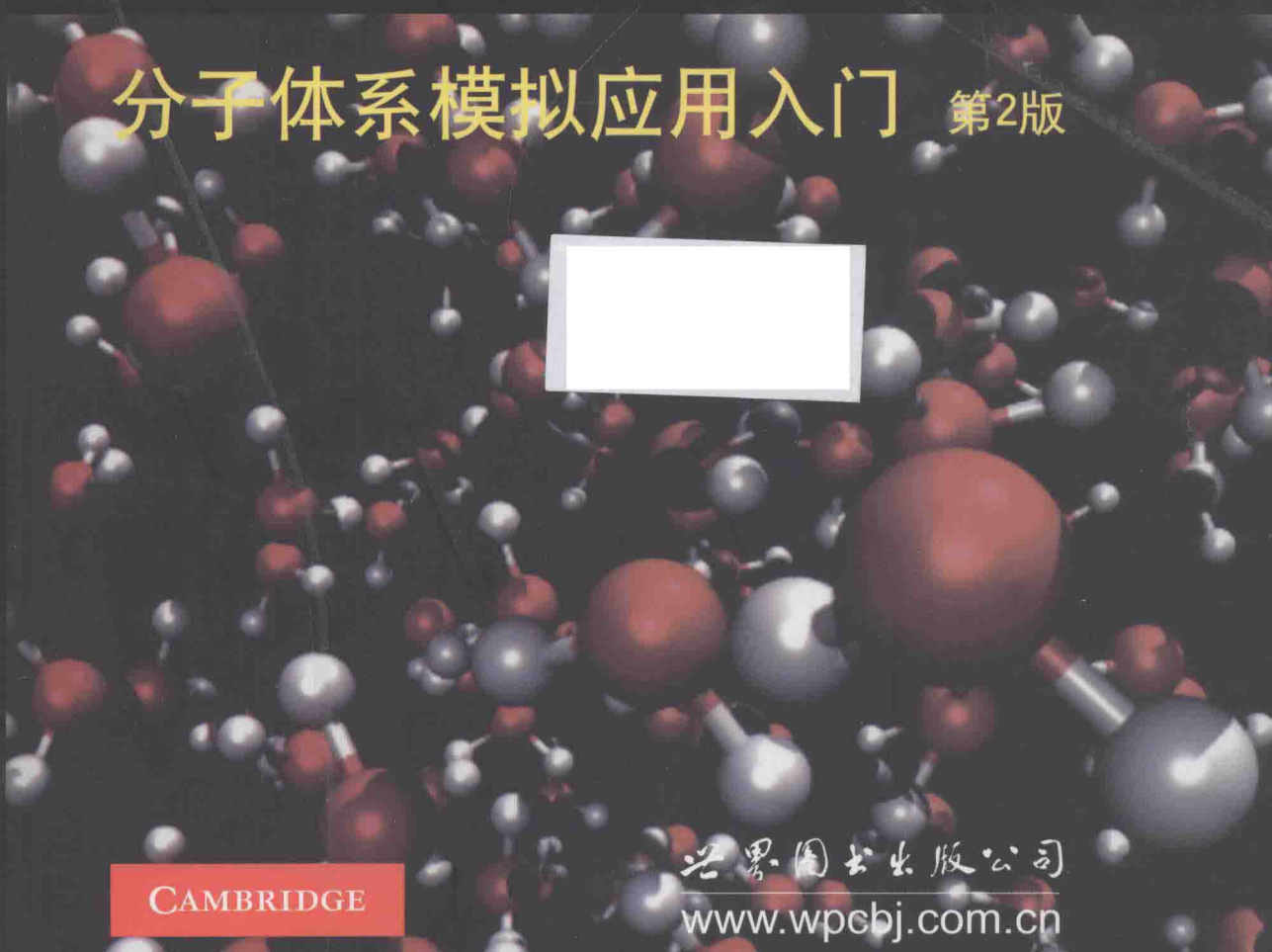
Martin J. Field

A Practical Introduction to the Simulation of Molecular Systems

Second Edition

分子体系模拟应用入门 第2版



CAMBRIDGE

世界图书出版公司
www.wpcbj.com.cn

A PRACTICAL INTRODUCTION TO THE SIMULATION OF MOLECULAR SYSTEMS

Second Edition

MARTIN J. FIELD

Institut de Biologie Structurale – Jean-Pierre Ebel, Grenoble, France

CAMBRIDGE UNIVERSITY PRESS

图书在版编目（CIP）数据

分子体系模拟应用入门 = A practical introduction to the simulation of molecular systems：第2版：英文/（法）菲尔德（Field，M. J.）著. —影印本. —北京：世界图书出版公司北京公司，2014. 8

ISBN 978-7-5100-8443-0

Ⅰ. ①分… Ⅱ. ①菲… Ⅲ. ①超分子结构—化学—英文 Ⅳ. ① 063

中国版本图书馆 CIP 数据核字（2014）第 188056 号

书　　名：	A Practical Introduction to the Simulation of Molecular Systems Second Edition
作　　者：	Martin J. Field
中 译 名：	分子体系模拟应用入门 第2版
责任编辑：	高蓉　刘慧
出 版 者：	世界图书出版公司北京公司
印 刷 者：	三河市国英印务有限公司
发　　行：	世界图书出版公司北京公司（北京朝内大街137号 100010）
联系电话：	010-64021602，010-64015659
电子信箱：	kjb@ wpcbj. com. cn
开　　本：	16开
印　　张：	22.5
版　　次：	2015年1月
版权登记：	图字：01-2013-4921
书　　号：	978-7-5100-8443-0　　**定　　价：** 89.00元

A PRACTICAL INTRODUCTION TO THE SIMULATION OF MOLECULAR SYSTEMS

Second Edition

Molecular simulation is a powerful tool in materials science, physics, biophysics, chemistry, drug design and many other areas of research. This updated edition provides a pragmatic introduction to a wide range of techniques for the simulation of molecular systems at the atomic level. The first part of the book concentrates on methods for calculating the potential energy of a molecular system, with new chapters on quantum chemical, molecular mechanical and hybrid potential techniques. The second part covers ways of investigating the conformational, dynamical and thermodynamical properties of systems, discussing such techniques as geometry-optimization, normal-mode analysis, and molecular dynamics and Monte Carlo simulation.

Now employing Python, the second edition includes a wealth of examples and program modules for each simulation technique. This allows readers to carry out the calculations and to appreciate the inherent difficulties involved in each. This is a valuable resource for researchers and graduate students wanting to know how to perform atomic-scale molecular simulations.

Additional resources for this title, including the program library, technical information and instructor-solutions, are available online at www.cambridge.org/9780521852524.

MARTIN J. FIELD is Group Leader of the Laboratoire de Dynamique Moléculaire at the Institut de Biologie Structurale – Jean-Pierre Ebel, Grenoble. He was awarded his Ph.D. in Quantum Chemistry from the University of Manchester, UK, in 1985. His areas of research include using molecular modeling and simulation techniques to study biological problems. More specifically, his current interests are in the development and application of hybrid potential techniques to study enzymatic reaction mechanisms and other condensed phase processes.

Reviews of the first edition:

'... a valuable teaching aid for those presenting this topic. It should be of interest not only to the physical chemist, but also to those involved in computational biophysics, biochemistry or molecular physics.' *Scientific Computing World*

'... this book is a valuable addition to my shelf and one that I must make sure doesn't disappear because my research group has taken off with it!' *Nell L. Allan. Chemistry and Industry*

Preface to the first edition

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 apologise 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 Cambridge University Press 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, 1998

Preface to the second edition

This edition of *A Practical Introduction* has two major differences from the previous one. The first is a discussion of quantum chemical and hybrid potential methods for calculating the potential energies of molecular systems. Quantum chemical approaches are more costly than molecular mechanical techniques but are, in principle, more 'exact' and greatly extend the types of phenomena that can be studied with the other algorithms described in the book. The second difference is the replacement of FORTRAN 90 by Python as the language in which the DYNAMO module library and the book's computer programs are written. This change was aimed to make the library more accessible and easier to use. As well as these major changes, there have been many minor modifications, some of which I wanted to make myself but many that were inspired by the suggestions of readers of the first edition.

Once again, I would like to acknowledge my collaborators at the Institut de Biologie Structurale in Grenoble and elsewhere for their comments and feedback. Special thanks go to all members, past and present, of the Laboratoire de Dynamique Moléculaire at the IBS, to Konrad Hinsen at the Centre de Biophysique Moléculaire in Orléans and to Troy Wymore at the Pittsburgh Supercomputing Center. I would also like to thank Michelle Carey, Anna Littlewood and the staff of Cambridge University Press for their help during the preparation of this edition, Johny Sebastian from TechBooks for answers to my many LaTeX questions, and, of course, my family for their support.

Martin J. Field
Grenoble, 2006

A Practical Introduction to the Simulation of Molecular Systems Second Edition(978-0-521-85252-4) by Martin J. Field, first published by Cambridge
University Press 2007

Contents

1

Preliminaries

1.1 Introduction

The aim of this book is to give a practical introduction to performing simulations of molecular systems. This is accomplished by summarizing the theory underlying the various types of simulation method and providing a programming library, called pDynamo, which can be used to perform the calculations that are described. The style of the book is pragmatic. Each chapter, in general, contains some theory about related simulation topics together with descriptions of example programs that illustrate their use. Suggestions for further work (or exercises) are listed at the end.

By the end of the book, readers should have a good idea of how to simulate molecular systems as well as some of the difficulties that are involved. The pDynamo library should also be a reasonably convenient starting point for those wanting to write 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 with interfaces that supply only a limited range of options. In contrast, the simulation algorithms in pDynamo can be combined arbitrarily and much of the data generated by the program is available for analysis. The drawback is 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. pDynamo has 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 pDynamo and the example programs are written. Details of how to obtain the library for implementation on specific machines are left to the appendices.

1.2 Python

All the example programs in this book and much of the programming library are written in the programming language Python. The rest of the library, which most readers will never need to look at, consists of code for which computational efficiency is paramount and is written in C. The reasons for the choice of Python were threefold. First, it is a powerful and modern programming language that is fun to use! Unlike languages such as C and FORTRAN, it is an interpreted language, which means that programs can be run immediately without going through separate compilation and linking steps. Second, Python is open-source software that is free and runs under a wide variety of operating systems and, third, there is a very active development community that is continually enhancing the language and adding to its capabilities.

Most computer languages are easiest to learn by example and Python is no exception. The following, simple program illustrates several basic features of the language:

```
1  """Example 0."""
2
3  import math
4
5  # . Define a squaring function.
6  def Square ( x ):
7      return x**2
8
9  # . Create a list of integers.
10 values = range ( 10 )
11
12 # . Loop over the integers.
13 for i in values:
14     x = float ( i )
15     print "%5d%10.5f%10.5f%10.5f" \
               % ( i, x, math.sqrt ( x ), Square ( x ) )
```

Line 1 is the program's *documentation string* which, in principle, should give a concise description of what the program is supposed to do. All the examples in this book, however, have documentation strings of the type `"""Example n."""` to save space and to avoid duplicating the explanations that occur in the text.

Lines 2, 4, 8 and 11 are blank and are ignored.

Line 3 makes the standard Python *module* `math` accessible to the program. Python itself and programs written using Python – including pDynamo – consist of modules which must be explicitly *imported* if their contents are to be used.

The `import` statement has a number of different forms and the one shown is the simplest. With this form, module items are accessed by prefixing the item's name with the module name, followed by a dot (`.`) character. Thus, the function `sqrt` from the module `math`, which is used on *line 15* of the program, is accessed as `math.sqrt`. An alternative form, which is sometimes preferable, is `from math import sqrt`. This makes it possible to refer to the function `sqrt` by its name only without the `math.` prefix.

Lines 5, 9 and 12 are *comments* which are included to make the program easier to understand. Python ignores all characters from the hash character (`#`) until the end of the line.

Lines 6–7 define a very simple Python *function*. Functions are named collections of instructions that can be *called* or *invoked* at different points in a program. They behave similarly in Python to functions in other languages, such as C and FORTRAN.

Line 6 is the function definition line. It starts with the word `def` which tells Python that a function definition is coming and terminates with a colon (`:`). The second word on the line, `Square`, is the name that we are giving to the function and this is followed by the function's *arguments* which appear in parentheses. Arguments are variables that the function needs in order to work. Here there is only one, `x`, but there can be many more.

The function definition line is followed by the *body* of the function. This would normally consist of several lines but here there is only one, *line 7*. Python is unusual among programming languages in that the lines in the function body are determined by line indentation. In other languages, such blocks of code are delimited by specific characters, such as the matching braces {...} of C or the `FUNCTION ... END FUNCTION` keywords of FORTRAN 90. The number of spaces to indent by is arbitrary – in this book it is always four – but all lines must be indented by the same amount and instructions after the end of the function must return to the original indentation level.

Line 7 is very simple. The second part of the line contains the expression `x**2` which computes the square of the function's argument `x`. The `**` symbol denotes the power operator and so the expression tells Python to raise `x` to the power of 2. The first part of the line is the keyword

`return` which says that the result of the squaring calculation is to be *returned* to the place from which the function was called.

Line 10 is the first executable line of the example and illustrates several more features of the Python language – *built-in functions*, *sequence types* and *variable assignment*. `range` is one of Python's built-in functions and is always available whenever the Python interpreter is invoked. It produces a sequence of integers, in this case ten of them, starting with the value `0` and finishing with the value `9`. Python, like C, but unlike FORTRAN, starts counting from zero and not from one. The integers are returned as a *list* which is one of Python's built-in sequence data types and is one of the things that makes Python so attractive to use. Finally the list of integers is assigned to a variable with the name `values`. Python differs from many languages in that variables do not need to be declared as being of a particular type. In C, for example, an integer variable would have to be declared with a statement such as "`int i ;`" before it could be used. These declarative statements do not exist in Python and so `values` can be assigned arbitrarily to refer to any data type.

Line 13 shows one of the forms of *iteration* in Python. The statement takes the list referred to by `values` and assigns each of its elements to the variable `i` in turn. The iteration stops when the end of the list is reached. The lines over which iteration is to occur are determined by line identation in exactly the same way as those in the body of a function.

Line 14 is the first line of the loop specified by the `for` construct in *line 13*. It takes the integer referred to by the variable `i`, converts it to a *floating-point number* using the built-in function `float` and then assigns it to the variable `x`.

Line 15 is printed as two lines in the text, due to the restricted page width, but it is logically a single statement. The presence of the backslash character (\) at the end of the line indicates to Python that the subsequent line is to be treated as a continuation of the current one.

The statement prints the values of `i` and of `x`, the square root of `x`, which is calculated by invoking the function `sqrt` from the module `math`, and the square of `x`, calculated using the previously defined function `Square`. These items are grouped together at the end of the line in a *tuple*. Tuples, like lists, are one of Python's built-in sequence data types and are constructed by enclosing the items that are to be in the tuple in parentheses. Tuples differ from lists, though, in that they are *immutable*, which means that their contents cannot be changed once they have been created.

The style in which the quantities are printed is determined by the *formatting string*, which is enclosed in double quotes `""`. This is placed after the `print` keyword and is separated from the tuple of items to be printed by the `%` character. Python employs a syntax for formatting operations that is very similar to that of the C language. Output fields start with a `%` character and so, in this example, there are four output fields in the string, one for each of the items to be printed. The first output field is `%5d`, which says that an integer, coded for by the letter `d`, is to be printed in a field `5` characters wide. The remaining fields are identical and have the form `%10.5f`. They are for the output of floating-point numbers (`f`) in fields `10` characters wide but with `5` of these characters occurring after the decimal point.

It is not, of course, possible to master a language from a single, short program but readers should gain in expertise and come to appreciate more fully the capabilities of the language as they work through the examples and exercises in the book. One of the great advantages of Python for learning is that it can be used interactively and so it is quick and easy to write simple programs to test whether one really understands what the language is doing.

1.3 Object-oriented programming

Python admits various programming styles but all the modules in pDynamo are written using an *object-oriented* approach in which the basic unit of programming is the *class*. A class encapsulates the notion of an *object*, such as a file or a molecule, and groups together the data or *attributes* needed to describe the object and the functions or *methods* that are required to manipulate it. Classes are used by *instantiating* them so that, for example, a program for modeling the molecules methanol and water would create two instances of the class molecule, one to represent methanol and one water.

There are other important aspects of object-oriented programming that pDynamo employs but which will only be alluded to briefly, if at all, later on. Two of these are *inheritance* and *polymorphism*. Inheritance is the mechanism by which a new class is defined in terms of an existing one. For example, a class for manipulating organic molecules could be derived from a more general class for molecules. The new class would inherit all the attributes and methods defined by its parent class but would also have attributes and methods specific for organic molecules. Polymorphism is related to inheritance and is the ability to redefine methods for derived classes. Thus, the general molecule class could have a method for chemical reactions but this would