Eötvös Loránd Tudományegyetem
Informatikai Kar
Információs Rendszerek Tanszék

Developing software calculating quantum systems' wave functions

János Márk Szalai-Gindl
Egyetemi tanársegéd

Antal Száva
Programtervező Informatikus BSc

Budapest, 2017

Contents

Chapter 1

Introduction

1.1 Scientific background

In quantum mechanics, apart from classical characteristics, particles can be described by waves too. The specific location and momentum of particles, like the electrons in a molecule, cannot be determined simultaneously at a given time (Heisenberg's uncertainty principle). Only the probability of possible locations can be given. Instead of the classical model of electrons revolving around the nucleus of a given atom, in this probabilistic representation electrons are desribed by wave functions. The most likely locations for electrons are described by the concept of atomic orbitals, spherical distributions around the nucleus of an atom. The various atomic orbitals can have different energy levels depending on the type of the atom, the orbitals distance from the nucleus and its symmetry.

To determine the energy levels of complex molecules (for example when considering proteins inside the human body) one needs to take into account all the nuclei and all the electrons. Electrons are not allocated to individual atoms anymore, rather they are shared by many atoms and can be described by molecular orbitals. A molecular orbital can be represented as the combination of atomic orbitals from each atom making up the molecule. In theory atomic and molecular orbitals can be calculated by solving the Schrödinger equation, which can be written as a large linear algebra problem, essentially finding eigenvectors and eigenvalues of the Hamiltonian matrix. Unfortunately except for very simple cases no analytic solution is known and for large molecules like proteins even the numerical solution cannot be carried out by current technologies. For this reason, approximations are required.

Hückel method uses the linear combinations of atomic orbitals in determining the energies of certain molecular orbitals inside of a molecule. Originally it only considered electrons with specific bonds that are those which were in pi orbitals. In

1963, Roald Hoffmann extended this theory by also taking into account the other outermost electrons of the molecular models, thus including all valence electrons. With this extension this is the so called extended Hückel method, the subject of this thesis work. Considering the mathematics behind the theory, the overlappings of atomic orbitals inside a molecule are described by so called overlap matrices. The overlap matrix and a Hamiltonian matrix (both are special kind of complex matrices) are used during the calculations.

To initilalize the calculations, the matrices and the atomic orbitals can be relatively easily calculated from the 3D positions of the atoms and the time consuming part of the program is the solution of the linear algebraic equations. The 3D positions can be obtained from certain model considerations, or more often from various physical measurements (X-ray crystallography or NMR) of the molecules. These structures are deposited into public databases, like Protein Data Bank (PDB) as will be described later. In a typical scenario, researchers take 3D structures from PDB and after some preparatory steps use this data as input for the extended Hückel calculations which outputs energy levels (eigenvalues) and molecular wave functions (eigenvectors).

The goal of this thesis work is to create an improved extended Hückel code, that is able - to consume standard PDB input format, - to handle large molecules with thousands of atoms, - to utilize parallelized calculations on multi-core machines, - to output large matrices in sparse format for later processing.

Chapter 2

User Documentation

Open Babel

I used the Openbabel's 2.4.1 version during my work, this package can be acquired by downloading it using the following link:
https://sourceforge.net/projects/openbabel/files/openbabel/2.4.1/openbabel-2.4.1.tar.gz/downlo

As the package is compressed to use up less storage, users will need to have a tool that can carry out decompression for the .tar.gz gzip format. I used Ubuntu's standard tool, the Archive Manager (File Roller) for these purposes. Another alternative is to open a terminal in the folder that contains the compressed package and run the following command:

```
tar -xvzf openbabel-2.4.1.tar.gz
```

After this, setting up the Openbabel software can be done by following the installation instructions contained in the Install file found in the Openbabel-2.4.1 folder that has just been decompressed. These instructions include creating a build directory as well as using cmake (CMake 2.4.8 or later required) and then the makefile to compile and install the package.

One important thing to notice is the possibility of specifying options before using cmake. An example for that is that users may determine the directory, where OpenBabel will be installed. If not specified, the /usr/local/ directory will be the default. This, however, means that it might require superuser privileges for one to use the package or as described later on in this document, to integrate the API of the software as a package. Thus the following command with the aforementioned option is advisable to be used (users may choose a target that is in their home folder):

```
cmake -DCMAKE_INSTALL_PREFIX=/home/username/openbabel ..
```

```
make install
```

Having successfully installed the package, environmental variables may need to be set up in order for it to be used (as described in the Install file):
BABEL_LIBDIR - the location of Open Babel format plugins
BABEL_DATADIR - the location of the data files

For personal configuration the BABEL_LIBDIR environmental variable had to be set. Generally, the following command can be used to set it ($PATH stands for the absolute link of your lib folder inside your extracted openbabel folder):

```
export BABEL_LIBDIR=$BABEL_LIBDIR:$PATH
```

Thus, the following command was executed containing the path for my configuration:

```
export BABEL_LIBDIR=$BABEL_LIBDIR:/home/toncsi/ClionProjects
    /openbabel-2.4.1/lib
```

Finally, I executed the following command for the sample input of a folded titin molecule, to convert between formats. Beware, that superuser rights may be needed for the output file to be written, if no user specified install locations were given.

```
sudo obabel Titin-folded-monomer.sdf -O Titin-folded-monomer
    .pdb
```

For further information please refer to the following User Documentation: `https://openbabel.org/wiki/Tutorial:Basic_Usage`

2.1 Input

The input needs two separate files as opposed to the original tightbind library of the YAEHMOP software. These two files are:

1. A molecule file of the supported formats

2. A configuration file containing the printing options ...

Chapter 3

Developer Documentation

3.0.1 Make and CMake

Make is a tool used for compiling and recompiling of programs useful for handling larger programs with many source files. It uses specific files called makefiles that describe how the different files of the program are related to each other and contains all commands that need to be run. In this terminology, there are two main types of files: targets that are to be updated by make and the preexisting prerequisites upon which targets depend in this process. By calling the make command, users may compile and link the program. This tool is independent of programming languages, compiler directives are included in makefiles.

Cmake is a building tool for C++ source files using makefiles to create executables. It has the major advantage of being cross-platform, as well as open-source. Users may create CMakeLists.txt configuration files, in which they can specify source files and link libraries statically and dynamically to the program by using predefined commands.

During my work I used the CLion Integrated development environment created by JetBrains which is suited for creating software in C or C++ using CMake.

3.1 Input

3.1.1 PDB format

A Protein Data Bank formatted file describes molecules with a three-dimensional structure, using various keywords of standardized order. With this description method protein and nucleic acid structures can be specified. Coordinates of atoms, connections between them, ligands and sidechain rotamers are characteristics that these files may contain and thus uniquely determine a molecule. The file format was named after a database of proteins, the Protein Data Bank `http://www.rcsb.org/pdb/static.do?p=file_formats/pdb/index.html`.

Certain keywords (HEADER, TITLE and AUTHOR) are used for the description of main aspects of the file, namely the protein's name, the scientists having put together this structure into the file and further information.

Each PDB file may contain several models for proteins. These models start with the MODEL keyword and continue with a serial number for the model. The ENDMDL word indicates the end of such a section.

Coordinates may be described with two different keywords: ATOM or HETATM. The key to their difference is that ATOM records are used for standard chemicals whereas for those amino acids and nucleotides that are non-polymer, the HETATM keyword is to be included in the beginning of each line. These coordinate lists are ended with the TER keyword.

Connectivity among the atoms may be described using the CONECT keyword.

```
HEADER    DNA                                        26-JAN-81   1BNA
TITLE     STRUCTURE OF A B-DNA DODECAMER. CONFORMATION AND DYNAMICS
...
AUTHOR    H.R.DREW,R.M.WING,T.TAKANO,C.BROKA,S.TANAKA,K.ITAKURA,
AUTHOR   2 R.E.DICKERSON
...
ATOM      1  O5'  DC A   1      18.935  34.195  25.617  1.00 64.35           O
ATOM      2  C5'  DC A   1      19.130  33.921  24.219  1.00 44.69           C
ATOM      3  C4'  DC A   1      19.961  32.668  24.100  1.00 31.28           C
ATOM      4  O4'  DC A   1      19.360  31.583  24.852  1.00 37.45           O
ATOM      5  C3'  DC A   1      20.172  32.122  22.694  1.00 46.72           C
...
ATOM    487  C4   DG B  24      17.231  22.893  27.570  1.00 13.89           C
TER     488       DG B  24
HETATM  489  O   HOH A  25      19.736  30.706  18.656  1.00 51.86           O
...
HETATM  568  O   HOH B 104      18.692  31.584   4.596  1.00 72.98           O
MASTER  355    0    0    0    0    0    0    6  566    2    0    2
END
```

Listing 3.1: Example PDB file for the deoxyribonucleic acid

3.2 64 bit architecture

3.3 Parallelism

3.4 Output

3.4.1 Market Matrix format

One of the main purposes of using the Matrix Market (MM) file format is to describe sparse matrices, matrices that have almost only zero elements. When carrying out computations with matrices, we are representing their elements using arrays or multidimensional arrays depending on the dimensions of the original matrices. This way however, in case of a sparse matrix, it might happen that in most cases we are allocating memory to store zero elements. Thus it can be concluded, that this technique is not affordable in terms of memory management. Consequently, using the characteristics of sparse matrices, it serves as a better representational model to only store the coordinates of the elements that are non-zero and their respective values.

The previously described concept is the general idea giving the basis for the Matrix Market format as well. When representing a matrix in the MM format the user needs to give the format of the matrix in the first line of the file: either coordinate or array. This decision depends on whether the matrix to be represented is sparse or dense, the coordinate format is to be used for sparse matrices. The next property to be given after a non-breaking space delimiter is the matrix qualifier describing the type of the elements in the matrix. Alternatives include Real and Complex, names denoting the types of matrices represented Integer for those with only integer elements. Users can also choose a Pattern qualifier for cases when a pattern determines the nonzero elements. Then shall one disclose information regarding the symmetry of the matrix: general, symmetric and hermitian or skew-symmetric.

This is followed by the structured comments part, a section extending previous versions of this format. Useful for providing more information regarding the file, this section may also contain customized comments regarding the matrix.

The final part of a Matrix Market formatted file is the description of coordinates. The first line of this section refers to the dimensions of the matrix followed by the number of nonzero elements in the matrix. Then come the nonzero elements in each line with the standard structure of rownumber, columnnumber, value of matrix element.

Example for the Matrix Market format (taken from The Matrix Market Exchange Formats: Initial Design paper):

```
1      0       0       6      0
0    10.5      0       0      0
0      0     .015      0      0
0   250.5      0     -280   33.32
0      0       0       0     12
```

Listing 3.2: Example sparse matrix of 5 dimensions

```
%%MatrixMarket matrix coordinate real general
%=================================================================
%
% This ASCII file represents a sparse MxN matrix with L
% nonzeros in the following Matrix Market format:
%
% +----------------------------------------------+
% |%%MatrixMarket matrix coordinate real general |
% |%                                             |
% |% comments                                    |
% |%                                             |
% |     M  N  L                                  |
% |     I1  J1  A(I1, J1)                        |
% |     I2  J2  A(I2, J2)                        |
% |     I3  J3  A(I3, J3)                        |
% |         . . .                                |
% |     IL JL  A(IL, JL)                         |
% +----------------------------------------------+
%
% Indices are 1-based, i.e. A(1,1) is the first element.
%
%=================================================================
  5  5  8
    1     1    1.000e+00
    2     2    1.050e+01
    3     3    1.500e-02
    1     4    6.000e+00
    4     2    2.505e+02
    4     4   -2.800e+02
    4     5    3.332e+01
    5     5    1.200e+01
```

Listing 3.3: Market Matrix file for the matrix

3.5 Tests

3.5.1 Unit tests

Through the process of unit testing developers may grasp the chance to check if their software indeed works the way they expect it to. This is achieved by taking the smallest parts, units of the code and testing their functionalities. Ensuring that every tiny method in a robust application operates correctly may seem as a overly time-consuming task, but it can lead to the avoidance of several serious errors and thus to cost efficiency in the long run. For this reason, this type of testing comes as paramount significance while and after developing software in the corporate world.

Each unit is to be tested completely separately and independently from other parts of the application. Consequently, the outcome of a unit test may in no way be influenced by another one. If this is not the case then the content of these methods need serious reconsideration.

When taking into consideration preexisting packages used in this thesis work, it can be concluded that such software require no further unit testing as we presuppose that they were developed correctly and function well. This way tests of this kind were created exclusively to newly created pieces of code that form larger unity in the entirety of the project. In order to avoid duplicated code and easy maintenance of application, the original modules were included in the test projects using the CMake file. Test classes for performing operations on and with the existing classes and a main source file running the unit tests were introduced for the important modules. This method allowed tests to be taken in a fashion that is most similar to the release running of the application. During this type of work no further test frameworks were used, tests were prepared manually bearing in mind the most important use cases.

During the course of making tests, the principles of building up a test object and environment was given emphasis. Thus in general test classes were made with a function responsible for any tasks that were connected to the creation of a simulation environment. As examples, input/output methods, database calls are often functionalities included in such parts of the test class. Instantiation of objects of the classes under test happen just before the call of these functions or constitute as the first operation inside of them.

Following the principles of unit testing and thus aiming at not leaving any states or files created throughout this process, functions responsible for restoring the state before the testing commenced were introduced as well. Such tear down functions may entail the closing of open filestreams as well as the removal of temporary files that were created in the process of unit testing. This functionality is required so that ultimately, as previously mentioned, no resources or files are created.

TestOpenBabelUtils class

Files related to the checking of the input module are located in the tests folder. Testing the OpenBabelUtils class was carried out in a way that a TestOpenBabelUtils test class was created in this folder. The integration of the OpenBabelUtils class was achieved by introducing a pointer to an OpenBabelUtils object as private member of the class.

This architectural decision was made because of many reasons. On one hand this instance is solely used for test purposes and thus one object is used throughout the testing process (thus resembling singleton objects used in C++), hence there was no need for copying existing objects. Another reason is because of simple memory management, storing only a pointer to an object has the clear advantage of less allocated memory. The member was made private in order to adhere to the paradigm of object oriented design. Further test classes will follow these principles as well.

The main.cpp file of test projects contain asserted calls to the test functions. In order to test all parts of if statements, in some cases more than one tests were introduced for the same function.

The following test functions were introduced in the **TestOpenBabelUtils** class:

- **buildUp(filename)**: this function is responsible for the aforementioned building up of test environment in the way of dynamically instantiating an OpenBabelUtils object and creating an input file of the format specified by the argument
- **obtainInputFileFormat()**: call to the obtainInputFileFormat() function of private OpenBabelUtils object
- **openBabelConversion()**: call to the openBabelConversion() function of private OpenBabelUtils object
- **convert()**: call to the convert() function of private OpenBabelUtils object
- **initInPutFileSuccessFileOpen(filename)**: calls the private member's initInputFile function with filename argument, then checks if the input file is open

- **initOutPutFileSuccessFileOpen(filename)**: calls the private member's initOutPutFile function with filename argument, then checks if the output file is open
- **obtainInputFileFormatSuccessFileName()**: may be called after an outer call to private member's obtainInputFileFormat(), after such a call determines if the name of the file without the end string containing the file format was successfully acquired from the initial name
- **obtainInputFileFormatSuccessFormat()**: may be called after an outer call to private member's obtainInputFileFormat(), after such a call determines if the newly created file name has the correct string as an end to (this specifies the format of the file)
- **openBabelConversionSuccessInputClosed()**: this function determines after an openBabelConversion() function invoke, if the input file was closed successfully
- **openBabelConversionSuccessOutputClosed()**: this function determines after an openBabelConversion() function invoke, if the output file was closed successfully
- **tearDown(fileName)**: as previously mentioned, this function ensures that temporary files are removed and as the OpenBabelUtils class has filestream initializer functions solely responsible for opening files, it also closes both input and output filestream should they have remained open during the test

TestPdbParser class

- **buildUp(filename,molecule)**: this function creates a PdbParser object, and then populates a temporary PDB formatted file with random data
- **readSuccessFileExists()**: in this class, a private fileChecker member was introduced. This function uses this filestream to check if the output file is existing, by trying to open it
- **readSuccessFileNotempty()**: checks if the file created by the PdbParser is non-empty
- **tearDown(fileName)**: removes the files created by the PdbParser object during test and destroys the private member of the class

TestSettingsParser class

Compared to the previous test classes TestSettingsParser class was extended with a reference to a map data structure as an additional private member. This allows the possible injection and opportunity for minor redesign in case a user would like to work test with an existing properties data set.

- **buildUp(filename)**: creates the SettingsParser class by using the testProps member and then populates a temporary file with random property data
- **readSettingsSuccessPropsNotEmpty()**: function checking if the properties member is non-empty
- **writeSettingsSuccessFileNotempty()**: function checking if the output file created by the SettingsParser class is non-empty
- **tearDown(fileName)**: removes the files created by the SettingsParser object during test and destroys the private member of the class

TestMatrixMarketWriter class

Using the TestMatrixMarketWriter class several casese were tested including the use of cutoff values when working with the MatrixMarketWriter class. Two other cases were when writing to the file strictly adhering to the standards of the MatrixMarket format and when only the coordinates and values of the matrix were recorded in a sparse format.

- **buildUp(filename)**: creates a temporary sparse matrix, then instantiates a MatrixMarketWriter object using the filename argument and the temporary matrix
- **buildUp(filename, cutoff)**: an overloaded version of the buildup function, additionally a cutoff value may be added as well to be used for MatrixMarketWriter object instantiation
- **tearDown(fileName)**: removes the file created by the MatrixMarketWriter object during test and destroys the private member of the class
- **writeSuccessFileNotempty()**: checks if after a writing function call, the created file in non-empty

3.5.2 System tests

Chapter 4

Bibliography

[1] Worldwide Protein Data Bank: Introduction 2010 `http://www.wwpdb.org/documentation/file-format-content/format33/sect1.html` April 16, 2017

[2] Worldwide Protein Data Bank: Title Section 2010 `http://www.wwpdb.org/documentation/file-format-content/format33/sect2.html` April 16, 2017

[3] Worldwide Protein Data Bank: Coordinate Section 2010 `http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html` April 16, 2017

[4] Worldwide Protein Data Bank: Connectivity Section 2010 `http://www.wwpdb.org/documentation/file-format-content/format33/sect10.html` April 16, 2017

[5] N M O'Boyle, M Banck, C A James, C Morley, T Vandermeersch, and G R Hutchison. "Open Babel: An open chemical toolbox." J. Cheminf. (2011), 3, 33. DOI:10.1186/1758-2946-3-33 April 18, 2017

[6] The Open Babel Package, version 2.3.1, Oct 2011 `http://openbabel.org` April 18, 2017

[7] Khanacademy: The quantum mechanical model of the atom `https://www.khanacademy.org/science/physics/quantum-physics/quantum-numbers-and-orbitals/a/the-quantum-mechanical-model-of-the-atom` April 27, 2017

[8] `http://chemed.chem.purdue.edu/genchem/topicreview/bp/ch8/mo.html` April 27, 2017

[9] `https://dasher.wustl.edu/chem478/reading/extended-huckel-lowe.pdf` April 27, 2017

[10] http://study.com/academy/lesson/what-is-an-energy-level-of-an-atom-definitio html April 27, 2017

[11] Purdue university, College of Science, Chemical Education Division Groups: Molecular Orbital Theory 2004 https://chem.libretexts.org/Textbook_Maps/Physical_and_Theoretical_Chemistry_Textbook_Maps/Map%3A_Quantum_States_of_Atoms_and_Molecules_(Zielinksi_et_al.)/10%3A_Theories_of_Electronic_Molecular_Structure/10.06%3A_Semi-Empirical_Methods%3A_Extended_H%C3%BCckel April 27, 2017

[12] Ronald F. Boisvert, Roldan Pozo, Karin A. Remington, National Institute of Standards and Technology December 1996 The Matrix Market Exchange Formats: Initial Design http://math.nist.gov/MatrixMarket/reports/MMformat.ps April 29, 2017

[13] National Institute of Standards and Technology, Mathematical and Computational Sciences Division: Text File Formats 14 August 2013 http://math.nist.gov/MatrixMarket/formats.html April 29, 2017

[14] GNU Operating System: GNU Make Manual, May 22, 2016 http://www.gnu.org/software/make/manual/make.html, May 8, 2017

[15] The IEEE and The Open Group: make utility page, 2016 http://pubs.opengroup.org/onlinepubs/9699919799/utilities/make.html#tag_20_76_13_04 May 8, 2017

[16] Kitware: CMake, date of submit unknown, https://cmake.org/ May 8, 2017