger equation that appropriates to our particular problem. An orthogonal matrix A can be written $A = PDP^{-1}$ where D is a diagonal matrix. The process of finding P and D is diagonalization.

$$Ax = \lambda x \tag{2.11}$$

The eq. (2.11) is solved with normal calculation that is inaccurate, highly slow, yielding only eigenvalues but not eigenvectors. Diagonalization is crucial for this process. The methodology of diagonalization has several suggestions, for example Jacobi, householder, QR, LR method and etc. Finally, the diagonalization results to the eigenvalues and eigenvectors. However, each diagonalization method shows the equivalence of eigenvalues, but the eigenvectors vary by each method.

2.3 Householder method

Alston Scoot Householder created a diagonalization method where is called householder method[7-8]. The householder method is able to calculate a symmetric and non-symmetric matrix. The overlap matrix is a symmetric matrix, then used the symmetric method.

The principle of householder method for symmetric matrix, where 4 x 4 overlap matrix is showed in eq. (2.12) to eq. (2.24)

Therefore, the equation can solve, which is eq. (2.12). The equation form of householder matrix is used

$$Q = I - \frac{uu^T}{H}, \tag{2.12}$$

where Q is a householder matrix, I is an identity matrix, u is a vector, and

$$H = \frac{1}{2}u^T u = \frac{1}{2}|u|^2. \tag{2.13}$$

The uu^T in eq. (2.13) is outer product that is a matrix with the elements $(uu^T)_{ij} = u_i u_j$. The matrix is an orthogonal matrix where symmetric ($Q^T = Q$). So,

$$\begin{aligned} Q^T Q = QQ &= \left(I - \frac{uu^T}{H}\right)\left(I - \frac{uu^T}{H}\right) \\ &= I - 2\frac{uu^T}{H} + \frac{u(u^T u)u^T}{H^2} \\ &= I - 2\frac{uu^T}{H} + \frac{u(2H)u^T}{H^2} \\ &= I \end{aligned}$$

This shows that Q is also an orthogonal matrix.

Let x be an arbitrary vector and consider the transformation Qx. Choosing

$$u = x + ke_1 \tag{2.14}$$

where $k = \pm|x|,\ e_1 = [1\ \ 0\ \ 0 \ldots\ \ 0]^T$

$$\begin{aligned} Qx &= \left(I - \frac{uu^T}{H}\right)x = \left[I - \frac{u(x+ke_1)^T}{H}\right]x \\ &= x - \frac{u(x^T x + ke_1^T x)}{H} = x - \frac{u(k^2 + kx_1)}{H}, \end{aligned} \tag{2.15}$$

but

$$\begin{aligned} 2H &= (x + ke_1)^T(x + ke_1) = |x|^2 + k(x^T e_1 + e_1^T x) + k^2 e_1^T e_1 \\ &= k^2 + 2kx_1 + k^2 = 2(k^2 + kx) \end{aligned} \tag{2.16}$$

So,

$$Qx = x - u = -ke_1 = [-k\ \ 0\ \ 0\ \ \ldots 0]^T. \tag{2.17}$$

Householder Reduction

A symmetric $n \times n$ matrix is applied by the following transformation.

$$PA = \begin{bmatrix} 1 & 0^T \\ 0 & Q \end{bmatrix} \begin{bmatrix} A_{11} & x^T \\ x & A' \end{bmatrix} = \begin{bmatrix} A_{11} & x^T \\ Qx & QA' \end{bmatrix} \tag{2.18}$$

The first column and row of matrix A (or A_{11}) does not reduce in this step. The vector x represents the first column of the matrix A with A_{11}. A' is an element of the matrix A, but it is omitted the first column and row element. The matrix Q of dimensions $(n-1) \times (n-1)$ is using eq. (2.13) - (2.15) into eq. (2.19). The transformation can reduce the first column of the matrix A.

$$\begin{bmatrix} A_{11} \\ Qx \end{bmatrix} = \begin{bmatrix} A_{11} \\ -k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$A \leftarrow P_1 A P_1 = \begin{bmatrix} A_{11} & (Qx)^T \\ Qx & QA'Q \end{bmatrix} \tag{2.19}$$

The product is the tridiagonalize matrix in the first column and row. Fig. 2.2 show a diagram of the first step transformation for 4 x 4 matrix.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & Q & \\ 0 & & & \end{bmatrix} \cdot \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & & & \\ A_{31} & & A' & \\ A_{41} & & & \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & Q & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} A_{11} & -k & 0 & 0 \\ -k & & & \\ 0 & & QA'Q & \\ 0 & & & \end{bmatrix}$$

Figure 2.2 A diagram of the first step transformation for 4 x 4 matrix.

The second step, the second column and row of the new matrix A is reduced as the transformation for 3×3 matrix.

$$A \leftarrow P_2 A P_2,$$

where

$$P_2 = \begin{bmatrix} I_2 & 0^T \\ 0 & Q \end{bmatrix} \tag{2.20}$$

that I_2 is a 2×2 identity matrix.Q is a $(n-2) \times (n-2)$ matrix constructed by choosing for x the bottom $n - 2$ elements of the second column of A. The total of transformation is taken with since $n - 2$.

$$P_i = \begin{bmatrix} I_i & 0^T \\ 0 & Q \end{bmatrix} ; \qquad i = 1, 2, \dots, n-2$$

So it attains the tridiagonal form.

P_i is an extravagant form when it is the multiplication matrix of P_iAP process. It is convenient to transform.

$$A'Q = A' \cdot \left(I - \frac{uu^T}{H}\right) = A' - \frac{A'u}{H}u^T = A' - vu^T,$$

where $v = \frac{A'u}{H}$. (2.21)

Therefore,

$$QA'Q = \left(I - \frac{uu^T}{H}\right)(A' - vu^T) = A' - vu^T - \frac{uu^T}{H}(A' - vu^T)$$

$$= A' - vu^T - \frac{u(u^TA')}{H} + \frac{u(u^Tv)u^T}{H}$$

$$= A' - vu^T - uv^T + 2guu^T,$$

where $g = \frac{u^Tv}{2H}$. (2.22)

Letting

$$w = v - gu \tag{2.23}$$

$$QA'Q = A' - wu^T - uw^T, \tag{2.24}$$

which is carried on $i = 1, 2 \ldots n - 2$.

2.4 Overview program

Fig. 2.3 show the flowchart of the developed program.

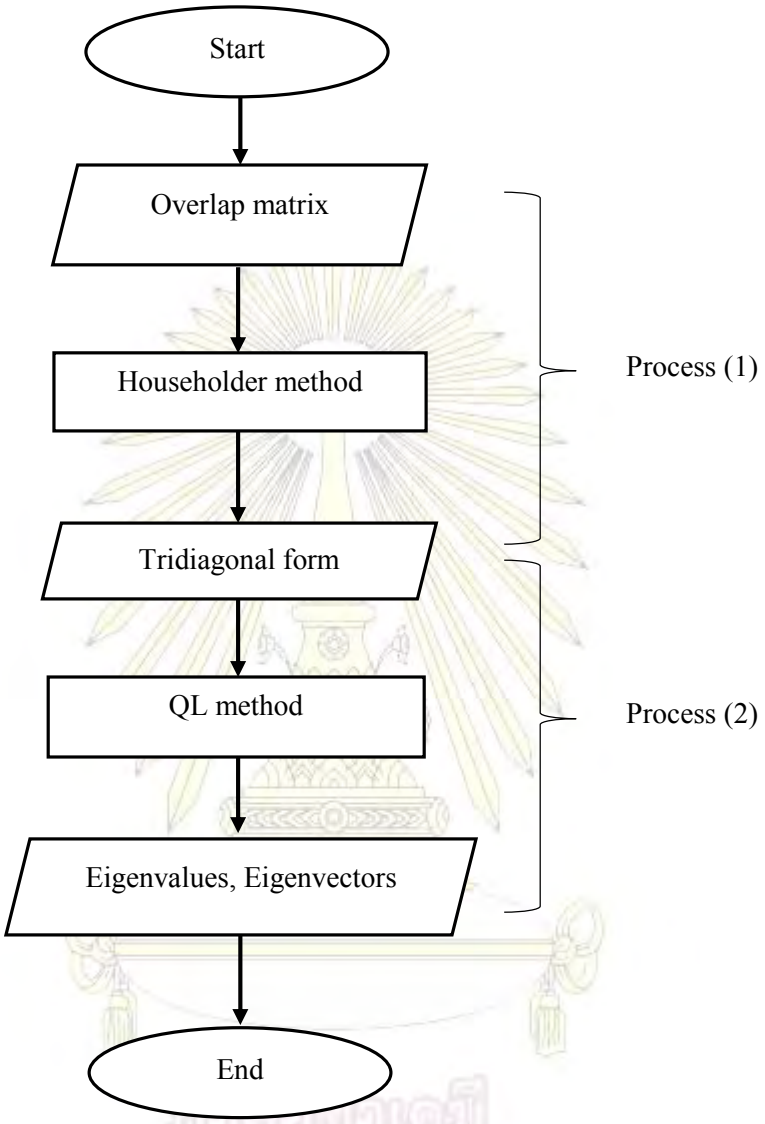

Figure 2.3 show the process of our program.

The program consists of two processes. Firstly, the process starts when the overlap matrix is an input of the program. The overlap matrix is calculated by householder method, producing tridiagonal form. Secondly, the process reduces tridiagonal form to eigenvalues and eigenvectors with QL method. Finally, the product, which is eigenvalue and eigenvector, compares with those value of GAMESS program.

Chapter 3

Results and Discussions

In our experiment, the matrix that use in calculation is the carbon monoxide (CO) molecule's matrix that was calculated using STO-3G basis set. The geometry of the CO molecule is shown in Table 3.1. The overlap matrix of the system is calculated by GAMESS program, presented in Fig 3.1.

Atom	Atomic number	x(Å)	y(Å)	z(Å)
C	6.00	0.00	0.00	0.00
O	8.00	0.00	0.00	1.20

Table 3.1 The coordinate of tested carbon and oxygen atom listed in Angstroms (Å).

$$\begin{pmatrix}
1.000000 & 0.248362 & 0.000000 & 0.000000 & 0.000000 & 0.000002 & 0.037781 & 0.000000 & 0.000000 & -0.063250 \\
0.248362 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.037893 & 0.370442 & 0.000000 & 0.000000 & -0.325482 \\
0.000000 & 0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.215964 & 0.000000 & 0.000000 \\
0.000000 & 0.000000 & 0.000000 & 1.000000 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 0.215964 & 0.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & 0.063850 & 0.448480 & 0.000000 & 0.000000 & -0.313655 \\
0.000002 & 0.037893 & 0.000000 & 0.000000 & 0.063850 & 1.000000 & 0.236704 & 0.000000 & 0.000000 & 0.000000 \\
0.037781 & 0.370442 & 0.000000 & 0.000000 & 0.448480 & 0.236704 & 1.000000 & 0.000000 & 0.000000 & 0.000000 \\
0.000000 & 0.000000 & 0.215964 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & 0.000000 & 0.000000 \\
0.000000 & 0.000000 & 0.000000 & 0.215964 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000 & 0.000000 \\
-0.063250 & -0.325482 & 0.000000 & 0.000000 & -0.313655 & 0.000000 & 0.000000 & 0.000000 & 0.000000 & 1.000000
\end{pmatrix}$$

Figure. 3.1 The 10 × 10 matrix of CO with STO-3G basis set for testing the calculation.

Number	1	2	3	4	5
Eigenvalue	1.087015	1.066760	0.974941	0.840045	0.868850
Eigenvector	-0.892754	0.378601	0.224913	-0.227167	0.122886
	0.109069	-0.181245	-0.383595	0.734274	-0.358258
	0.021151	-0.075257	0.098939	-0.432913	0.198831
	0.201020	0.572951	0.533855	0.681842	-0.199386
	-0.062165	-0.307163	0.606297	0.378358	0.018181
	0.046397	0.214001	-0.369018	-0.196580	0.083630
	-0.079831	-0.315840	0.347496	0.174081	0.007737
	0.029287	0.116211	-0.133071	-0.036073	0.055528
	0.010158	0.034604	0.002221	-0.153021	-0.300958
	-0.030438	-0.126515	0.218280	-0.333866	-0.837031

Number	6	7	8	9	10
Eigenvalue	0.608067	0.393984	1.312259	1.443475	-0.045337
Eigenvector	0.023828	0.094353	0.001515	0.000210	-0.000024
	-0.136380	-0.753471	0.001769	0.000570	0.000308
	0.076726	0.245194	-0.015962	-0.003008	0.000111
	-0.247610	0.294223	0.002570	-0.005439	-0.004148
	-0.316940	0.172692	-0.378259	-0.106634	0.000590
	-0.616858	0.024750	0.522432	0.162739	-0.060706
	0.403404	-0.018104	0.506673	0.351786	-0.040218
	-0.137789	0.007092	-0.187679	-0.137216	0.537891
	-0.103571	0.004751	-0.26106	0.557104	0.498334
	-0.078891	0.002168	0.276726	-0.447045	0.117484

Figure 3.2 Eigenvalues and eigenvectors of the matrix of CO are calculated on our program.

Number	1	2	3	4	5
Eigenvalue	0.248521	0.784036	0.784036	0.795263	0.850769
Eigenvector	-0.096994	0.000000	0.000000	-0.652124	-0.306810
	0.484598	0.000000	0.000000	0.523604	0.108449
	0.000000	-0.686969	0.167552	0.000000	0.000000
	0.000000	-0.167552	-0.686969	0.000000	0.000000
	0.501233	0.000000	0.000000	-0.339859	-0.292197
	0.112022	0.000000	0.000000	-0.310579	0.654118
	-0.568426	0.000000	0.000000	0.276493	-0.350932
	0.000000	0.686969	-0.167552	0.000000	0.000000
	0.000000	0.167552	0.686969	0.000000	0.000000
	0.410933	0.000000	0.000000	0.110279	-0.507646
Number	6	7	8	9	10
Eigenvalue	1.076710	1.215434	1.215964	1.215964	1.813304
Eigenvector	0.358628	0.549158	0.000000	0.000000	0.202497
	0.216537	0.449637	0.000000	0.000000	0.479772
	0.000000	0.000000	0.702890	-0.077110	0.000000
	0.000000	0.000000	-0.077110	-0.702890	0.000000
	-0.434693	-0.368582	0.000000	0.000000	0.472304
	0.507558	-0.396243	0.000000	0.000000	0.220224
	0.247077	-0.333197	0.000000	0.000000	0.552469
	0.000000	0.000000	0.702890	-0.077110	0.000000
	0.000000	0.000000	-0.077110	-0.702890	0.000000
	0.562919	-0.303925	0.000000	0.000000	-0.389899

Figure. 3.3 Eigenvalues and eigenvectors of the matrix of CO are calculated on GAMESS.

Comparisons of eigenvalues of fig. (3.2) with fig. (3.3) are unequal. The problem is caused by:

1. One of the householder process calculates $\frac{uu^T}{H}$, when $uu_{ij}^T = 0$ and $H = 0$, producing some undefined values.This case $\frac{\mathrm{uu^T}}{\mathrm{H}} = \frac{0}{0}$ is changed $\frac{\mathrm{uu^T}}{\mathrm{H}} = 0$ because the next step is eq. (3.1). If $\frac{\mathrm{uu^T}}{\mathrm{H}}$ is zero, then Q can be calculated in this form.

$$\mathrm{Q} = \mathrm{I} - \frac{\mathrm{uu^T}}{\mathrm{H}} \tag{3.1}$$

2. The methods of our program and GAMESS are differences. Householder method have 4 sub-methods that are tred 1, tred 2, tred 3 and tred 4. The sub-methods transform the directly process of householder. GAMESS use one of sub-methods, but our program directly implements the householder method from the theory. Unfortunately, the results are differences.

Chapter 4
Conclusion

A program is developed to diagonalize the overlap matrix via householder method, calculating for eigenvalues and eigenvectors. Unfortunately, the program is unsuccessful. There are two problems. The calculation of $\frac{uu^T}{H}$ within the householder process, when the value $uu^T = 0$ and $H = 0$. Then, the method in the developed program reduces the tridiagonal to eigenvalues with QL method deferring to GAMESS.

The correction of implementation has been needed to give the equivalent results with GAMESS program. The understanding of householder method is the key to developing the parallel version, computing on graphics processing unit.

Appendix

1. Data in STO-3G, basis set of carbon and oxygen atom.

```
C  STO-3G
S   3
  1     71.6168370              0.15432897
  2     13.0450960              0.53532814
  3      3.5305122              0.44463454
L   3
  1      2.9412494             -0.09996723              0.15591627
  2      0.6834831              0.39951283              0.60768372
  3      0.2222899              0.70011547              0.39195739

O STO-3G
S   3
  1    130.7093200            0.15432897
  2     23.8088610             0.53532814
  3      6.4436083              0.44463454
L   3
  1      5.0331513             -0.09996723              0.15591627
  2      1.1695961              0.39951283              0.60768372
  3      0.3803890              0.70011547              0.39195739
```

2. Source code of householder method

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
void house (int r,float **A,float d[],float c[]){
      int i=0,j=0,s=0,p=0,n=r;
      float k=0,H=0,uMag=0;
      float Aa[r-1][r-1], u[r-1], uu[r-1][r-1],Q[r-1][r-1], QAQ[r][r], QA[r-
1][r-1], I[r-1][r-1];
      for(p=0;p<(r-2);p++){
            printf("p = %d\n",p);
            for(i=0;i<(r-1);i++){
                        u[i]=A[p][i+1+p];
            }
            for(i=0;i<r-1-p;i++){
                  for(j=0;j<r-1-p;j++){
                        I[i][j]=0;
                  }
            }
      for(i=0;i<r-1-p;i++){
            I[i][i]=1;
      }//identitymatrix
      for(i=0;i<r-1-p;i++){
            for(j=0;j<r-1-p;j++){
```

```
                Aa[i][j]=A[i+1+p][j+1+p];
            }
        }
        printf("\n");
        for(i=0;i<r-1-p;i++){
            k = (pow(u[i],2)) + k;
            }
        k = sqrt(k);
        if(u[0] < 0.0){
            k = -(k);
        }
        u[0] = u[0]+k;
        H=0;
        for(i=0;i<r-1-p;i++){
            H= (pow(u[i],2))+ H;
        }
        H = pow(H,0.5);
        H = pow(H,2);
        H = H*0.5;
        for(i=0;i<r-1-p;i++){
            for(j=0;j<r-1-p;j++){
                uu[i][j] = (u[j]*u[i]);
                if(i!=j){
                    uu[j][i] = uu[i][j];
                }
            }
        }
        for(i=0;i<r-1-p;i++){
                for(j=0;j<r-1-p;j++){
                if(uu[i][j]!=0.0 && H !=0.0){    //check
                    uu[i][j]=uu[i][j]/H;
                }
                else break;
                }
        }
        for(i=0;i<r-1-p;i++){
            for(j=0;j<r-1-p;j++){
                Q[i][j] = I[i][j] - uu[i][j];
            }
        }
        printf("Q\n");
        for(i=0;i<r-1-p;i++){
            printf("[");
            for(j=0;j<r-1-p;j++){
                printf("%f\t",Q[i][j]);
            }printf("]\n");
        }
        for (i=0;i<r;i++){                //QA and QAQ set zero.
            for(j=0;j<r;j++){
                QA[i][j] = 0.0;
```

```
                QAQ[i][j] = 0.0;
            }
        }
        k = -k;
        QAQ[0][0] = A[p][p];
        QAQ[0][1] = k;
        QAQ[1][0] = k;
        printf("QAQ\n");
        for(i=0;i<r-p;i++){
            printf("[");
            for(j=0;j<r-p;j++){
                printf("%f\t", QAQ[i][j]);
            }printf("]\n");
        }
        printf("Aa\n");
        for(i=0;i<r-1-p;i++){
                        printf("[");
                for(j=0;j<r-1-p;j++){
                        printf("%f\t",Aa[i][j]);
                }printf ("]\n");
        }
        for(i=0;i<r-1-p;i++){
            for(j=0;j<r-1-p;j++){
                for(s=0;s<r-1-p;s++){
                        QA[i][j]=(Q[i][s]*Aa[s][j])+(QA[i][j]);
                }
            }
        }
        for(i=0;i<r-1-p;i++){
                for(j=0;j<r-1-p;j++){
                                for(s=0;s<r-1-p;s++){

        QAQ[i+1][j+1]=(QA[i][s]*Q[s][j])+QAQ[i+1][j+1];
                        }
            }
        }
        printf("QAQ\n");
            for(i=0;i<r-p;i++){
                printf("[");
                for(j=0;j<r-p;j++){
                        printf("%f\t", QAQ[i][j]);
                }printf("]\n");
        }
        printf("\n");
        for(i=0;i<r-p;i++){
            for(j=0;j<r-p;j++){
                A[i+p][j+p]=QAQ[i][j];
            }
        }
        printf("A\n");
```

```
        for(i=0;i<r;i++){
              printf("[");
              for(j=0;j<r;j++){
                    printf("%f\t",A[i][j]);
              }printf("]\n");
        }
        k=0;
        H=0;
        }//end loop
        printf("\nd = [");
        for(i=0;i<n;i++){
              d[i] = A[i][i];
              printf("%f\t",d[i]);
        }
        printf("]\n\n");
        printf("c = [");
        for(i=0;i<n-1;i++){
              c[i] = A[i][i+1];
              printf("%f\t",c[i]);
        }
        printf("]\n");
}
```

Line 1-8: The include files are used and declare a type of variables.

Line 9: The main loop of program when end loop (line132) since p < (r-2) (or size of matrix minus one).

Line 10: The program checks by printing the value of p.

Line 11-13: The process inputs the value u [i] from the top row of real matrix (A[p][i+1+p]). The size matrix u is r-1.

Line 14-21: The process set the variable I that it is an identity matrix.

Line 22- 26: The process set value of Aa (A') from the real matrix (A).

Line 28-34: The process calculates k, which is square root of summation all value u ($k = \pm|u|$). Then, if the value of u[0] < 0.0 then k = -k.

Line 35: The process inputs value of u[0]+k to u[0].

Line 36: The variable H is set to zero.

Line 37-42: The process calculates H by $H = \frac{1}{2}|u|^2$.

Line 43-50: The process calculates the matrix uu (uu^T) with multiply matrix u and matrix u^T (matrix transpose).

Line 51-58: The process calculates $\frac{uu^T}{H}$ with uu is devised by H.

Line 59-70: The process calculates matrix Q by $I - \frac{uu^T}{H}$ and print it.

Line 71-76: The process set zero the matrix QA and QAQ.

Line 77-80: The process input value A_{11} into QAQ_{11} and k into QAQ_{12} and QAQ_{21}.

Line 81-94: The program checks by printing the values of matrix QAQ and Aa.

Line 95-101: The process calculates the matrix QA with matrix Q multiply matrix Aa.

Line 102-109: The process calculates the matrix QAQ with matrix QA multiply matrix Q.

Line 110-117: The program checks by printing the matrix QAQ.

Line 118-122: The values of matrix QAQ input to matrix A.

Line 123-129: The program checks by printing the matrix A.

Line 130-131: The process resets the variable k and H to zero before the next loop.

Line 132: This line is the end loop of the method.

Line 133-138: The process inputs diagonal of A into d and print.

Line 139-145: The process inputs subdiagonal of A into c and print.

2. Source code of QL method

QL method is a process of calculation the tridiagonal form to eigenvalues and eigenvectors. L is the lower triangular matrix (the *left* part) that includes the diagonal, and Q is a rotation matrix. QL method

is a favorite to use after QR method that called QR and QL method[4]. The method can use tridiagonal form of householder method because it uses diagonal, subdiagonal and real matrix A.

```
#include <math.h>
#define NRANSI
#include "nrutil.h"

float pythag(float a, float b){             //code for function pythagoras
        float absa,absb;
        absa=fabs(a);
        absb=fabs(b);
        if (absa > absb) return absa*sqrt(1.0+SQR(absb/absa));
        else return (absb == 0.0 ? 0.0 : absb*sqrt(1.0+SQR(absa/absb)));
}

void tqli(float d[], float e[], int n, float **z){
  int m,l,iter,i,k;
  float s,r,p,g,f,dd,c,b;
  for (i=2;i<=n;i++) e[i-1]=e[i];
  e[n]=0.0;
      for (l=1;l<=n;l++) {
            iter=0;
            do {
                  for (m=l;m<=n-1;m++) {
                        dd=fabs(d[m])+fabs(d[m+1]);
                        if ((float)(fabs(e[m])+dd) == dd) break;
                        }
                        if (m != l) {
                        if(iter++ == 30)nrerror("Too many iterations in tqli");
                                    g=(d[l+1]-d[l])/(2.0*e[l]);
                                    r=pythag(g,1.0);
                                    g=d[m]-d[l]+e[l]/(g+SIGN(r,g));
                                    s=c=1.0;
                                    p=0.0;
                                    for (i=m-1;i>=l;i--) {
                                          f=s*e[i];
                                          b=c*e[i];
                                          e[i+1]=(r=pythag(f,g));
                                          if (r == 0.0) {
                                                d[i+1] -= p;
                                                e[m]=0.0;
                                                break;
                                          }
                                         s=f/r;
                                         c=g/r;
                                         g=d[i+1]-p;
                                         r=(d[i]-g)*s+2.0*c*b;
                                         d[i+1]=g+(p=s*r);
                                          g=c*r-b;
                                          for (k=1;k<=n;k++) {
```

```
                                                    f=z[k][i+1];
                                                    z[k][i+1]=s*z[k][i]+c*f;
                                                    z[k][i]=c*z[k][i]-s*f;
                                            }
                                    }
                                    if (r == 0.0 && i >= l) continue;
                                    d[l] -= p;
                                    e[l]=g;
                                    e[m]=0.0;
                            }
                    } while (m != l);
            }
    printf("QL Pass");    //check QL method
    }
    #undef NRANSI
```

3. Source code of diver run for householder and QL method.

```
#include <stdio.h>
#include <math.h>
#define NRANSI
#include "nr.h"
#include "nrutil.h"
#include "test.h"
#include "QL2.h"
#define TINY 1.0e-6

#define n 10                //    edit size of matrix
int main () {
      int i,j,k;
      float *p,*d,*q,*e,*f,**a,**b;
      static float c[n][n] = {                          //edit   matrix   for
calculate
    {1.00000000000000022204,     0.24836239031011142497,
0.00000000000000000000,  0.00000000000000000000,    -0.00000000000000012371,
       0.00000150822730372276,    0.03778103087794688897,
      0.00000000000000000000,          0.00000000000000000000,          -
0.06325041378995543973},
    {0.24836239031011142497,     1.00000000000000044409,
0.00000000000000000000,  0.00000000000000000000,    -0.00000000000000019096,
0.03789346793677286079,    0.37044193335519737253,
      0.00000000000000000000,          0.00000000000000000000,          -
0.32548200232478652349},
    {0.00000000000000000000,      0.00000000000000000000,
1.00000000000000088818,  0.00000000000000000000,     0.00000000000000000000,
      0.00000000000000000000,          0.00000000000000000000,
      0.21596354654766608538,          0.00000000000000000000,
      0.00000000000000000000},
    {0.00000000000000000000,     0.00000000000000000000,
0.00000000000000000000,   1.00000000000000088818,   0.00000000000000000000,
      0.00000000000000000000,          0.00000000000000000000,
      0.00000000000000000000,    0.21596354654766608538,
      0.00000000000000000000},
    {-0.00000000000000012371,    -0.00000000000000019096,
0.00000000000000000000,    0.00000000000000000000,  1.00000000000000088818,
      0.06385039699942257618,          0.44848024226430266426,
      0.00000000000000000000,    0.00000000000000000000,          -
0.31365485536680176581},
    {0.00000150822730372276,     0.03789346793677286079,
0.00000000000000000000,   0.00000000000000000000,   0.06385039699942257618,
1.00000000000000022204,    0.23670393651084761788,
      0.00000000000000000000,          0.00000000000000000000,          -
0.00000000000000005734},
    {0.03778103087794688897,     0.37044193335519737253,
0.00000000000000000000,   0.00000000000000000000,   0.44848024226430266426,
```

```
0.23670393651084761788,   1.00000000000000022204,0.00000000000000000000,
            0.00000000000000000000,          0.00000000000000013628},
    {0.00000000000000000000,     0.00000000000000000000,
0.21596354654766608538,   0.00000000000000000000,   0.00000000000000000000,
0.00000000000000000000,   0.00000000000000000000,   1.00000000000000066613,
            0.00000000000000000000,          0.00000000000000000000},
    {0.00000000000000000000,     0.00000000000000000000,
0.00000000000000000000,   0.21596354654766608538,   0.00000000000000000000,
0.00000000000000000000,   0.00000000000000000000,   0.00000000000000000000,
1.00000000000000066613,    0.00000000000000000000},
    {-0.06325041378995543973,   -0.32548200232478652349,
0.00000000000000000000,    0.00000000000000000000, -0.31365485536680176581,
-0.00000000000000005734,   0.00000000000000013628,   0.00000000000000000000,
0.00000000000000000000,   1.00000000000000066613}};
      d = vector(1,n);
      e = vector(1,n);
      p = vector(1,n);
      q = vector(1,n);
      f = vector(1,n);
      a = matrix(0,n-1,0,n-1);
      b = matrix(1,n,1,n);
      for(i=0;i<n;i++){
            for(j=0;j<n;j++){
                  a[i][j] = c[i][j];
            }
      }
      house(n,a,d,e);       //householder process
      printf("House ok\n"); //check householder
      printf("a\n");
      for(i=0;i<=n-1;i++){
            printf("[");
            for(j=0;j<=n-1;j++){
                  printf("%f\t",a[i][j]);
            }printf("]\n");
      }
      printf("e=\n[");
      for(i=0;i<=n;i++){
                  printf("%f\t",e[i]);
                  }printf("]\n");
      for(i=0;i<=n;i++){
            p[i]=e[i-1];
      }
      printf("p=\n[");
            for(i=0;i<n;i++){
                  printf("%f\t",p[i]);
            }printf("]\n");
      for(i=0;i<=n;i++){                                  //reverse subdiagonal
into e
            if(i==0/*||i==1*/){
                  e[i]=0.0;
```

```
            }
            else e[i+1]=p[n-i];
        }                //
        printf("\n p(reverse p )=\n[");
        for(i=0;i<=n+1;i++){
            printf("%f\t",e[i]);
        }printf("]\n");
        for(i=0;i<=n;i++){                    //reverse diagonal from d to q
                if(i==0){
            q[i]=0.0;
                }
                else q[i]=d[n-i];
             }
            printf("\n q(reverse d )=\n[");
            for(i=0;i<=n;i++){
                printf("%f\t",q[i]);
            }printf("]\n");
        for(i=0;i<=n;i++){                                  //input q to d
            d[i]=q[i];
        }
        for(i=1;i<=n;i++){
            for(j=1;j<=n;j++){
                b[i][j]=a[i-1][j-1];
            }
        }
        tqli(d,e,n,b);                    //QL method where a is changed to b
        printf("b\n");
            for(i=1;i<=n;i++){
                printf("[");
                for(j=1;j<=n;j++){
                    printf("%f\t",b[i][j]);
                }printf("]\n");
            }
        printf("\nEigenvectors for a real symmettic matrix\n");
        for(i=1;i<=n;i++) {
            for(j=1;j<=n;j++){
                f[j]= 0.0;
                for(k=1;k<=n;k++){
                    f[j] += (c[j-1][k-1]*b[k][i]);
                }
            }
            printf("%s %3d %s %10.6f\n","eigenvalue",i," =",d[i]);
        //print eigenvalue
            printf("%11s     %14s     %9s\n","vector","mtrx*vect.","ratio");
        //print eigenvector
            for(j=1;j<=n;j++){
                if(fabs(b[j][i]) < TINY)
                    printf("%12.6f %12.6f %12s\n",
                        b[j][i],f[j],"div. by 0");
                else
```

```
                        printf("%12.6f %12.6f %12.6f\n",
                              b[j][i],f[j],f[j]/b[j][i]);
             }
             printf("Press Enter to continue...\n");
              (void) getchar();
       }
       free_matrix(b,1,n,1,n);
       free_matrix(a,0,n-1,0,n-1);
       free_vector(f,1,n);
       free_vector(e,1,n);
       free_vector(d,1,n);
return 0;
}
```

Bibliography

1. IRA N.LEVINE. *Quantum Chemistry*, Prentice-Hall: Brooklyn, New York; 1991; Vol.4.

2. Lewars, G.E. *Computational Chemistry*, Springer: London, New York; 2011; Vol.2; p.156-157.

3. Kiusalaas J. *Numerical methods in engineering with python*, Cambridge University press: New York; 2005; p.357-363.

4. Press, W.H.; Teukolsk, S.A.; Vetterling, W.T.; Flannery, B.P. *Numerical Recipes in C the art of scientific computing*, Cambridge University press: New York; 1992; Vol.2; p.469-481.

5. Schmidt, M.W.; Baldridge, K.K.; Boatz, J.A.; Elbert, S.T.; Gordon, M.S.; Jensen, J.H.; Koseki, S.; Matsunaga, N.; Nguyen, K.A.; Su, S.; Windus, T.L.; Dupuis, M.; Montgomery J.A. General Atomic and Molecular Electronic Structure System, *Comput. Chem*., 1993, *14*, 1347-1363.

6. Pornpiganon, C., 2014, *Program development for molecular orbital calculations with CUDA (MOCCA)*, Master's Thesis, Chulalongkorn University.

7. Housholder A.S., Unitary triangularization of a nonsysmetric matrix, *J. Assoc. Comp*., 1958, 339-342.

8. Aukenthaler,T.; Blum,V.; Bungartz, H-J.; Huckle, T.; Johanni, R.; Krämer, L.; Lang, B.; Lederer, H.; Willems, P.R., Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, *Parallel Comput.*, 2011, *37*, 783-794

Vitae

My name is Napat Sitthimonchai. I was born in Saraburi on the 25th of May 1992. I was graduated from secondary school at Princess Chulabhorn's College Pathumthani, Pathumthani.
I was a Bachelor's student at Department of Chemistry, Faculty of Science, Chulalongkorn University, Bangkok during 2011 to 2014. I resived scholarship for the best activists from The Government Pharmaceutical Organization (GPO) in 2013. I ever did many of University's activities such as vice president of academic of associate for factory of science in 2012, vice president of CU Academic Expo in 2012 and CU-TU traditional football in 2012. My address is 393 Moo.10 SuwannaShorn Road, NongSuang, Wihandang, Saraburi, 18150.