a practical introduction to the simulation of molecular systems

分子体系模拟应用入门

Martin J Field

CAMBRIDGE
世界图书出版公司

A PRACTICAL INTRODUCTION TO THE SIMULATION OF MOLECULAR SYSTEMS

MARTIN J. FIELD

Laboratoire de Dynamique Moléculaire, Grenoble

CAMBRIDGE
UNIVERSITY PRESS

世界图书出版公司

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK http://www.cup.cam.ac.uk
40 West 20th Street, New York, NY 10011-4211, USA http://www.cup.org
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

First published 1999

Printed in the United Kingdom at the University Press, Cambridge

Typeset in Times Roman 11/14pt

A catalogue record for this book is available from the British Library

Library of Congress Cataloging in Publication data
Field, Martin (Martin J.)
A practical introduction to the simulation of molecular systems /
Martin J. Field
p. cm.
Includes bibliographical references and index.
ISBN 0 521 58129 X (hardbound)
1. Molecules – Models – Computer simulation. I. Title.
QD480.F5 1999
541.2'2'0113–dc21 98-37540 CIP

ISBN 0521 58129 X hardback

This edition of A Practical Introduction to the Simulation of Molecular Systems by Martin J.Field is published by arrangement with the Syndicate of the Press of University of Cambridge, Cambridge, England.

Licensed edition for sale in the People's Republic of China only. Not for export elsewhere.

A PRACTICAL INTRODUCTION TO THE SIMULATION OF MOLECULAR SYSTEMS

This book provides a practical introduction to the range of different techniques available for the simulation of molecular systems. The text includes a library of program modules written in FORTRAN 90 with which the simulations discussed in the text were performed.

Molecular simulation and modeling methods are undergoing rapid development. They are increasingly important tools for fundamental and applied research in academia and industry in such diverse fields as drug design and materials science. Each chapter describes a general class of methods or algorithms and then illustrates their use with example programs, written using the module library. Topics covered include energy functions, optimization of geometry and reaction-path-location techniques, normal mode analysis, molecular dynamics and Monte Carlo simulations and free-energy calculations. The examples and the module library all make use of molecular mechanical energy functions, but almost all of the techniques outlined in the book can be used with other energy functions, such as those based on quantum mechanical methods.

This book will be of interest to advanced undergraduates, graduate students and researchers who use molecular simulation techiques, particularly in theoretical and computational chemistry, biophysics and computational molecular physics.

Preface

The reason that I have written this book is simple. It is the book that I would have liked to have had when I was learning how to carry out simulations of complex molecular systems. There was certainly no lack of information about the theory behind the simulations but this was widely dispersed in the literature and I often discovered it only long after I needed it. Equally frustrating, the programs to which I had access were often poorly documented, sometimes not at all, and so they were difficult to use unless the people who had written them were available and preferably in the office next door! The situation has improved somewhat since then (the 1980s) with the publication of some excellent monographs but these are primarily directed at simple systems, such as liquids or Lennard-Jones fluids, and do not address many of the problems that are specific to larger molecules.

My goal has been to provide a practical introduction to the simulation of molecules using molecular mechanical potentials. After reading the book, readers should have a reasonably complete understanding of how such simulations are performed, how the programs that perform them work and, most importantly, how the example programs presented in the text can be tailored to perform other types of calculation. The book is an *introduction* aimed at advanced undergraduates, graduate students and confirmed researchers who are newcomers to the field. It does not purport to cover comprehensively the entire range of molecular simulation techniques, a task that would be difficult in 300 or so pages. Instead, I have tried to highlight some of the basic tasks that can be done with molecular simulations and to indicate some of the many exciting developments which are occurring in this rapidly evolving field. I have chosen the references which I have put in carefully as I did not want to burden the text with too much information. Inevitably such a choice is subjective and I apologize in advance to those workers whose work or part of whose work I did not explicitly acknowledge.

There are many people who directly or indirectly have helped to make this book possible and whom I would like to thank. They are: my early teachers in the field of computational chemistry, Nicholas Handy at Cambridge and Ian Hillier at Manchester; Martin Karplus and all the members of his group at Harvard (too numerous to mention!) during the period 1985–9 who introduced me to molecular dynamics simulations and molecular mechanics calculations; Bernie Brooks and Rich Pastor, at the NIH and FDA, respectively, whose lively discussion and help greatly improved my understanding of the simulations I was doing; and all the members of my laboratory at the IBS, past and present, Patricia Amara, Dominique Bicout, Celine Bret, Laurent David, Lars Hemmingsen, Konrad Hinsen, David Jourand, Flavien Proust, Olivier Roche and Aline Thomas. Finally, special thanks go to Patricia Amara and to Dick Wade at the IBS for comments on the manuscript, to Simon Capelin and the staff of CUP for their guidance with the production of the book, to the Commissariat à l'Energie Atomique and the Centre National de la Recherche Scientifique for financial support and to my wife, Laurence, and to my sons, Mathieu and Jeremy, for their patience.

Martin J. Field
Grenoble

Contents

1
Preliminaries

1.1 Introduction

The aim of this book is to provide a practical introduction to performing simulations of molecular systems. To do this, the necessary background about the theory behind various types of simulation is covered and a library of program modules called DYNAMO is provided, which can be used to perform the simulations. The style of the book is pragmatic. Each chapter, in general, contains some theory about a particular topic together with a description of relevant program modules and examples of their use. Suggestions for further work (or exercises) are given at the end.

By the end of the book, readers should have a good idea of how to perform various types of simulation as well as some of the difficulties that are involved. The module library should also provide a reasonably convenient starting point for those wanting to write their own programs to study the systems they are interested in. The fact that users have to write their own programs to do their simulations has advantages and disadvantages. The major advantage is flexibility. Many molecular modeling programs come in a single package and so, inevitably, can provide only a limited range of options. In contrast, a comprehensive library of procedures can be employed to do whatever the user wants and all data in the program will be available for analysis. The drawback is, of course, that the programs have to be written – a task that many readers may not be familiar with or have little inclination to do themselves. However, those who fall into the latter category are urged to read on. The modules have been designed to be easy to use and should be accessible to everyone even if they have only a minimum amount of computing experience.

This chapter explains some essential background information about the programming style in which the modules and the example programs are

written. Details of how to obtain the source code of the library for implementation on specific machines are left to the appendix.

1.2 The DYNAMO module library

The module library and the example programs provided with this book are all written in the programming language FORTRAN 90. The reasons for this choice were threefold. First, FORTRAN 90 is a powerful and modern programming language. Unlike many modern languages, it is not *object-oriented*, but it has, among its features, a compact syntax for specifying operations on arrays and it allows modular code to be written. Second, FORTRAN is still a very widely used programming language for scientific applications. Third, the author has done most of his programming in FORTRAN!

The modules and their procedures, which are coded in FORTRAN 90, have been written so as to be as clear as possible, even if this has been at the expense of efficiency in some cases. It has also meant that the number of options available has been kept limited so as not to obscure too much of the flow of control in the code. The great majority of the modules were written specifically for this book or were adapted from procedures used by the author in his programs.

One of the major improvements of FORTRAN 90 over FORTRAN 77 and a feature that we shall employ extensively is the notion of a *module*. In FORTRAN 77, data (most often in *common blocks*) and procedures (*functions* and *subroutines*) were kept separated. In contrast, FORTRAN 90 allows data and procedures that are related to be grouped together into coherent units using modules. Items in a module can be classified as being *private* or *public*. Public data and procedures are accessible to other modules or programs whereas private data and procedures can be accessed only from within the module. This is advantageous because it provides a measure of protection for a module's variables and it means that the unnecessary details of the implementation of specific algorithms or data structures can be hidden from the remaining parts of the program. For this reason, only the public procedures and variables will be presented when a module is described in the rest of the book.

The shorthand that will be used for listing the (public) contents of a module is as follows

```
MODULE EXAMPLE
! . Parameter declarations.
... ... ...
! . Scalar variable declarations.
... ... ...
! . Array variable declarations.
```

```
... ... ...
! . Type variable declarations.
... ... ...
! . Function and subroutine declarations.
CONTAINS
   SUBROUTINE EXAMPLE_SUBROUTINE1 ( ... arguments ... )
      ... argument declarations ...
   END SUBROUTINE EXAMPLE_SUBROUTINE1
   ... ... ...
END MODULE EXAMPLE
```

The public parameters are declared first, followed by any variables (usually in the order scalar, array and type). The procedure declarations come last after the CONTAINS statement. For the function and subroutine declarations, only the name of the procedure, its calling sequence and the details of its arguments are given. Although not strictly necessary in FORTRAN 90, all parameter and variable declarations list explicitly the nature of the variable (i.e. whether it is CHARACTER, INTEGER, LOGICAL or REAL) and, if it is an array, its dimension. All argument declarations have the INTENT statement which indicates whether the argument is for input only, for output only or both for input and for output.

This is all the information that is needed concerning the declaration of the public parts of a module. However, there are several extra technical points that can be made about the programming used in the modules (and the example programs).

- All arrays are allocated dynamically. That is, they are declared either as ALLOCATABLE or as pointers and then allocated to have the correct size when needed. This makes the module programming more complicated but greatly increases the modules' flexibility because they do not need to be recompiled for larger systems. The use of allocatable arrays has been preferred, in general, over the use of pointers because keeping track of the latter can be difficult.
- In the modules and the example programs, all floating point (REAL) numbers comprise 64 bits although in the listings that appear in the book this is omitted for the sake of clarity. The integer and logical types are left to assume their default values for the particular machines being used (usually 32 bits but sometimes 64 bits).
- Standard FORTRAN 90 has been used throughout. This means that the programs should run (and produce the same results!) on any machine for which a FORTRAN 90 compiler exists. The newer revision of FORTRAN 90, FORTRAN 95, introduces a number of minor, albeit useful, changes but these have not been used in the programming in this book.
- Both function and subroutine procedures are used. A function is employed only in cases in which all of the arguments are input arguments (and so remain

unchanged by the procedure), when there is only one output result and when variables of other modules or public variables of the function's own module are left unchanged. A function may, however, change private variables of its own module or generate an error.

- Most module variables are explicitly initialized when declared. This has been done so that the modules are ready to use immediately without the necessity of first calling special initialization procedures.

1.3 Miscellaneous modules

Most of the modules in the DYNAMO library are scientific modules concerned with the manipulation and simulation of molecular systems, which will be discussed in detail in the remainder of the book. There are, in addition, important modules that deal with a miscellany of other tasks such as input and output and the declaration of physical constants. These will be described only briefly because we shall not encounter the majority of them again. They may be divided into four categories.

The first category contains utility modules that take care of such functions as file management and parsing.

`DEFINITIONS` contains parameter definitions used by the modules and the example programs. In particular, it contains *all* the machine-dependent parameters that are used by the library. The most important of these is the parameter `DP`, which is short for *default precision*. It defines the model used for the floating point numbers and is used whenever a real number is declared in a module or program.

`FILES` deals with file management. In general, users are supposed to manage their own files, but `FILES` contains a subroutine that will locate the next available FORTRAN *stream* or *unit* number that is available.

`IO_UNITS` defines as the parameters `INPUT` and `OUTPUT` the unit numbers of the input and output streams (i.e. where the program is to read from and where it is to write to).

`PARSING` contains procedures that read and process data from a formatted file. It is employed by all modules that read formatted files and means that all input involving such files in the book is 'free format' (fixed format input is *not* used!).

`STATUS` contains the subroutine that is called whenever an error results from the execution of a module or program. This procedure, `ERROR`, prints out an error message and then, to avoid complications, terminates the execution of the program immediately.

`STRING` contains procedures that manipulate character strings.

TIME contains a procedure that prints the current date and time. FORTRAN 90, unlike the revision FORTRAN 95, has no intrinsic procedures that deal with CPU time and so this module does not contain any.

Of these modules, the beginning user is likely to need to use only DEFINITIONS and IO_UNITS, the former for the parameter DP and the latter for the parameters INPUT and OUTPUT.

Modules of the second category store parameter data about various chemical, physical and mathematical constants. There are two.

CONSTANTS contains fundamental mathematical, physical and chemical constants.

ELEMENTS contains data about each element. This includes the element symbol, its atomic mass and a value for its radius (or 'effective' size).

In both these modules the data are defined in terms of parameters. In other words, the information can be accessed, but it cannot be changed.

The third category of modules deals with standard numerical mathematical tasks. They are the following.

BAKER_OPTIMIZATION is a module that finds a stationary point of a multidimensional function. The algorithm it uses will be described in more detail in section 6.5.

CONJUGATE_GRADIENT has a subroutine for the optimization of a multidimensional function using a conjugate gradient algorithm.

DIAGONALIZATION contains procedures that find the eigenvalues and eigenvectors of real symmetric matrices.

LINEAR_ALGEBRA has procedures for performing standard operations in linear algebra, such as the normalization of vectors.

RANDOM is the module that contains procedures for generating random numbers drawn from uniform and Gaussian distributions. This is the only module for which it is advisable to call the module's initialization procedure explicitly before use because it sets the value of the seed for the random-number generator.

SORT contains procedures for sorting.

SPECIAL_FUNCTIONS has a procedure for computing the complementary error function.

STATISTICS contains procedures for performing various types of statistical analyses on data. It is explained in more detail in section 10.2.

The fourth category of modules contains a single example. It is the module DYNAMO whose main task is to declare all the remaining modules in the library

using USE statements, one for each module. This means that it is only necessary to define this module in an example program to make all the other modules in the library accessible. There is also a single procedure, DYNAMO_HEADER, which prints out a title indicating the version number of the module library. The use of this module is described further below.

1.4 Example programs

In most of the chapters, example programs that illustrate the various modules in the library are given. All the programs have a standard format and should provide models for users wanting to write their own. The format is

```
PROGRAM EXAMPLE

! . Module declarations.
USE DYNAMO

IMPLICIT NONE

! . Parameter and variable declarations.
... ... ...

! . Initialization.
... initialization commands ...

! . Program commands.
... ... ...

! . Termination.
... termination commands ...

CONTAINS

   ... Program functions and procedures ...

END PROGRAM EXAMPLE
```

The order of statements is as follows.

1. The program starts with the PROGRAM statement followed by the name of the program. In the examples found in this book, all the programs have the name EXAMPLE followed by an integer.
2. The next statement declares that the module DYNAMO should be used by the program. As described in the previous section, this module lists all the modules that are in the program library. It simplifies the use of the module library because otherwise each module in the library that was required by the program would have to be specified in a separate USE statement. In this way only a single

statement is needed. The USE facility is extremely important because it means that the compiler can check the type characteristics of the module variables and the syntax of calls to module procedures. These checks help to eliminate lots of errors before the program is run.

3. The next statement, which should always be used if local variables are being defined, is IMPLICIT NONE. This compels the types of all variables to be declared explicitly.
4. Following the IMPLICIT NONE statement are declaration statements for all the parameters and variables that occur in the program.
5. After all the declarations come the executable statements. These can be divided into three parts.
   (a) An initialization section that can include statements that initialize module variables and write out information about the date and time, the version number of the module library and, perhaps, a title to indicate what the program is doing.
   (b) A section that contains the statements that perform the task for which the program has been written.
   (c) A termination section that includes statements that tidy up any dynamically allocated storage space, close files and write out miscellaneous information such as, for example, the date and time.
6. The executable commands are followed by the CONTAINS keyword and any local functions or subroutines that are used in the program's executable statements.
7. The text of the program terminates with the END PROGRAM statement.

Some simplifications are made in the presentation of the example programs for clarity. Only a few of the examples use local procedures and so the block following the CONTAINS statement seldom occurs. Also, to reduce the space required by the program listings, some of the declaration statements, including the IMPLICIT NONE keyword, and the initialization and termination commands are omitted. In case of confusion, readers are advised to look at the full texts of the example programs in the source code distribution which do, of course, include these statements.

To use a program once it is written it is necessary to compile it and link it with the modules from the module library. How this is done is machine dependent but where to find this information will be left until the appendix.

1.5 Notation and units

Finally, a few general points about the notation and units used in this book and the program library will be made. In the text, all program listings, module definitions and DYNAMO library procedures and variables have been represented by using characters in typewriter style, e.g. NATOMS.