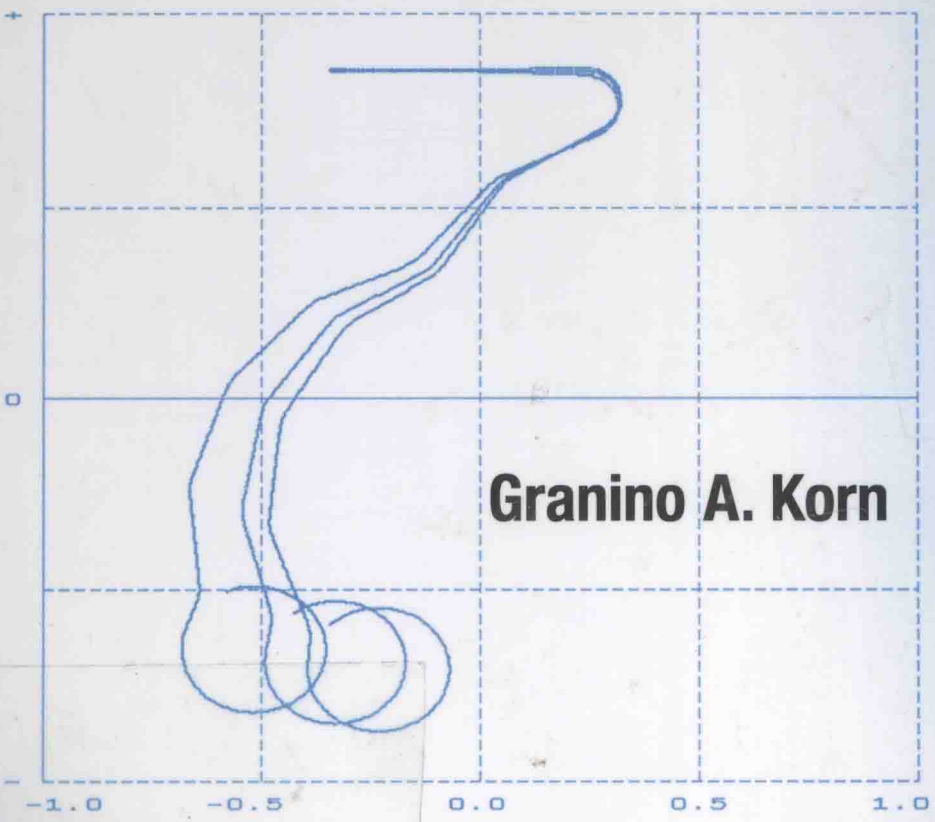
SECOND EDITION

Advanced Dynamic-System Simulation

Model Replication and Monte Carlo Studies



Granino A. Korn

INCLUDES CD-ROM

WILEY

SECOND EDITION

ADVANCED DYNAMIC-SYSTEM SIMULATION

Model Replication and Monte Carlo Studies

GRANINO A. KORN
University of Arizona

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Korn, Granino A. (Granino Arthur), 1922–
Advanced dynamic-system simulation : model replication and Monte Carlo studies / by Granino A. Korn. – Second edition.
pages cm
Includes bibliographical references.
ISBN 978-1-118-39735-0 (hardback)
1. System analysis–Simulation methods. 2. Open source software. 3. Computer software–Development. I. Title.
QA402.K665 2013
003′.85–dc23

2012034771

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

Simulation is experimentation with models. In this book we describe efficient interactive computer programs that model dynamic systems such as control systems, aerospace vehicles, and biological systems. Simulation studies can involve many hundreds of model changes, so programs must be fast and user-friendly.

For hands-on experiments with each program example, the book CD now includes industrial-strength open-source simulation software for both Windows™ and Linux, not just toy demonstration programs. The Desire modeling/simulation program implements very fast and respectably large simulations on personal computers. Runtime-compiled programs display results immediately to permit true interactive modeling.

A readable mathematical notation, for example,

```
x = 23.4  |  alpha = 0

d/dtx = -x*cos(w*t) + 2.22*a*x

Vector y = A*x + B*u
```

lets readers try different parameter values without learning details of the programming language. Note that one can read ebook pages and run live simulations on the same computer display.

In Chapter 1 we introduce our subject with a few familiar differential-equation models and a small guided-missile simulation. The remainder of the book presents more advanced topics; most of our example programs were rewritten to clarify the modeling technique and to increase computing speed.

Chapter 2 begins with a newly revised systematic procedure for programming ***difference equations*** and applies this to model plants with digital controllers. We then discuss ***limiters and switches*** and model useful devices such as track/hold circuits, trigger circuits, and signal generators with simple difference equations. Last but not least, we propose a simplified technique for numerical integration of switched variables.

Advanced simulation programs must handle differential and difference equations with vector and matrix assignments. In Chapter 3 we introduce runtime ***vector compilation***. This speeds up conventional vector and matrix operations, but more significantly, personal computers can now implement model ***replication*** (***vectorization***), a technique originally developed for supercomputers. A single vector-model run replaces hundreds or thousands of conventional simulation runs. Chapter 3 also demonstrates the convenience of user-defined ***submodels***.

In the remaining chapters we describe applications of vectorization. In Chapter 4 we discuss parameter-influence studies and introduce ***vectorized statistics***

computation, including rapid estimation of probability densities. We then introduce ***Monte Carlo simulation of random processes***. In Chapter 5 we apply Monte Carlo simulation to several real engineering systems. Vectorization lets us study ***time histories of random-process statistics***. An inexpensive 3-GHz personal computer running under 64-bit Linux can exercise over 1000 random-input control-system models in 1 second.

In Chapters 6 and 7 we demonstrate ***vector models of neural networks***; our simple vector notation has been particularly useful for short courses on neural networks. In Chapter 6 we apply backpropagation, functional-link, and radial-basis-function networks to classical regression and pattern-classification problems and describe several competitive-learning schemes. In the newly added Chapter 7 we turn to ***dynamic neural networks*** for prediction, pattern classification, and model matching. The chapter includes a new method for online prediction and simplified programs for recurrent networks.

Chapter 8 deals with vectorized programs for ***fuzzy-set controllers***, ***partial differential equations***, and ***agroecological models replicated at over 1000 points of a landscape map***. The Appendix gets a small selection of reference material out of the way of the main text.

The writer would like to express his sincere thanks to Professor M. Main of the University of Colorado for his assistance with Windows graphics, to Dr. R. Wieland of the Leibniz Center for Agricultural Landscape Research (ZALF) for much good advice, and above all to Theresa M. Korn for her consistent help with this and many other projects.

GRANINO A. KORN

Wenatchee, Washington

CONTENTS

CHAPTER 1

DYNAMIC-SYSTEM MODELS AND SIMULATION

SIMULATION IS EXPERIMENTATION WITH MODELS

1-1. Simulation and Computer Programs

Simulation is experimentation with models. For system design, research, and education, simulations must not only construct and modify many different models but also store and access a large volume of results. That is practical only with models programmed on computers [1, 2].

In this book we model changes of system variables with time; we represent physical time by the simulation time variable **t**. Our models then attempt to predict different time histories **y1 = y1(t)**, **y2 = y2(t)**, ... of system variables such as velocity, voltage, and biomass. ***Static models*** simply relate values of system variables **x(t), y(t)**, ... at the same time **t**; a gas pressure **P(t)**, for instance, might be a function **P = aT** of the slowly changing temperature **T(t)**.

Dynamic-system models predict values of model-system ***state variables*** **x1(t)**, **x2(t)**, ... by relating them to past states **[x1(t), x2(t), ...]** (Sec. 1-2). Computer simulation of such systems was applied first in the aerospace industry. Simulation is now indispensable not only in all engineering disciplines, but also in biology, medicine, and agroecology. At the same time, discrete-event simulation gained importance for business and military planning.

Simulation is most effective when it is combined with mathematical analyses. But simulation results often provide insight and suggest useful decisions where exact analysis is difficult or impossible. This was true for many early control-system optimizations. As another example, Monte Carlo simulations simply measure statistics over repeated experiments to solve problems too complicated for explicit probability-theory analysis. All simulation results must eventually be validated by real experiments, just like analytical results.

Computer simulations can be speeded up or slowed down for the experimenter's convenience. One can simulate a flight to Mars or to Alpha Centauri in one second. Periodic clock interrupts synchronizing suitably scaled simulations with real time permit "hardware in the loop": One can "fly" a real autopilot—or a human pilot—on a tilt table controlled by computer flight simulation. In this book we are interested

Advanced Dynamic-System Simulation: Model Replication and Monte Carlo Studies, Second Edition. Granino A. Korn.

in very fast simulation because we need to study many different model changes very quickly. Specifically, we would like to

- ***enter and edit programs*** in convenient editor windows.
- use ***typed or graphical-interface commands*** to start, stop, and pause simulations, to select displays, and to make parameter changes. Displays of simulation results ought to ***appear immediately*** to provide an intuitive "feel" for the effects of model changes (***interactive modeling***).
- ***program*** systematic parameter-optimization studies and produce cross-plots and statistics.

1-2. Dynamic-System Models

(a) Difference-Equation Models[1]

The simplest way to relate present values **x(t)** and past values **x(t − Δt)** of a ***state variable*** **x = x(t)** is a difference equation such as the simple recurrence

$$\mathbf{x(t) = F[x(t), x(t - \Delta t)]}$$

More general difference-equation models relate several state variables and their past values. In Chapter 2 we discuss such models in detail.

(b) Differential-Equation Models

Much of classical physics and engineering is based on ***differential-equation models*** that relate delayed interactions of continuous ***differential-equation state variables*** **x1(t), x2(t), ...** with first-order ordinary differential equations (***state equations***)[2]

$$\mathbf{(d/dt)\ xi = fi(t; x1, x2, \ldots; y1, y2, \ldots; a1, a2, \ldots)} \qquad \mathbf{(i = 1, 2, \ldots)} \quad (1\text{-}1a)$$

Here **t** again represents the time, and the quantities

$$\mathbf{yj = gj(t; x1, x2, \ldots; y1, y2, \ldots; b1, b2, \ldots)} \qquad \mathbf{(j = 1, 2, \ldots)} \quad (1\text{-}1b)$$

are ***defined variables***. **a1, a2, ...** and **b1, b2, ...** are constant ***model parameters***.

A computer-implemented ***simulation run*** exercises such a model by solving the state-equation system (1-1) to produce time histories of the system variables **xi = xi(t)** and **yj = yj(t)** for **t = t0** to **t = t0 + TMAX**. An ***integration routine*** increments the model time **t** and integrates the derivatives (1-1a) to produce successive values of **xi(t)** (Sec. 1-7), starting with given initial values **xi = xi(t0)**.

[1] We refer to recursive relations in general as difference equations, whereas some authors reserve this term for relations formulated in terms of explicit finite differences [11].

[2] We reduce higher-order differential equations to first-order systems by introducing derivatives as extra state variables. Thus, $\mathbf{d^2x/dt^2 = -kx}$ becomes

dx/dt = xdot **dxdot/dt = −kx**

(see also Sec. 1-10).

Each state variable **xi** is a model output. There are three types of defined variables **yj**:

1. model inputs (specified functions of the time **t**),
2. model outputs, and
3. intermediate results needed to compute the derivatives **fi**.

The defined-variable assignments (1-1b) must be sorted into a procedure that derives updated values for all **yj** from current values of the state variables **xi**, already computed **yj** values, and/or **t** without "algebraic loops" (Sec. 1-9).

Some dynamic systems (e.g., systems involving linkages in automotive engineering and robotics) are modeled with differential equations that cannot be solved explicitly for state-variable derivatives as in Eq. (1-1a). Simulation then requires solution of algebraic equations at each integration step. Such ***differential-algebraic-equation systems*** are not treated in this book. References 6 to 11 describe suitable mathematical methods and special software.

(c) Discussion

Much of classical physics (Newtonian dynamics, electrical-circuit theory, chemical reactions) uses differential equations. As a result, most legacy simulation programs are basically differential-equation solvers and relegate difference equations to accessory "procedural" program segments. Modern engineering systems, though, often involve digital controllers and thus sampled-data operations that implement difference equations. In this book we introduce a program package specifically designed to handle such problems. We start with differential-equation problems in Chapter 1 and go on to difference equations and mixed continuous/sampled-data models in Chapter 2.

1-3. Experiment Protocols Define Simulation Studies

Effective computer simulation is not simply a matter of programming model equations. It must also be truly convenient to modify models and to try many different experiments (see also Sec. 1-5). In addition to program segments that list model equations such as those in Sec. 1-2, every simulation needs an ***experiment-protocol program*** that sets and changes initial conditions and parameters, calls differential-equation-solving simulation runs, and displays or lists solutions.

A simple experiment protocol implements a sequence of successive commands: say

```
a = 20.0  |  b = −3.35   (set parameter values)
x = 12.0  (set the initial value of x)
drun      (make a differential-equation-solving simulation run)
reset     (reset initial values)
a = 20.1  (change model parameters)
b = b − 2.2
drun      (try another run)
.....
```

Each **drun** command calls a new simulation run. The command **reset** resets initial conditions for new runs.

A ***command intepreter*** executes typed commands immediately. Users can inspect the solution output after each simulation run and then enter new commands for another run. Command-mode operation permits interactive programming and program debugging [2].

Graphical-user-interface (GUI) simulation programs replace typed commands with windows for entering model parameters and menus and/or buttons for executing such commands as **run** and **reset** using mouse clicks. This is convenient for special-purpose simulation programs with simple experiment protocols. Typed and programmed commands entered in a console window (command window) permit a much wider choice of operations.

A programmed ***simulation study*** combines experiment-protocol commands into a stored program called an ***experiment-protocol script***. Such a program can branch and loop to call repeated simulation runs (e.g., for parameter optimization or statistical studies). Proper experiment-protocol scripts require a full-fledged computer language with functions, procedures, program loops, conditional execution, and file operations.

Simulation studies can involve many model and parameter changes, so program execution must be prompt and fast. We can ***interpret*** experiment-protocol scripts. But "dynamic" program segments that implement simulation runs update system variables hundreds or thousands of times. Such time-critical operations must be ***compiled***.[3]

1-4. Simulation Software

Equation-oriented simulation programs such as ACSL™ accept model equations in a more or less human-readable notation, sort defined-variable assignments as needed, and feed the sorted equations to a Fortran or C compiler [1]. Berkeley Madonna and Desire (see below) have runtime equation-language compilers and execute immediately. ***Block-diagram interpreters*** (e.g., Simulink™ and the free open-source Scicoslab program) let users compose block-diagram models on the display screen. Such programs execute interpreted simulation runs immediately but relatively slowly. To improve computing speed, most block-diagram interpreters admit precompiled equation-language blocks for complicated expressions, and production runs are sometimes translated into C for faster execution. Alternatively, ACSL, Easy5™, and Berkeley Madonna have ***block-diagram preprocessors*** for compiled simulation programs. Differential-algebraic (DAE) models need substantially more complicated software, preferably using the Modelica Language [3–6]. Dynasim™ and Maplesim™ are examples.

[3] ***Interpreter*** programs translate individual commands one-by-one into the computer's machine language. ***Compilers*** speed program execution by translating complete program segments.