# 1

# Introduction to System Simulation Techniques and Applications



## 1.1 Overview of System Simulation Techniques

Systems are the integrated wholes composed of interrelated and interacting entities; these could be engineering systems or non-engineering systems. Engineering systems are the whole composed of interacting components such that certain system objectives can be achieved. For instance, motor drive systems are composed of an actuating component, a power transfer component and a signal measurement component, so as to control the motor speed or position, among other objectives.

The field of non-engineering systems is much wider. From universe to micro world, any integrated whole can also be regarded as a system, since there are interrelated and interacting relationships.

In order to quantitatively study the behavior of a system, the internal characteristics and interacting relationship should be extracted, to construct a model of the system. System models can be classified as physical models and mathematical models. Following the rapid development and utilization of computer technology, the application of mathematical models is more and more popular.

A mathematical model of a system is a mathematical expression describing the dynamical behavior of the system. They can be used to describe the relationship of quantities in the system and they are the basis of system analysis and design. From the viewpoint of the type of mathematical model, systems can be classified as continuous-time systems, discrete-time systems, discrete event systems and hybrid systems. Systems can also be classified as subclasses of linear, nonlinear, time invariant, time varying, lumped parameters, distributed parameters, deterministic and stochastic systems.

System simulation is a subject within which the system behavior can be studied on the basis of the mathematical models of the actual systems. Usually computer simulation of the systems is the main topic of the subject, including the topics of systems, modeling, simulation algorithms, computer programming, display of simulation results, and validation of simulation results.

Of all the topics listed above, simulation algorithms and computer programming are the most important topics, and determine whether the original problems can be solved. The modeling and simulation result display and validation can be solved easily with MATLAB, and Simulink, the most authoritative and practical computer language. Using MATLAB and Simulink is the innovative characteristic of the whole book. In Section 1.2, a brief introduction to the historical development and future expectation of computer mathematical software and simulation languages will be given. In Section 1.3, development of MATLAB/Simulink programming is presented, and practical examples

*System Simulation Techniques with MATLAB® and Simulink®*, First Edition. Dingyü Xue and YangQuan Chen.

are explored, such that the reader can start to experience the powerful facilities of MATLAB. In Section 1.4, the main contents and the characteristic behavior of systems are presented.

## 1.2 Development of Simulation Software

Historically, computer simulation techniques went through the following stages of development: In the 1940s, analog simulation was the major way of simulation. Digital simulation began in 1950s, and in the 1960s, the first simulation languages and packages began to emerge. In the 1980s, development of object oriented simulation techniques was the leading trend. With the popularity and wide availability of digital computers, in the past 30 years, a great many professional computer simulation languages and tools have appeared, such as CSMP, ACSL, SIMNON, MATLAB/Simulink, MatrixX/System Build and CSMP-C. Because MATLAB/Simulink has become more and more popular and powerful, most of the above-mentioned simulation packages are no longer available. MATLAB/Simulink has became the de facto standard computer language and tool for system simulation.

### *1.2.1 Development of Earlier Mathematics Packages*

The rapid development of digital computers and programming languages powered research into numerical computation. In the early stages of the development of scientific computation, a lot of famous packages emerged such as the LINPACK package [1] – linear algebraic equation solver, the eigenvalue-based package EISPACK [2, 3], the NAG package [4] developed by the Numerical Algorithm Group in Oxford, and the subroutines provided in the well-established book *Numerical Recipes* [5]. These packages were very popular and had a very good reputation among the users worldwide.

The well-established EISPACK and LINPACK packages are mainly used to solve eigenvalue problems and singular value decomposition based linear algebra algorithms. These packages were all written in Fortran.

For instance, to find all the eigenvalues of a real square matrix $\boldsymbol{A}$ of size $N$, and the eigenvalues are represented by $\boldsymbol{W}_{\mathrm{R}}$ and $\boldsymbol{W}_{\mathrm{I}}$, for real and imaginary parts, and the eigenvector matrix is represented by $\boldsymbol{Z}$, the following subroutine calls are suggested in the EISPACK package

```
CALL BALANC(NM,N,A,IS1,IS2,FV1)
CALL ELMHES(NM,N,IS1,IS2,A,IV1)
CALL ELTRAN(NM,N,IS1,IS2,A,IV1,Z)
CALL HQR2(NM,N,IS1,IS2,A,WR,WI,Z,IERR)
IF (IERR.EQ.0) GOTO 99999
CALL BALBAK(NM,N,IS1,IS2,FV1,N,Z)
```

Before the above subroutine calls, you should write a piece of code to assign the matrix to the program. Then with the above statements, the main program can be written. After the compiling and linking process, the executable file can be generated. The results can finally be obtained with the executable file.

A large number of numerical subroutines are provided in the NAG package and in the book, *Numerical Recipes* [5]. The NAG package is even more professional since many more subroutines are provided. In *Numerical Recipes*, a large number of high-quality subroutines, written in C, Pascal and Fortran, are provided; these subroutines can be used directly by researchers and engineers. There

are more than 200 effective and reliable subroutines, and the subroutines are trusted by researchers worldwide.

Readers with a knowledge of Fortran and C programming might already know that, in those two programming languages, the scientific computation of matrices and graphics are rather complicated. For instance, to solve a linear algebraic equation, the elements in the matrices should be assigned first. Then a subroutine has to be written to implement the solution algorithms, such as the Gaussian elimination algorithm, and finally the result has to be output. If the subroutine written or selected is not reliable, misleading conclusions may be reached. Normally such a low-level subroutine can consist of over 100 statements. A small programming error can result in wrong conclusions.

Writing programs with packages has the following disadvantages

- **Inconvenience**. If the user is not familiar with the package being used, it might be very difficult to write programs with it, and it is always error-prone. If the slightest error is made in the program, erroneous results and misleading conclusions can be obtained.
- **Trivial procedures are involved**. A main program has to be written, and compiling and linking to the program should be made to generate executable files. A lot of effort is needed to debug the program and to validate the program.
- **Too many executables**. To solve a specific problem, a dedicated program has to be prepared. An executable file must be generated for this specific problem. The code reuse is not good, where a lot of similar problems may have to be solved.
- **Not suitable for data transfer between independent programs**. Each program can solve one particular problem. It might be difficult to transfer data from one standalone program to another. And it might not be suitable for solving one common problem by several standalone programs.
- **Difficult to allocate the array size**. In many mathematical computation problems the most important variables can be matrices. In most packages, the dimensions of the matrices might be set very low; for instance, in the package for control systems analysis and design in [6], the dimension is normally set to 10. It cannot be used to solve very high order systems.

Also, most earlier packages were written in Fortran. The plotting facilities of standard Fortran are not very good. Some other packages such as GINO-F [7] have to be used instead. However, on some platforms this package may not be available.

Apart from the above-mentioned shortcomings there is yet another difficult problem. A program written in Fortran or C cannot be easily transported to other platforms, since the source code on different platforms may not be compatible. For instance, a program written for Microsoft Windows cannot be executed at all on Linux without changes. Modifications must be made to the source code, and the source code has to be recompiled to generate executables. This is a rather difficult task, especially when plotting facilities are part of the source code.

Despite this, the development of mathematical packages is still going on. The most advanced numerical algorithms are implemented in mathematical packages, and more effective, more accurate and faster mathematical packages are still being produced. For instance, in the field of numerical linear algebra, the brand new LAPACK is becoming the leading mathematical package [8]. However, the objective of the new packages is no longer to support the average user; they are provided as low-level support to mathematical languages. In new versions of MATLAB, the base packages LINPACK and EISPACK have been abandoned, and LAPACK is used instead to provide support for linear algebra computation.

### 1.2.2 Development of Simulation Software and Languages

It can be seen from the limitations of these software packages that it might be rather complicated to complete simulation tasks with them. It is not wise to restart everything from low-level programming, and abandon the well-established packages, since the packages carry the experience and effort of scientists in the field. Low-level programming cannot achieve such a goal. Thus high-level packages and languages with good reputation, such as MATLAB/Simulink, should used instead to perform simulation tasks.

Simulation techniques gained the attention of scholars and experts worldwide, and the International Simulation Councils Inc (SCi) was founded in 1967 to formalize simulation language standards. Computer Simulation Modeling Program (CSMP) can be regarded as the earliest simulation language using that standard.

In the early 1980s, Mitchell and Gauthier Associates released a brand new simulation language ACSL (Advanced Continuous Simulation Language) [9], based on SCi's standard. Due to its powerful facilities for simulation and analysis, ACSL dominated simulation languages in relevant research communities.

In ACSL, the user should write a model program file with its dedicated syntaxes. The file was then compiled and linked with the ACSL library, to create an executable file. ACSL commands can then be used to perform simulation and analysis tasks. The main difference between ACSL and Fortran is that ACSL is much easier to program, and the library is more powerful. ACSL can directly call the subroutines written in Fortran. Many ACSL blocks (macros) are provided, such as transfer function block `TRAN`, integrator block `INTEG`, lead-lag block `LEDLAG`, time delay block `DELAY`, dead zone nonlinearity block `DEAD`, hysteresis block `BAKLSH` and rate limited integrator block `LIMINT`. These blocks can be used to describe a simulation model of the system. ACSL commands can then be used to analyze simulation results and to draw curves.

After the ACSL source program has been written, the compiler and linker are used to create an executable file. Running the program will automatically generate a prompt: `ACSL>`. At the prompt, relevant commands can then be issued.

**Example 1.1** *Consider the well-known Van der Pol equation described by* $\ddot{y} + \mu(y^2 - 1)\dot{y} + y = 0$. *If* $\mu = 1$, *a set of state variables* $y_1 = y$, $y_2 = \dot{y}$ *can be selected, the Van der Pol equation can be represented as* $\dot{y}_1 = y_1(1 - y_2^2) - y_2$, $\dot{y}_2 = y_1$. *The following statements in ACSL can be written for describing the equation*

```
PROGRAM VAN DER POL EQUATION
CINTERVAL CINT=0.01
CONSTANT  Y1C=3.0, Y2C=2.5, TSTP=15.0
   Y1=INTEG(Y1*(1-Y2**2)-Y2, Y1C)
   Y2=INTEG(Y1, Y2C)
TERMT (T.GE.TSTP)
END
```

*where the display step size is specified as* `CINT` $= 0.01$. *The initial values are represented by the variables* `Y1C` *and* `Y2C`. *The final simulation time* `TSTP` *is assigned as 15. The time variable* `T` *is assumed. Compiling and linking the source ACSL program, an executable file can be generated. Executing the program, the prompt* `ACSL>` *is given. At this a prompt, the following commands can be used:*

```
ACSL> PREPAR T, Y1, Y2
ACSL> START
ACSL> PLOT Y1, Y2
```

*These inform the ACSL model to reserve the variables* `T`, `Y1` *and* `Y2`*, and the phase plane trajectory of* `Y1` *and* `Y2` *can be obtained. The internal parameters in the system can also be set, with the command*

```
ACSL> SET Y1C=-1, Y2C=-3
```

Other packages and simulation languages similar to ACSL appeared at the same time, such as the SIMNON package [10] and ESL [11]. These packages have similar statement structures, since they were based on the same standard.

The emergence and popularization of MATLAB brought mathematical computation to a completely new level. The Simulink environment equipped researchers with new solution methodologies and schemes. Since the release of MATLAB many other software packages appeared, which imitated the syntaxes and ideas of MATLAB, such as Ctrl-C, Matrix-X, O-Matrix, and the CemTool proposed by Professor Kwan at Seoul National University. Octave [12] and Scilab [13] are still available as free software. In this book, MATLAB is extensively and exclusively used for discussing different kinds of simulation problems.

Computer algebra systems, or symbolic computation systems, brought into the field brand new ideas and solutions. Deriving analytical formulae by using programming languages such as C, even by very experienced programmers, may not be easy; indeed, sometimes it is impossible. High-quality computer algebra systems were developed generation by generation. The earlier muMath was developed by IBM, and the Reduce software introduced new solutions to such problems. The dominating Maple [14] and Mathematica [15] soon took the lead in computer algebra systems, and became very successful.

In earlier versions of Mathematica, there was an interface called MathLink to communicate with MATLAB. To better solve computer algebra problems in MATLAB, a Symbolic Math Toolbox was developed, which used Maple as its symbolic computation engine to combine the two major systems together. Then the engine was replaced by muPad.

Since these software packages and languages are usually too expensive for average users, more users are interested in getting free, open-source languages. The MATLAB-like languages such as Octave and Scilab attracted the attention of software users, but the facilities provided by this software are not powerful enough to compete with the sophisticated MATLAB language.

MATLAB and Simulink are the leading-edge tool in scientific computation and system simulation research. It is also the top selected computer language in research fields such as automatic control. In this book, MATLAB and Simulink will be extensively illustrated.

## 1.3 Introduction to MATLAB

### *1.3.1 Brief History of the Development of MATLAB*

The creator of MATLAB, Professor Cleve Moler, is an influential scientist in numerical analysis, especially in numerical linear algebra [1, 2, 3, 16, 17, 18]. In the late 1970s, while he was the director of the computer department of the University of New Mexico, he found it inconvenient to solve linear algebra problems numerically with the then popular EISPACK [2] and LINPACK [1] packages. He then conceived and developed an interactive MATLAB (which stands for matrix laboratory); it indeed brought great convenience for users to solve related problems. With the use of MATLAB, matrix computation becomes a very easy problem. Earlier version of MATLAB can only be used to solve matrix computation problems. There were very few functions related to matrix computation. Since its emergence, MATLAB has received a great deal of attention and was welcomed by educators and researchers alike world wide.

Cleve Moler and Jack Little co-founded The MathWorks Inc. to develop MATLAB-related products. Cleve Moler is still the Chief Scientist at MathWorks. In 1984, the first commercial MATLAB was released, with the supporting language changed from Fortran to C. Powerful graphics, multimedia facilities and symbolic computation were gradually introduced into MATLAB. All these make MATLAB more powerful still. Earlier versions on PCs were called PC-MATLAB, and the workstation version is called Pro MATLAB. In 1990, MATLAB 3.5i was released and it was the first version executable on Microsoft Windows, where the command window and graphics windows can be displayed separately. The SimuLAB environment emerged later, introducing block diagram based simulation facilities, and it was renamed Simulink the following year.

In 1992, the epoch-making MATLAB 4.0 was released by MathWorks, and in 1993, a PC version was released. Graphical user interface programming was introduced, and in 1994, version 4.2 had an enhanced interface design with new methods.

In 1997, MATLAB 5.0 was released and more data types such as cells, structured arrays, multidimensional arrays, classes and objects were supported. Object oriented programming was possible for the first time. In 2000, MATLAB 6.0 was released with many useful windows such as a history command window and several different graphics windows could be displayed at the same time. The kernel of LAPACK [8] and FFTW [19] were used instead of the original LINPACK and EISPACK. The speed of computation and the numerical accuracy and stability were greatly enhanced. Graphical user interface design methods were more flexible, and the interface with C was greatly improved. In 2004, MATLAB 7.0 was introduced, the innovations and concepts of multi-domain physical modeling and simulation were very attractive to engineers.

In 2012, MATLAB 8.0, also known as MATLAB R2012b, was released. In particular, Simulink modeling and simulation facilities were significantly updated.

MathWorks is currently releasing two versions a year now, named version a and version b. At the of writing, the current one was released on September 2012, named 2012b. This version is used in this book to address simulation facilities with MATLAB/Simulink.

MATLAB has now become the de facto top scientific computation and simulation language. The current MATLAB is no longer merely a "matrix laboratory"; it is now a promising, completely new, high-level computer language. MATLAB has been referred to as a "fourth generation" computer language. It is playing an important role in education, academic research and industry. The MATLAB language becomes ever more powerful, to adapt to ever growing needs. More and more software and languages are now providing interfaces to MATLAB and Simulink, where it is becoming a standard in many fields. It is not difficult to reach the conclusion that, in the fields of scientific computation and system simulation, MATLAB will keep its unique and leading position for a long time to come.

### *1.3.2 Characteristics of MATLAB*

MATLAB can run on almost all computers and operating systems. For instance, in Microsoft Windows, Linux and Mac OS X, MATLAB, source code written on one is completely compatible with the others. It can be claimed that MATLAB is independent of computers and operating systems.

From the viewpoint of the authors, the relationships between MATLAB and other computer languages such as C, are similar to the relationship between C and assembly language. Although the execution efficiency of C is much higher than MATLAB, the readability, programming efficiency and portability of MATLAB is much higher than C. Thus for scientific computation purposes, MATLAB should be adopted. In this way, the efficiency of programming, its reliability and the quality of the programs is much higher. For researchers in the area of scientific computation system

simulation, MATLAB can easily reproduce all the functions implementable with C or Fortran. Even if programmers have no knowledge of C or Fortran, they can still design high quality, user-friendly, reliable, high quality programs with high efficiency.

Generally, MATLAB has a very high accuracy of numerical computation, mainly because of the double-precision scheme adopted for the computation. Also, advanced well-tested algorithms with a good reputation are adopted in MATLAB functions. In matrix-related computation, the accuracy can reach the $10^{-15}$ level. Also, symbolic computation can derive analytical solutions to many problems.

Simulink is another shining point in MATLAB applications. The block diagram based modeling techniques and its leading-edge multi-domain physical modeling technique bring users new solutions to simulation problems. The finite state machine system provided in Stateflow, for example, provides practical new tools for the modeling and simulation of discrete event systems and hybrid systems. The interface to external hardware bridges the gap between pure numerical simulation and hardware-in-the-loop simulation and real-time control.

MATLAB/Simulink is now widely used in automatic control, aerospace engineering, the automobile industry, biomedical engineering, speech and image processing and computer engineering applications. In many fields, MATLAB/Simulink has already become the number one computer language.

## 1.4 Structure of the Book

### *1.4.1 Structure of the Book*

As in the learning of any computer language, active and repetitive practice are essential to mastering MATLAB and Simulink. Only regular practice will improve your ability in MATLAB programming and your application skills. For the student readers, the statements, programs and models should be used in person to gain more knowledge and skill with MATLAB. First-hand knowledge is very important for mastering computer languages and tools.

In this first chapter, system simulation concepts are briefly discussed. The development of computer packages and simulation languages is also briefly introduced. In Chapter 2, the concentration is on the fundamentals of MATLAB programming. Useful topics in MATLAB programming such as data types, statement structures, function programming, graphical visualization and graphical user interface design are logically presented. Essential knowledge of MATLAB programming are fully covered in this chapter. In Chapter 3, simulation-related scientific computation problem solutions with MATLAB and applications are explained. The topics of numerical linear algebra, differential equations, nonlinear equation solutions and optimization, dynamic programming, data interpolation and statistical analysis are presented. These topics are the essential mathematical fundamentals for solving simulation problems. In Chapter 4, primary knowledge on Simulink modeling is presented. A brief introduction to commonly used Simulink model groups is given first. The use of the Simulink environment is presented, and examples are used to demonstrate Simulink applications in mathematical modeling. Modeling and simulation of linear systems are presented, followed by the simulation studies of stochastic continuous systems. In Chapter 5, intermediate knowledge of Simulink modeling is presented. Application skills of commonly used blocks, nonlinearities modeling, algebraic loop avoidance, zero-crossing detection and solutions of various differential equations are extensively studied. Simulation result visualization via gauges and virtual reality techniques are also illustrated. Subsystem modeling and block masking techniques are presented in this chapter, and the F-14 aircraft control problem is used to demonstrate the use of subsystem model techniques. In Chapter 6, advanced techniques in Simulink modeling are presented. Mainly programming based modeling techniques are introduced. Statement based modeling methods are introduced first, and then linearization and S-function programming are introduced. Optimization

based optimal controller design problems are demonstrated. Chapter 7 presents engineering system simulation and multi-domain physical modeling technique. Simulation tools such as Simscape, SimPowerSystems, SimElectronics and SimMechanics are presented, through examples, and the simulation of electrical, electronic and mechanical systems is presented. In Chapter 8, we look at some non-engineering system simulation techniques, including pharmacodynamical modeling and control problems, image and video processing problems and discrete event system modeling problems. In Chapter 9, hardware-in-the-loop real-time simulation and control problems are considered.

### *1.4.2 Code Download and Internet Resources*

The MATLAB functions and models developed for the book can be downloaded directly from the book service website at Wiley (`http://www.wiley.com/go/xue`) or from:

`http://mechatronics.ucmerced.edu/simubook2013wiley`

However, we suggest that you do not use all the downloaded files directly. It would be better to input the functions and models yourself, since this is also a useful stage of learning and practical experience. If the solutions obtained by the users are different from the ones given in the book, the downloaded materials can be used for comparison.

The whole set of PDF and HTML manuals for MATLAB and its related toolboxes can be downloaded directly from the official MathWorks website. There are also a lot more free third-party toolboxes downloadable from internet. Moreover, active and experienced MATLAB users may answer various of your questions. The commonly used websites and forums are:

- MathWorks Website: `http://www.mathworks.com`.
- User forum: `http://www.mathworks.com/matlabcentral/newsreader/`.

Here are two suggestions for the use of forums: first, when a problem is encountered, first try to solve the problem by yourself. Sometimes the answers obtained by one's own effort can be of great benefit. Second, actively contribute to the questions to which you know the answers, or participate in discussion, so as to improve the skills of others.

### *1.4.3 Fonts Used in this Book*

The fonts in the book are illustrated as follows, to help you understand better the materials presented here:

- Times-Roman fonts are for constants in formulae such as e, d$x$, and x axis.
- Italic Times-Roman font are provided by MATLAB to represent variables such as $x$ $t$ in MATLAB equations, while bold italic Times-Roman font are used to present vectors and matrices, such as $\boldsymbol{A}$, $\boldsymbol{x}$ and $\boldsymbol{f}(t, \boldsymbol{x})$.
- Typewriter font is used to represent program listings, as well as function names, such as `eig()`, `tic`, `stateflow`.
- The text in interfaces, Simulink group and block names are denoted by bold Helvetica font, such as **File** menu, **OK** button and **Step** block.

## Exercises

**1.1** A large number of demonstration programs are provided in MATLAB. To invoke the main demonstration program type the `demo` command in the MATLAB command window. Run the demonstration program and get a feel of the powerful facilities provided in MATLAB.

**1.2** Programs and models designed for this book can be downloaded from the website for the book, and all the code is repeatable. It is advisable to input the program and block diagrams yourself, rather than use the downloaded ones directly, so as to understand better the materials presented in the book.

**1.3** A powerful on-line help system is provided in MATLAB. Also the command `lookfor` allows you to search for keywords and function names. You can also use the `help` or `doc` commands to search for information, including syntax, of a particular MATLAB command. For example, a Riccati matrix equation is given by

$$\boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{P} + \boldsymbol{Q} = \boldsymbol{0}$$

and

$$\boldsymbol{A} = \begin{bmatrix} -27 & 6 & -3 & 9 \\ 2 & -6 & -2 & -6 \\ -5 & 0 & -5 & -2 \\ 10 & 3 & 4 & -11 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 & 3 \\ 16 & 4 \\ -7 & 4 \\ 9 & 6 \end{bmatrix}, \quad \boldsymbol{Q} = \begin{bmatrix} 6 & 5 & 3 & 4 \\ 5 & 6 & 3 & 4 \\ 3 & 3 & 6 & 2 \\ 4 & 4 & 2 & 6 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} 4 & 1 \\ 1 & 5 \end{bmatrix}.$$

Try to use the `lookfor riccati` command to find a possible Riccati equation solver, then use the `help` command to find the syntax of the solver and solve $\boldsymbol{P}$ for the above equation.

## References

[1] J J Dongarra, J R Bunch, C B Moler, *et al.* LINPACK user's guide. Philadelphia: Society of Industrial and Applied Mathematics (SIAM), 1979

[2] B T Smith, J M Boyle, J J Dongarra. Matrix eigensystem routines – EISPACK guide, Lecture notes in computer sciences, volume 6. New York: Springer-Verlag, (2nd Edition), 1976

[3] B S Garbow, J M Boyle, J J Dongarra, *et al.* Matrix eigensystem routines – EISPACK guide extension, Lecture notes in computer sciences, volume 51. New York: Springer-Verlag, 1977

[4] Numerical Algorithm Group. NAG FORTRAN library manual, 1982

[5] W H Press, B P Flannery, S A Teukolsky, *et al.* Numerical recipes, the art of scientific computing. Cambridge: Cambridge University Press, 1986

[6] J L Melsa, S K Jones. Computer programs for computational assistance in the study of linear control theory. New York: McGraw-Hill, 1973

[7] CAD Center. GINO-F Users' manual, 1976

[8] E Anderson, Z Bai, C Bischof, *et al.* LAPACK users' guide. SIAM Press, 3rd Edition, 1999

[9] E E L Mitchell, J S Gauthier. Advanced continuous simulation language (ACSL) – user's manual. Mitchell & Gauthier Associates, 1987

[10] K J Åström. Computer aided tools for control system design, In: Jamshidi M and Herget C J. (eds.) Computer-aided control systems engineering. Amsterdam: Elsevier Science Publishers B V, 1985, 3–40

[11] R E Crosbie, S Javey, J L Hay, *et al.* ESL – a new continuous system simulation language. Simulation, 1985, 44(5): 242–246

[12] Octave Language Webpage. `http://www.octave.org/`

[13] SciLAB Language Webpage. `http://scilabsoft.inria.fr/`

[14] F Garvan. The Maple book. Boca Raton: Chapman & Hall/CRC, 2002

[15] S Wolfram. The Mathematica book. Cambridge: Cambridge University Press, 1988
[16] G E Forsythe, M A Malcolm, C B Moler. Computer methods for mathematical computations. Englewood Cliffs: Prentice-Hall, 1977
[17] G E Forsythe, C B Moler. Computer solution of linear algebraic systems. Englewood Cliffs: Prentice-Hall, 1967
[18] D Kahaner, C B Moler, S Nash. Numerical methods and software. Englewood Cliffs: Prentice Hall, 1989
[19] M Frigo, S G Johnson. The design and implementation of FFTW3. Proceedings of IEEE, 2005, 93(2):215–231