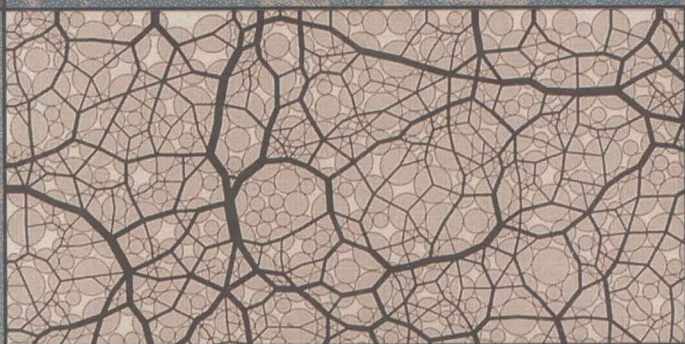
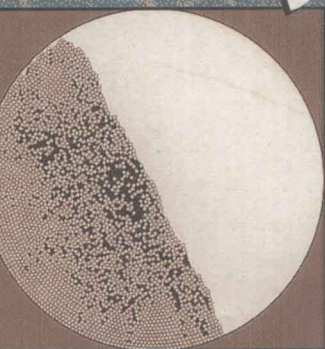


K. H. Hoffmann · M. Schreiber
Editors

Computational Physics

Selected Methods
Simple Exercises
Serious Applications



Springer

Karl Heinz Hoffmann
Michael Schreiber (Eds.)

Computational Physics

Selected Methods
Simple Exercises
Serious Applications

With 145 Figures, 15 Tables
and a 3.5" MS-DOS Diskette



Springer

Professor Dr. Karl Heinz Hoffmann
Professor Dr. Michael Schreiber

Institut für Physik
Technische Universität Chemnitz-Zwickau
D-09107 Chemnitz
Germany

Library of Congress Cataloging-in-Publication Data.

Hoffmann, K. H. (Karl Heinz), 1953–. Computational physics : selected methods, simple exercises, serious applications / K. H. Hoffmann, M. Schreiber. p. cm. Includes bibliographical references and index.

ISBN 3-540-60689-0 (Berlin : alk. paper)

1. Physics—Problems, exercises, etc. — Methodology. 2. Numerical calculations. 3. Mathematical Physics. I. Schreiber, Michael, 1954–. II. Title.

QC32.H67 1996 530'.01'13—dc20 96-2830

ISBN 3-540-60689-0 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1996
Printed in Germany

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Please note: Before using the programs in this book, please consult the technical manuals provided by the manufacturer of the computer – and of any additional plug-in-boards – to be used. The authors and the publisher accept no legal responsibility for any damage caused by improper use of the instructions and programs contained herein. Although these programs have been tested with extreme care, we can offer no formal guarantee that they will function correctly. The programs on the enclosed disc are under copyright protection and may not be reproduced without written permission from Springer-Verlag. One copy of the program may be made as a back-up, but all further copies offend copyright law.

Typesetting: Camera-ready copy from the editors using a Springer T_EX macro package
Cover design: Erich Kirchner, Heidelberg

SPIN 10525808

56/3144 – 5 4 3 2 1 0 – Printed on acid-free paper

Preface

Computational physics is the field in physics that has experienced probably the most rapid growth in the last decade. With the advent of computers, a new way of studying the properties of physical models became available. One no longer has to make approximations in the analytical solutions of models to obtain closed forms, and interesting but intractable terms no longer have to be omitted from models right from the beginning of the modeling phase. Now, by employing methods of computational physics, complicated equations can be solved numerically, simulations allow the solution of hitherto untractable problems, and visualization techniques reveal the beauty of complex as well as simple models. Many new and exciting results have been obtained by numerical calculations and simulations of old and new models.

This book presents samples of many of the facets that constitute computational physics. Our aim is to cover a broad spectrum of topics, and we want to present a mixture ranging from simple introductory material including simple exercises to reports of serious applications. This is not meant to be an introductory textbook on computational physics, nor is it a proceedings volume of a research conference. This book instead provides the reader with an overview of computational physics, its basic methods, and its many areas of application. Our coauthors lead the reader into new and “hot” topics of research, but the presentation does not require any specific knowledge of the topics and methods. We hope that a reader who has gone through the book can appreciate the wealth of computational physics and is motivated to proceed with further reading.

The topics covered in this book cover a wide spectrum, with a coarse division into “Monte Carlo” type and “molecular dynamics” type chapters. We start with discussing random numbers and their generation on computers. Then these random numbers are used in a variety of applications, which center around “Monte Carlo methods”. In these applications the focus is first on classical systems in physics, chemistry, biology, material science, and optimization. Then quantum-mechanical problems are investigated by Monte Carlo procedures. On our way we also encounter quantum chaos and fractal concepts, which are of increasing importance nowadays. The transition from “Monte Carlo” to “molecular dynamics” occurs in the chapter on hybrid methods, which combine elements of both. Then “molecular dynamics” methods are presented, with fluids and solids covered. A chapter on finite-element methods follows, and the two final chapters present principles of parallel computers and associated programming models.

As usual in physics, only active interaction with the matter at hand provides

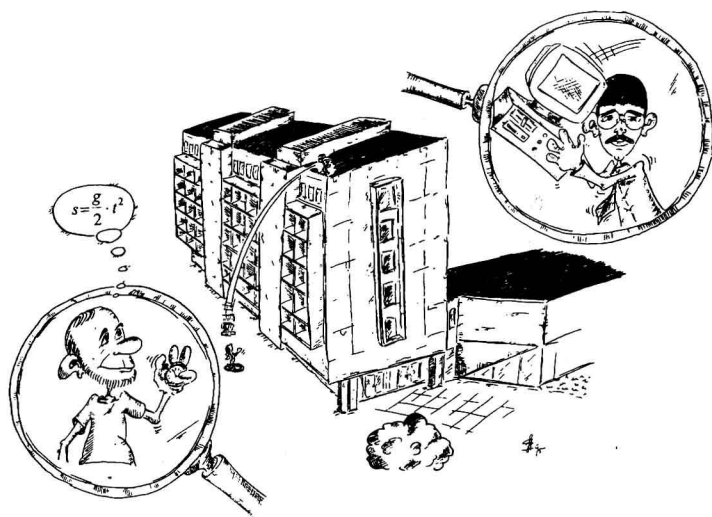
deep insight, and thus we include a diskette that contains sample programs and demonstrations to support the interaction of the reader with the text. The sample programs and demonstrations are selected to provide a glimpse of current research activities, even though the limitations of the available hardware and/or the limited patience of some readers might require a reduction in the dimensionality or size of the application. Also some exercises are included to further foster an active use of this book.

The material in this book is born out of lectures the authors gave at a Heraeus Summer School on computational physics at the Technical University in Chemnitz. The aim of the summer school was the same as the aim of this book: to give a sampler of the field. Due to the gracious funding by the Dr. Wilhelm Heinrich Heraeus and Else Heraeus Foundation the editors (see figure) were able to present two weeks of intense lecturing and “learning by doing” to more than 80 students. We would like to use this opportunity to thank the Heraeus Foundation for making the summer school and this book possible.

But most important we like to thank our coauthors for their contributions to this volume (as well as for their lectures at the summer school). We very much appreciate their willingness to contribute even under the severe limitations that their everyday teaching and research activities (and administrative duties) put on their time. And finally we thank Jörg Arndt, Peter Blaudeck, Andre Fachat, Göran Hanke, Karin Kumm, Sven Schubert, and Peter Späht for their technical help and Springer-Verlag for making this volume a reality.

Chemnitz, December 1995

Karl Heinz Hoffmann and Michael Schreiber



With this original answer to the question “How to measure the height of the building of the Institut für Physik in Chemnitz with a computer and a stop watch only?” the editors give a peculiar interpretation of the topic “Physics with a computer”.

Contents

Random Number Generation*

Dietrich Stauffer	1
1 Introduction	1
2 The Miracle Number 16807	2
3 Bit Strings of Kirkpatrick–Stoll	4
4 A Modern Example	5
5 Problems	6
6 Summary	8
References	8

A Few Exercises with Random Numbers

Peter Blaudeck	9
--------------------------	---

Monte Carlo Simulations of Spin Systems*

Wolfgang Janke	10
1 Introduction	10
2 Spin Models and Phase Transitions	11
2.1 Models and Observables	11
2.2 Phase Transitions	12
3 The Monte Carlo Method	17
3.1 Estimators and Autocorrelation Times	18
3.2 Metropolis Algorithm	19
3.3 Cluster Algorithms	20
3.4 Multicanonical Algorithms for First-Order Transitions	25
4 Reweighting Techniques	26
5 Applications to the 3D Heisenberg Model	30
5.1 Simulations for $T > T_c$	31
5.2 Simulations near T_c	33
6 Concluding Remarks	36
Appendix: Program Codes	39
References	40

* Software included on the accompanying diskette.

Metastable Systems and Stochastic Optimization*

Karl Heinz Hoffmann	44
1 An Introduction to Complex Systems	44
2 Dynamics in Complex Systems	46
2.1 Thermal Relaxation Dynamics: The Metropolis Algorithm	46
2.2 Thermal Relaxation Dynamics: A Marcov Process	47
2.3 Thermal Relaxation Dynamics: A Simple Example	48
3 Modeling Constant-Temperature Thermal Relaxation	49
3.1 Coarse-Graining a Complex State Space	50
3.2 Tree Dynamics	51
3.3 A Serious Application: Aging Effects in Spin Glasses	53
4 Stochastic Optimization: How to Find the Ground State of Complex Systems	55
4.1 Simulated Annealing	55
4.2 Optimal Simulated Annealing Schedules: A Simple Example	56
4.3 Adaptive Annealing Schedules and the Ensemble Approach to Simulated Annealing	57
5 Summary	60
Appendix: Examples and Exercises (with S. Schubert)	60
References	62

Modelling and Computer Simulation of Granular Media

Dietrich E. Wolf	64
1 The Physics of Granular Media	64
1.1 What are Granular Media?	64
1.2 Stress Distribution in Granular Packing: Arching	65
1.3 Dilatancy, Fluidization and Collisional Cooling	67
1.4 Stick-and-Slip Motion and Self-Organized Criticality (with S. Dippel)	70
1.5 Segregation, Convection, Heaping (with S. Dippel)	71
2 Molecular Dynamics Simulations I: Soft Particles	75
2.1 General Remarks	75
2.2 Normal Force	76
2.3 Tangential Force	78
2.4 Detachment Effect	80
2.5 Brake Failure Effect (with J. Schäfer)	81
3 Molecular Dynamic Simulations II: Hard Particles (with J. Schäfer)	83
3.1 Event-Driven Simulation	83
3.2 Collision Operator	83
3.3 Limitations	84
4 Contact Dynamics Simulations (with L. Brendel and F. Radjai)	84
4.1 General Remarks	84
4.2 Contact Laws and Equations of Motion	86
4.3 Iterative Determination of Forces and Accelerations	88

4.4	Results	89
5	The Bottom-to-Top Restructuring Model	89
5.1	The Algorithm and its Justification (with E. Jobs)	89
5.2	Simulation of a Rotating Drum (with T. Scheffler and G. Baumann)	91
6	Conclusion	92
	References	93

Algorithms for Biological Aging*

	Dietrich Stauffer	96
1	Introduction	96
2	Concepts and Models	97
3	Techniques	98
4	Results	99
	References	101

Simulations of Chemical Reactions

	Alexander Blumen, Igor Sokolov, Gerd Zumofen, and Joseph Klafter	102
1	Introduction	102
2	The Basic Kinetic Approach	102
3	Numerical and Analytical Approaches for Reactions Under Diffusion	104
4	Reactions in Layered Systems	109
5	Reactions Under Mixing	113
6	Reactions Controlled by Enhanced Diffusion	116
	References	119

Random Walks on Fractals*

	Armin Bunde, Julia Dräger, and Markus Porto	121
1	Introduction	121
2	Deterministic Fractals	122
2.1	The Koch Curve	122
2.2	The Sierpinski Gasket	124
3	Random Fractals	124
3.1	The Random-Walk Trail	124
3.2	Self-Avoiding Walks	125
3.3	Percolation	126
4	The "Chemical Distance" ℓ	127
5	Random Walks on Fractals	131
5.1	Root Mean Square Displacement $R(t)$	131
5.2	The Mean Probability Density	131
6	Biased Diffusion	138
7	Numerical Approaches	140
7.1	Generation of Percolation Clusters	141
7.2	Simulation of Random Walks	142

8 Description of the Programs	143
References	145

Multifractal Characteristics of Electronic Wave Functions in Disordered Systems*

Michael Schreiber	147
1 Electronic States in Disordered Systems	147
2 The Anderson Model of Localization	148
3 Calculation of the Eigenvectors	151
4 Description of Multifractal Objects	154
5 Multifractal Analysis of the Wave Functions	156
6 Computation of the Multifractal Characteristics	160
7 Topical Results of the Multifractal Analysis	162
References	165

Transfer-Matrix Methods and Finite-Size Scaling for Disordered Systems*

Bernhard Kramer and Michael Schreiber	166
1 Introduction	166
2 One-Dimensional Systems	167
2.1 The Transfer Matrix	168
2.2 The Ordered Limit	169
2.3 The Localization Length	170
2.4 Resolvent Method	172
3 Finite-Size Scaling	175
4 Numerical Evaluation of the Anderson Transition	177
4.1 Localization Length of Quasi-1D Systems	177
4.2 Dependence of the Localization Length on the Cross Section	179
4.3 Finite-Size Scaling Numerically	182
5 Present Status of the Results from Transfer-Matrix Calculations	185
References	187

Quantum Monte Carlo Investigations for the Hubbard Model*

Hans-Georg Matuttis and Ingo Morgenstern	189
1 Introduction	189
1.1 The Hubbard Model	189
1.2 What to Compute	191
1.3 Quantum Simulations	191
2 Grand Canonical Quantum Monte Carlo	192
2.1 The Trotter-Suzuki Transformation	192
2.2 The Hubbard-Stratonovich Transformation	194
2.3 The Partition Function	196
2.4 The Monte Carlo Weight	198
3 Equal-Time Greens Functions	199

3.1	Single Spin Updates	200
3.2	Numerical Instabilities	200
4	History and Further Reading	201
	Appendix A: Statistical Monte Carlo Methods	202
	Appendix B: OCTAVE	203
	Appendix C: Exercises	205
	References	207

Quantum Dynamics in Nanoscale Devices*

	Hans De Raedt	209
1	Introduction	209
2	Theory	212
3	Data Analysis	214
4	Implementation	215
5	Application: Quantum Interference of Two Identical Particles	219
	References	223

Quantum Chaos

	Hans Jürgen Korsch and Henning Wiescher	225
1	Classical and Quantum Chaos	225
2	Quantum Time Evolution	227
3	Quantum State Tomography	229
3.1	Phase-Space Distributions	229
3.2	Phase-Space Entropy	230
4	Case Study: A Driven Anharmonic Quantum Oscillator	231
4.1	Classical Phase-Space Dynamics	232
4.2	Quantum Phase-Space Dynamics	232
4.3	Quasienergy Spectra	238
4.4	Chaotic Tunneling	239
5	Concluding Remarks	243
	References	243

Numerical Simulation in Quantum Field Theory*

	Ulli Wolff	245
1	Quantum Field Theory and Particle Physics	245
1.1	Particles, Fields, Standard Model	245
1.2	Beyond Perturbation Theory	246
2	Lattice Formulation of Field Theory	247
2.1	Path Integral	247
2.2	Lattice Regularization	249
2.3	Field Theory and Critical Phenomena	250
2.4	Effective Field Theory	251
3	Stochastic Evaluation of Path Integrals	252
3.1	Monte Carlo Method	253

3.2 Metropolis Algorithm for φ^4	254
4 Summary	255
Appendix: FORTRAN Monte Carlo Package for φ^4	255
References	256

Modeling and a Simulation Method for Molecular Systems

Dieter W. Heermann	258
1 Introduction	258
2 Brief Review of the Simulation Method	258
3 Modeling of Polymer Systems	260
4 Coarse-Graining	261
5 The Monomer Unit	262
6 Bonded Interactions for BPA-PC	263
7 Parallelization of the Polymer System	264
References	266

Constraints in Molecular Dynamics, Nonequilibrium Processes in Fluids via Computer Simulations

Siegfried Hess	268
1 Introduction	268
2 Basics of Molecular Dynamics	269
2.1 Equations of Motion	269
2.2 Extraction of Data from MD Simulations	270
3 Potentials, Constraints, and Integrators	270
3.1 Interaction Potential and Scaling	270
3.2 Thermostats	272
3.3 Integrators	276
4 Nonequilibrium Phenomena	278
4.1 Relaxation Processes	278
4.2 Plane Couette Flow	280
4.3 Viscosity	281
4.4 Structural Changes	283
4.5 Colloidal Dispersions	284
4.6 Mixtures	284
5 Complex Fluids	285
5.1 Polymer Melts	285
5.2 Nematic Liquid Crystals	287
5.3 Ferrofluids and Magneto-Rheological Fluids	290
References	291

Molecular-Dynamic Simulations of Structure Formation in Complex Materials

Thomas Frauenheim, Dirk Porezag, Thomas Köhler, and Frank Weich . . . 294

1	Introduction	294
2	Simulation Methods	295
3	Total Energies and Interatomic Forces	297
	3.1 Classical Concepts	297
	3.2 Density-Functional Theory, Car-Parrinello MD	299
4	Density-Functional Based Tight-Binding Method	302
	4.1 Creation of the Pseudoatoms	303
	4.2 Calculation of Matrix Elements	304
	4.3 Fitting of Short-Range Repulsive Part	305
5	Vibrational Properties	306
6	Simulation Geometries and Regimes	307
	6.1 Clusters, Molecules	307
	6.2 Bulk-Crystalline and Amorphous Solids	308
	6.3 Surfaces and Adsorbates	309
7	Accuracy and Transferability	310
	7.1 Small Silicon Clusters, Si_n	310
	7.2 Molecules, Hydrocarbons	310
	7.3 Solid Crystalline Modifications, Silicon	314
8	Applications	315
	8.1 Structure and Stability of Polymerized C_{60}	315
	8.2 Stability of Highly Tetrahedral Amorphous Carbon, <i>ta</i> -C	319
	8.3 Diamond Surface Reconstructions	321
9	Summary	325
	References	326

Finite Element Methods for the Stokes Equation

Jochen Reichenbach and Nuri Aksel 329

1	Introduction	329
2	Stokes Equation	330
	2.1 Conservation Equations	330
	2.2 Function Spaces and Variational Formulation	330
	2.3 Saddle Point Problem	332
	2.4 General Boundary Conditions	333
	2.5 Example	335
3	Discretization	336
	3.1 General Formulation	336
	3.2 Finite Elements for Saddle-Point Problems	337
4	Final Remarks	340
	References	340

**Principles of Parallel Computers
and Some Impacts on Their Programming Models**
Wolfgang Rehm and Thomas Radke 341

1 Introduction 341

2 Overview on Architecture Principles 341

3 General Classification 343

4 Multiprocessor Systems 344

5 Massively Parallel Processor Systems 347

6 Multiple Shared-Memory Multiprocessors 348

7 Multithreading Programming Model 349

8 Message-Passing Programming Model 351

9 Summary 352

References 353

Parallel Programming Styles: A Brief Overview
Andreas Munke, Jörg Werner, and Wolfgang Rehm 354

1 Introduction 354

2 Programming Models 354

 2.1 Definition 354

 2.2 Classification 355

3 Programming a Shared Memory Computer 356

 3.1 The KSR Programming Model 356

 3.2 Levels of Parallelism 357

 3.3 Program Implementation 357

 3.4 Examples 358

4 Programming a Distributed Memory Computer Using PARIX 362

 4.1 What is PARIX 362

 4.2 PARIX Hardware Environment 363

 4.3 Communication and Process Model Under PARIX 363

 4.4 Programming Model 364

 4.5 An Example, PARIX says “Hello World” 365

5 Programming Heterogenous Workstation Clusters Using MPI 367

 5.1 Introduction 367

 5.2 Basic Structure of MPICH 368

 5.3 What Is Included in MPI? 369

 5.4 What Does the Standard Exclude? 370

 5.5 MPI Says “Hello World” 370

 5.6 Current Available Implementations of MPI 373

6 Summary 373

References 373

Index 375

Random Number Generation*

Dietrich Stauffer

Institut für Theoretische Physik, Universität zu Köln, D-50923 Köln, Germany
e-mail: stauffer@thp.uni-koeln.de

Abstract. The sad situation of random number generation is reviewed: there are no good random numbers. But life has to go on anyhow, and thus we explain how to produce reasonable random numbers efficiently, emphasizing multiplication with 16807 and the Kirkpatrick-Stoll R250 generator.

1 Introduction

Molecular Dynamics and Monte Carlo are the two standard simulation methods of the last decades. Monte Carlo simulations use random numbers to produce random fluctuations. Today, they are no longer made at the roulette tables in Monaco, but on computers. In the good old days, people printed tables of random numbers from which the user could read them off. This, of course, is somewhat tedious when simulating a square lattice of size one million times one million, today's world record [1]. About a decade ago, computer chips became available which produced random numbers through the thermal noise of the electrons, about one number per microsecond. This is not fast enough for many quality applications. Besides, for testing purposes we would like to have *reproducible* random numbers: when we have made a program more efficient without changing the results, we want to run it again and indeed get exactly the same results, and not just roughly the same, within the statistical errors of the Monte Carlo simulations. Moreover, when we switch from one computer to another, we would like again to get the same results: portability is important. Thus special chips using thermal noise are not suitable for this purpose.

Also, the random numbers should be produced quickly since Monte Carlo simulations consume lots of time and we never have enough of it. Thus we need efficient methods, and on many computers it is very slow to call a function or subroutine to produce one random number. Thus a good random number generator should be:

- (1) random
- (2) reproducible
- (3) portable
- (4) efficient

Using the built-in random number generator of your computer can make your program inefficient and nonportable. (Seymour Cray knew what he was doing:

* Software included on the accompanying diskette.

his random number generators for the good old CDC series or modern Crays were efficient.) Besides, the user then does not understand what is going on.

Thus we now review why the above criteria are difficult to fulfill and what to do about it, by programming your own random numbers.

2 The Miracle Number 16807

Linear congruential random number generators multiply the last random integer by some big factor, add another integer to it, treat the sum modulo some power of two, and normalize this integer to the interval between zero and unity. This all sounds very complicated, sometimes is presented in this complicated fashion in the literature, and may cause you to give up programming your own random numbers. Thus simply forget these complications and look at the following Fortran or Basic statement, which works for most 32-bit machines:

```
IBM = IBM*16807
```

(fans of Pascal and C should end this line with a semicolon; and enemies of International Business Machines may use a different variable name). If you start with an odd integer for IBM, e.g., through $IBM = 2 * ISEED - 1$, then this single program line should give you, again and again, integers IBM distributed randomly between -2^{31} and 2^{31} . Just try it out. Why does it work?

If you multiply two ten-digit integers, the result will be an integer with about twenty digits and is difficult to obtain by paper and pencil. You may estimate, however, the leading digit correctly without too much effort. On a computer, you may not be able to store more than ten digits for each integer. Then most computers simply throw away, without any error message, the somewhat predictable leading digits and keep only the ten least significant digits. Of course, computers work with binary digits (bits) and not with decimal ones, and with four-byte integers (32 bits) all leading bits beyond the least significant 32 bits are thrown away if the product of two integers has more than 32 bits. In terms of decimal numbers restricted to be at most 999, this would mean that the product of 123 and 899 is not 110577 but merely 577. It is clear that these least significant digits are difficult to predict, that means for a user they look pretty random.

In your youth you have learned that $a*b$ equals $b*a$, and that the product of two positive numbers is again positive. In linear algebra or quantum mechanics you found out that the first statement was a lie, and now you realize the same for the second statement: $IBM*16807$ may be negative even when IBM was positive. The reason is that the first (most significant) bit of an integer indicates the sign. Thus before the leading bits of the product were thrown away, the product was positive; but then only the last 32 bits were kept, and the leftmost (most significant) bit may be zero (positive 32-bit number) or one (negative 32-bit number). So, plus times plus is minus, in about half the cases.

Some ancient DEC computers may not have liked this overflow above the 32-bit limit, but otherwise I am not aware of computers where the above Fortran statement causes trouble. Thus we have not only an efficient one-line random number generator, but also a portable one.

If for some reason you want only positive random numbers IBM, then you have to add 2^{31} to them if they are negative. This number 2^{31} is too large to handle for the 32-bit computer, but $2^{31} - 1 = 2147483647$ is fine. Thus try

```
IF(IBM.LT.0) IBM = IBM + 2147483647 + 1
```

and it works if the computer is too stupid to find out that you really want to add 2^{31} .

If you want to normalize this number to the interval between 0 and 1, you multiply a positive random integer by $2^{-31} = 4.656612 \times 10^{-10}$. If they are both positive and negative, use

```
Z = 0.5+2.328306 E-10 * IBM
```

to get a random number z between 0 and 1. Of course, this normalization from an integer to a real number costs a lot of computer time; you could do it faster by learning how a floating point number is stored in your computer and then constructing one via bit operations treating the random integer as a bit string for the mantissa.

However, in most cases this normalization is not needed, and you may stay within integer arithmetic. For example, some command GOTO 1 should be executed with probability p . Normally this is done with

```
IF(Z.LE.P) GOTO 1
```

requiring a random number between 0 and 1. This normalization is avoided by

```
IF(IBM.LE.IP) GOTO 1
```

provided you have defined once (and not millions of times, i.e., for each random number) the variable $IP = 2147483648.0*(2.0*P-1.0)$

```
IF(P.GT.0.999) IP=2147483647
```

```
IF(P.LT.0.001) IP=-2147483648
```

which varies between -2^{31} and 2^{31} . Now the computer runs faster. (The last two "if" statements are precautions, seldomly needed, in case rounding errors cause trouble in the conversion to integers if $p = 0$ or $= 1$.)

The number $16807 = 7^5$ is not entirely arbitrary; historically earlier was 65539, and 65549 has also been used. So you may mix them, using in most of your program lines multiplication with 16807, but sometimes also 65539. Do not try to produce different samples just by changing the multiplier from 16807 to 16809, then 16811, and so on. Also, your IBM numbers must always be odd integers; to be safe I start with an integer ISEED and then state once $IBM = 2*ISEED-1$, as mentioned already above.

If you simulate at zero temperature (see Sect. 4), then the probabilities are 0, $1/2$, and 1 only. With integer random numbers IBM varying between -2^{31} and $+2^{31}$ the conditions and Boltzmann integers then have to be formulated exactly as stated above (not IBM .LT. IP for example), to avoid a spin flipping when it should not flip. With floating point random numbers you need double precision real*8. This detail may be important for the fraction of frozen spins in spin glasses [7].

3 Bit Strings of Kirkpatrick–Stoll

The principle of Tausworth shift generators has been around for a long time but physicists started to use it mainly after Kirkpatrick and Stoll made it popular in a physics journal [2]. For many years it was regarded as superior to multiplication with 16807; this is no longer true but at least it offers a completely different alternative. It requires bit-manipulation functions which had not yet been standardized before Fortran 90 (they are part of the C language standard) and which were initially demanded by the Pentagon for signal analysis.

Imagine you have two 32-bit integers M and N . Then the exclusive-or operation puts a bit equal to one if and only if the two corresponding bits in M and N are different; otherwise exclusive-or puts this bit to zero. Thus this bit-by-bit exclusive-or $\text{IEOR}(M, N)$ (Cray called it $M.\text{xor}.N$ but unfortunately this did not become the standard) treats 32 bits in parallel and does for each of these bits what a logical operation would do for one bit only with logical (Boolean) variables. Obviously such bit-handling operations can be used in lots of problems where the essential information consists of independent bits, such as in Ising models or cellular automata where it is called multispin coding [3]. Fortran manuals usually hide these tricks in an appendix on the functions which the compiler has stored.

Imagine you have an array of 250 integers N consisting of completely random bits. Then the next integer $N(251)$ is produced via $N(251) = \text{IEOR}(N(1), N(148))$ and generally

$$N(K) = \text{IEOR}(N(K-250), N(K-103))$$

where again 250 and 103 are magic numbers which should not be changed. An alternative choice is the simple subtraction

$$N(K) = N(K-250) - N(K-103)$$

but this is less widespread than the exclusive-or method, also called R250.

To work with it we first need 250 random integers. It is not recommended to take the results of IBM*16807 directly as such integers since the last bits are not random enough; for example the least significant bit is always one since IBM is always odd. Instead we set a bit in N equal to one if and only if the result of IBM*16807 is negative. Thus our 32-bit integers N are initialized through

```
DO K = 1, 250
  ICI=0
  DO I=1,32
    ICI=ISHFT(ICI,1)
    IBM=IBM*16807
    IF(IBM.LT.0) ICI=ICI+1
  ENDDO
  N(K)=ICI
ENDDO
```

Here again ISHFT is a bit-manipulation function shifting the first argument by one bit to the left. Instead of ICI=ICI+1 , one could also have used a bit-by-bit or-function ICI=IOR(ICI,1) ; on most compilers integer and bitstring operations